Applications of Molecular Dynamics simulations for biomolecular systems and improvements to density-based clustering in the analysis

Inaugural dissertation to obtain the academic degree
doctor rerum naturalium (Dr. rer. nat.)

submitted to the department of Biology, Chemistry, Pharmacy,
Freie Universität Berlin

by
Jan-Oliver Felix Kapp-Joswig

2022

1st reviewer
Prof. Dr. Bettina G. Keller

Freie Universität Berlin
Institute of Chemistry and Biochemistry
Theoretical Chemistry
Arnimallee 22, 14195 Berlin

2nd reviewer
Prof. Dr. Beate Paulus

Freie Universität Berlin
Institute of Chemistry and Biochemistry
Theoretical Chemistry
Arnimallee 22, 14195 Berlin

Date of defence: 13.12.2022

GROMACS reminds you:

'If 10 years from now, when you are doing something quick and dirty,
you suddenly visualize that I am looking over your shoulders and say to yourself:
"Dijkstra would not have liked this",
well that would be enough immortality for me.'

—Edsger Dijkstra

Acknowledgement

I would like to express my deep gratitude to my primary supervisor Professor Bettina Keller for her invaluable guidance and support during my doctoral studies. To begin with, she accepted me into her research group when I had little experience in Computational Chemistry, trusted my abilities, and sparked my enthusiasm for the field. I want to thank her for the encouragement and motivation when I encountered difficulties, the freedom to pursue my own ideas, and the interesting science we did together.

I would also like to thank Professor Beate Paulus a lot for being always approachable and quick with good advice, and for taking on the role of the second reviewer for this thesis.

Furthermore, I want to thank Professor Roderich Süssmuth and Professor Christoph Rademacher and their groups for our productive collaborations.

I thank the DFG for the generous funding via SFB765, RTG2473, and EXC 2008 'UniSysCat', the HLRN and PC^2 for the granted computational capacity, and the ZEDAT for the solid technical support.

I will keep countless good memories of the last five years in Theoretical Chemistry at the FU Berlin and I consider myself very lucky to have met so many great people. In particular I want to name my former colleagues Stevan Aleksić who animated me to join the group as an intern in the first place and Oliver Lemke from whom I inherited a passion for clustering algorithms. There are of course also my long-time early office partners Felix Witte, Tim Küllmey, and Christian Becker who made me enjoy my work ever since we started together in a small room at Takustraße 3.

Likewise, I would like to thank Marius Wenz for his friendship, his helpful feedback on my writing, and his great commitment to our various group activities. I also thank Stefanie Kieninger that I could ask her for help with different small and big problems I faced in my work. And I do not want to forget to thank Jennifer Anders who shared a substantial amount of time with me on the langerin project and Dr. Jan Götze for many stimulating discussions and ideas. At the risk of not doing justice to all the other group members and colleagues, I just want to say that I am very grateful for the time we spent together. I will miss our numerous BBQs, movie evenings, Halloween parties, 'Feierabend'-beers, lunchtimes, and coffee breaks.

Finally, I want to thank my friends and family who may not have seen me a lot lately but who are always there for me. Among others, I thank Jan Zawallich for having an open ear and his incentive words. This also includes my parents who appreciate what I am doing and supported me all the way since the beginning. Especially my wife, Larissa, has been a big help and deserves a title on her own for hanging in there with me until the end.

Selbstständigkeitserklärung

Hierdurch versichere ich, dass ich meine Dissertation selbstständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen und wurde bisher nicht veröffentlicht.

Abstract

Molecular Dynamics simulations provide a powerful tool to study biomolecular systems with atomistic detail. The key to better understand the function and behaviour of these molecules can often be found in their structural variability. Simulations can help to expose this information that is otherwise experimentally hard or impossible to attain. This work covers two application examples for which a sampling and a characterisation of the conformational ensemble could reveal the structural basis to answer a topical research question. For the fungal toxin phalloidin—a small bicyclic peptide—observed product ratios in different cyclisation reactions could be rationalised by assessing the conformational pre-organisation of precursor fragments. For the C-type lectin receptor langerin, conformational changes induced by different side-chain protonations could deliver an explanation of the pH-dependency in the protein's calcium-binding. The investigations were accompanied by the continued development of a density-based clustering protocol into a respective software package, which is generally well applicable for the use case of extracting conformational states from Molecular Dynamics data.

Molekulardynamik Simulationen bieten ein mächtiges Instrument für die Studie biomolekularer Systeme mit atomarer Auflösung. Der Schlüssel dazu, die Funktion und das Verhalten dieser Moleküle besser zu verstehen, kann oft in ihrer strukturellen Variabilität zu finden sein. Simulationen können helfen diese Information freizulegen, was auf anderem Wege experimentell nur schwer oder unmöglich zu erreichen wäre. Diese Arbeit umfasst zwei Anwendungsbeispiele, wo eine Generierung und Charakterisierung des konformationellen Ensembles die strukturelle Basis dafür aufdecken konnte eine aktuelle Forschungsfrage zu beantworten. Für das Pilz-Toxin Phalloidin—ein kleines bicyclisches Peptid—konnten beobachtete Produktverhältnisse in verschiedenen Cyclisierungsreaktionen nachvollzogen werden, indem die konformationelle Vororganisation von Precursor-Fragmenten beurteilt wurde. Für den C-Typ Lektin-Rezeptor Langerin konnten Konformationsänderungen, ausgelöst durch verschiedene Seitenketten-Protonierungen, eine Erklärung für die pH-Abhängigkeit der Kalzium-Bindung dieses Proteins liefern. Die Untersuchungen wurden durch die Weiterentwicklung eines Dichte-basierten Clustering-Protokolls in ein entsprechendes Software-Paket begleitet, das generell sehr gut für den Anwendungsbereich geeignet ist, konformationelle Zustände aus Molekulardynamik Daten zu extrahieren.

Contents

List of abbreviations

API	application programming interface
BFS	breadth-first-search
CLR	C-type lectin receptor
CRD	carbohydrate recognition domain
CTLD	C-type lectin-like domain
CommonNN	common-nearest-neighbour
DC	dendritic cell
DFS	depth-first-search
GMM	Gaussian mixture model
IC	independent component
KDE	kernel density estimate
LC	Langerhans cell
MC	Monte Carlo
MD	Molecular Dynamics
MI	mutual information
ML	Machine Learning
MM	Molecular Mechanics
MSM	Markov-state model
MST	minimum spanning tree
OOP	object-oriented programming
PBC	periodic boundary condition
PC	principal component
PCA	principal component analysis
PCCA	Perron-cluster cluster analysis
PRR	pattern recognition receptor
QM	Quantum Mechanics
RMSD	root-mean-square deviation
RMSF	root-mean-square fluctuation
*t*ICA	time-lagged/time-structure based independent component analysis

List of Figures

List of Tables

Part I.

Introduction

{ 1 }

Overview

Motivation and thesis organisation

The work presented on these pages circulates about one big topic: Molecular Dynamics (MD). This fascinating method melds many different subtopics and scientific fields. Speaking broadly, to completely understand MD as the valuable research tool it is, one needs to acknowledge the underlying physics, as well as the chemistry, biology, or engineering background to which it is applied, and a fair deal of the mathematics involved, with an emphasis on statistics of large amounts of data. Last but not least, daily practice requires to cope with modern computer technology und programming. MD really is in theory and application what can be called interdisciplinary. Numeric simulations bring together the experimental observations we make and the theoretical models we build. How we use and appraise them in modern scientific research also has a philosophical dimension.[1]

The central content of this thesis are three concrete MD based research projects. Of all the possible areas of application for this method, I concentrated on the equilibrium sampling of biomolecules (mostly proteins) and the identification of long-lived conformational states to explain their biological function and observed behaviour. I will present two molecular systems that have been investigated in the last years and describe the development of a density-based clustering protocol that has been of major relevance for the respective analyses. To put these projects into context, the related theory and the methods used will be addressed here as well. I tried to provide a rough overview of the most important aspects—about what needs to come together to perform a MD simulation of a biomolecule and to draw sensible conclusions from it. Necessarily, this has to be limited and the primary intend will be to outline the big picture. At the same time, I will restrict myself mostly to standardised classical MD as we used it. The focus lies on the extraction of meaningful information from MD data in terms of characteristic conformational states that can be eventually linked to biomolecular function.

Against this background, the thesis is structured as follows. I would like to continue the introduction with a brief general review of some of the historical circumstances and milestones that in the long run led to the development of MD as the research instrument it is today (chapter 2). Then, there will be a summary of the projects I worked on (chapter 3) with a comment on what makes them interesting and what connects them.

A basic theoretical part (part II) should level out the foundation for the main body of this thesis. This includes a selection of typical ingredients for MD simulations (chapter 4) with short excursions into polarisable force fields (section 4.3) and steered MD (section 4.9). It also contains a chapter on the standard approach that we employed in conformational analyses (chapter 5), and one on kinetic Markov models (chapter 6). A versatile concept that will reappear frequently throughout this work are connected networks in terms of *graphs*, which are addressed separately in chapter 7.

Subsequently, one part is dedicated to each research topic: the cyclic peptide phalloidin as a representative of the phallo- and amatoxins (part III), langerin as a member of the C-type lectin receptor family (part IV), and density-based common-nearest-neighbour (CommonNN) clustering in comparison to other clustering algorithms (part V). Each project part is divided into a general

contextual chapter and one to recap the journal article that has been published on the respective research topic. These associated publications can be found in the appendix under *Publications*.

Please note that the theoretical groundwork affiliated with the clustering project is not included in the regular theoretical part but has been moved to the general project part (chapter 13). For the langerin project, sections 11.1 and 11.2 represent new preliminary results as a practical outlook on possible continuations of the study. At the end of the langerin part, there is furthermore a chapter of rather pragmatic nature with basic advice on how to run MD simulations in GROMACS (chapter 12).

I WOULD LIKE TO MAKE A FEW FORMAL REMARKS on how the content of this thesis is presented. To support an easy orientation within the text, I make use of margin notes (see left) to highlight a specific detail that a specific paragraph is about. In text, I use *italic* font to put emphasis on important terms when they occur for the first time. Certain abbreviations are furthermore listed in the *List of abbreviations* and will mostly be written out in full on first use. Text elements that are clickable hyperlinks in the PDF version of this thesis—URLs, citations, and figure and table references—are highlighted in a lighter grey. A monospace `typewriter` fontface will be used for URLs. Sporadically, references to online sources may be mentioned in footnotes and figure captions only, while regular citations will be collected in the *References* at the very end.

content landmark

Code and (Python flavoured) pseudo-code examples are typeset inline or in separate blocks using a monospace fontface and language specific syntax highlighting. Inline filenames, data structures and types, module names and such are treated in this sense as code elements. Note that two spaces are universally used for each level of indention to save some horizontal space against the general recommendation to use four spaces in practice instead.

```
def heaviside(x):
  """Positive integers to 1, negative to 0"""
  return int(x > 0)
```

When command-line snippets are shown, they are visually distinguished from code snippets via a prepended $ sign, like in: `$ gmx check -f out.xtc`.

I try to maintain a consistent notation in the presented mathematical equations, which mostly have a physical context. The majority of the conventions followed is fairly standard but I mention them here just to avoid confusion. For reference see [2].

Physical quantities and mathematical variables will be in general *italic* symbols, e.g. $f(x) = ax + b$. This includes arbitrary functions and physical constants, e.g Avogadro's number N_A. It also includes vector quantities which are, however, additionally distinguished as ***bold*** symbols. Whenever a vector is defined in the way of $\boldsymbol{x} = (x_1, x_2, x_3)$ it is implied that it should be treated as a column vector unless stated otherwise, i.e.

$$\boldsymbol{x} = (x_1, x_2, x_3) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}^\mathsf{T} .$$

Inline row vectors will be distinguishable by a smaller font of the elements and the omission of commas like in $\boldsymbol{x}_\mathrm{row} = \left(\begin{smallmatrix} x_1 & x_2 & x_3 \end{smallmatrix}\right)$.

Upright letters are used to discern symbols explicitly as non-variable, that is for labels, especially mnemonic indices, e.g. k_B (where the index B stands for the label 'Boltzmann'). This extends to physical units, e.g. $m = 1\,\mathrm{kg}$, as well as to established mathematical constants and functions, e.g. $\mathrm{e}^{\mathrm{i}\pi}$, sin, log, etc.

Shell letters (blackboard fontface) are used only to denote special mathematical sets, e.g. an n-dimensional real space $\mathbb{R}^n$. Although arbitrary, a calligraphic font is used to distinguish certain other sets, e.g. a time series $\mathcal{Q}$ as a set of configurations, or in general a data set $\mathcal{D}$. Just to be consistent with the common literature notation the Lagrangian will be set as $\mathcal{L}$ as well.

{ 2 }

A brief history of Molecular Dynamics

How came about what all this is about

Since ancient times, humankind is fascinated by moving objects and occupied with the discovery of mathematical laws to describe physical motion. The phrase πάντα ῥεῖ—everything flows—expresses the idea of early greek philosophers like Herakleitos and Plato that 'all things are in motion like streams'.[3] When arguably the foundation for modern natural science was laid around 500 BCE, it was already felt by the scholars of the time that nature is intrinsically dynamic and that motion is essential to understand the world around us. The major subject of investigation always was the observation of the sky, particularly the night sky, that dates back in a systematic form as far as 2000 BCE to babylonian astrology.[4] Historically, it was motivated by religion, agricultural praxis, or used for orientation.

At the latest, however, starting with Nicolaus Copernicus (1473–1543) who studied the planets orbiting in our solar system and popularised the heliocentric theory, the analysis and prediction of celestial body motion became a science. He was followed among others by great names like Tycho Brahe (1546–1601), Johannes Kepler (1571–1630), Galileo Galilei (1564–1642), René Descartes (1596–1650), and Giovanni Cassini (1625–1712) but it was not until 1687 that Isaac Newton (1643–1727) published his three fundamental laws of motion in the seminal *Philosophiæ Naturalis Principia Mathematica*.[5] With Newton's theory of gravity, the way was paved for a 'system of the world' as we more or less see it today. The connection of effective acceleration to forces that drive body motion, unified the dynamic description for many (macroscopic) physical objects—celestial as well as terrestrial—and set a corner stone for all classical mechanics. Moreover, as Steven Strogatz expressed it, *'in the 300 years since Newton, mankind has come to realize that the laws of physics are always expressed in the language of differential equations.'*[6]

Figure 2.1 shows three illustrative examples of physical systems that can be and are routinely described using Newton's laws. For most purposes, it is sufficiently accurate to describe planetary motion by Kepler orbits, that is by parametrised solutions to Newton's equation, where each orbit is separately considered as a two-body problem (figure 2.1a). The influence of planet-planet interactions is ignored here as well as relativistic effects, which would lead to deviations of the orbits from their idealised elliptic form. Over time, the error made by this approximation accumulates, which is why Kepler orbit parameters are only valid within a certain time window.

Bodies, much smaller than planets, moving in the gravitational field of the earth near the earth surface can be described accurately by the same physical laws (figure 2.1b). Projectiles like fireworks exhibit typical parabola shaped flight trajectories if external forces like friction or wind are neglected. But even atoms in molecules can be treated to some extent as Newtonian particles—and this is the basis also for this thesis—when the present inter-atomic interactions, originating from their electronic structure, are approximated with parametrised forces (figure 2.1c). This makes Newton's laws with limitations valuable for the study of problems over length scales of several picometres to billions of kilometres and time scales of femtoseconds to hundreds of years, i.e. give or take 20 orders of magnitude.

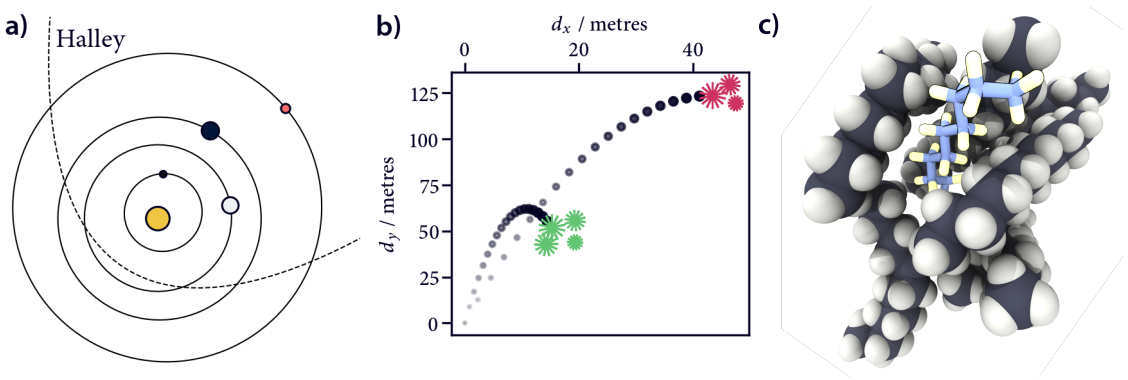


Figure 2.1 Examples for dynamic Newtonian systems

a) Kepler orbits for selected planets and the Halley comet around the sun. These ellipses are idealised approximations to the real trajectories drawn based on the orbit elements obtained from NASA (valid for 1800 CE – 2050 CE), ignoring the inclination and the longitude of perihelion (Halley data from The SkyLive). For visual clarity, the planet diameters are scaled by a factor of 2000, and the sun diameter is scaled by a factor of 50. **b)** Projectile motion for firework rockets started from the same positions but with different initial velocities and shooting angles, set up to explode five seconds after ignition. The trajectories are solutions to Newton's equation in the gravitational field of the earth and in absence of other forces. **c)** Molecular system of several interacting carbohydrates. In Molecular Dynamics (MD) simulations, Newton's equations are used to propagate atomic positions according to forces from interatomic interactions.

Of course, the fundamental theoretical progress in the study of motion did not came to a halt after Newton. Major contributions related to *equations of motion* can be attributed to Leonhard Euler (1707–1783), Joseph-Louis Lagrange (1736–1813), and William Rowan Hamilton (1805–1865). Before many interesting problems could be addressed and more than only the most simple systems could be studied under the influence of physical forces in dynamic detail, however, another important aspect needed to be promoted: *numerical methods* to produce approximations to the solutions of differential equations. In many cases, and this is true for almost any interesting molecular system, the equations of motion are complex and can not be solved analytically. Again it were Newton, Euler, Lagrange, but also Carl Friedrich Gauss (1777–1855) to whom we owe early advances into the direction of a numeric treatment of these problems.

The practical usefulness and feasibility of numerical methods experienced an upswing with the invention and development of modern computers in the first half of the last century, which coincided with the catastrophe of World War II.[1] The *electronic numeric integrator and computer* (ENIAC), one of the first general-purpose computers, was constructed for the numeric calculation of ballistic trajectories of artillery projectiles and also came to use in the context of the Manhatten project at the Los Alamos National Laboratory.[7, 8] There it was also that the *mathematical analyzer numerical integrator and automatic computer* (MANIAC) model I, an indirect successor of the ENIAC, was eventually put to more civil applications than only research related to the thermonuclear process and the hydrogen bomb. Besides playing anti-clerical chess,[9] it ran the probably first Monte Carlo (MC) simulation of a liquid—a 2-dimensional system of rigid spheres interacting via pairwise spherical potentials—done by Nicholas Metropolis, Arianna and Marshall Rosenbluth, and Augusta and Edward Teller.[10] Another famous study of the time on anharmonic 1-dimensional crystals is attributed to Fermi, Pasta, Ulam, and Tsingou.[11, 12]

Berni Alder from the university of California can be considered the 'inventor' of Molecular Dynamics (MD) as an alternative tool to MC simulations for the study of many-body problems. Among other things, he did pioneering work on phase transitions of hard sphere systems,[13] a matter that caused

[1] See `computerhistory.org/timeline/computers` for a short illustrative overview on the history of computers

heated discussions at the time and started sort of a footrace between advocates of the previously established MC and the newer MD approach.[14] Alder and his close working partner Tom Wainwright also count as early explorers in the realm of impactful scientific visualisation to communicate their results.[15] Landmark publications in the evolution of MD as an accepted research tool (and numeric simulations in science in general) are for example also Vineyard's studies on copper crystals,[16] and Rahman's work on correlated motion in liquid argon,[17] making the transition to realistic systems. Among the next steps were simulations of liquid water and the construction of appropriate water models to describe this exceptionally tricky system.[18, 19]

The first classical MD simulation of a protein molecule might have been that of bovine pancreatic trypsin inhibitor (BPTI) performed in the group of Martin Karplus in 1977.[20] The small protein of 58 amino acids plus four crystal water molecules (almost 1000 atoms) was simulated for about 9 ps in its folded state. Before that, the same molecule has already been subjected to a folding protocol consisting of energy minimisation and thermalisation stages by Michael Levitt and Arieh Warshel.[21] The authors were aware of the complications associated with protein folding due to the vastness of potential conformations that the molecule could adopt—earlier expressed in terms of the Levinthal paradox.[22] Notably, a surprisingly elaborate kind of excited state simulation (using the so called semi-classical trajectory approach) of the fast retinal isomerisation in rhodopsin has also been achieved around the same time,[23] a process that still receives much attention.[24]

Some of the methodological fundaments and technical details, which remain relevant for the execution of MD simulations today, had already been settled at this time. An example is the essential Verlet list without which most simulations would be prohibitively expensive, due to the unfavourable scaling in the computation of non-bonded interactions.[25] Also the combination of typically used inter-atomic interactions—the force field—had been established in a basic form.[26, 27] Others still needed to be invented or systematically improved in the light of simulations of macromolecular systems, like suitable dynamic integrators or algorithms that satisfy constraints.[28] Berendsen and van Gunsteren gave one of the earliest comprehensive overviews in this regard shortly after the above mentioned BPTI simulation.[29] To name only a few milestones from the early 1980s, this is when Andersen proposed the means for simulations at constant temperatures and pressure,[30] closely followed by Parrinello and Rahman,[31, 32] de Leeuw applied Ewald sums in the context of MD to solve the electrostatic interactions on a periodic lattice, and Swope published about the velocity-Verlet integrator.[33] Also, free energy perturbation emerged as a motivation for MD simulations.[34, 35] The first notable textbook dedicated to the topic of MD, *Computer simulation of liquids* by Allen and Tildesley,[36] marks the turning point at which these simulations become more and more standardised and gives practical advice for the setup. Slowly, force fields started to mature for example with the OPLS parameter set.[37, 38]

From than on, the methodology was consolidated further,[39] also with a focus on trajectory analysis and visualisation,[40] and the number of applications exploded in the 1990s. MD was applied to simulate melts of polymer chains with up to 400 monomers using a bead spring model that is an early example for coarse grained dynamics,[41] similar to what has previously been done by Rapaport.[42] Grubmüller studied ligand binding with steered MD.[43] Jarzynski showed that equilibrium free energy differences can be obtained from non-equilibrium work.[44] First steps into the direction of including charge polarisation effects in simulations in terms of the AMOEBA force field were made,[45] although polarisation as such has been already considered much earlier.[46] Later in the decade, the use of Markov models for the analysis of trajectory data was introduced.[47] And finally, an explicit solvent folding simulation of the villin headpiece subdomain (about 10,000 atoms) by Kollman set new standards by crossing the 1 µs total simulation time mark—a time scale on which folding processes of peptides and very small proteins can already be adequately studied.[48]

Simulations of this magnitude necessitated the development of professional programs and parallel computing.[49] Many popular simulation suites like GROMACS[50] and AMBER[51] stem from this era and started to substitute the various codes used by research groups around the world. But this was only the beginning. With the next millennium, advances in computer hardware and in particular the implementation of MD routines for GPUs, boosted the performance of simulations tremendously.[52, 53] A broad array of medium sized problems became tractable even for individual researchers with only access to limited resources. High performance computing clusters, on the other end, partly specifically just dedicated to the efficient execution of MD simulations, led to a series of ephemeral records in terms of treated system sizes and achieved simulation times on a regular basis. In 2006 for instance, the first atomistic simulation of an entire satellite virus (50 ns, about 1 million atoms) and its analysis got published.[54] Today, the Anton 3 supercomputer of D. E. Shaw research, which went operational last year, can produce over 100 µs per day for respectable system sizes of about 100,000 atoms. For a massive HIV-1 capsid simulation with about 72 million atoms, it reaches still almost 2 µs per day.[55] Just for comparison, a smaller version of this system has been simulated already for 1 µs in total, which has been quite sensational only four years earlier.[56] Currently, more than a billion atoms can be simulated in extreme cases and open the way to study biomolecular processes on a cellular level.[57] In a fast reaction to the recent pandemic, large scale simulations of SARS-Cov-2 spike proteins and the complete virion have been performed to help understand and eventually control the virus.[58, 59] The continuing demand for longer simulations of larger systems in shorter effective time also led to the conquering of novel computation infrastructure like the Azure cloud.[60] This is contrasted by the simulation strategy of the Folding@home project that distributes the sampling workload over a multitude of compute nodes, made available by private donors. Just this year, 1.8 ms of accumulated simulation time for a series of peptides helped with assessing their therapeutic potential as inhibitors of cancer cell growth.[61]

The field of MD simulations has evolved rapidly. Its general framework is very stable, a lot of its elements are still used in the same form as 50 years and more ago, but over time, the technique has diversified into many different sub-branches and flavours. MD can be considered a *dispersed* method, having a high input complexity with regard to its setup.[62] Although a fair portion of what a simulation consists of is standardised nowadays, users of this tool still need to be experts in general to make the right decisions about simulation parameters, the preparation of the system, or quantities to compute. It is easy for a non-expert to get nonsensical simulation results. Maybe the next years will bring further improvements in terms of user-friendliness and automation of standard tasks.[2] This is for instance especially critical for the organisation of free energy perturbation series in large drug discovery campaigns.[63, 64] Here, many individual simulations need to be set up robustly in a correct way and it should be ideally convenient and fast so that it can be done by researchers with a primary focus on drug design rather than the technical internals of a MD program.

But also in other aspects the evolution of MD will not stall. The used force fields are still continuously improving, not only for proteins[65] as the main object of interest, but also for lipids[66], carbohydrates,[67] nucleic acids,[68] and metal ions in conjunction with the newest generation of water models.[69] Still the hardest challenge is probably posed by the parametrisation of small organic (drug) molecules.[70] This is one of the parts for which a recent trend could be a real game changer: the advent of Machine Learning (ML) in the molecular sciences. ML models trained with Quantum Mechanics (QM) based calculations can already yield accurate force field parameters for individual molecules and are very quick in doing so.[71] Generally, ML will have a rather big impact on how MD

[2] Several (mostly commercial) platforms committed to this goal have appeared in the last years, like Schrödinger's Maestro/Desmond, Acellera's PlayMolecule/ACEMD, or OpenEye's Orion suite, not to forget Quantistry started at FU Berlin in 2019.

simulations will be carried out in the future, be it by providing intelligent coarse-graining schemes,[72] or by learning reaction coordinates for enhanced dynamics.[73] There will be even areas where MD simulations, which are still a quite expensive tool, are not required any more at all. Nothing made this more obvious than the recent success of AlphaFold that can find meaningful folded protein states faster than any MD protocol ever could.[74] It is also already not a completely unrealistic endeavour for example to predict structures from equilibrium distributions of complex systems without repeatedly running a simulation of importance sampling like MD.[75] The decision of when to use MD because it is the tool of choice and when to use something else, will become more and more important if we aim on a sensitive use of our available resources.

ML assisted tools will impact beyond that the way how we analyse MD data in terms of feature selection and extraction, dimensionality reduction, and kinetic model construction.[76] The applicative potential of ML to support or substitute MD is immense but also without it, there are far-reaching developments underway.[77] Examples from our group for methodology that might change the way how and when we use MD, are the estimation of kinetic models via the square root approximation without the need for conformational sampling[78] and path re-weighting to retain unbiased dynamics from enhanced simulations.[79]

Alongside theories and practical experiments, computer simulations are also from an official side considered a part of the *'third pillar'* of 21st century science. In bloomy words: *'computational science enables researchers and practitioners to bring to life theoretical models of phenomena too complex, costly, hazardous, vast, or small for "wet" experimentation'*.[80] The days are definitely over that nobel laureates are quoted on the need to resort to numeric simulations as an *'indignity'*.[81] Quite on the contrary, computational modelling has become a central and indispensable element in the advancement of new theories by testing the models predicted by existing theories.[82]

{ 3 }

Projects and research questions

The exploration of conformational spaces

As mentioned in the preface and honoured with the last chapter, this thesis is about Molecular Dynamics (MD). The three main research projects I have worked on in the last years and that are included here make extensive use of it. The first two of them are application centred. In both, we used classical MD simulations to produce a data set of conformational snapshots for a biologically relevant system: phalloidin, a small bicyclic peptide found in a poisonous mushroom, and langerin, a medium sized receptor protein playing a big role in our immune system.

The questions we wanted to answer were simple—at least initially—and quite different for these two molecules. Phalloidin confronted us with the problem why seemingly similar synthetic pathways could lead to the desired natural product with varying success. Langerin challenged us with the riddle how it is affected by environmental changes in the pH so that it looses its ability to bind calcium. While these questions have not much in common at a first glance, the key to their answer was the same in both cases: a characterisation of the respective conformational ensembles. What unites the two projects, is the molecular simulation we used to assess the conformations the system can be observed in, their relative population, and the timescales on which they change.

Moreover, for both these projects it was not enough to analyse the absolute conformations of one specific molecule. Rather we needed to compare the conformational ensembles of several systems with each other because the answer to our question was found in the differences between them or the recognisable population shifts upon a perturbation. For phalloidin, the distinct conformational predisposition of precursor model systems, with respect to the orientation of a characteristic bridge before a final cyclisation reaction, could be used to predict and explain the varying product selectivities. For langerin, a hidden conformation that is exclusively accessible in an acidic environment and that could only be revealed by careful investigation of respectively protonated states, was identified as a promising calcium-affinity modulating candidate.

A MD simulation generated the primary source of information about the systems and all of our following investigations were based upon this. But also beyond that, the tools we used in the subsequent analysis were very similar in the two projects. For the characterisation of the conformational ensembles, we needed a reduced projection of the high dimensional atomic trajectories onto suitable coordinates. That means we needed to choose low dimensional representations that are comprehensible and able to explain what we wanted to learn about the systems. Even for the simple heptapeptide phalloidin, a direct visual evaluation of the original data could only deliver insufficient anecdotal evidence. For langerin, collecting meaningful observations by watching simulation movies was even more hopeless, not only because of the sheer amount of aggregated data but also because the conformational changes of interest are very subtle. To filter out the relevant pieces, we had to extract expressive features—be it a single smartly defined dihedral angle that illustratively described the bridge orientation in phalloidin or an inter-atomic distance that reduced the complex langerin dynamics to the movement of a small flexible loop region. As another example, hydrogen bonded interactions were useful to characterise

and separate conformational states of langerin as well as to understand the relative conformational stability of phalloidin precursor fragments.

The complexities we faced in tackling our research questions were unevenly greater, however, for langerin then for phalloidin. We used a larger portfolio of dimensionality reduction techniques, had to be more rigorous in our characterisation and isolation of conformational states, used a Markov-model to argue about their relative live times, and subjected selected structures to pK_a-value estimation and steered MD pulling experiments. Also the number of considered langerin sub-systems grew quickly once we considered more and more protonation, calcium-binding, and unfolding states. Systems directly related to phalloidin were further investigated later in the group in terms of the α-amanitin and its derivates. Still, the fundamental approach remained always the same: we tried to derive knowledge from a dynamic series of atomic positions or respective probability distributions.

In this context, another relationship that can be found in all our work here, lies in the way how we addressed problems conceptually. It can be taken as an underlying philosophy that we preferred tools that are easy to grasp, easy to operate, and cost efficient over those that are rather elaborate, complicated, and expensive. This is reflected by the use of classical MD in the first place and we stretched the domain of applicability with our projects in which we predicted product ratios as the outcome of an organic reaction and modelled a metal containing protein under pH changes. Both situations can be more accurately described with higher levels of theory—reactive dynamics, constant-pH dynamics, QM/MM or *ab-initio* dynamics to only name a few—but at much greater cost. The same goes for other tools where we used for example readily available empirical pK_a-value estimates and rupture force affinity proxies instead of more exact but tricky free energy perturbation schemes. Our reasoning always was that the most accurate measure could still be of limited explanatory power if we could only afford to obtain it for a very small number of molecular conformations. We valued the thorough and extensive sampling of the conformational ensembles, to attain a Boltzmann-weighted distribution of what we measure, higher than accurate single observations for small data sets. Similarly, we cut down our studied systems to a minimal relevant subset, i.e. integral parts of phalloidin precursors ignoring peripheral chains or protecting groups and only the binding domain of a single langerin monomer. The quality of our results justifies this approach. In parts, we reach the limitations of the tools we use and we recognise where more apt descriptions are in order. We exhausted simplistic investigation possibilities first from which refined studies could be started as appropriate.

Finally, a last important element that the phalloidin and langerin projects have in common is the close connection to laboratory experiments. Without the background to eventually rationalise and improve the phalloidin synthesis in cooperation with the Süssmuth group[1] and without the agreement between the theoretical prediction and the practical result, the whole study would have been in vain. Similarly, the starting point for the langerin investigation were calcium affinity measurements, NMR-experiments, and point mutations done by the Rademacher group[2] without which the study would never have been kicked of. MD simulations are a powerful tool to get atomistic insight into the behaviour of various systems but it is the exchange and balance with experimental data that makes them really useful.

The third project I worked on falls into the category of method development although it is still tightly bound to the previous application examples. As the production of conformational data sets and their characterisation is the basis of our studies, tools to extract conformational states from MD simulations are invaluable to our undertakings. We use low-dimensional projections of MD trajectories to bring out significant aspects of our systems. Eventually, we relied on a specific analysis as the key element to separate conformational states in these projections, referred to as *clustering*.

[1] R. Süssmuth, Biological Chemistry, Technische Universität Berlin
[2] C. Rademacher, Molecular Drug Targeting, Universität Wien

Clustering allows us to identify groups of related molecular structures in terms of coherent sets, which can be characterised individually, compared to each other and to the complete ensemble, and finally linked to potential functionality.

Molecular conformational states are associated with potential energy minima, i.e. sets of structures disjoint from other structures by sufficiently high energy barriers. A particular formulation of clustering—density-based clustering—finds clusters matching exactly this notion: clusters are dense groups of data points separated by low density. In the group, we use and develop a density-based clustering procedure called common-nearest-neighbour (CommonNN) clustering, which is particularly well suited for the application to molecular data sets. The analytic challenge posed by our langerin data set stimulated the re-implementation of this clustering approach in an easy to use and efficient Python package. It allowed us to process large amounts of structures, to screen many cluster parameter combinations, and to explore the hierarchical nature of the data. A clustering of this kind led to the discovery of the prominent langerin conformation that constitutes our central result and was the basis of a kinetic model that put it in proportion to other conformational states.

The work on the CommonNN clustering methodology in parallel to our langerin study is an excellent example for how application and method development can go successfully hand in hand. Our improvements to the clustering facilitated our langerin analysis. At the same time, the clustering profited from the direct validation via the application and was developed into a direction targeted towards practical usefulness. In this course, the algorithms used for the clustering were conceptually simplified, the internal components of the procedure were reviewed in detail, and the feature set of the clustering package was greatly enriched. In particular, the formulation of the clustering in a hierarchical manner beyond the traditional threshold-based picture might take the procedure to the next level in the near future. From a technical implementation standpoint, the generic formulation of the clustering procedure that we have employed, greatly improves the reusability of the code for a broad array of possible applications.

To sum up, the projects we studied helped to shed light onto the specific scientific problems they entail. Beyond that, I hope that our general approach could serve as a template for how to address similar research questions and can contribute incrementally to the further maturing of MD simulations and their analysis in practically relevant research. In particular, I would like to see density-based clustering promote into the standard method of choice to isolate conformational states from configurational trajectories.

Part II.

Theoretical basics

{ 4 }

The Molecular Dynamics formalism

Ingredients to run basic molecular simulations

In this chapter, I would like to discuss (a few of) the integral parts that are needed to run classical MD simulations. The selection of things that are addressed in detail is sort of biased towards what I consider most important to understand the basics. MD simulations do, however, require a complex interplay of a multitude of technical details in practice, which would go far beyond what I can describe here. I will try to point out where the provided overview is limited. To begin with, I give a rough example for a possible program structure that can be used to realise finite time numerical simulations of molecular systems in general. The following sections will then give credit to individual aspects.

Figure 4.1 summarises a typical realisation of a MD simulation in a respective program. Roughly, a MD simulation takes a starting structure (a configuration) of a molecule as input and produces a time series of atomic positions by repeating two fundamental operations: a computation of current forces acting on each atom, and the propagation of positions according to these forces for a tiny step forward in time. A minimal setup of a molecular system in the context of such a simulation brings together the following elements: 1) a current configuration, i.e. usually cartesian coordinates for each particle contained in the system. 2) Current particle velocities (optional on initial input). 3) A set of interactions, e.g. potentials to model bonds, angles, dispersive interactions, external influences etc. 4) A list of drivers through which the simulation should be progressed. 5) Optionally a mechanism to account for periodic boundary conditions (PBCs). 6) Optional reporters that collect and output information about the current state of the simulation.

parts of a simulation

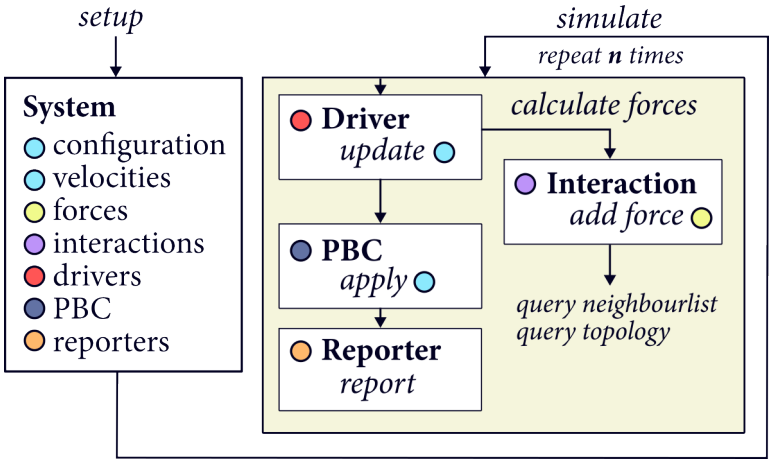


Figure 4.1 Minimum simulation scheme A simulated system from the MD perspective is basically a set of atomic coordinates (the configuration) and velocities. These are updated during a simulation in steps by drivers that may evaluate present interactions (compute forces) and propagate the configuration. For the evaluation, atom specific (topology) parameters may need to be looked up and supporting elements like neighbourlists may be used. Optionally, PBCs can be applied and information about the run can be reported.

When a simulation is started, a number of steps is executed sequentially in order to propagate the system further in time. A driver goes through the list of specified interactions and triggers the computation of forces acting on the atoms in the system based on the current coordinates. Depending on the interaction, this can entail different logic. Basic types of commonly used interactions are

discussed in section 4.2. For simple harmonic bonds for example, it has to be known on which pair of atoms it should be evaluated and which parameters (equilibrium bond length and force constant) should be used for that. The mechanisms of how to get these information, i.e. the way how *topological information* is stored and accessed in a program, can be fairly intricate and should not be discussed here. Non-bonded interactions that should not be evaluated for fixed pairs but rather for atoms at a certain current maximum distance from each other, may need to make use of structures like neighbourlists (section 4.5), which in turn have to be updated as needed. What an interaction actually calculates can be arbitrarily complex but what it outputs is always an additive force contribution. Subsequently, the driver uses the summed up forces to modify the current coordinates and optionally also velocities according to an integration scheme. A few basic integrators are discussed in section 4.6. Drivers decide when (and how often) to evaluate specific interactions and may fulfil also other roles, modifying the state of the system in any possible way. For example they can act as thermostats (discussed in section 4.8). After each driver has successfully done its part, the coordinates of the system's particles may be corrected for PBCs of which basic schemes are found in section 4.4. Finally, different reporters can be used to write the current coordinates to disk (every other time step) or print out information like temperature, energies, or the simulation progress. The reported output will be the basis for subsequent analyses (see chapter 5). These steps are repeated until the desired simulation length has been reached.

In parts, what will be described in the following sections of this chapter is underpinned by a small MD program implemented in Cython that allows example implementations for educational purposes while being a bit more robust and efficient than an ad-hoc implementation.[1] Most widely used software packages to realise MD simulations are highly optimised for performance and their code base can be daunting and hard to navigate in search for a specific implementation detail. An exception is Lumol, implemented in Rust.[2]

justification

The theoretical basis for a simulation like this, are Newton's equations of motion (see section 4.1). We treat the atoms we simulate as Newtonian particles that interact via empirical potentials and approximate the complex, chaotic solutions to the underlying mechanics numerically. That this approach is justified and can produce practically usable results, rests upon a number of assumptions and theorisations. On a quantum mechanical level, we need the Born-Oppenheimer approximation that allows us to treat the motion of atomic cores and electrons separately.[83] MD propagates the cores classically on a potential energy surface governed by the electronic structure of the molecular system. This energy surface is in turn substituted with an approximate function, the quality of which strongly influences the validity of the simulation. In the same line of basic argumentation falls the Ehrenfest theorem, according to which under certain conditions the time-dependence of the expectation values of the quantum mechanical position and momentum operator resemble the motion of a classical particle.[84] From the standpoint of statistical mechanics, the most important point might be the ergodic theorem (see chapter 5) that equates macroscopic ensemble averages with time-averages of single molecules as they are accessible from simulations. On a technical side, MD has to rely on accurate propagation schemes that minimise numerical errors and ensure the reproduction of a thermodynamic ensemble.[85] While it is under debate if classic numeric integrators can 'shadow' the true dynamics of a system,[86] it is widely excepted that MD simulations are valid in a statistical sense in terms of ensemble distributions, averages, and autocorrelation functions.[87, 88] Finally, the successful execution of a simulation and its potential to model molecular behaviour realistically, depends on a multitude of fine-tunable parameters and settings.[89]

[1] See the repository on GitHub: `github.com/janjoswig/Eski`
[2] See the repository on GitHub: `github.com/lumol-org/lumol`

4.1 Equations of motion

Newton's second law of motion tells us that the acceleration of point particles (or rigid bodies approximated by their center of mass) is proportional to the net force acting upon them. The dedicated fundamental equation of motion is essentially a second order differential equation: *Newton's 2nd law*

$$\boldsymbol{F} = \boldsymbol{M}\frac{\mathrm{d}^2\boldsymbol{q}}{\mathrm{d}t^2}\,, \tag{4.1}$$

where t denotes time and $\boldsymbol{q}(t) = (q_1, \ldots, q_n)$ is a position vector in a suitable set of n chosen coordinates to describe a physical system at a specific point in time. The mass matrix $\boldsymbol{M}$ is a diagonal matrix[3] whose elements hold a corresponding mass for each of the n positional coordinates. Consequently, $\boldsymbol{F}$ is a vector of n force components. With $\mathrm{d}\boldsymbol{q}/\mathrm{d}t = \dot{\boldsymbol{q}} = \boldsymbol{v}(t)$ as the instantaneous velocity vector, $\mathrm{d}^2\boldsymbol{q}/\mathrm{d}t^2 = \ddot{\boldsymbol{q}} = \boldsymbol{a}(t)$ as the systems acceleration, and the definition of momentum $\boldsymbol{p} = \boldsymbol{M}\boldsymbol{v}$, other possible formulations of Newton's second law are

$$\boldsymbol{F} = \boldsymbol{M}\boldsymbol{a} = \boldsymbol{M}\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t}\,. \tag{4.2}$$

The latter expression in terms of the systems momenta is often referred to as the most general one because it does not necessitate that the particle masses are actually constant. As far as we should be concerned here about molecular mechanics, however, masses will always be in fact constant, which means in particular that we will stay exclusively within the non-relativistic limit.[4] The total force $\boldsymbol{F}$ acting on a system can in general be a function of the systems current positions, velocities, and time $\boldsymbol{F}(t, \boldsymbol{q}, \dot{\boldsymbol{q}}) = (F_1, \ldots, F_n)$, depending on which forces are actually present. To describe a physical system by Newton's equation, essentially means to write down all the acting forces, i.e. to find the corresponding *force law*. Individual forces have the important property that they are additive vector quantities so that the net force $\boldsymbol{F}$ on a system can be expressed as a sum

$$\boldsymbol{F} = \sum \boldsymbol{F}_{\text{contrib}} \tag{4.3}$$

of contributing forces $\boldsymbol{F}_{\text{contrib}}$, which can each influence all or only a subset of the systems coordinates. Once the force law is formulated, the resulting second order differential equation can be solved analytically in some simple cases. Given two initial conditions—usually set positions and velocities—one might be able to find an agreeing equation for $\boldsymbol{q}(t)$ that will describe the system at hand at all times. For most practical cases, however, one needs to resort to numerical approximations as solutions.

Let's consider a few very simple examples, beginning with a system of a single, free particle with mass m in two dimensions. The current position of the particle is given in cartesian coordinates as $\boldsymbol{q} = (x, y)$ with velocity $\boldsymbol{v} = (\dot{x}, \dot{y})$, while the adjective 'free' connotes that the system has two corresponding *degrees of freedom* and can move unrestrictedly. In the case of this particle being subject to a constant force or no force at all, the force law will be just $\boldsymbol{F}_{\text{const}} = m\mathrm{d}^2\boldsymbol{q}/\mathrm{d}t^2 = m\boldsymbol{a}_{\text{const}}$. With an initial particle position $\boldsymbol{q}_0$ and velocity $\boldsymbol{v}_0$, a solution to this equation would be *constant force example*

$$\boldsymbol{q}(t) = \boldsymbol{q}_0 + \boldsymbol{v}_0 t + \frac{\boldsymbol{a}_{\text{const}}}{2}t^2 \tag{4.4}$$

with $\dot{\boldsymbol{q}}(t) = \boldsymbol{v}_0 + \boldsymbol{a}_{\text{const}}t$ and $\ddot{\boldsymbol{q}}(t) = \boldsymbol{a}_{\text{const}}$, where $\boldsymbol{a}_{\text{const}}$ is possibly 0. In accordance to Newton's first law, a particle that experiences no force is either at rest ($\boldsymbol{v}_0 = 0$) or moves with constant velocity along

[3] If the mass is actually the same for all coordinates (e.g. if only one particle is considered), the mass is usually given just as a scalar m.

[4] A formulation of Newton's second law that allows for changing mass is $F = \mathrm{d}p/\mathrm{d}t = M\,\mathrm{d}v/\mathrm{d}t + v\,\mathrm{d}M/\mathrm{d}t$. This can be used even in the classical limit for example to account for airplanes losing mass due to fuel consumption.

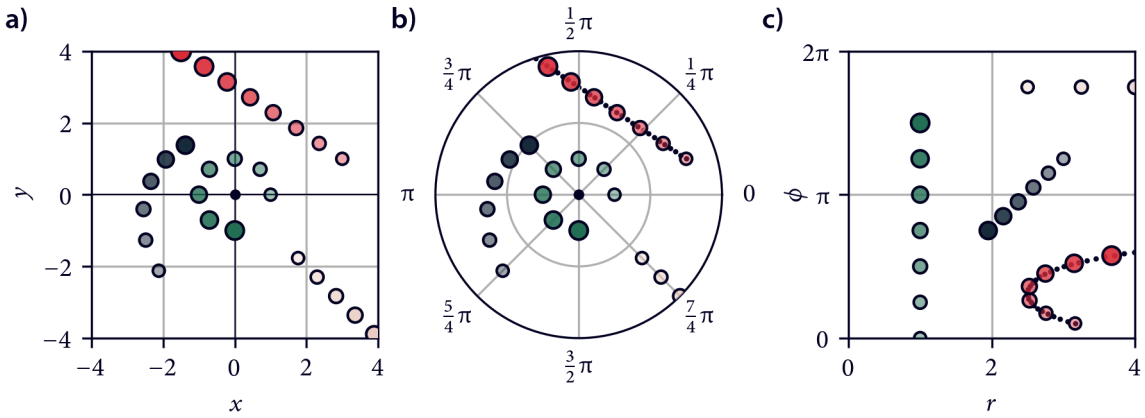


Figure 4.2 Free particles in cartesian and polar coordinates
Plots in **a)** cartesian coordinates, **b)** polar coordinates embedded in a cartesian reference frame, and **c)** polar coordinates as a separate projection. The red particle motion follows equation 4.4 with no acceleration in cartesian coordinates and has been converted to polar coordinates for the other subplots. The black dotted line is a numeric solution to the same trajectory but propagated in polar coordinates, which necessitates the consideration of coordinate fictitious forces. The green, pale, and dark particle trajectories follow equation 4.4 in polar coordinates and are translated into cartesian coordinates for subplot **a)**. While the pale motion can be indeed understood as force-free, the circular motion described by the green and dark particle implies the presence of forces.

a straight line $\boldsymbol{q}(t) = \boldsymbol{q}_0 + \boldsymbol{v}_0 t$. The red particle in figure 4.2a illustrates this case for $\boldsymbol{q}_0 = (3,1)$ and $\boldsymbol{v}_0 = (-1.5,1)$.

polar coordinates

We want to realise, however, that the choice of cartesian coordinates is not at all binding. Let's for example consider the description of a free particle in two dimensions with polar coordinates where the particle position is given as $\boldsymbol{q}^{\text{pol}} = (r, \phi)$ with the radial distance of the particle to the origin r and the rotational angle ϕ with respect to an axis of reference (say the x-axis). The polar velocities are the changing rates with respect to these coordinates $\boldsymbol{v}^{\text{pol}} = (\dot{r}, \dot{\phi})$. Positions in polar coordinates and cartesian coordinates can be interconverted via $x = r\cos\phi$ and $y = r\sin\phi$, or vice versa $\phi = \text{atan2}(y, x)$ and $r = \sqrt{x^2 + y^2}$. The encoded information is exactly the same but a particular kind of particle motion takes different forms in both these coordinate systems. Depending on the specific situation, one set of coordinates may be preferable over the other.

If we now re-use the kinematic equation 4.4 as $\boldsymbol{q}(t) = \boldsymbol{q}_0 + \boldsymbol{v}_0 t$ with $\boldsymbol{q}_0^{\text{pol}} = (1,0)$, $\boldsymbol{v}_0^{\text{pol}} = (0, \pi/4)$ and no respective acceleration as shown in figure 4.2c with the green particle, we again observe a straight line trajectory—at least in the $r\phi$-plot. In cartesian coordinates on the other hand it becomes obvious that the same trajectory does actually describe a uniform circular motion. This is because the situation with a constant angular velocity $\dot{\phi}$ implies a constant acceleration of the particle towards the center of the rotation and is as such not without the influence of a force. The acceleration from the cartesian perspective expressed in polar coordinates is in general $\boldsymbol{a}^{\text{cart}} = (\ddot{r} - r\dot{\phi}^2)\hat{\boldsymbol{r}} + (r\ddot{\phi} + 2\dot{r}\dot{\phi})\hat{\boldsymbol{\phi}}$ and the velocity is taken as the tangential velocity $\boldsymbol{v}^{\text{cart}} = \dot{r}\hat{\boldsymbol{r}} + r\dot{\phi}\hat{\boldsymbol{\phi}}$. The unit vectors of the polar coordinate system are $\hat{\boldsymbol{r}} = \cos\phi\hat{\boldsymbol{x}} + \sin\phi\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{\phi}} = -\sin\phi\hat{\boldsymbol{x}} + \cos\phi\hat{\boldsymbol{y}}$, with for example $\hat{\boldsymbol{x}} = (1,0)$ and $\hat{\boldsymbol{y}} = (0,1)$ being the standard cartesian unit vectors. These polar unit vectors are rotating themselves wih ϕ, which gives rise to the fictitious terms $r\dot{\phi}^2$ (referred to as the centrifugal term) and $2\dot{r}\dot{\phi}$ (called the Coriolis term). When we have $\dot{r} = 0$, $\ddot{r} = 0$, and $\ddot{\phi} = 0$, a particle in polar coordinates will still be subject to an acceleration $\boldsymbol{a}^{\text{pol}} = (-r\dot{\phi}^2, 0)$ just because the coordinate system in wich it is described rotates.

Consequently, the force law for particle motion in polar coordinates does in fact look a bit more complicated than in cartesian coordinates:

$$\boldsymbol{F} + m\begin{pmatrix} r\dot{\phi}^2 \\ -2\dot{r}\dot{\phi} \end{pmatrix} = m\begin{pmatrix} \ddot{r} \\ r\ddot{\phi} \end{pmatrix}. \tag{4.5}$$

When we want to use equation 4.4 as a solution to this problem, we need to set the actual forces acting on the system equal to the coordinate fictitious ones, i.e. in particular for the circular motion we need to include a centripetal force $F_{\text{centripetal}} = -mr\dot{\phi}^2$. The extra terms in the force law make the derivation of an analytical solution in the general case quite difficult.

Figure 4.2 shows two further examples for particle trajectories in cartesian and polar coordinates. The pale dots mark an 'outward' motion for $\boldsymbol{q}_0^{\text{pol}} = (2.5, 7/4\pi)$, $\boldsymbol{v}_0^{\text{pol}} = (1, 0)$, and $\boldsymbol{a}^{\text{pol}} = (0, 0)$ in which case both coordinate fictitious force terms amount to 0 and the dark dots describe an 'inward spiral' motion for $\boldsymbol{q}_0^{\text{pol}} = (3, 5/4\pi)$, $\boldsymbol{v}_0^{\text{pol}} = (-0.3, \pi/7)$, and $\boldsymbol{a}^{\text{pol}} = (0, 0)$, where both centrifugal and Coriolis contribution play a role. The black dotted line in figure 4.2b represents a numeric approximation to the force-free straight line motion propagated in polar coordinates according to an Euler-Cromer integration scheme (see section 4.6).

To find an analytical solution to the force law of a system is only feasible in very few cases. Examples, besides the free particle motion under the influence of a constant force discussed above, would be a harmonic oscillator or (with considerable effort) a pendulum under the influence of gravity,[90] if the problem is smartly reduced to one dimension by choosing a suitable internal coordinate. The pendulum problem in two planar cartesian coordinates becomes more difficult. Numerical approaches can provide an approximate solution to equations of motion, given a set of initial conditions. But also without actually employing a numeric scheme, it is possible already to gain some insight about a system by just looking at how the position and momentum combined would change for different initial conditions. A visualisation in which the change in position $\dot{q}_i = v_i = p_i/m$ is plotted versus the change in momentum $\dot{p}_i = F_i$ with respect to the ith system coordinate is called a phase (space) plot, diagram, or portrait. Figure 4.3 shows such vector field plots for a one-dimensional linear motion, the vertical component of a projectile, and a pendulum with added friction term. Starting at any point in the plot, we can anticpate how position and momentum will change with time and follow a specific trajectory the system will take. A numeric computation of trajectories can achieve in principle the same: the production of a trajectory through phase space.

phase plots

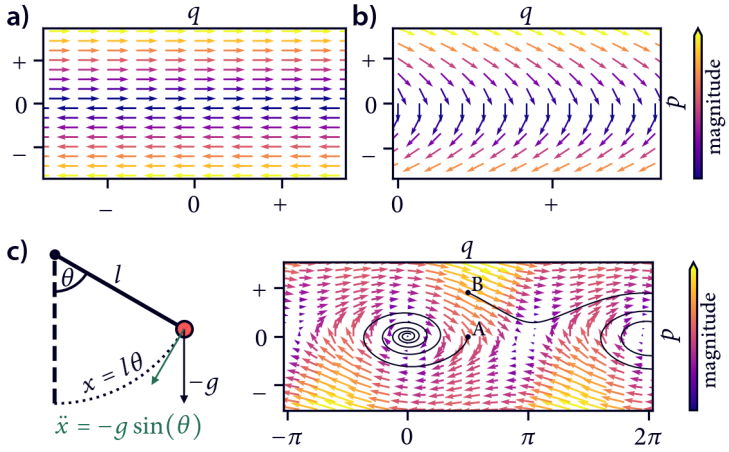


Figure 4.3 Phase diagrams The change in position and momentum with respect to one coordinate is represented for pairs of starting conditions (q_0, p_0) by a vector $(p_0/m, F)$. Vectors can be optionally normalised in length and their magnitude is indicated by color. **a)** Particle being either at rest or moving with constant velocity in one direction. **b)** Particle under constant acceleration in negative direction. **c)** Damped pendulum motion according to $\ddot{\theta} = -g/l \sin(\theta) - \gamma\dot{\theta}$. For two starting conditions A and B, the trajectory was computed using an Euler-Cromer integrator.

The derivation of force laws for particle motion can be complex. A helpful concept may be to inspect the Lagrangian of a system

$$\mathcal{L} = E_{\text{kin}} - E_{\text{pot}}\,, \tag{4.6}$$

acknowledging that forces and motion have their origin in energetic terms. Equations of motions can be generated from it, using the Euler-Lagrange equation (assuming the system has a stationary state):

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}} = 0\,. \tag{4.7}$$

Lagrangian

For the case that the system's potential energy does only depend on particle positions and the kinetic energy only on velocities (which is the case for mechanical systems we should be concerned here with), we have in particular that $\partial\mathcal{L}/\partial\boldsymbol{q} = -\partial E_{\text{pot}}/\partial\boldsymbol{q} = \boldsymbol{F}$ and recover Newton's equation (equation 4.1). Forces that fulfil this requirement of coming from a position-only dependent potential can be called *conservative* forces. As an important property, work done in a field of conservative forces is independent of the path taken between the two end states. Non-conservative (dissipative) forces that depend on particle velocities can be present in a system as well but will make equation 4.7 not equate to 0, that is the total energy of the system is not constant. Using the Lagrange formalism, one can describe particle motion in any set of coordinates, possibly adding a number of constraints to obtain a system with fewer effective degrees of freedom.[91] Pre-requirement for the set of generalised coordinates is that they are complete, independent, and holonomic. Independency means that if we fix the motion along one coordinate, all the others still need to be able to explore their whole value range freely. As a consequence, there must not be any redundancies in the used coordinates. Completeness means that we need to be able to describe the state of the system at all times. And holonomicity states that the number of used coordinates needs be equal to the actual number of degrees of freedom in the system. In particular this means that used constraints have to be expressible as functions of coordinates only. For the pendulum problem in figure 4.3c for example, the use of two cartesian coordinates x and y violates independency. Holding either one of them fixed, will not allow the system to move anymore. Intrinsically, the pendulum has only one degree of freedom. We can add a holonomic constraint, though, to replace x and y with a single new coordinate.[92] Alternatively, the system's Hamiltonian can be formally used to derive equations of motion.

For molecular systems in the context of MD, we construct a position dependent potential energy function and correspondingly conservative forces from empirical contributions. Which molecular interactions are typically considered in this, will be the topic of the next section.

4.2 Types of molecular interactions

standard forces

The forces acting on particles that are considered in molecular simulations are usually corresponding to inter- and intra-molecular interactions of some kind. In classic MD, it is common to construct the potential energy expression for a given system additively for example from the following contributions:

$$E_{\text{pot}} = \underbrace{E_{\text{bonds}} + E_{\text{angles}} + E_{\text{torsions}}}_{\text{bonded}} + \underbrace{E_{\text{dispersion}} + E_{\text{electrostatics}}}_{\text{non-bonded}}\,. \tag{4.8}$$

Atoms (or groups of atoms) are treated as Newtonian particles with a certain mass and fixed partial charge, and the modelled interactions between them are supposed to represent chemical bonds (involving two atoms), valence angles (involving two geminal atoms via a third central atom, i.e. two bonds), and torsion angles (involving two vicinal atoms over a third and fourth atom, i.e. three bonds),

as well as dispersion and electrostatics between non-bonded atom pairs. The above terms represent summations over all the respective individual interactions, for instance the set of all chemical bonds $\mathcal{B}$ between atoms a and b present in a system $E_{\text{bonds}} = \sum_{a,b\in\mathcal{B}} E_{\text{bond}}(a,b)$, etc. Other interactions are possible, as for example improper dihedral angle or mixed bond-angle terms, constraining forces, and forces from external fields. Historically, there were also dedicated interaction terms used for hydrogen bonds.[93]

Each of these contributions can take on different functional forms. The combination of interactions that is employed in a simulation is called a *force field* or *potential field*, providing energies and forces for every system configuration. All considered forces also form the *topology* of a molecular system. The details of how topologic information is handled by different simulation programs, i.e. how the forces and corresponding parameters for complex systems are communicated, stored, and accessed efficiently, can be fairly intricate, however, and I will not go into it here. A force field is meant to provide an approximation to the actual forces acting on atoms, originating from the electronic structure of a system and interactions of the atomic cores in a QM description. It breaks down the (probably unknown) potential energy surface into a set of quasi-physical contributions, which can be parametrised, cheaply evaluated (analytically), and mixed and matched for a wide array of molecular systems.

Picking a set of reasonable interactions by hand can be hard for complex systems. For this reason, a multitude of consistent template force frameworks exist for various types of systems, i.e. well designed pools of interactions from which the suitable ones can be selected more or less automatically for specific molecules. These are also referred to as force fields and prominent examples are those from the AMBER, CHARMM, GROMOS, or OPLS family, which are often associated with respective simulation program suites.[94] They are based on slightly varying philosophies for the determination of interaction parameters. Equation 4.8 may look different in detail for each of them. In its original form, which still remains largely valid, though, this fundamental equation was supposedly published first in 1969.[27]

In the following, examples of a few typically used potentials will be discussed briefly. Figure 4.4 gives on overview on the definition of bonds and (torsion) angles that these potentials depend on.

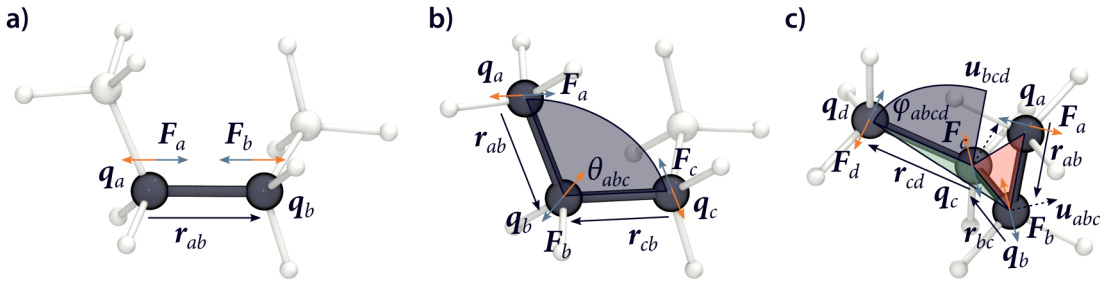


Figure 4.4 Bonded interaction types and forces

a) Pair of atoms at positions q_a and q_b connected by a chemical bond. The acting forces F_a and F_b depend on the bond distance r_{ab} and either pull the two atoms closer together (blue) or further away from each other (orange). **b)** Three atoms (q_a, q_b, q_c) connected by two bonds forming an angle θ_{abc}. The respective forces on the atoms act to either tighten (blue) or widen this (angle). **c)** Four atoms (q_a, q_b, q_c, q_d) over three bonds forming the dihedral angle ϕ_{abcd}, which is equal to the angle between the normal vectors u_{abc} and u_{bcd}, i.e. the angle between the two planes described by $r_{ab} \times r_{bc}$ and $r_{bc} \times r_{cd}$. The ϕ-dependent forces act on the atoms involved to either tighten or widen this angle. Similarly, improper dihehdral angles are defined as the angle between any two planes formed by four atoms not necessarily in this typical torsion arrangement.

A COMMONLY USED, PARAMETRISED MODEL FOR ATOMIC BONDS in classic MD is the harmonic spring potential

$$E_{\text{bond}} = \frac{1}{2}k(r_{ab} - r_0)^2\,, \tag{4.9}$$

harmonic bond

where r_{ab} is the (euclidean) distance between two bonded atoms a and b at positions $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$, and r_0 is the equilibrium distance at which the interaction energy is minimal (here $E = 0$).[5] The strength of the bond is scaled with the force constant k: the larger k, the steeper and narrower the potential. We have the distance vector $\boldsymbol{r}_{ab} = \boldsymbol{q}_b - \boldsymbol{q}_a$ with length $r_{ab} = ||\boldsymbol{r}_{ab}||$ and normalisation $\hat{\boldsymbol{r}}_{ab} = \boldsymbol{r}_{ab}/||\boldsymbol{r}_{ab}||$. The internal force implied by the potential is

$$F = -\frac{\mathrm{d}E}{\mathrm{d}r_{ab}} = -k(r_{ab} - r_0)\,. \tag{4.10}$$

While this gives us a value for how the potential changes with the bond distance, for the evaluation of the force in a simulation by its components we are actually interested in how the potential changes with the position of the bonded particles, e.g. $\partial E/\partial \boldsymbol{q}_b$. From this, we can get the force vector $\boldsymbol{F}_b$ acting on particle b as

$$\boldsymbol{F}_b = -\frac{\mathrm{d}E}{\mathrm{d}r_{ab}}\frac{\partial r_{ab}}{\partial \boldsymbol{q}_b} = F\hat{\boldsymbol{r}}_{ab}\,. \tag{4.11}$$

The bonded interaction is symmetric so that $\boldsymbol{F}_a = -\boldsymbol{F}_b$. Figure 4.5 shows a harmonic spring potential and the resulting force on atom a.

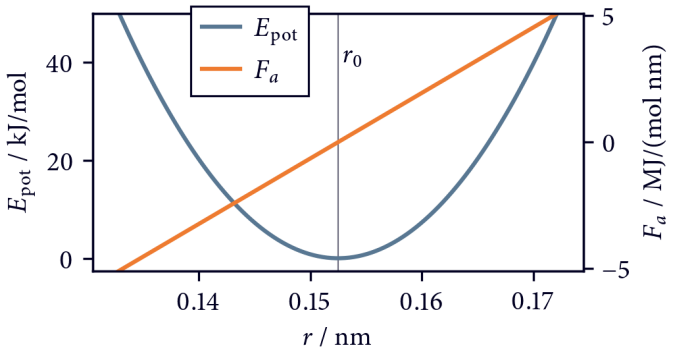


Figure 4.5 Harmonic bond potential Distance dependent potential energy according to equation 4.9 for a C-C single bond with $r_0 = 0.1525\,\text{nm}$ and $k = 259,408\,\text{kJ/(mol nm}^2)$, and corresponding force acting on one of the particles. Parameters taken from the AMBER99SB-ildn force field.[95]

When tabulated parameters for this potential are looked up, attention must be paid to how the potential energy function is defined exactly (this actually extends to any kind of potential). In AMBER for example, equation 4.9 is understood alternatively as $E_{\text{bond}} = k(r_{ab} - r_0)^2$, which implies a force constant differing by a factor of 2. AMBER force fields ported to GROMACS on the other hand use equation 4.9 as is. It should be also checked carefully in which units force constants and other parameters are given. While k in molecular units can for example be communicated in kJ/(mol nm^2) (as in GROMACS), kcal/(mol Å^2) are also common (as in AMBER). In SI(-derived) units, this quantity is given in kg/s^2 or N/m. Force constants can also be derived from and converted to vibrational (angular) frequencies ω as

tabulated paramters

$$\omega = \sqrt{\frac{k}{\mu}}\,, \tag{4.12}$$

[5] For the evaluation of interaction forces in a simulation, the actual energy value at the minimum is not important. If actual energies should be calculated, however, a suitable (tabulated) minimum value for the binding energy needs to be subtracted.

where μ is the reduced mass of the partaking atoms $\mu = (m_a m_b)/(m_a + m_b)$. Frequencies in turn relate to the perhaps more abundantly tabulated wavenumbers $\tilde{\nu}$ or wavelengths λ via

$$\tilde{\nu} = \frac{1}{\lambda} = \frac{\omega}{2\pi c} \tag{4.13}$$

with speed of light c. For the force constant used in figure 4.5, one obtains for example $\tilde{\nu} = 1103.86/\text{cm}$, which corresponds to a vibrational period of about $T = 2\pi/\omega = 0.03\,\text{ps}$.

A harmonic potential as in equation 4.9 is not the only possible option to model chemical bonds in MD simulations. Another example would be the more realistic yet more complicated Morse potential[96] with

Morse potential

$$E_{\text{bond}} = \epsilon\left[1 - \exp\left(-\alpha(r_{ab} - r_0)\right)\right]^2 \quad \text{with } \alpha = \sqrt{\frac{k}{2\epsilon}} \tag{4.14}$$

and implied force

$$F = 2\epsilon\alpha \exp\left(-\alpha(r_{ab} - r_0)\right)\left[1 - \exp\left(-\alpha(r_{ab} - r_0)\right)\right] . \tag{4.15}$$

Here, an additional parameter ϵ controls the depth of the potential energy well compared to a dissociation limit. As figure 4.6 illustrates, a Morse potential differs quite substantially from the harmonic picture with respect to a steeper increase of the energy at short bond distances and asymptotic behaviour at large distances. In terms of what is accessible to a molecular system at thermal energy $k_B T$, both potentials agree, however, quite well so that the simple harmonic form can be mostly used safely.

Bond potentials that account for a correct anharmonic description are among other things interesting in simulations where particle dissociation should be incorporated[97] or where vibrational spectra should be reproduced.[98]

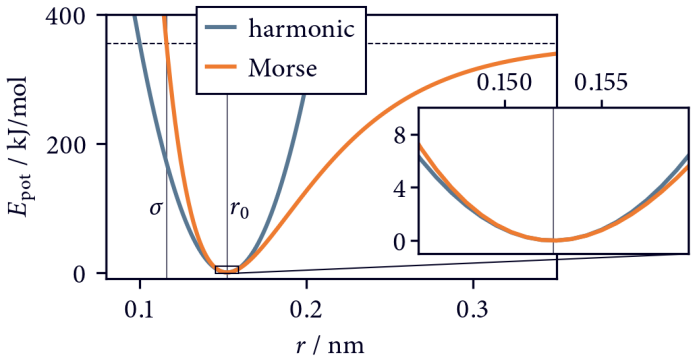


Figure 4.6 Harmonic vs. Morse potential
Distance dependent harmonic potential energy for a C-C single bond (see also figure 4.5) compared to a Morse potential according to equation 4.14 with $\epsilon = 355.64\,\text{kJ/mol}$. $V(\sigma) = \epsilon$, $\sigma = -\ln(2)/\alpha + r_0$.

Similar to chemical bonds between two atoms, valence angles formed by three atoms each can as well be modelled with a harmonic term

harmonic angle

$$E_{\text{angle}} = \frac{1}{2}k(\theta_{abc} - \theta_0)^2 \tag{4.16}$$

with force

$$F = -\frac{\mathrm{d}E}{\mathrm{d}\theta_{abc}} = -k(\theta_{abc} - \theta_0) . \tag{4.17}$$

Calculating the actual forces on the atoms a, b, and c involved is, however, a bit less straightforward than in the bond case. The angle θ_{abc} can be computed as the scalar product of the distance unit vectors between the central and the other two atoms

$$\cos\theta_{abc} = \hat{\boldsymbol{r}}_{ab} \cdot \hat{\boldsymbol{r}}_{cb} \tag{4.18}$$

and through the derivatives

$$\frac{\partial \theta_{abc}}{\partial \boldsymbol{q}_a} = \frac{\cos\theta_{abc}\hat{\boldsymbol{r}}_{ab} - \hat{\boldsymbol{r}}_{cb}}{\sin\theta_{abc} r_{ab}} \quad \text{and} \tag{4.19}$$

$$\frac{\partial \theta_{abc}}{\partial \boldsymbol{q}_c} = \frac{\cos\theta_{abc}\hat{\boldsymbol{r}}_{cb} - \hat{\boldsymbol{r}}_{ab}}{\sin\theta_{abc} r_{cb}} \tag{4.20}$$

we get the force vectors $\boldsymbol{F}_a$ and $\boldsymbol{F}_c$ in analogy to equation 4.11. The force on the central atom can then be obtained as $\boldsymbol{F}_b = -(\boldsymbol{F}_a + \boldsymbol{F}_c)$, because the total internal force should amount to zero. Instead of a plain harmonic potential, angles can also be modelled by a cosine potential that allows angle transitions across 180° to an equivalent equilibrium angle with energy

cosine angle

$$E_{\text{angle}} = \frac{1}{2}k(\cos\theta_{abc} - \cos\theta_0)^2 \tag{4.21}$$

and force

$$F = k(\cos\theta_{abc} - \cos\theta_0)\sin\theta_{abc}\,. \tag{4.22}$$

Figure 4.7 compares the two potentials for the H-O-H angle in water. Other special angle potentials are for example the restricted-bending and the Urey-Bradley potential used in CHARMM.[99]

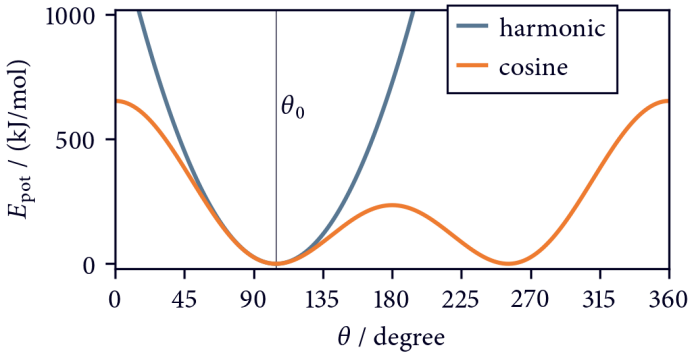


Figure 4.7 Harmonic vs. cosine angle potential Angle dependent harmonic potential energy according to equation 4.16 for a H-O-H angle with $\theta_0 = 104.52°$ and $k = 836.8\,\text{kJ/(mol rad}^2\text{)}$, compared to a cosine potential according to equation 4.21. Parameters taken from the AMBER99SB-ildn force field.[95]

Torsion angles are in principle just regular angles (see figure 4.4) and can also be treated harmonically in analogy to equation 4.16. For improper dihedrals, this is actually common practice when a certain molecular geometry should be enforced, e.g. the planarity of aromatic rings or the relative orientation of substituents on a tetrahedron. Chemically it makes more sense, though, to model proper torsion dihedrals periodically for which a possible potential would be

periodic torsion

$$E_{\text{torsion}} = A[1 + \cos(n\phi - \delta)] \tag{4.23}$$

with the energy amplitude A that takes the place of a force constant: the larger A, the stronger the interaction. Two additional parameters, the multiplicity n and phase shift δ, control the shape of the potential. The respective torsion angle force is given by

$$F = nA\sin(n\phi - \delta)\,. \tag{4.24}$$

To calculate the force exerted on the atoms involved during a simulation we can first compute the dihedral angle as

$$\phi_{abcd} = \text{atan2}(r_{\text{bc}}(\boldsymbol{u}_{bcd}\cdot\boldsymbol{r}_{ab}), \boldsymbol{u}_{abc}\cdot\boldsymbol{u}_{bcd})\,, \tag{4.25}$$

where $\boldsymbol{u}_{abc} = \boldsymbol{r}_{ab}\times\boldsymbol{r}_{bc}$ and $\boldsymbol{u}_{bcd} = \boldsymbol{r}_{bc}\times\boldsymbol{r}_{cd}$ are the normal vectors to the two planes formed by the atoms a, b, c, and b, c, d. By convention, the angle ϕ_{abcd} takes on values in the half-closed interval [-180°, 180°) where 0° describes the *syn*-periplanar setting. Negative angles correspond to anti-clockwise

rotation while positive angles denote a clockwise rotation. To get the force vectors, we need the derivatives

$$\frac{\partial \phi_{abcd}}{\partial \boldsymbol{q}_a} = -\frac{r_{bc}}{u_{abc}^2}\boldsymbol{u}_{abc}\,, \tag{4.26}$$

$$\frac{\partial \phi_{abcd}}{\partial \boldsymbol{q}_d} = \frac{r_{bc}}{u_{bcd}^2}\boldsymbol{u}_{bcd}\,, \tag{4.27}$$

$$\frac{\partial \phi_{abcd}}{\partial \boldsymbol{q}_b} = \left(-\frac{\boldsymbol{r}_{ab}\cdot\boldsymbol{r}_{bc}}{r_{bc}^2} - 1\right)\frac{\partial \phi_{abcd}}{\partial \boldsymbol{q}_a} + \left(\frac{\boldsymbol{r}_{bc}\cdot\boldsymbol{r}_{cd}}{r_{bc}^2}\right)\frac{\partial \phi_{abcd}}{\partial \boldsymbol{q}_d}\,, \quad \text{and} \tag{4.28}$$

$$\frac{\partial \phi_{abcd}}{\partial \boldsymbol{q}_c} = \left(-\frac{\boldsymbol{r}_{bc}\cdot\boldsymbol{r}_{cd}}{r_{bc}^2} - 1\right)\frac{\partial \phi_{abcd}}{\partial \boldsymbol{q}_d} + \left(\frac{\boldsymbol{r}_{ab}\cdot\boldsymbol{r}_{bc}}{r_{bc}^2}\right)\frac{\partial \phi_{abcd}}{\partial \boldsymbol{q}_a}\,. \tag{4.29}$$

Figure 4.8 shows a harmonic angle potential in comparison with periodic potentials using different amplitudes and multiplicities. Often, dihedral angles are represented by a combination of more than one of these terms with varying values for n and δ.

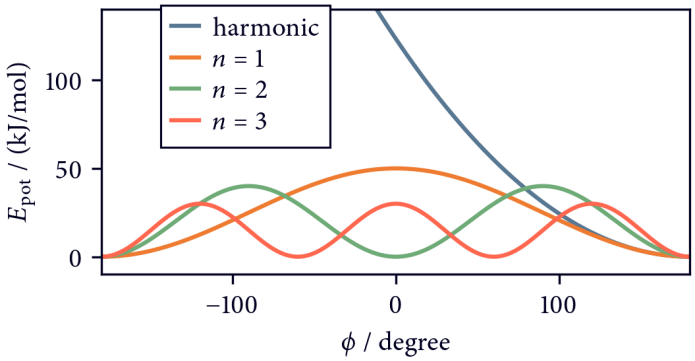


Figure 4.8 Harmonic vs. periodic dihedral potential Angle dependent harmonic potential energy (k = 25 kJ/(mol rad^2)) and periodic energies according to equation 4.23 for a C-C-C-C torsion angle with (n = 1/2/3, A = 25/20/15 kJ/mol, δ = 0/180/0°). Parameters chosen arbitrarily.

Figure 4.9 shows a combination of periodic potentials resembling the situation in butane, overlaying interactions for the C-C-C-C and C-C-C-H torsion angles. It should be noted, though, that for torsion angles of this type, usually also 1,4-non-bonded interactions have to be considered to get the full picture. Typically, these are scaled down by a force field specific factor.[93] Together with 1,5- and 1,6-interactions, the realistically expected potential for the torsion angle rotation in butane is in total quite different to what is shown in the figure. For bond and angle potentials, 1,2- and 1,3-non-bonded interactions are in contrast normally excluded because the interaction is sufficiently modelled by the used bonded terms. For some torsion angle potentials, 1,4-non-bonded contributions are excluded as well, for example when Ryckaert-Belleman potentials are used.[100] Arguably, dihedral angles are the most difficult to get right in a simulation. They depend sensitively on how the force field is constructed overall and can in general not be well transferred to other force fields.

butane torsion

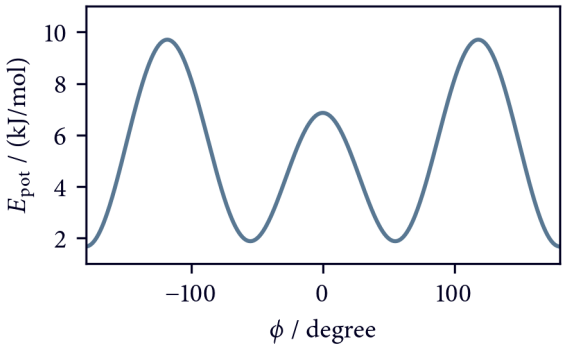


Figure 4.9 Dihedral potential for pseudo-butane Angle dependent potential energy according to equation 4.23 for the C-H_2C-CH_2-C torsion with contributions from one C-C-C-C (δ = 0/180/180°, n = 1/2/3, A = 0.75312/1.04600/0.8368 kJ/mol) and four C-C-C-H (δ = 0°, n = 3, A = 0.66944 kJ/mol) angles. Parameters taken from the AMBER99SB-ildn force field.[95]

The non-bonded interactions in MD simulations are conceptually treated similarly to bond potentials involving two atoms at a time, only that the number of considered non-bonded terms scales quadratically with the number of particles in the system, i.e. in general each particle interacts with any other particle (disregarding exclusions). Dispersive interactions are commonly modelled using a

Lennard-Jones

12,6-Lennard-Jones potential as a function of the distance between two particles

$$E_{\mathrm{LJ}} = 4\epsilon\left[\left(\frac{\sigma}{r_{ab}}\right)^{12} - \left(\frac{\sigma}{r_{ab}}\right)^{6}\right] \tag{4.30}$$

with force

$$F = 24\frac{\epsilon}{r_{ab}}\left[2\left(\frac{\sigma}{r_{ab}}\right)^{12} - \left(\frac{\sigma}{r_{ab}}\right)^{6}\right]. \tag{4.31}$$

The interaction is parametrised with the distance σ at which the interaction energy is zero and the potential depth ϵ. Alternatively, σ can be converted to the equilibrium distance $r_0 = \sqrt[6]{2}\sigma$, for which $E_{\mathrm{pot}}(r_0) = -\epsilon$. The quantity $r_{\mathrm{vdW}} = \sigma/2$ is known as the atom specific van der Waals-radius. Instead of σ and ϵ, it is also common to write out the potential with factors $A = 4\epsilon\sigma^{12}$ and $B = 4\epsilon\sigma^{6}$. The potential is constructed from a repulsive soft-sphere (Pauli) term that false of quickly ($1/r^{12}$) with increased distance and an attractive (van der Waals) term ($1/r^{6}$). Figure 4.10 shows this potential for two interacting argon atoms, including the attractive and repulsive parts, and the corresponding force expected on one of the atoms.

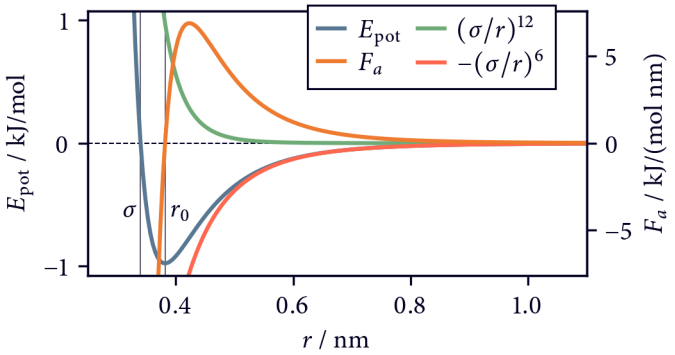


Figure 4.10 Lennard-Jones potential Distance dependent potential energy according to equation 4.30 for Ar-Ar with σ = 0.3401 nm and ϵ = 0.978638 kJ/mol. Parameters taken from the OPLS-AA/L force field.[101]

Lennard-Jones parameters are determined and tabulated for homo-dimers. The parameters for interactions of mixed dimers are usually obtained by averaging, although this may not always be a satisfying approach in which case specialised parameters can be used for individual pairs. A common averaging strategy uses for example arithmetic averages for σ and geometric averages for ϵ-values according to the Lorentz-Berthelot combination rules

combination rules

$$\sigma_{ab} = \frac{\sigma_{aa} + \sigma_{bb}}{2} \quad \text{and} \tag{4.32}$$

$$\epsilon_{ab} = \sqrt{\epsilon_{aa}\epsilon_{bb}}\,. \tag{4.33}$$

Besides the 12,6-Lennard-Jones type potential, there exist variations with different exponents. For metal ions, a 12,6,4-potential has promising properties.[102, 103] Other popular alternatives are the Buckingham potential, which differs in the repulsion being modelled with an exponential term,[104] or the Born-Meyer-Huggins potential with two separate attractive terms, which is for example used to model glasses.[105]

As a side note regarding the performance of force and energy calculations, the Lennard-Jones equations 4.30 and 4.31 can be implemented in a way to avoid two operations that are rather costly (if

they are repeated many times): instead of the actual distance between the atoms, the squared distance can be used, which saves a computation of the square root, and the σ^6 term can be reused saving a floating point division. Furthermore, it is possible to store ϵ in memory as already including a prefactor of 4 saving this repeated multiplication or to defer the multiplication to after all Lennard-Jones contributions have been calculated, but this may not make practical sense for every implementation.

implementation

```
rv = get_distance_vector(a, b)
rsq = distance_squared(rv)

sr2 = (s * s) / rsq
sr6 = sr2 * sr2 * sr2

energy = 4 * e * sr6 * (sr6 - 1)
force = 24 * e * sr6 * (2 * sr6 - 1) / rsq
fva = -force * rv
fvb = force * rv
```

The calculation of non-bonded interactions can become the bottleneck of a simulation when actually all possible pairs are considered because the number of pairs scales on the order of $\mathcal{O}(n^2)$ with the number of particles. To limit the number of pairs, it is common practice to truncate the Lennard-Jones potential at a certain cut-off distance and to calculate the interaction only for pairs for which $r_{ab} < r_c$. For pairs at $r_{ab} \geq r_c$, the contribution will be zero. The reasoning behind this, is that the interaction is essentially short-range in its nature and that contributions beyond a cut-off can be neglected without substantial error (for mor details see section 4.5). Moreover, the error can basically be made arbitrarily small by choosing a sufficiently large cut-off.

non-bonded cut-off

A disadvantage of this truncation, is that the potential and force profiles may become discontinuous, which can negatively affect energy conservation.[106] While this is not necessarily a problem within the normal accuracy limit of classic MD simulations, a number of modifiers (also called tapering functions) can be used to ensure that the respective functions are zero at the cut-off distance. It is often distinguished between *shifting* functions $s(r; r_c)$ that modify interactions over the whole range $[0, r_c]$ and *switching* functions $s(r; r_1, r_c)$ that only act within a range $[r_1, r_c]$. Shifting can, however, just be considered a special case of switching when $r_1 = 0$. The modifiers are in general either multiplicative or additive so that the changed potential or force function is either obtained as $f' = f \cdot s$ or $f' = f + s$. The choice of modification is a very sensitive one and is usually strongly coupled with the used force field framework. Tabulated parameters are often strictly valid only for specific cut-off values and certain potential modifications.

Figure 4.11 shows a few example modifications to the Lennard-Jones potential energy function. The probably most simple approach, is to just use the energy at the cut-off as a constant to be added to the potential. This will set the energy at the cut-off to zero but the derivate at the cut-off will be in general still non-zero. When this modifier is applied to a potential, forces remain unchanged so that the sampling during a simulation is overall not altered. This is for example the default behaviour in the simulation package GROMACS.[107]

modifiers

An example for a simple multiplicative shifting function that decays smoothly to zero at the cut-off but increases the energy values over a larger distance range more drastically, is

$$s(r; r_c) = \left[1 - \left(\frac{r}{r_c}\right)^p\right]^q \quad \text{with } p, q \in \mathbb{Z}\,. \tag{4.34}$$

For $p = 2$ and $q = 2$, this is a standard shift setting in CHARMM.[99] A more complex example for a modification as an additive series expansion is given by the Stoddard-Ford potential

$$E_{\mathrm{LJ}} = 4\epsilon \left\{ \left[\left(\frac{\sigma}{r_{ab}} \right)^{12} - \left(\frac{\sigma}{r_{ab}} \right)^{6} \right] + \left[6 \left(\frac{\sigma}{r_{\mathrm{c}}} \right)^{12} - 3 \left(\frac{\sigma}{r_{\mathrm{c}}} \right)^{6} \right] \left(\frac{r}{r_{\mathrm{c}}} \right)^{2} - 7 \left(\frac{\sigma}{r_{\mathrm{c}}} \right)^{12} + 4 \left(\frac{\sigma}{r_{\mathrm{c}}} \right)^{6} \right\} . \quad (4.35)$$

Finally, a pretty involved polynomial switching function implemented in GROMACS is

$$s(r; r_1, r_{\mathrm{c}}) = 1 - \frac{10(r - r_1)^3 (r_{\mathrm{c}} - r_1)^2 + 15(r - r_1)^4 (r_{\mathrm{c}} - r1) - 6(r - r_1)^5}{(r_{\mathrm{c}} - r_1)^5} . \quad (4.36)$$

Such a switching leads to a smooth decay while influencing a potential function in a relatively small distance range. A possible disadvantage of this, is that artificially high forces can be observed in the switching region.

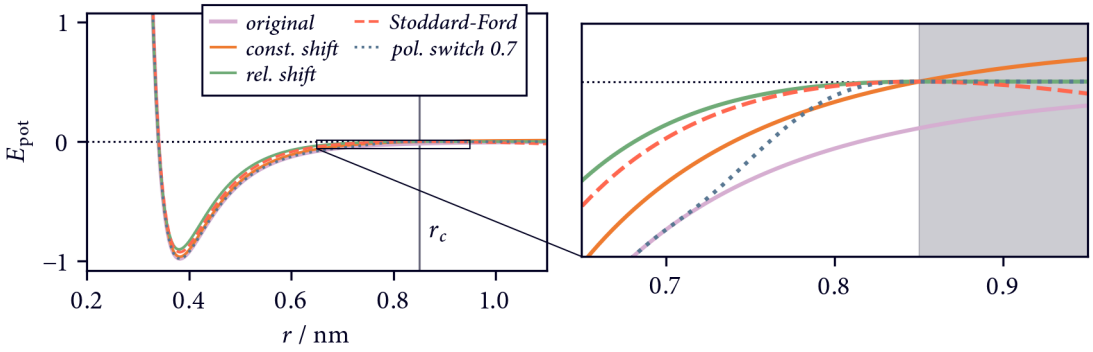


Figure 4.11 **Potential cut-off modifiers**

Original Lennard-Jones potential (pink, compare figure 4.10) and modified potentials that reach 0 at the cut-off distance $r_{\mathrm{c}} = 2.5\sigma$: *constant shift* (orange) just adds $-E_{\mathrm{LJ}}(r_{\mathrm{c}})$ to the potential, *relative shift* (green) multiplies the potential by equation 4.34 with $p = 4$ and $q = 2$, *polynomial switch* (blue) multiplies it by equation 4.36 where $r \geq r_1 = 0.7$, and *Stoddard-Ford* (red) applies a shift in accordance with equation 4.35. Note that for *rel. shift* and *pol. switch*, the energy for distances beyond the cut-off was explicitly set to zero $E_{\mathrm{LJ}}(r \geq r_{\mathrm{c}}) = 0$.

tail correction

A further approach to counter the effect of truncated potentials, is to correct energies by an approximate long-range term. This is known as tail correction. For each particle in the system, energy contributions from partners within the cut-off distance are calculated explicitly. Assuming a homogeneous distribution of particles beyond the cut-off, a contribution

$$E_{\mathrm{LJ}}^{\mathrm{tail}} = \frac{n\rho}{2} \int_{r_{\mathrm{c}}}^{\infty} 4\pi r^2 g(r) E_{\mathrm{pot}}(r) \mathrm{d}r \quad (4.37)$$

can be added, where n is the number of particles in the system, $\rho = n/V$ is the number density, and $g(r)$ is a radial distribution function that is for simplicity normally set to $g(r) = 1$. For the standard Lennard-Jones potential in equation 4.30, this integral gives[108]

$$E_{\mathrm{LJ}}^{\mathrm{tail}} = \frac{8}{3} \pi \rho \epsilon \sigma^3 \left[\frac{1}{3} \left(\frac{\sigma}{r_{\mathrm{c}}} \right)^{9} - \left(\frac{\sigma}{r_{\mathrm{c}}} \right)^{3} \right] . \quad (4.38)$$

Assuming a large enough cut-off so that the repulsive part of the interaction can be dropped and using a potential with a constant shift, the simpler expression[36]

$$E_{\mathrm{LJ}}^{\mathrm{tail}} = -\frac{8}{3} \pi n \rho \frac{\epsilon \sigma^6}{r_{\mathrm{c}}^3} \quad (4.39)$$

can be derived. For indeed perfectly homogeneous systems, for example composed of only one kind of particle, this is a very good approximation. If a system contains mixed particle types, the interaction parameters can be averaged. The approach remains valid to some extent also for inhomogeneous systems.[109] Again, the use of tail correction needs to be consistent with the employed force field and it depends on the origin of the interaction parameters if its application is necessary or advised.[89]

Electrostatic interactions are the second type of non-bonded interactions that are typically included in MD simulations. They can be modelled using a Coulomb potential between point charges z_a and z_b at distance r_{ab}:

$$E_{\text{Coulomb}} = \frac{f_{\text{C}}}{e_{\text{rel}}} \frac{z_a z_b}{r_{ab}} \quad \text{with } f_{\text{C}} = \frac{1}{4\pi e_0} . \tag{4.40}$$

The electric conversion factor f_{C} contains the dielectric permittivity in vacuum e_0 and amounts to $f_{\text{C}} = 138.935458\,\text{kJ nm/(mol } z_{\text{e}})$ in molecular units. Here, z_{e} denotes the elementary charge. The atomic charges z_a and z_b can therefore be conveniently given in multiples of z_{e}. The relative permittivity e_{rel} is set to 1 if no implicit medium is assumed in which the interaction takes place but can take on a medium specific value if needed. This potential causes a force

Coulomb

$$F = \frac{f_{\text{C}}}{e_{\text{rel}}} \frac{z_a z_b}{r_{ab}^2} . \tag{4.41}$$

Figure 4.12 illustrates the Coulomb potential for an interacting charge pair in vacuum and in implicit water.

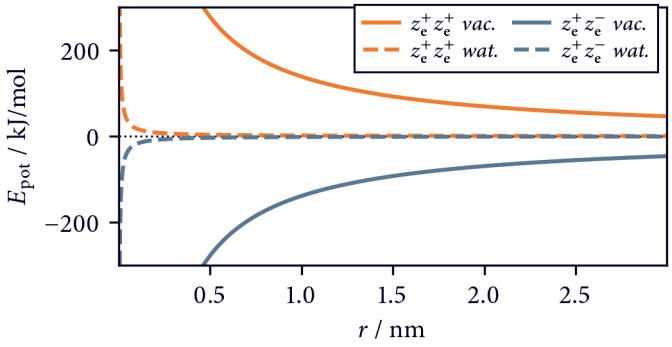


Figure 4.12 Coulomb potential Distance dependent potential energy according to equation 4.40 for two interacting elementary charges with equal (orange) and opposite (blue) sign in vacuum ($e_{\text{rel}} = 1$) and implicit water ($e_{\text{rel}} = 80$).

In principle, the same considerations as for the dispersive interaction terms apply in regard of a possible truncation of the electrostatic potential. Similar shifting and switching schemes can be applied. The problem with this, is that while a 12,6-Lennard-Jones potential falls of relatively quickly with an increased distance between the interacting particles, Coulombic interactions decay much more slowly ($1/r$). This means that the long-range character of the potential can not be neglected by applying a finite cut-off without accepting a large error. Approaches like a tail correction as described above with equation 4.37 will not work in this case. What does work, however, is something conceptually quite similar: By assuming a homogeneous medium with a given permittivity e_{rf} beyond the cut-off distance, it is possible to define a physically justified, shifted Coulomb potential

reaction-field

$$E_{\text{Coulomb}} = f_{\text{C}} z_a z_b \left(\frac{1}{r_{ab}} + \frac{e_{\text{rf}} - e_{\text{rel}}}{e_{\text{rel}} + 2e_{\text{rf}}} \frac{r_{ab}^2}{r_c^3} - \frac{3e_{\text{rf}}}{e_{\text{rel}} + 2e_{\text{rf}}} \frac{1}{r_c} \right) . \tag{4.42}$$

This is the so called reaction-field approach in its basic formulation for which several improvements exist.[110] Figure 4.2 shows an example for a Coulomb potential plus reaction-field.

Besides the continuum approach, most widely used methods to treat electrostatics work via periodic lattice sums. Without going into the complex details of it, the basic idea is to divide the Coulomb

lattice sums

interaction into a near and a far part. The near part can be evaluated directly, possibly using cut-offs and shifting schemes. The long range part, on the other hand, is solved by various different schemes in reciprocal space where the infinite sum of point charges on the lattice converges more quickly. Such methods are popular because of their accuracy and general applicability as long as a system is in total charge neutral and in fact periodic. Compared to reaction field approaches that basically scale linearly with the number of atoms in a system, typical lattice sum schemes are more expensive, scaling on the order of $\mathcal{O}(n \log n)$.[111] Recently, the u-series has been proposed as a more efficient scheme.[112] Another potential drawback of lattice sums is that the pairwise interaction picture is essentially lost, that is pairwise energy contributions can not be easily recovered.[110]

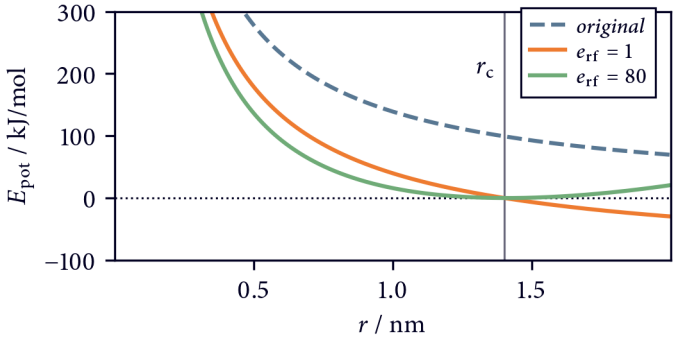


Figure 4.13 Coulomb potential with reaction-field Distance dependent potential energy for two interacting elementary charges with equal sign using $e_{rel} = 1$ and a reaction-field with $e_{rf} = 1$ (orange) or $e_{rf} = 80$ (green) according to equation 4.42 in comparison to the unmodified potential.

The set of different interactions as they are described here, have proven themselves usable in countless simulations. A rather strong limitation of this classic combination (mostly of the way how electrostatic interactions are treated) is, however, that it usually falls short in providing accurate descriptions of ions. To recall, atomic partial charges are usually treated as scaled point charges on top of atomic positions, such that they can only reflect one very specific (average) situation. Without modification, this model is not very well suited to reproduce consequences for energetics and geometry arising from changes in the atomic charges, their relative strength and their actual distribution. A more realistic, flexible model can potentially improve the description of ionic species, first of all metal ions. But even beyond that, there are many aspects concerning biological systems where polarisability can play a major role.[113] Examples are π-π interactions of aromatic fragments, or cysteine anions, because for delocalised or diffuse charges, the point charge model can be insufficient.[114] It has been also shown that the induction of peptide bond dipoles can be important for helix formation.[115] Improvements are likely also for intrinsically disordered proteins, where flexible charges can change in different states of (un)folding.[116] Further, it is considered important for water dynamics, and in general for phase transitions, e.g. from aqueous solution into a protein cavity or cell membrane. As a slightly more exotic example, it helps to describe the binding and diffusion of diatomic gas molecules like O_2 in a protein. [117]

Improved charge models

In the next section (section 4.3), a brief overview on polarisable force fields, aiming on countering this shortcoming, will be given. For the sake of completeness, I would like to name a few alternative approaches first, though. Within the limit of fixed atomic point charges, the only possible variation would be to scale the charges. For calcium this has been successfully done and helps with reproducing the energetics of hydration and protein-binding,[118, 119] but polarisation is still only treated in a mean-field way. Otherwise, a tuning of ion properties in a simulation can be achieved via Lennard-Jones interactions.[120] It has been shown, however, that a parametrisation of this potential for (higher charged) ions is very difficult and that it is essentially impossible to optimize hydration free energies, coordination numbers, and ion-oxygen distances at the same time with a single set of parameters. The use of an extended 12,6,4-Lennard-Jones potential with an additional $1/r^4$ term accounting for

ion-induced dipole interactions constitutes a considerable improvement here.[102, 103] This modified potential was already successfully used to model metalloproteins,[121, 122] and is being actively improved.[123] Further possibilities in particular aiming on geometric criteria, are to use a number of dummy atoms without mass but with charges distributed around a central atom to mimic the coordinative sites,[124] or to use covalent potentials, moving away from a solely non-bonded treatment of ions.[125] Both these latter approaches tend to make strong assumption, though, about the expected behaviour of the considered ions.

4.3 Polarisable force fields

Considerations to include polarisation effects into MD simulations date far back to the early days in the development of this method.[46] Only the formal difficulties and in particular the computational cost associated with this, did defer its widespread use in standard simulation procedures to more recent times. As described in the last section, the fully classical MD framework does usually use a fixed point charge model to account for electrostatic interactions. Despite the general success of this approach and the evolution of well refined respective force fields, it is insufficient for many also biologically relevant systems. It became clear that the next major step in advancing protein force field accuracy requires a different representation of the molecular energy surface.[126]

Three dominating approaches exist to account for polarisability of atoms in MD simulations, though many flavours and alternatives are available. The first is called the fluctuating charge model.[127] It is conceptually the most simple and allows to alter atomic partial charges while keeping the general fixed point charge picture. Polarisation is, however, limited to the exchange of charge along covalent bonds unless additional dummy sites are introduced. The second approach uses additional particles, called Drude oscillators, with non-zero charge attached to a central atom via harmonic spring potentials. In this model, actual polarisation can be represented by the motion of these charges, where the force constant of the binding potential corresponds inversely to polarisability.[128] And the third approach uses multipoles that describe the distribution of charges around atoms and models electrostatic contributions as interactions between and polarisation of these multipoles.[129]

methods

Both the Drude oscillator and the mulitpole models have been intensively improved since 2013 in terms of the Drude,[130, 131] and the AMOEBA force fields.[132, 133] It is not clear yet if either one of the approaches could turn out as general preferable. Theoretically, the two descriptions should be basically equivalent, though.[134] We want to focus here on the AMOEBA force field as the arguably more intuitive way to account for polarisation. The use of multipoles offers also one important conceptual advantage over moving charge particles, which is *permanent electrostatics*. This entails everything that improves the fixed point charge model without actually including a form of polarisation, i.e. a perturbation of charges. For one thing, a multipole defines an anisotropic charge distribution around an atom. To explain interactions in terms of aromatic systems, lone pairs, or σ-holes, this can already capture a lot of the necessary information. It also covers the effect of *charge penetration* because charge distributions that posses an effective radius can interact differently (less strongly) if they are close to each other and partly overlapping. Polarisation is then incorporated on top, achieving self-consistency by using an iterative scheme. The inclusion of explicit polarisation necessitates the substitution of the commonly used 12,6-Lennard-Jones potential against a 14,7-potential to avoid redundancies in the description. The inclusion of further effects like charge transfer already led to the spin-off of the independent AMOEBA+ force field.[135]

The performance of polarisable force fields has been evaluated before the background of the question: '*do they really offer a more accurate description of the modelled systems?*'. The answer to that question seems to be '*yes*' in general, although the details are still inconclusive. The matter is complicated by the

fact that the results produced by different force fields can vary considerably even among mature non-polarisable force fields.[136, 137] A long lasting issue is, among other things, the discrepancy between descriptions for folded and intrinsically disordered proteins.[138, 139] A stabilising effect by a certain force field may be desired in one case and completely inapt in another. It has been shown in the context of folding simulations, that helix stability is generally well captured by the AMBER99SB force field (a modification of which was mainly used in our simulations), at least in contrast to CHARMM36a2 and OPLS-AA, which over- or under-estimate it.[140] For polarisable force fields, there seems to be a general tendency towards higher protein flexibilities.[126] They have been found to accurately model disordered states but can have problems with stabilities of native states, which is usually reversed for non-polarisable force fields.[141] In the reference papers for the Drude2013 and AMOEBA2013 force fields,[130, 132] protein flexibilities were, however, supposedly on an acceptable level. On the one hand, a more flexibel description is appreciated in some cases as the classical AMBER99SB force field was rendered responsible for an over-stabilisation of a salt-bridge in a PDZ-domain compared to a polarisable Drude model.[142] But on the other hand, using the Drude2013 force field, a non-negligible destabilisation of the folded native states of a β-hairpin peptide, a villin headpiece variant, and a WW domain was be observed. In contrast, the non-polarisable force field AMBER14SB was found to over-stabilise folded conformations, while CHARMM36m and GROMOS54A7 were identified as the most appropriate choice in these cases.[143] Interestingly, exactly the opposite is noticed for the C-terminal β-hairpin of GB1.[144] In any case, the stability issues found addressing in the most recent version of the Drude force field (Drude2019).[131] The newest AMOEBA force field concentrates among other things on respective optimisations for nucleic acids.[133] In a current benchmark on G-quadruplexes, however, the Drude force field turned out to be preferable over the AMOEBA approach in terms of maintaining the stability of the simulated structure.[145] Although detailed comparative assessments of conformational stabilities are otherwise scarce, AMOEBA has been successfully used in simulations of metalloproteins[146, 147] and enzymes[148, 149] without reported stability issues. AMEOBA showed also a very good performance in small molecule conformational searches.[150]

stability

Another important aspect of force field accuracy concerns the modelling of molecular kinetics. For polarisable force fields the number of available studies in this direction is limited, possibly due to the higher computational cost of these simulations. Classic force fields often overestimate the conformational relaxation rates of proteins. Polarisation can apparently help to slow these down but for a truely accurate description, transition state energies would need to be actively integrated into the force field development process.[151]

kinetics

4.4 Periodic boundary conditions

Molecular simulations are limited to a rather small amount of particles. Large scale simulations—maybe dealing with millions of atoms—require huge computational resources and even they can capture only a vanishingly small fraction of the atoms partaking in a wet lab experiment. Typically, a simulated system is confined in a simulation box with a narrow volume unless it is specifically desired to simulate in vacuum where particles can drift away from each other freely. Without further provisions, a simulation in such a box would be heavily influenced by artificial interactions of the system with the box boundaries and observations drawn from it would be hardly relatable to the real system that is being modelled. Unwanted modelling artifacts arising from this scale discrepancy are often summarised as finite size and edge effects[152]. The smaller the box, the more drastic would be the distortion of the model. Yet, the computation of bulk properties by using large enough boxes is technically unfeasible.

A solution to the problem are periodic boundary conditions (PBCs). They are used to approximate the situation in the real system by an infinite repetition of the simulation box that can be understood as a unit cell. Instead of hitting a wall at the simulation borders, particles are allowed to cross them and enter a periodic image of the simulation box. Still only one realisation of each particle is actually physically represented, keeping the total number of modelled particles small. The periodically repeated unit cells are exact replications of the primary simulation box and form a space filling lattice. This

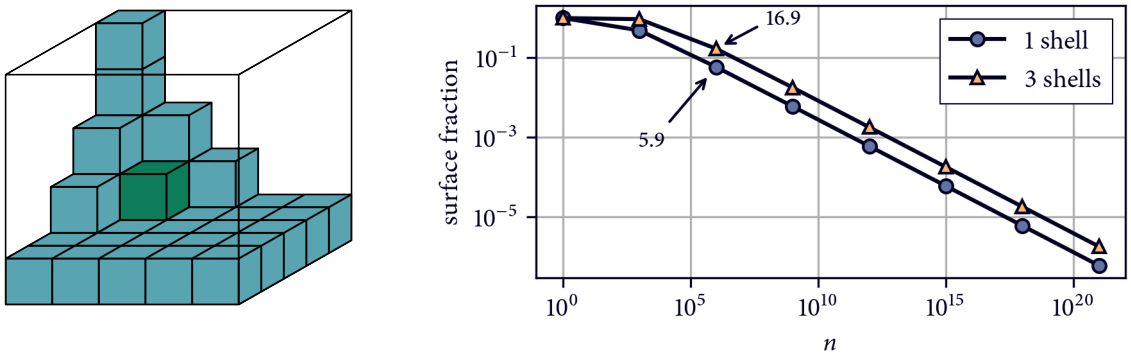


Figure 4.14 Surface fraction in cubic boxes of different particle numbers
Assuming cubic particles, the box of 125 particles on the left has only 27 particles (22 %) not directly in contact with the boxes surfaces. All of them are found within the first three shells from the surface. For typical particle numbers in molecular simulations of up to 10^6, still about 6 % are in contact with the surface and 17 % are among the first 3 shells.[108]

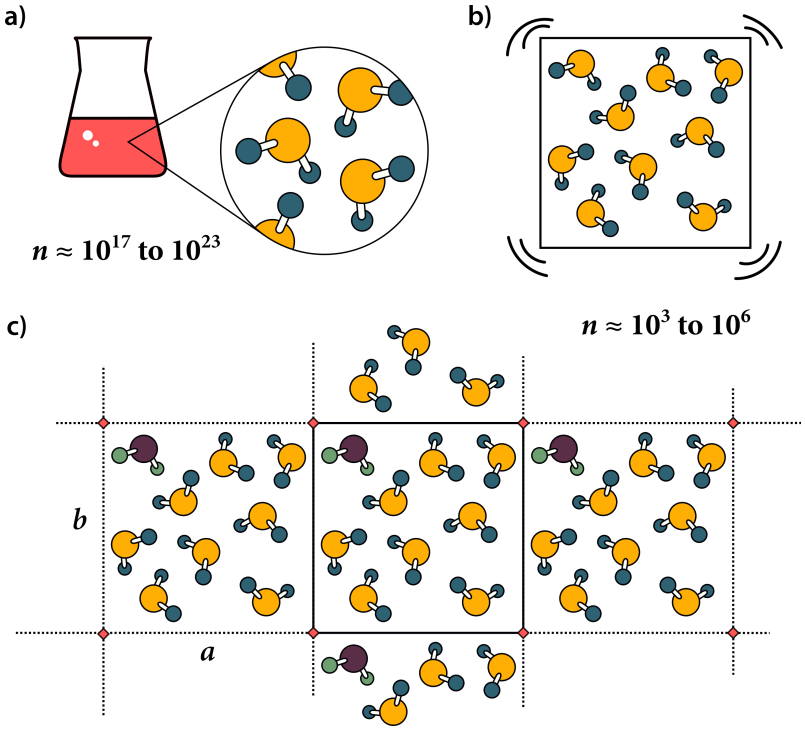


Figure 4.15 Periodic boundary conditions a) In practical experiments, the amounts of handled substance are typically in the molar to micromolar range, which is orders of magnitude more than is realistically achievable in molecular simulations. These are in most cases limited to a few thousands or millions of atoms. **b)** Dealing with small numbers of simulated particles in narrow simulation volumes can have undesired effects on the observed properties of a system. **c)** Under periodic boundary conditions, simulation artifacts arising from limited box volumes can be avoided by treating the system on an infinite lattice of equivalent unit cells as an approximation to the real system. This disregards of course the fact that there are no infinite periodic real systems in nature.

lattice $\boldsymbol{L}$ has the property that each point on the lattice is part of an infinite set of equivalent points that can be transformed into one another by integral translation along lattice vectors (also called box vectors), i.e.[152]

$$\boldsymbol{L} = \{k\boldsymbol{a} + l\boldsymbol{b}\} \qquad \text{with } k, l \in \mathbb{Z} \tag{4.43}$$

in 2D like shown in figure 4.15c with square boxes spanned by the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. The lattice points in figure 4.15c are marked with red diamond shapes. The location of each simulation box is defined by a single lattice point. It should be noted that the shown lattice is not unique and that any other set of lattice points could have been chosen to define it on the basis of the particle coordinates. Furthermore, while a particular box layout implies a lattice, a lattice does not impose a certain box layout. Considering only the lattice, the choice of a particular box is arbitrary.

The treatment of periodic boundaries in MD touches two aspects: the first question is, how and when exactly the coordinates of individual atoms are modified while they are crossing a box border. This step is equivalent to conditionally *'putting (or folding) particles back into the simulation box'* and is entirely optional. The second question is, how to consider interactions of particles with each other across a periodic boundary, which especially concerns the calculation of pairwise distances. This can be formalised by a *minimum image convention* that determines the actual interaction partners for each particle, i.e. the set of closest partners found over all periodic images. The calculation of the minimum image may depend on whether all particles are ensured to be in the primary simulation box in the first place.

4.4.1 Position folding over periodic boundaries

As an illustration for the practical application of PBCs, let's consider the 1-dimensional case of two particles with positional coordinates $\boldsymbol{q} = (q_1, q_2)$ simulated in a box that is given by a box vector $\boldsymbol{a}$ with length a. We put the lattice point of the primary simulation box at the origin of the reference frame for the particle's coordinates. This means that in this view, the center of the box is located at $0.5a$, while the lower and upper bound of the box coincide with 0 and a, respectively. In principle, it is possible to place the origin of the particle coordinates in the center of the box or at any other location as well, in which case shifted bounds would need to be considered. Figure 4.16a shows this system at a given configuration $\boldsymbol{q}(t)$ in time, from which a single simulation time step shifts the system to a configuration $\boldsymbol{q}(t+\tau)$ in figure 4.16b. During this propagation, one of the particles moved outside of the simulation box in positive direction. The particle can be translated back into the box to a periodically equivalent location by subtracting the box vector from its current position $q_i' = q_i - a$.

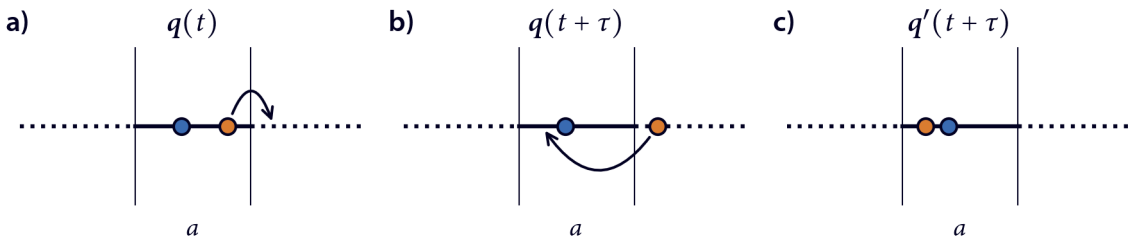


Figure 4.16 Periodic boundary example in 1D
a) Two particles in a 1D simulation box at a certain point in time. **b)** After the propagation of the current particle positions within a discrete time interval, the orange particle has moved out of the simulation box, **c)** The orange particle has been placed back into the simulation box.

Programmatically, this operation can be performed after every simulation time step (or a different interval) by checking for each particle if it has moved beyond the box borders and putting it back if it did:

```
a = 2  # 1D box vector with bounds 0 and 2
for qi in q:
  if qi < 0:
    qi += a
  elif qi >= a:
    qi -= a
```

In one line, the above conditional statements can be substituted with the more general approach of computing the modulo $q_i \bmod a$, which gives the PBC-corrected position as the remainder after the division of the actual position by the length of the box.

```
for qi in q:
  qi = qi % a
```

This variation will also successfully re-locate particles from periodic images that are arbitrarily far away while the first method implies that all particles have been in the primary simulation box at time t and have only traveled to a directly adjacent box at time $t + \tau$.

When using the modulo, a bit of care needs to be taken because its definition is ambiguous and varies among programming languages, which is mainly due to a different handling of integer division. In principle, the modulo operator is equivalent to something like:

Modulo sidenote

```
def modulo(dividend, divisor):
  return dividend - (divisor * (dividend // divisor))
```

where `//` represents a division that returns an integer. In Python for instance, integer division corresponds to a rounding of the division result towards $-\infty$ (flooring). In C, however, the result is always rounded towards 0 (truncated). How this difference manifests itself in the modulo for negative dividends, can be seen in figure 4.17. The Python realisation is the one that we want for the correction of positions over periodic boundaries.

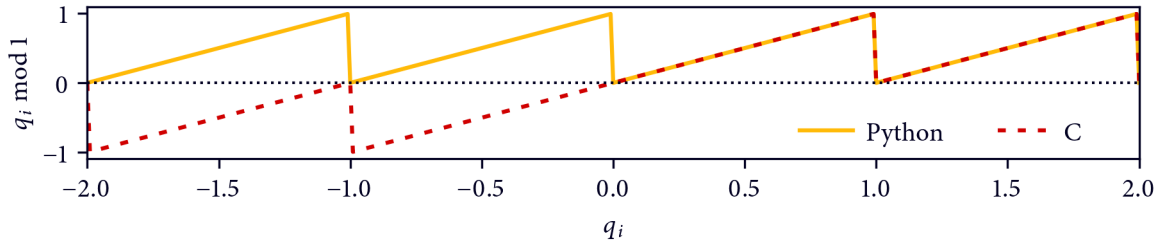


Figure 4.17 Difference in the modulo operator for Python and C
Due to the fact how integer division is implemented in different programming languages, the result of the modulo operation for negative dividends can be inconsitent. Note that we do not need to consider the case of negative divisors in this context because box vectors can only take on positive values.

As an alternative to relying on the modulo operator of a certain programming language, it is of course possible to explicitly calculate a remainder using flooring.

```
for qi in q:
  qi -= floor(qi / a) * a
```

The shown 1D scheme can be scaled to infinite dimensional systems if the box vectors are all orthogonal, that is for most practical applications to orthorhombic unit cells in 3D. In fact, it is sufficient to know the upper bounds of the box in each dimension, which can be given as an array of length 3 if the

origin of the simulation box is located at $\boldsymbol{q}_o = (0,0,0)$. Each particle position is given by a vector $\boldsymbol{q}_i = (q_{i,x}, \boldsymbol{q}_{i,y}, q_{i,z})$ and the box vectors $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are in this case aligned with the positional coordinates x, y, and z, e.g. $\boldsymbol{a} = (1,0,0)$, $\boldsymbol{b} = (0,1,0)$, $\boldsymbol{c} = (0,0,1)$ for a cubic box with side length 1. For the correction of the periodic postions, we need to consider each dimension separately:

```
# Upper box bounds
bounds = [a, b, c]
for qi in q:
  for d in range(3):
    qi[d] = qi[d] % bounds[d]
```

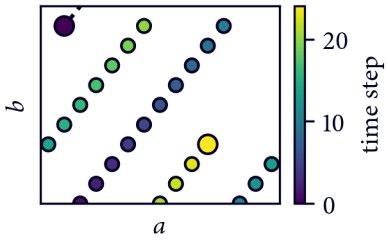


Figure 4.18 Periodic boundary example in 2D A particle moving with constant velocity through a periodically bounded box with orthogonal box vectors while crossing the box boundaries multiple times. The particle positions are corrected after each step using the scheme above. The current time step is indicated for each postion by colour.

Orthorhombic unit cells are conceptionally easy to treat but from a modellers perspective, other box layouts may be more beneficial in terms of simulation efficiency. An example for a common simulation setup, are molecular systems in which the simulation box contains a single macromolecule (a protein) surrounded by solvent and maybe a few counter ions. A popular alternative space-filling box geometry for the simulation of such a system is the dodecahedron.[107] Compared to a cubic box with the same periodic distance between unit cells, a dodecahedral box has a smaller volume by a factor of 0.71.[6] This means that while maintaining the same distance between the solvated macromolecule (in the center of the box) and its nearest periodic image, the volume and therefore the number of needed solvent molecules to fill the box, can be reduced by switching from a cube to a dodecahedron. Keeping the periodic distance constant (and in fact large enough[7]) may be important to avoid simulation artifacts due to an interaction of the macromolecule with itself beyond the periodic boundaries.

Dodecahedral and other box shapes, like the hexagonal prism or the truncated octahedron, pose some difficulties for the implementation of PBCs. Fortunately, it can be shown that these complex box layouts (as well as the friendly orthorhombic unit cell) can be transformed into the general case of a triclinic box of which they are essentially a special case and which is comparably painless to deal with.[156] A triclinic unit cell distinguishes itself by the absence of any symmetry elements, i.e. by the fact that there are neither restrictions on the lengths of the box vectors, nor on their relative orientation.

triclinic boxes

So let's now have a look at a practical PBC implementation for the general case of a triclinic box. For this purpose, it is reasonable to express the box layout in terms of a matrix $\boldsymbol{A}$ in which the box

[6] Refer to the GROMACS documentation for details: `manual.gromacs.org/documentation/current/onlinehelp/gmx-editconf.html`

[7] While there is no definite advice for a specific box size, a commonly applied heuristic is to keep the distance between the solute and the box borders at about at least 1 nm. This is based on the rational that this distance is a common setting for the cut-off used on short range non-bonded interaction terms, which in turn is essentially a requirement of the used force-field.[153] Another fix-point is the correlation distance in liquid water of about 0.8 nm.[154] The effective box size should also take into account that a macromolecule can undergo conformational changes during a simulation, which can increase its circumscribed radius.[152] For a recent discussion on the topic see [155].

vectors $\boldsymbol{a}_d$ are collected as column vectors with respect to the positional reference frame, that is in 3D with $\boldsymbol{a}_1 = \boldsymbol{a}$, $\boldsymbol{a}_2 = \boldsymbol{b}$, and $\boldsymbol{a}_3 = \boldsymbol{c}$:

$$\boldsymbol{A} = \begin{pmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{pmatrix}, \tag{4.44}$$

which would be for the again specialised case of a cubic box with box length 1:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4.45}$$

The position $\boldsymbol{q}_i$ of a particle can be expressed in the basis of these box vectors

$$\boldsymbol{q}_i = \boldsymbol{A}\boldsymbol{q}_i^* \tag{4.46}$$

where $\boldsymbol{q}_i^*$ denotes the position in fractional coordinates with respect to the box. The above matrix-vector multiplication is equivalent to the following summation:

$$\boldsymbol{q}_i = \sum_d \boldsymbol{a}_d \boldsymbol{q}_i^* . \tag{4.47}$$

In reverse, we can now obtain a particle position in fractional box coordinates using the inverse of the box vector matrix $\boldsymbol{B} = \boldsymbol{A}^{-1}$ as

$$\boldsymbol{q}_i^* = \boldsymbol{B}\boldsymbol{q}_i . \tag{4.48}$$

If our initial intent was to ensure that all particles are placed in the primary simulation box, this would mean that we want to manipulate these fractional coordinates so that they are restricted to the interval $0 \leq \boldsymbol{q}_{i,d}^* < 1$ in each dimension d. This can be done again by either taking the modulo $\boldsymbol{q}_{i,d}^* \bmod 1$ or by directly substracting the nearest lower integer, e.g. $\boldsymbol{q}_{i,d}^* = \boldsymbol{q}_{i,d}^* - \mathrm{floor}(\boldsymbol{q}_{\mathrm{i,d}}^*)$. The corrected positions in the original coordinates can be re-obtained afterwards according to equation 4.46.

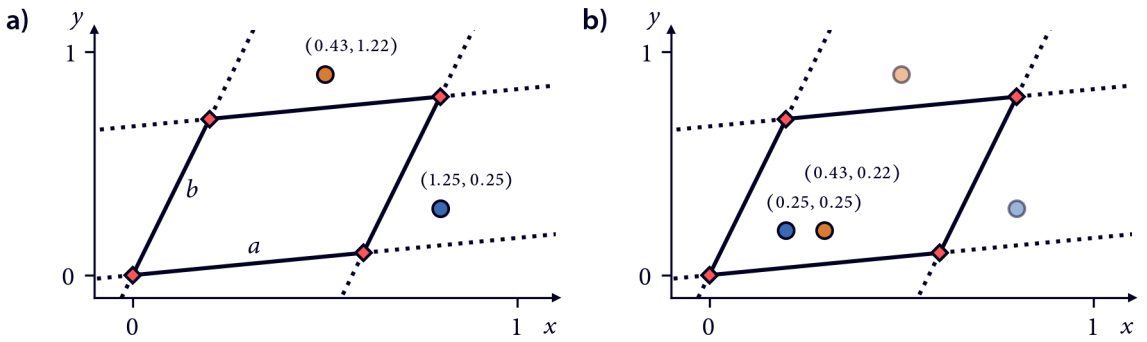


Figure 4.19 Periodic boundary example in 2D (triclinic box)
a) Two particles with positions outside of the primary simulation cell that are given in fractional coordinates with respect to the box vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ obtained according to equation 4.48. **b)** The two particles have been shifted back to their periodic image in the simulation box.

As mentioned in the beginning, the described procedure of putting particles back into the simulation box when they leave it, can be substantially modified or left out entirely. A variation of this would be for example to re-position entire molecules as soon as they cross a boundary (or their center of mass does) instead of individual atoms. This implies an additional complexity for the implementation and an awareness of the topological nature of the molecules but may be advantageous if certain operations or analyses are complicated for molecules broken over periodic boundaries. It should be noted, that

periodic boundary manipulations like this can also be performed in post-processing steps on the produced trajectory and do not necessarily need to be included into a simulation itself. Omitting the PBC-correction and allowing the particles to populate multiple periodic images of the simulation box can be helpful in certain situations where especially the unbroken path of individual particles is of interest, e.g. for the investigation of diffusion and related properties.

4.4.2 Minimum image convention

Let us now come to the second aspect of periodic boundaries in MD, which is the treatment of inter-particle interactions across box borders. In contrast to the first step discussed up to here, a correction in this regard is mandatory if correct simulation results should be obtained. The interaction terms commonly used in molecular simulations that represent for instance bonds, angles, and dispersion, are in general many-body terms and depend on the spatial coordinates of more than one atom. For each atom, it has to be ensured that the interactions it is involved in are calculated with respect to sensible positions of its interaction partners, not only considering the primary simulation box.

Let's illustrate this again with a simple example first. Figure 4.20 shows a system of n particles in a square simulation box in 2D. We want to evaluate an interaction that is based on the inter-particle distances from one particle $\boldsymbol{q}_i$ to all the other particles $\boldsymbol{q}_{j\neq i}$. For some of the interaction partners in the primary simulation box, there exists a periodic equivalent at a closer distance, which is the one that should be used to evaluate the interaction. Effectively, we want to find the set of closest particles in all periodic images with respect to $\boldsymbol{q}_i$, which is described as the minimum periodic image centred on $\boldsymbol{q}_i$.

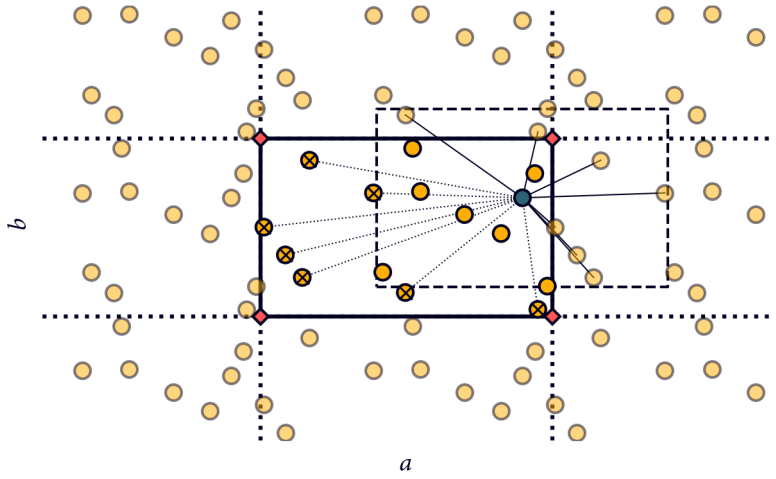


Figure 4.20 Minimum image convention in 2D
Particles in the primary simulation box are colored in solid yellow. Periodic images of these particles in adjacent simulation boxes are drawn in transparent yellow. The green particle interacts with all other particles in the central box in a distance dependent manner. For those particles that are crossed out, there is a periodic expression of this particle that is actually closer to the green particle. The calculated dotted distances need to be PBC-corrected and the solid, shorter distances to the periodic equivalents should be used instead to evaluate the interaction. The dashed box centred around the green particle contours its minimum image.

It should be noted, that the treatment of interactions under the minimum image convention is only strictly sensible for bonded and short-range non-bonded terms. For long-ranged non-bonded terms, a consideration of the closest interaction partner alone may actually be a rough truncation and contributions from interactions over the (infinite) lattice may not be negligible (see also sections 4.2 and 4.5).

In analogy to the programmatic solution for a correction of particle positions over the periodic boundaries, distances can be corrected by checking for each dimension if the respective component of the distance vector exceeds the bounds of the minimum image, i.e. half the length of the corresponding box vector, in either direction from the particle in its center. If it does, the component needs to be shifted by the respective component of the box vector. For one particular distance $\boldsymbol{d}_{ij}$ between two particles $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ in 2D, this could be realised like this:

```
# Upper box bounds
bounds = [a, b]
for d in range(2):
  dij_d = qi[d] - qj[d]
  if dij_d <= -0.5 * bounds[d]:
    qij_d += bounds[d]
  elif dij_d > 0.5 * bounds[d]:
    dij_d -= bounds[d]
```

Alternatively, distances can again be corrected by performing an operation, which can account for periodicity over arbitrary unit cells similar to taking the modulo when correcting positions. For a short (maybe not quite up-to-date) comparison of different realisations thereof in the context of minimum image computation for orthorhombic boxes including the effect of compilers and computer architectures see [157].

In the more interesting general case of triclinic boxes, the minimum image convention follows a scheme that is quite similar to what has been presented before for the correction of particle positions as well: assuming we have the matrix of box vectors $\boldsymbol{A}$ and its inverse $\boldsymbol{B}$, we would use $\boldsymbol{B}$ to find the particle positions in fractional coordinates of the box basis according to equation 4.48. Then we can compute a distance vector $\boldsymbol{d}^*_{ij}$ in fractional coordinates, which can be subsequently shifted and transformed back into the original positional coordinates.[91] Note that we use a rounding function (which refers to a nearest integer operation) instead of flooring to shift the distances in this case.

$$\boldsymbol{d}^*_{ij} = \boldsymbol{q}^*_i - \boldsymbol{q}^*_j \tag{4.49}$$

$$\boldsymbol{d}^*_{ij} = \boldsymbol{d}^*_{ij} - \mathrm{round}(\boldsymbol{d}^*_{ij}) \tag{4.50}$$

$$\boldsymbol{d}_{ij} = \boldsymbol{A}\boldsymbol{d}^*_{ij} \tag{4.51}$$

To summarize this section, let us finally have a look at a complete simulation example using the presented PBC-correction and minimum image convention for triclinic boxes, although for an easier plotting of the results, a cubic box in two dimensions should suffice here. The example system consists of nine argon atoms interacting purely via harmonic quasi-hard sphere potentials (no attraction, only repulsion) with radii $\sigma = 0.340\,100$ nm (taken as the Lennard-Jones parameter from the OPLSAA/L force field) and a very high force constant. Initial particles where chosen accordingly on a grid. To introduce some asymmetry into the interactions, the parameter with respect to one of the atoms has been scaled to 0.8σ. Snapshots from the simulation showing the current atom positions are collected to be inspected in figure 4.21.

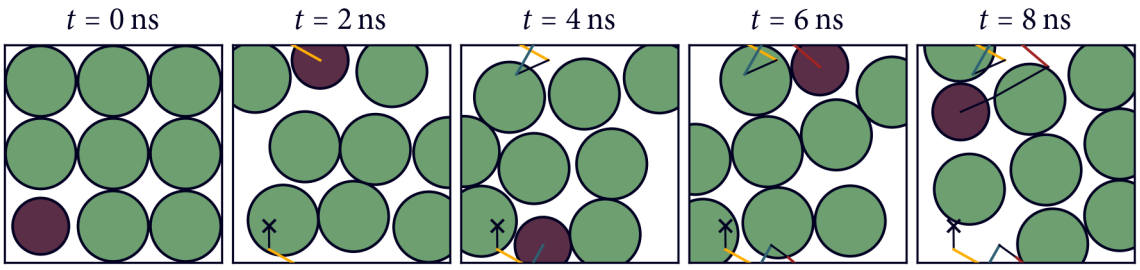


Figure 4.21 Hard sphere argon simulation in 2D
The atom coloured in purple interacts with the green atoms using different repulsive parameters than the green particles with each other. A black x marks the starting position of the purple atom in the pictures of the subsequent time steps. The path taken by the purple atom in 2 ns intervals is marked by a black line while every time the atom crosses a periodic boundary, the path segment is coloured differently. The simulation was carried out using an Euler-Maruyama integration scheme with a timestep of 1 fs, and a friction coefficient of 1/fs at 150 K.

4.5 Neighbour lists

In a molecular system of n particles, non-bonded interactions need to be in principle evaluated for $(n^2 - n)/2$ pairs (Gauss summation) avoiding self-interaction and double counting—but not considering that some of these pairs may be irrelevant because the corresponding particles are connected already by bonded interaction terms. In section 4.2, it was already mentioned that certain interactions of limited range can be cut off after a radius r_c so that their contribution is neglected for pairs at larger distances. Basic interaction modifiers have been discussed dealing with the introduced discontinuity. As a rule of thumb, the error that is made by employing a cut-off and neglecting the long-ranged part of an interaction is acceptable in a n_d-dimensional system for potentials that decrease on the order of $\mathcal{O}(1/r^{n_d})$ or faster when $r \to \infty$. In this case, the error can be treated as a finite integral for homogeneous systems, which can be treated in terms of a tail correction.[108, 109] The typical Lennard-Jones potential fulfils this requirement while electrostatic Coulomb interactions do not. Note that also the minimum image convention (compare section 4.4) can be viewed as a form of cut-off approach in itself, since interactions are only evaluated with respect to the closest interaction partner in all periodic images—not with respect to all periodic partners.

non-bonded cut-off

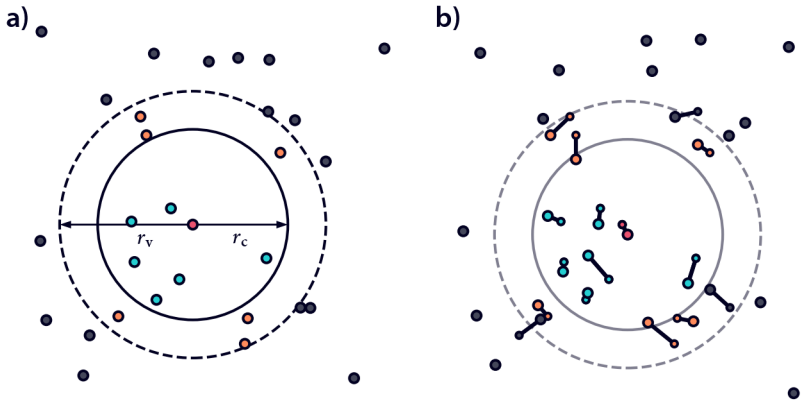


Figure 4.22 Verlet list a) By using a cut-off r_c, the number of considered non-bonded interaction partners for the central red particle can be reduced to the blue particles. To avoid a recomputation of the neighbours at each step, additionally the orange particles within a buffer zone are considered as potentially relevant. **b)** Two orange particles entered the cut-off region. One grey particle almost entered it as well, which should be avoided.

The truncation of pairwise potentials alone, can, however, not lead in general to a significant increase in computational performance. This is because, although the actual evaluation of a force is skipped when a pairwise distance exceeds r_c, still $\mathcal{O}(n^2)$ distances need to be calculated at every

simulation time step in the first place. A standard approach to address this bottleneck and to reduce the number of necessary distance computations, is found in the buffered Verlet list.[25] Figure 4.22 illustrates, how it works.

The basic reasoning behind the Verlet list approach, is that within each MD propagation step atoms travel only a very short distance. Pairs that have been identified as close enough that a force evaluation would be forth it at some point in time t, will most likely form a relevant pair also at time $t + \tau$. A redundant distance calculation to find pairs, i.e. atoms that are neighbours with respect to the cut-off radius, may therefore be saved. The straightforward approach would now be, to determine the neighbour pairs only in regular intervals and to recycle this information for a number of simulation time steps. We may run into problems with this, though, when we use exactly the cut-off radius to find the neighbours, because the assumption that the neighbourhood does not change in between time steps may not always hold. Atoms may enter the relevant cut-off range and an error will be made when their contribution is not considered.

Verlet's solution to this, is to compute the neighbours at an increased radius r_v buffered with a certain safe zone. In the beginning, the resulting neighbour list will contain pairs that are strictly not relevant because they exceed the cut-off distance r_c. On the upside, particles may drift now during propagation from the buffer zone into the cut-off range, which is no problem because they will be considered. We only need to ensure that the buffer is large enough so that no (or acceptably few) particles can drift from outside the safe zone into the cut-off range, before the neighbour list is re-computed. Two basic options exist here: first we could choose a fixed update interval that could in turn be based on the expected maximum distance travelled by the particles in a certain time. This distance is essentially a function of temperature. Taking the form of the respective potential into account, it is also possible to set the buffer size indirectly via an error tolerance (so done for example in GROMACS via `verlet-buffer-tolerance`). Alternatively, one could keep track of the maximum current atom displacement and update the neighbour list only on demand. While this can be more accurate, it causes additional overhead and can be less performant.

Verlet list

Besides the storage of neighbour pair information, also the way how these neighbours are computed in the first place can be improved, compared to a brute force calculation of all pairwise distances. While naive search is not always a bad option, especially not in high-dimensional spaces,[158] the usually 3-dimensional and regular simulation box used in MD simulations is prone to be tackled differently. The most widely applied neighbour search schemes are either *tree*-based or *grid*-based.[159] Tree-based methods, like *kd*-trees,[160, 161] separate the set of atoms in a simulation system iteratively into smaller and smaller portions according to same criterion. After this has been done, neighbour queries supported by the tree structure can be efficiently realised because neighbours of a specific point are likely to be found in the same leaf node or can be systematically searched for in other leaf nodes. A potential drawback of trees can be that they may require a rather long time to build, and as a neighbour list is usually only constructed once before the tree needs to be re-built, this can be a non-negligible factor. Another complication is that trees often require dynamic memory allocation but implementations can differ substantially here.

tree search

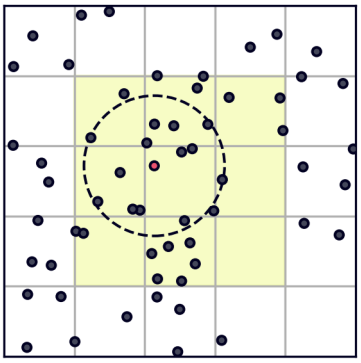


Figure 4.23 2D Grid neighbour search To find the neighbours of the red point efficiently, we first need to tile the simulation box into regular grid cells. The minimum size requirement for these is the non-bonded cut-off value r_c. Then we can limit our search for neighbours to the points in the same cell as the red particle and in all neighbouring cells. This are only nine in total compared to 25 if we had to search the complete box. The gain in performance is larger if r_c is small compared to the simulation box size.

grid search

In contrast, grid-based approaches may introduce a smaller overhead. The basic idea is, similar to trees, to decompose the positional space of the atoms into fixed size grid cells before neighbour searches are performed. A simulation box can be for example tiled into virtual cells of size r_c in each dimension by mapping atom coordinates to a cell index, like

$$I_i^{\text{cell}} = q_i \bmod r_c \tag{4.52}$$

where q_i is a positional component in the ith dimension, mod is the modulo operator (see also section 4.4 for a short discussion), and it is assumed that the box origin lies at 0 with respect to all dimensions. If we known the cell index for a certain point and the points associated with a given cell, all neighbours can be found by only searching one cell and its direct neighbours (27 cells in total in three dimensions). Figure 4.23 illustrates this in two dimensions. In practice, this indexing can be a bit more involved and may require a pre-sorting of cell and particle indices for good performance.[162, 163] Furthermore, there are improved computer architecture specific pair search schemes available.[164]

4.6 Integrators

An integral part of every MD simulation is a numeric integration scheme, or more precisely a method to find an approximate solution to differential equations, i.e. the equations of motion of a given molecular system. MD integrators use finite time differences τ to solve the equations in steps. The number of available methods is immense, there is no single superior one, and new methods are still actively developed. This section should only give a rough overview.

harmonic oscillator

In section 4.1 we saw a few examples already for simple dynamic systems, but let's consider one more for a brief look on typical MD integrators. The equation of motion for the one dimensional classical harmonic oscillator is $m\ddot{q} = -kq$, where $q = r_{ab} - r_0$ could be the displacement of a chemical bond from the equilibrium setting, modelled according to equation 4.9. This can be immediately stated using Hooke's law but could be also derived using the Lagrangian $\mathcal{L} = E_{\text{kin}} - E_{\text{pot}} = 1/2\, m\dot{q}^2 - 1/2\, kq^2$, and equation 4.7 with $\partial\mathcal{L}/\partial\dot{q} = m\dot{q}$ and $\partial\mathcal{L}/\partial q = -kq$. An analytic solution for this case would be

$$q(t) = q_0 \cos \omega t + \dot{q}_0/\omega \sin \omega t\,, \tag{4.53}$$

$$\dot{q}(t) = \dot{q}_0 \cos \omega t - q_0 \omega \sin \omega t \tag{4.54}$$

$$\ddot{q}(t) = -\omega^2 (q_0 \cos \omega t + \dot{q}_0/\omega \sin \omega t) \tag{4.55}$$

starting with initial position q_0 and velocity and $\dot{q}_0$, and with a relation for the force constant $k/m = \omega^2$ (compare equation 4.12). Figure 4.24 shows the analytic evolutions of bond length and respective velocity for a C-C bond vibration with $k = 259,408\,\text{kJ/(mol nm)}$ and equilibrium bond length $r_0 = 0.1525\,\text{nm}$.

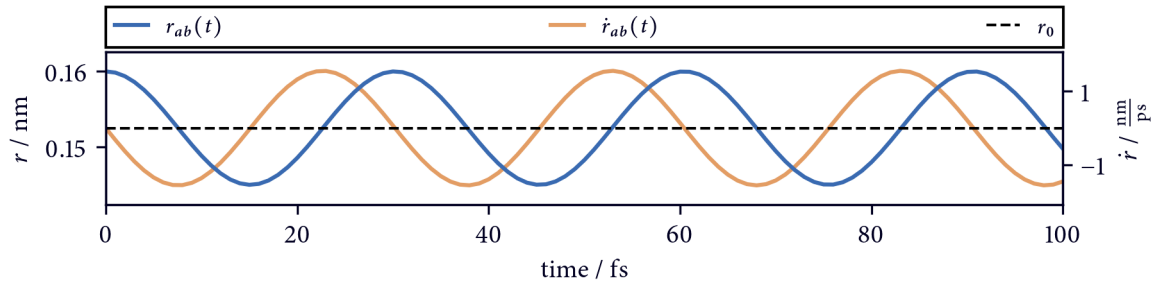


Figure 4.24 **Analytic solution for a harmonic C-C bond vibration**
Starting at $q_0 = r_{ab}(t = 0) - r_0 = 0.16\,\text{nm}$ and $\dot{q}_0 = 0$, the trajectory taken by the system with respect to the single coordinate is shown for 100 fs. The period of the vibration is about 30 fs.

A straightforward approach to the numeric prediction of this trajectory is given by the Euler method. Thinking of the kinematic equation for uniform motion (equation 4.4), we can understand the Euler method as using this as an approximation at any integration time step. Given current positions and velocities, first the forces are computed. Assuming the respective acceleration to be constant, we can propagate the system for a very short time (the chosen integration time step τ) before we recompute the forces and repeat. The Euler method can be also thought of as an approximation to the real trajectory by a local Taylor series, cut off after the first term. In other words, it tries to find an approximation to a nearby point on a curve by following the tangent to the curve at the current point. The propagation steps for position and velocity with respect to a single coordinate are:

Euler method

$$q(t+\tau) = q(t) + \dot{q}(t)\tau\,, \tag{4.56}$$
$$\dot{q}(t+\tau) = \dot{q}(t) + \ddot{q}(t)\tau\,. \tag{4.57}$$

Actually it is often also encountered in form of the 'midpoint' algorithm (note the acceleration term in the position update) with

$$q(t+\tau) = q(t) + \dot{q}(t)\tau + 1/2\,\ddot{q}(t)\tau^2\,, \tag{4.58}$$
$$\dot{q}(t+\tau) = \dot{q}(t) + \ddot{q}(t)\tau\,. \tag{4.59}$$

While this does in principle work, it has a few problems, which makes the scheme inappropriate for MD simulations. As figure 4.25 indicates, it can not reproduce the analytic solution very accurately for an integration time step of 1 fs. Lowering this to 0.1 fs can defer the problem only for a short time.

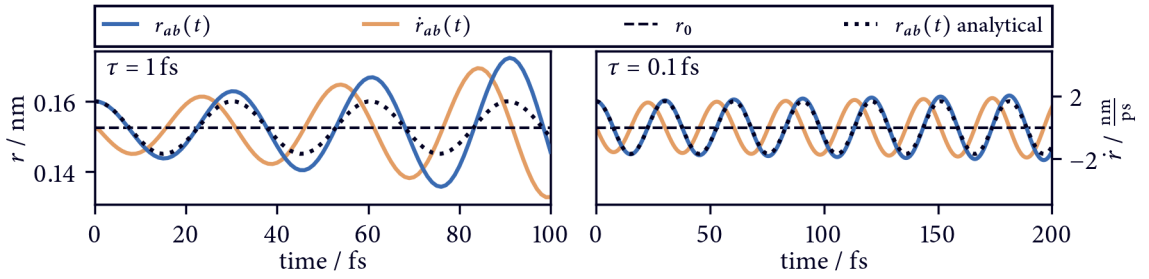


Figure 4.25 **Midpoint Euler solution for a harmonic C-C bond vibration**
For the same system as in figure 4.24, the positions and velocities were approximated using the Euler scheme. For any time step, the method does not preserve energy and can become unstable.

The approximation error introduced with the Euler method is proportional to the used step size τ^2 and the accumulated error over time is linearly proportional to τ, which makes it a 1st order method. The midpoint variation is 2nd order in the positions but this is still not very accurate in a MD context.

phase space volume

Usually, 3rd order position accuracy is taken to be required.[165] It makes the method in reverse very expensive if a very small step size has to be chosen. But more severely, this method will not conserve the total energy of the system, in fact it will lead to energy amplification. This can be in particular seen in the plots of $\dot{r}$ versus r in figure 4.26 through which it becomes clear that the phase space volume is not conserved. As a consequence, the method may not be stable over longer simulation times. Moreover, the Euler method is not time reversible (also called asymmetrical). Integrating a curve backwards, will usually not result in the same trajectory as forward integration because in each integration interval only information at the beginning of the interval is used.[166]

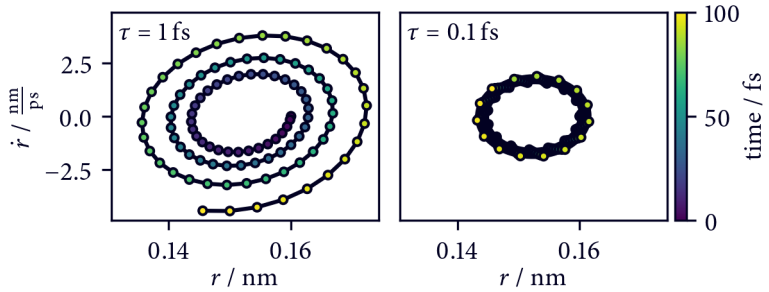


Figure 4.26 Midpoint Euler for a harmonic C-C vibration (phase space) The outward spiral pattern of the trajectory in phase space indicates an amplification of the energy.

implicit/explicit

The standard Euler method is sometimes called the *forward* or *explicit* Euler, in contrast to the *backward* or *implicit* variation. The backward Euler uses the positions $q(t+\tau)$ to compute the forces but in order to find these in the first place, an energy minimisation is actually needed at each step. Its use in MD simulations seems to be disappointing at best.[167] Other notions of explicit versus implicit are *open* versus *closed* or *predictor* versus *predictor-corrector* methods.[165]

There is yet another possible formulation, wich is the semi-explicit or symplectic Euler, so far addressed as Euler-Cromer method in previous sections. The difference in the propagation steps is small but impactful:

$$\dot{q}(t+\tau) = \dot{q}(t) + \ddot{q}(t)\tau\,, \tag{4.60}$$

$$q(t+\tau) = q(t) + \dot{q}(t+\tau)\tau\,. \tag{4.61}$$

Note that the position update uses the velocity from the next time step.[8] While somewhat counter intuitive, this variation is energy preserving, at least for oscillatory problems.[168]

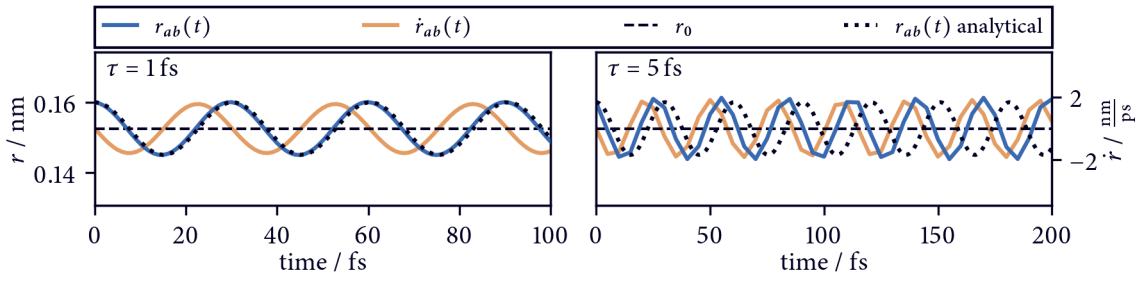


Figure 4.27 Euler-Cromer solution for a harmonic C-C bond vibration
For the same system as in figure 4.24, the positions and velocities were approximated using the Euler-Cromer scheme. For the given problem, the method conserves the energy. The time step can be increased drastically but then the real particle trajectory is not reproduced very exactly anymore.

As figure 4.27 and 4.28 show, the Euler-Cromer scheme has the interesting property of conserving the energy in the respective system, even for very large time steps. The phase space volume is

[8] The midpoint Euler above can be actually obtained by using the average of velocities at the current and the next time step $q(t+\tau) = q(t) + 1/2(\dot{q}(t) + \dot{q}(t+\tau))\tau$ and substitute $\dot{q}(t+\tau) = \dot{q}(t) + \ddot{q}(t)\tau$.

preserved, although distorted quite a bit when the time step is increased. Two points should we note here. Accuracy in terms of MD integrators is more than just the order of the method. Algorithms that preserve the phase space volume can be more usable than others of the same order that do not.[169] And second, accuracy can primarily mean to sample from a thermodynamic ensemble rather than shadowing actual particle trajectories exactly.

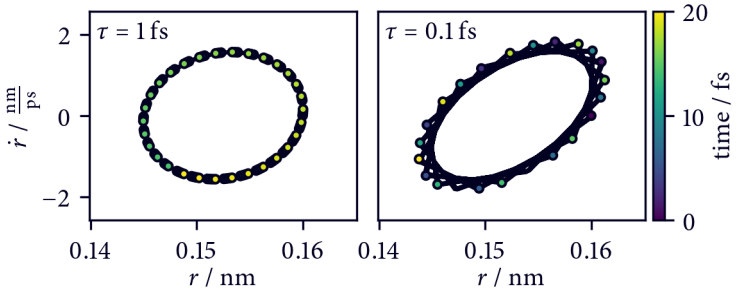


Figure 4.28 Euler-Cromer for a harmonic C-C bond vibration (phase space) The stable circular pattern of the trajectory in phase space indicates a conservation of energy.

But also the Euler-Cromer scheme is in general not time reversible. An integrator that is time reversible is the Verlet integrator.[25] It is also know as the explicit central difference algorithm and contains the update steps *Verlet*

$$q(t+\tau) = 2q(t) - q(t-\tau) + \ddot{q}(t)\tau^2\,, \tag{4.62}$$

$$\dot{q}(t) = (q(t+\tau) - q(t-\tau))/(2\tau)\,. \tag{4.63}$$

It can be derived by summing the approximations for a step in positive and negative direction from a harmonic Taylor expansion at $q(t)$. The motion is integrated independently of the velocities, which can be obtained from the positional change. The algorithm is not self-starting, though, and $q(t-\tau)$ needs to be obtained from a different source beforehand.

A reformulation[9] of the Verlet scheme is found in the leap-frog integrator that is also used in our simulations.[170] The position update is here done using the velocity of a half step *leap-frog*

$$\dot{q}(t+1/2\,\tau) = \dot{q}(t-1/2\,\tau) + \ddot{q}(t)\tau\,, \tag{4.64}$$

$$q(t+\tau) = q(t) + \dot{q}(t+1/2\,\tau)\tau\,. \tag{4.65}$$

This algorithm has a 3rd order position accuracy and exhibits essentially no energy drift. The trajectories produced by the leap-frog and the Verlet algorithm are identical. There is another mathematically equivalent formulation in terms of the so called velocity Verlet scheme with[33]

$$q(t+\tau) = q(t) + \dot{q}(t) + 1/2\,\ddot{q}(t)\tau^2 \tag{4.66}$$

$$\dot{q}(t+\tau) = \dot{q}(t) + 1/2(\ddot{q}(t) + \ddot{q}(t+\tau))\tau\,. \tag{4.67}$$

The leap-frog and the Verlet integrator will (without further modification of velocities) produce identical trajectories when the starting points are correspondingly shifted. The advantage of the latter is that it is self-starting and gives synchronised instead of overlapping velocities and positions, which can be advantageous if temperature coupling is used (see section 4.8). Compared to the plain Verlet, velocity Verlet and leap-frog can be numerically more stable because the velocities are not obtained as averages of positions, i.e. from two numbers of possibly very similar proportion.[171] *velocity Verlet*

As an interim conclusion, we can state that suitable integration algorithm for MD have ideally a relatively high order, and are time reversible and phase-space preserving. Another important practical point, however, is also how many force calculations per time step are necessary because this is the most expensive part of the whole simulation. Accurate methods like the 4th order Runge-Kutta method *higher order methods*

[9] The leap-frog scheme can be traced back to the Verlet scheme if the expression for $\dot{q}(t+1/2\,\tau)$ is put in the expression for $q(t+\tau)$ and we then use $\dot{q}(t-1/2\,\tau) = (q(t) - q(t-\tau))/\tau$.

(which is, however, not symplectic by the way) are therefore typically not used because they require multiple force calculations per time step.[165] It should also be mentioned that the overall accuracy in a simulation is further determined by other factors like non-bondend cut-offs (see section 4.2) and that higher order integrators may be wasted in the light of larger errors from these sources. A further point in the general assessment of integrators is their interplay with constraints and respective algorithms that satisfy these.[171]

stochastic dynamics

Besides deterministic integrators, there exists a broad family of stochastic integrators as well. The equations of motion addressed by these integrators are non-differentiable stochastic differential equations like the Langevin equation to describe Brownian motion[172]

$$m\ddot{q} = -\gamma m\dot{q} + \eta(t) \tag{4.68}$$

where particle acceleration is determined by the interplay between a friction term with scaling coefficient $\gamma > 0$ and a random thermal noise term, which can for example come from a Wiener process $\eta(t) = \sigma \dot{W}_t$, with proportionality $\sigma = \sqrt{2k_\mathrm{B}Tm\gamma}$. For a molecular system governed by a potential of internal forces, the dynamical equation can also be written as

$$m\ddot{q} = -\partial E_\mathrm{pot}/\partial q - \gamma m\dot{q} + \eta(t)\,. \tag{4.69}$$

A solution to this can be for example approximated using the Euler-Maruyama integrator with update steps[173]

$$q(t+\tau) = q(t) + \dot{q}(t)\tau\,, \tag{4.70}$$

$$\dot{q}(t+\tau) = \dot{q}(t) - \partial E_\mathrm{pot}/\partial q\,\tau/m - \gamma\dot{q}\tau + \sigma/m\,\mathcal{N}(0,\tau)\,, \tag{4.71}$$

or for the high friction limit (overdamped case), where $\gamma m\dot{q} \gg m\ddot{q}$ and the left side of equation 4.69 can be set to zero, just

$$q(t+\tau) = q(t) - \partial E_\mathrm{pot}/\partial q\tau/(\gamma m) + \sigma/(\gamma m)\mathcal{N}(0,\tau)\,. \tag{4.72}$$

Here $\mathcal{N}(\mu, \sigma_\mathcal{N})$ denotes a sample from a normal distribution with mean μ and variance $\sigma^2_\mathcal{N}$. While the scheme is completely valid for the overdamped case, it has divergence issues for the unmodified Langevin equation.[174] Quite a few more sophisticated methods exist, some of which have very interesting properties for the simulation of biomolecular systems.[85, 175, 176]

4.7 Velocity generation

When the initial conditions at the start of a molecular simulation are chosen, it may be desired to set the velocities of individual atoms to a meaningful value. A popular way to do so, is to select them in accordance with probabilities from a Maxwell-Boltzmann distribution at a certain temperature T. Let's consider a system of n particles in three dimensional cartesian coordinates with a respective velocity vector $\boldsymbol{v} = (v_1, ..., v_{3n})$, or alternatively n per particle velocity vectors $\boldsymbol{v}_a = (v_x, v_y, v_z)$. The Maxwell-Boltzmann probabilities for velocities along single coordinates v_i are given as

$$p(v_i) = \left(\frac{m_i}{2\pi k_\mathrm{B}T}\right)^{1/2} \exp\left(-\frac{m_i v_i^2}{2k_\mathrm{B}T}\right), \tag{4.73}$$

where m_i is the corresponding particle mass and k_B is Boltzmann's constant. For absolute velocities $v_a = ||\boldsymbol{v}_a||$ of particles a, the equation takes the form:

$$p(v_a) = 4\pi v^2 \left(\frac{m_a}{2\pi k_\mathrm{B}T}\right)^{3/2} \exp\left(-\frac{m_a v_a^2}{2k_\mathrm{B}T}\right). \tag{4.74}$$

The generation of velocities can follow a simple scheme,[10] beginning with the sampling of values v_i from a normal distribution with the standard deviation $\sqrt{k_B T/m_i}$ of equation 4.73. In a second step, the center-of-mass velocity is computed from the velocity vectors $\boldsymbol{v}_a$ as

$$\boldsymbol{v}_{\text{com}} = \sum_{a=1}^{n} \boldsymbol{v}_a \frac{m_a}{m_{\text{tot}}} \tag{4.75}$$

with the system's total mass m_{tot}, which is then subtracted from the obtained velocities so that $\boldsymbol{v}'_a = \boldsymbol{v}_a - \boldsymbol{v}_{\text{com}}$. The generated velocity distribution may now not exactly match the desired temperature. To correct this, all velocities can be scaled by a factor, $\boldsymbol{v}' = \alpha \boldsymbol{v}$ with

$$\alpha = \sqrt{\frac{T_{\text{target}}}{T_{\text{system}}}}\,. \tag{4.76}$$

For this, the current temperature of the system T_{system} needs to be calculated which can be done via computing the total kinetic energy from the particle velocities:

$$E_{\text{kin}} = \frac{1}{2} \sum_{a=1}^{n} m_a ||\boldsymbol{v}_a||^2\,, \tag{4.77}$$

$$T = \frac{2E_{\text{kin}}}{n_{\text{dof}} k_B}\,. \tag{4.78}$$

A system of n particles has in general $3n$ degrees of freedom that distribute over translation, rotation, and vibrational modes. If the center-of-mass translation of a system as a whole is set to zero, 3 degrees of freedom have to be subtracted from this number. The same goes for 3 more if also the center-of-mass rotation is suppressed. Further, a number of coordinates can be fixed by constraints, so that the final number of degrees of freedom is $n_{\text{dof}} = 3n - n_{\text{com}} - n_{\text{constr}}$. Figure 4.29 shows an expected velocity distribution for argon atoms next to a distribution generated by the described scheme.

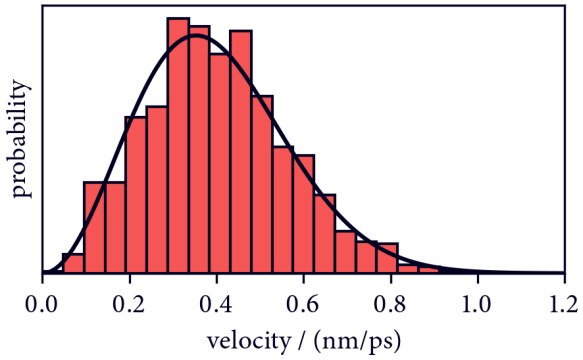


Figure 4.29 Maxwell-Boltzmann distribution Distribution of particle velocities for argon at 300 K according to equation 4.74 (black line). Generated velocities for a system of 1000 argon atoms (red bars).

A short practical remark: in SI units k_B is given in J/K, masses should be given in kg, and velocities in m/s, so that energies come out in J. This can be rather inconvenient because we have to deal with unhandy numbers (e.g. $m_{\text{Ar}} \approx 6.64 \cdot 10^{-26}$ kg). It can be preferable to work in molecular units by using R in kJ/(K mol) instead of k_B, masses in g/mol, and velocities in nm/ps. Energies are then obtained in kJ/mol.

4.8 Thermostats

In molecular simulations, it is usually desired to sample conformations from a thermodynamic ensemble. Without extra controls and a non-stochastic integrator, this is the microcanonical *NVE*

[10] So seen for example in GROMACS and Lumol

ensemble in which the number of particles n, the system's volume V, and its total energy E are constant. Thermostats can be used to sample from a canonical NVT ensemble at constant temperature T instead. Here, 'constant' temperature means constant on average and might be better understood as conserved. The instantaneous kinetic energy and therefore temperature of (small groups of) particles is in contrast not fixed and expected to fluctuate. A correct average temperature and correct standard deviation due to fluctuations of the right magnitude correspond to a correct canonical ensemble.

A reason for why one would want to use a thermostat in a simulation, is the better comparability of results with experiments that where performed at constant temperature. On the other hand, it can be used to counter practical problems with energy conservation that may arise for example from numerical inaccuracies, the presence of external forces, or technical parts like the truncation of non-bonded interactions. It has to be paid attention, though, that a thermostat is not used purely out of the motivation to conceal fundamental issues with non-physical simulations.

There exists a number of technical realisations of controlling the temperature in a simulation. Not all satisfy both the demands of keeping the temperature at a set average and ensuring a physical distribution of temperatures or respectively their correct fluctuation.

One possibility, is to use the velocity generation scheme described in section 4.7 to repeatedly reset particle velocities after a certain amount of simulation time has passed, i.e. after time τ_T or after τ_T/τ steps, where τ is the simulation time step. This is basically what the Andersen thermostat entails.[30] The resetting of velocities models collisions of particles with a fictitious infinite heat bath. There is room for variation with respect to the number of simultaneously affected particles. Instead of modifying all velocities at every step (called the massive collision scheme), particles can be individually selected with a certain probability (stochastic collisions), e.g. $p_a = 1 - \exp(-\xi\tau)$ where ξ scales the collision frequency. On the upside, this approach is very effective in maintaining the desired temperature and does not negatively effect ergodicity. On the downside, randomising the velocities regularly disturbs the dynamics of the system.[177] Some extra care needs to be taken when it comes to handling constraints.[178]

An alternative is to use a weak coupling scheme as in the Berendsen thermostat in which the current velocities in the system are relaxed to match a target temperature.[179] Individual particle velocities are not reset completely but scaled by a factor similar to what has been described in equation 4.76. Choosing

$$\alpha = \sqrt{1 + \frac{n_{\text{couple}}\tau}{\tau_T}\left(\frac{T_{\text{target}}}{T_{\text{system}}} - 1\right)} \tag{4.79}$$

one obtains the new velocities as $\boldsymbol{v}' = \alpha\boldsymbol{v}$. Here, τ is the simulation time step, τ_T is the coupling constant (i.e. roughly the effective relaxation time towards the target temperature), and n_{couple} is the simulation interval in steps after which the coupling is actually applied. Since the rescaling suppresses temperature fluctuation, the produced ensemble is technically incorrect, which is, however, less severe for larger systems.

The thermostat mostly used in our simulations is a modification of the Berendsen scheme. It is called the v-rescale thermostat in GROMACS but can elsewhere be found as the Bussi, or *canonical sampling through velocity rescaling* (CSVR) thermostat.[180] The addition to the original rescaling approach is that the target temperature to which the system is relaxed is randomised to ensure correct fluctuations. This is essentially achieved through a stochastic integration step that propagates the current kinetic energy to a new value from the canonical distribution. The velocities of the system can than be scaled towards this kinetic energy as a target value.

Similar to thermostats, there exist methods for pressure control as well. These should not be discussed here, only so much as that pressure in a simulation can be computed from the kinetic energy and the virial tensors

$$\boldsymbol{P} = \frac{2}{V}(\boldsymbol{E}_{\text{kin}} - \boldsymbol{E}_{\text{vir}}) \tag{4.80}$$

from which the scalar pressure can be obtained as $P = \text{trace}(\boldsymbol{P})/3$. Note that the kinetic energy tensor can be computed as $1/2 \sum_a^n m_a \boldsymbol{v}_a \otimes \boldsymbol{v}_a$ using the outer product of velocity vectors for individual particles. The virial only depends on dispersive interactions (typically Lennard-Jones) and can be computed from contributions of considered pairs a and b as

$$\boldsymbol{E}_{\text{vir}} = -\frac{1}{2}\sum_{a,b} \boldsymbol{r}_{ab} \otimes \boldsymbol{F}_{ab}\,. \tag{4.81}$$

If a non-bonded cut-off is used, similar consideration as for energy correction (see for example equation 4.37) apply also for the pressure.

4.9 Steered Molecular Dynamics

As we saw in section 4.2, a MD simulation is governed by the forces considered to be present in a system. This can be extended from intra-molecular force contributions with physical interpretation to additional external (artificial) forces. In so called *steered* MD experiments,[181, 182] an otherwise conventional MD setup is employed with the addition of a *pulling* force along a pre-set reaction coordinate. The aim of adding this force is to assess the response of the simulated system. It can for example be used to force a system out of a very stable state and observe transitions that are too slow to be sampled within realistically achievable time scales. Moreover, it can provide measurements of how strong the probed interactions are that resist the applied pulling force.

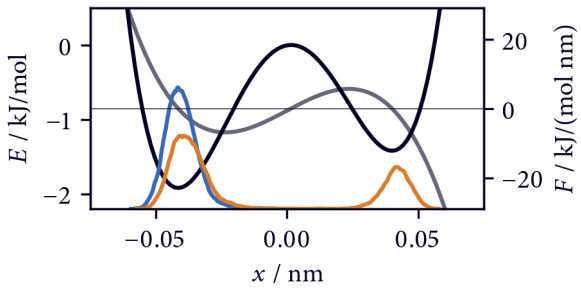


Figure 4.30 Steered dynamics in a double well potential Particle simulated in an asymmetric double well potential of the form $E = ax^4 - bx^2 + \sigma\sqrt{a/(2b)}x$, with $a = 60$, $b = 20$, and $\sigma = 0.5$, using an Euler-Maruyama scheme. Thermal energy is not sufficient for the particle to cross the central energy barrier (blue). Applying a constant bias in x direction, the particle can overcome the barrier (orange). The applied force plus thermal contribution needs to be larger than the maximum force created by the barrier.

In detail, pulling experiments can be designed in slightly different ways. For one thing, a constant force could be applied that uniformly nudges atoms in a certain direction. Figure 4.30 shows a very simple example of this. It might be used to pull a small molecule trough a membrane or ions through a channel.[183–185] In this case it is often common to track the displacement of the pulled group along the pull coordinate. The applied force can help that certain energetic barriers can be overcome. It should be noted, though, that the system can still be stuck in stable states when the force is not large enough. On the other hand, very large artificial forces can disturb a system substantially, creating unrealistic simulation artefacts. In general, it might be desirable to find a compromise and run repeated simulations with different forces to see how the system reacts.

Alternatively, the applied force can be constantly increased with time. This can be useful if the force necessary to overcome a certain barrier is not known but the barrier should be crossed in any case. In contrast to *constant force* steered MD, this can be called a *constant velocity* experiment. Instead of the force, one has to choose a rate at which the force should build up. Again one usually needs to find a

compromise between fast pulling to save simulation time and slow pulling to give a molecular system time to adjust to the increased force. It may especially play a role if pulling experiments should be compared for the same system in different conformations because than the pulling should probably be done faster than the relaxation time of the probed conformation. In constant velocity steered MD simulations, the simulated system is never really in equilibrium. It can be for example used to assess the force necessary to pull out a ligand from a binding-site,[186] to separate protein monomers, subdomains, or complexes[43, 187] or to unfold a helix.[188]

During such an experiment, the force acting on the atoms pulled upon can be recorded. The maximum recorded force can be taken as an indicator for the slope of the steepest energy well. To make this maximum force manifest itself in terms of a so called rupture event, the time-dependent force increase should not be modelled directly as an external force but rather has a harmonic spring force, though. Here, one has again multiple options to realise this in terms of a reaction coordinate. It is possible to choose an arbitrary axis along which to pull. Figuratively spoken, one could think of this situation as attaching a spring to a certain atom, which is than displaced at a predefined rate and translates into a force on the atom in proportion to a force constant that also needs to be chosen. As long as there are resiting forces holding the atom pulled upon in place, the force will steadily build up, before a point is reached where it becomes to high. This point will be visual in a force trajectory as a sudden decrease of the force felt by the atom because it can finally follow the spring it is attached to.

Less illustrative but often preferable is actually to choose an internal coordinate that a harmonic force could be applied to. When a pulling is done in a fixed direction in space, it may become necessary to suppress translational and rotational degrees of freedom in the system to maintain the intended pulling. This can be realised via position restraints and it might be okay. In general, however, it is an additional artificial factor influencing the system and if possible, it should be avoided. One can choose for example a certain distance, say the center-of-mass distance between two groups that should be pulled apart, or an angle. The pulling causing an increased force along this coordinated can be imagined as an displacement of the respective equilibrium value. No position restraints or other restrictions need to be put on the system. It should be ensured, however, that the simulation box is large enough in any direction to account for the effective size increase due to the pulling (see also section 4.4). Fixing the orientation of a molecule in space and pulling into a certain fixed direction, can be computationally cheaper because than the box volume may only need to be increased along one axis.

Rupture forces are interesting because they can provide an estimate of the steepness of potential energy barriers. Assuming a situation in which a system is only altered slightly, a steeper energy barrier is, however, also an indicator for a deeper energy minimum before the barrier. Loosely, this stretches the argumentation of the Hammond postulate, stating that for comparable transition states, endothermic reaction rates are determined by the stability of the starting material.[189] In our langerin study presented in chapter 11, we take the force needed to pull out a calcium ion from the protein's binding site as a weak proxy for the energy of the bound structure. In other words, protein states of decreased calcium-binding affinity should be on first order detectable by comparably low rupture forces and how tightly the ion is bound should affect how hard it is to remove it from the binding site. In this context, where relative rupture forces are compared it should be stressed that the pulling velocity should be carefully balanced so that the system has enough time to adjust to the pulling. A too swift increase of the pulling force can lead to artificially high rupture forces but the slower the pulling, the higher the probability that the system undergoes conformational transitions, which may dilute the comparison between conformational states.

Beyond that, it is indeed possible to translate the force needed to provoke a certain reaction of a system into a work estimate. Approaches to do so, are based on the Jarzynski-equality

$$\exp(-\beta\Delta G) = \langle \exp(-\beta W) \rangle \,, \quad \text{with } \beta = k_\mathrm{B} T \tag{4.82}$$

that connects the free energy difference of two end states A and B with the average work done in the transformation process—independently of the speed of the process.[44] It provides a justification for the estimate of equilibrium free energy differences from non-equilibrium work. It is also possible to construct a potential of mean force.[188, 190] For the practical realisation of these estimates see [191] and recently [192].

{ 5 }

Molecular trajectory analysis

Making sense of high-dimensional data

A successfully performed MD simulation does usually generate *output* of some kind. We start with an initial configuration of the simulated system at time $t_0 = t(s = 0)$ and propagate it in time for example by regular time intervals of $\Delta t = \tau$ to get configurations at time steps $t(s+1) = t(s) + \tau$. After n_s time steps, the simulation will finally arrive at a configuration at time $t(n_\mathrm{s})$. At the very least, the simulation output will probably contain the configuration of the system at this last time step for a subsequent analysis. Typically, though, configurational snapshots are taken every nth time step, say after each picosecond. It is also common to record only a relevant subset of the configuration (say only protein atoms) or to write out different parts of the configuration at different intervals (say protein and solvent combined only after each nanosecond). Additionally, computed quantities like temperature, pressure, or energies can be included in the output at certain time steps. Most of the quantities we should be concerned with here, however, can be derived more or less directly from the system's coordinates and can be computed also at a later stage. As in the previous chapters, the configuration of a molecular system (neglecting momenta) will be denoted as a vector $\boldsymbol{q}(t)$ or rather $\boldsymbol{q}(s) = (q_1, \ldots, q_{n_\mathrm{d}})$ where n_d is the dimensionality of the system (e.g. $n_\mathrm{d} = 3n_\mathrm{a}$ cartesian coordinates). A MD trajectory is then an ordered set of configurations $\mathcal{Q} = \{\boldsymbol{q}_0, \ldots, \boldsymbol{q}_{n_\mathrm{s}}\}$ with n_s denoting the total number of output steps rather than propagation steps. $\mathcal{Q}$ is a statistical sample set ideally obeying the underlying Boltzmann distribution of the system, i.e. containing lower energy configurations with higher probability. $\mathcal{Q}$ can also be written as a function with discrete domain $\mathcal{Q}(s) = \boldsymbol{q}_s$, with $s \in [0, n_\mathrm{s}]$, mapping time steps to configurations.

simulation output

With that, MD analysis mostly entails the analysis of time series and corresponding distributions. An important foundation for the fact that we are able to conclude anything meaningful from a time series of molecular configurations, is the ergodic theorem.[193] For a quantity $f(\boldsymbol{q})$, its expected value is given by an integral over all possible configurations of the configurational space Ω, that is $\mathbb{R}^{3n_\mathrm{a}}$ for a system of n_a atoms in cartesian coordinates, as

ergodic theorem

$$\langle f \rangle = \int_\Omega \rho(\boldsymbol{q}) f(\boldsymbol{q}) \, \mathrm{d}\boldsymbol{q} \,. \tag{5.1}$$

Each configuration is weighted here by its observation probability ρ. Practically, the same should be obtained by conducting a measurement of f for a very large number of molecules n_m, distributed according to ρ at a certain point in time (as for example done in a macroscopic experiment):

$$\langle f \rangle = \lim_{n_\mathrm{m} \to \infty} \frac{1}{n_\mathrm{m}} \sum_{i=1}^{n_\mathrm{m}} f(\boldsymbol{q}_{t,i}) \,, \tag{5.2}$$

with $\boldsymbol{q}_{t,i}$ being the configuration of the ith molecule at time t. The ergodic theorem states that this average over configurations of molecules can be, in the limit of infinite sampling as the time goes to

infinity, substituted with a time-average for a single molecule adopting configurations distributed according to ρ:

$$\langle f \rangle = \lim_{n_s \to \infty} \frac{1}{n_s} \sum_{s=0}^{n_s} f(\boldsymbol{q}_s)\,, \tag{5.3}$$

with $\boldsymbol{q}_s$ being the configuration of the molecule at time $t = t_s$. Of course, limited sampling in MD simulations can be problematic when ensemble averages are estimated from a time-series.

In the analysis of MD data, one is beyond that often not only concerned with the computation of a quantity from full trajectories to reproduce wet-lab experimental data, but also with an assertion of the underlying probability distributions and dynamic processes. These are normally not accessible in macroscopic experiments but can give valuable insight into which molecular states are responsible for an observation in terms of an average and how it can be broken down into possible contributions. In other words, a configurational sample set from a MD simulation allows us to separate subsets of configurational states (i.e. conformations[1]) for which quantities can be computed separately. This chapter will be focused on analyses techniques to identify and separate conformational states from a complete molecular ensemble rather than on the quantities that can be calculated from it. It will concentrate on proteins as the systems of interest but it is in general also valid for other kinds of molecules.

conformations

A molecular configuration $\boldsymbol{q}$ can be very high-dimensional, proportionally to the number of considered atoms that can easily be on the order of 10^3 to 10^6. It is therefore a non-trivial task to extract conformational information from the data. For a first-order impression, a time series of molecular coordinates can be inspected visually in full in terms of a movie. An expert might be able to spot relevant conformations and to draw rough conclusions about changes over time and differences to other systems. In spite of the potential usefulness of such observations to guide subsequent research efforts, they will, however, always remain anecdotal and a purely visual analysis will become more difficult with the increasing complexity of the systems.

In the following, a generally applicable workflow (section 5.1) to extract meaningful information from high-dimensional MD data will be discussed. Then, a few standard methods are explained as they found usage in our work (section 5.2). A separate section each is dedicated to specific transformation techniques applied to the effect of dimensionality reduction (section 5.3) and to mutual information analysis (section 5.4). The broader topic of Markov models is addressed in its own chapter (chapter 6), as well as the theory related to clustering that is included in the respective part (part V) due to its central relevance for this work.

5.1 A universal workflow

MD HAS A LONG HISTORY as briefly outlined in chapter 2 and so does its analysis. A set of standard tools and approaches has evolved that is commonly applied to extract and present information from molecular trajectories. Despite being fairly established, the right combination of analyses may be non-trivial to find in specific use-cases, though. A successful trajectory analysis is tightly bound to

[1] Note the use of *conformation* here in a chemical sense to describe a sub-ensemble of similar configurations that can be structurally different but are associated with the same (smoothed) energy basin. In this sense, it includes the IUPAC definition of 'conformation' as *'the spatial arrangement of the atoms affording distinction between **stereoisomers** which can be interconverted by rotations about formally single bonds'*.[194] This use is in contrast to a possible alternative distinction of a conformation as a single point in positional space from a single point in phase space (including momenta). I refrain, on the other hand, from the stereochemical use of *configuration* to distinguish stereoisomers that can not (readily) interconvert, like for the E/Z-isomerie of double bonds.[194] For a short discussion of molecular conformations from the perspective of density-based clustering, see section 13.1.

a sensible setup of the respective simulations. Beyond asking the question *'How can I technically simulate a system?'*, it may help to contemplate also over *'Why am I simulating this system?'* and *'What do I want to do with the trajectory data?'*. The answers to these questions may take real shape only with the progression of a research project but they should at least in parts be sorted out *before* a simulation is actually done. It should be avoided to acquire trajectory data, which can take a substantial amount of time and resources, and then having to realise that it is not clear what to do with it or that a slightly different setup would have been actually necessary to successfully carry out a certain analysis.

prior considerations

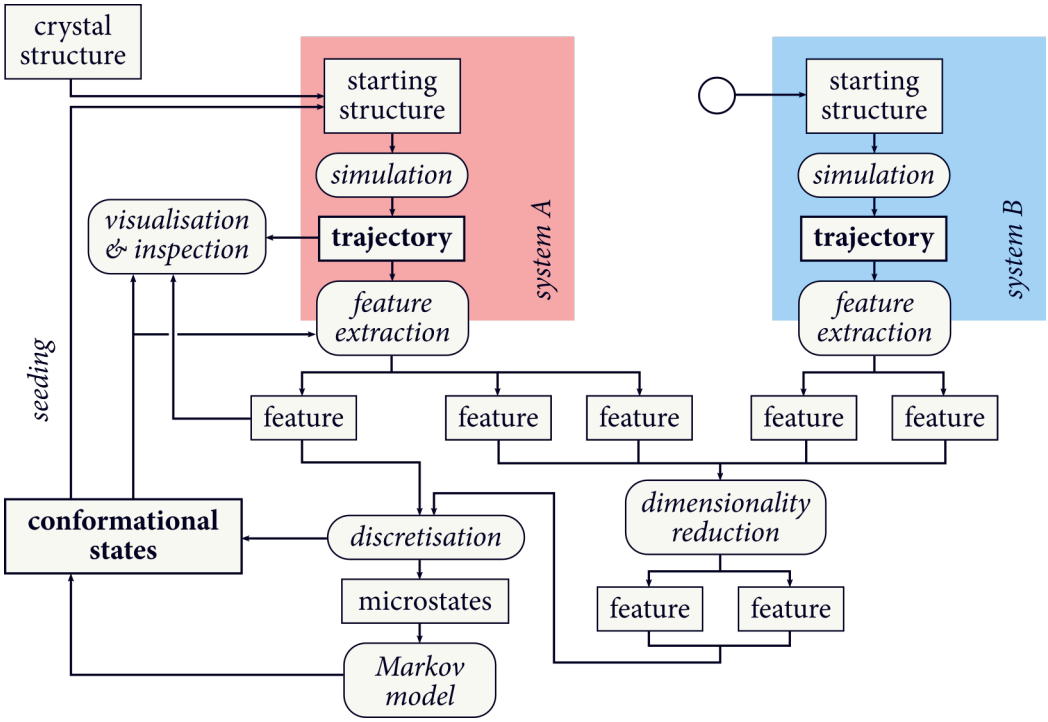


Figure 5.1 A conformational MD analysis workflow

MD simulations produce trajectories of molecular configurations starting from a given input structure. Since high-dimensional trajectories are difficult to analyse, a main effort is to select meaningful features that provide a reduced representation of the simulated system. Features can be inspected directly, compared for different systems, or further processed to give new features. Conformational states can be identified through clustering (discretisation) in feature spaces. Those are the primary result of the workflow and their inspection can be the basis to draw conclusions in a research project. They can be, among other things, also be used for sensible seeding of new simulations. Markov models can give insight into the kinetic relations between conformations or help to identify them in the first place.

The example analysis workflow in figure 5.1 should present an overview of how MD analyses can be roughly organised. It assumes as common ground that the main interest lies in the identification and characterisation of conformational states. Note that this implies some faith into the general paradigm that molecular structure is directly connected to function and differences in structure are indeed the basis to explain observed behaviour. The planning of a simulation and analysis project starts with the sensible choice of starting configurations. What this should comprise will be heavily influenced by what is expected to learned from the simulation. Often one will use available crystal or NMR solution structures but decisions have to be made in terms of the size of the model (*Do I need all residues or only a specific part of the structure? Do I need crystal water or co-crystallised ligands? Do I need to add missing parts like an unresolved side chain or loop segment, or a whole membrane?*) and in terms of environmental conditions (*Do I need to model a specific pH or salt concentration?*). It also should be considered whether the simulation of a single individual system is enough or if a meaningful analysis

would require a comparative assessment between separate simulations with different structures or setups (force fields, temperatures, protonation states, ligand complexes, mutations, etc.). In this case, it should be taken extra care that the simulations remain consistent over axes of variation that could distort a comparison. For example, it might be not ideal if systems at different temperatures use different thermostats.

feature selection and extraction

Once an array of simulations has been organised—with a later form of analysis in mind—a direct visual inspection of the obtained trajectories is possible as mentioned earlier. What one is normally interested in, however, is the selection or extraction of relevant (dynamic) *features*,[2] onto which a trajectory can be projected. The goal is to identify (collective) variables that matter to describe the behaviour of a system. Effectively we want to reduce the complexity of the data and to make it comprehensible. In this sense, a feature can be virtually anything that maps the full coordinates of a system in each time step to a single value, for example a distance between two atoms, a backbone dihedral angle, or an indicator for the existence of a hydrogen bond (see also section 5.2).

The extraction of features can be rationally guided. By using our already acquired knowledge of the system, we might be able to select a suitable coordinate that captures the aspect we want to analyse well. It can also be an iterative process, in which different kinds of standard features are investigated to explore the system and to find suitable representations. Sometimes a single feature (a 1-dimensional projection) will contain enough information, but often a set of features will be used in combination. Features can be analysed as time series, to track respective changes in the course of a simulation, or alternatively as distributions, to assess the relative populations of configurations. They can be compared to other features in the same system to reveal differences and correlations. The same shared feature can also be extracted for multiple systems to do a comparison between those.

projection error

It should be noted that while featurisation is in general indispensable to understand a molecular system, it always comes at the cost of a *projection error*. By reducing the full configuration of a system to a low dimensional projection, we are not able any more to resolve every variation in the data. While this error is often not quantified, it is desired to choose projections that are as simple as possible while limiting the loss of (relevant) information. There are methods to choose features according to an importance criterion rather than purely experience based decisions (section 5.3). These should allow one to keep only features that encode a maximum amount of information. In practice, a combination with a rational pre-filtering of features is still often required because these methods can usually not handle very high-dimensional data themselves.

Most importantly, molecular configurations in low-dimensional projections can be clustered (see part V) to separate out conformational states. These can be the key to explain the studied phenomena. In particular, a clustering into such states can be used as the basis to construct a Markov model of the conformational dynamics and to analyse the equilibrium distribution sampled by possibly manny individual replicas in a statistically sound way (see chapter 6). From the relative population of meta-stable conformations and the transition processes between them, comprehensible yet detailed descriptions of molecular systems can be obtained.

5.2 Basic features

A *feature* x could be any 1-dimensional component, directly or indirectly derived from information about a system (usually atomic positions). A collection of features (other than the full set of cartesian coordinate features), i.e. any representation of a system in terms of chosen coordinates,

[2] Technically, a differentiation between feature *selection* and *extraction* can be made with respect to whether a subset of already present features is selected or new features are computed (extracted) from the present features. The line is a bit blurry and I will not insist on it here.

can be called a *projection* $\boldsymbol{q}' = (x_1, ..., x_{n_\mathrm{f}})$ where n_f denotes the number of features. A projection $\boldsymbol{q}'$ can be still considered a configuration but in terms of positions in the feature space rather than positions in the original space. It is also sometimes called more generally a *feature vector*.

The most simple features one could select for a reduced view onto the system, are cartesian coordinates themselves without further modification. It is for example common to to pick out only the cartesian coordinates for individual C_α atoms reducing the full configuration of a system $\boldsymbol{q}$ to the position of a single atom i with $\boldsymbol{q}' = \boldsymbol{q}_i = (q_x, q_y, q_z)$. Since it is usually not very telling to look at the time series of a set of positions directly, one can instead use its RMSF with respect to a reference position

RMSF

$$\sigma_i = \sqrt{\frac{1}{n_\mathrm{s}} \sum_{s=0}^{n_\mathrm{s}} \left(\boldsymbol{q}_{s,i} - \boldsymbol{q}_i^{\mathrm{ref}}\right)^2}\,. \tag{5.4}$$

The RMSF of an atomic postion is basically its standard deviation if $\boldsymbol{q}^{\mathrm{ref}}$ is the actual mean position. To detect relatively flexible or rigid regions in a system, one can compare the fluctuation for different atoms. Note, that in principle the RMSF can be computed also for arbitrary features.

In contrast to that, one can also take a set of features like cartesian atomic positions and convert them into a new feature. The RMSD is an example for that, measuring the average displacement of feature values from a reference at a given time step as

RMSD

$$\widetilde{d}_s = \sqrt{\sum_{i}^{n_\mathrm{f}} w_i \left(x_{s,i} - x_i^{\mathrm{ref}}\right)^2}\,, \tag{5.5}$$

where w_i denotes the weight of each feature. For atomic positions, the deviations could for example be weighted by the respective atomic mass. So while the RMSF and the RMSD are often considered to be very similar, the first is a time-average over displacements with respect to a given projection and the latter is a position-average over features, giving us a new feature.[3]

RMSDs are a good example for a feature that can be routinely analysed directly as a time series or as a distribution. In the former case, it can be for example used as a basic sanity check (see for example section 11.2) or to spot large scale conformational transitions. Similar to RMSF values, the feature is relatively universal because one does not need to have prior knowledge about a system to compute it for the backbone of a protein for example. This is in contrast to say a specific interatomic or center-of-mass distance, which can only be effectively selected if its importance is already suspected. For large numbers of features (many atoms), the RMSD can become, however, insensitive to local deviations. Note also that its quality depends on the choice of features it was computed for and to the used reference structure. It can furthermore contain a large projection error since it only measures absolute deviations and not their direction.

Features can be relatively simple properties or measures of a molecule, for instance also (torsion) angles. There is, however, no limit in how complex features or rather their derivation can be. Examples for less straightforward features are the solvent accessibility surface area (SASA) of a protein,[195] or the pocket volume of a binding-site.[196] In principle, any computed quantity like $\mathrm{p}K_\mathrm{a}$-values, energies, or let's say dipole moments, can also serve as a feature.

Besides continuous features, there are many commonly used categorial features as well (see also section 13.1 for a short discussion on this type of data). An example are DSSP classifications.[197, 198] Through a comparison of a protein structure with a database, the DSSP program can assign a secondary structure element (helix, sheet, loop, or a further specialised category) to each amino

DSSP

[3] In other contexts, also the RMSF of atoms can be considered a feature, though. It is just not a time series mapping the state of the system at a given point in time to a value.

acid residue. If this is done for time frames from a MD simulation, the evolution of the assignments can be used as an indicator for the protein's stability, either as a basic sanity check or to spot large conformational transitions that involve secondary structure changes.

H-bonds

Another context where categorial data can be encountered, are features that denote the presence or absence of interactions, e.g. hydrogen bonds. While hydrogen bonds may be chemically transient in reality and have a theoretically rather complex nature,[199] they are for the sake of simplicity in an analysis often converted into binary information. The identification of these bonds mostly follows geometric criteria for the arrangement of donor, hydrogen, and acceptor atoms (D–H···A), although it has been argued that this can be flawed.[200] Using the Baker-Hubbard criterion, the presence of an H-bond can be assumed if the H···A distance is smaller than 2.5 Å and the D–H···A angle is larger than 120°.[201] An alternative definition used by GROMACS requires the D···A distance to be at most 3.5 Å long and the H–D···A angle to be at most 30° wide. VMD relies on the D···A distance below 3.0 Å and requires that 180° minus the D–H···A angle is smaller than 20°. Yet another criterion for hydrogen bonds between water molecules derived from calculated spectra uses $d(\mathrm{D}\cdots\mathrm{A}) < 3.3\,\text{Å} - 0.00044\,\theta(\mathrm{H}–\mathrm{D}\cdots\mathrm{A})^2$, accounting for the fact that the donor-acceptor distance can be larger when the hydrogen-donor-acceptor angle is close to 0°.[202] Once H-bond existences have been collected for each time frame in a simulation, relative populations and population differences can be evaluated but also for example correlations between bonds.

Other types of interactions (e.g. hydrophobic contacts or ionic interactions in say salt bridges) can be assessed in a similar way. An advanced treatment could use not only the binary existence information but also the geometric centres of where an interaction takes place or a similar spatial property. Such information is for example used in pharmacophore modelling,[203] or in the dynamic adaptation of the concept in terms of dynophores.[204]

These are only a few examples of what could be selected or extracted as features from MD trajectory data to achieve a reduced, comprehensible representation of a molecular system and its properties. For a modern perspective on this, be also referred to [205]. Their evaluation of features, on the other hand, is often done in more or less the same way, namely as mentioned either in terms of the time series directly or in terms of its distribution.

5.2.1 Time series and distributions

For the estimation of a distribution from a time series, a simple way is the binning of value ranges into a histogram. This requires to set a bin width, which can be chosen arbitrarily to a desired level of detail or according to a rule of thumb that depends on the number of samples in the series. The Freedman-Diaconis rule suggests for example a uniform bin width of $2\,\mathrm{iqr}(x)/\sqrt[3]{n}$ based on the interquartile range of the sample set and the number of samples n it contains.[206] There is further the possibility to report absolute counts (the number of samples per bin) or to normalise the result. A normalisation can be either done in terms of a discrete probability mass function (the sum of all bin values is equal to 1) by dividing each bin count by n or in approximation to a continuous probability density (the integral over all bins is equal to 1). For a comparison of features, either kind of normalisation should be employed if the number of samples differs. If the bin width differs as well, a density approximation is preferable.

Alternatively, a kernel density estimate (KDE) can be employed choosing a kernel and a bandwidth

$$\rho(y) = \sum_{s=1}^{n} K(y - x_s; h) \tag{5.6}$$

where the kernel K is a window function, producing a certain kind of shape (e.g. a Gaussian) centred around each sample point x_s. The local density estimate for points y is then just the sum of these per

point contributions. For large sample sets, the bandwidth has a bigger influence than the choice of kernel. A too small value will result in a noisy estimate, a too large value can over-smooth the data. For a good initial bandwidth value, there is for example the Silverman rule amounting to $0.9m/\sqrt[5]{n}$, with $m = \min(\sigma(x), \mathrm{iqr}(x)/1.349)$.[207] More elaborate schemes exist, though.[208]

Figure 5.2 shows examples for histograms and KDEs for a distance feature extracted from a MD simulation, illustrating the effect of bin width, bandwidth, and kernel. The tophat kernel used in figure 5.2g is just a block of a certain width, much resembling histograms, only that each sample contribution is centred around the point and not added to a respective bin. If later smoothing is needed, interpolations can be done (5.2d) but it should be noted that this can distort the result. For visualisation, either of the shown approaches (except maybe 5.2f) would be acceptable. For quantitative analyses, the small difference may matter, and for smaller data sets, they are likely to become more pronounced. Density estimates can also be used in the context of density-based clustering (see section 14.5 and the following). After all, what density-based clustering tries to achieve is a separation of dense data regions (visible as peaks in a feature distribution) into disjoint groups.

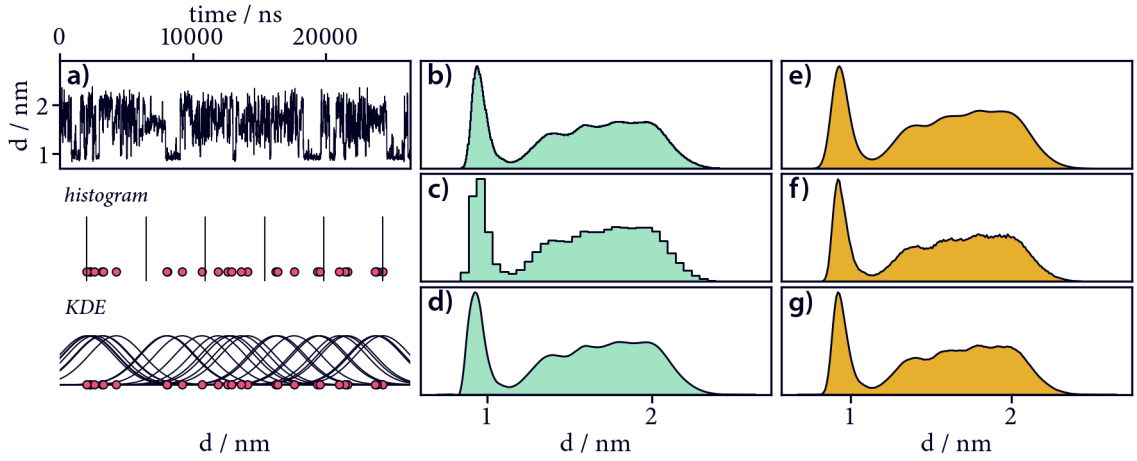


Figure 5.2 Histograms versus kernel density estimates

a) Time series for a single distance feature with about 20 million data points (2000 shown). Histograms using **b)** 442 bins (Freedman-Diaconis), **c)** 40 bins **d)** 40 bins plus quadratic interpolation (`scipy.interpolate.interp1d`). KDE (`sklearn.neighbors.KernelDensity`) using **e)** a Gaussian kernel with 0.03 bandwidth (Silverman), **f)** a Gaussian kernel with 0.003 bandwidth, **g)** a tophat kernel with 0.02 bandwidth (Freedman-Diaconis). The histograms use the full set of samples, while the KDEs use only every 100th point for the kernels and densities are estimated for 200 points in the data range. The diagrams in the lower left corner depict the binning of points into a histogram and the kernels for points that are summed up in KDE.

Histograms and KDEs generalise to higher dimensions but their applicability degrades relatively quickly. In the context of 2-dimensional representations, it is sometimes desired to convert an approximate probability density with respect to a given projection into a pseudo free energy surface. The following short sequence of operations can be useful for visualisation (assuming the density is stored as grid points in a 2d NumPy array `P`):

```
P = -np.log(P)  # ignore divide by zero
P -= np.min(P)
P[P > threshold] = threshold
```

This can then be plotted as a heatmap or contour-plot in which the lowest energy (highest probability state) is set to 0, energies are given in units of $k_\mathrm{B}T$, and with a finite maximum value (see figure 5.9).

The comparison of two feature distributions can be done for example in terms of absolute differences (subtracting one probability density from the other) or in terms of relative changes (dividing one by

the other), at least if the same feature is compared over different setups, simulations, or systems. For entirely different features, i.e. two basically unrelated features in the same system, this might, however, be meaningless. Measures to quantify the difference or pairwise relationship between two probability densities are for example the Kullback-Leibler divergence,[209] or the mutual information (MI) score described in section 5.4.

pairwise correlation

We can also relate two features from the same trajectory by the means of correlations. Correlation coefficients are in general bounded to values between −1 and 1. Positive correlation can be usually interpreted as a high tendency of two features to adopt a similar value while in reverse negative correlation indicates opposite values at the same time. No correlation (0) can be associated with randomness. A popular example for a correlation measure is given by the Pearson coefficient

$$c_{\text{Pearson}} = \frac{\text{cov}(x,y)}{\sigma_x \sigma_y} = \frac{\sum_{s=0}^{n_s}(x_s - \mu(x))(y_s - \mu(y))}{\sqrt{\sum_{s=0}^{n_s}(x_s - \mu(x))^2}\sqrt{\sum_{s=0}^{n_s}(y_s - \mu(y))^2}}, \tag{5.7}$$

as the covariance of two feature variables divided by the product of their standard deviations. The respective mean of each variable is denoted with μ. The coefficient can be computed for continuous or discrete features and measures *linear* correlation, i.e. how well a straight line can interpolate the relation $y = f(x)$. The sign of the correlation coefficient corresponds to the slope of a linear regression. In this sense a high absolute correlation can also be interpreted as how well the value of one feature can be predicted based on a value in the other feature.

An alternative that emphasises the notion of predictive potential is Matthew's coefficient for binary data (also called ϕ-coefficient). It is computed from the number of times both features show the same one value (referred to as *true positive*) n_{11}, the same other value (*true negative*) n_{00}, or a different value (*false positive* and *false negative*) n_{10} and n_{01} as

$$c_{\text{Matthew}} = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{(n_{11}+n_{10})(n_{11}+n_{01})(n_{00}+n_{10})(n_{00}+n_{01})}}. \tag{5.8}$$

Matthew's coefficient can be extended for multi-label categorial data.[210] Further, there exists a large portfolio of other methods to measure (also non-linear) correlations.[211]

autocorrelation

For the analysis of time series (where the order of the values of a feature really matters), we have basically one closely connected option, which is the assessment of the autocorrelation. Given a series of values, this type of correlation measures how much a value at time t influences possible values at time $t + \tau$. For the time discrete case we can write

$$c_{\text{auto}}(k) = \frac{1}{n_s - k}\sum_{s}^{n_s-k}(x_s - \mu(x))(x_{s+k} - \mu(x)) \tag{5.9}$$

where x_s is a feature value at time $t = t_s$ and k is the lag time τ in steps. For $\tau = 0$, we basically obtain the variance of the data by which the autocorrelation can be normalised to the value interval $[-1, 1]$. Formally, autocorrelation can also be described as a convolution. Figure 5.3 shows a simple example.

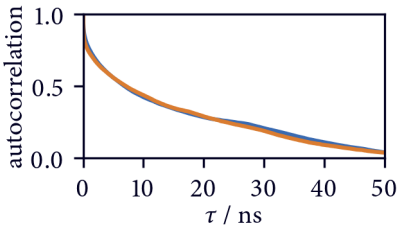


Figure 5.3 Autocorrelation For a single distance feature in two independent systems, the autocorrelation has been computed for lag times up to 50 ns for comparison. The decay indicates quick relaxation on the nanosecond time scale. The two system exhibit the exact same behaviour.

Conceptually similar to autocorrelation, are analyses targeting towards the live-time of certain states, e.g. the residence time of a water molecule in a protein cavity or the persistency of hydrogen-

bonded interaction. In principle, the whole point of kinetic Markov models is as well to identify conformational states with high meta-stability, i.e. significant autocorrelation.

5.2.2 Bootstrapping

Considering a single set of samples, one can compute statistical quantities like the mean and standard deviation. In the context of limited sampling, it might be desired, though, to get an estimate on how valid a sample set is and how reliable an inference of the true underlying population would be from it. A way to do this, is referred to as bootstrapping.[212] It essentially works like this: by generating new sample sets from the set in question using random draws with replacement, one can compare quantities computed from the new samples to assess how robust these observations are. A sample set that exhibits low variation under resampling is likely to allow stronger inference on the real population.

In our analysis, we use bootstrapping for example to compute confidence intervals on feature histograms. Our sample set is in this sense a list of histograms for the same feature estimated on individual simulation replica (blocks of the complete trajectory). This set of histograms gives us one weight-averaged histogram for the full data. In a bootstrapping round, we resample with replacement from our set of histograms to get a new set (of the same length) from wich we can compute a different average histogram. This process is repeated many (e.g. 1000) times and we can evaluate the distribution of averages. Typically, we report the 95 % confidence interval, which is the value range that contains 95 % of the averages for each histogram bin. For reference, this is how this bootstrapping can be implemented in Python using NumPy and scikit-learn. The function expects a list of histograms as the `base` argument and returns a confidence interval on the mean:

```
def bootstrap(base, n=1000, weights=None, **resample_kwargs):
  means = numpy.array([
    numpy.average(
      sklearn.utils.resample(base, **resample_kwargs),
      weights=weights, axis=0
      )
    for _ in range(n)
    ])
  return numpy.array([
    numpy.percentile(means, 2.5, axis=0),
    numpy.percentile(means, 97.5, axis=0)
    ])
```

An alternative strategy could try to resample directly from the full feature trajectory for a comparison of histograms. In this case one would probably need to tune the size of the generated samples if the trajectory is very large. Bootstrapping has many other applications in the context of MD.[155]

5.3 Dimensionality reduction

Rationally guided feature selection, which the basic features in the last section are in principle examples for, is an integral part of the exploration related to a MD data set. It is grounded in assumptions one makes about what kind of features are important to reflect conformational transitions in a molecule, either from system specific knowledge or by making an educated guess. As some of these features, like backbone dihedral angles, tend to be relevant for many different systems, this may not be a bad guess. Dimensionality reduction methods, on the other hand, take a set of input features and find a new set of output features that can be ranked in accordance with a criterion. Technically, also manual feature picking is done to the effect of reducing the dimensionality but the emphasis of

the use of dedicated methods for it, lies in the quantifiability of the selection process. As mentioned earlier, however, such a dimensionality reduction can be supported by limiting the number of input features beforehand based on a rational.

PCA

A classic method usable for dimensionality reduction is found in principal component analysis (PCA). The first principal component (PC) of a set of points (represented in the input feature space) is aligned with the axis along which the data shows the largest variance. In other words, it is a feature that explains or contains the largest part of the overall variance when the data is projected onto it. The second PC is a feature linearly uncorrelated, i.e. orthogonal, to the first PC that explains the largest part of the variance not captured by the first PC. For n_f input features there will be also n_f output features with maximised and decreasing associated variance. If large variance is considered more important than low variance, a projection onto only a subset of the output features can reduce the dimensionality of the data without loosing important information. For molecular systems this can be justified because large amplitude conformational changes can correlate with significant functional transitions.

Practically, PCs can be found by constructing the covariance matrix from the mean-free (or optionally standardised, see also equation 13.9) input data. The eigenvectors of this matrix are the PCs while their corresponding eigenvalues are a measure for how much variance is explained by them, i.e. how much information is preserved when the input data is projected onto them. The elements of the eigenvectors also measure how much the original features contribute to it.

example data

Let's consider an example to illustrate this. Figure 5.4 shows a six dimensional data set comprising three Gaussian distributions of which we could imagine that the data points stem from a MD simulation. For visual clarity, the three distributions are highlighted in different colours but in a real scenario these are not normally known beforehand. In fact, it rather would be the purpose of the feature selection to find a suitable representation to identify these as separable conformational states. We see already for this simple example that the view on the data changes considerably with the features through which we look at it. Here it may be possible to find a good set of features among the six dimensions but a PCA offers a more systematic way to achieve this, which does also not immediately break down if we need to consider hundreds of possible features.

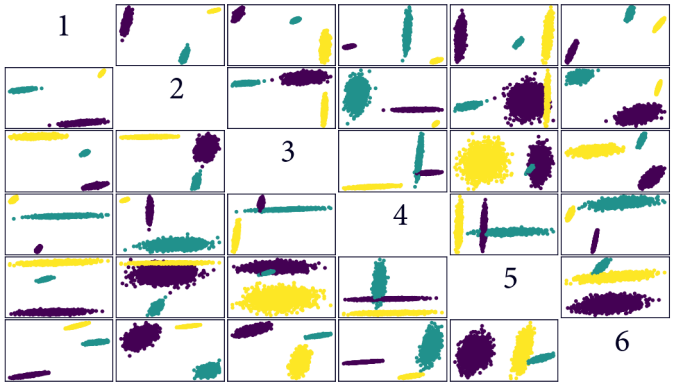


Figure 5.4 6-dimensional example data set of multivariate Gaussian states For each pair of dimensions, a 2-dimensional projection is shown in respective boxes on the left. The individual distributions the data was constructed from, are distinguished by different colours. In a conformational analysis of a MD data set, it could be the goal to find exactly these state assignments by the help of a suitable projection. Which would be the most faithful one in retaining the information in the input data?

Figure 5.5 shows the result of a PCA on the data above. By looking at the eigenvalue spectrum, it becomes clear that the first two PCs are probably sufficient to describe the data well. Combined, they cover over 90 % of the overall variance and the following components add little information. The elements of the eigenvectors indicate that of the original features the first is contributing most to the second, and the second most to the first PC. The fifth and sixth input dimension are rather unimportant. In the projection of the original data onto the first two PCs, the distributions are well separated.

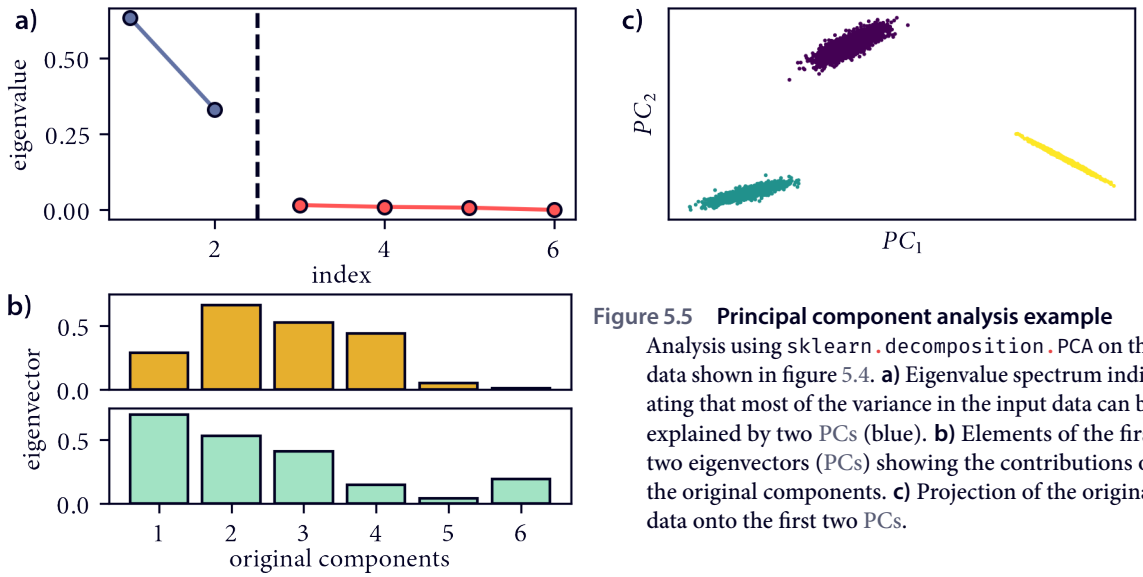


Figure 5.5 Principal component analysis example
Analysis using `sklearn.decomposition.PCA` on the data shown in figure 5.4. **a)** Eigenvalue spectrum indicating that most of the variance in the input data can be explained by two PCs (blue). **b)** Elements of the first two eigenvectors (PCs) showing the contributions of the original components. **c)** Projection of the original data onto the first two PCs.

For biomolecular systems, a PCA can be for example carried out on cartesian input coordinates (maybe filtered to only backbone or C_α atoms) or on torsion angles. In the latter case the input features should be converted into sine and cosine parts.[213, 214]

A similar dimensionality reduction technique achieving a feature transformation that can be even more appropriate in a MD context is time-lagged/time-structure based independent component analysis (*t*ICA).[215–217] It is in particular useful if the assumption underlying an application of PCA that large variance can be considered highly important does not hold for a given system. Functionally significant conformational transitions in molecules can be also rather small. It might be instead a better assumption that important conformational changes are in general rare, that is collectively slow and not often observed in a simulation. Independent components (ICs) constitute a set of features that aim on representing basically exactly this. By solving a similar eigenvalue problem as in PCA considering additionally a time-lagged covariance matrix, one can obtain these components. Figure 5.6 illustrates the result of this for the same data set as before in terms of a projection onto the first two ICs. This time, even one component may have been sufficient because it covers already 98 % of what can be considered the *kinetic* variance of the data in terms of maximised autocorrelation. Note that this type of analysis only works here because the data was generated in a time correlated manner, faking transitions with unequal probabilities between the states (marked in figure 5.6b). There are relatively few transitions back and forth between the purple and the green state, none between purple and yellow, and many between yellow and green. The first IC is well aligned with the axis along which rare transitions take place, that is where autocorrelation is high. In the picture, the components are scaled to reflect kinetic distances, so that states that are farer away from each other are also involved in fewer transitions.[218]

tICA

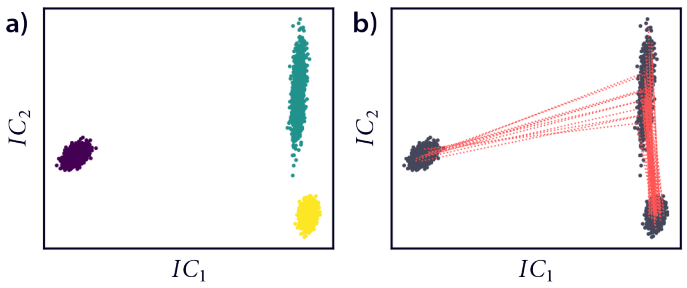


Figure 5.6 Time-lagged independent component analysis example
Analysis on the input data in figure 5.4 using `pyemma.coordinates.tica`. **a)** Projection onto the first two ICs. Note the relative difference to figure 5.5c. **b)** For time-autocorrelated data, the analysis identifies components of rare (slow) spatial transitions.

lag time

The success of this analysis for ICs depends on the choice of the input features and on the specified lag time. For both there is no general recipe but there has been proposed a heuristic based on VAMP scores.[219] Roughly, the strategy is to select input data with a relatively high score for a given lag time, and then to choose a lag time for which high scores are achieved while convergence is reached with the inclusion of a sensibly low number of dimensions.[4] In general, the obtained projections can be complex, however, and can change significantly and abruptly with the lag time, as illustratively shown recently for a series of random walks.[220]

MSMs

The found ICs have been shown to be approximations to the eigenvectors of the transfer operator underlying dynamical molecular processes.[216] Thus a corresponding projection can be an ideal basis for the estimation of a Markov model (see chapter 6). *t*ICA is a linear transformation method but has been extended to a non-linear kernel variant that claims to alleviate the need for the estimation of a Markov model on top of a *t*ICA projection.[221]

Another dimensionality reduction technique that has become very popular lately is *t-distributed stochastic neighbour embedding* (*t*SNE).[222] It has been also recently adopted for the analysis of MD data.[223]

5.4 Mutual information

information content and surprise

Entropy from an information theoretical perspective, is a measure for the expected *information content* of a probabilistic event. Alternatively, the notion of expected *surprise* can be used, both meaning the logarithm of inverse probability. Considering the probability distribution $p(x_i)$ of a discrete random variable x with possible outcomes x_i, entropy can be expressed in terms of the following equation:

$$S(x) = -\sum_i p(x_i) \log_b p(x_i)\,. \tag{5.10}$$

coin flip

This is known as the Shannon entropy.[224] The unit of measurement depends on the logarithm used in here, typically $\log_2$ (bits), $\log_e$ (nats), or $\log_{10}$ (dits). Let's examine the entropy of a simple example: a coin flip with probabilities $p_1 \in [0,1]$ and $p_2 = 1 - p_1$ for a single event leading to the coin landing on either side (see figure 5.7). Note the short-hand writing of $p(x_1)$ as p_1 here. Depending on p_1, the result of a flip can be more or less *surprising*. For the extreme case of $p_1 = 0$ or $p_1 = 1$, the coin will always land on the same side, meaning the result is absolutely certain and the *information gain* by repeating the flip is minimal. To be a bit more precise, the information content of the outcome x_1 for $p_1 = 1$ is 0 and it is undefined for $p_1 = 0$. In the limit of $p_1 \to 0$, the surprise associated with x_1 will approach infinity. On the other hand, if $p_1 = 0.5$, the outcome is as uncertain as possible and in this case the entropy is maximised.

[4] For an explanation in the context of the analysis of a small peptide see `emma-project.org/latest/tutorials/notebooks/00-pentapeptide-showcase.html`

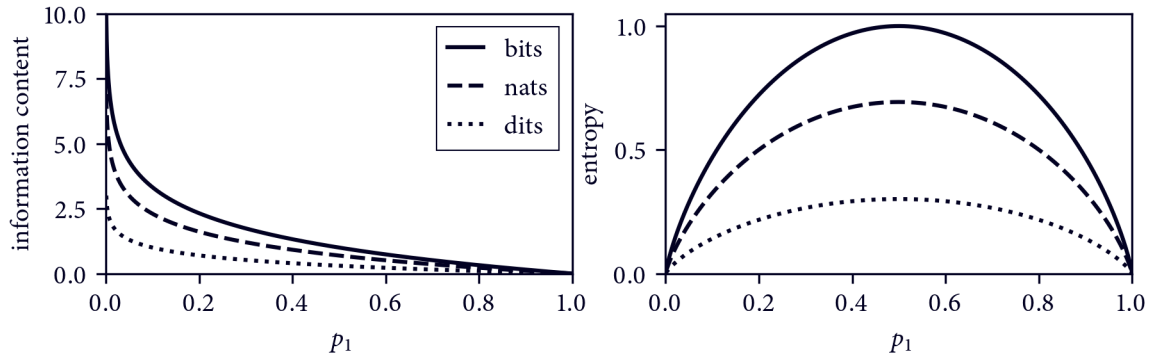


Figure 5.7 Entropy coin flip example
The information content of a coin flip event $-\log_b(p_1)$ decreases with its probability (left). The gain in information by repeating the experiment (the expected information content of a single event), i.e. the entropy according to equation 5.10, depends on the probabilities of all possible results (right). It is maximised when the outcome of the experiment is maximally uncertain, that is when all possible outcomes are equally probable.

interpretation

In a loose interpretation, we can say that the repetition of a probabilistic experiment is more interesting when the entropy is high because all possible outcomes tend to be equally probable and the experiment reveals more information per repetition. The entropy of data is a measure for the information content per bit and is used in lossless data compression.[225] This information theoretical concept of entropy translates directly to the Gibbs entropy used in thermodynamic chemistry to describe the statistically favoured state of a molecular system as the most likely one (the one that maximises the entropy) in the absence of energetic influences.[226]

For the consideration of two random variables x and y, one can define the conditional entropy

$$S_{xy}(x, y) = -\sum_i \sum_j p_{xy}(x_i, y_j) \log \frac{p_{xy}(x_i, y_j)}{p_y(y_j)}, \tag{5.11}$$

conditional entropy

in which p_{xy} is the joint probability of an event with respect to the discrete outcomes x_i and y_j. This non-symmetrical measure estimates how much uncertainty in y remains if x is known. Figure 5.8 exemplifies this with two subsequently tossed coins that are either uncorrelated or where the second toss is magically influenced by the first one. The conditional entropy can be taken as an estimate of how much a random variable depends on another one.

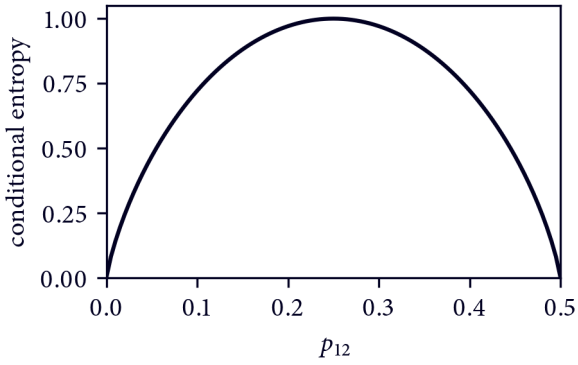


Figure 5.8 Conditional entropy coin flip example Given the probability that when the first coin lands on one side, the second coin lands on the respectively other side p_{12}, and with $p_{21} = p_{12}$ and $p_{11} = p_{22} = 0.5 - p_{12}$, how much uncertainty is left in the second coin toss? The conditional entropy (here in bits) is maximal if both coins are completely independent of each other. It is minimal for perfectly synchronised coins, i.e. when the second coin always lands on the same ($p_{12} = 0$) or the other side ($p_{12} = 0.5$). The marginal probabilities that the second coin lands on either side are equal. Note the writing of $p_{xy}(x_1, y_2)$ as p_{12} here.

A related measure that estimates symmetrically how much two random variables depend on each other, is the MI

$$MI(x, y) = \sum_i \sum_j p_{xy}(x_i, y_j) \log \frac{p_{xy}(x_i, y_j)}{p_x(x_i) p_y(y_j)}. \tag{5.12}$$

mutual information

Note the use of the product of the marginal distributions p_x and p_y in the denominator of the logarithmic ratio here. In the analysis of MD data, this estimate can be taken to relate pairs of individual features to each other. The score can be interpreted as a measure for how much information content about a another feature is entailed by a feature. It measures to what extend the features are correlated and can be used for example to reveal *communications* between amino acid residues in protein networks. In this context, MI scores are usually normalised for instance by the minimum or maximum entropy of the considered features to allow a better comparison between the score for different feature pairs:

$$\widetilde{MI} = \frac{MI(x,y)}{\min(S(x),S(y))} \,. \tag{5.13}$$

Figure 5.9 illustrates the joint consideration of two simulated protein features. It is not untypical to look at the torsion angles in such a system as it is often expected that relevant conformational information is well captured by these internal degrees of freedom. In general, however, a MI analysis is not restricted to a specific kind of features. The shown distribution reveals some degree of dependency between the two features, which will be captured by a non-zero MI value.

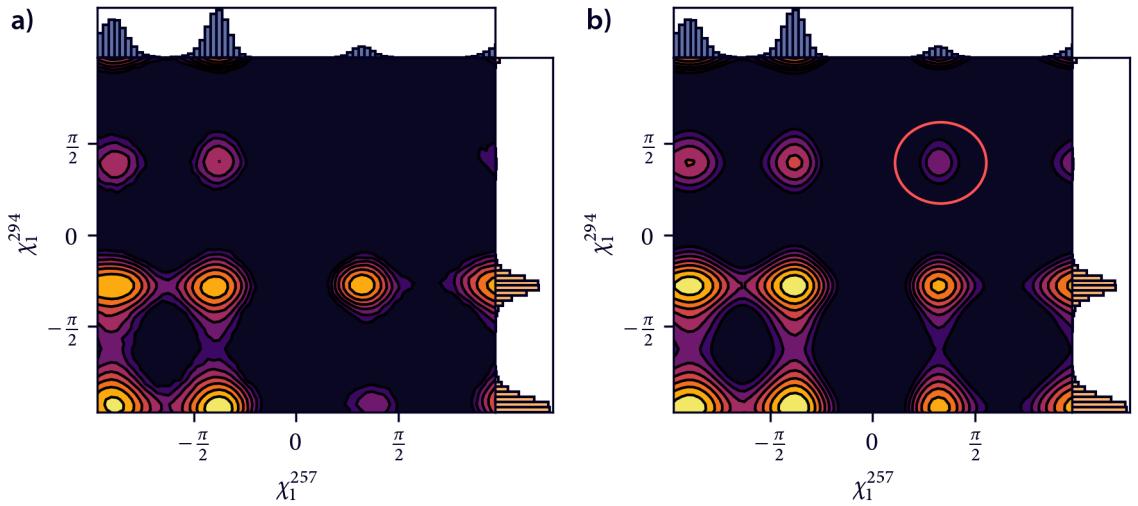


Figure 5.9 Mutual information protein example

Considering two side chain torsion angles of two amino acids in a protein, extracted from a MD trajectory, **a)** shows the joint probability distribution with respect to these features as a 2-dimensional contour plot and the marginal distributions as 1-dimensional histograms on the right and upper sides. In **b)** the 2-dimensional outer product of the marginal distributions is shown for comparison. The features exhibit some degree of mutual dependency because the joint and the product distribution differ, for example in the area highlighted with a red circle. The highlighted combination of marginal states is not observed in the simulation but would be if the two features were completely independent. The normalised MI score amounts to $\widetilde{MI} = 0.049$ (see equation 5.13). Whether this is significant has to be evaluated in the context of the application and other mutual dependencies in the system.

protein networks

An interpretation of individual, absolute MI values can be difficult, even when they are normalised. A productive analysis could for example consider the full set of torsion angles in a protein to identify significant dependencies by asserting feature pairs in a common setting relative to each other. The obtained information of pairwise correlations between n features can be represented in terms of a square MI matrix with n^2 elements or in a graph structure. Both can be evaluated in multiple ways, among other things by spectral clustering.[227]

It is common practice to filter the result using a significance threshold. Due to finite sampling, even completely uncorrelated features may still exhibit non-zero MI scores.[228] The amount of this effect can be for example estimated by re-sampling from the considered feature distributions.[229]

The excess MI found between hence uncorrelated features can be subtracted from the uncorrected MI values, either individually or in terms of a mean value determined from all feature pairs and multiple sampling rounds. Instead of such a rigorous re-sampling, a simple re-sorting of the original feature trajectories could be done for a rough assertion as well. Alternatively, a low arbitrary threshold value may be set based on the given data in order to reduce the number of considered MI pairs and to make the result comprehensible. In the analysis presented later in section 11.1, the largest eigenvalue of the MI matrix M was removed to the same effect by using

$$M'_{ij} = M_{ij} - \frac{\left(\sum_i M_{ij} \cdot \sum_j M_{ij}\right)}{\sum M} \,. \tag{5.14}$$

It is furthermore possible, to condense a MI matrix from its feature-wise form to a residue-wise form. This is handy because a detailed evaluation of all feature correlations can become complex and a reduction to a perspective on which residues influence each other, without the breakdown into individual feature contributions, may be easier to grasp. This can be achieved using a projection matrix that indicates which features should be combined into which residue. A suitable projection matrix has n_{residues} rows and n_{features} columns, where an element takes the value 1 if a residue contains a respective feature and the value 0 otherwise, e.g.

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{5.15}$$

assuming a system of three residues and six features where the first residue comprises one, the second two, and the third three features. The projection of the MI matrix is done as $PMP^\top$. One has to be aware, however, about the fact that such a projection effectively summing up all per-residue contributions can distort the overall picture when the number of features varies substantially among the residues. Residue pairs involving more features may appear as stronger correlated than pairs with only few features that still can be of high importance after all. It can be for example considered to normalise the per-residue score by the number of features that went into it or to just take the highest observed score for each residue pair. A further point of consideration concerns which features to compare exactly in the first place. For protein torsion angles, it is for example possible to treat the backbone dihedrals as one feature (as a Ramachandran projection) and to correlate these to each side chain torsion separately. Ideally, the effects of a variation in this regard on the output should be considered.

comments

As a final remark, pairwise mutual information calculations can become fairly expensive for long trajectories (millions of data points) if many features (a few hundred) are considered. In the context of the analysis presented in section 11.1, the needed functionality has been implemented in Cython for the sake of efficiency while maintaining an accessible Python interface to be used in interactive workflows.[5]

software

[5] Visit `github.com/janjoswig/shiny-md-collection/tree/main/shiny/procedures/mutual_information` for an example notebook containing the implementation

{ 6 }

Markov models

Pseudo-random jump processes to describe molecular kinetics

Bernoulli: law of large numbers

In his milestone work *Ars conjectandi*[230], Jakob Bernoulli (1654–1705) proofed the *law of large numbers* for independent random variables. It says that in experiments where the result of a single repetition of the experiment is determined in terms of a probability, the average of the results from n repetitions should converge to the expected value when n growth towards infinity. It is required, though, that the probability distribution underlying the outcome of the experiment indeed has a finite expectation value (take a Cauchy or a Pareto distribution with $\alpha \leq 1$ as counter examples). Figure 6.1 illustrates the significance of Bernoulli's law with the classic example of differently coloured balls in a jar, the ratio of which can be determined by a series of drawing experiments.

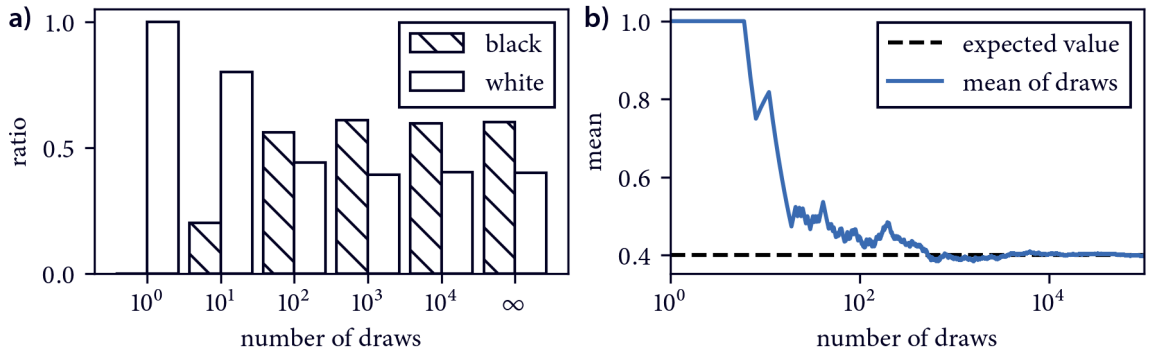


Figure 6.1 Empirical illustration of the law of large numbers
Consider the following simple system: a non-transparent jar is filled with black (value 0) and white (value 1) balls in the ratio 3:2 (see also figure 6.2a for an illustration). Suppose this ratio is unknown, it can be determined by repeatedly drawing a ball out of the jar, recording its color and putting it back. **a)** In the limit of infinite draws, the ratio of recorded results converges towards the true ratio of balls in the jar. **b)** Alternative representation in terms of the mean of results.

Markov

Two centuries later, Andrei Markov (1856-1922) showed in another game-changing work that the law of large numbers also holds for pseudo-random variables,[1] i.e. for those experiments in which the result of an individual repetition depends on the outcome of the last repetition (or an initial condition).[231] Figure 6.2 illustrates this in analogy to the Bernoulli example with a simple system of two coupled jars on which repeated drawing experiments are carried out that depend on the previous repetition.[232] In the previous example we had two possible outcomes for a single event, i.e. two states we could find the system in with different probabilities. Now we have still two states but the probability of the next state changes with the state we come from.

[1]For an interesting account on Markov's live and the background of his study see this blog article: https://www.americanscientist.org/article/first-links-in-the-markov-chain

Figure 6.2 Markov's extension of the law of large numbers
a) Illustration of the Bernoulli jar system from figure 6.1. **b)** Consider the following simple 2-state system: a black and a white non-transparent jar are filled with black (value 0) and white (value 1) balls in the ratios 3:2 and 4:1. Supposed these ratios are unknown, they can be determined by drawing a ball out of one of the jars, recording its color, putting it back, and continuing with the next draw from the jar that has the same colour as the ball drawn last. As long as there is at least one black ball in the white jar and one white ball in the black jar, the ratios of recorded results converges to the true ratios in the limit of infinite draws. Moreover, one can determine the ratio of drawing from the black or the white jar, i.e. the ratio of black to white balls overall (which is 2:1).

transition probability

A process of repeated draws on a system like this constitutes a Markov chain, or more generally a Markov model. The trait that the only factor determining the next step in such a stochastic process is the current state of the system (not the previous history) can be referred to as *memoryless*. The relative probabilities in this system can be condensed into a *transition matrix*

$$T = \begin{pmatrix} 0.6 & 0.4 \\ 0.8 & 0.2 \end{pmatrix}, \tag{6.1}$$

where the elements T_{ij} denote the conditional probability of going to (drawing) state j starting from state i (of the last draw). The values in each matrix row should add up to 1. This matrix has interesting properties. For instance, its kth power gives us the probabilities of arriving in a certain state after k steps, given a current or initial state. Among other aspects, it is this predictive potential why Markov chains are in widespread use, ranging from metrology to finance.

example model

The first actual application of a Markov chain was by Markov himself, analysing an excerpt from Pushkin's Eugene Onegin, where each character represents a discrete state.[231] It can be demonstrated for example that the probability of obtaining a vowel after a consonant and vice versa is significantly larger than expected for a fully random, uncorrelated series of letters. Also in the broader context of text processing, Shannon explored his ideas on information entropy with a Markov model for the english language.[224] Quite similarly, we can build a Markov model for FASTA sequences of proteins. Figure 6.3 shows the result of this exercise for langerin (see also section 11) and related receptors. What can be learned from this is for example that the kind of amino acid built into the sequence at a certain position can be strongly influenced by the preceding acid. The highest correlation can be found for proline and glycine.

But we can do more with the estimated transition matrix. The first left eigenvector of this matrix does for instance correspond to the equilibrium probability distribution of states predicted by the model (figure 6.3c). It can be interpreted as an expected distribution of amino acids in other (related) sequences, or as a prediction for sequences of the same type. In this example the sampled distribution gives more or less the same. Depending on the context, also the higher eigenvectors can have an intuitive interpretation.

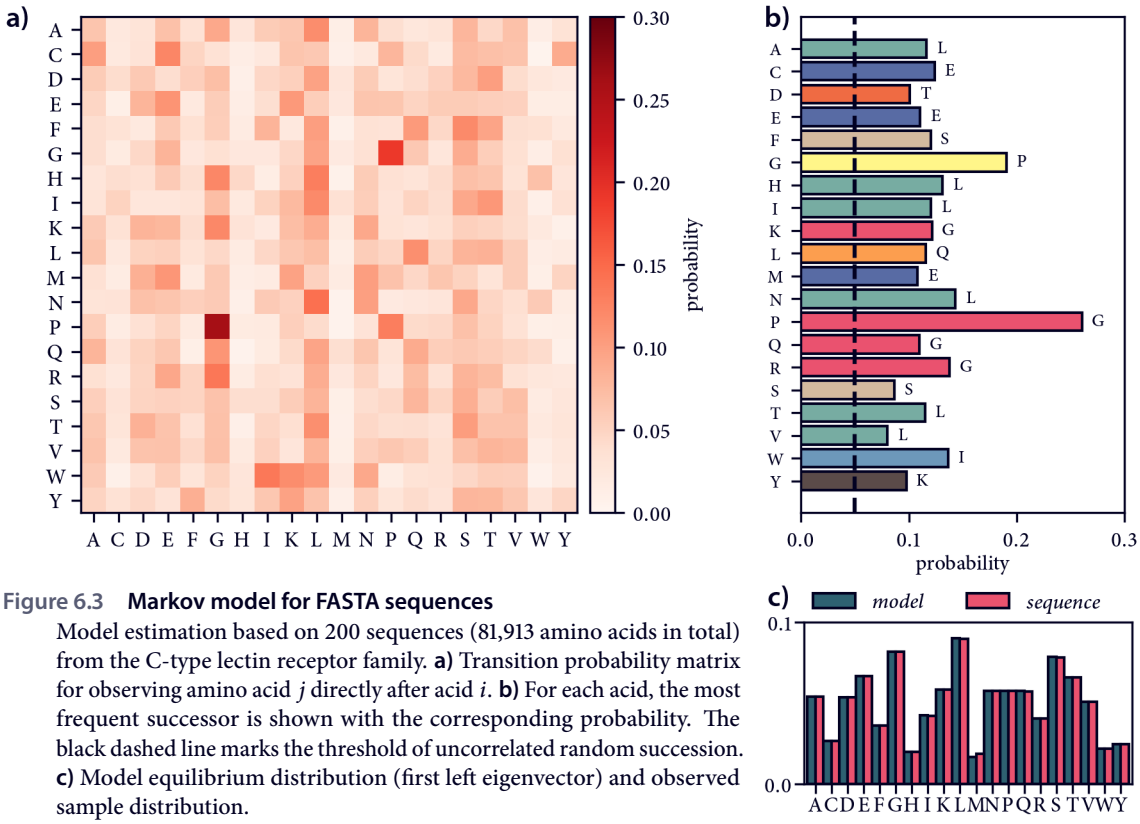


Figure 6.3 Markov model for FASTA sequences

Model estimation based on 200 sequences (81,913 amino acids in total) from the C-type lectin receptor family. **a)** Transition probability matrix for observing amino acid j directly after acid i. **b)** For each acid, the most frequent successor is shown with the corresponding probability. The black dashed line marks the threshold of uncorrelated random succession. **c)** Model equilibrium distribution (first left eigenvector) and observed sample distribution.

Markov models can also be used to describe conformational transitions in molecules and can be built from MD trajectory data.[47, 233] In this context, the models are often referred to as Markov-state models (MSMs), based on the discrete conformational states they comprise.[234]

In essence, a transition probability matrix approximates the continuous transfer operator of the true underlying dynamics of a molecular system. Let's assume a given probability density $\rho_t(\boldsymbol{q})$ on molecular configurations $\boldsymbol{q}$, different from the equilibrium distribution $\pi(\boldsymbol{q})$, at any given point in time t. Weighting this density by the stationary density as $\widetilde{\rho}(\boldsymbol{q}) = \rho(\boldsymbol{q})\pi(\boldsymbol{q})^{-1}$, its relaxation towards the stationary state can be described in terms of a time discretised transfer operator $\mathcal{T}(\tau)$, so that we have[235]

transfer operator

$$\widetilde{\rho}_{t+\tau}(\boldsymbol{q}) = \mathcal{T}(\tau)\widetilde{\rho}_t(\boldsymbol{q})\,. \tag{6.2}$$

By repeated acting of $\mathcal{T}$, $\widetilde{\rho}_t(\boldsymbol{q})$ will come closer to equilibrium by a certain time interval τ, while once it has reached it, it remains invariant on further transportation. Fast configurational transitions will lead to fast equilibrations of the respective regions of the density, while slow transitions take longer to approach equilibrium. The nice thing about this is that by studying the transfer operator (or rather approximating transition matrices) we can describe a molecular system in terms of global conformational processes connecting portions of the configurational space. The eigenvalues of a transition matrix can be associated with time scales on which a probability density relaxes to equilibrium with respect to a conformational process. This can be expressed via the relation

$$\lambda_i(\tau) = \exp(-\tau/t_{\mathrm{r},i})\,, \tag{6.3}$$

where λ_i denotes the ith eigenvalue and $t_{\mathrm{r},i}$ the respective relaxation time. As shown by the variational approach to conformational dynamics, these time scales can be taken as lower bounds allowing the

systematic improvement of models.[236, 237] The corresponding eigenvectors can be interpreted as indicators for which configurations (or which conformational states) are involved in each process. While the first eigenvector corresponds to the stationary target density, the following associate with relaxation processes of decreasing time scale. By focusing on the slow processes—on the premise that these might be the decisive ones for a contingent functionality of the molecule—while neglecting the faster processes as largely irrelevant, we can effectively reduce the complexity of the system greatly. Note the similarity to the dimensionality reduction methods discussed in section 5.3. While Markov models of the conformational dynamics of a molecule can be regarded as a dimensionality reduction technique themselves, a direct construction of them without prior filtering to a suitable projection can be rarely actually done. This is mainly because in practice there are quite a few technical challenges associated with the estimation of Markov models.[217]

discretisation

To begin with, a MD trajectory from which a Markov model should be constructed is (while being discretised in time) continuous in configurational space. The first step in the modelling process is therefore usually a discretisation of the configurational space into conformational states. A conventional Markov model relies on a finite number of states for which pairwise transition probabilities can be determined. The discretisation step involves a form of clustering and a large number of possible realisations exist for it (see also chapter V). The problem of achieving a good discretisation is all but trivial. Before it can be reasonably attempted, one needs a sufficiently low dimensional projection in which a clustering is feasible in the first place. The discussed dimensionality reduction method *t*ICA can help immensely with that not only by pre-conditioning the clustered space to resolve slow dynamic processes but also because distances can be identified here with kinetic distance (see section 5.3).

discretisation errors

With the aim of estimating a Markov model, we have a few strict demands on a discretisation of such a space into clusters and the most important one is probably that it needs to reflect the energetic structure underlying the clustered configurations. Precisely, we want to respect and resolve energy barriers. Configurational changes within clusters should be fast relative to transitions between clusters. In this sense, we want our clustering to be *kinetically relevant* although being based on geometric information.[238] In particular we want to avoid intrinsic barriers within the identified states. A violation of this aspect can result in two negative effects: on the one hand, it can make outgoing transitions from the state depend on from where exactly in the state the transition occurs—not only on the state as a whole. In other words, it would make a transition depend on from where the system entered the state, i.e. on the longer history of the stochastic process that should be modelled. This would stand in conflict to the basic assumption that the process is memoryless, which is needed to approximate it as a Markov chain in the first place. This behaviour can be described as *non-Markovian*. On the other hand, misplaced state boundaries can lead to what is called the recrossing problem.[239] Transitions into states via a pathway that does not involve the barrier it should be actually separated by, are kind of fake transitions that are too fast (occur too frequently) and can effectively overestimate the transition probability (or underestimate the required time scale). Another perspective on the discretisation is to think of the state indicator functions as basis functions from which the transfer operator eigenfunctions are constructed. A good discretisation is therefore one from which the construction can be done smoothly.

projection errors

Besides a bad discretisation, also the projection error that has been made when preparing the clustered space, may distort energetic barriers or prevent a clustering that resolves them adequately. In general, any form of coarse-graining prior to the discretisation (e.g. by neglecting solvent degrees of freedom when the solute is studied) can lead to a breaking of the Markov property.

microstates

The classic approach to arrive at a state discretisation that offers kinetic resolution, is to use a very fine clustering into a lot of so called microstates. This can be for example realised with a (simple regular) grid discretisation or by using a prototype-based clustering algorithm like k-means (see section 14.3). Here, transitions between neighbouring microstates can also be fast but the rational is

that a transition over barriers would need to cross a large number of states. The number of states one can actually use, is, however, not only balanced by the available resources (computer memory) but also by the sampling situation of the data. If the number of data points is low compared to the number of states, many states will be involved in only few transitions, leading to unfavourable statistics.

An alternative strategy that can suffice with a relatively low number of clusters and that avoids problems with unresolved barriers, is to use a clustering into so called *core-sets*. In general, a core-set is a set of data points that are a sure member of a respective cluster. By using for example a density-based clustering technique (see in particular chapter 15), one can identify states that correspond well with energetic minima, that is to a set of configurations representing the same conformation.[240] Configurations in a cluster found by density-based clustering are connected by relatively high density, while present energetic barriers (regions of low data point density) separate clusters, and can also be excluded from the state assignment entirely if desired. A basic assumption for a partial discretisation like this to work, is that the un-clustered noise or transition region is left quickly relative to the slow processes of interest.[241] Markov models on core-sets have to be treated somewhat differently than models on microstates, though, but in principle it is possible to use any discretisation for the estimation.[242] Another potential advantage over microstate models can be that, as a consequence of the lower cluster number and the natural correspondence of clusters with conformational states, the post analysis of the Markov eigenvectors is arguably easier. For models comprising thousands of states further (spectral) clustering techniques may need to be employed (see also section 14.2) to build actual meta-stable conformational states from the small conformational microstates. Density-based clustering furthermore allows to explore molecular free energy surfaces in a hierarchical fashion up to a desired level of detail in the resolved conformational processes and can in consequence be used for hierarchical MSM estimation.[243, 244]

With a suitable discretisation at hand, the next modelling step would be to count transitions between states for a chosen lag time τ. A standard method would be to use a sliding window for that, or to count only independent transitions without overlap.[238] Transition probabilities can then be obtained from the absolute counts by normalising each matrix row with the total number of outgoing transitions from the respective state. In modern practice, transition probabilities are, however, often obtained by maximum likelihood or Bayesian estimators.[245] Through iterative optimisation, transition matrices are found that reproduce the observed data best. Special considerations went into the estimation of reversible matrices (see equation 6.4 further below). Another major concern is the correction of models for non-equilibrium sampling situations and the quantification of the modelling error.[246] Core-set Markov models can be furthermore based on approximations of committor functions from milestoning.[242]

A central modelling decision is the choice of τ, which can be fairly subjective. Although MD trajectories are discrete in time, the analysed time interval is kind of arbitrary and depends on the time scale of the conformational processes that should be observed. The lag time is a lower limit for the fastest conformational transition process that can be resolved by the model. Its value does therefore indirectly tune the level of detail captured. To achieve a model that exhibits Markovian behaviour, τ usually needs to exceed a certain value. At small lag times, the requirement that the underlying process is memoryless may not be given with respect to the discretisation used but this may average out for larger values. In theory, the quality of the model should be independent of the lag time, i.e. the estimated relaxation times should be consistent when τ is varied, which can be tested by the abundantly used implied time scale plots.[233] A suitable lag time might be the Markov time, i.e. the smallest value above which the models show this expected behaviour.

lag time

A further important property of transition matrices is that they should fulfil *detailed balance* if they are estimated from equilibrium sampling. In equilibrium, there should be intuitively in the limit of infinite sampling always be a transition from state j to i if there is one from i to j. The molecular

detailed balance

process in question should be microreversible. i.e. each transition should be countered with its reverse transition. In particular there must not be any cyclic (periodic) behaviour. Matrices determined from finite sampling may not always satisfy this. Detailed balance can be enforced by averaging of the form $T' = (T + T^\top)/2$, resulting in a symmetric matrix. Symmetry is not a necessary requirement, though. In general, the equation

$$\pi_i T_{ij} = \pi_j T_{ji} \tag{6.4}$$

should hold, that is the probability of state i in equilibrium times the transition probability $i \to j$ should equal the probability of j times the transition probability $j \to i$. Detailed balance is not always needed for transition matrices in other contexts. For the FASTA showcase above (figure 6.3), there is for instance no reason to believe why a P→G transition should imply a G→P transition at any point.

connectedness

Finally, a Markov model is normally not allowed to contain disjoint subsets of states or states that can only be reached but not escaped. In our initial jar example, this meant that at least one ball of the other jar's color needed to be contained in each jar. For molecular simulations, this means that separate replica, sampling different regions of the state space, need to contain connecting transitions at some point. The set of connected states for which a model is estimated, is sometimes called the active set. Connectedness is the pre-requirement for ergodicity to be in principle achievable, namely that in the limit of infinite sampling every state is visited an infinite number of times. The system has a unique equilibrium distribution that is independent of initial conditions and the ergodic theorem (compare equation 5.3) holds. If connectedness is not fulfilled, one model can be estimated for each connected sub-set or it would be necessary to continue the sampling to eventually connect all states. A large influence in this regard has the technical realisation of the sampling. Besides effective seeding to promote an exhaustive exploration of the configurational space, also the choice of integrator or thermostat for example may enable or break ergodicity.[247] Enhanced sampling techniques can be used in this context as well but one should be aware that the dynamics of a system are distorted by the introduced bias. In some cases, re-weighting strategies may be applicable to correct for this.[79]

There is much room for exploratory freedom associated with the construction and evaluation of MSMs. Only a small portion of the broader topic has been addressed here but many variations exist. For two recent reviews on the topic, see [248] and [249].

{7}

Graph theory

About connectivity, hierarchies, and trees

In the sense in which they will be discussed here, *graphs* are a universal construct that is used for the study of all sorts of topological questions. Historically, a graph was perhaps employed more or less explicitly for the first time by Euler in the 18th century: the *seven bridges of Königsberg* problem became a landmark for the mathematical fields of graph theory and topology.[250] In short, the city that gave the problem its name comprised four major areas—two islands in the middle of a river and both of the river banks—between which one could travel via seven bridges. Euler wanted to answer the question if it is possible to visit all the areas by crossing each bridge exactly once. The fundamentally new about using a graph for this, was to recognise that only the relative connectivity between the areas matters and not the actual layout of the city as shown in figure 7.1. *history*

Today, graphs are used to formalise a multitude of problems where the connectivity between arbitrary objects is of importance. To name only one obvious example, chemical structures can be described from a graph theoretical point as inter-connected atoms without considering actual atomic positions, which was done already early on.[251] Essentially, also the topology of a molecule as a typical MD concept is nothing else than a graph of atom types connected by a set of interactions.

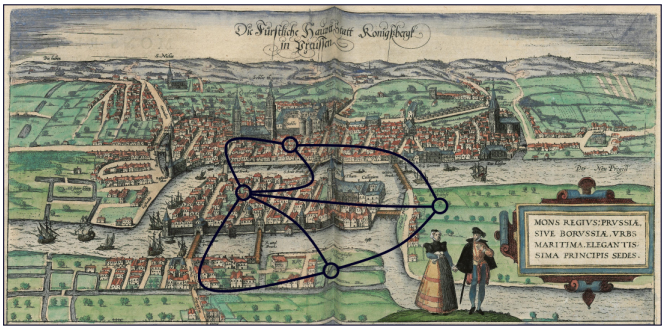


Figure 7.1 The seven bridges of Königsberg
A graph can be used to represent the city of Königsberg in an abstract way as a number of areas (black circles) connected by a number of bridges (black lines). Historic map from G. Braun, F. Hogenberg, *Civitates orbis terrarvm III* via Universitätsbibliothek Heidelberg.

A graph G is generally defined as the tuple (V, E, w) consisting of a finite set of vertices (also called nodes) $V = \{v_1, v_2, ...v_n\}$, a finite set of edges $E \subseteq \{(v_i, v_j) \,|\, v_i, v_j \in V\}$, and a weight function $w : E \to \mathbb{R}$.[252, 253] *formal definition*

Each vertex v in a graph represents a certain object, which can be virtually anything: a single point in a (metric) data space, a geometric object, a container of objects, or any other (abstract) entity one could think of—with the only limitation that individual vertices in a graph need to be distinguishable from each other with a unique identifier. A common identification of vertices does for example use integer indices, i.e. a consecutive enumeration of objects in the graph. A vertex can be equipped with an arbitrary number of properties or attributes. In some sense, there is no such thing as duplicate vertices: if each vertex represents a data point as a sample from some data space, each sample is a unique vertex although two vertices may have the exact same attribute values (e.g. coordinates in a *vertices*

metric space). Alternatively, each vertex could in this case represent a point in the data space itself while duplicate samples for the same point may be reflected by a count property on the vertex.

edges

Each edge e in a graph connects exactly two vertices v_i and v_j. Self-connecting edges, where $i = j$, are allowed as well. If an edge is denoted as the ordered tuple $e_{i,j} = (v_i, v_j)$, the graph is *directed*, that is there can exist an independent edge (v_j, v_i). The edges $\{(v_1, v_j)\}$ are called the outgoing edges of vertex v_1 while the edges $\{(v_j, v_1)\}$ are called its incoming edges. If $(v_i, v_j) = (v_j, v_i)$, the graph is *undirected* and individual edges can be denoted as the set $\{v_i, v_j\}$ without ordering. There is no differentiation between outgoing and incoming edges in undirected graphs. In some graphs, more than one edge is allowed for the same node pair in which case the graph can be called a *multigraph*. Similar to vertices, edges can be equipped with an arbitrary number of properties. In particular each edge can have a weight $w_{i,j} = w(e_{i,j})$ in which case the graph is called *weighted*. Edge weights are, however, optional and a graph can be also *unweighted*.

weight ambiguity

It is usually a choice of representation if a graph is explicitly constructed with unweighted edges so that edges are either contained in E or not, or if a graph is formally weighted but connecting (present) edges are all equal in weight (say $w = 1$) and non-connecting (absent) edges are (implicitly) equal in weight as well (say $w = 0$). In a weighted graph, it is on the other hand also possible to distinguish between absent edges and present zero-weight edges. It is also just a question of representation how edge weights are interpreted, that is a high weight can for example correspond to high relevance, short (i.e. strong) connection, or high gain but also to low relevance, far (i.e. weak) connection, or high cost.

vertex degree

The *degree* of a graph vertex is defined as the sum over all weights of edges connecting the vertex to other vertices. For directed graphs, it can be distinguished between the *in-degree* and *out-degree* only considering incoming or outgoing edges.

$$\deg(v_i) = \underbrace{\sum_j w_{i,j}}_{\text{outgoing}} + \underbrace{\sum_j w_{j,i}}_{\text{incoming}} \tag{7.1}$$

For unweighted graphs, the vertex degree is just the number of edges to other vertices, wich is equivalent to a situation where present edges have all weights of $w = 1$.

graph size

The *size* of a graph can be expressed as the number of vertices in V that is the *cardinality* of the vertex-set $\text{card}(V) = |V|$. Alternatively, it can be expressed as the *volume* of a graph that is the sum over all edge weights of edges involving vertices in the graph $\text{vol}(G) = \sum_{v \in V} \deg(v)$.

subgraphs

A graph $G' = (V', E')$ is a subgraph of G—conversely, G is a supergraph of G'—if the vertices in G' are also in G, that is $V' \subseteq V$, and if the edges in G' are also in G, that is $E' \subseteq E$. A subgraph is called an *induced* subgraph if all edges in E that are connecting vertices in V', are also in E'. In other words, edges of the supergraph that have both endpoints in the node set of a subgraph need to be inherited by the subgraph. See figure 7.2 for an illustration of these relations.

connectivity

A graph is *connected* if for any pair of nodes in the graph there is a *path* of edges from one node to the other, so that all the intermediate nodes are also in the graph. Connectivity between vertices can be denoted with $v_i \sim v_j$ based on if there is a connecting edge or a path of edges in between them. Indirect connection of nodes via intermediate nodes follows the argumentation that if $v_a \sim v_b$ and $v_b \sim v_c$, then $v_a \sim v_c$. A graph is called *complete* or *strongly connected* if for any pair of nodes in the graph there is a direct connection between them. A connected subgraph of G is called a *connected component*. If there is no edge in G that connects a node in a connected component G' to a node not in G' (i.e. an edge that could be added to G' to extend the component), then G' can be explicitly called a *maximally connected component*. When one refers to the connected components of a graph, it is usually implied that these are maximally connected, though. An induced subgraph of G that is complete can be furthermore called a *clique*.

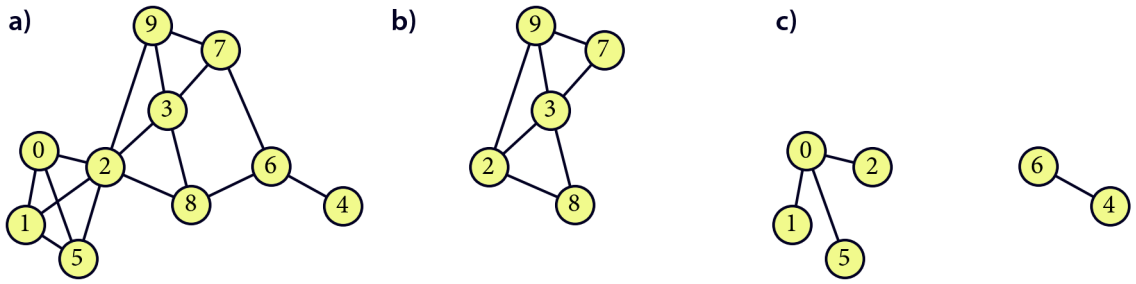


Figure 7.2 Examples of undirected, unweighted graphs
a) A graph G comprising ten nodes numbered successively from 0 to 9 and connected by several edges. Its largest clique is formed by nodes 0, 1, 2, and 5. **b)** An induced subgraph of G. **c)** A subgraph of G that is not connected and contains the two maximally connected components {0, 1, 2, 5} and {4, 6}. Note that the edges connecting nodes 1, 2, and 5 have been dropped so that the respective component becomes a tree without circular connections.

trees and forests

Depending on the overall connectivity in a graph, a few typical names are used to describe graphs of a certain kind. A graph is for example called a *tree* if there are no circular connections, in other words, if the graph is *acyclic*. Formally, if there is a an edge (v_a, v_b) and an edge (v_b, v_c), there must not be an edge (v_a, v_c). Another requirement for trees is that they need to be connected. A graph of more than one tree can be called a *forest*.

For directed graphs, additional categorisations can be made with regard to in- and out-degree of the graph nodes. Each node may for instance be only allowed to have at most one incoming edge.[1] If the number of outgoing edges, on the other hand, is limited to say two, four, or eight, a tree may qualify as a *binary* tree, *quadtree*, or *octree*.

hierarchies

An undirected tree can be converted into an implicitly directed tree by selecting one of its nodes as the *root*. The root is the starting point of the tree from which all other nodes are literally grown. In originally directed trees, there is an obvious choice for the root if the maximum in-degree of nodes is limited to 1. There is a *parent-child* relationship between the nodes, which makes a tree in general a hierarchical structure. The root of a tree can be found by following the present edges up to to the last parent. By the common definition, the root node has no incoming edges. It is, however, mostly for practical reasons also possible to set the root node formally as its own parent. Tree nodes that have no *children* can be referred to as *leafs*.

Any connected subgraph of a tree may be called a *branch* and does also qualify as a tree. Most importantly, branches of a tree can be recursively defined by setting any child node in the tree as the root of a new subtree. Sometimes, this kind of branch selection is figuratively described as cutting of a branch from the trunk of the tree. Figure 7.3 illustrates a typical tree structure.

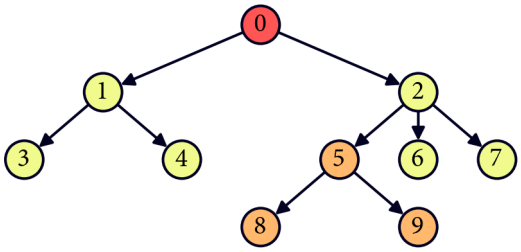


Figure 7.3 Example of a directed tree Graph of ten nodes with tree like connections and a maximum in-degree of 1. Node 0 (highlighted in red) is the ultimate root of the tree with node 1 and 2 as its two immediate children. Nodes 5, 8, and 9 (highlighted in orange) are an example of a possible branch in which 8 and 9 are leaf nodes.

[1] There exist multiple naming conventions: if a maximum in-degree of 1 is a basic requirement for a *tree* in the first place, a tree with a higher maximum in-degree can be called a *polytree*. Otherwise, a tree that additionally fulfils the requirement can be called an *arborescence*. A graph of more than one arborescence may be called a *branching*.

7.1 Connected component search

A STANDARD OPERATIONAL TASK IN THE CONTEXT OF GRAPHS, is the identification of connected components. Certain data clustering analyses (see chapter 14) for example can be formulated as a search for such connected components. Fundamentally, connected components can be explored one at a time by *traversing* the nodes in a graph along the present edges, beginning with a given root node. Graph traversal in turn can be realised by several classic algorithms. One of these is called *breadth-first-search (BFS)*: starting with a given root node, all the nodes of the same connected component will be traversed in a specific order, where 'breadth-first' refers to an exploration of nodes in shells or levels. Nodes with the same number of connections between themselves and the root node will be visited before nodes with a larger number of connections separating them from the root. This is in contrast to, say, a *depth-first-search (DFS)* traversal in which nodes with a larger number of connections to the root tend to be visited first.[253]

graph traversal

DFS is a recursive algorithm, whereas BFS is co-recursive. While DFS analytically breaks down the traversal of a graph into smaller and smaller pieces until it reaches a base case, BFS starts at the base case and synthetically constructs the whole traversal. Both can be also implemented iteratively.

Let's consider this taking the simple example tree in figure 7.3, referring to the tree as T. For a co-recursive BFS, one proceeds as follows: we pick a root node (also called source node) and add all nodes on the other end of its outgoing edges to a list of nodes that will be visited next. Then we process all these nodes and update the list of nodes explored on the next level with the nodes they are connected to. A working code example making use of a graph structure provided by the third party Python library `networkx` could look like this:

BFS co-recursive

```
def bfs_corecursive(graph, root):
  """Co-recursive BFS traversal passing on a parent node list"""
  def _inner_bfs(graph, parent_list):
    new_parent_list = []
    for parent in parent_list:
      yield parent
      for a, b in graph.out_edges(parent):
        new_parent_list.append(b)
    if new_parent_list:
      yield from _inner_bfs(graph, new_parent_list)
  parent_list = [root]
  yield from _inner_bfs(graph, parent_list)
```

Calling this function with 0 as the root node in T, yields the tree nodes in the order `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`. Starting with 0 as the first and only parent node on the first level, 1 and 2 will be the processed parents on the next level, and 3, 4, 5, 6, and 7 on the third level. The base case (yielding a parent node and adding its children to the next level parent list) will be repeated until there are no more child nodes on the next level. The co-recursive character of this is emphasized by the implementation of the search as a generator function. In contrast to normal recursion, the search is done 'bottom up' and we can lazily evaluate the currently processed parents while we go without having explored the full tree yet.[2]

An iterative implementation, typically uses a FIFO (first-in-first-out) queue as the fundamental data structure instead of the parent list that is passed on to further evaluation in the co-recursive example. We repeatedly add parent nodes that should be processed to the queue and take nodes off of it as long as it is not empty.

BFS iterative

[2]Note, however, that lazy evaluation is also possible for the recursive DFS discussed later.

```
def bfs_iterative(graph, root):
  """Iterative BFS using a queue"""
  q = deque()
  q.append(root)
  while q:
    root = q.popleft()
    yield root
    for a, b in graph.out_edges(root):
      q.append(b)
```

The result will be a traversal of T in the exact same order as before. I personally prefer the iterative solution as the one that is arguably easier to understand.

Note that due to the consideration of only outgoing edges and the fact that each node in the tree can only have one parent, we did not need to check during the traversal whether we visit the same node twice. In general, however, one needs to keep an indicator structure to check for this if the considered graph is not directed or if a node can have more than one parent. Typically, such an indicator can be a boolean array of length n for a graph of n nodes but can also be interpreted as a list (or set) of already 'visited', 'discovered' or still 'considered' nodes.

track visited nodes

Furthermore, it is possible to modify the graph traversal such that explored nodes (or respectively the connections they are reached through) are evaluated according to some criterion before the traversal is continued. Basically, this results in a conditional processing of further nodes so that effectively not the whole tree might be explored but only a certain sub-graph. For the implementation of the CommonNN clustering procedure (see section 15.2), this will be exploited. A modification of the iterative BFS above, using an indicator structure as well as conditional proceeding is shown below.

conditional traversal

```
def bfs_conditional(graph, root, condition):
  """Iterative BFS (modified)"""
  q = deque()
  visited = [0] * len(graph)
  q.append(root)
  visited[0] = 1
  while q:
    root = q.popleft()
    yield root
    for a, b in graph.out_edges(root):
      if visited[b]: continue
      if not condition(a, b): continue
      q.append(b)
      visited[b] = 1
```

Calling this with an admittedly silly condition function as `list(bfs_conditional(T, 0, lambda x, y: (x + y) < 10))` will result in the reduced output `[0, 1, 2, 3, 4, 5, 6, 7]`.

A RECURSIVE DFS can be build around the base case of yielding a graph node after a DFS traversal of all its child nodes:

DFS recursive

```
def dfs_recursive(graph, root):
  """Recursive DFS"""
  for a, b in graph.out_edges(root):
    yield from dfs_recursive(graph, b)
  yield root
```

This results in an output of the nodes in T in the order of `[3, 4, 1, 8, 9, 5, 6, 7, 2, 0]`. The deeper nodes are yielded before their parent nodes. Note that since this recursively continues on

smaller and smaller branches of the tree until a leaf node is reached, the full stack of function calls is build 'top down' before it es executed in reverse. Before the root node is being output, the whole tree has been explored.

DFS order

DFS is, however, flexibel in terms of the order it yields the traversed nodes. The shown implementation can be referred to as *post-order* DFS. It is possible to yield parent nodes before the continued traversal instead, i.e. in *pre-order*. The output of the function below results in `[0, 1, 3, 4, 2, 5, 8, 9, 6, 7]`.

```
def dfs_recursive_pre(graph, root):
  """Recursive DFS (pre-order)"""
  yield root
  for a, b in graph.out_edges(root):
    yield from dfs_recursive(graph, b)
```

Another possible variant (that makes only strict sense for binary trees) is to process the graph nodes in *in-order* by yielding a parent after its left branch has been explored and before the traversal is continued on the right branch (the other branches in our case).

```
def dfs_recursive_in(graph, root):
  """Recursive DFS (in-order)"""
  children = [b for a, b in graph.out_edges(root)]
  if children:
    left, *other = children
    yield from dfs_recursive_in(graph, left)
    yield root
    for b in other:
      yield from dfs_recursive_in(graph, b)
  else:
    yield root
```

The obtained order of this will be `[3, 1, 4, 0, 8, 5, 9, 2, 6, 7]` for T. Other processing orders are conceivable.

An iterative DFS implementation uses a stack, i.e. a LIFO (last-in-first-out) queue, much like the BFS used a FIFO queue. This led to the very common simplification that a DFS is just a BFS with a stack instead of a queue. Note that the code below yields the nodes in T as `[0, 2, 7, 6, 5, 9, 8, 1, 4, 3]`, which means it prioritises the rightmost branches but is otherwise identical to the recursive implementation. If the order of branch processing matters, nodes added to the stack need to be sorted accordingly.

```
def dfs_iterative(graph, root):
  """Iterative DFS (pre-order)"""
  stack = []
  stack.append(root)
  while stack:
    root = stack.pop()
    yield root
    for a, b in graph.out_edges(root):
      stack.append(b)
```

connected components

The shown traversal algorithms essentially explore the connected graph component of which the root node is a part of. When they should be used to find all the connected components in a graph, the exploration can be done successively. After one component has been fully discovered, a new root node can be picked for the exploration of the next component if there are still nodes in the graph that have not yet been visited.

7.2 Minimum spanning trees

A SPANNING TREE OF A CONNECTED GRAPH $G(V, E)$ is a tree that connects all vertices V in the graph with $n_E = n_V - 1$ edges, i.e. by using only a minimally required subset of E. As stated in the introductory section of chapter 7, a tree is by definition not allowed to contain circular connections. If G is disconnected, which means it contains disjoint subsets in V, there exist separate spanning trees for each of these subsets (connected components) and the set of trees can be called a minimum spanning forest.

A spanning tree is called a minimum spanning tree (MST), if the total edge weight of the edges in the tree is as small as possible. There may be more than one possible spanning tree per connected graph with the same total edge weight. If the edges of a graph are not weighted, all spanning trees are in this sense equal and therefore MSTs. Figure 7.4 gives an example for a connected graph and corresponding (minimum) spanning trees.

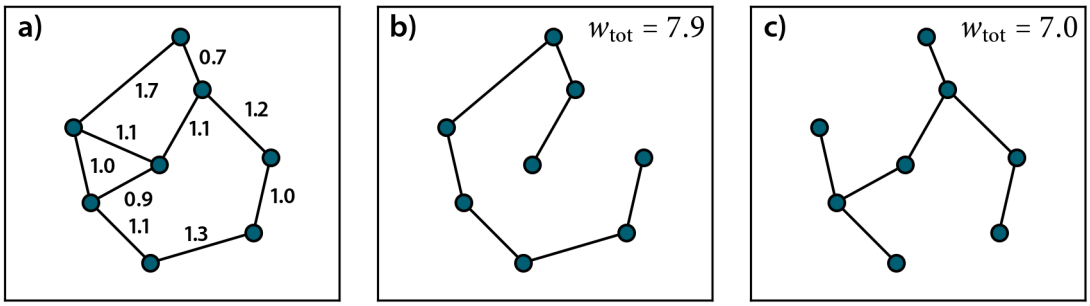


Figure 7.4 Minimum spanning tree example
a) Connected graph G with edges weighted by the distance between two vertices. **b)** A spanning tree of G with the at least required number of edges to connect all vertices. **c)** A MST of G with the smallest possible total edge weight.

MSTs find widespread application to solve optimisation problems. For example, such a tree can be the cheapest solution for a network of supply lines (water, electricity, internet etc.) between a couple of villages, where the villages are represented as vertices in a graph and the weight of the edges connecting them corresponds to a cost of building the respective supply line. For us, MSTs will become important in the context of clustering. The construction of a MST for given input data and a revision of the remaining edges in order of their weight is identical to what is referred to as single linkage clustering (see section 14.1).

There are several classic algorithms to find a MST of a given graph,[254] one of which should be shortly discussed in the following subsection.

7.2.1 Prim's algorithm

WHEN WE CONSTRUCT a MST T_{mst} for a given graph G following Prim's algorithm (alternatively called Jarník's algorithm), we proceed as follows: we start by selecting any vertex v_i in G as the root of the tree and add it to T_{mst}. Then we add all edges e_{ij} connecting the selected vertex to other vertices in G to a structure that keeps track of unchecked edges, i.e. edges we may want to follow to find the next vertices that can be added to T_{mst}. An appropriate realisation of this structure would be a priority queue that allows us to efficiently access the edge with the currently lowest weight. As long as the queue of edges to check is not empty, we pop the lowest weight edge e_{ij} off of the queue. If the vertex v_j that is reached by this edge is already in T_{mst}, the connection is redundant and we will continue by taking the next edge in the queue. Otherwise, the vertex v_j is added to T_{mst} and all edges e_{jk} from this

vertex to others in G are added to the queue of edges. Translated to (pseudo)code using the concrete example of Pythons `heapq` as a priority queue for unchecked edges, this could look like this:

```
# Start with arbitrary vertex (node) in G
spanning_tree.add_node(0)
queue = []
for eij, weight in get_edges(0):
  heapq.heappush(queue, (weight, eij))
while queue:
  weight, (i, j) = heapq.heappop(queue)
  if j in spanning_tree:
    continue
  spanning_tree.add_edge(i, j, weight=weight)
  for weight, ejk in get_edges(j):
    heapq.heappush(queue, (weight, ejk))
```

Prim's algorithm is called *greedy* because the tree is build by a sequence of locally optimal decisions as we always add the next vertex along the lowest weight edge—the currently cheapest or best connection—to the tree. Greedy algorithms are not in general guaranteed to find globally optimal solutions but Prim's algorithm does indeed always find a minimum spanning tree, which may not be unique, though. In the presented way, the algorithm finds a tree for the respective connected component, which the starting vertex is a member of. If G is disjoint, there are still vertices in G that are not in T_{mst} after the algorithm is finished and the procedure can be repeated with one of the remaining vertices as new root of another tree. There are different variations of Prim's algorithm, including an *eager* and a *lazy*.[254]

Part III.

Phallo- and amatoxins

{ 8 }

Fungal toxins

The death cap mushroom and its poisonous peptides

Phalloidin is one of seven closely related toxic heptapeptides, the phallotoxins, found in the death cap mushroom *amanita phalloides*.[255] Discovered and isolated in 1937,[256] it is also one of the first known bicyclic peptides.[257] Besides phallotoxins, *amanita* mushrooms contain an array of other toxic compounds among which are the very similar cyclic virotoxins and the bicyclic amatoxins composed of eight amino acids.[258] *overview*

The main clinical danger posed by these mushrooms when eaten is attributed to the amatoxins that are neither decomposed by the acidic environment and the proteases of the human stomach nor metabolised otherwise.[259] Their toxicological mechanism is rather complex and not yet fully understood in detail, which contributes to the fact that a viable antidote is still missing despite a number of therapeutic approaches.[260] It is, however, a central mechanistic element that amanitins are potent selective inhibitors of RNA polymerases, effectively preventing protein synthesis in affected cell. This happens predominantly in the liver.[261, 262] *toxicity*

Virotoxins and phallotoxins, on the other hand, are orally non-toxic but critical when inserted into the blood stream. They bind to filamentous F-actin of the cytoskeleton in hepatocytes and prevent the depolymerisation into monomeric G-actin, thereby disrupting the actin equilibrium and destroying the respective liver cells.[258] This is illustrated in figure 8.1 that shows phalloidin binding to F-actin bridging between G-actin monomers.

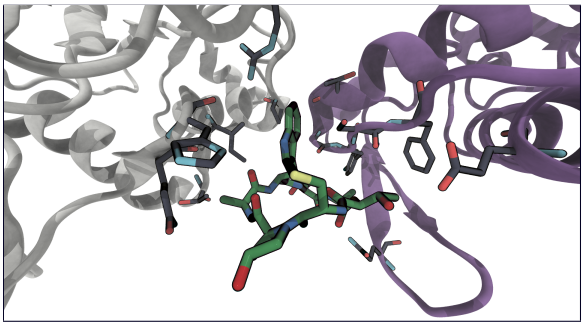


Figure 8.1 Phalloidin binding to F-actin
The phalloidin binding pocket is mainly located in between two actin monomers (white and purple protein backbone) and phalloidin (stick representation with green carbon atoms) interacts with several hydrophobic and charged residues on both of them (stick representation with black carbon atoms). Interestingly, phalloidin binding does not induce any actin conformational change.[263]

Because of its binding selectivity, phalloidin is used in fluorescently labelled form to identify F-actin in microscopy experiments.[264] This mainly serves as a way to highlight the cytoskeleton of the studied cells. In this context, its small size is an advantage over much larger antibodies often used for this kind of protein labelling. It allows for dense labelling, i.e. potentially higher resolution images, and less drastic structural and mechanistic perturbations to the marked target.[263] *F-actin*

Optimised protocols exist to extract cyclopeptides from *amanita* mushrooms.[265] Besides for the above mentioned practical application of phalloidin, these are for example exploited for toxicological studies like investigations of the extraordinary cyclopeptide tolerance of certain fly species.[265] Nonetheless, a robust synthetic route towards phalloidin and related compounds is very desirable, for one thing because these mushrooms can apparently not be cultivated.[258] Beyond that, synthetic access to these compounds is a key requirement for their systematic study under structural variation.

Peptides are in general becoming more and more interesting from a therapeutic viewpoint. With insulin being only the earliest and best known example, a broad array of peptide drugs has been approved worldwide (more than 30 since the year 2000), with various indications ranging from severe chronic pain to the treatment of osteoporosis.[266] Unlike small organic drug molecules, peptides are supposed to be well able to inhibit protein-protein interactions, among other things due to their larger size and conformational flexibility.[267]

In particular, cyclic peptides, like phalloidin, are in the focus because they are usually structurally well defined, balancing flexibility with potentially higher implied bioactivity.[268] Furthermore, they do not possess charged termini without the need for caps and often have promising transport properties, at least in comparison to their linear counterparts. The details, however, of which factors are decisive in this are still not very well understood. Ongoing research effort is for example targeted towards how structural modifications of these compounds affect membrane permeabilities.[269, 270] In general terms, an attention drawing matter of investigation are their structure-reactivity relationships.[271] Studies in the group related the membrane diffusion abilities of cyclosporines to their conformational adaptability in both polar and hydrophobic media.[272–274]

There are multiple examples for cyclic (especially bi- or tricyclic and stapled) peptide compounds that entered the market as approved drugs or are in clinical tries.[61, 268] Amanitin specifically might in conjugated form be used in oncology to target cancer cells when its toxicity for other host cells is oppressed.[260]

{ 9 }

The phalloidin project

Rationalising pathways in natural product synthesis

Several synthetic routes have been proposed for the preparation of phalloidin **1** (structural formula in figure 9.1) and its derivates. This endeavour is challenging for multiple reasons. For one thing, the compound features three non-canonical amino acids among its seven building blocks: D-threonine, *cis*-(4*S*)-4-hydroxyproline (Hyp) and (4*R*)-4,5-dihydroxyleucine (Loh). While the first two are readily available, enantioselective access to the latter leucine derivative is non-trivial. Furthermore, the bicyclic arrangement of phalloidin is due to an unusual tryptathionin bridge linking residue 3Cys and 6Trp. During the synthesis, two ring closures have to be achieved: typically one macrolactamisation for the base peptide and a thioether coupling for the bridge formation. Eventually, the success of the synthesis critically depends on the order in which these steps are executed and on the position where the lactamisation takes place.

synthetic challenges

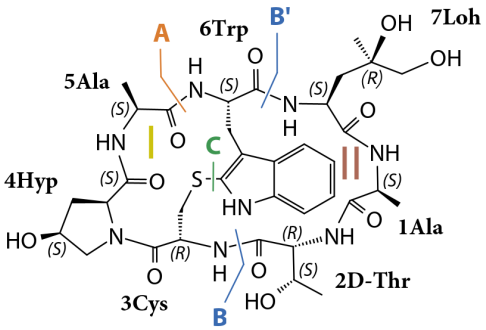


Figure 9.1 Structural formula of phalloidin The base peptide ring consists of the tetrapeptide turn I (3Cys, 4Hyp, 5Ala, 6Trp) and the pentapeptide turn II (6Trp, 7Loh, 1Ala, 2D-Thr, 3Cys). For natural phalloidin, the tryptathionin bridge is located 'above' the ring plane (here in the foreground). The major retrosynthetic approaches are highlighted with coloured cuts and will be discussed in the context of figure 9.3.

As a prominent synthetic byproduct to the naturally occurring phalloidin **1**, a kinetically hindered conformer—an atropoisomer **1'**—can be observed. In this isomer, the stereochemistry of the compound remains exactly the same but the tryptathionin bridge is located 'below' the ring plane (in the background in terms of figure 9.1). Figure 9.2 shows example structures for both the isomers alongside a simplified cartoon representation.

atropoisomers

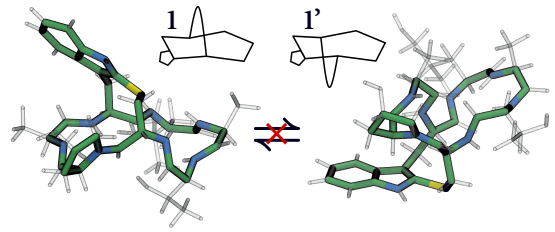


Figure 9.2 Structure of phalloidin atropoisomers Phalloidin **1** in a natural conformation as found in the crystal structure CCDC-147 213 of 7Ala-phalloidin[255] (bridge 'above' the ring) and in an atropoisomeric conformation **1'** (bridge 'below' the ring). The two are not in equilibrium. The atropoisomer **1'** was generated from the natural structure by artificially forcing the bridge through the ring.

We tackled the question, how the formation of atropoisomers can be understood based on different synthetic approaches. In this course, it was also assessed whether the isomers are indeed not in conformational exchange with each other, and whether the modelled atropoisomeric structure **1'**

research question

indeed matches the synthetic byproduct previously recognised as the relevant atropoisomer.[255, 275] We published our study in G. Yao, J.-O. Joswig, B. G. Keller, R. D. Süssmuth, *Chem. Eur. J.* **2019**, *25*, 8030–8034 under the title *'Total Synthesis of the Death Cap Toxin Phalloidin: Atropoisomer Selectivity Explained by Molecular-Dynamics Simulations'*. The full article including supporting information can be found in the *Publications* section of the appendix alongside an author contribution statement. Here, we used classical MD simulations to simulate phalloidin and its synthetic pre-cursors to assess the respective conformational ensembles and to make predictions about which synthetic route would lead predominantly to the desired natural product. The theoretical results assisted in the design and rationalisation of a laboratory synthesis that was carried out in the Süssmuth group[1] at TU Berlin in a continued cooperation. This synthesis is the first complete total synthesis of naturally occurring phalloidin including the problematic amino acid 7Loh.

study result

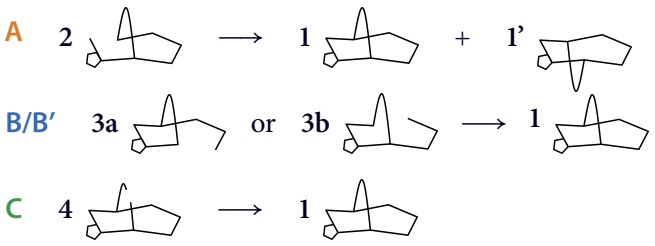


Figure 9.3 Reported synthetic approaches towards phalloidin The last step in the preparation of phalloidin can be **A** a ring closure between 5Ala and 6Trp (turn II formed first),[255, 275] **B/B'** a ring closure between 2Thr and 3Cys or between 6Trp and 7Loh (turn I formed first),[276] or **C** the bridge formation.[277]

Figure 9.3 summarizes the synthetic approaches reported in the literature and puts the new pathway into context. A recent review of phallo- and amatoxin syntheses can be also found in [257]. The first reported one by Paolillo *et al.* prepared 7Ala-phalloidin in 8 steps with a low overall yield of 1.3 %. The authors follow a linear strategy to assemble a heptapeptide that is then cyclised in a tryptathionin bridge formation to give precursor **2** (pathway **A**).[255] The approach is based on the early work of Wieland *et al.* on phallotoxins.[278] Atropoisomerism for phalloidin is mentioned but there is no report on synthesis byproducts. In another article published at the same time, however, the authors discuss atropoisomers for (2L-Thr,7Ala)-phalloidin, which was synthesised among a series of other derivatives in the same way.[279] Not much later, Guy *et al.* published an alternative solid-phase synthesis via precursor **2** (pathway **A**) that leads in 18 steps to 7Ala-phalloidin with the same overall yield of 1.3 %. This time, two atropoisomers are isolated in a ratio of 1:2.5 with 3.2 % overall yield for the undesired isomer.[275] Recently, a similar solid-phase synthesis over a turn II fragment but with the final cyclisation between 4Hyp and 5Ala was used to produce a fluorescent 7Leu-phalloidin conjugate with reported yields of 2-3 %.[280]

synthetic pathways

A promising synthetic approach that differs from these turn II centred ones, was proposed by Lokey *et al.* with a solid-phase synthesis in which the base peptide cyclisation is executed before the bridge formation via an intermediate **4** (pathway **C**). It gives 7Glu-phalloidin in 50 % overall yield in 19 steps.[277]

We, however, concentrated on yet another strategy taking the route via turn I fragments **3a** or **3b** (pathway **B/B'**). By MD simulations of reduced phalloidin precursor models representing turn I (residues 3, 4, 5, and 6) or turn II (residues 6, 7, 1, 2, and 3) species, it could be shown that turn I fragments have a higher predisposition to form natural phalloidin in ring closure reactions. In the modelled turn I system, the tryptathionin bridge was—judging by the dihedral angle between the bridge and the fragmental ring plane—consistently found in an orientation resembling the situation in the natural product. In the considered turn II system on the other hand, the bridge orientation alternated between both ring sites. Whether the bridge is in an orientation above or below the ring plane in the moment when the second ring closure takes place, presumably decides over which product

[1] R. Süssmuth, Biological Chemistry, Technische Universität Berlin

is formed. Hence, the distribution of bridge orientations in a precursor fragment should correlate with the observed product ratio. And indeed the experimental result reported by Guy *et al.* is in good agreement with the theoretical expectation (compare figure 2 in the main article). Similarly, as the simulation analysis suggests, the obtained results from phalloidin syntheses via precursor **3a** (called the [4+3] strategy in the main article) or **3b** (called the linear strategy) gave natural phalloidin exclusively in improved overall yields of 12 and 17 % respectively.[276]

As a minor addition to the analysis of phalloidin precursors, we subjected representative snapshots from simulations of the natural end-product and its atropoisomer to the prediction of CD spectra. While the structure of the natural conformer is well known, that of the atropoisomer has not been fully determined yet and is only presumed based on computational models.[275] CD spectra, however, have been measured for both the atropoisomers of two phalloidin derivatives. They show prominent absorptions around 250 nm in positive direction for the natural orientation and in negative direction for the other.[275, 279] For canonic phalloidin, only the natural form has been measured.[276] The agreement of the predicted spectra with the expectation varied greatly over the tested functionals and setups. Considering the small number of selected input structures and the suboptimal inclusion of implicit methanol solvation, the best matching result was still satisfactory and gives reason to believe that the produced and simulated atropoisomeric structure is correct (see figure 3 in the main article). A major source of complexity is in particular posed by the fact that the tryptathionin bridge can adopt two different arrangements even in the natural atropoisomer. The bridge can either rest above **turn I** as in the crystal structure (see figure S12 and S13 in the SI) or flip over to **turn II**, potentially affecting the helicity of the compound. Both structures are in moderately fast exchange and the helicity detected in the predicted spectra tends to be of opposite sign for them.

CD spectra

The approach used in our study is apparently (and surprisingly) quite unique. We simulated a natural product synthesis precursor with classical MD to estimate its eligibility to form a specific product based on the distribution of conformations it can be found in. We cannot cite any literature where something similar has been explicitly attempted to support a preparative organic synthesis. The study of product formation in terms of reaction pathways, rates, and transition states, usually calls for more complex theoretical descriptions like reactive or *ab-initio* MD.[281, 282] Such means are still prohibitively expensive in many cases to enter daily laboratory practice. Among the fundamental techniques of organic synthesis, however, rough hypothetical assessments of (pre-)transition states based on which conformational arrangement in the starting material is the most likely one have long been the excepted standard. Consider for example the Zimmerman-Traxler model for stereoselective aldolations in which product formation is purely rationalised by potential steric clashes between the reactants in their possible relative orientation.[283] Using classic MD to hypothesise reacting conformations on a basic, inexpensive level seems to be an obvious choice.

outlook

The present case, the macrolactamisation of a peptide, is very well suited for this type of analysis. Which cyclisation product is formed, apparently depends strongly on the conformational pre-organisation of the starting compound. If different precursor conformations lead to different products because the ring is closed from different sites or angles, the observed product ratio should be strongly influenced by the relative population of these conformations. An implied assumption of this, is that the reaction rates of the different assessed precursor conformations are approximately the same, which should be the case for this type of peptide cyclisation in general. It should not be forgotten, however, that this may not always hold. It is of course also required that the investigated system can be described by MD with sufficient accuracy. This is usually given for peptide systems, even when non-standard amino acids demand a force field parametrisation like in our case. Care needs to be taken in the design of the simulated model so that it most closely resembles the actually reacting species without being more complex than necessary. It needs to be decided on a case-by-case

basis if solvent, other reactants, or peripheral parts of a compound like protecting groups should be included into the computation.

Within the boundaries of these limitations, we would like to see the use of MD increase in the context of the planning and understanding of organic synthesis. Maybe in this way, about 20 years of effort made to synthesise phalloidin via the suboptimal **turn II** pathway could have been replaced with the better route via **turn I** fragments earlier. Recently, a similar approach was used in the group to assess linear precursor fragments of α-amanitin with respect to their reactive predisposition in tryptathionin bridge formations.[284]

Part IV.

C-type lectin receptors

{ 10 }

The human immune system

Dendritic cells and the pattern recognition receptor langerin

In the human immune system, dendritic cells (DCs) are an essential part and the discovery of their significance was rewarded with the Nobel prize in Physiology or Medicine in 2011.[285, 286] Together with macrophages they are among cells in the first line of defence against invading pathogens like viruses, bacteria, fungi, and other potentially harmful microorganisms. They are found in different tissues such as mucosae and skin—directly at sites where the danger of foreign body intrusion is greatest—and their main purpose is to capture all kinds of pathogenic threats. While they are themselves counted among the evolutionary conserved innate immune response that reacts towards intruders with a rather general and non-specific program, they are crucial for the initiation of not only innate but also adaptive immunity (figure 10.1 shows a very simplified illustration). One of their major characteristics in this regard is that they can act as efficient, professional antigen presenting cells. When a pathogen is seized, internalised by a DC, and subsequently degraded, its antigenic building blocks can be loaded onto MHC molecules displayed on the cell surface to be forwarded to T cells. This process requires the cell to mature, in which course pathogen uptake is downregulated, and to migrate for example to the lymph nodes. Activated T cells trigger specific immune responses, tailored towards a particular kind of pathogenic menace. They can for example either travel to locations in the body where an infection took place or remain in the lymphoid organs to further activate B cells. Because of their central character, DCs are a popular target for therapeutic studies and may for example be exploited in vaccination strategies.[287] For a general introduction to the basic mechanisms of the human immune system see [288].

dendritic cells

For the detection of pathogens, dendritic (and other) cells depend on PRRs that distinguish between different kinds of hazards via conserved pathogen-associated molecular patterns. The composition of expressed receptors can vary strongly among cell types and is an important characteristic, so that individual receptors can be used as cell markers. A prominent class of these receptors are C-type lectin receptors (CLRs), which recognise a variety of carbohydrate ligands on the surfaces of their targets.[289, 290] In our studies, we focused in particular on one representative receptor, langerin (CD207, also CLEC4K), which is primarily found on Langerhans cells (LCs), a type of cell that has been encountered as early as 1868 by Paul Langerhans as the first known DC in human skin, although not correctly identified as such at the time.[291] LCs play a complex role in the immune system, exhibiting classic macrophage as well as DC phenotypical behaviour.[292] They share a common origin and cell precursors with macrophages and the ability to proliferate, but their acting as migrating antigen presenting cells is typical for DCs. In a rare disease called LC histiocytosis, tumors are caused by uncontrolled excess proliferation of these cells.[293]

pattern recognition receptors

langerin & Langerhans cells

The CLR langerin can be classified as a group II member of the multifaceted C-type lectin-like domain (CTLD) superfamily, which contains type II transmembrane proteins that bind their glycan ligands in a calcium dependent fashion.[294, 295] The same group comprises the subgroups of asialoglycoprotein receptors (ASGR, MGL), DC receptors in the DC-SIGN group (including CD23

and LSECtin), macrophage receptors (e.g. MCL, Mincle, dectin-2), and scavenger receptors. Although, these receptors are structurally very similar because they posses a CTLD with a common fold (shown later for langerin in figure 10.3) that acts as a CRD, they are functionally heterogeneous and not all of them are well understood. The fold of the CTLD allows for enormous sequential variation without giving up the basic structure.[296] Structural variation among CLRs is mainly observed in loop regions associated with calcium(II) binding sites.[297]

Group II C-type lectin receptors

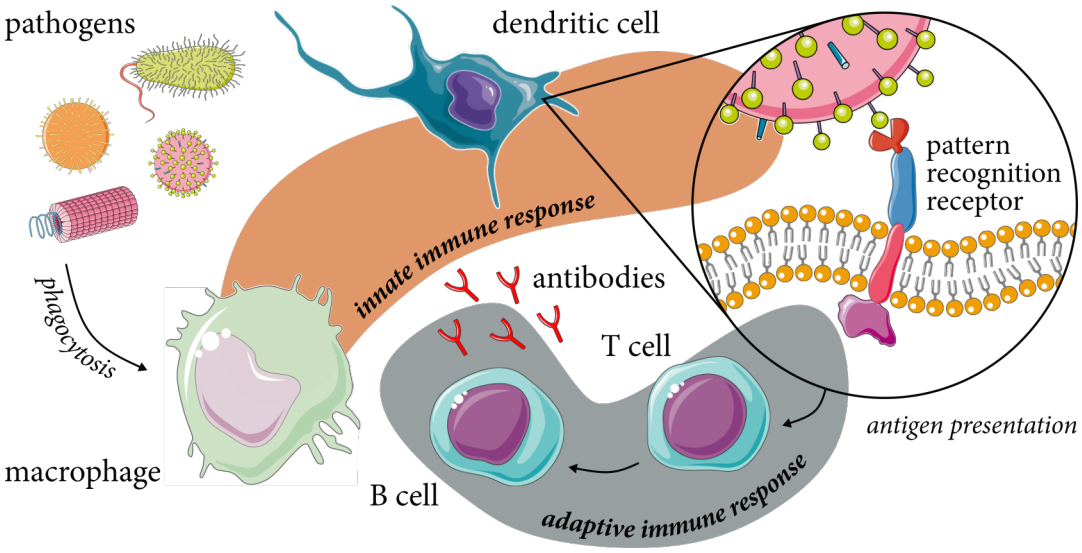


Figure 10.1 Dendritic cells in the human immune system
When pathogens invade the human body, they may among other things be intercepted by macrophages, which try to eliminate them through phagocytosis followed by lysosomal degradation. Similarly, they may be captured by receptors on DCs, which can trigger a variety of cell and receptor specific processes. DCs moderate between the innate and adaptive immune system for example by presenting antigens from degraded pathogens to T cells, which in turn may for one thing activate B cells to produce specialised antibodies. Note that the shown PRR does not represent a specific receptor but drafts a general architecture that may cover domains for extracellular recognition, membrane embedding, and intra-cellular signalling. Icon components by Servier Medical Art.

langerin literature

Figure 10.2 shows that langerin as a system, which was discovered in 1999,[298, 299] is still a hot topic in the literature. Since the start of this thesis in 2018, over 400 new publications have been recognised by SciFinder^n under the search term *langerin*. The number of publications additionally linked to the keyword or general concept of MD, however, is vanishingly small in comparison, indicating that langerin is studied prevalently on a (cell) biological rather than computational atomistic level.

SARS-CoV-2

The count of langerin associated publications rose up in 2020 after the numbers were steadily declining since 2015. This could be perhaps connected to the worldwide COVID-19 pandemic and an increased general interest in pathogen binding receptors. A few articles even directly link langerin to an infection with SARS-CoV-2. Recently, higher blood levels of langerin were causally associated with an increased risk of hospitalisation and a need for respiratory support or death due to COVID-19.[300] Langerin (as well as DC-SIGN) effectively binds SARS-CoV-2, but there is evidence that while LCs do not get infected themselves they capture and transmit the virus to cells carrying ACE2,[301–303] which is accepted to be the main host cell receptor for SARS-CoV-2 cell entry.[304]

HIV-1

Langerin is of broad interest in research also in the context of infections with HIV. The role of LCs and langerin in this regard is discussed controversially. Investigations are complicated by the fact that *in vivo* as well as authentic model studies are very difficult.[305] It has been found for instance that langerin poses a restrictive barrier against HIV-1 and can effectively prevent the infection of immature LCs and the transmission of the virus to T cells. Langerin binds HIV-1 gp120 due

to a specificity for mannose and internalises the virus in Birbeck granules,[306] a special form of cytoplasmic organelles exclusive to LCs.[307] Mechanistically it is suggested that TRIM5α associated with langerin acts as a restriction factor by inducing autophagic degradation of the endocytosed receptor-virus complex, triggered by the virus binding event.[308] This mechanism is in contrast for example not accessible to DC-SIGN, which is expressed on other types of DCs and was identified as a source of HIV-1 dissemination. On the other hand, LCs were shown to be productively infected with HIV-1 via the CD4 entry receptor and CCR5 co-receptor,[309] and that transmission to T cells is indeed possible.[305, 310, 311] Both pathways, direct HIV fusion and capture by langerin that guides the virus to its degradation, are likely competing. In a third alternative route, captured viruses may not be completely degraded but actually protected in virus-containing compartments—literally turning the cell into a trojan horse of preserving virus reservoirs for later infections.[310, 312] An additional influencing factor for what happens if the virus encounters a LC may be the cell's maturisation state, which changes its physiology and its susceptibility to infections, as well as that immune responses of cells can be different in inflamed tissues.[312] Furthermore, interactions of LCs with other DC types may be important, i.e. LC-DC cross-talk, in which langerin plays a role as adhesion protein for cell clustering.[313]. It has also been reported that there is a complex interplay between HIV and HSV infection.[305] Moreover, the human complement system can be involved non-trivially as shown by the observation that HIV-1 opsonisation can have the unintended effect of increasing the probability of infection.[314] Finally, there may yet exist a completely unknown receptor as the missing link.[315]

Unfortunately, what is learned about langerin in the context of HIV may not always be taken one-to-one to explain how langerin responses to other pathogens. For influenza-A for example it was shown that langerin acts as an entry point promoting the dissemination of the virus but it is not known why that is.[316]

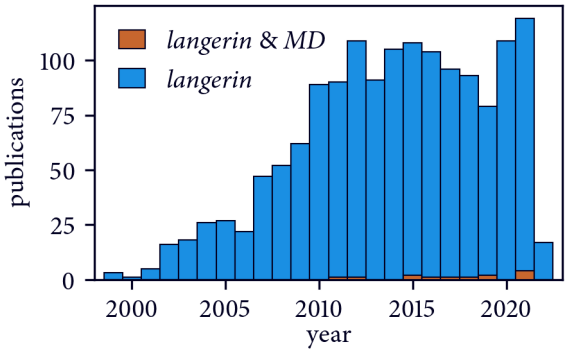


Figure 10.2 Langerin publication query Publications per year according to a SciFinder^n query (in March 2022) for the search term *langerin* with and without connection to MD as a concept or keyword. Discrepancies with the search results presented in [317] may be due to improvements in the search engine—in particular, the result presented here already includes entries for *CD207* as an alias to *langerin*.

receptor targeting

The complexity posed by the co-evolution of protective strategies of the immune system on one side and pathogenic counter-strategies that hijack these mechanisms on the other, makes it extremely difficult to develop therapeutic approaches to support human immunity. Because of the key importance of PRRs, it has become a main objective to design selective effectors that modulate the binding and signalling capabilities of these receptors.[318] Although CLRs are reckoned to be difficult subjects or even 'undruggable', significant progress has been made in this direction.[297] It is for example possible to tailor ligands that either bind preferentially to langerin or DC-SIGN despite the fact that both have a wide array of native ligands in common, which could allow blockage of one receptor while the other remains active.[319] Also, mannosylated antigen targeting to langerin before the background of using this receptor for vaccination did lead to an increased uptake but not to improved antigen presentation,[320] which could, however, be potentially improved using a different targeting strategy.[321] Furthermore, a secondary previously identified binding site[322] has been targeted for

DC-SIGN to increase binding to its native glycan ligands.[323] Similarly, thiazolopyrimidines have been found to inhibit ligand binding in murine langerin by addressing a secondary binding site.[324]

allostery

The phenomenon that a remote section of the protein interacting with an effector can modulate the protein's activity is called allostery and CLRs are excellent examples where allosteric regulation plays a major role.[325] CTLDs are fine-tuned, yet robust networks through which information can be transferred in various ways. Understanding the architecture of these networks at atomistic detail may be crucial to learn how CLRs fulfil their biological functions and how they could be manipulated. Progress in this direction was also the goal of our investigation of langerin, which will be addressed separately in the next chapter 11. MD simulations are an excellent tool to study allosteric protein regulation on the atomic level.[326]

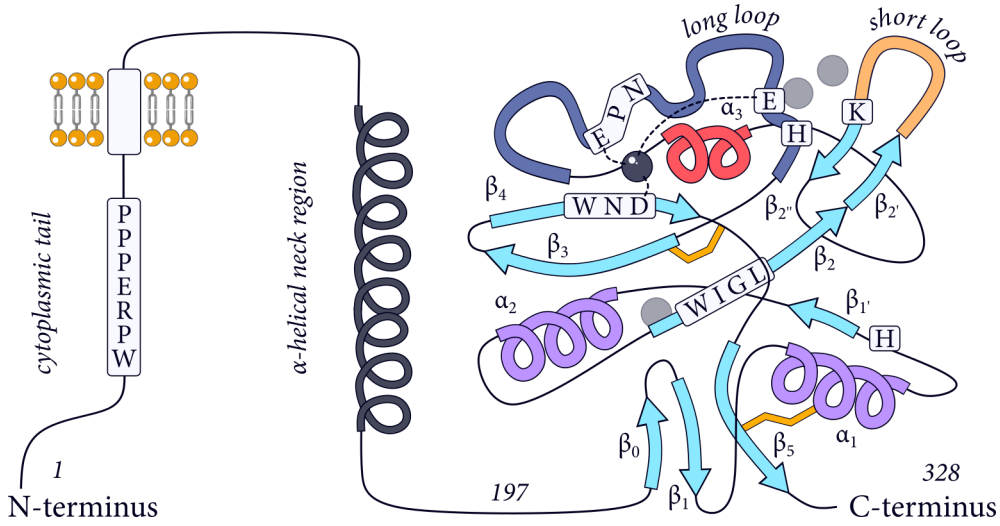


Figure 10.3 Structural domain organisation in langerin

The receptor has a short cytoplasmic tail comprising a proline rich motive that is likely to undergo protein-protein interactions,[299, 327] followed by a hydrophobic transmembrane region and a long extracellular α-helical neck that is involved in coiled-coil trimerisation important for multivalent ligand binding.[328] The C-terminal CRD exhibits the typical CTLD fold, comprising two extended β-sheets ($\beta_{0,1,5}$ and $\beta_{2,3,4}$, turquoise) flanked by α-helices (purple, the structurally rather flexibel α_3-helix in red). Calcium(II) in the canonic binding site (2) is drawn as sphere (grey) while alternative binding sites (1, 3, 4) not present in langerin are made transparent. Also shown are conserved cystein disulfide bridges (yellow).[294, 295, 329, 330]

CLR structure

Figure 10.3 illustrates the structure of langerin schematically. Its most prominent feature is the C-terminal CRD that harbours a single calcium(II) binding site formed by mainly three acidic amino acid residues, E285, E293, and D308, in the so called long loop (blue segment). In the presence of calcium, the protein binds carbohydrates through this site. Other calcium-binding sites are observed in CLRs but not in langerin (indicated transparently). The CRD can be described as compact and globular. In its center sits a highly conserved WIGL motif (residues 252 to 255) that serves as a sequence identifier but was not attributed a dedicated function.

carbohydrate binding

The selectivity of langerin binding towards mannose but also fucose, glucose, and *N*-acetylglucosamine is commonly attributed to the partially conserved EPN motif (E285, P286, N287).[329, 331, 332] Also conserved is the WND motif (W308, N307, D308) associated with the binding site. Figure 10.4 shows a typical binding mode of a mannose disaccharide to the primary binding site in langerin via the two vicinal equatorial hydroxyl groups 3-OH and 4-OH.

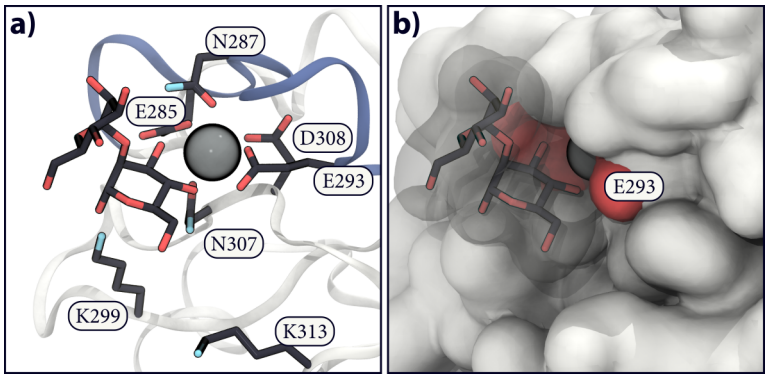


Figure 10.4 Langerin mannose binding Structure of the langerin calcium(II) binding site in complex with Manα1–2Man **a)** in licorice representation and **b)** in surface representation. Crystal structure PDB-ID 3P5F.[332]

K299 and K313, which possess positively charged side chains, can interact with ligands in the periphery and are decisive factors for the stability of binding complexes of langerin with 6-sulfogalactose while galactose itself is not efficiently bound.[332, 333] Similar stability enhancements are observed for sulfonated glucose,[319] and heparan sulfates where furthermore a preference for ligands with longer chain length can be found.[334, 335] Interestingly, ligand affinity in langerin is species dependent partly because K299 and K313 are for example missing in murine langerin.[336]

{ 11 }

The langerin project

Explaining pH-dependent calcium-binding

We embarked on our study of the CLR langerin motivated by a plain and simple observation: the binding affinity of this receptor towards its co-factor calcium(II) is pH-dependent.[329, 337] While the protein binds calcium ions rather well at neutral pH, the binding constant is significantly weakened at moderately acidic pH values of about 6. As carbohydrate binding is in turn calcium dependent, a decrease in calcium-binding affinity goes hand in hand with a loss of glycan ligand binding capability.

CLR calcium-binding

At a first glance, this may not be very surprising. A pH dependency of calcium-binding has been long known for other members of the CLR family, a classic example being ASGPR.[338] This endocytic receptor binds pathogenic targets on the surface of liver cells and releases them after clathrin-mediated endocytosis to be recycled and returned to the outer cell membrane, leaving its cargo for lysosomal degradation. In the extracellular milieu, where the receptor binding to pathogens occurs, we can assume an above neutral pH (a physiological pH of 7.4) and plenty of calcium in solution.[339] After internalisation, the pH is swiftly lowered in the early endosome to about 6.0 to 6.5.[340, 341] At the same time, calcium ions are transported out of the compartment.[342] This triggers the dissociation of protein-ligand complexes, opening the way for further processing stages. The pH-dependent calcium-binding of ASGPR is thus a necessity to enable its biological function. A similar pH-dependency is found for example in DC-SIGN.[343] The tight control of biological processes by cellular pH is furthermore a general motif in nature.[344]

recycling langerin

It seems reasonable to suppose that langerin acts in a comparable fashion and that the observed pH dependency is a manifestation of its function as an endocytic receptor for which extracellular binding and endosomal release are tightly controlled by its environment. Experiments with gold-labelled antibody DCGM4 that is selectively bound by langerin, suggest in fact a recycling of the receptor and an association of Birbeck granules—whose formation is induced by langerin—with the endosomal recycling compartment.[307, 345] Those also propose that internalisation takes a classic clathrin-mediated route and that Birbeck granules themselves are no endocytic vesicles. At least in the context of HIV-1 internalisation, however, it is contradictingly supposed that entire Birbeck granules including protein-virus complexes are destroyed by autophagosomes.[308] Receptor recycling in cells is a very complex topic involving different forms of endocytotic processes, sorting mechanisms in early endosomes, and multiple degradation and recycling pathways.[346] The exact fate of langerin and its cargos after internalisation remains essentially unknown, especially considering that it may be different for different kinds of ligands.[290] Nonetheless, understanding how langerin is influenced by pH-changes is certainly a crucial piece of the overall puzzle.

research question

On a closer look at the observation that langerin shows a pH-dependency in its calcium-binding, this is actually quite astonishing. It was found that the pH-dependency is a sole property of the CRD (see figure 10.3) and cannot be attributed to a change in the oligomerisation state.[329, 337] In fact, however, the CRD alone shows not much potential to be effected by a higher proton concentration. A

protonation of the acidic residues in the calcium-binding site right away is rather unlikely due to the low pK_a values of the respective carboxyl groups in the calcium coordinating side chains. The only amino acids that are sensitive to a change in pH in the relevant range are the two histidines H229 and H294. Indeed, point mutation of H294 to alanine could show that this site constitutes a pH sensor since the degree to which the calcium-binding affinity of the protein is reduced upon acidification is significantly lower when this residue is missing. Additionally, mutation of H294 and K257 leads to alterations in the calcium-binding as was illustrated by NMR chemical shift perturbations. This is in contrast to other residues that influenced the calcium bound and unbound state when mutated but did not change the perturbation pattern upon calcium binding.[337] Mechanistically, this gives rise to the question, though, how H294 protonation could effect the calcium-binding site because there is a non-negligible spatial gap between the two sites. There needs to be some sort of allosteric regulation via which the protonation signal is transported to the remote calcium-binding site. The experiments—counting in a MI analysis based on MD simulation data—revealed a complex involvement of a larger portion of the protein network including K257 and the short loop region but the atomistic details remained unclear.

study results

We published our study of the CLR langerin and give an answer to this question in J.-O. Joswig, J. Anders, H. Zhang, C. Rademacher, B. G. Keller, *J. Biol. Chem.* **2021**, *296*, 100718 under the title *'The molecular basis for the pH-dependent calcium affinity of the pattern recognition receptor langerin'*. The full article including supporting information can be found in the *Publications* section of the appendix alongside an author contribution statement. In there, we used classical MD simulations of langerin in various protonation states to elucidate the pH-induced structural changes through which the allosteric modulation of the calcium-binding affinity is realised in this protein. Laboratory experiments, including point mutations and affinity measurements were carried out by the Rademacher group[1] in a continued cooperation. Figure 11.1 summarizes the key result: protonation of H294 facilitates the formation of a new conformation in which K257—in the neutral state occasionally engaged in a hydrogen bonded interaction with H294—bridges over to D308, effectively moving a positive charge into close vicinity of the calcium-binding site.

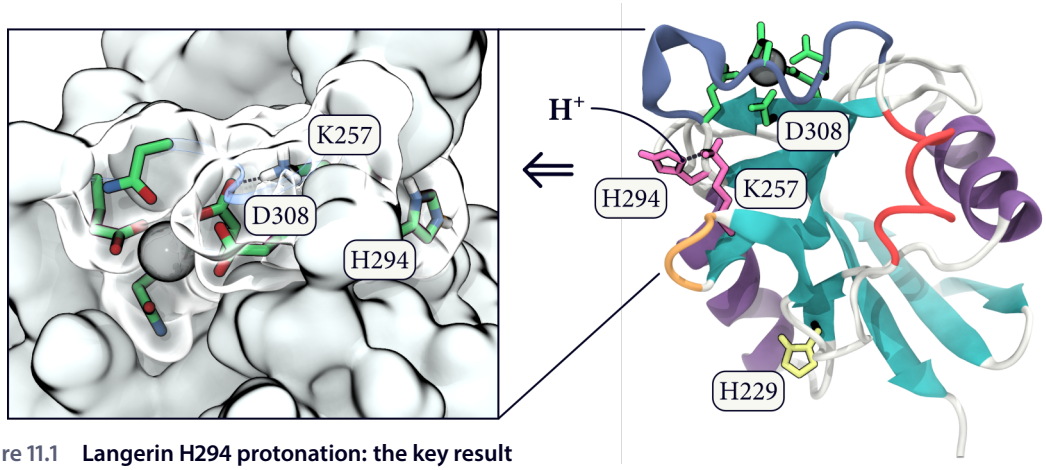


Figure 11.1 Langerin H294 protonation: the key result
On the right side, a crystal structure of the langerin CRD is shown in the assumed neutral state (both H294 and H229 have neutral side chains). According to our findings, side chain protonation makes a new protein conformation accessible that features a K257–D308 H-bond (left side).[347] This conformation is a hot candidate to explain how H294 protonation affects the calcium binding site.

[1] C. Rademacher, Molecular Drug Targeting, Universität Wien

The CRD of the CLR langerin hid its secrets rather well. We started by concentrating on simulations of the calcium bound protein in a presumed neutral and histidine protonated state. For the practical fine points of how the simulations were carried out see chapter 12. Ideally, we expected to detect a rather global conformational transition of the protein in the protonated state that involves H294 as well as the calcium-binding site. This means we hoped to find a structural change that obviously translates the information of an environmental pH change into something that indirectly (allosterically) affects how well the protein can bind calcium. A classic example for a protein regulated functionally in such a fashion is nitrophorin 4 that releases nitric oxide when undergoing a pH-induced transition from a closed to an open conformation.[348]

neutral vs. protonated bound state

Soon we had to learn, though, that the investigation of quite a few structural features could not bring convincing differences between the two langerin states to light (compare figure 2 in the paper[347]). The protein is—independently of the protonation state—overall quite rigid, except for the segments associated with the short-loop, the α_3-helix, and a part of the long-loop. Secondary structure elements that could be identified in the protein sequence are virtually unaffected by the modelled protonation. Also, fluctuations and respectively deviations of individual atomic positions are not very informative here. At the same time, gauging local features like backbone and side chain dihedral angles gave us only an impression of isolated changes with respect to single protein residues but it was not possible to establish a link to H294 and the calcium-binding site. In particular, we could not detect any striking influence on the calcium-binding site itself. In short, there was no pH-induced large scale conformational transition that could be rendered responsible for a change in the calcium-binding affinity.

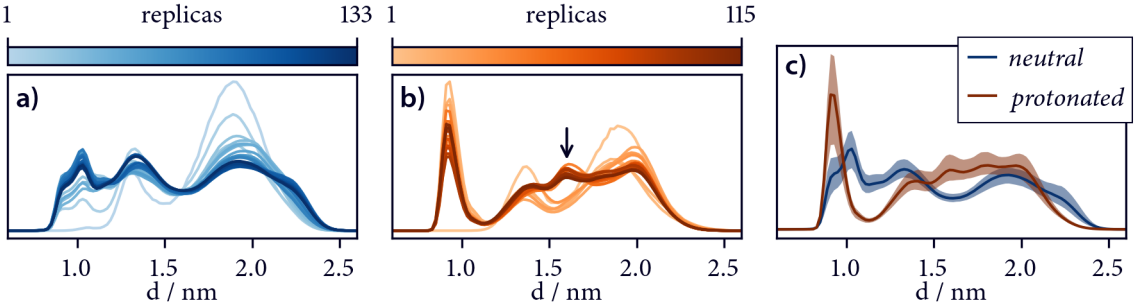


Figure 11.2 Convergence of the short-loop/long-loop distance
For simulations of calcium bound langerin in the **a)** neutral and **b)** histidine protonated state, histograms of the observed M260–G290 C_α-distance are shown as the mean over an increasing number of replicas. **c)** Final distributions for both the states with 95 % confidence interval on the mean (transparent area) obtained by bootstrapping (1000 samples).

loop distance

A straightforward inspection of the system was hindered also by a slow convergence of the simulation data. Figure 11.2 illustrates this taking the M260–G290 C_α-distance as an example, which we used to get an impression of the short-loop movement with respect to the long-loop. This distance distribution changed substantially with the progressive execution of additional simulation replicas. In particular for the protonated state, about half of the simulation effort (~10 µs) was necessary to observe a set of conformations at intermediate loop distances as a noticeable population (indicated by the arrow). As it turned out, especially these conformations are connected to the K257–D308 bonded structure that was eventually identified as potentially crucial for the modulation of the proteins calcium-binding affinity. Similarly, loop orientations with shorter distances were for both protonation states observed representatively only after a considerable amount of simulation time. While the distributions over all replicas appear to be reasonably equilibrated for the acquired data set on average,

the broad confidence intervals indicate that separate replicas sample only subsets of the ensemble and that it can make a big difference for the analysis which replicas are taken into account.

From the differentiated short-loop/long-loop distance distributions we could continue to further narrow our search for pH-triggered changes in langerin to the inter-loop region. While it was suspected from the beginning that the short-loop would play a major role, it became now clear that H294 protonation kind of causes a separation of closed and rather open loop orientations. Yet, at this point it was not obvious that this entailed the population of an entirely new conformation that becomes only accessible after protonation. On the contrary, it seemed that while both protonation states cover the same conformational space, there is just a change taking place with regard to which conformations are favoured. In other words, it looked like as if the protonation led to a population shift—a redistribution of the conformational ensemble—in terms of a very subtle allosteric mechanism that does not manifest itself in a distinct (large scale) conformational change covering the calcium-binding site. Allostery in protein networks can show itself in many different ways.[349] This includes the considered possibility of allosteric regulation in the absence of conformational changes,[350, 351] although it can well be in these cases that *'not observed'* does not actually mean *'not existent'*.[352]

allostery?

We obtained a similar picture like that shown by the loop distance above by leveraging PCA (see theory in section 5.3, compare figure 3 in the paper[347]). The first eigenvector of this analysis—a collective coordinate along which the protein exhibits a motion with the largest observed amplitude—coincides well with the distance between short- and long-loop. A two dimensional projection of the simulation data into the reduced space of the first two PCs reveals more detail, though, than the inter-loop distance alone. As it was the case for the loop distance distributions, the PC projections needed excessive sampling to be reasonable converged and to show the differences between the protonation states as clear as in the final analysis. PCA has the beneficial trait that the two protonation states that cover a very similar conformational range could be subjected to a joint analysis in which a set of PCs was obtained that are likewise suitable to condense the information for both systems, i.e. it was possible to represent the two states within the same projection for a direct comparison. Again, for the protonated state we could notice a population emerging in the center of the distribution.

PCA

We went on by taking a closer look at the (projected) conformational states that experience a change in their population upon protonation. For the separation of the conformations we used CommonNN clustering (see chapter 14) that was continuously developed further back-to-back alongside our langerin study and is particularly appropriate to extract clusters of conformations that are not necessarily constrained to a specific shape or spatial layout but correspond to low energetic, highly populated groups of molecular structures.

PC clustering

The most fruitful characterisation of the obtained states was achieved by collecting information on hydrogen bond occupancies from the simulation data (compare figure 4 in the paper[347]). In parallel, a hydrogen bond analysis for the complete ensembles of the neutral and the protonated state exposed several interesting interactions that are perturbed upon histidine protonation. Figure 10 in the SI of our paper[347] shows a filtered summary of the full analysis in which we considered changes in the relative observation frequency of interactions as well as pairwise Pearson correlations between these interactions. The combination of the discovery of virtually two sets of correlated hydrogen bonds in the short- and long-loop region with the fact that these could be strongly associated with distinct conformational clusters in the PC projection of the protonated state, enabled us to draw a detailed image of what happens to the protein upon protonation and to identify the key conformation presented in the beginning. This key conformation is aptly described by the presence of two highly correlated side chain/side chain hydrogen bonds: the already mentioned K257–D308 bridge, which is barely populated in the neutral state, and H294–E261, which is not observed in the neutral state at all. The set can be supported by a bridge of K257 to the E293 backbone. Consistent with the notation in

H-bond analysis

our paper, I will refer to this conformational state as the *G* conformation as it can be associated with the *green* cluster in the PC projection. The other set of correlated interactions is competing with *G* and comprises foremost N288–E261, N288–M260, G262–E261, and K257–E261 hydrogen bonds. In the neutral state, this second set is present and correlated as well but could not be linked to a cluster in the PC projection. In the protonated state, H294 can engage with the backbone of N291 in the context of this set as well. It will be referred to as the *O* conformation related to the *orange* cluster in the projection of the protonated state.

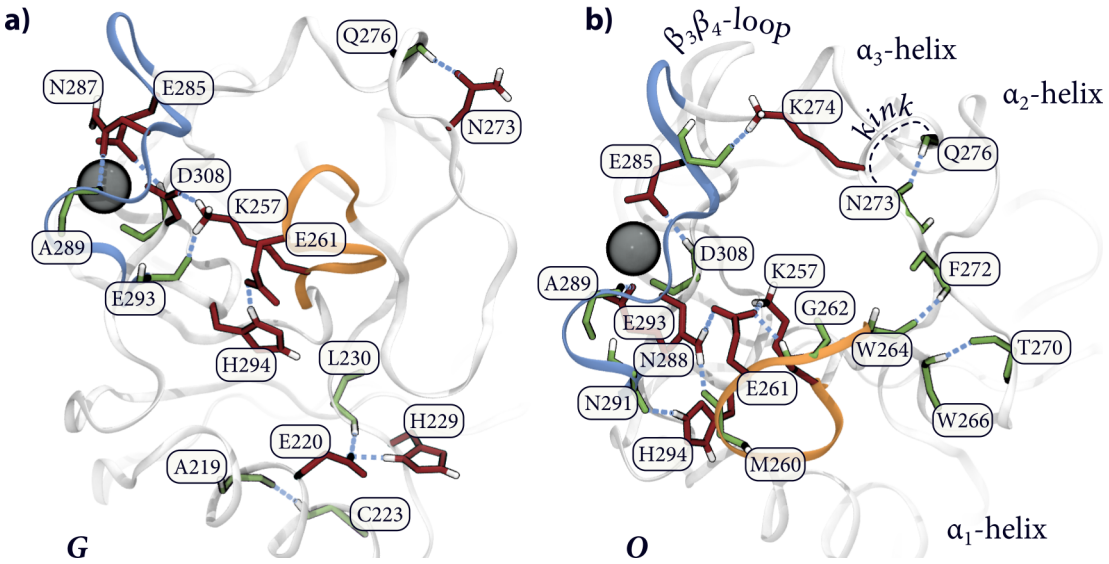


Figure 11.3 Competing H-bond patterns in protonated langerin

In the histidine protonated calcium bound state, two prominent, largely mutually exclusive sets of pairwise correlated hydrogen bonds can be found among the conformational ensemble. The illustration here is an extended version of figure 5 in our langerin paper with the added complexity of weaker correlated interactions.[347] **a)** The *G* conformation that potentially explains a decrease in the calcium-binding affinity upon protonation is strongly characterised by simultaneous formation of K257–D308 and H294–E261 bridges. Secondary contributions are K257–E293, D308–E285, and A289–N287 related to the calcium-binding site. Remote correlation can be noticed with Q276–N273 (side chain) in the α_3-helix and interactions originating from protonated H229 close to the α_1-helix: H229–E220, L230–E220, and C223–A219. **b)** The other set of interactions, the *O* set, is centred around N288 as the major player: N288–E261, N288–M260, G262–E261, and K257–E261 tightly couple short- and long-loop in a very close arrangement. Additional factors are H294–N291, and A289-E293 and D308–E285 in the binding site. Among the more remote correlations are interactions in the α_3-helix and the sequence connecting this helix to the short-loop: W266-T270, F272–W264, and Q276–N273 (backbone). Note that this combination of interactions implies a peculiar 'kink' of the α_3-helix between N273 and Q276 and occasionally a K274–E285 H-bond. Amino acid backbones are coloured in green, side chains in red, hydrogen in white, and H-brides with dashed blue lines (all shown interactions involve N-H donors)

extended H-bond analysis

Interestingly, both sets *G* and *O* can be associated with an increased population of a D308–E285 interaction in the binding site as a shared feature. This bond is drastically more frequent in the histidine protonated state. Furthermore, the two sets of dominating interactions are accompanied by other weaker correlated interactions that are partly taking place in remote areas of the protein or can be related to the calcium-binding site (see figure 11.3). There is for one thing A289 in the long loop that can undergo an interaction with the side chain of E293, which can be connected to the *O* set, or with the side chain of N287, which can in contrast be linked to the *G* set. A289–E293 and A289–N287 are highly anti-correlated. The absolute observation frequency of the two bonds does not change upon protonation though. On the other hand, there is Q276 in the α_3-helix, which can be found to interact with the side chain of N273 in conjunction with the *G* set. In the *O* set, however, Q276 can bridge over to the N273 backbone, which gives rise to an eye-catching distortion of the early segment

in the helix, marked as a *kink* in figure 11.3b. Both these Q276–N273 interactions are again highly anti-correlated, while only the one involving the N273 backbone is more populated after histidine protonation. Beyond this, the variation in the *O* set correlates significantly with a pair of H-bonds in the sequence connecting the α_3-helix and the short-loop, i.e. W266-T270 and F272–W264. It can also enable a K274–E285 H-bond, which is very rare, though. There may be as well other structural consequences that have not yet been considered separately as suggested by a rearrangement of the $\beta_3\beta_4$-loop and the α_2-helix in the background. Finally and somewhat surprisingly, the *G* set correlates also with interactions where H229 is located like H229–E220, L230–E220, and C223-A219 that are in reverse anti-correlated with interactions in the *O* set. H229–E220 and L230–E220 are highly correlated themselves and very abundant in the histidine protonated state but close to absent in the neutral state.

In our continued analysis, we focused on the main characteristic of the *G* set because the K257–D308 H-bond did strike us as most important for the modulation of the calcium affinity and concentrated on the short-loop/long-loop region. The extended view of the present perturbations to the protein network shows, however, that potential implications for other areas should not be entirely forgotten. It should also be mentioned that while our interest lies primarily on H294 as a proton sensor, which is backed by wet-lab experimental results,[337] when this residue is protonated, H229 is almost certainly protonated, too. Consequently, we incorporated the H294 together with the H229 protonation in our low-pH simulations and contingently observed effects are always a result of both these protonations. An investigation of the influence of independent protonations at the two sites has so far not been attempted. Experimental data on a possible H229 mutant and its calcium-binding are not available either. H229 was rejected, however, as a significant factor based on the observation that around its location only rather weak chemical shift perturbations were recorded upon calcium-binding The rejection follows the argumentation that if H229 should have an effect on the calcium-binding, calcium-binding should in turn have an effect on H229, which is apparently not the case with Y217 and S232 being the closest residues experiencing a stronger perturbation. Moreover, H229 did not stand out in the respective MI analyses at least not with an obvious connection to the calcium-binding site.[337] A short review of the analysis with our updated data can be found in section 11.1. From our structural analysis of the simulation data, we recognised the H229 protonation only through rather local effects as well, apart from the above presented long range correlation of hydrogen bonded interactions.

H229

Now, after the exploration of the K257–D308 comprising *G* conformation, we had two tasks ahead of us to confirm that it is indeed relevant. First, we needed to assess if it really was a representative conformational state with a sufficiently long live-time and not just a fluctuation within a broader ensemble of structures. Second, we needed to show that it could really have the supposed effect of lowering the calcium-binding affinity. The first objective was relatively straightforwardly addressed using a Markov model of the conformational dynamics in which a sufficiently metastable macrostate could have been recovered that agrees well with the conformation in question (compare figure 7 in the main article and figure 11 to 18 in the SI[347]).

Markov model

The second part—proofing a potential decrease in calcium-binding affinity caused by the K257–D308 interaction—turned out to be exceptionally tricky. A basic validation of the possibility that the positively charged K257 side chain in the vicinity of the calcium-binding site creates an unfavourable interaction with the calcium ion was done by inspecting the Coulomb contributions of the K257–Ca^{2+} pair (compare figure 20 and 21 in the SI and figure 6 in the main body of our paper[347]). This illustrates that indeed there is a significant repulsive interaction between K257 and the calcium ion in the K257–D308 bridged state that is higher populated after histidine protonation. At the same time, the analysis confirms that the protonated H294 side chain contributes negligibly to the calcium repulsion. Rigorous approaches to calculate protein-ion binding free energies are, however,

calcium affinity

conceptually complex, imply large computational costs, and significant progress in the field has been only made quite recently.[122, 353, 354] Within the scope of our study, this was out of reach at the time, and we were not willing to put in the necessary effort without having a first order assessment of the potential relevance of the conformations we discovered. Chemical intuition tells us that the K257–D308 hydrogen bonded interaction is very likely to effect the calcium-binding but we needed a relatively cheap, well accessible technique to estimate roughly if this could be actually true. We decided to use *steered*-MD simulations for exactly this purpose (see section 4.9 for theory), in which the calcium ion is pulled out of the binding site and the force necessary to remove it completely is recorded.

The measurement of calcium-release rupture forces for an array of langerin states (compare figure 8 in the paper[347]), confirmed that we are not on the wrong track with the conjecture that K257 is a big factor for the modulation of the calcium affinity by bridging over to D308 in the histidine protonated state. As the main message, the rupture forces are noticeably decreased when the pulling experiments are started from the *G* conformation—featuring the K257–D308 H-bond—compared to simulations of the neutral state that is virtually missing this interaction. Secondary observations, speaking in favour of the importance of K257–D308, are for one thing that the rupture forces in K257A mutants are independent of the protonation and mutation state of H294. Further, for H294A in which K257 is present but the K257–D308 interaction is only weakly populated, the rupture forces are comparable to the neutral state. Further testing of the *G* conformation by for example mutating E261 to aspartate is difficult because such mutations can have various other implications. On the one hand, E261D has a negative effect on the stability of the *G* set and starting pulling simulations from a respective analogue of this conformation showed that the decrease in rupture forces is largely compensated by the mutation. On the other hand, E261D had no effect on wet-lab experimental measurements of the calcium-binding affinity (compare figure 9 in our paper[347]), which could be owed to the fact that E261 is also a stabilising factor in the *O* set—preventing K257 from an engagement with D308—for which we have at this point no reason to suspect that it decreases the calcium-binding affinity. Conventional simulations of E261D showed that the K257–D308 interaction is still favourable in the histidine protonated state, in line with the observed binding affinities in praxis (compare figure 19 in the SI of our paper[347]).

steered-MD

It is a limitation of the *steered*-MD approach that while we aim on explaining a difference in the calcium affinity between the neutral and the protonated langerin ensembles tested in laboratory experiments, we essentially probe conformational sub-states in the computer experiments. On the one hand, a comparison of the rupture forces for the neutral crystal structure with those for the protonated *G* set taken from the computer model agrees well with the trend for the 'real' ensemble: both show a reduction in calcium-binding capability upon protonation. But on the other hand, a comparison with the forces for the E261D modified *G* set does not: in the computer experiment, E261 mutation diminishes the rupture force reduction, indicating that E261 is important for the modulation of the calcium-binding affinity. In the laboratory experiment, however, E261D mutation had no effect on the pH-sensitivity of the protein. A possible explanation is that the E261D ensemble comprises conformational states that effect the calcium-binding affinity, different from the one subjected to the pulling experiment. Unfortunately, a comprehensive probing of all conformational states in both the neutral and the protonated ensemble becomes quickly intractable.

steered-MD limitations

The assessment of the calcium affinity from classic MD simulations has another, potentially crucial flaw. In standard force fields like the well established AMBER99SB variant that we used in our simulations, metal ions including divalent calcium are treated only rudimentary. The calcium ion is modelled as a simple point charge in connection with soft sphere Lennard-Jones parameters. Apart from missing terms that could account for coordinative contributions with possible geometric consequences, a major drawback of this model is that electrostatic interactions between the ion and

calcium model

residues in the protein as well as with the surrounding solvent are probably overestimated.[118, 119] Before this background, it is actually a bit surprising how well the simplistic *steered*-MD analysis does in comparison with the laboratory experiments. We might get away with it here because we are only evaluating rupture force differences and the error we make is made systematically in all individual experiments. It can be suspected, though, that the behaviour of the system is in reality quite a bit different to what we see in the simulation. In particular, the implications of the K257–D308 interaction could be more substantial, considering that the D308–Ca^{2+} interaction is not accurately described and could be actually much weaker. A fully satisfying model of the system using a higher level of theory may call for very elaborate and costly QM/MM descriptions. As a compromise, polarisable force fields represent an addition to the purely classical MD picture that could counter the problems of an inapt calcium ion model and the present protein-ion interactions. First results of langerin simulations using an AMOEBA force field can be found in section 11.2.

With the discovery of the K257–D308 interaction, its identification as part of a stable and relevant protein conformation, and an assessment of how it affects calcium-binding, we have now a neat atomistic description of how pH-induced changes can mechanistically control the calcium affinity in langerin. Although the allosteric transition in this context is quite small and does not include significant structural changes in the orthosteric binding site, it really is a clear switching mechanism. Still there remain many open questions. To begin with, when H294 was identified as a pH-sensor, it was already clear that mutation of this residue was only responsible for a partial decrease in the pH-sensitivity of the calcium affinity. Even without this sensor, langerin is still influenced by a change in pH in this regard. If H229 is excluded as a potential second sensor, a direct protonation of the calcium coordinating residues in the binding site comes back on the table. Although as mentioned earlier, the low pK_a values of these acidic residues normally prohibit a significant protonation in the pH regime of about 6, we could show that a shared protonation of two adjacent side chains—especially E285 and D308—in the binding site may be feasible (compare figure 11 in the paper[347]). Because of the presumably intense competition between a binding site protonation and calcium-binding, it seems likely that such a protonation would pre-dominantly occur in the absence of calcium. Furthermore, via the K257–D308 interaction, a protonation of D308 may be a conceivable consequence. It is thinkable that a binding site protonation is the dominating factor that eventually modulates the calcium-binding affinity, rendering the H294 mediated mechanism that brings K257 close to the binding site a supporting trigger to shuttle protons towards D308 and make the protonation there more efficient. In our work, we chose the relatively cheap PROPKA 3.1[355, 356] method to be able to estimate pK_a-values for a very large number of different conformations. As our results show, the computed values are strongly conformation dependent and broadly distributed, which confirms the importance of basing this analysis on a diverse set of structural samples. It is possible to calculate pK_a-values with higher accuracy[357] but if the higher implied cost allows the consideration of only a few structures, this can be a critical trade-off.

second pH-sensor

Another level of complexity is posed by the observation that the long-loop region harbouring the calcium-binding site can unfold, which goes hand in hand with a loss of the calcium-binding capability. In our classical simulations, this was never observed in the presence of calcium sitting in the binding site and holding the long-loop tightly in place but happened frequently in simulations of the protein when the calcium ion was removed (compare figure 10 in the paper[347]). Binding site protonations can significantly accelerate this unfolding. The effect of histidine protonations was inconclusive, though, since a protonation seemed to affect the unfolding pathways but did not accelerate the unfolding overall. For the complete picture, pH controlled calcium-binding in langerin has to be considered in the context of the unfolding, calcium-binding, and protonation states summarized in figure 11.4.

long-loop unfolding

In conclusion, our study of the CRD of the CLR langerin is the first investigation of this scale with MD. A conformational transition upon histidine protonation has been identified, rendering K257 as centrally important to transport the allosteric signal that the pH has changed in the protein environment to the calcium-binding site. Further research will need to be put forward to confirm the significance of this finding. A focus will need to be to overcome the limitations of the classic MD description (see section 11.2 for first steps into this direction). This will be especially important in the context of free energy estimates. Our presented approach will hopefully be inspirational for future investigations also of similar proteins. The CLR family offers much room for this and will possibly provide comparative insight into the behaviour of these systems (see also figure 32 in the SI of our article[347]). It may also promote the general description and understanding of calcium-binding proteins that fulfil various biological roles in nature.[358] As a quick outlook, the concept of mutual information will be addressed as an example for further analysis of allosteric communication within the CLR-CRD scaffold, in the next section.

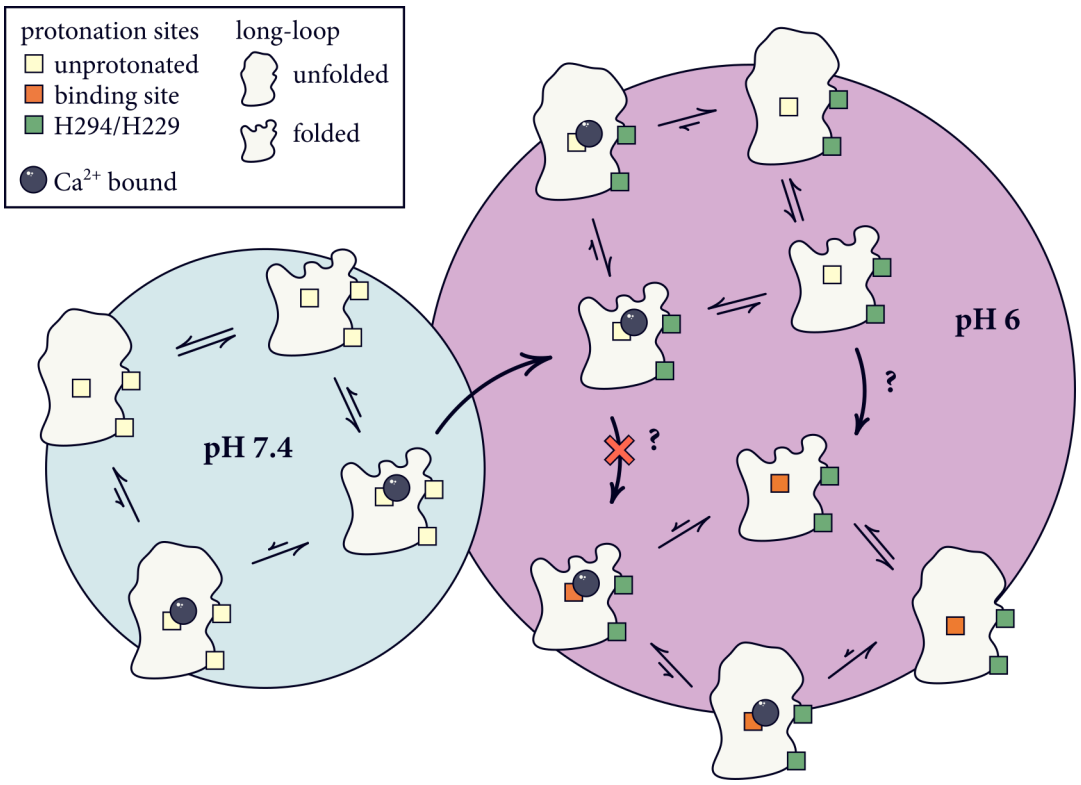


Figure 11.4 Studied protonations and binding/unfolding equilibria in langerin
In a neutral environment (pH 7.4) given a sufficiently high calcium concentration, the presumably dominating state is canonically folded calcium-bound langerin. This state is carbohydrate binding competent. Unfolding in the presence of calcium is not favourable but this is alleviated in the absence of calcium. Though unfolding was observed as being irreversible on the timescale of our simulations, we can make no statement about the actual folding/unfolding equilibrium. Experimentally, unfolding has been linked to P286 *cis*/*trans* isomerisation, which is not shown for the sake of simplicity.[337] In a mildly acidic environment, a histidine protonation is very likely, shifting the calcium-binding equilibrium towards the unbound state (the calcium bound state is, however, still preferred). It should be stressed that H229/H294 protonation is equally likely and has not been studied separately. Unfolding equilibria are virtually not affected. The possibility of an additional binding site protonation has been considered but is preliminary ruled out in the presence of calcium. Such a protonation drastically favours calcium release and unfolding.

11.1 Mutual information analysis

thermodynamic vs. structural allostery

In our langerin study, we used the produced MD trajectories to compare the conformational ensembles of protonation states in the hope to detect differences in the populated structures that could explain an effect on the calcium-binding site—and we eventually did. In the context of allostery, this can be seen as the *thermodynamic* or *free energy surface* centred approach: allosteric effector binding gives rise to a perturbation of the system that favours or disfavours certain conformational states.[359, 360] On first order, it interprets allosteric transitions as a switching between *active* and *inactive* protein states. The case of langerin demonstrates that these transitions can be subtle and difficult to describe.

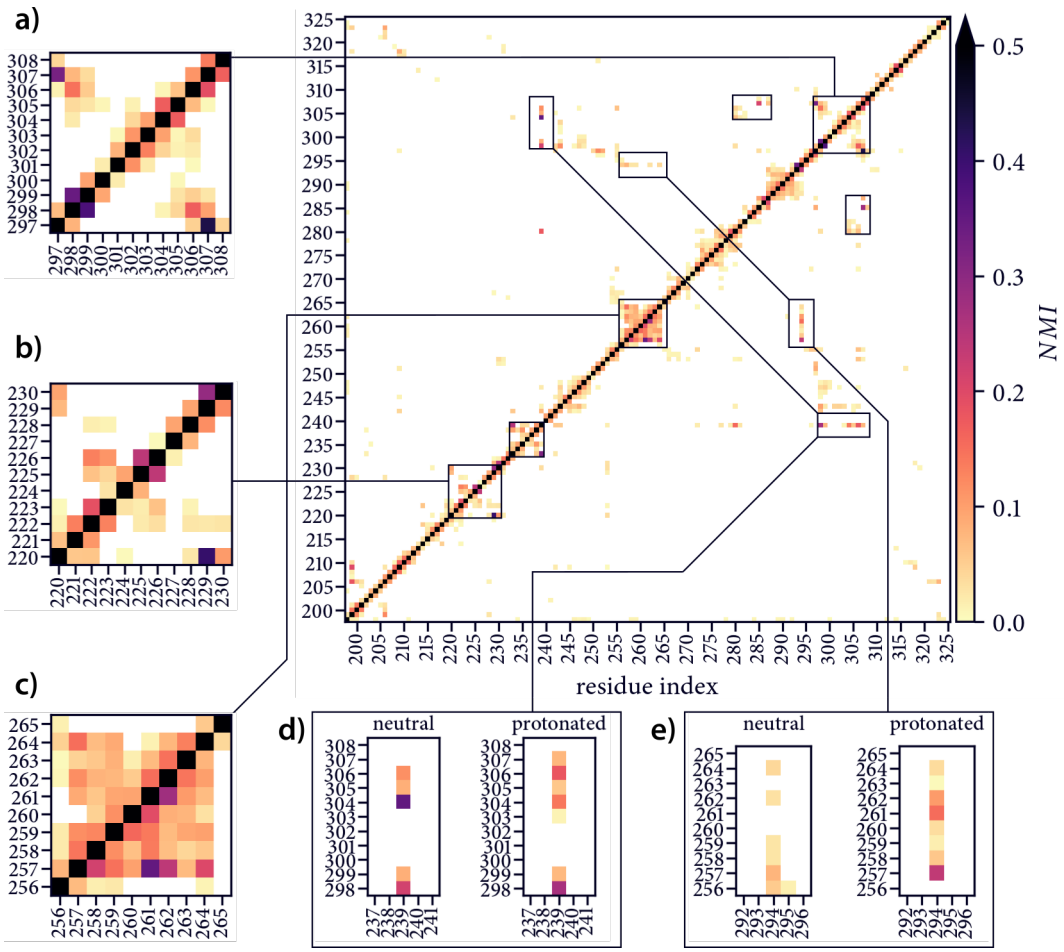


Figure 11.5 Mutual information matrix for calcium-bound langerin
Residue-wise projected, normalised MI on backbone and side chain torsion angle trajectories. The values in the neutral state are in the upper left, and those of the protonated state in the lower right triangle. Before the background of possible allosteric communication, several relationships over a longer sequence distance can be of interest. **a)** Between the WND motif and the β_3-strand. **b)** Between H229 and the α_1-helix. **c)** Within the short-loop. **d)** Between the WND motif/β_3-strand and the α_2-helix. **e)** Between H294 and the short-loop. Not zoomed in but highlighted are communications in the α_2-helix and between long-loop and WND-motif, i.e. within the calcium-binding site.

A MI analysis like the one performed already for the previous publication,[337] is another approach that can be used to reveal allosteric communication pathways in proteins (see section 5.4 for theory). It gives a representation of a system in terms of pairwise dependencies between feature distributions. This reflects a *structure* centred approach: regions in a protein that are interconnected in terms of this measure are likely to transport information.[359, 360] Perturbations on one site of a MI network may affect the network all together. A study from this point of view allows arguing about possible allosteric regulation in terms of its structural basis without describing the allosteric (conformational) transition itself.

Both the perspective in terms of population shifts and in terms of communication networks provide information about the same underlying allosteric process but individual systems may be not equally well approachable from either side. Especially in the (apparent) absence of conformational changes, a structural approach still provides insight into available allosteric pathways. It is also a main factor in the description of *dynamic* allostery. The terminology is admittedly not very clear since of course also thermodynamic population shifts are connected to structure and communication over structural networks have a thermodynamic basis.[361]

Here, we take kind of a hybrid approach comparing MI networks for calcium-bound langerin in the neutral and in the histidine protonated state. From the perturbation of the network upon the protonation, one might learn which protein regions are jointly affected by it, hence being the structural basis for the allosteric effect.

For each pair of dihedral angle distributions (backbone and side chain) extracted from our MD trajectories of the langerin CRD, a pairwise MI score was computed. This gives essentially a square matrix of n^2 elements where n is the total number of considered angle features. Then, this angle-wise result was projected into a condensed residue-wise form to become comprehensible (see section 5.4 for theory). Figure 11.5 shows the final MI matrix. Besides expectedly strong communication between sequentially neighbouring residues, other potentially interesting connections become visible here. In particular there is a correlation between H294 and the short-loop, and between the calcium-binding site (the WND motif), the β_3-strand and the α_2-helix. The magnitude of the computed scores, i.e. the intensity of the communication, changes due to the protonation.

MI matrix

An expressive alternative way to illustrate MI networks is depicted in figure 11.6 in the terms of graphs. In these graphs, each residue is represented as a node and the edges connecting pairs of nodes are scaled in width proportionally to the respective MI score. Interconnected regions of protein residues become even more clearly visible in this way than in the matrix picture. Especially when differences in the MI scores between the protonation states are considered (figure 11.6c), connected sub-networks in the langerin CRD that are potentially important for allosteric communication become apparent. We see a strong increase in communication within the short-loop, especially involving K257 upon histidine protonation. At the same time, the connection between E293 and H294, virtually the only visible connection between short-loop and long-loop, is weakened. Note that the conformational transition we have observed can not be anticipated based on this view, but it is clear that it involves the short-loop including K257. This can be mainly attributed to the structural inflexibility of D308. Changes affecting this residue are at least not detectable in terms of torsion angle populations. As a word of caution related to this, it should be noted that a high MI score should not be misinterpreted as a molecular interaction. Significant mutual dependencies can have their origin in such interactions (like for the H229–E220 H-bond) but this is not generally the case. For H294 and K257 we see exactly the opposite: in the neutral state, where a H-bond between these residues can be formed, the MI score is lower than in the protonated state, where the same is forbidden. In reverse, a low dependency does not exclude the possibility of strong functional interactions.

MI graphs

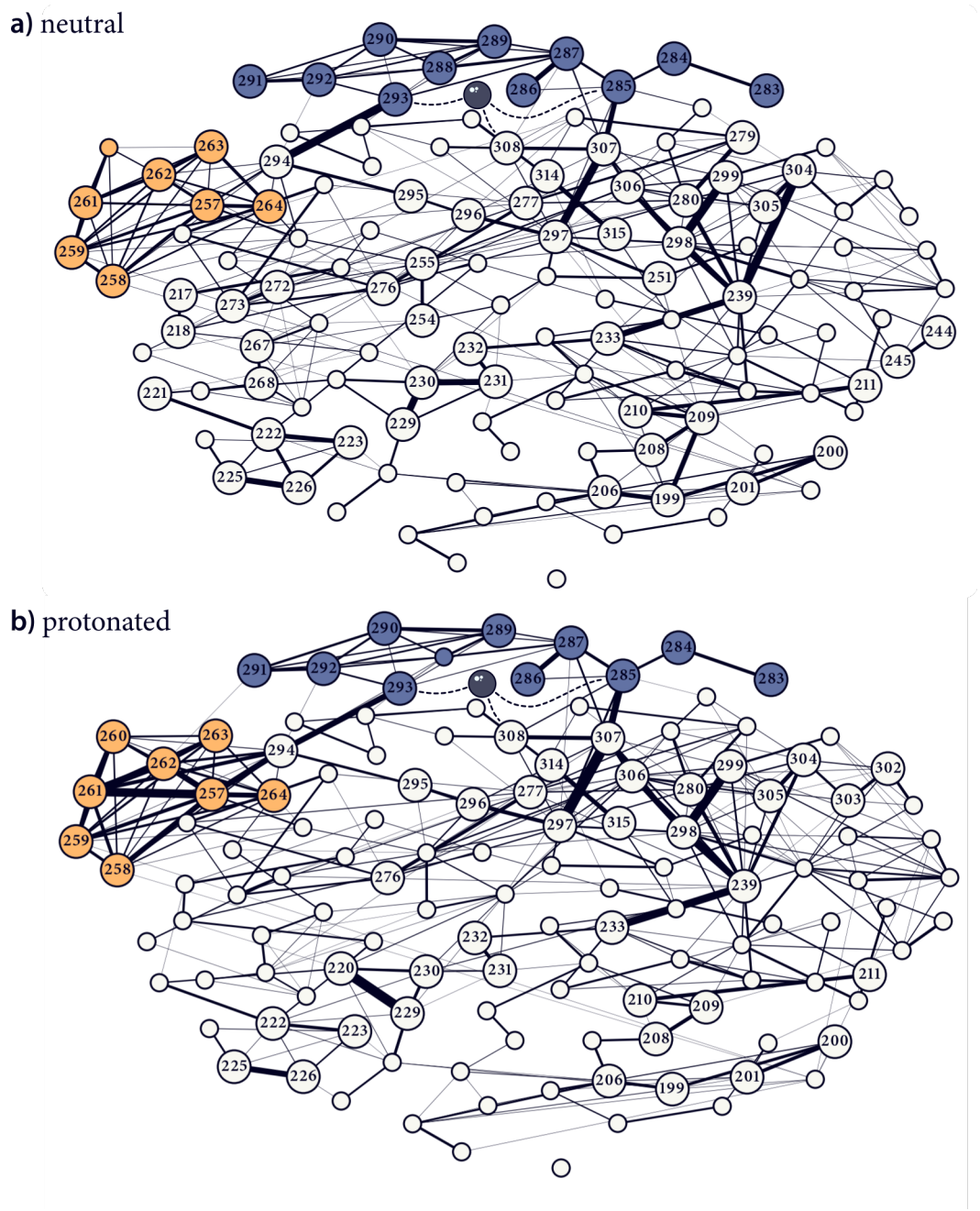
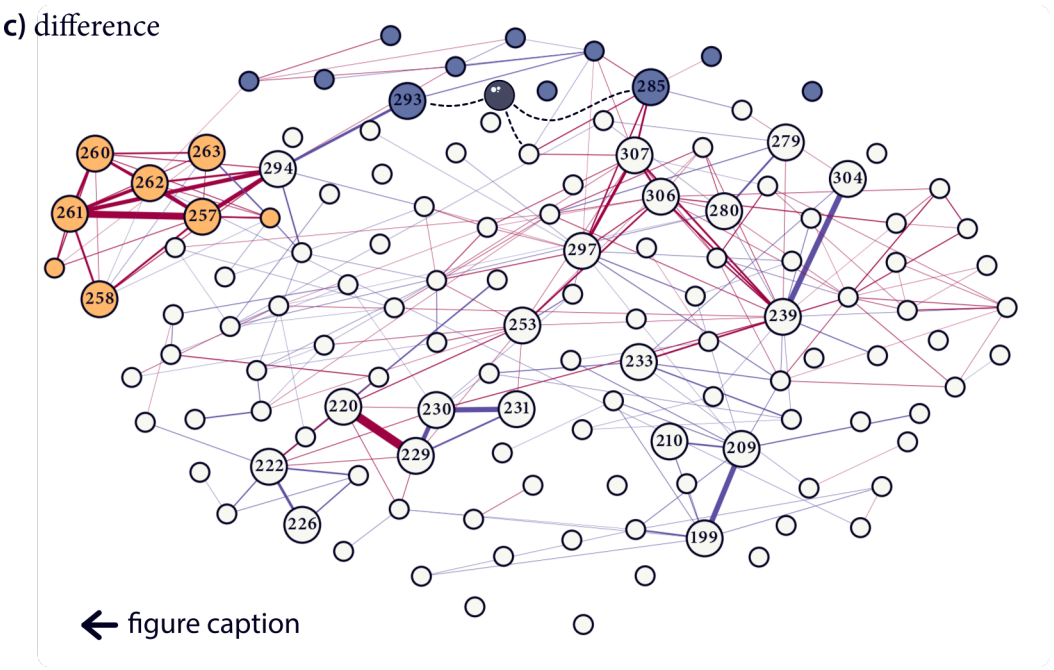


Figure 11.6 **Mutual information graphs for calcium-bound langerin**
Networks for **a)** the neutral and **b)** the histidine protonated state. **c)** Difference plot where purple and red edges highlight communications that are more pronounced before and after the protonation, respectively. The node positions were found using a spring layout, constrained with C_α distances in the crystal structure 3p5g, using atomic positions of the same as initial guess (the plot was created using `networks.draw`). Residues of the short-loop are coloured in orange and those of the long-loop in blue. Residues mutually connected to other residues above a threshold of 0.12 are highlighted with bigger nodes and ID labels.



Interestingly, another area of intensive communication involves a series of residues starting at the calcium-binding site (E285, N307) and proliferating via W306 of the WND motif to the β_3-strand (N297) and the α_2- (Q239) and α_3-helices (F280). Moreover, communication in this area is also affected by the histidine protonation. Figure 11.7 shows an illustration of this network. The identified protein region resembles remarkably well a route to what has been identified and targeted as a cryptic binding pocket in DC-SIGN, located between α_2-helix and β_0/β_1-strand.[322, 323] A similar allosteric regulation is suspected for murine langerin.[324] It might be that this is a common communication channel in the CRD fold.

further allosteric pathways?

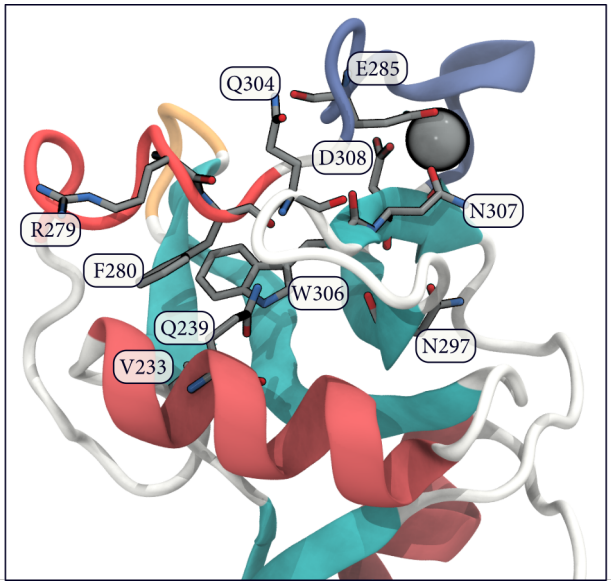


Figure 11.7 Potential further allosteric communication in langerin The MI differences between the neutral and the histidine protonated langerin state as well as the absolute magnitude of the scores in both states hints towards a network of connected residues that has not been explicitly part of our analysis yet. It involves the calcium-binding site, the β_3-strand and the α_2- and α_3-helices.

Still inconclusive remains the role of H229. In the MI networks, this histidine and its surroundings appear as largely isolated, although locally the protonation seems to have a very strong effect. Long range communication between here and the previously described sub-network via residues 230, 231, 232, 233, and 239 is still noticeably present and seems to be weakly disturbed upon protonation as well. This becomes a bit more clear in figure 11.8 that shows the MI networks reduced to the most important connections in terms of MSTs (see also section 7.2).

a) neutral **b)** protonated

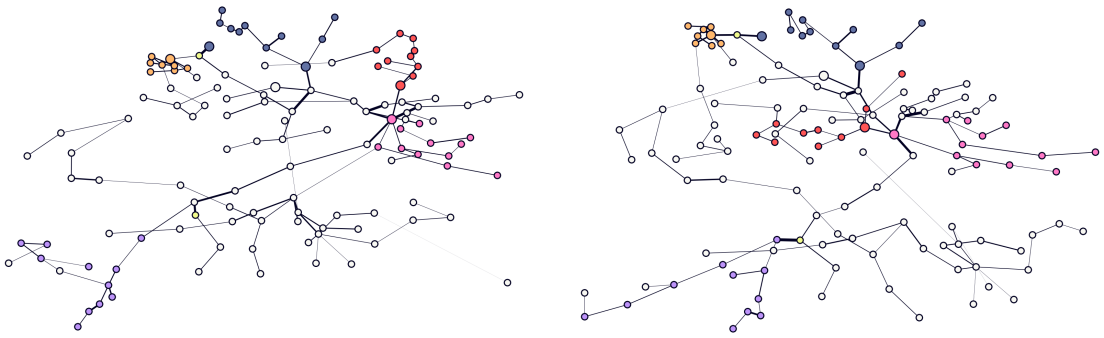


Figure 11.8 Minimum spanning trees of mutual information graphs
Trees connecting all nodes in the networks by using only the most important connections (found using `networkx.algorithms.minimum_spanning_tree`). Node positions were determined from a spring layout constrained by the MI scores and with positions from figure 11.6 as initial guess. Interesting regions are highlighted by colour: short-loop (257 to 264) in orange, long-loop (283 to 293) in blue, H229/H294 in yellow, α_1-helix (216 to 226) in purple, α_2-helix (235 to 246) in pink, α_3-helix (273 to 282) in red.

spectral clustering

Taking the analysis of the MI networks one step further, it is possible to cluster this kind of data when the MI matrix is interpreted as a pairwise similarity matrix. We can for example use spectral clustering (see section 14.2) to find a separation of the graphs into sub-graphs based on a normalised cut, i.e. a cut along connections of low importance leaving two relatively balanced sub-networks. Figure 11.9 shows the result of this when two clusters each are considered for both the neutral and the protonated langerin MI graphs. It is apparent that the changes in the underlying network give rise to a different preferred partitioning for each case. In the neutral state, short-loop, long-loop and the helix regions highlighted in figure 11.7 are part of the same cluster, separated from the lower protein segment. After the protonation, the assignment changes and part of the calcium-binding site including E285 and D308 is assigned to the same cluster as the lower protein together with the high-communication region around the α_2/α_3-helices. While it is not immediately clear, how to interpret this in terms of a functional mechanism, it emphasises that the allosteric regulation in langerin upon histidine protonation may yet be more complex and global than what we considered up to here. The conformational transition involving K257 and D308 may only be a piece of the overall puzzle.

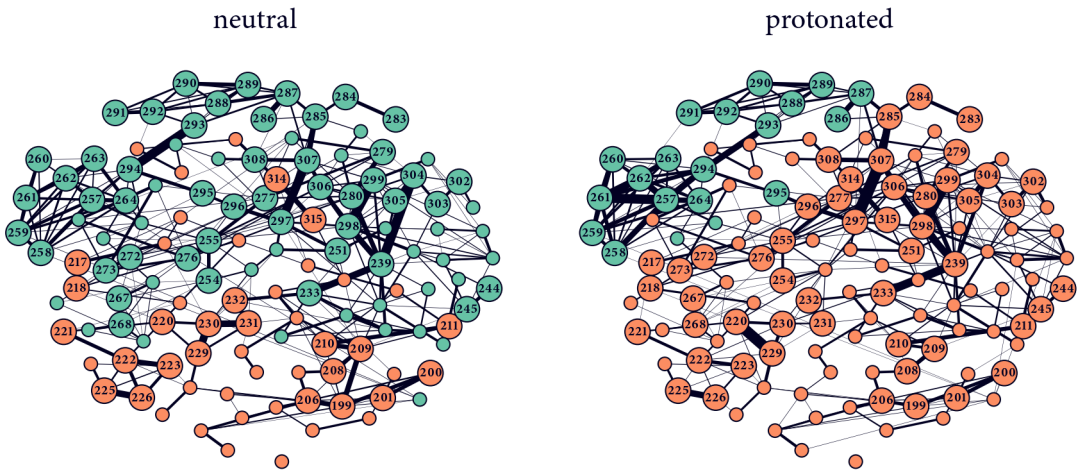


Figure 11.9 Spectral clustering of langerin MI graph
The clustering was performed with `sklearn.cluster.SpectralClustering`. Graph nodes are coloured by their cluster label assignments.

11.2 Polarisable force field simulations

As mentioned during the recap of our langerin study, the conventional MD model may be suboptimal for langerin with respect to the present calcium(II) ion. This could potentially be improved using a polarisable force field description that accounts in one way or another for redistributions of partial charges (see section 4.3). To make an explorative step into this direction, MD simulations using a polarisable AMOEBA force field (AMOEBA2018[133]) have been carried out for the histidine protonated, calcium bound langerin state. The setup of an AMOEBA simulation is not much different to a standard MD simulation setup as described in chapter 12.[2] Based on 600 ns simulation time,[3] we can already get a feeling for how the polarisable description affects this system. One simulation each has been started from the langerin crystal structure 3p5g, and two representative frames of the *G* and the *O* set (compare figure 11.3). The acquired data set is too small, though, to draw premature general conclusions and should mainly serve as an indicator whether the simulations are sensible in principle. As the polarisable description constitutes a conceptual improvement, we kind of expect to see minor changes that reflect the true behaviour of the system more correctly. This concerns especially the calcium-binding site. Very large differences, however, could be a sign for technical issues.

test simulations

Figure 11.10 shows selected basic analyses of the AMOEBA simulations in comparison to the non-polarisable ones. It can be recognised right away that the system's behaviour changes quite a bit. From the short-loop/long-loop distance (figure 11.10a), it is apparent that the previously observed strong preference for open or closed loop settings is weakened. The highest population samples intermediate loop distances, potentially altering the view of what is going on in the inter-loop region substantially. Relative populations should, however, not be over-interpreted at this point because the simulation time is short and the distribution is most likely not converged. More important from a stability perspective is here that the distance value range is not altered overall.

structural analysis

[2] See also `simtk.org/projects/openmm-amoeba` for an OpenMM setup example script

[3] The initial test runs were kept at a minimum because the polarisable simulation is considerably more expensive. On 1 GPU + 1 CPU we obtain about 20 ns/day using OpenMM 7.7 (that is a factor of ten times slower than a comparable simulation using a classical force field).[362]

A striking observation, on the other hand, is that the backbone flexibility (figure 11.10b) as measured by the RMSF of C_α atoms is clearly increased through the inclusion of polarisation. This includes in particular but not exclusively the helical regions. Interestingly, the only primary residue in the calcium binding site affected by this is E293. The secondary structure remains, however, still well intact. Furthermore, the backbone RMSD with respect to the crystal structure (figure 11.10c) appears to be stable over the course of all simulations, indicating no large scale conformational change.

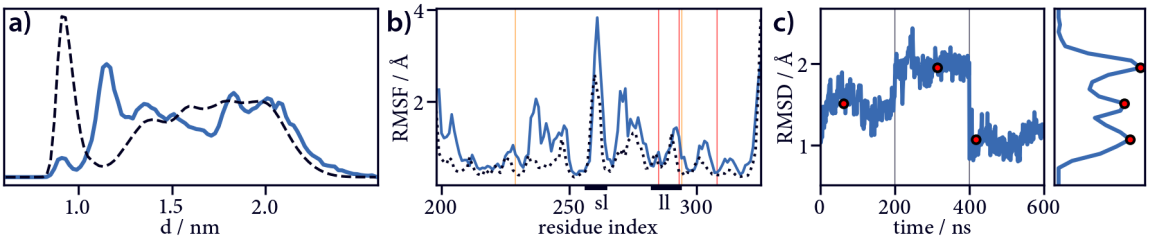


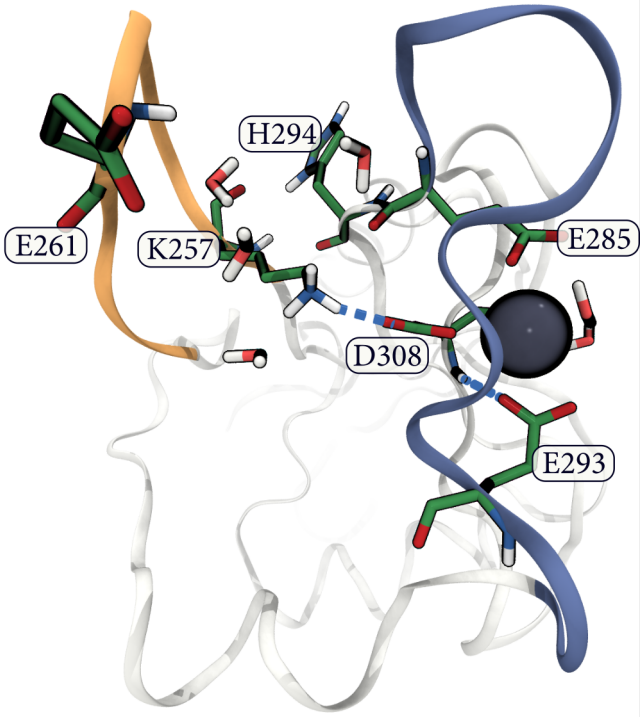
Figure 11.10 Basic feature analysis for histidine protonated langerin using the AMOEBA2018 force field
a) Short-loop/long-loop distance measured as the M260–G290 C_α distance (compare figure 11.2). While covering the same value range, the polarisable simulation (blue) notably samples loop settings around 1.2 nm. These have been virtually unobserved in the non-polarisable picture (black, dashed). **b)** C_α RMSFs reveal increased flexibilities, in particular with regard to the α_2- and α_3-helix, the segment around H229, the short-loop, the late long-loop, and the β_3-strand. Note, that E285 and D308 of the calcium-binding site are not affected in contrast to E293 (red vertical lines, histidines marked with orange lines). **c)** The backbone RMSD with respect to the crystal structure is unremarkable and does not indicate major conformational transitions. Black vertical lines mark the separation of individual simulation replicas. The maxima of the RMSD distribution (red dots) could be used as sensible seeds for new simulations.

flexibility through polarisation?

Protein flexibility is a difficult topic and the effect of included polarisation is not fully conclusive yet (for a general discussion see section 4.3). There is, however, no serious reason to mistrust the AMOEBA simulation of langerin fundamentally, although the possibility has to be considered that the observed flexibility increase can be at least partially a simulation artefact. At the same time, the non-polarisable picture could as well be too rigid. A better assertion of the effect that the choice of force field might have on the system could involve separate simulations with other state-of-the-art classical force fields (e.g. AMBER99SB-disp,[138] AMBER19SB,[65] or CHARMM36m[363]) or a different polarisable model (e.g. Drude2019[131]). A concise validation of which force field offers the most realistic description in this specific case, is essentially hard to achieve without benchmarking against experimental results, which are not readily available. The only indication that we currently have is that in NMR measurements, residues of the long-loop could not be resolved, even in the presence of access calcium strongly favouring the calcium-bound state.[337] However, this bound state appears very inflexible in our AMBER99SB simulations. Maybe, the flexibility of the protein is indeed actually larger at least with respect to this region.

K257–D308

Despite the first-off inspection of the langerin AMOEBA simulations presented in figure 11.10, it is of course a major concern whether the K257–D308 bonded interaction of the *G* set is representatively sampled in these simulation. The minimum distance between the two side chains (amine hydrogens and carboxy oxygens) is depicted in figure 11.2 and shows indeed a good agreement with the previous simulation. Close arrangements are relatively low populated but this should again not be over-interpreted due to the limited sampling. It is, however, notable that wider distances are sampled as well, which have not been seen in the non-polarisable description. These wider distances partly correspond to conformations where K257 undergoes little interaction with any other protein residue



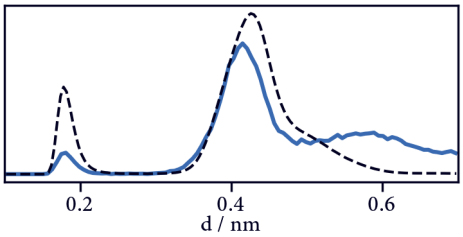


Figure 11.11 K257–D308 hydrogen bonded conformation (AMOEBA2018) As shown in the plot above, langerin samples comparable K257–D308 side chain distances in the polarisable (blue) and in the non-polarisable (black, dashed) description. Notably, wider distances (around 0.6 nm) are sampled with polarisation, which where previously unobserved. The example structure was observed after about 100 ns and has a live time of several nanoseconds. Water molecules at most 0.4 nm away from the calcium ion or the lysine amine hydrogens are shown as well. Note the bidental binding mode of E285 towards the calcium ion, an accurate description of which can be crucial for a correct modelling of the calcium-binding site.[364]

and reaches out into the solvent. This might be connected to the polarisable water description (see comment further below), favouring a lysine water interaction. Also notable is that the lysine-aspartate interaction appears not to be strongly correlated to the H294–E261 hydrogen bonded interaction and the implied characteristic short-loop orientation of the *G* set. It is consistent with the experimental data (see figure 9 in our paper[347]) that suggests that the H294–E261 interaction is not important. These further signs of increased conformational flexibility could extent our view on what is going on in the inter-loop region of langerin rather substantially. Definite conclusions can not be drawn, though, without more simulation data and a comparison to the situation in the neutral protein. Structural differences with respect to the calcium-binding site have not been spotted in the first-order analysis but the polarisable description will be most likely a more faithful representation than the one given by the canonical point charge model. Not only energetically but also geometrically in terms of the coordination number (expectedly ~7 for calcium[365]) and acetate binding, there is little doubt that the current AMOEBA model constitutes an improvement.[364]

Commenting on the influence of water in our setup, another potentially very important difference between the polarisable and the previous simulations is the used water model. As also shown in figure 11.2, individual water molecules can coordinate to vacancies at the calcium-binding site or enter the loop-coupling zone. In our previous simulations we relied on the simple TIP3P water model in consistency with the used force field but this has its limitations even among 3-point water models.[69, 366] The AMOEBA force field comes with its own polarisable version of water.[129, 367, 368] Although there is still room for improvement on this side,[369, 370] the incorporated polarisation effects have a potentially big influence on the relative protein-solvent/calcium-solvent interactions.[141, 371] Explicit solvent effects have not been included so far in our analysis but an accurate water model would be the premise for their consideration.

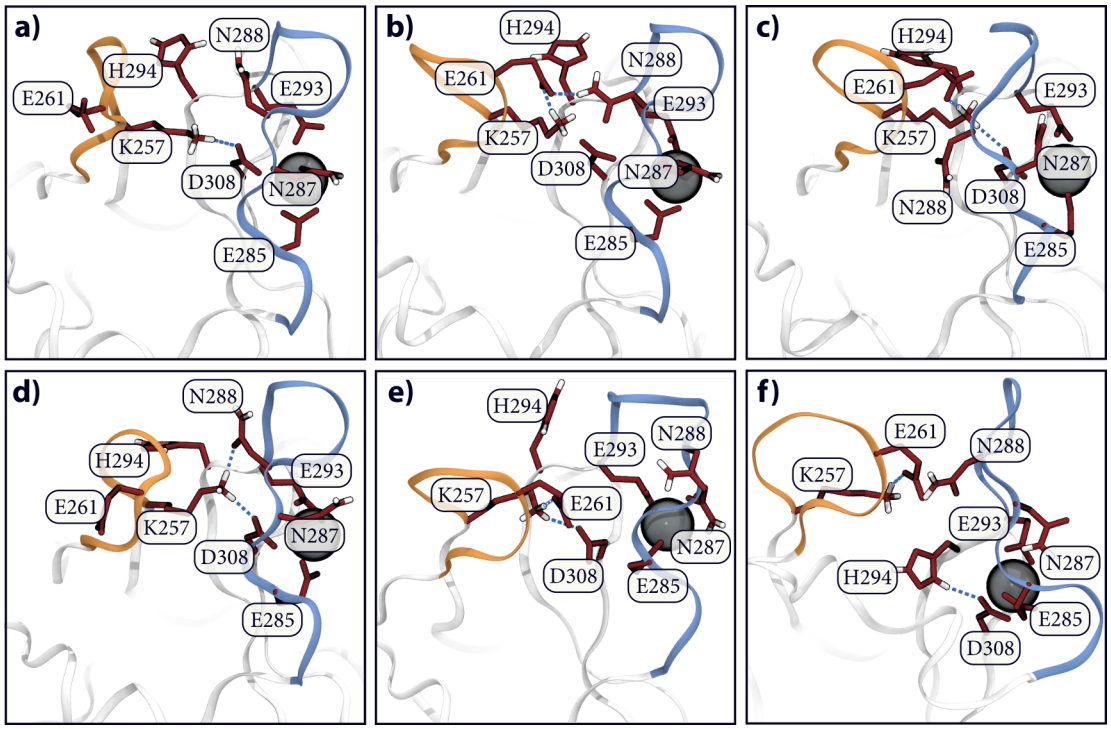


Figure 11.12 **Conformations for histidine protonated langerin using the AMOEBA2013 force field**
The sampled conformations resemble only partly what has been found in the non-polarisable simulations. **a)** A K257–D308 H-bond is frequently observed but strong correlation with H294–E261 as in the *G* set is not apparent. **b)** K257–E261/N288-E261 interactions as in the *O* set. **c)** Simultaneous interaction of K257 with E261 and D308, and **d)** of K257 with N288 and D308, which have not been observed in the non-polarisable description. **e)** D308 leaving its coordinative role in the bindings site, a weaker form of which is also noticeable in **b)**. **f)** Severe unfolding of the long-loop in conjunction with the α_3-helix which is likely a simulation artefact. Increased loop flexibility is also present in **c)** and **e)**.

As a negative example for when a polarisable AMOEBA simulation went wrong, figure 11.12 shows selected conformational snapshots from trajectories (2 µs over 10 replicas) using the older AMOEBA2013 force field.[132] These have been performed prior to the above presented AMOEBA2018 simulations. While on a first glance, the sampled structures look indeed interesting and also resemble in parts what has been observed in the non-polarisable simulations, the flexibility of the protein is increased to an extent that the long-loop region unfolds. This is very suspicious and also figure 11.12, showing backbone RMSDs with respect to the crystal structure, reveals that the simulations are probably not as stable as they should be. This counter example is mainly included here to emphasise that it is easy to get nonsensical simulation results although the run might finish without crashing. Also, in this case the temperature and potential energy evolutions during equilibration and production appear to be fine, not offering an explanation for the observed instability either. Both the AMOEBA setups are virtually identical except for the force field version. Extrapolated mutually induced polarisation was used, rendering the effect of the convergence cut-off (`mutualInducedTargetEpsilon` in OpenMM) for the otherwise employed iterative optimisation unimportant. In general, however, this setting might have a rather large effect on the accuracy of a simulation.

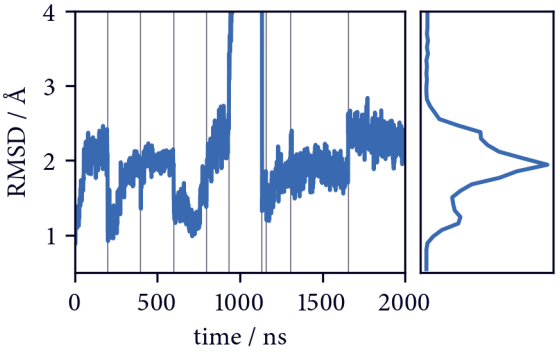


Figure 11.13 RMSD trajectories (AMOEBA2013) In contrast to the RMSD values shown in figure 11.10c for the AMOEBA2018 simulations, here at least 6 out of 10 replica exhibit instable behaviour. Black vertical lines mark the separation of individual simulation replicas.

{ 12 }

Simulation setup

Protocols for langerin and practical advice

The default preparative steps that are mandatory for the setup of a MD simulation are not much different for members of the C-type lectin family then for any other proteinogenic molecular system in comparison. Nonetheless, the correct execution of these steps is crucial and can be tricky at times although they are widely standardised. In this chapter, I would like to discuss practically how the simulations for this work have been performed. In a nutshell, the MD setup procedure includes the following parts: 1) structure inspection and planning, 2) structure cleanup and preparation 3) energy minimisation, and 4) equilibration. Note that the slightly specialised setups for the performed steered-MD and polarisable simulations are not separately addressed here.

basic steps

The simulation software packages we used for these steps are GROMACS[50, 107, 372–376] and OpenMM[362]. Basic structure manipulations can be done directly by altering the respective structure text files. Beyond this we used mostly GROMACS tools for these tasks but it is worth mentioning that the PDBFixer from the OpenMM cosmos can be helpful as well, in particular for adding missing protein residues. Structure visualisations are done with VMD[377] or with NGLview[378] directly in Jupyter notebooks. The NGLview package is nice because a lot of MD setup and analysis tasks can be combined in an interactive Python session without breaking the workflow but VMD offers superior functionality especially for image production via the Tachyon renderer although its handling is rather clumsy.

programs

12.1 Structure inspection

For human langerin, multiple crystal structures are available from the RSCB protein data bank, released between 2009 and 2018 (see table 12.1). With the exception of 3KQG that covers a trimeric assembly of the langerin extracellular domain including a part of the α-helical neck region, the structures contain only the CRD. All of the structures were obtained in the presence of calcium at neutral pH or above and contain a calcium ion bound at the canonic site. The only langerin mutant listet here is 4AK8 because it was obtained at a lower pH of 6, but there is also a K313I and a K313I/N288D double mutant (4N34 to 4N38). Many published structures contain sure ligands bound via the calcium binding site. Besides human langerin, crystal structures for the murine variant are available as well (5M62, 5K8Y).

crystal structures

The considered structures show remarkably little differences between each other. Figure 12.1 illustrates this in an overlay. Chain A in the trimeric crystal structure 3KQG is the only one that stands out in terms of a short-loop arrangement closed towards the long-loop. In the other structures the short-loop adopts a wider opened conformation. A K257–H294 hydrogen bonded interaction can be assumed for all structures with the open loop setting, indicating an ε-protonation of H294 and a neutral side chain—interestingly also for the F241L mutant structure that was obtained at a lower

structure overview

pH. Free side chain orientation vary expectedly to a larger extent. We did not detect any structurally critical crystal water molecules.

Table 12.1 Langerin RCSB PDB structure overview

ID	main author	pH	year	resolution / Å	comment
5G6U	Porkolab, V.	7	2018	1.844	
4N33	Feinberg, H.	7	2013	1.85	
4N32	Feinberg, H.	7	2013	1.75	
4AK8	Chabrol, E.	6	2013	1.4	F241L
3P5I	Feinberg, H.	7	2010	1.8	
3P5H	Feinberg, H.	7	2010	1.6051	
3P5G	Feinberg, H.	7	2010	1.6027	
3P5F	Feinberg, H.	7	2010	1.7501	
3P5E	Feinberg, H.	7	2010	1.7012	
3P5D	Feinberg, H.	7	2010	1.8013	
3P7H	Skerra, A.	6.9	2010	2.3	
3P7G	Skerra, A.	6.9	2010	1.5	
3P7F	Skerra, A.	6.9	2010	2.5	
3KQG	Feinberg, H.	8	2010	2.3	trimer
3C22	Thepaut, M.	7	2009	1.5	

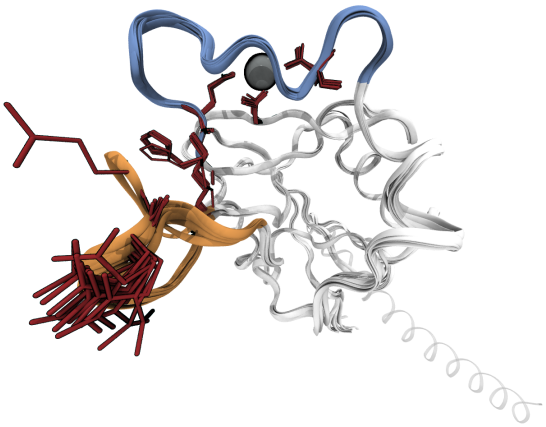


Figure 12.1 Langerin PDB structure overlay Available crystal structures listed in table 12.1 aligned on top of each other. Side chains of residues K257, E261, E285, E293, H294, and D308 highlighted in red. The short-loop (residues 258 to 264) representation is coloured in orange and that of the long-loop (residues 283 to 294) in blue. Calcium(II) in the binding site drawn as a grey sphere.

structure alignment

As a side note, an alignment of two structures in VMD can be achieved with a short custom TCL function by selecting a common set of positionally inflexible atoms (here the backbone atoms of residues 308, 256, and 239) in the reference and in the target structure, computing a translation matrix for the superimposition of both structures, and moving all atoms in the target structure by applying this translation. Such a function can be directly defined in the VMD console or sourced from a respective file.

```
proc align {reference target} {
  # expects two molelue IDs
  set sel_a [atomselect $reference {resid 308 256 239 and backbone}]
  set sel_b [atomselect $target {resid 308 256 239 and backbone}]
  set sel_b_all [atomselect $target all]
  set M [measure fit $sel_b $sel_a]
  $sel_b_all move $M
}
```

For the initial seeding in a MD simulation of the langerin CRD, the majority of the shown structures is basically equivalent and we settled predominantly on 3P5G because of the relatively recent publica-

tion date and overall good scores (resolution: 1.60 Å, R-value free: 0.219, low amount of clashes and outliers). In the relevant residue range (G198 to P325), chain A of the structure is complete. It contains alternative atom positions only for the S277 side chain (figure 12.2b), where it makes presumably very little difference which one is chosen. Figure 12.2a gives an overview over titrable amino acids in the protein. In the pH range of 6 to 7.4, we can generally assume the side chains of aspartic and glutamic acid to be deprotonated (negatively charged), lysine protonated (positively charged), and tyrosine protonated (neutral) as well. Histidine may change its protonation state from a single δ- or ε-protonation (neutral), where the latter is normally preferred if there are no influencing factors, to a complete protonation (positively charged). There a no free cysteine side chains that are not bound in disulfide bridges in the protein.

3P5G

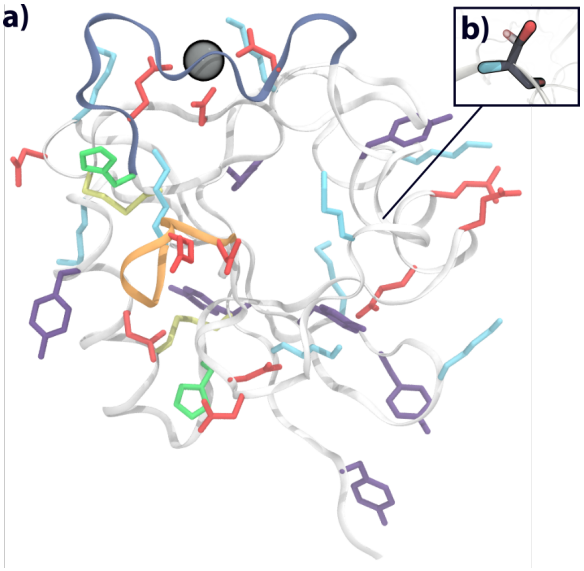


Figure 12.2 Titrable side chains in the langerin CRD
Langerin crystal structure 3P5G with **a)** side chains of titrable residues coloured by residue type from acidic to basic: aspartic and glutamic acid (red), histidine (green), cystein (yellow), tyrosin (purple), lysine (blue). **b)** Alternative atom locations for S277.

12.2 Structure preparation

The first step in setting up the raw crystal structure for a simulation is to add missing hydrogen atoms and to define the topology of a specific protonation state. This is conveniently done with the GROMACS tool `pdb2gmx` even if GROMACS is not been used to drive the actual simulation. The execution of this tool on the command-line could look like this:

```
$ gmx pdb2gmx -f ../conf.pdb -ignh -his -merge all
```

Here, I assume that roughly the following directory layout is used for the structure preparation with a dedicated root directory for this kind of work within a bigger project:

directory layout

```
$SETUP_DIR
└── 3p5g
    ├── 3p5g.pdb
    ├── conf.pdb
    └── h3
        ├── conf.gro
        ├── topol.top
        └── ...
```

As a practical recommendation, it makes sense to create a new sub-directory for each preparation step, so that a certain operation has its input files in a parent and its output files in a child directory. A

separation of input and output on progressive directory levels does not only make the order of the applied steps obvious—which would not be the case if all files are cramped successively within the same folder—but also allows a quick redo of operations if things go wrong and easy branching if the setup steps should be altered in the future.

The above stated GROMACS command is in this context executed from the `h3`[1] folder highlighted in red, which marks the addition of hydrogens to an input structure (`-f ../conf.pdb`) and the setting of a specific protonation state. Its main output will be the `conf.gro` (the modified structure) and `topol.top` (the systems GROMACS-topology) files where the default output location can be modified with `-o` and `-p` flags. The `-ignh` option ensures that any hydrogen atoms in the input file are ignored prior to the automatic addition of these atoms to avoid that they interfere with this step. To interactively set the protonation state of histidine residues, the `-his` flag is passed. Similar flags exist for aspartic and glutamic acid, and other titrable residues. If they are not set, GROMACS will prepare a default protonation state in each case, which for histidine is a single δ- or ε-protonation depending on possible hydrogen bonded interactions in the environment that may stabilise one or the other state. As a further hint, the `-merge all` option can be specified to combine all molecules in the input structure into one chain, which limits the amount of produced output files. Otherwise GROMACS will for example create separate topology files for our protein and the calcium ion. If it is in contrast intended to split the protein and calcium into two separate moleculetypes, `-merge no` should be used, which is the default. Here we deliberately chose to model the termini of the protein in their charged form (NH_3^+ and COO^-) to stay consistent with the already existing langerin simulations. This default behaviour can be changed with the `-ter` flag. For a bigger protein that is stably folded the modelled state of the termini is assumed to have a minor influence even if in the real system their state may be different because the modelled sequence segment is only a subset of the complete protein sequence. In general it should be considered, though, to use capped termini if this agrees better with the real system, which depending on the used force field would require to add respective terminal residues to the structure. This is especially the case if unwanted interactions between the termini are likely. It should also be kept in mind here, that a real system under biological conditions may differ from a real system subjected to experiments under laboratory conditions. Because `pdb2gmx` is meant to be a setup tool for simulations with GROMACS, the user is also required to name a force field and a water model, which, however, can still be altered at a later stage and completely ignored if GROMACS is not used for the simulation itself. It should also be mentioned that this tool is only suited for the preparation of protein systems for which the set of standard amino acids are known. In other cases, the creation of structure and topology files can be more complicated and often needs to be done separately and manually.

pdb2gmx flags

limitations

In the above example, the input file to the `pdb2gmx` command used here was not `3p5g.pdb` (a crystal structure as obtained from a data base) but `conf.pdb`, which implies that there may have been necessary pre-processing steps. Raw crystal structure may contain multiple copies (chains) of the same protein, additional ligand molecules, crystal water, and potential short-comings that need to be addressed prior to the automatic setup. For one thing, if alternative atom positions are present in the structure as in 3P5G for SER277 (see figure 12.2), GROMACS will silently choose one of these, which may not be intended. A selection of the interesting individual entries in a crystal structure is best done with a short script that can be stored alongside the structure to record what has been done (or at least a description if the pre-processing has been achieved in a different way). The following lines do for example extract only the relevant parts of `3p5g.pdb` (ignoring water and ligands, and keeping only protein chain A including calcium) into `conf.pdb` leaving the meta information intact.[2]

Structure pre-processing

[1] Identifier for the H229/H294 protonated state. For details see the SI of [347]

[2] For details on the general structure of PDB files see `wwpdb.org/documentation/file-format`

```
allowed_res_ids = set(range(198, 326))
allowed_res_ids.add(500)  # calcium
with open("3p5g.pdb") as in_file:
  with open("conf.pdb", "w") as out_file:
    for line in in_file:
      if not line.startswith(("ATOM", "TER", "HETATM")):
        out_file.write(line)
        continue
      if not line[21] == "A":  # chain
        continue
      if not int(line[22:26]) in allowed_res_ids:
        continue
      if line[16] == "B":  # altLoc
        continue
      if line[16] == "A":
        line = line[:16] + " " + line[17:]
      out_file.write(line)
```

structure mutation

Somtimes, crystal structures need to be altered to generate other systems of interest. For langerin for example, we built the calcium unbound state in lack of a crystal structure from the calcium bound state by just deleting the calcium ion from the structure. For the simulation of langerin mutants, the respective amino acid side chains have been modified as well. Mutations to alanine are particularly simple because the side chains of other residues can essentially be removed entirely up to the C_β-atom.

```
ATOM    273  N   HIS  ->  ATOM    273  N   ALA
ATOM    274  CA  HIS  ->  ATOM    274  CA  ALA
ATOM    275  C   HIS  ->  ATOM    275  C   ALA
ATOM    276  O   HIS  ->  ATOM    276  O   ALA
ATOM    277  CB  HIS  ->  ATOM    277  CB  ALA
ATOM    278  CG  HIS  ->  x
ATOM    279  ND1 HIS  ->  x
ATOM    280  CD2 HIS  ->  x
ATOM    281  CE1 HIS  ->  x
ATOM    282  NE2 HIS  ->  x
```

Non-consecutive atom IDs as a result of the deletion of atom entries are usually not a problem for setup tools like `pdb2gmx` but if required this is easy to fix by looping through the file and renumbering ATOM, HETATM, and TER entries in position 7 to 11. Similarly, mutation of glutamate to aspartate is straightforward as it only requires to delete the C_γ-atom (or alternatively C_δ-atom) and to rename the carboxylate oxygen atoms. This introduces some conformational stress in terms of the now elongated C_β–C_γ bond but with careful structure minimisation this is normally solvable.

```
ATOM    200  N   GLU  ->  ATOM    200  N   ASP
ATOM    201  CA  GLU  ->  ATOM    201  CA  ASP
ATOM    202  C   GLU  ->  ATOM    202  C   ASP
ATOM    203  O   GLU  ->  ATOM    203  O   ASP
ATOM    204  CB  GLU  ->  ATOM    204  CB  ASP
ATOM    205  CG  GLU  ->  x
ATOM    206  CD  GLU  ->  ATOM    205  CG  ASP
ATOM    207  OE1 GLU  ->  ATOM    207  OD1 ASP
ATOM    208  OE2 GLU  ->  ATOM    208  OD2 ASP
```

For more complex mutations that involve an actual addition of atoms at sensible positions, it can help to have a template of the target amino acid to be aligned with respect to the backbone on top of the residue that should be mutated (see the alignment snippet above for such an operation in VMD) and to copy the complete target acid with the obtained coordinates into the file. Alternatively, side chains can be modelled using software like the VMD molefacture plugin.

Once a particular structure for a molecular system including all desired protonations has been set up, the next step usually is to add solvent (normally water) unless a treatment in vacuum should be done. This necessitates in general the prior definition of a simulation box that can be filled with solvent molecules. GROMACS provides another tool for this step that is used like:

```
$ gmx editconf -f ../conf.gro -o box.gro -d 1 -princ -c -bt cubic
```

simulation box

Again, it is assumed that a new sub-folder (e.g. `d1_cubic`) is created for this next step from which the command is executed. Here it has to be ensured that the box is big enough so that the solute is adequately distanced from the box borders, which becomes important if PBCs are used at a later stage (see section 4.4). A distance of 1 nm as set with the flag `-d 1` is generally considered sufficient with normal cut-off values for the calculation of short-range non-bonded interactions. If larger structural changes are expected to happen during a simulation, the box may, however, need to be much larger. This might be for example the case in steered-MD experiments where the distance between groups that are pulled away from each other is expected to increase considerably. GROMACS provides several box geometries that can be chosen via the `-bt` option. While it can be beneficial in terms of efficiency to select a complex box type like `dodecahedron`, let's stick here for simplicity with the cube. The other used flags `-princ` and `-c` align the principle axis of the protein with the z-axis of the box and center the molecule in it, which is of rather cosmetic reasons if a special orientation of the system is not required.

A created simulation box can be now filled with solvent.

```
$ gmx solvate -cp ../box.gro -cs spc216.gro -o solv.gro -p topol.top
```

solvation

This requires the specification of the solute structure (`-cp ../box.gro`) and a template of a (ideally pre-equilibrated) solvent box via `-cs` were `spc216.gro` is a standard water box provided by GROMACS.[3] Here it becomes useful if the solvation step is carried out in its own sub-directory because GROMACS modifies the topology of the system (input `-p topol.top`) by adding the number of solvent molecules to the respective `[ molecules ]` section. It is advised to copy the topology without solvent to the solvation directory and to modify it there, leaving the original untouched GROMACS itself by default creates backup files if files are altered but keeping the overview over those backups and restoring files in question from the correct backup can become quite messy.

charge neutralisation

To finalise the setup, we can in a last step add additional counter ions to give the system a neutral net charge. The wild type langerin CRD in a default protonation state has an equal number of positively and negatively charged side chains so that it is already neutralised by itself but the presence of the calcium ion introduces a total charge of +2. Side chain protonations of the two histidine residues increase this to a number of +4. We can use yet another GROMACS tool to for example replace a few water molecules with chloride in these cases. The working of the `genion` tool is a bit counter-intuitive because it requires us to run a dummy simulation. So we call the GROMACS pre-processor to create a run input file (`ions.tpr`) with an empty run parameter file (`dummy.mdp`) first.

```
gmx grompp -f dummy.mdp -c ../solv.gro -p topol.top -o ions.tpr
gmx genion -s ions.tpr -p topol.top -o ions.gro \
  -nname CL -pname CA -neutral
```

Again, the tool will modify the system's topology by adding the number of counter-ions. The `-nname` and `-pname` flags control which type of ions are used, and the `-neutral` option tells the program to add as many ions needed to neutralise the system, which is more robust than giving it a concrete number of ions. In our langerin setup, we avoided the additional complexity to model a physiological salt concentration beyond the plain neutralisation of the system. With one equivalent $CaCl_2$ per

[3] Other solvent templates may for example be obtained from `virtualchemistry.org`

simulation box and an approximate box volume of 300 nm^3 (which corresponds to a cubic box length of roughly 7 nm) we are in a salt concentration regime of 50 mM. This is already very high compared to the realistically expected calcium concentration (about 20 mM) although not too high in comparison with typical NaCl (about 150 mM) or KCl (about 260 mM) concentrations.[379]

Now, the system is at a preparation stage where the last created file `ions.gro` is ready to serve as a starting structure. It can be further processed with GROMACS but if OpenMM should be used for the following computations this is possible as well. There are basically two options do this. First, the structure could be converted to a PDB file (using the GROMACS tool `trjconv`) that can then be loaded in an OpenMM script using the `openmm.app.PDBFile` reader. Note, however, that this ports only the structure while the topology information is lost, which is fine if this should be set up with OpenMM anyways. Otherwise, to keep the GROMACS topology including a potentially specified force field, use

OpenMM

```
from openmm import app
gro = app.GromacsGroFile('ions.gro')
top = app.GromacsTopFile(
  'topol.top',
  periodicBoxVectors=gro.getPeriodicBoxVectors(),
  includeDir=DATA_PREFIX + '/share/gromacs/top'
  )
```

where `DATA_PREFIX` is the location of the shared folder of the installed GROMACS version under which for example the force field definitions are found.

12.3 Energy minimisation

After the basic setup of a molecular system as described in the last section, the obtained structures are in a probably high energetic state. The addition of solvent around the solute may have introduced strain and minor clashes and the conformation of the solute as obtained from a crystal structure is likely not favourable in the same way in solution. Starting a simulation right away from such a state may lead to computational instabilities because of potentially very large force acting on individual atoms, and can make the system *blow up*. In order to avoid this, a relaxation of the structure to a local energy minimum should be performed. If the structure was not modified significantly before the solvation, it is usually sufficient to minimize only after it has been solvated. A minimisation of the solute alone in vacuum can actually make things worse since a minimum structure in the absence of any surroundings may even be further away from the favourable state in solution than the crystal structure minimum. On the other hand, if severe modifications have been done (e.g. a critical residue mutation), a minimisation of the protein alone may be advisable. It is possible to freeze certain atoms during a minimisation via positional restrains. This can be used to specifically optimise only a certain part of a molecule (e.g. only a mutation site) while avoiding larger rearrangements in other parts. Position restrains are also an option for edge cases were a direct minimisation of a system in solution fails. They can for example be used to hold the solute molecule in place while only the solvent around it is minimised first, which is then followed by a minimisation of the complete system. The langerin CRD is a thankful system in this regard and its minimisation did not require an elaborate approach.

minimisation strategies

position restraints

There are different minimisation algorithms available to chose from. The *steepest descent* method that is implemented in GROMACS does just fine in most cases. For a more rigorous optimisation the *conjugate gradient* method is available as well. A combination of both, an initial steepest descent run followed by conjugate gradient—possibly in combination with a minimisation approach in stages for parts of the system—can yield in general good results even in difficult cases. If very tight convergences

optimisation algorithms

are desired (as may be needed for normal mode analyses), GROMACS needs to be compiled in double precision, though. It should also be kept in mind that the standard GROMACS structure GRO-files have a limited precision of 10^{-3}nm and are not suited to reflect accurate convergences. In an MD context, higher precision is normally not needed but if it should not be lost, a g96 text file or a TRR trajectory file needs to be used. Another potent algorithm available in GROMACS and OpenMM is the *limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newtonian minimizer* (*L-BFGS*).[380]

To run a steepest descent minimisation in GROMACS, we need to specify parameters for this task. Settings that were sufficient to minimise the langerin CRD (at least using GROMACS 2019) are:

```
integrator = steep
emtol = 100     ; kJ / (mol nm)
emstep = 0.01   ; nm
nsteps = -1
coulombtype = PME
cut-off_scheme = verlet
```

basic settings

The two essential parameters here are `emtol`, setting the convergence criterion, and `emstep`, setting the initial maximum step size. If the maximum residual force on atoms of the system falls below `emtol` the run finishes successfully. In case of convergence issues `emstep` can be reduced. For tighter convergence, `emtol` can be lowered. Setting `nsteps = -1` does not limit the run in terms of a maximum number of iterations. For our langerin systems that comprises about 30,000 atoms (~2, 000 protein atoms plus ~9, 000 water molecules), the minimisation is inexpensive and converges routinely within 5, 000 iteration, which takes about 4 minutes in serial on workstations of the QCNet.

other settings

Further specified options are `coulombtype = PME` to chose a PME scheme to treat the electrostatic interactions instead of using a plain cut-off (see also section 4.2). The option `cut-off-scheme = verlet` controls what kind of neighbourlist implementation should be used to limit for which atom pairs short-range non-bonded interactions are calculated—here by using a buffered Verlet-list (see section 4.5). The chosen option is actually the default but this was not the case in older GROMACS versions. It is listed here, nonetheless, because it is quite critical. For energy minimisations (no dynamics), the neighbourlist is updated every time the energy is calculated, so it has no effective relevance for the computation. The other possible option, `cut-off-scheme = group`, makes use of charge groups. It was, however, deprecated already in GROMACS 5.1 but was not removed until GROMACS 2020, making GROMACS 2019 the last release in which this option is still available.

vacuum settings

Unfortunately, vacuum runs depend indirectly on the group scheme because the Verlet approach is not (yet) available for non-periodic systems. Therefore, for now they not supported by recent GROMACS versions. In earlier versions, the following altered settings can be used for a steepest descent minimisation in vacuum:

```
coulombtype = cut_off
rcoulomb = 0
rvdw = 0
cut-off_scheme = group
nstlist = 0
rlist = 0
ns_type = simple
pbc = no
```

The `coulombtype = cut_off` option is put back to its default because the PME scheme cannot be used in a non-periodic system, either. Periodicity needs to be explicitly turned of with `pbc = no`. The respective short-range cut-offs, `rcoulomb` and `rvdw`, should be set to 0 to essentially eliminate these truncations. Equally, `rlist = 0` should be used, which defines the short-range cut-off for the neighbourlist. In consequence, we do not need to update the neighbourlist and can set `nstlist =`

`0` so that it is only calculated once in the beginning. By default, the neighbourlist is calculated by searching for neighbours on a grid (`ns_type = grid`) but in a vacuum calculations without cut-offs a brute force approach should be used (`ns_type = simple`). It should also be noted that vacuum runs can only be executed in serial.

To prepare a minimisation with the above settings we call the GROMACS pre-processor by passing an input structure (`-c ../ions.gro`), its topology (`-p ../topol.top`), and the parameters (`-f steep.mdp`) to it. *execution*

```
$ gmx grompp -f steep.mdp -c ../ions.gro -p ../topol.top -o steep.tpr
```

The actual minimisation is then started from the created run input file by using the main GROMACS work horse `mdrun`.

```
$ gmx mdrun -nt 1 -deffnm steep -v -pin on
```

Here, `-nt 1` states that the calculation should be done using one single OpenMP thread, i.e. one CPU. If this flag is omitted, GROMACS guesses the number of threads to use and claims all available resources. Setting `-pin on` is generally advised if not all CPUs should be used on multi-core machines, which pins individual threads to fixed CPUs and avoids a swapping of threads.[4]

The `-v` flag leads to verbose output and `-deffnm steep` is actually a shortcut for

```
$ gmx mdrun -s steep.tpr \
  -o steep.trr -g steep.log -e steep.edr -c steep.gro
```

The output files produced are structure files (a normal text file `steep.gro` and a full precision binary file `steep.trr`), a log file `steep.log`, and an energy file (`steep.edr`). The energy file is similar to a log file that stores information about the run in a condensed binary form. Besides the minimised structure, it is the most important file to check if a minimisation was successful. Data can be extracted from this file into a text file using the GROMACS `energy` tool and can then be read[5] to plot for example the potential energy evolution of the system throughout the minimisation as shown in figure 12.3. *evaluation*

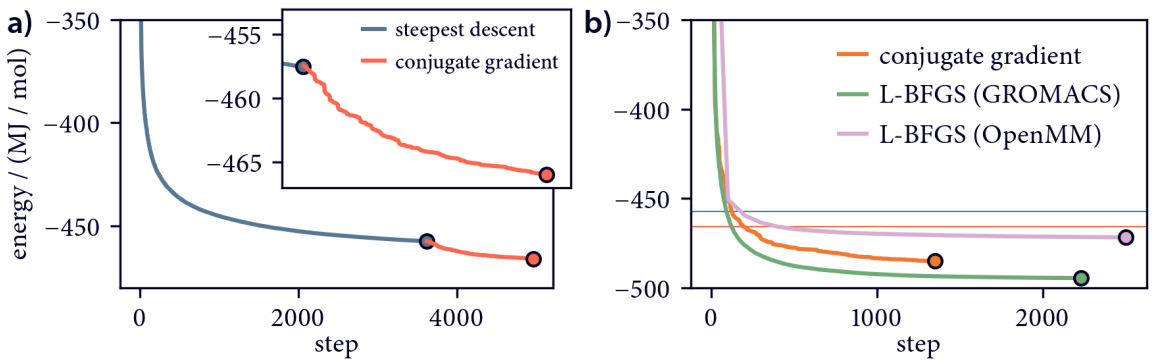


Figure 12.3 Potential energy minimisation

a) Steepest descent (GROMACS, `emtol = 100`) followed by conjugate gradient optimisation (GROMACS, `emtol = 1`). **b)** Direct conjugate gradient with flexible water versus L-BFGS (GROMACS, `emtol = 1`; OpenMM, `tolerance = 1`). Target energy values from runs in **a)** are marked with horizontal lines. Note that the set convergence criterion in GROMACS is lower than what can be reached by the machine in single precision. OpenMM does not give feedback in this regard. All runs use the Amber99sb-ildn force field bundled with GROMACS.

Normally, rough convergence as provided by the steepest decent step in figure 12.3a is sufficiently accurate to subject the system to further processing steps. It should be noted that the conjugate

[4] For more information on how to control the behaviour of `mdrun` in terms of resources and performance see also `manual.gromacs.org/documentation/current/user-guide/mdrun-performance.html`

[5] Alternatively, there is for example Panedr to read energy files into Pandas data frames in Python.

gradient implementation has a problem with constraints, including the SETTLE mechanism used for water. For the continued run from a pre-converged state, the optimisation is nonetheless successful with rigid water. LINCS constraints were used none in all runs. If conjugate gradient is used without the preliminary steepest decent step, convergence can be difficult, but in the present case it was possible to accurately minimize the system by explicitly using flexible water (`define = -DFLEXIBLE` in mdp-file). Interestingly, the direct approach seems to converge to a different energy minimum (compare figures 12.3 and 12.4). In OpenMM, the standard minimiser implements a L-BFGS scheme, wich can be called by

```
simulation.minimizeEnergy()
state = simulation.context.getState(getEnergy=True)
energy = state.getPotentialEnergy()
```

after setting up a `simulation` object for the system, which is the convenience route for directly using `openmm.LocalEnergyMinimizer.minimize()` with a given `openmm.Context`. Here, the keyword argument `tolerance` corresponds to the GROMACS setting `emtol` but convergence is reached if the root-mean-square error of forces on all atoms falls below it. If as many iterations as needed should be allowed, `maxIterations = 0` should be used. GROMACS provides an L-BFGS variant as well, which does not run in conjunction with constraints at all and is limited to serial execution. This implementation recommends to use a PME solver also for the Lennard-Jones interactions (`vdwtype = PME`). Structurally, the different optimisations yield consistent results as shown in figure 12.4, with deviations in side chain orientations that will not play a big role for the next processing steps.

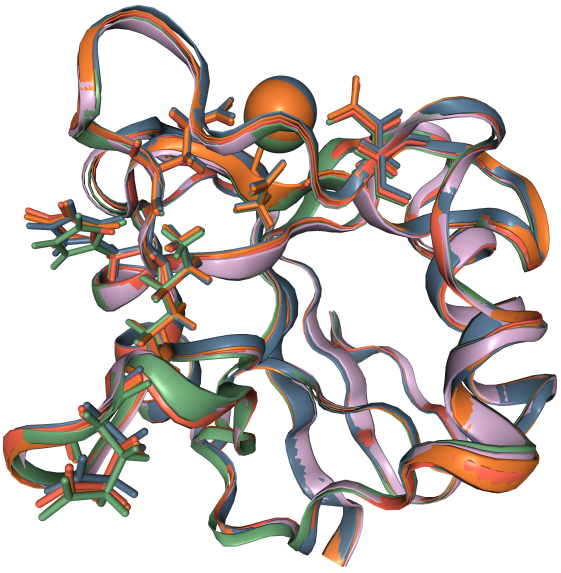


Figure 12.4 Minimised langerin structures Optimisation results for the langerin CRD using different algorithms for the same starting structure: steepest descent (blue), steepest descent + conjugate gradient (red), conjugate gradient (orange), L-BFGS in GROMACS (green), L-BFGS in OpenMM (pink).

12.4 Ensemble equilibration

Starting from a completed energy minimisation of a molecular system, it is usually required to let at least one equilibration run follow before the actual production simulation can be started. Technically, an equilibration is a production run itself and if no special precautions have to be taken here, a certain time interval at the beginning of a production simulation can serve as an equilibration period, which should not be included when equilibrium properties are studied. If, however, a set of parameters is required to properly equilibrate a system that is different from the production settings, these runs need to be executed separately. In general, a system can be assumed to have reached an

equilibrium state when fluctuations of the energy around a mean value have been stabilised and there is no critical drift over time. If a thermostat is used to simulate at a constant temperature, this should hold also for temperature fluctuations, as well as for pressure and density if additionally a barostat is employed.

equilibration strategies

It depends on the system what kind equilibration protocol is most suitable and how long the equilibration period should be. A standard approach, if temperature and pressure should be controlled, is to first let the temperature equilibrate without pressure coupling before the pressure is equilibrated afterwards while maintaining the temperature coupling. On the other hand, if in production only the temperature should be controlled but the system should have a well defined starting pressure, i.e. a reasonable density, it is possible to let the pressure equilibrate without temperature coupling before in a following run the barostat is turned off and the thermostat is turned on. The same goes for constant total energy simulations in which neither temperature nor pressure should be controlled but where it should be ensured that the energy in the system corresponds to a certain temperature. In this case, temperature (and pressure) can be equilibrated before switching off the coupling in production.

Similar to energy minimisations, it is possible to use position restraints on portions of the system to for example equilibrate solvent and solute separately, which may be necessary if a direct equilibration attempt of the whole system leads to instabilities. It is also common, to use somewhat more accurate settings during equilibrations to ensure a stable run in early stages where a system might still be fragile and to relax these settings in favour of performs for production once a stable state has been acquired.

langerin NPT ensemble

Here, I would like to discuss a relatively rigorous equilibration protocol for the langerin CRD to level the field for productions in the *NPT* ensemble. This does not agree to a hundred percent with the employed setup for the simulations used in our langerin study (compare SI of [347]) and is rather meant as a set of slightly refined recommendations. Eventually, we want to use the following standard settings in our final MD runs, which are close to what GROMACS employs as its defaults. Note that these settings depend partly on the used force field, which in this case is AMBER99SB-ildn.[95]

```
integrator = md
nsteps = 100000000
dt = 0.002  ; ps

nstcalcenergy = 5000
nstenergy = 50000
nstlog = 500000
nstxout_compressed = 2500
compressed_x_grps = Protein CA

constraints = h_bonds

coulombtype = PME
fourierspacing = 0.15  ; nm
pme_order = 6

tcoupl = v_rescale
tc_grps = Protein non_Protein
tau_t = 1   1  ; ps
ref_t = 300 300  ; K

pcouple = Parrinello_Rahman
tau_p = 2.0  ; ps
ref_p = 1.0  ; bar
compressibility = 4.5e-5

gen_vel = no
continuation = yes
```

First, we have chosen an integrator to drive the dynamics. The value `integrator = md` corresponds to a leap-frog integration scheme (see also section 4.6). Next, it is specified that we want to run

the simulation for `nsteps = 100000000` steps at a time step of 2 fs (`dt = 0.002`), amounting to a total of 200 ns. We took this length as a rough standard length for a simulation replica (long enough to observe larger timescale dynamics but short enough to be easy manageable). Options for the generated output follow: energies are calculated in intervals of 10 ps (`nstcalcenergy`) and written to the energy file every 100 ps (`nstenergy`). Energy calculations and output generation impact the simulation performance negatively and when there is no general interest in these quantities beyond simple sanity checks, too frequent operations in this regard can be avoided. This also limits the size of the output files. The value for energy output counts for the output of other quantities like temperature and pressure as well. It should be noted that GROMACS provides statistics (average, drift, etc.) for these data calculated at `nstcalcenergy` but only individual values at `nstenergy` can be read out later from the energy file (e.g using `gmx energy`). For the same reason of avoiding unnecessary output, log entries are only generated every 1 ns (`nstlog`). Coordinate output is set to an interval of 5 ps with `nstxout_compressed = 2500`, which should be sufficient for most analyses, and `compressed_x_grps` restricts the output to the protein and the calcium ion. Keeping solvent coordinates as well would increase the trajectory file size enormously and should be done only if explicitly needed (possibly using a lower output frequency).[6] The `constraint = h_bonds` option is an important one because it defines that all heavy atom bonds to hydrogen should be constrained using by default a LINCS algorithm. This allows us to use the chosen time step of 2 fs. It is critical that this setting complies with the recommendation in conjuntion with the used force field. Like already for the minimisations, we use a PME scheme to treat electrostatics (`coulombtype`). Two additional settings can be played with in productions here: `fourierespacing` is related to the number of grid points used in the FFT part of this method and indirectly gives a lower bound for the accuracy, and `pme_order` controls the interpolation.

In our simulations, temperature will be controlled using a velocity-rescale thermostat (option `tcoupl`) (see section 4.8) and pressure using a Parrinello-Rahman barostat (option `pcoupl`). We will start our equilibrations by switching on the thermostat. The velocity-rescale thermostat as implemented in GROMACS is well suited for equilibrations to reach a target temperature and for productions to hold the temperature and produce a correct canonical ensemble.[180] In the first run, the option `continuation = no` is used to tell the program that this is a fresh simulation. In subsequent runs, this is set to `continuation = yes` while the starting configuration including velocities is inherited from the final state of the last run. To generate initial velocities that correspond to a certain temperature, we use `gen_vel = yes` and `gen_temp = 300` (see also section 4.7). In continuation runs this should be strictly set to `gen_vel = no` unless the velocities obtained from a prior equilibration should be overridden. The generation of velocities at the beginning of the first temperature equilibration run is actually optional and the system can be alternatively warmed up solely by coupling to the thermostat. Starting with velocities at a given temperature just shortens the equilibration period or avoids problems with equilibrations to target temperatures over a too broad range. Especially for high temperatures, it can be necessary, though, to warm up a system in stages to avoid that high temperature initial velocities blow up an essentially low temperature minimum starting configuration.

We define furthermore two coupling groups (`tc_grps`) so that temperature is controlled separately for the protein and the solvent (the calcium ion is counted as non-protein). The set groups need to be either standard groups recognised by GROMACS or passed over to `grompp` as and index file (`.ndx`). For each group, we set a target temperature in Kelvin with `ref_t` and a coupling constant `tau_t` in picoseconds that defines how tightly temperature should be controlled. Note that this is not

[6] Note that simulations in OpenMM are more convenient in this regard because separate reporters can be used to write out the whole system including at a lower frequency. Have a look for example at the `mdtraj.reporters.DCDReporter` that allows to specify an atom subset for the output.

the frequency of thermostat coupling during a simulation that can be separately set with `nsttcoupl` and is by default equal to the frequency of neighbourlist updates. Overall, example settings for the temperature equilibration are (leaving out unchanged production settings from above):

```
nsteps = 200000
dt = 0.001  ; ps

nstcalcenergy = 10
nstenergy = 100
nstlog = 10000
nstxout_compressed = 0

gen_vel = yes
gen_temp = 300
continuation = no
```

Here, we reduced the time step to 1 ps for a higher accuracy in equilibration and let the total simulation time amount to 200 ps. The frequency of energy output is increased to 0.1 ps to get better statistics when we analyse these. In contrast, coordinate output is not necessary at all. Preparing and running a simulation with the given settings passed as `nvt.mdp` text file could look this:

```
gmx grompp -f nvt.mdp -c ../ions.gro -p ../topol.top -o nvt.tpr
gmx mdrun -nt 6 -deffnm nvt -pin on
```

A quick equilibration run like this does not require a lot of resources. On 6 CPUs one can expect a performance of about 10 ns d^{-1} for the langerin CRD, which makes the run finish in under 15 min. Figure 12.5 shows temperature, pressure, and energy evolutions for equilibration runs with different `tau_t` values. Note right away the small drift in the energy that is, however, normal and not to worry about.

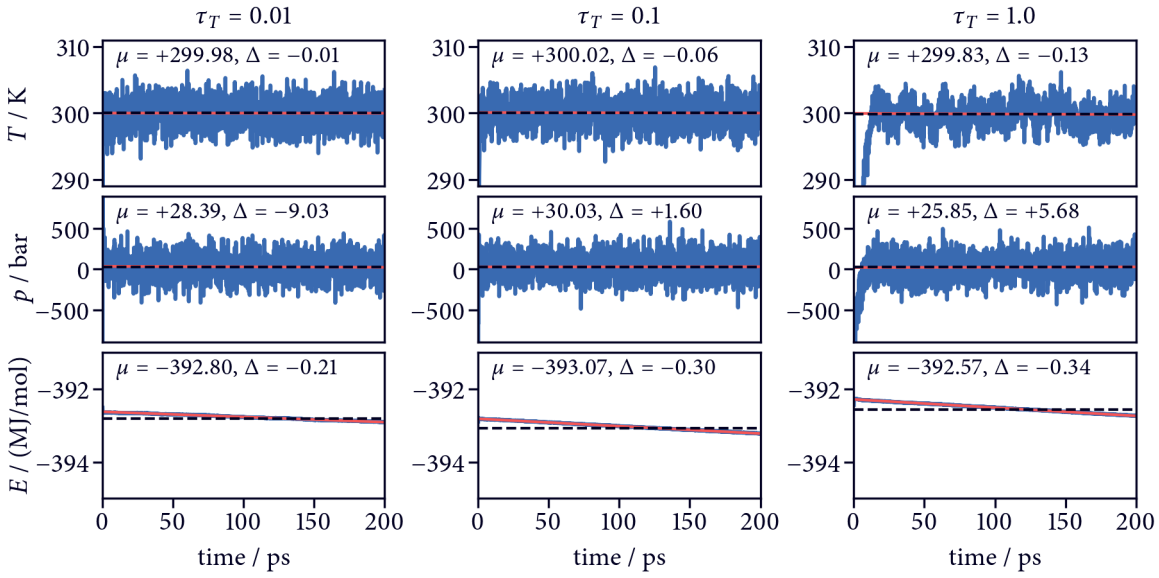


Figure 12.5 **Langerin *NVT* equilibration**
Temperature, pressure, and conserved energy evolution in 200 ps simulations using a velocity-rescale thermostat (GROMACS 2020) with coupling values of τ_T = 0.01, 0.1 (default), or 1 ps. The mean μ over the last 75 % percent of each trajectory is marked with a dashed black line. A least-squares fit of the data measuring the drift Δ is added with a red line.

The output of these equilibrations looks satisfactory as the temperature fluctuates stably around an average that matches the set target value and has essentially no drift for all tested coupling values. The

equilibration is fairly quick so that we also could have chosen shorter runtimes, say 50 to 100 ps. It can be seen how a weaker coupling (τ_T = 1.0) leads to slightly slower equillibration but still yields a good result. In production we can therefor safely use this value. A fluctuation of the instantaneous temperature about a few degrees is expected due to the small size of the system. As figure 12.4 illustrates, the fluctuation is less pronounced for the larger *non-protein* group.

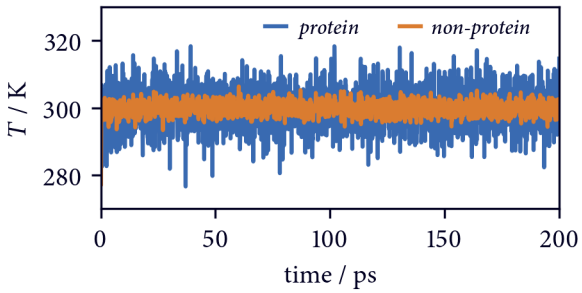


Figure 12.6 Thermostat coupling groups The temperatures for the two groups *protein* and *non-protein* are equally well equilibrated. Fluctuations are notably reduced for the larger solvent group. The plot corresponds to an equilibration with τ_T = 0.01 (compare figure12.5).

Pressure is not controlled yet but stable as well. Since we want to simulate at a reference pressure closer to ambient conditions, though, we will turn on a barostat next. In recent GROMACS version, a stochastic cell rescaling (C-rescale) barostat is available that can be used for equilibration and production equally well.[381] Prior to this, one usually needed to employ two separate *NPT* equillibrations: one to quickly reach the reference pressure, and one to sample from the correct ensemble. Figure 12.7 and 12.8 show equillibration runs to this effect using first a Berendsen and than a Parrinello-Rahman barostat with different coupling constants. The runs were continued from the *NVT* run with the smallest coupling value, which was now set to τ_T = 0.1 in all runs. The second pressure equilibratation was continued from the first with τ_T = 1. As can be seen from the figures, the first run equillibrates very quickly, pressure and density are well maintained, and the coupling constant has no big impact. Notably, though, the density fluctuates less with weaker coupling. For the Parrinello-Rahman barostat, the reference pressure is slightly less well maintained with larger coupling constants and the density starts to oscillate over longer time intervals. We should therefor leave the coupling at a value close to τ_p = 1 in production.

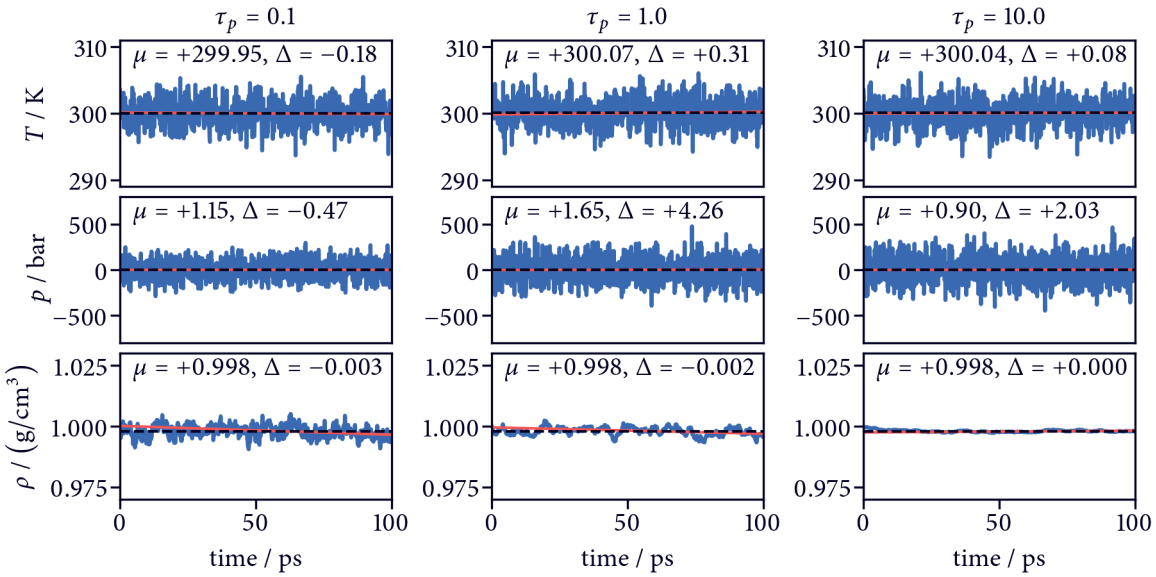


Figure 12.7 **Langerin *NPT* equilibration (1)**

Temperature, pressure, and density evolution in 100 ps simulations using a Berendsen barostat (GROMACS 2020) with coupling values of τ_p = 0.1, 1 (default), or 10 ps. The mean μ over the last 75 % percent of each trajectory is marked with a dashed black line. A least-squares fit of the data measuring the drift Δ is added with a red line.

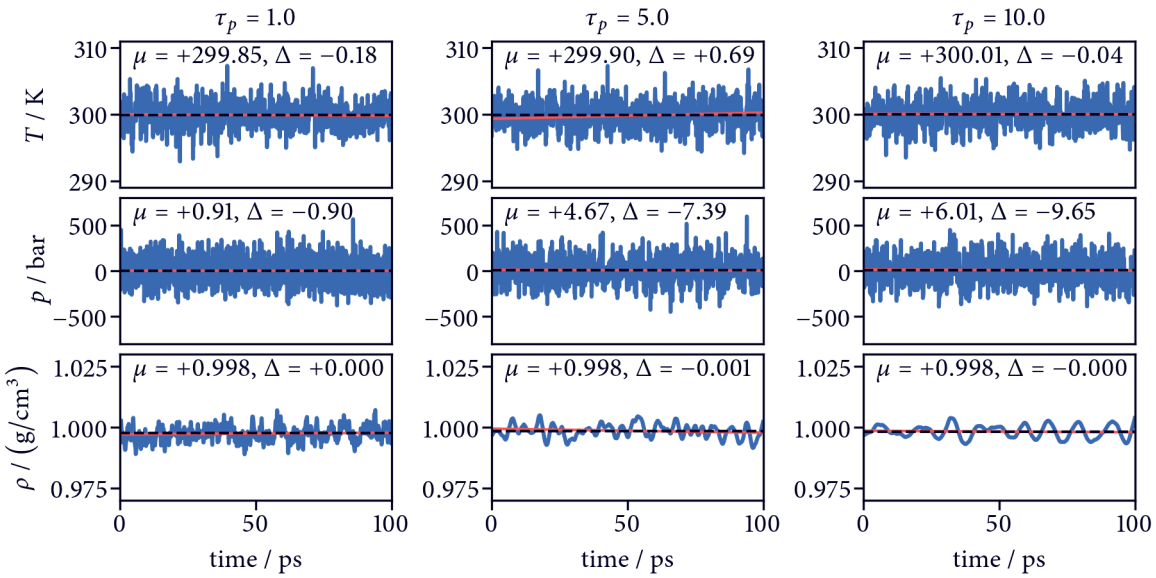


Figure 12.8 **Langerin *NPT* equilibration (2)**

Temperature, pressure, and density evolution in 100 ps simulations using a Parrinello-Rahman barostat (GROMACS 2020) with coupling values of τ_p = 1 (default), 5, or 10 ps. The mean μ over the last 75 % percent of each trajectory is marked with a dashed black line. A least-squares fit of the data measuring the drift Δ is added with a red line.

Part V.

Clustering algorithms

{ 13 }

Clustering—the basics

Terminology and definitions

Clustering[1] is an analytic process that identifies associations of some kind (i.e. *clusters*) among a number of considered objects. Phrasing this differently, 'clustering is a synonym for the decomposition of a set of entities into *natural groups*'.[382] The word 'natural' appearing in this quote indicates already that this definition leaves room for interpretation. Clustering as a task has as such no precise, unambiguous goal, and what a cluster actually is can not be universally defined. Sometimes, clustering is even reduced to just achieving 'a grouping of objects' as a common denominator without further specifications.[383] A sorting of arbitrary objects into groups can be done in many different ways, and how it is done best, does usually depend on the question of *why* it is done. The same set of objects can be clustered based on all of its properties or only a selection thereof, and even if the same properties are used, different clustering approaches will in general yield different results. I will proceed on the common ground, that clustering tries to establish some sort of relation between the clustered objects, meaning that objects within the same identified group should be alike with respect to (some of) their properties, and in reverse, objects in different groups should be less alike. This still rather open notion can be expressed as the paradigm of *strong within-cluster relationship* versus *weak between-cluster relationship*, that should be satisfied by a structure found among objects through a clustering.[382] How different clustering techniques understand the broader notion of relationship, will be discussed in section 13.2.

general definition

object relationship

Obviously, relationships among objects can be subjective and finding relationships has a lot to do with human intuition. Humans are in a way intuitively very good at the task of clustering objects visually by distinguishing different kinds of objects according to spatial proximity, colour, or other—sometimes oblivious—criteria. Just by looking at a set of objects, we can often immediately see or feel how individual objects could be appropriately split into groups. The problem is of course, that not every set of objects can be conveniently visualised and even worse, our intuition would depend on the exact way how the visualisation is achieved. Additionally, human analysis is usually limited to a rather low number of objects or a low number of simultaneously considered properties, not to mention that while we may be able to identify groups of objects in an instance, documentation and isolation of the result to make further use of it is more often than not extremely tedious.

human analysis

Figure 13 should illustrate the ambiguity that usually accompanies clustering tasks with a casual example. It shows a photograph of plastic ducks which can be analysed by clustering in many different ways. Most important would be the question, what the clustering should be done for. Do we want to split the image into areas of different colour so that each identified cluster contains pixels of related colours? Do we want to identify groups of pixels that form individual ducks so that each cluster contains only the set of pixels for one respective duck? Or do we want to identify groups of ducks, that is for example ducks with a certain orientation or local rafts of ducks? The overall goal may be

a casual example

[1] The term *clustering* will be used here either as a verb to describe the act of performing the respective analysis (to cluster objects), or as a noun to describe the outcome of the analysis (a clustering of objects).

accomplished by either considering individual pixels or entire ducks or parts of the ducks as the set of objects to cluster and the examined object properties will be different in each case. Along the line, there are many open detail questions: how fine should the clustering be? Do we only want to separate red from yellow or do we want to distinguish different shades of yellow? Is there a maximum number of clusters we want to deal with? Should each cluster be restricted to contain a minimum or maximum number of object members? Some of these clustering tasks can be in principle well done by human visual analysis but in practice this is seldom really feasible or efficient.

Figure 13.1 Clustering an image of plastic ducks The picture on the left shows a crowd of plastic ducks. The formulation of a clustering task involves several questions: what do we want to achieve—a separation into areas of different colour? A grouping of pixels belonging to each duck? A grouping of related ducks? What do we consider as objects—individual pixels? Individual ducks? By which property do we want to cluster—colour of pixels? Orientation of ducks? For many of these tasks, we have an intuitive idea about what the result looks like. Photo by Marcus Lenk on Unsplash.

computer analysis

Computer aided clustering schemes try to produce fast and reliable groupings of possibly very large numbers of objects based on objective, quantitative criteria. As such, computer analysis is supposed to complement intuition driven human analysis. It turns out, however, that it is not an easy endeavour to find automatic clustering schemes that are universally adequate. A vast number of clustering techniques has been proposed, based on various underlying principles, expressed in many different method frameworks, and realised through an even larger number of concrete implementations of the respective algorithms. The reason for this might be, that clustering commonly serves a specific practical purpose and the question of what a clustering scheme should do and the notion of what a cluster should be, depends on the formulation of the clustering problem—which in turn varies strongly among disciplines. Different clustering approaches have been developed before different application backgrounds, making different assumptions about the clustered objects and making different demands on the clustering results.[384] The matter is complicated by the fact that clustering approaches are hard to compare with each other and that in general it cannot be argued about the *right* or *wrong* of a specific clustering result. Clustering schemes produce clusterings in accordance with their underlying design. In this sense, any clustering result is basically *correct*—disregarding of course that clustering implementations may contain bugs that make them incorrect. This does not mean, however, that any clustering result is also *relevant* or *meaningful* in any situation. The question of what is a good clustering is biased by the view and beliefs of whoever uses it.[384] Consider the six different groupings of ball-like objects in figure 13.2. Based on a personal opinion, you may find one of the groupings most persuasive but without further context all of them are equally appropriate.

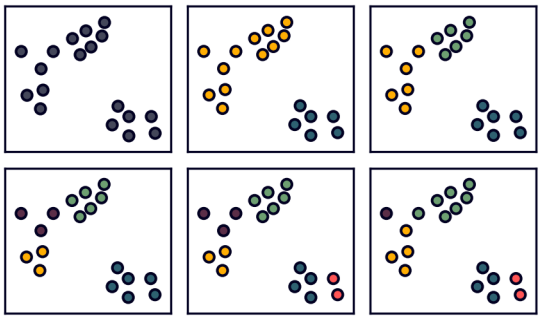


Figure 13.2 Clustering is imprecise The formulation of what a clustering should achieve and the definition of what a cluster should be is ambiguous and depends on the perspective of the user. All the clusterings on the left can be legitimate (including the trivial case of just one group).

Eventually, the usefulness of a clustering result with respect to the initial intention decides whether it is indeed valid. The validity of a clustering result can be effectively judged only either by an expert through careful analysis of the obtained clusters (called *manual* evaluation) or by a further use of the clustering in yet another application to deliver a result that can be better evaluated (*indirect* evaluation). Manual evaluation puts the focus on human intuition (or let's call it expertise for that matter) back on the table. Expert users need to have a certain expectation about the result of a clustering that is predicated on an intuitive or experience-based idea of the identified clusters, which may also be called some sort of domain knowledge. A clustering is good, if it meets the expectation. Clustering schemes can be chosen or results can be refined in order to match the expectation as closely as possible. Things become difficult, though, if manual inspection, which mostly means visual inspection, is not feasible for example when a clustering contains many groups or if groups are split based on many object properties. A fundamental issue about expectation is also that it could be unjustified. If a clustering does not meet the expectation, it could be either the clustering or the expectation that needs adjustment. It can also be a problem, if there is no expectation. In many practical situations, it is not exactly known what kind of clustering is most suitable for a given set of objects. Generally speaking, clustering is often employed in the hope of *finding* some sort of structure in the set of clustered objects. What is actually done by a clustering process is that a certain structure in accordance with the design of the chosen clustering method is *imposed* upon the objects.[385] Clustering always produces a structure as a kind of hypothesis about the clustered objects[384] but there is no guarantee that this structure is actually well-founded or—to make a connection to the introductory definition of clustering—the *natural* structure among the objects.

manual evaluation

Indirect evaluation can partly counter the vagueness that we are confronted with when the quality of clusterings should be assessed. Instead of comparing clusters obtained by a specific procedure to a subjective expectation that may still be flawed, it can be helpful to get objective feedback from an application which the clusters are used for. The clustering result itself is of secondary importance if it proves functional for something that should be done with the clustering. The result of a subsequent step validates or invalidates the clustering step. Unfortunately, such indirect evaluation is not always practical, for example because the groups obtained through a clustering are themselves the result of interest or because the application they are used for does not give a definite positive or negative response. It can also be problematic, if the validating application gives falsely positive feedback, that is if things seem to work well but they actually do not. Similarly, things may work well in an application for the wrong reason which makes it difficult to infer anything about the quality of a clustering in general. An example for indirect evaluation with a direct connection to the content of this thesis would be the construction of kinetic Markov-models from molecular simulations. In this context, clustering is used to define conformational (micro-)states on top of which the model is estimated. In principle, a good clustering is one that allows the construction of a good Markov-model. In reality, the question whether one has got a good model is not that straightforward, though, in particular not if the aim is to rank clustering results quantitatively. Moreover, also a suboptimal clustering, for example one where certain conformational states (i.e. relevant information) are missing, may give a seemingly good model.

indirect evaluation

There are two other forms of clustering validation that are worth mentioning: *internal* and *external* evaluation. Internal evaluation tries to assess the quality of a clustering result based on the corresponding grouping of objects itself. Making assumptions about how identified clusters should be constituted in the ideal case, clustering results can be quantified with respect to these assumptions. In fact there exists a whole bunch of statistical measures that can be used for checking to what extent a clustering satisfies some idealised theorization. A few of them will be discussed in section 14.3 in the context of *k*-means—a particular clustering approach for which internal validation criteria can be well applied. There is, however, the potential risk of running into a circular argumentation.

internal evaluation

A conjecture about what ideal clusters are, is likely to be one that could be used as the underlying principle of a clustering method in the first place. Clustering schemes that find clusters according to some criterion for how the resulting clusters should be organised will of course be considered favourable by a validation that uses the same criterion. Clustering schemes on the other hand that are based on different assumptions, will be probably considered less favourable. An agreement with a certain validation criterion is therefore only meaningful if different concrete clustering algorithms, based on the very same principle, are compared to find out which one produces clusterings in closest agreement with the initial clustering objective. In other words, internal validation can proof if a specific algorithm or implementation is successful in realising an underlying principle. Conversely, an agreement or disagreement of a clustering result with a validation criterion tells us basically nothing if it is not the intend of the clustering method to satisfy the validation criterion in the first place. In any case, internal validation measures may just be nonsensical because the assumption they are based on are not aligned with the actual nature of a specific object set, and the validity of the validation technique itself often needs to be scrutinised.

external validation

External validation uses a reference clustering to compare that to the result of a specific clustering method. It assumes that for a given set of objects the *true* groups are known so that the groups obtained in a clustering can be judged by the number of individual objects correctly assigned to a certain group. Because it is in general impossible to know the true group structure of arbitrary object sets, external validation uses exemplary benchmark sets for which it is presumed that the inherent groups are obvious or at least apparent to the expert that created the example. These sets are often synthetic and relatively simple. Benchmarking different methods of clustering against an array of representative test object sets, can give valuable insight to the fitness of a method for a given purpose. It can tell us, if a clustering solves a designed problem in the desired fashion and can reveal major differences between clustering techniques. It can also be used to check if a specific clustering algorithm produces results in line with its own principles. On the other hand, a good clustering performance in benchmarks can not be transferred one-to-one to the assessment of realistic objects sets. It can only give a hint if a clustering method is in principle suitable for the analysis of a specific object set under the constraint that the set in question resembles the benchmark set but it can not validate the actual clustering result. And yet again, benchmark sets are based on certain assumptions about identified clusters—assumptions that may be flawed because they are based on intuition or because they were made to comply with a specific purpose or idea about clustering.

clustering applications

Clustering is a complex and sometimes confusing topic that entails a fair amount of tension between objectivity and intuition—between theoretical conciseness and practical relevance. It can be understood from multiple standpoints and its proper usage depends on perspective, which makes it not surprising that the terminology in the context of clustering is often not very consistent. Despite all this, a combination of computational clustering techniques based on objective criteria with human inspection and decision making can be very powerful.[386] In practice, clustering is often a tool for exploration and a way to get some relational insight about the considered objects from different angles. It cannot be considered a tool that provides definitive answers about the actual group structure of the objects in all cases. Acquiring knowledge through clustering, is often an iterative and incremental process. Clustering is applied to various problems in a wide array of scientific and industrial fields.[387] A very early documented example goes back to 1855 when during a massive cholera outbreak in London, cases of death were tracked on a city map and revealed major sources of infection where they accumulated in clusters.[388] Of course neither the term cluster in the present formal sense nor the respective analysis has been established at this time, although an intuitive comprehension of object groups may as well existed already much earlier. For instance, [383] names Aristotle's classification of living things as one of the first-known clusterings. Clustering as an acknowledged computational tool has its roots in the 1960s beginning with biological taxonomy.[385, 389] Naturally, the focus of this

thesis lies on the clustering of molecular objects, that is conformational snapshots obtained from MD simulations, which experienced an upswing in the 1990s.[390–392]

forms of object labelling

To cluster objects usually means to put group *labels* on individual objects that indicate a *membership*. Clustering procedures can be technically discriminated based on how this assignment is made: in the simplest case, each object will be given exactly one label, e.g. an integral number (object *a* belongs to group 1, object *b* to group 2, and so forth). If in the end all of the initially considered objects are labelled, the clustering is called *complete*, or *exhaustive*, or is described as *full* clustering. The property of assigning objects to one and only one group makes a clustering *exclusive*, *hard*, *strict* or *crisp*. If on the contrary, some of the objects may be left out of the assignment—that is they are treated as noise or outliers—the clustering can be called *partial* or explicitly a clustering *with outliers*. Furthermore, the counterpart of hard clustering would be either *overlapping* if objects are allowed to be a member of more than one cluster, or *soft* (also *fuzzy*) if objects are assigned to clusters with a certain degree of membership. *Probabilistic* label assignments are more or less the same as fuzzy just with the additional flavour that group memberships are seen as probabilities while the true memberships of objects to groups are assumed to be crisp.[383]

clustering vs. classification

To assign group labels to objects is also the aim of *classification* which describes the association of objects into one of multiple *classes* or *categories*. Traditionally, clustering can be therefore seen as a form of classification and both are in fact widely used as synonyms in literature especially from a statistical standpoint.[393] Arguably, they are not referring to the same thing, though. On the one hand, classification through clustering is possible as the clusters identified by clustering can be interpreted and used as categorical classes. On the other hand, clustering is not necessarily always classification because the resulting clusters can be interpreted and used differently, e.g. as a form of discretisation or condensation. Moreover, classifications can be done not only through clustering but through a variety of other techniques. In machine learning terms, clustering and classification are predominately distinguished more strictly.[383, 394] Clustering can be understood as an *unsupervised* learning process through wich objects are labelled without any kind of labels already being present that could guide the procedure. Clustering basically invents the labels in the first place.[384] Classification of objects on the contrary is equivalent to object labelling with present labels for those or other objects that are used to control the process, which is understood as *supervised* learning. Classification uses labels that have been established through something else. If clustering is used for classification, it may be referred to as unsupervised classification to discern it from forms of supervised classification. It should be noted, however, that this differentiation is softened by the fact that there is indeed such a thing as semi-supervised and supervised clustering.[395, 396] Coming back to the somewhat silly duck example at the beginning in figure 13, one could ask now the question if humans are actually very good at intuitive clustering or rather at classification. Can we identify individual ducks because we can cluster them ad-hoc or because we already know from experience what a duck looks like? The difference between the process of grouping objects into groups that are de facto unknown and created while the grouping happens—independent of previous groupings—and that of grouping objects into groups that are already fixed may be subtle but it contributes to the overall bewilderment that may at times accompany clustering as an analysis tool. In any case, it can make a rather big difference for the choice of clustering methods whether they should be used to classify data or not.

Up to here, we rather abstractly talked about 'objects' as the entities that are being subjected to a clustering. The next section will try to clarify what this actually means concretely for computational clustering—using either the concept of objects as points embedded in a metric space or as nodes in a graph network.

13.1 Data sets and representations

The topic of clustering that is concerned with finding groupings of objects based on relationships between these objects, leads us to the concept of *data*. To cluster objects means to cluster data of some specific shape or form. The rather open term *data set*, i.e. a collection of *data*, is unfortunately in turn not defined very sharply and its use varies among disciplines. According to the UNECE cited in the OECD glossary of statistical terms,[397] 'data is the physical representation of information in a manner suitable for communication'[398]. Depending on the actual context, the kind of stored information can be very different. As far as we should be concerned throughout this chapter, a data set is a collection of identically structured *records* where each record is basically a listing of values for arbitrary variables. Following a particularly practical definition in the IBM z/OS documentation, a record in a data set 'is the basic unit of information used by a program'[399]. As such, records can be usually understood as tabular data, which makes a data set essentially a data table (a file) in which each column stands for a variable and each row represents a single coherent group of values for these variables—neglecting the complexity that is posed by a multitude of file-formats that structure this information in different ways. Alternative names for records are *examples* or *samples*. An individual variable within a record can also be referred to as a *field* or *feature*, and its values (a single *datum*) can be of numerical, ordinal, or nominal nature—or actually of any other imaginable abstract type.

general definiton

A classic example is Ronald Fisher's *Iris plant* data set which collects 150 records of flowers for which the sepal and petal length and width (the dimensions of two different leaf types) have been measured (see table 13.1).[400] For each measured flower, additionally the botanical class (one of *setosa*, *versicolour*, or *virginica*) is known.

Iris data

ID	s_l	s_w	p_l	p_w	class
1	5.1	3.5	1.4	0.2	*setosa*
2	4.9	3.0	1.4	0.2	*setosa*
3	4.7	3.2	1.3	0.2	*setosa*
4	4.6	3.1	1.5	0.2	*setosa*
…					

Table 13.1 Extraction of Fisher's *Iris plant* data set Each entry is marked with a unique record ID and identified with a category (class). The quantities s_l (sepal length), s_w (sepal width), p_l (petal length), and p_w (petal width) are measured in centimeters.

In the following we will have a closer look at how to represent, visualise, and analyse a tabular data set like this before the background of clustering. Fisher's *Iris plant* data set has four real-valued numeric features with identical physical dimension (a length in centimetres). These features as such span a feature space in which each record can be represented as a single point, i.e. a feature vector in four dimensions. The data set of $n = 150$ entries can be generally denoted as

$$\mathcal{D} = \{x_1, ..., x_n\}\,, \tag{13.1}$$

feature space

where each data point is a sample from the $m = 4$ dimensional feature space $x_i \in X$ with $x_i = (x_{i,1}, ..., x_{i,m})$. In this special case, the feature space coincides with $\mathbb{R}^4_{>0}$. For clustering analyses, it is generally beneficial if the feature space is a real space $X \subset \mathbb{R}^m$ because then basic mathematical operations with respect to the samples in a data set (in particular distances between points) are well-behaved, and in fact there are clustering methods that do explicitly depend on this. Figure 13.3 shows a visualisation of the *Iris* data as two separate 2-dimensional projections for the sepal and the petal features. Each data sample appears in the plots as a single point at coordinates corresponding to the values of the respective orthogonal features, and coloured according to the known biological flower class of the sample.

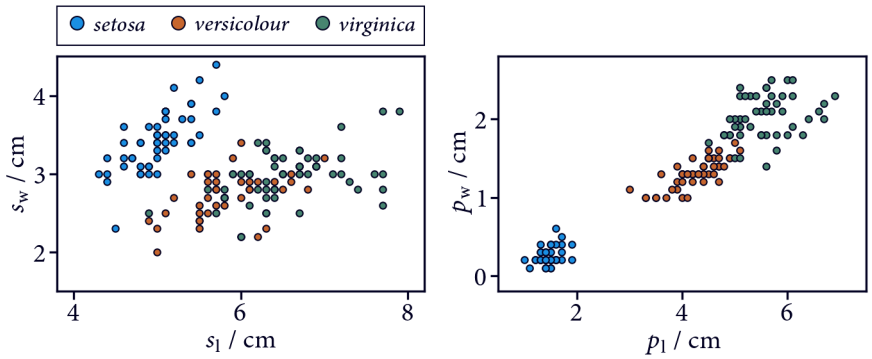


Figure 13.3 *Iris plant* data set with biological classes Data points plotted in the 4-dimensional feature space of s_l (sepal length), s_w (sepal width), p_l (petal length), and p_w (petal width) with their biological classification indicated by colour.

For the representation of this tabular data set in a computer program, the data can take the general form of a $n \times m$ matrix D, so that there is one row for each data point and one column for each feature and the matrix element D_{ij} holds the jth attribute of the ith data point. Different programming languages provide their own concrete data structures for this. In Python, the native choice of data structure would be a list of lists which would look for the *Iris* data like:

point coordinates

```
D = [
  [5.1, 3.5, 1.4, 0.2,],
  [4.9, 3.0, 1.4, 0.2,],
  [4.7, 3.2, 1.3, 0.2,],
  [4.6, 3.1, 1.5, 0.2,],
  ...
]
coordinates_i = D[i]
```

This can be replaced by a `ndarray` with the third party library NumPy, or with a `DataFrame` in Pandas, but there is a wide bouquet of other structures to represent this kind of information. Usually, the important characteristic property of these 'matrix'-like data structures is that they are *indexable* (allows access to the element ij) or *iterable* (allows to loop over all points). Implementations of clustering methods may make strict requirements on how the data is stored and how individual points (or better their coordinates) are accessed.

While for clustered objects identified with points in a (metric) data space different cluster methods can make direct use of point coordinates, it is often not the location of the points itself that is of interest but only the spatial distance between them. So instead of representing a data set as a $n \times m$ matrix of coordinates, it can be possible (or required) to have a $n \times n$ *distance matrix* of pairwise distances where each element D_{ij} holds the distance between point i and point j. In principle, the same concrete data structures as for point coordinates can also be used for distances, so using the Euclidean distance for the *Iris* example one would have something like:

pairwise distances

```
D = [
  [0.00, 0.54, 0.51, 0.65, ...],
  [0.54, 0.00, 0.30, 0.33, ...],
  [0.51, 0.30, 0.00, 0.24, ...],
  [0.65, 0.33, 0.24, 0.00, ...],
  ...
]
distance_ij = D[i][j]
```

Similar to a data structure for point coordinates, central properties of distance data structures are that individual points (or rather their distances to other points) can be accessed by some form of indexing and that one can iterate over all objects in the data set. It should be noted, though, that the memory complexity for distance matrices of $\mathcal{O}(n^2)$ can prohibit the explicit storage of all matrix

elements for larger data sets, so that sparse data structures should be leveraged, which for example avoid the storage of 0-valued elements and the duplicate storage of symmetric elements. In this way, the number of physically stored distances can be reduced to at most $(n^2 - n)/2$. Distances can also be given implicitly as a distance function while still only point coordinates are physically stored. A distance metric $d : X \times X \to \mathbb{R}$ is a function that maps two points x and y of the data space to a value so that the following is fulfilled:

$$d(x, y) \geq 0 \qquad \text{non-negativity} \tag{13.2}$$

$$d(x, y) = d(y, x) \qquad \text{symmetry} \tag{13.3}$$

$$d(x, y) = 0 \Leftrightarrow x = y \qquad \text{indiscernibility} \tag{13.4}$$

$$d(x, y) \leq d(x, z) + d(z, y) \qquad \text{triangle inequality} \tag{13.5}$$

For sampled data in a data set $\mathcal{D}$, the identity of indiscernibles 13.4 can usually not hold because a data set can contain duplicates. If two samples have identical coordinates, a distance between them should be zero, but from a zero distance does not follow that two points are the same sample so that equation 13.4 becomes $d(x, y) = 0 \Rightarrow x = y$. It is a practical question, if duplicate samples should be removed from a data set prior to a clustering. In the context of clustering, the concept of pairwise object distance is construed within the broader concept of object similarity that will be addressed further in section 13.2. If distance evaluations are decoupled from the general internals of a clustering method, it can be said that a clustering operates on a latent space that is the actual data space can be unknown or is regarded irrelevant for the grouping into clusters.

point neighbourhoods

Taking the representation of a data set through pairwise distances one step further, it is also possible to look at the data in terms of neighbourhoods. A clustering may depend on the information if two points are considered neighbours of each other rather than on how far two points are away from each other exactly. There are two basic, practically used forms of defining neighbourhoods derived from a distance function: *fixed radius near* and *k-nearest* neighbourhoods. Using a fixed distance (radius) r, the sampled neighbours of a point x in the data set can be denoted as the set $\mathcal{B}_r$

$$\mathcal{B}_r(x) = \{y \in \mathcal{D} \mid d(x, y) < r\} \quad \text{open-ball} \tag{13.6}$$

or

$$\mathcal{B}_r(x) = \{y \in \mathcal{D} \mid d(x, y) \leq r\} \quad \text{closed-ball} \tag{13.7}$$

that is the collection of samples that lie within or below r measured from the location of x. While it may make a fundamental mathematical difference if $\mathcal{B}_r$ is defined to be the open-ball or closed-ball around x, it is usually considered a technicality when it comes to clustering. The same goes for the question if $y = x$ should be contained in the neighbourhood or not, which means if formally a point x is its own (closest) neighbour. Note that the neighbourhood of a point is in this sense directly the near neighbourhood with respect to r and not any set of points containing $\mathcal{B}_r$. Instead of a fixed distance, the k nearest samples to a point can be used to define neighbourhoods as the set $\mathcal{B}_k$

$$\mathcal{B}_k(x) = \{y \in \mathcal{D} \mid d(x, y) \leq d_k(x)\} \tag{13.8}$$

where the k-nearest distance d_k from a point x is the distance for which there are *at least* k points at $d(x, y) \leq d_k$ and *at most* $k - 1$ points at $d(x, y) < d_k$. Again it may be optionally required that $y \neq x$. Other definitions of neighbourhoods—derived from distances or not—are conceivable. For the choice of data structures to represent neighbourhood information there are multiple options among which there are two fundamentally different approaches. The first would be to use a $n \times n$ matrix comparable to a distance matrix but with binary entries indicating pairwise neighbour relationships in a true/false manner, which could result in the following for the *Iris* example when a fixed radius $r = 0.52$ is used on the previously shown distances:

```
D = [
  [1, 1, 0, 1, ...],
  [1, 1, 0, 0, ...],
  [0, 0, 1, 0, ...],
  [1, 0, 0, 1, ...],
  ...
]
is_neighbour_ij = D[i][j]
```

This type of matrix can be called an *adjacency matrix*. Since it contains a lot of 0-valued elements in some cases, this type of information is prone to be stored in sparse data formats. Alternatively, it is possible to represent the same information in the form of a sequence of neighbourhoods, for example by keeping a list of point indices to refer to the neighbours of each point:

```
D = [
  [0, 1, 3, ...],
  [0, 1, ...],
  [2, ...],
  [0, 3, ...],
  ...
]
neighbours_i = D[i]
```

The term *neighbour list* is sometimes used in this context but it is not exactly optimal because it can be ambiguous whether it refers to the neighbours of an individual point (i.e. a single neighbourhood) or the full list of neighbourhoods. It is usually again an important property of these data structures that points (that is their neighbours) can be indexed and that one can iterate over all points. As it is the case for distances, it may be possible that neighbourhood determination can be decoupled from the actual clustering.

A universal pattern behind the representation of a data set either in terms of object attributes (point coordinates), pairwise distances, or neighbourhood relations, is to understand a data set as a graph in which each object is a vertex and distances or neighbour relations are indicated by possibly weighted edges (see also chapter 7). Thinking about data sets as graphs is really more a mindset and it is not tied to a specific data structure. In fact, all the presented matrix-like data structures can be viewed as manifestations of graphs making the node-edge nature of data objects more or less explicit. The graph picture is more general than a certain conceptualisation of a specific type of information because it unifies them all. A node in a graph, i.e. an object in the data set, may have attributes that may in turn correspond to coordinates in a metric space—but this is not actually required. Edges between nodes can encode distances or neighbourhood relations, which may be derived from object coordinates or may be the primary source of information, as a general form of connectivity but this is again optional (the graph does not need to be complete). Note that the terms distance, adjacency, and as we will see later (dis)similarity or also *affinity*, and the associated matrix types can become a little bit blurry in general here. All have in common that they correspond to edge weights of an (implicit) graph and while it can make sense to use one of them over the other depending on the situation, they are also often used interchangeably. It depends on the clustering method if it is practically advantageous to think about the data as a graph and which kind of information is captured in it.

graph data

On the other hand, there are certain data structures that are especially valuable for the representation of graph data because they alleviate certain operations beyond the access of individual points and iterations over the data set. In particular, these operations are membership lookups, the addition and removal of data objects, the splitting and merging of data (sub)sets, and the modification of inter-object relations. A simple, explicitly graph-like representation of the *Iris* data using a Python

dictionary for the set of nodes and a Python set for the set of edges (indicating binary neighbourhood connections) could look like this:

```
D_nodes = {
  0: [5.1, 3.5, 1.4, 0.2,],
  1: [4.9, 3.0, 1.4, 0.2,],
  2: [4.7, 3.2, 1.3, 0.2,],
  3: [4.6, 3.1, 1.5, 0.2,],
  ...
}
D_edges = {
  (0, 1), (0, 3), ...
}
```

Many specialised implementations of graph data structures are available in different programming languages, the `networkx` module being only one example for a third party Python package for exactly this purpose. A plot of the *Iris* data in terms of a graph is shown in figure 13.4.

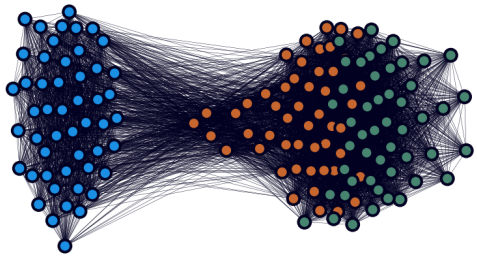


Figure 13.4 ***Iris plant* data set as a graph** Data points plotted as nodes in a graph coloured by their biological classification. The edges are scaled in length by weights corresponding to the Euclidean distance between the data points. Distances $d(x, y) > 3$ have been omitted.

We stated that the *Iris* data feature space is a real space, and so it can be the case for other data sets, in particular for MD data where the considered data space is often a configurational space with (projected) cartesian coordinates. Generally, however, the feature space associated with a data set does not have to be restricted to real valued components as the records in a data set can be composed of basically arbitrary variables. The graph perspective on the clustered objects becomes even more appropriate then because graph nodes can be equipped with any attribute and non-standard attribute combinations do not need to be forced into a strict matrix-like layout. Care needs to be taken, however, in these cases about how basic mathematical operations can be defined. Without going into great detail of working with non-numeric data and the problems that it can involve, here are a few general points to keep in mind. It is often possible and good advice to convert the features of a data set in a way that makes them numeric.

non-numeric features

For example, binary categorical values (e.g. yes/no, true/false, present/absent) can usually be represented by 0 and 1 respectively. It just may need to be considered that to compute a distance between points in a binary space, the Euclidean distance may be not very appropriate and other distance measures should be preferred. Take for example, the following data set of three points with four binary attributes

binary features

```
D =[
  [1, 0, 1, 1],
  [0, 0, 0, 1],
  [0, 0, 1, 1],
  ]
```

The Euclidean distances of $d_{0,1} = \sqrt{2}$ and $d_{0,2} = 1$ between the points are not very intuitive while the Manhatten (or in this case equivalently Hamming) distances $d_{0,1} = 2$ and $d_{0,2} = 1$ can be well interpreted as the number of features in which two points differ.

Things become more difficult with nominal categorical values (e.g. countries DE/FR/GB) where the only valid operations are absolute comparisons (e.g. DE = DE, FR ≠ GB). A common approach to transform these, is to introduce a number of dummy features, one for each categorical value that a nominal feature can adopt. This is called a one-hot encoding. A single nominal feature comprising three categories would be replaced by three binary features like shown here with an example of four data points:

nominal features

```
D_nominal = [
  ["a"],
  ["b"],
  ["c"],
  ["a"],
  ]
D_one_hot = [
  [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1],
  [1, 0, 0],
]
```

It is now possible to use the Manhatten distance again so that points differing in one nominal feature have a distance of $d = 2$ from each other. Note, however, that the Hamming distance works well with categorial data even without the encoding. There are other possible comparisons, e.g. via the Sørensen-Dice or Jaccard index, just to name a few.

It is arguably most difficult to adequately represent ordinal, categorical values (good/neutral/bad, low/rather low/medium/rather high/high). These can be either treated in the same way as nominal values which will, however, ignore the fact that there is an ordering in the categories and distances between each pair of categories are not all equal (e.g. $d(\text{good}, \text{bad}) \neq d(\text{good}, \text{neutral})$). Alternatively, they can be mapped to numerical values (e.g. good: 3, neutral: 2, bad: 1) to reflect the ordering (that is good > neutral > bad) but it may be difficult to ensure that the numeric distances (e.g. here $d(\text{good}, \text{neutral}) = d(\text{bad}, \text{neutral}) = 1$) are aligned with the true, intrinsic distance of the categories.

ordinal features

For special data objects or attribute values, it may be furthermore necessary to select specialised treatments of comparisons. There is for example the Levenshtein distance to compare strings. Especially problematic may be mixed data spaces with partly numeric, partly categorical features (more on this further below).

A TYPICAL PROBLEM WITH FEATURE SPACES can occur when individual features have very different dynamic ranges, or different physical dimensions or meaning. In particular, this can be the case for mixed data. Let's first have a look at the influence of feature ranges. Figure 13.5 shows the histogramed features of the *Iris* data set and it can be noticed that the distributions for the lengths features (s_l, p_l) cover larger intervals than the respective width features (s_w, p_w). For the calculation of distances between data points, this can have the consequence that a difference with respect to a length has a larger influence than a difference with respect to a width, just because widths differences are consistently smaller than length differences. In other words, individual features may dominate distance measures.

feature ranges

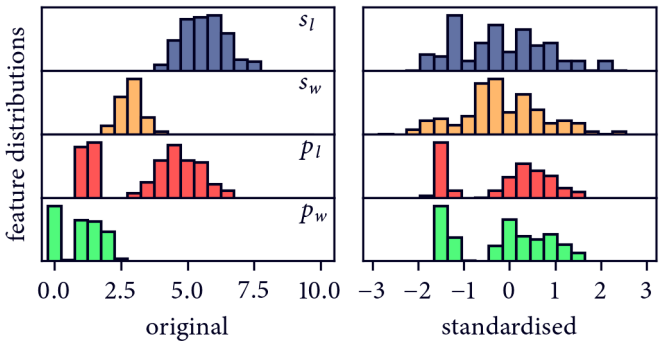


Figure 13.5 *Iris plant* data feature standardisation Feature distributions before (left) and after (right) standardisation of the features through z-scaling, i.e. removal of the mean and normalisation by the standard deviation. Note that it is *not* advised to do this in this case.

To counter the effect of disproportionate distribution ranges of individual features, we can apply a standardisation that will in some sense make each feature equally important. A typical standardisation protocol is found in the so called z-scaling. This will transform feature values (here x_j is the jth component of a single data point) by subtracting the mean of the respective feature μ_j and dividing by the feature's standard deviation σ_j

z-scaling

$$x'_j = \frac{x_j - \mu_j}{\sigma_j} \,. \tag{13.9}$$

As the result, each standardised feature will have a mean of $\mu'_j = 0$ and a standard deviation of $\sigma'_j = 1$. Alternative protocols are min-max scaling that normalises a feature to a given closed interval $[\min_j{}', \max_j{}']$ (e.g. $[0,1]$) by subtracting the minimum feature value and dividing by the feature range

min-max scaling

$$x'_j = \frac{x_j - \min_j}{\max_j - \min_j}(\max_j{}' - \min_j{}') + \min_j{}' \tag{13.10}$$

and max-abs scaling that divides by the maximum absolute feature value and effectively normalises to the interval $[-1,1]$

max-abs scaling

$$x'_j = \frac{x_j}{\max(|\max_j|, |\min_j|)} \,. \tag{13.11}$$

The question is now, if a standardisation or normalisation is actually advisable for the *Iris* data set. All features in this set have the same physical dimension and by re-scaling the features, the relative proportions of the features are distorted. By scaling the widths to the same ranges as the lengths, we equalise their influence on distance calculations but this may lead to false interpretations because the influence of widths and lengths is not the same in reality. Another way to think about it, is that both width and length are coupled quantities that are two sub-features of say an area-feature. By re-scaling the sub-features the jointly described feature is skewed. The *Iris* data set is therefore actually a counter example in which case a strict feature-wise standardisation is not recommended. If the ranges are to be scaled, this should be done proportionally. It would be a possibility to scale the sepal features (s_l, s_w) and the petal features (p_l, p_w) as separate pairs, equalising the ranges corresponding to different leafs while maintaining the proportions of features describing the same leaf but this is still debatable. Another example in which case independent feature-wise normalisation is not appropriate are molecular configurational spaces given as let's say m real-valued features. It is possible to scale all m dimensions proportionally to a new range but setting all of them individually to the same range distorts the structural information of the data. Feature standardisation and normalisation should only be applied if separate features have very different meaning, for example 'age', 'height', and 'weight' in a data set containing samples of different persons. These features may cover very different ranges (e.g.

always normalise?

height: $[150, 200]$, age: $[30, 60]$, etc.) and a combined distance measure to which each feature should contribute equally calls for a re-scaling of each feature to say the interval $[0, 1]$.

mixed data

Different feature ranges are especially encountered in mixed data sets. An example would be a molecular feature space that contains not only real-valued dimensions but also binary features (e.g. interaction indicators). The binary features are naturally confined to the interval $[0, 1]$ and to equate their influence on distance calculations, a proportional re-scaling of numeric spatial coordinates may be an option. We face, however, a much more crucial problem here because the question is, what would be a good distance measure to combine binary indicators with spatial coordinates? Obviously, using the Euclidean distance to which a binary feature contributes a value of 0 or 1 would be difficult to balance against the influence of the numeric features, that is it would be hard to decide what the relative importance of the presence or absence of a single interaction is compared to a structural change along a specific spatial dimension. In this case it may be advisable to use a set of different distance metrics suitable for a comparison along only a subset of the features and to average over the weighted contributions from these distances. Such an approach of computing partial distances that can be re-combined into a single distance is formalised for example in the Gower distance.[401, 402]

reference labels

In the *Iris* data set, each record carries a categorical value that associates it with a biological flower class besides the four measured numerical features. This nominal feature can in principle be included into the feature space as well if it is seen as a source of information on which basis the data points should be grouped into clusters. Here we want to treat the class field differently, namely as a source of reference. We can interpret the points of the data set as labelled data and treat the class-field as a true classification, which can be compared to classifications that are based on a clustering. In other words, we can use the class field for an external validation of clusterings for the *Iris* data and we will consider a clustering good if it is able to reproduce the reference labels. This is based on the premise that firstly the class field represents indeed true information, i.e. we assume that there is no mistake in the assigned biological class, and second we will presume that it is actually possible to reproduce the reference labels using the information of the remaining features.

geometric vs. kinetic clustering

As a last remark, it would also be imaginable to use the ordering of the records (the record ID in table 13.1) as a feature on its own. In this particular case it might be not very meaningful because the ordering of the samples is arbitrary. It may, however, be the case in other data sets that the ordering encodes information itself, as for example in MD data were each data point is a structural snapshot at a given point in time. Clusterings disregarding the ordering of data points that only use the positions of objects in a feature space can be referred to as geometric clusterings while kinetic clustering on the other hand tries to incorporate the temporal relation between clustered objects.

No matter how we choose to represent objects in a data set to be analysed by a clustering and no matter which object properties we consider for this, a clustering $\mathcal{C} = \{C_1, ..., C_k\}$ is always a decomposition of n objects into k subsets $(C_j \in \mathcal{C}) \subseteq \mathcal{D}$. In the next section we will discuss different clustering approaches in a general fashion while selected methods and algorithms are presented in chapter 14.

13.2 Definitions of similarity and clustering categories

At the beginning of this chapter, clustering was described as an analytical process to identify groups of objects that are characterised by strong relationships between objects within the same group and weak relationships between objects in different groups. This conception is wilfully very open and a relationship can be basically anything. Practical clustering procedures need to transform it into something tangible and need to specify how relationships are actually defined and determined. Based on how this transformation is done and how the clustering works, clustering procedures can

clustering categorisation

be roughly categorised. These categories should, however, not be seen too dogmatic and are rather an orientation. Throughout the literature, substantially different categorisations are proposed—and criticised.[243, 383, 384, 403] We already encountered one form of categorisation previously in this chapter with the assessment of how clusterings assign labels to objects (e.g. in a strict or fuzzy manner). This is a very practical way to look at it but it tells us little about the actual clustering. In a way, it is primarily a modelling decision if a clustering uses a certain type of label assignment and methods can be flexible in this regard.

Clustering methods can be understood as being comprised of three different aspects. The heart of each clustering technique is (or I should say should be) a *model* that is some idea of what potential clusters in a group of objects actually are, although the model may not be always obvious for every method. Categorisations based on cluster models are popular and I will address a few of them. Most paramount are *connectivity-* and *prototype-based* models. It should be noted, though, that the borders between models can be blurry and that their association with a specific category may only reflect or emphasise one model property. A good way to describe the cluster model of a given method can also be to think about the kind of output that is produced.

cluster model

A certain model can be paired with an *inductive principle*, which may be either a mathematical formalism—an *objective function* that a model should strive to optimise—or rather a more or less formal description of how a model should be constructed. Such an inductive principle provides an opinion on what is a *good* clustering. Explicit categorisations of inductive principles are rather rare, which may be owed to the fact that for many clustering methods the underlying principle is not exactly easy to grasp so that the focus simply shifts towards the model. When an inductive principle does not manifest itself in a mathematical formalism, it may also become unclear where the model and the underlying principle differ. Inductive principle and model need to match each other but in principle the same model can be evaluated in the light of different objectives and one inductive principle may apply to different kinds of models.

inductive principle

Lastly, a clustering method needs to be realised through an *algorithm* while in turn different *implementations* are possible for the same algorithm. If a clustering is actually applied, it is an implementation of an algorithm that eventually needs to be chosen for it. Clustering algorithms and their concrete implementations produce a model for the clustered data while an inductive principle suggests what the best model should be.[384] Methods can be exact in their formulation of an underlying principle but concrete algorithms may in many cases only provide approximations. Especially, when there is no concise mathematical formulation for what an algorithm should yield, algorithms are only heuristic protocols to construct good models. Algorithms are in general very convertible and the same context (model and inductive principle) can lead to a variety of algorithms and realisations. Categorisations of clustering methods that claim to be focused on the model or the underlying principle are sometimes scrambled with a focus on algorithmic or implementation details, that is in particular the type of data structures that are used or produced. Examples are the categories of *graph based* or *grid based* models. Graph based puts an emphasis on the fact that the output of a clustering is a graph or that the method explicitly uses a graph representation of the data. With some effort, however, most if not all clustering methods could be transformed to use some kind of graph structure—particular ones may just be more obviously suited to be expressed in graph theoretical terms. I would therefore argue that a description of a model as graph based (possibly as a subcategory of connectivity-based) is not very informative and foremost a property of the algorithmic formulation or implementation. Whether a clustering method uses a form of grid, is also more a question of procedure rather than of what the model for the data is. Of course it can be the case that the cluster model decidedly is grid based and a grid is the final result of a clustering (possibly with the grid cells as a certain kind of prototype) but in the way it is normally used, the grid plays often a rather auxiliary role (compare section 14.5). Many clustering methods can be formulated with or without involving a grid.

algorithms

A prominent concept to substantiate object relationships is to leverage the term *similarity*. Similarity is in fact so abundantly used that many alternative definitions of clustering include it in their basic formulation. It can for example be found that the aim of clustering is to identify groups among objects that *maximise intra-cluster similarity* and *minimise inter-cluster similarity*.[384, 387] I personally think, that such a statement can be problematic because of two reasons: first, against what is suggested here, clustering can by far not always be expressed as an optimisation problem in which a certain quantity is minimised or maximised. It may of course well be true that clustering is intrinsically an optimisation (or at least that an objective to optimise for is in general desirable) and that an objective function can just not be established or discovered, or is too hard to solve. But the statement does in this form not aptly describe clustering in reality where it can be studied from a rather procedural level. And second, while 'similarity' is used here without obligations, the term has actually a distinct meaning and by far not every clustering approach does rely on it literally. We have to be careful to not bloat the concept of similarity to an extend where it just replaces 'relationship' in the broad initially proposed definition and becomes essentially hollow. It remains legitimate to say, that clustering has the aim to identify groups of 'similar' objects when it is supposed to mean generally 'alike' or 'somehow related' but similarity as such should probably be treated with a bit of caution. Clustering procedures that use similarity as an underlying idea are based on a similarity function $s(x, y)$ that maps two objects to a single value. Similarity is something that can be measured and compared quantitatively, which means it can take either a low or high value. Instead of similarity, the same can be expressed inversely using a *dissimilarity* function $d(x, y)$. Such a function can be just a common distance metric (compare eq. 13.2 to 13.5). A short distance between objects with respect to the space they are embedded in corresponds to low dissimilarity, i.e. high similarity. Because of the frequently made analogy between dissimilarity and distance, *proximity* can be used synonymously to similarity. It should be noted, however, that similarity measures are not necessarily always classic distance measures. Similarity and dissimilarity functions can be much less formal than distance metrics in the sense that neither non-negativity, symmetry, nor satisfaction of the triangle inequality are strictly required, although especially symmetry may be generally helpful. Possible measures of similarity include divergences, correlations, mutual information, kinetic distances, but also generic notions of pairwise object connectivity as the number of shared neighbours (like in the CommonNN scheme described in section 14.9 and chapter 15 in detail). It can be also something arbitrary like the number of text messages that are send from user to user in a social network. Thinking about data sets in terms of graphs, similarity can be expressed as anything that can serve as the weight of an edge connecting two nodes in the graph.

object similarity

Similarities can be confined to arbitrary value ranges. They can for example be expressed in a binary form, i.e. $1 \equiv$ 'similar', $0 \equiv$ 'not similar'. Normalised distances (dissimilarities) can be converted to a similarity via $s(x, y) = 1 - d(x, y)$, so that similarity is restricted to the interval $[0, 1]$, but it is also possible to have similarities in the interval $[0, \infty]$ with a relation $s(a, b) = d(a, b)^{-1}$ (with $d(a, b) = 0 \Rightarrow s(a, b) = \infty$). In principle, similarities can be obtained through any (non-linear) mapping of distances, say radial functions like for example a generalised Cauchy function $s(x, y; a, b) = 1/(d(x, y)^a + b)$. A combined similarity/dissimilarity measure could furthermore take on values in an interval as say $[-1, 1]$.

similarity values

Clusterings that employ a similarity concept, can produce models in accordance with the idea that based on the similarity of two objects x and y, it can be decided if they should be part of the same cluster. One should refrain, however, from the tempting conclusion that in reverse two points x and y are similar (in the sense of a high pairwise similarity value) in general if they are in the same cluster. Also, an actual maximisation or minimisation of the similarity can not be claimed for all similarity based clustering methods. The property of linking pairs of objects through similarity, affiliates methods with the *connectivity-based* category of clusterings. To use a set of representative connections between

connectivity clustering

objects is a kind of model for a data set where clusters are groups of inter-connected objects. Inductive principles for connectivity-based models can select a set of relevant connections (based on similarities) and can be rather subtle. Conceptually, pairwise similarity to guide clustering models can for example take a strong and a weak form. If for any point x in the data set that is a member of cluster C_i the condition is fulfilled that for any other point y, also a member of C_i, the similarity $s(x, y)$ is larger than the similarity $s(x, z)$ of x and any other point z in cluster C_j with $i \neq j$, this clustering contains strongly similar groups. In a graph picture of the data, this can be related to the identification of strongly connected sub-graphs. In other words, the strong version requires that points within the same cluster are strictly more similar to each other than to points in different clusters. Note that this condition can be fulfilled for more than one grouping of the same object set—in particular it is true for the trivial cases of having only one group (no actual clustering) and having each object in its own group (over clustering). For an actual optimisation of some sort, additional requirements would need to be made as for example that the minimum similarity within a cluster should be as large as possible (that is the maximum dissimilarity between two points within the same cluster is minimal). The weak version of this principle, on the other hand, would only require that for any point x in cluster C_i the most similar point y in the data set is also a member of cluster C_i. In the graph picture, this is equivalent to a search for connected components. Note again, that without any further specification this condition can be fulfilled for more than one grouping of the same object set, as well.

A connectivity-based element can be found in many clustering methods and their models. A common trait of such models is that they do not necessarily make any assumptions about the shape, size, number, or spreading of the clusters that should be found in a data set. These characteristics will be just an implication of how individual objects are connected to each other. Connectivity-based clustering has a predisposition to be formulated using graph concepts. As mentioned already, clusters can be understood in this way as connected sub-sets within the graph for the whole data set. The term *connected component* (see also chapter 7) will reoccur multiple times in the context of connectivity-based clustering. A universal clustering paradigm within the framework of data graphs is to state that the aim of clustering is to *maximise within-cluster density* and to *maximise between-cluster sparsity*. where in this context within-cluster density means many significant connections of objects within the same group and between-cluster sparsity refers to few connections of low importance between objects in different groups.[382]

graph sparsity

Pairwise similarities between objects are used for example in agglomerative clustering, a family of clustering methods that will be discussed in section 14.1 in more detail. This clustering approach uses, in addition to similarity, a so called *linkage* criterion that states how similarity is further treated to assign objects to the same group. Interestingly, agglomerative clustering does in general not maximise the within-cluster similarity while minimising the similarity between clusters overall in the sense of an optimisation according to a global objective function. Instead it provides just an iterative protocol that yields a hierarchy of clusterings by making a series of locally optimal decisions in the sense of the linkage formulation. As such, the model it produces for the data is a tree of clusterings in accordance with a connectivity-based principle. A single optimal solution can be selected from the model hierarchy only by further requirements for example on the number of identified clusters or using a threshold on the similarity.

similarity clustering

Another broad family of clustering methods that make use of pairwise similarities is spectral clustering, which will be briefly described in section 14.2. Through eigenvalue decomposition of a similarity matrix, precisely its graph Laplacian, the input data is embedded into a lower dimensional space that can reveal the cluster structure. The objects in a data set can then be clustered in this embedding using a method of choice. Therefore, spectral embedding is arguably only a helpful transformation that facilitates a clustering and not an actual clustering technique itself. Conceptually,

it is tightly related to the general idea of rearranging a similarity matrix into a block-diagonal form in which clusters appear as blocks of elements with high similarity values.

Besides clustering methods with a connectivity-based model of how clusters should be identified, which is more or less directly tied to object similarity (i.e. connections can be found based on a similarity definition—a connection can be seen as a form of similarity), there are methods that have a different idea about this. One widely-used alternative approach is to infer that a data set can be described as a composition of a number of equally behaved clusters. The paradigm for finding clusters can for these methods sometimes be expressed as a *maximisation of within-cluster homogeneity* or generally as the idea of splitting an inhomogeneous data set into more homogeneous subsets. A similarity concept may play a role here as well but similarity between objects is not fundamental. Because of the shift in the notion of object relationship from pairwise similarity to a comparison of objects with some shared representative, these methods can be categorised as *prototype-based*. Such a method could for example hypothesise that a data set could be modelled by a set of k representatives μ_i so that the following objective function is satisfied:

homogeneity

prototype clustering

$$\min_{C} \left\{ \sum_{i=1}^{k} \sum_{x \in C_i} d(x - \mu_i) \right\} . \tag{13.12}$$

This objective states that the best model for the data is the one that partitions the objects in such a way that the sum over some distance function computed for all points within a cluster and the cluster representative is minimal over all clusters. Each data object will be associated to its best fitting representative. A popular variant of this is the k-means method when the distance function d is the squared Euclidean distance and the used representatives are cluster centroids (*centroid based* is a special type of prototype-based), meaning the arithmetic mean points of each cluster. This method will be addressed in section 14.3. In contrast to connectivity-based methods, prototype-based clustering does commonly make more or less strict assumptions about the number, shape, size, and general distribution of ideal clusters.

Another example for a prototype-based clustering approach are GMMs also known under the family of expectation maximisation algorithms (see section 14.4 for an example). Sometimes these will be explicitly referred to as *distribution based* models. The idea behind these, is that a data set can be modelled by a set of overlapping normal distributions. Each data point is assumed to be sampled from one of these underlying distributions (the prototype they will be associated with) and the aim of the clustering is to identify the set of distributions that represents the observed data in the best possible way. The inductive principle can in these cases often be formulated as a likelihood maximisation. Other related methods may take different kinds of distributions as their basis. Here it becomes particularly clear that similarity between objects plays a very subordinate role and is overruled by the aim to find homogeneous groups within the data of which each can be described by a single simple distribution. In categorisations, these clustering methods are sometimes also called *model based*, which, however, makes little sense before the background that essentially every clustering implies a cluster model of some form.

expectation maximisation

Yet another principle of clustering models, which should be especially important to us, comprises the category of *density-based* clusterings. The central idea from this point of view is that clusters cover regions in the data space in which the object density is relatively high, and that are separated from each other by relatively low density. Already early on in the history of clustering, this was taken as a 'natural' notion of clusters.[404] Presumably, it is the 'discontinuity in "closeness"' that makes us intuitively perceive groups of objects when we see them. A density-based model provides a prescription for how object density can be estimated and consequently how clusters of high density can be identified. As such it is normally predicated that the data set is sampled from an unknown

density clustering

underlying probability distribution. For the actual clustering, it is, however, usually not enough to have a solely density-based model. Eventually, a notion of how object density is evaluated needs to be paired with either a connectivity-based element so that clusters are identified as pairwise connected (groups of) objects of relatively high density, or with a prototype-based element so that cluster representatives are aligned with data regions of high density.

Prototype-based models can be described as being focused on mode seeking, which means they usually try to identify the maxima of a data distribution. Note, however, that density-based clustering in general is sometimes referred to as *modal clustering*. Examples for prototype-based, density-based clusterings are mean-shift and density-peaks (see section 14.10). Connectivity-based models on the other hand can be related to the concept of *level-sets* (see section 14.6) of the approximated probability density. From this perspective, clusters are universally the connected components of a super level-set. Examples for connectivity-based, density-based clustering methods are DBSCAN (section 14.7), Jarvis-Patrick (section 14.8) and CommonNN clustering (section 14.9). As connectivity-based clustering in general, these clusterings make no assumptions on the shape, size, or number of identified clusters. Since the notion of level-sets is intrinsically hierarchical, they also produce in essence a hierarchical tree of clustering results much like in linkage clustering.

Taking a short detour, density-based clustering schemes are also most valuable for the identification of molecular conformational states from molecular simulations. The notion of a *conformation* as an ensemble of molecular structures associated with a potential energy minimum (a region of high Boltzmann density) separated from other minima by transition regions (of low Boltzmann density) has a very natural and intuitive correspondence to the density-based clustering assumption. The property of these clusterings that they find the high density regions independent of the spatial layout of these regions allows to identify molecular conformations truly as what they are, detached from purely structural information. A conformational ensemble can be structurally quite diverse as long as the respective atomic arrangements are separated by no (or very low) energetic barriers. Although density-based clustering is in principle geometric, i.e. does not make explicit use of temporal information, the result in a molecular context can be interpreted as kinetically relevant. A low energetic separation of molecular structures can be equated with fast interconversion, whereas conformations identified as different clusters will be in relatively slow exchange. The definition of a conformation as being associated with an energetic minimum is of course somewhat blurry (especially for complicated potential energy surfaces), but so is the definition of density-based clusters. The complete conformational ensemble of a molecule can be seen as a hierarchy of sub-ensembles. With increasing granularity, one can split the whole ensemble first into the most clearly separated sub-ensembles (the ones with the highest barriers in between them) and continue by further splitting along the smaller separations. Along the way, individual conformations can be considered with varying extent, shrinking from the complete region they exist in up to the transition boarders, down to a splitting point or until they are reduced to only the closest structures to the minimum. The term *core set* is often used in this context to denote those data points that are a 'sure' member of a cluster. Illustrative examples for this association can be found in applications of CommonNN clustering in chapter 15.

molecular conformations

core sets

Sometimes, clusterings are categorised by the fact whether they produce a *hierarchical* or *flat* result.[2] It is true that individual clustering methods can be predominately hierarchical or flat: agglomerative clustering is intrinsically hierarchical while k-means gives a single flat partitioning. I would argue, though, that it is not profound to a method to be either of it but that it is rather a consequence of the existence or missing of an explicit objective function according to which a

hierarchical vs. flat

[2]Instead of flat, the opposite of hierarchical is regularly also referred to as *partitional*. I will avoid this, however, since there can be some confusion with the term *partition* used to refer to a crisp, non-overlapping, possibly also exhaustive clustering (e.g. full partitioning vs. partial clustering).

single clustering result can be optimised. Flat clusterings can be often applied quasi-hierarchical by reusing the method on a subset of the data identified as a cluster. Hierarchical clusterings can be used quasi-flat by taking a certain level of the hierarchy as a single partition, i.e. by for example applying a termination criterion. CommonNN clustering is a good example for a method that can be practised in a flat manner (using a specified density threshold) or fully hierarchically.

(non-) parametric clustering

Another form of categorisation judges clustering methods by the parameters that they depend on. Prototype-based methods are typically *parametric*, which means that for example the number of clusters or their constitution needs to be pre-defined. An optimal solution to the clustering problem is sought given the constrains of the mandatory parameters. The outcome of the clustering can depend non-trivially on these parameters. In contrast to this, *non-parametric* clusterings do not depend on parameters that pre-define the number of obtained clusters or their constitution. They can still use parameters but these are seen as tuning parameters to set the granularity of the process. The outcome of the clustering usually varies systematically with these parameters. Parametric clustering can be related to flat clustering that produces a single result while non-parametric clustering can be related to hierarchical clustering in the sense that a screening of the granularity parameter produces a hierarchy of clusterings.[383]

clustering clusterings

I would like to close this section with mentioning a very different, yet tempting approach to categorise clustering methods, that is to compare them in terms of the output they produce on example data sets.[403] Using an index of external validation, it can be evaluated how well the cluster label assignments obtained through two different methods agree, without assuming that either one of them represents the ground truth. By comparing a set of clusterings in a pairwise fashion, one does essentially build up a similarity matrix since the used index of external validation can be interpreted as a similarity measure between clustering results. Based on this matrix it is possible to cluster clusterings and to find groups of seemingly related algorithms. This perspective offers a purely assignment centred approach beyond categorisations based on how clustering methods assign labels, the structures they produce, how they operate, or what the cluster model is. Any categorisation can only provide an idea about how a specific method may be distinguished from others but the myriad of available clusterings may after all be judged solely by whether they find the result that fits the application best.

{ 14 }

Clustering methods

Many tools for the same purpose

The pool of available clustering methodologies to choose from is very large. As outlined in section 13.2, they may differ in the kind of cluster model they try to adapt to the clustered data, the presence or absence of an explicit objective (that can be optimised in a mathematical sense), and in the algorithms that will eventually make them usable. Technically, a lot of clustering algorithms are actually a combination of more than one fundamental algorithm, which can lead to a possibly even larger number of detailed algorithmic variations and implementations. This can make things difficult if we want to compare clustering methods from a bird's eye perspective since what can be ultimately compared are always only individual algorithms and specific implementations thereof. While in principle different realisations of the same method should yield the same if not very similar results, this may not be true in reality in any case, for example when they employ constrains or make decisions in inconclusive situations. On top of contingent qualitative differences, individual implementations of basically the same algorithm can vary drastically in their efficiency.[405]

The name used to refer to a clustering can either describe the basic idea that the method is based on, or rather a specific algorithm, or even just a concrete implementation. In the following, I will discuss a few commonly used clustering methods with corresponding algorithms and (rough) example implementations. The selection is influenced by whether knowledge about them is potentially useful to fully understand CommonNN clustering and to put it into context. For each presented method, I will try to point out what is the cluster model, what could serve as an inductive principle, what are typical variations, and what are suitable cases of applications, that is for which data sets is a method in general suitable.

14.1 Linkage clustering

Let's start our short survey of clustering procedures with one of the ancestral methods that has its origins back in the early 1950s.[406] *Single-linkage* clustering is an archetype of connectivity-based clustering, which makes use of pairwise object similarities or dissimilarities to find groups of objects. The method aged rather well and is still widely applied. As one of the standard clustering techniques, it has been used for the development of a multitude of other clustering methods or as a building block in more complex procedures.

In a nutshell, single-linkage clustering is based on the following idea: the two most similar objects in a given data set should be grouped into the same cluster. The argument can be continued, so that the next two most similar objects in the data set should also be grouped into the same cluster, followed by the next most similar pair of objects, and so forth. This reasoning is intrinsically hierarchical. *principle*
It leads to the somewhat unfortunate use of *hierarchical clustering* as a synonym for single-linkage clustering and closely related methods. In a bottom-up fashion, objects are iteratively merged into larger and larger clusters at decreasing granularity of similarity until eventually all objects end up in

the same cluster. At each step, merged objects are less and less similar. An alternative term to describe this bottom-up approach of this method is *agglomerative* clustering. Figure 14.1 illustrates this process with a simple example.

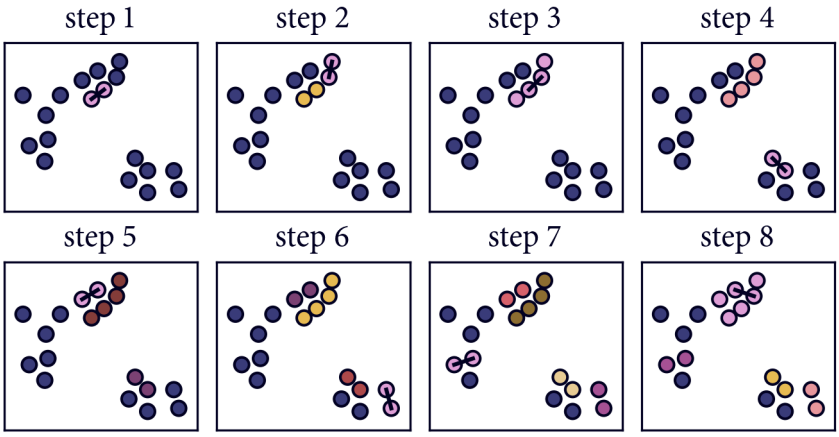


Figure 14.1 Single-linkage clustering in a nutshell
Data points are merged iteratively into clusters *bottom-up*, two data points at a time. In step 1, the two closest (most similar) data points are connected and form the first cluster. In step 2, the next two closest points are connected and form a second cluster. In step 3, the two next closest points are already part of separate clusters in which case the two sub-clusters are merged into a bigger super-cluster. The process is continued until all points are part of the same cluster or until a stoppage criterion is reached. A user might be interested in either the hierarchy of clusters or clusters at certain stages.

A hierarchy of clusterings with atomic objects at the bottom and a single cluster containing all objects at the top is called a *full* hierarchy. The connectivity-based model that will be produced by single-linkage clustering is essentially a rooted tree of clusterings. An individual clustering result may be selected from the hierarchy for example by adding the constraint that a specific number of clusters should be obtained. Such an additional demand on the cluster result can be seen as a *stoppage* or *threshold criterion*. Instead of the (full) hierarchy, the preferred model for the data will consequently be a flat clustering.

stoppage criterion

Figure 14.2 shows an application of single-linkage clustering to the *Iris* data set. Note, that the used similarity measure is just the Euclidean distance here. A specification of three target clusters does, however, not yield a result that is in good agreement with the expectation. We can observe that the orange cluster has swallowed up the green cluster almost entirely. Such behaviour is not untypical for single-linkage clustering that has therefore a bad reputation for being sensitive to chaining effects, i.e. to spurious data points that prevent larger clusters from separating.

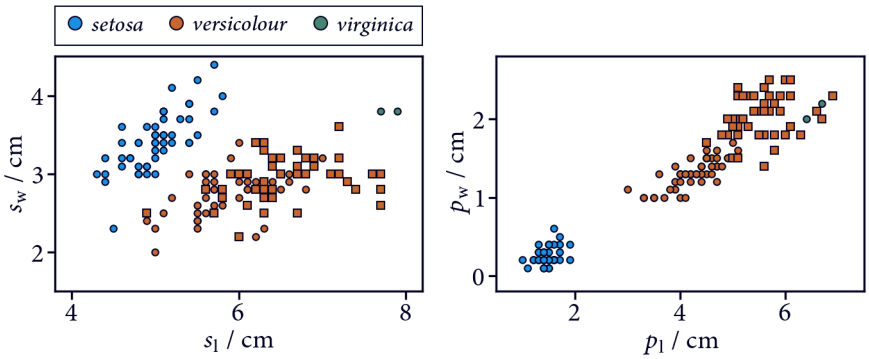


Figure 14.2 ***Iris* data set single-linkage (3 clusters)**
Data points (compare figure 13.3) with cluster labels found by agglomerative clustering (`sklearn.cluster.AgglomerativeClustering` using single-linkage). 68 % of the cluster labels match the true classification labels. Non-matching assignments are marked with squares.

Alternatively, the hierarchical tree of clustering results can be investigated to select a specific clustering at an appropriate distance (similarity) threshold. A typical way to plot single-linkage hierarchies to this effect is to use a dendrogram as shown in figure 14.3. In such a plot, the similarity measure is put on one axis while the indices of individual data points are shown on the other one. *dendrogram*
A merge of points and respectively clusters into a bigger cluster is represented by two legs of a tree that meet at a specific similarity level. From this, we can identify a similarity level at which (among others) three clusters are obtained that agree better with the expectation (see figure 14.4). The cluster hierarchy can give deep insight into the structure of a data set. Of course it is also possible to select a combination of clusters as branches of the tree with differing similarity thresholds. It may, however, become increasingly difficult to fully grasp these hierarchies when larger data sets are considered.

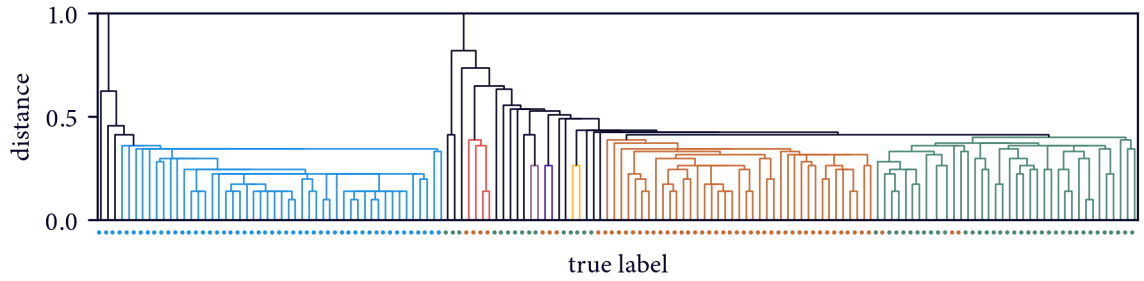


Figure 14.3 ***Iris* data single-linkage dendrogram**
Cluster hierarchy (compare figure 14.2). At the lower end of the dendrogram are individual data points where a coloured dot indicates the true class label. The legs of the tree show which points are merged into clusters with a respective distance (similarity) at which the merge occurs. Selection of a distance threshold $d = 0.41$ results in seven clusters (coloured legs) among which the three largest clusters match the true classification rather well.

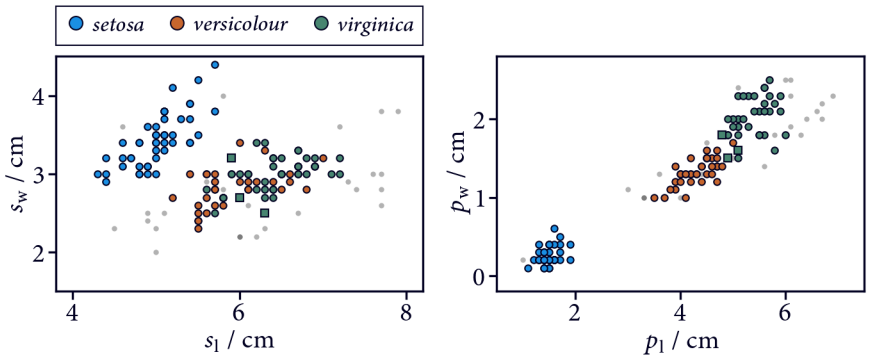


Figure 14.4 ***Iris* data set single-linkage (distance threshold)**
Data points (compare figure 13.3) with cluster labels found by single-linkage clustering after choosing a distance threshold from the cluster hierarchy (compare 14.2). 81 % of the cluster labels match the true classification labels (98 % if smaller clusters are neglected as noise). Non-matching assignments are marked with squares.

SciPy hierarchy

The way how such a dendrogram can be communicated deserves special attention and a good example for that is given by SciPy with the `scipy.cluster.hierarchy`. Let's say we have a set of n data points. In each single-linkage clustering iteration, two clusters a and b are joined to form a new parent cluster c. Clusters are labelled with successive integers (starting with 0), where the first $n-1$ clusters are the singletons represented by the n data points in the set. There will be $i = n-1$ iterations, giving rise to the new clusters $\{n, n+1, \ldots, 2n-2\}$ that will be collected in rows of a $i \times 4$ hierarchy matrix Z. The elements $Z_{i,0}$ and $Z_{i,1}$ contain the labels of the clusters that are merged to give the $c = (n+i)$th cluster. $Z_{i,2}$ holds the distance value of the respective merge and $Z_{i,3}$ holds the total number of data points in the new cluster.

Hierarchies in this format can be subjected to a lot of Scipy's hierarchy related functionalities (including the plotting of dendrograms) and it may thus be beneficial to adhere to this format in general. As a plus, storage of such a matrix is relatively efficient. It may be, however, a disadvantage in certain situations that at each iteration only the information on which clusters are merged is kept while the information on which points were responsible for the merge is lost.

i	c	a	b	d_{ab}	size
0	5	0	1	d_0	2
1	6	2	3	d_1	2
2	7	4	5	d_2	3
3	8	6	7	d_3	5

Table 14.1 **SciPy single-linkage hierarchy format** Example Z matrix for a set of 5 data points. In iteration i, cluster c is formed by merging a and b at distance d_{ab}.

latent space

Since single-linkage clustering does in essence only make use of pairwise similarities and not of individual object properties, single-linkage can be understood as being detached from the (metric) data space that objects may be embedded in. In this sense, the data space is treated as a latent space. Object coordinates with respect to the data space may well be necessary to compute similarities in the first place but as far as the clustering is concerned similarities could have a discretionary source. In principle, single-linkage clustering can be adopted for arbitrary similarity concepts.

other linkages

Besides single-linkage, which merges clusters at each step by the minimum distance between existing clusters, there are also other types of linkages. In *complete-linkage* for example, the maximum distance between existing clusters is taken to decide which clusters are merged next. *Average-linkage* uses the mean distance between clusters. A linkage uses a similarity measure and provides a prescription of

how to extend the similarity estimate to non-singleton clusters. In a way, a linkage criterion is also a kind of (greedy) clustering objective, suggesting an optimal cluster merge at each hierarchy step. Depending on the linkage, the choice of meaningful distance functions may be limited.

implementation

Traditionally, linkage clustering is thought of and implemented based around a distance or similarity matrix. At each step, the minimum value in this matrix decides over which points are merged. For the next step the matrix has to be reduced by the just used elements and respectively modified with the distances to the newly formed cluster. A generic way to realise most common linkage schemes at this point is given by the Lance-Williams dissimilarity update formula.[407, 408] A procedure like this has a rather costly runtime complexity of $\mathcal{O}(n^3)$ with respect to the number of points in the data set n. If a distance matrix is explicitly stored during the clustering, it also has a memory demand on the order of $\mathcal{O}(n^2)$. It should be noted, though, that in some cases leveraging priority queues to store and retrieve similarities in the needed order is preferable over the pure matrix centred approach.[409]

single-linkage and minimum spanning tree

For single-linkage clustering specifically, there exist also other neat solutions. SLINK clustering deserves to be mentioned here for example.[410] In the context of CommonNN clustering, which will be discussed later, however, one fact is of particular interest: all the information that is required for the single-linkage clustering of a data set is contained in a minimum spanning tree (MST) for the data.[411] This basically means that the clustering problem can be substituted by the well studied problem of building MSTs (see also chapter 7). Through a revision of the edges in a MST in order of there similarity weight, a hierarchy can be constructed that is equivalent to a single-linkage clustering result.

divisive clustering

As a last remark, it is in principle also conceivable to use *divisive* top-down strategies instead of the described agglomerative approaches to split an initially un-clustered data set into smaller and smaller sub-clusters. Because there exists a combinatorial number of $2^{n-1}-1$ possible solutions for each cluster split, where n is the number of objects in the cluster, divisive methods are, however, computationally very difficult to operate and are usually heuristically heavy as for example in DIANA clustering.[412] Recall that the number of possible merges in agglomerative clustering only amounts to $n(n-1)/2$, where n is the current number of clusters available for a merge. Divisive strategies play, however, an important role in the context of the interesting question of how to cut graphs in an optimal way under various constraints. Consider for example the *minimum cut* problem of splitting a graph into components by removing a minimum number of edges or edges with a minimal total weight.[413] A variation of this would be to find a *normalised cut* that is a split into components where the cost of the split amounts to the total weight of the removed edges normalised by the total edge weight of the resulting components,[414] which favours cuts into 'balanced' components. Spectral clustering for example, which will be addressed in the following section, can be considered an approximate solution to the relaxed normalised cut problem.[415] Another common example for graph cut problems is *sparsest cut* where the total weight of removed edges is normalised by the number of nodes in the smallest of the resulting components, which favours components of equal size.[416]

graph cuts

14.2 Spectral clustering

Pairwise data object similarities are the fundament of another broad family of clustering procedures referred to as *spectral* methods. As mentioned in section 13.2, a spectral embedding of a data set does not actually represent a clustering itself in the classic sense but rather tries to provide a convenient transformation of the data space into a lower dimensional representation in which a clustering can be done easier or the cluster structure becomes more obvious. In general, a spectral

clustering requires a similarity matrix S (often also called affinity matrix in this context) as input. From that, one constructs the graph Laplacian L as[415]

graph Laplacian

$$L = D - S\,, \tag{14.1}$$

where D stands in for the diagonal degree matrix $D_{ii} = \sum_j S_{ij}$. Note that the elements of the similarity matrix correspond to edge weights $w(i, j)$ between a pair of data objects i and j when the data is thought of in a graph representation (recall that the degree of a graph node is the total weight of edges connecting the node to other nodes as defined in chapter 7).

S is generally required to describe an undirected graph, i.e. it has to be symmetric which does also make L a symmetric matrix. Furthermore, the used similarities need to be well-behaved, which means they are not allowed to be negative and need to be actual similarities, not distances for instances. A typically used similarity is for example found in symmetric k-nearest graphs in which a pair of data points is considered similar ($w = 1$) if one of the points is a k-nearest neighbour of the other or if both points are mutually among their k-nearest neighbours. Another example for a continuous similarity are Gaussian neighbourhoods $w(i, j) = \exp(-||x - y||^2/(2\sigma^2))$ for a given σ-value, here with x and y as the coordinates of data point i and j.

similarity examples

We are then interested in the (smallest) eigenvalues and corresponding eigenvectors of L. In practice there are a few possible variations on how to find those and on how to normalise L beforehand,[415, 417] but let's ignore this here for the sake of simplicity. L has n (the number of graph nodes) non-negative eigenvalues $\lambda_1 \leq \lambda_2 \leq, ..., \leq \lambda_n$, with $\lambda_1 = 0$ of which the corresponding eigenvector is a constant one-vector. Laxly spoken, the neat thing about this is that the eigenvalues can be interpreted as a cost of separating the input graph into clusters. The kth eigenvalue gives us a separation of the graph nodes into k disjoint sets, that is if we are interested in an isolation of two clusters, we have to consider the 2nd eigenvalue λ_2. Zero-valued eigenvalues (no cost) are an indicator for the number of connected components (initially disjoint sub-graphs) in the input graph, i.e. if there are two zero-valued eigenvalue $\lambda_1 = \lambda_2 = 0$ the graph contains two connected components that are not connected to each other. From the associated eigenvectors, we can get the cluster memberships of individual graph nodes by evaluating their projection onto the first k chosen eigenvectors. This is typically done for example by a k-means clustering (see section 14.3) in the resulting low dimensional projection.

evaluation

Figure 14.5 shows an example for a spectral embedding of the *Iris* data set. In figure 14.5a we can see the eigenvalue spectrum for a Laplacian constructed from k-nearest neighbour similarities. Since we are interested in three clusters to be isolated, we focus on the first three smallest eigenvalues. While figure 14.5b shows the input similarity matrix (that exhibits a block diagonal form because the input point samples are ordered by their true classification label), figure 14.5c illustrates the 2-dimensional projection of the data points onto the 2nd and 3rd eigenvector. In this low dimensional projection, the group structure of the data becomes well identifiable.

A particular form of spectral clustering, which plays a considerable role before the background of MD and the estimation of kinetic Markov models to identify clusters of molecular metastable conformations, is found in Perron-cluster cluster analysis (PCCA).[417, 418] This method solves an eigenvalue problem for a stochastic matrix T, namely for example a transition probability matrix modelling a molecular trajectory as a Markov chain. The eigenvalues of this matrix start with $\lambda_1 = 1$ (the Perron eigenvalue) and one is typically interested in the next smaller eigenvalues for the identification of kinetic clusters. The eigenvectors of T are the same as those of a corresponding graph Laplacian L and so is the way how one uses a low dimensional projection onto these eigenvectors for the separation of weakly connected sub-blocks within T. The clustered objects are in this case the Markov states on top of which T was constructed and the obtained clusters can be interpreted as metastable conformational states between which transitions are slow (i.e. rare). PCCA as a method

PCCA

does as such, however, not refer to the spectral embedding part but rather to the procedure of how to analyse the eigenvectors as an alternative to the wide use of k-means. In its basic form, it exploits the sign structure of individual eigenvectors to assign states to one of two groups while iteratively considering higher eigenvectors. The more recent robust implementation does rely on the property of the eigenvectors that these ideally form a k-dimensional simplex if one considers k eigenvectors at the same time.[419]

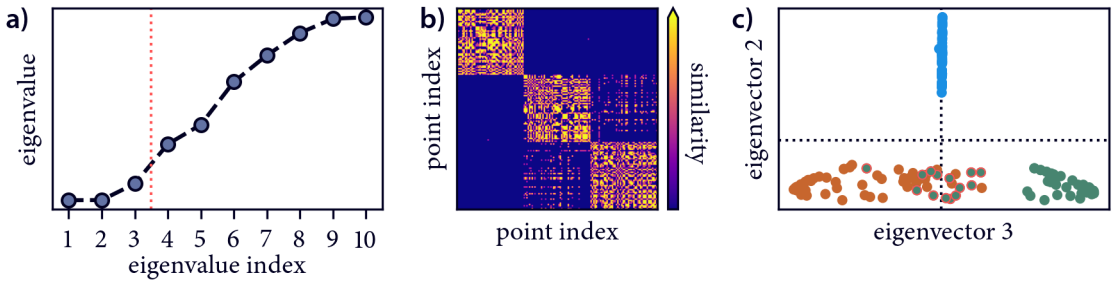


Figure 14.5 Spectral embedding of the *Iris* data set
a) The first ten eigenvalues in the sorted eigenvalue spectrum of the graph Laplacian. We want a partitioning of the data into 3 components based on the first three eigenvalues. **b)** Affinity (similarity) matrix in block-diagonal form with matrix elements coloured by value. **c)** Embedding of the data into the reduced space defined by the 2nd and 3rd eigenvector of the Laplacian. Points are coloured by their true classification label. The identification of clusters can be done by separating the points along each eigenvector (e.g. below and above 0 indicated by the dashed lines) or by using another clustering approach, e.g. k-means (see section 14.3). Points highlighted with a red outline will probably be assigned to the wrong cluster.

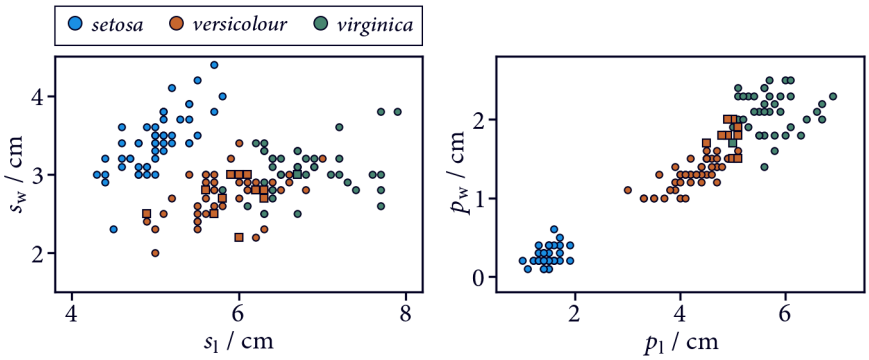


Figure 14.6 *Iris* data set spectral clustering (3 clusters)
Data points (compare figure 13.3) with cluster labels found by spectral clustering (`sklearn.cluster.SpectralClustering` using symmetric k-nearest neighbour similarities with $k = 26$). 91 % of the cluster labels match the true classification labels. Non-matching assignments are marked with squares.

14.3 *k*-Means clustering

The k-means method is arguably the most popular of all clustering procedures.[383] Being not much younger than single-linkage clustering, k-means has quite a long history and has been studied extensively from various perspectives.[420] Its popularity may be not least due to a solid mathematical framework. The fundamental idea behind this method, is that a set of objects $\mathcal{D} = \{x_1, ..., x_n\}$ can be partitioned into a fixed number of k subsets $\mathcal{C} = \{C_1, ..., C_k\}$, so that $\bigcup_{i=1}^{k} C_i = \mathcal{D}$ and $C_i \cap C_j = \varnothing \; \forall \; i \neq j$, and where each C_i is characterised by a representative μ_i, so that $\mathcal{D}$ can be

modelled by the set of representatives $\mathcal{M} = \{\mu_1, ..., \mu_k\}$. The inductive principle underlying the k-means method is a particularisation of equation 13.12

$$\min_{\mathcal{C}} \left\{ \sum_{i=1}^{k} \sum_{x \in C_i} d(x - \mu_i) \right\} = \min_{\mathcal{C}} \left\{ \sum_{i=1}^{k} \frac{1}{2|C_i|} \sum_{x,y \in C_i} d(x - y) \right\} \tag{14.2}$$

with d being the squared Euclidean distance in m dimensions

$$d(a, b) = d^2_{\text{Euclidean}} = \sum_{i=1}^{m} (a_i - b_i)^2 . \tag{14.3}$$

Algorithms implementing k-means strive to optimise the partition $\mathcal{C}$ according to equation 14.2, which states literally: '*pick the model (the set of k representatives) that minimises the total squared error*',[384] where *error* needs to be understood as the deviation of objects from the representative for the cluster they where assigned to. It is important to emphasise, that the squared Euclidean distance is used here, which requires that the clustered data points $x \in \mathcal{D}$ are given as feature vectors in $\mathbb{R}^m$. The representatives that minimise in this case the sum of squared distances for a given partition $\mathcal{C}$ are the arithmetic mean points of each cluster $\mu_i = 1/|C_i| \sum_{x \in C_i} x$. These so called *centroids* constitute a special form of prototype-based model. A minimisation of the sum of squared distances between points in a cluster and the mean is related to a minimisation of within-cluster variance via $|C_i|\sigma_i^2 = \sum_{x \in C_i} d^2_{\text{Euclidean}}(x - \mu_i)$. A minimisation of intra-cluster variance implies a maximisation of inter-cluster variance according to the law of constant variances.[405] In turn, it is also equivalent to a minimisation of the squared pairwise distances between points of the same cluster as stated by the right side of equation 14.2. Although I am reluctant to say that k-means is similarity-based, it is true that each point in the data set will be assigned to the cluster with the closest (that is most similar) centroid and that points in the same cluster can be characterised by a shared similarity to the same centroid. Also, a minimisation of squared pairwise distances between points of the same cluster can be understood as a maximisation of within-cluster similarity on average. Note, however, that two individual points in the same cluster can still be less similar to each other than to points in a different cluster. In particular, for any given point the most similar points may not always be found within the same cluster. As pointed out in section 13.2, the paradigm of the method may be more aptly described as a maximisation of within-cluster homogeneity.

The dependence of k-means on the squared Euclidean distance (L_2 norm) is non-negotiable. To exchange the distance metric that evaluates the deviation of objects and their cluster representatives means to change the inductive principle underlying the method (equation 13.12) and implies a different kind of prototype-based model for the data. The mean minimises the sum of squared deviations within clusters. Using for example the Manhatten distance (L_1 norm) instead, results in an objective for the identified clusters that is optimised by their medians—not their means. The median minimises the sum of absolute deviations within clusters. This is incorporated in other clustering methods like k-medians or in a generalised form for arbitrary metrics in PAM (k-medoids), where the prototypes are constrained to be among the clustered objects themselves so that $\mu_i \in \mathcal{D} \; \forall \; i$.[421] It is, however, possible to transform the clustered data in a way so that the Euclidean distance after the transformation corresponds to a different distance before the transformation, which allows the application of k-means for example in the context of cosine similarity, covariance (Mahalanobis distance), and correlation.

Equation 14.2 can be computationally quite hard to satisfy exactly.[422]. A classic algorithm that provides an iterative, locally optimal solution to the problem was presented by Lloyd,[423] but there are many other proposed heuristic algorithms.[424] It is based on the rational of two alternately applied optimisation steps: first, given a set of current centroids $\mathcal{M}$, the currently best partition of the data $\mathcal{C}$ can be constructed by assigning each data point to the closest centroid so that

$C_i = \{x \in D \,|\, d(x, \mu_i) \leq d(x, \mu_j) \;\forall\; j \neq i\}$. Second, given a current partition $\mathcal{C}$, the currently best centroids can be computed as the mean of the data points in each cluster. Starting with an initial set of centroids, the two steps can be repeated until for example the positions of the centroids or the within-cluster variances or the cluster assignments are converged, or if a maximum number of iterations is reached. It is generally considered an asset of the k-means method that this standard algorithm is easy to implement and computationally cheap. A single iteration requires only one distance calculation for each pair of n data points and k cluster centres while a single distance calculation requires a summation over contributions in m dimensions. The overall runtime complexity can therefore be given as $\mathcal{O}(nkmi)$ with the total number of iterations i. If k and m are fixed, the cost of the algorithm scales basically linear with the number of data points, assuming that the number of needed iterations is relatively small as well.

A problem with the Lloyd algorithm in this basic form, is that the found solution is not necessarily globally optimal and that it depends on the starting condition, that is on the initially chosen positions of the cluster centres. For randomly placed starting centroids, the final result is non-deterministic. Also, the number of necessary iterations to reach convergence can vary and there is the danger of picking unlucky candidates that may lead to empty clusters. Figure 14.7 demonstrates this with a simple example. A commonly accepted workaround, is to run the algorithm multiple times with different starting conditions and to select the best result according to equation 14.2, which would be the one in figure 14.7b (the k-means objective is denoted by J_2). Many k-means implementations have this already built in and proceed like this by default. Alternatively, swapping procedures can be employed to escape local minima.

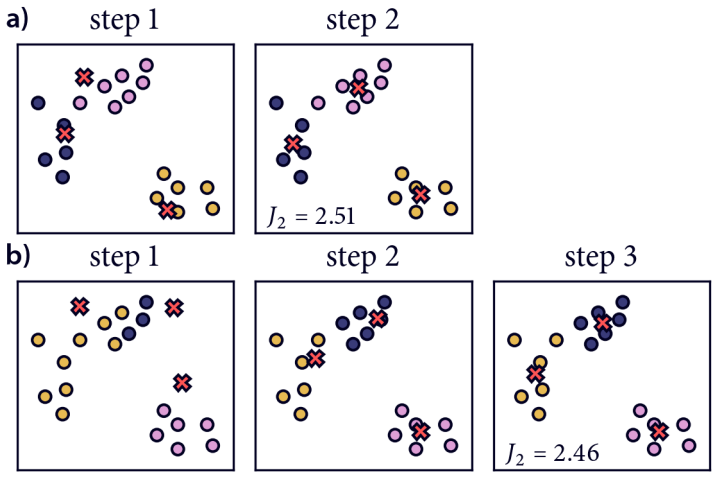


Figure 14.7 *k*-Means in a nutshell Clustering with two different sets of $k = 3$ randomly placed initial centroids in **a)** and **b)**. While the results are stable with respect to the assignments of the cluster in the lower-right corner, they do not agree for the other two clusters. Convergence in terms of changing centroid positions is reached more quickly in **a)**.

Nonetheless, since the choice of the starting centroids is so critical, a lot of effort went into the development of improved initialisation schemes.[425] Among the most successful strategies, is what is known as the k-means++ method.[426] Starting with an empty set of selected starting centroids $\mathcal{M}$, the first centroid to be added is chosen as a random point $x \in \mathcal{D}$. By defining the function $d_{\min,\mathcal{M}}(x)$ as the minimum distance between a point of the data set to already selected centroids in $\mathcal{M}$, the next centroids x' are added one by one with probability $d_{\min,\mathcal{M}}(x')^2 / \sum_{x \in \mathcal{D}} d_{\min,\mathcal{M}}(x)^2$. This should ensure that the starting centroids are well distributed over the whole data set to speed up convergence and decrease the chance to pick unlucky candidates.

Another potential drawback of k-means, is that the number of clusters that should be identified through the clustering needs to be specified beforehand. If the aim of the clustering is to isolate distinct groups of data points, we ideally need to have an idea about how many groups there are in the first place. When a guess is difficult to make, a typical strategy would be to try clusterings with different

values of k and to pick the best result. But how could we judge what would be best? Equation 14.2 is not very suitable to compare clusterings with different numbers of clusters as the within-cluster variance does always decrease with larger k. It can still be used, though, by trading within-cluster variance against the assumption that a low number of k is generally better, in the spirit of optimising this criterion with a minimum complexity in the obtained model. Figure 14.8 shows a plot of this objective versus k for the *Iris* data set. One could now choose the k-value in the plot up to which the sum of squared distances of data points to the centroid of their cluster decreases relatively fast, and after which the decrease is substantially diminished, which is given for $k = 3$ (although admittedly not very clearly) in the present example. This is called the 'elbow' method, as the optimal value for k is presumed to be found where the plotted line shows a kink in reminiscence of a bent arm.

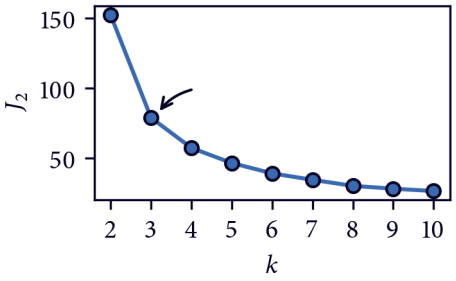


Figure 14.8 Choosing the k in k-means (elbow plot) The plot shows the sum of squared deviations of points from their respective cluster center, i.e. the evaluation of the k-means objective J_2, versus the number of identified clusters. The within-cluster variance is also termed *inertia* or *distortion* in the literature. The optimal k is indicated by the arrow.

Alternatively, the silhouette score can be used for a similar analysis (figure 14.9).[427] The silhouette score for a single data point is defined as

$$s_{\text{silhouette}}(x) = \frac{\bar{d}_{\text{nearest}}(x) - \bar{d}_{\text{same}}(x)}{\max\left(\bar{d}_{\text{same}}(x), \bar{d}_{\text{nearest}}(x)\right)} \tag{14.4}$$

with $\bar{d}_{\text{same}}$ as the mean distance between a point x and all other points in the same cluster, and $\bar{d}_{\text{nearest}}$ as the mean distance between a point x and all other points in the next nearest cluster (by centroid distance). The score is bounded by –1 and 1 where values close to 1 are better in the sense that within-cluster distances are low while between-cluster distances are high. In the present example, a preference for $k = 2$ or $k = 3$ is conveyed by the fact that many data points have silhouette coefficients above the average (indicated by the dotted vertical line) and the individual clusters show a distinct elbow-like characteristic (few points with low scores and many points with high scores). The relative sizes of the clusters can also be taken as an indicator, although it depends on whether equally sized clusters are actually desirable.

A third validation technique is illustrated in figure 14.10 with the Calinski-Harabasz score, also known as the variance ratio criterion.[428] In contrast to the two previous indicators, this score is optimal for a given clustering if it is maximised, and one should look for a maximum in the plot of it versus k. The score is defined for a partitioning of n data points into k subsets as

$$s_{\text{Calinski–Harabasz}}(x) = \frac{\operatorname{trace}(\Sigma_{\text{inter}})}{\operatorname{trace}(\Sigma_{\text{intra}})} \frac{n-k}{k-1} \tag{14.5}$$

with the within-cluster covariance matrix (the sum of covariance matrices for individual clusters)

$$\Sigma_{\text{intra}} = \sum_{i=1}^{k} \sum_{x \in C_i} (x - \mu_i)(x - \mu_i)^\top \tag{14.6}$$

and the between-cluster covariance matrix (the covariance of the cluster centres weighted by cluster size) where $\mu_{\mathcal{D}}$ is the mean of the complete data set

$$\Sigma_{\text{inter}} = \sum_{i=1}^{k} |C_i| (\mu_i - \mu_{\mathcal{D}})(\mu_i - \mu_{\mathcal{D}})^\top . \tag{14.7}$$

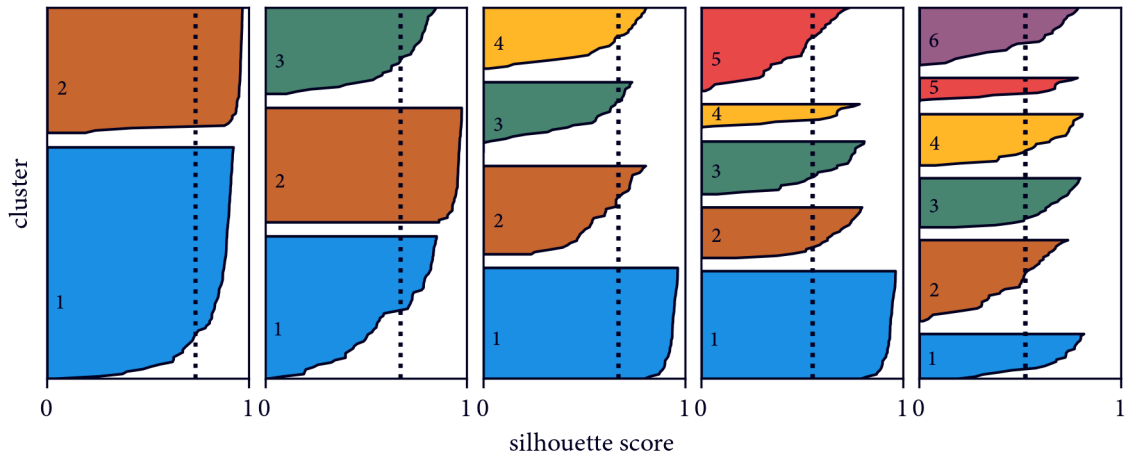


Figure 14.9 Choosing the *k* in *k*-means (silhouettes)
Silhouette coefficients of the obtained clusters for $k = 2, 3, 4, 5$, and 6. The profile of the scores for points in specific cluster is generally considered good if it shows a clear kink and the majority of data points has a score above average (dotted vertical line). The decision between $k = 2$ and $k = 3$ remains somewhat ambiguous in this case. Relatively equal cluster sizes speak in favour of $k = 3$ while the individual cluster profiles are more distinctly elbow-like for $k = 2$.

This score is high if within-cluster variances are low and clusters are well separated. In the present example, the Calinski-Harabasz score strongly prefers the clustering with $k = 3$.

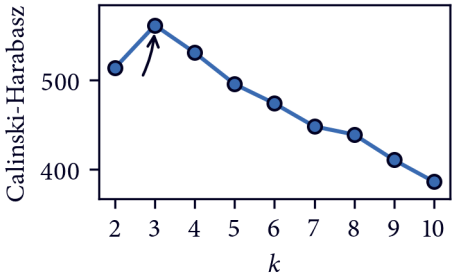


Figure 14.10 Choosing the *k* in *k*-means (Calinski-Harabasz) The clustering with $k = 3$ is clearly preferred because the score is maximised for this number of clusters.

The three presented evaluation techniques are all internal validation criteria, that is they judge the obtained clusterings based solely on the grouping itself. As mentioned in the introduction of this chapter, internal validation works well if we have a concrete idea about how ideal clusters should be constituted. The *k*-means approach has a strong opinion about this and the used validations have strong opinions on their own that are essentially very similar. Identified clusters are in general ideally compact, convex (globular), equally sized and well-separated. With this assumption in mind, we can make a judgement about which *k*-means clustering agrees best with our expectation. I would like to stress again that this validation tells us very little about the actual quality of the clustering as the true group structure of the data set can still differ from our expectation. A clustering not aligned with the ideal image of clusters represented by these validations will be ranked poorly even if it may reflect the true nature of the data better.

For the *Iris* data set, we are in the luxurious situation of having true group assignments available, on the premise of course that the provided expert classification into groups of plant species is indeed true. In this case, we can also validate the obtained clusterings by comparing them to the given group assignments using a set of external validation criteria, for which a few examples are condensed in figure 14.11.

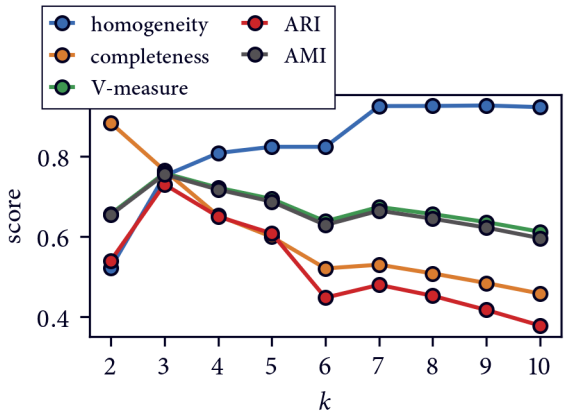


Figure 14.11 Choosing the *k* in *k*-means (external scores) While the homogeneity and completeness score are not very telling on their own, a combination of the two in the V-measure strongly favours the clustering with $k = 3$, which is confirmed by the adjusted Rand index (ARI) and the adjusted mutual information (AMI).

Here, the homogeneity score assesses to what extent each cluster contains only members of a single class. The score is bounded by 0 and 1 where 1 is best. Note, that this is very similar to homogeneity in terms of low cluster variance, which is optimised in *k*-means under the belief that low-variance clusters represent distinct (that is homogeneous) classes. Indeed we see the homogeneity score increasing with larger values for k. Complementary, the completeness score measures to what extent all members of a given class are assigned to the same cluster. Like the homogeneity score, completeness is bounded by 0 and 1 where 1 is best, as well. We see this score decreasing with larger values for k as more and more homogeneous groups appear, which, however, reproduce the same class. It feels natural to balance these two scores against each other, which is formalised in the V-measure as the harmonic mean of homogeneity and completeness.[429] For this score, we see an optimal (maximal) value for the *k*-means clustering with $k = 3$. Besides that, we can use the Rand index to define a similarity measure between two clusterings (the obtained one and the underlying truth). This index considers all pairs of objects in a data set and counts how many pairs are correctly or incorrectly assigned to the same or to different clusters based on the provided expected assignments. The adjusted version of the index (ARI) contains a correction for matching assignments by chance.[430] The *k*-means clustering with $k = 3$ is most similar to the expert classification. Finally, we can use the mutual information between the obtained and the true cluster label assignments to assess how much information about the underlying classes is captured by a given clustering, or more generally speaking how much two clusterings agree. The adjusted mutual information (AMI) does again account for expected agreements by chance.[431] We can see, that it is fairly straightforward to quantitatively compare clustering results with an assumption of ground truth. The problem with this is just that reference assignments are almost never available for practically interesting data sets and it would be very brave to speculate that a *k*-means clustering with $k = 3$ is universally a good choice even for data sets that are most similar to the considered case.

Figure 14.12 shows the result of a *k*-means clustering with $k = 3$ on which we could have settled using the presented validation schemes or because we knew how many clusters there should be in the first place. The agreement with the expectation is quite good, as the expected clusters can be well approximated by compact, globular, well-separated clusters of roughly equal spatial extent—which is the ideal view of clusters from the *k*-means perspective. The *k*-means approach is not suitable for data sets where there is reason to suspect that clusters can differ from this idealisation, at least if the aim is to generate a clustering where each cluster represents a separate interpretable class of objects. The approach is still valid and widely-used if there is a clear focus on homogeneity, meaning when it is accepted that true object groups can be split into multiple *k*-means clusters. A good example for this, are MD data sets. For these it can generally not be assumed that clusters of molecular conformations are globular, and low within-cluster variance may not be actually desirable. Still, *k*-means can be

used with a potentially very large number of clusters with the intention of data discretisation or condensation. As such it can for example be used to prepare a state-space for a MSM analysis, or as a preliminary clustering step to reduce the size of a data set from n points to k points so that further (clustering) steps can just operate on the obtained cluster centres.

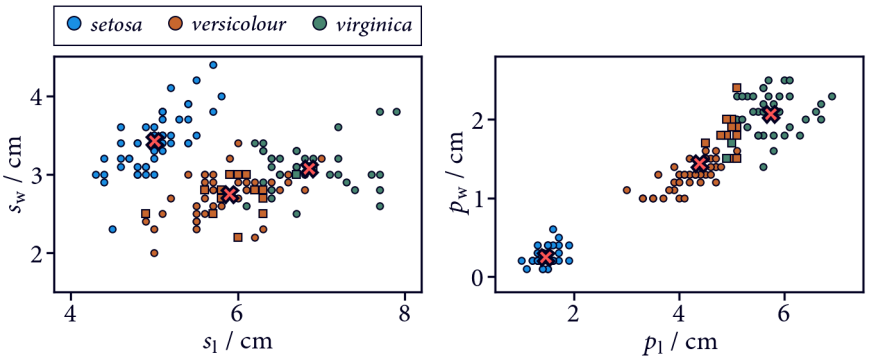


Figure 14.12 ***Iris* data set *k*-means (3 clusters)**
Data points (compare figure 13.3) with cluster labels found by k-means clustering (`sklearn.cluster.KMeans`) with $k = 3$. 89 % of the cluster labels match the true classification labels. Non-matching assignments are marked with squares. Cluster centres are drawn with crosses.

14.4 Gaussian mixture models

Clustering from a statistical point of view is nicely represented by GMMs or distribution-based mixture models in general.[432] If one has reason to believe that the observed samples in a data set are the manifestation of several overlapping well behaved probability distributions, say normal distributions, then it would be only logical to try to reproduce the observed data by fitting a certain number of said distributions to the data. It turns out that for the *Iris* data set, this is actually an excellent approach. The measured leaf proportions of plants belonging to the three different classes are indeed likely to be more or less normally distributed, which means that a good approximate model for the data can be achieved by a combination of three independent Gaussian distributions (one for each plant class). Figure 14.13 illustrates the result of this.

model distributions

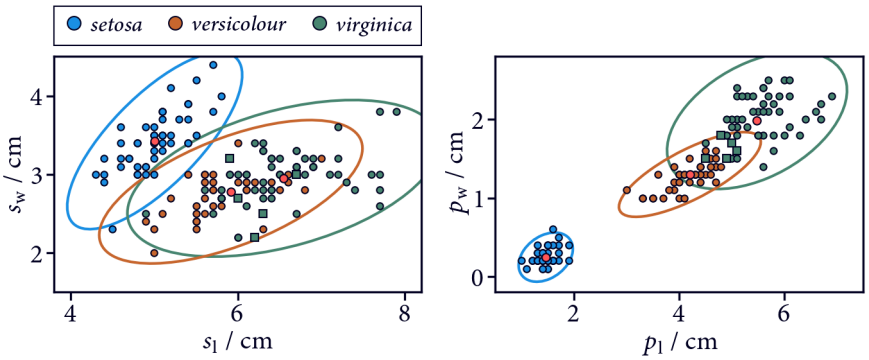


Figure 14.13 ***Iris* data set GMM (3 clusters)**
Data points (compare figure 13.3) with cluster labels predicted from a GMM (using `sklearn.mixtures.GaussianMixture`). The means of the used distributions are marked by red dots while the circumference of the ellipses corresponds to three standard deviations. 97 % of the cluster labels match the true classification labels. Non-matching assignments are marked with squares.

prototypes

In principle, mixture models are a form of prototype clustering where each data point is associated with the prototypical distribution it was probably sampled from—or with all of the underlying distributions with respective probabilities. What is achieved, is the decomposition of a full population of samples into k homogeneous sub-populations. There is no directly used similarity concept apart from that points identified with the same sub-population can be considered similar and that a sample probability kind of measures a similarity to a specific prototype.

maximum likelihood

Conceptually, the problem of finding the distributions that model the observed data best can be formulated in terms of a maximum likelihood approach. We want to find the distributions that are most likely to have produced the data set. Practically, a locally optimal solution to this can be found with the expectation maximisation algorithm.[433] Under the condition that we know beforehand how many distributions the data should be modelled with, one can start with an initial guess for the parameters (mean and variance) of each distribution. This initial guess can for example be generated with k-means. Then the probabilities for data points being sampled from each distribution are calculated and weighted so that the total probability of each sample over all distributions sums up to 1. The parameters are iteratively varied to improve them until convergence.

A clear weaknesses of mixture models is that while it can be a very good fit for data that truly originates from distributions that are similar to the chosen ones, it will more or less fail if one picked an inappropriate prototypical distribution. In other words, without decent knowledge about the clustered data, it may be hard to make a well founded decision about this. The same goes for the number of chosen distributions, although similar techniques as for k-means exist on how to tweak the number of underlying distributions, e.g. via the Bayesian information criterion (BIC-score).[434]

14.5 Density-based clustering using histograms

So far we have discussed classic examples for connectivity-based and prototype-based clustering methods that may or may not employ an explicit concept of similarity. Let's now proceed to *density-based* clustering, meaning methods that primarily identify clusters as groups of densely packed objects, rather than certainly behaved object groups. This type of clustering philosophy is most valuable for the clustering of molecular conformations and an overview over commonly used techniques should level the field to fully understand the CommonNN clustering approach (see section 14.9 and later chapter 15).

grid-based clustering

A clustering method that is on first glance almost trivially simple but yet very illustrative and with modifications also very powerful, makes use of (regular) spatial grids. We want to have a look at this method, to demonstrate some of the basic ideas and variations of density-based clustering and to build the bridge to connectivity-based and prototype-based cluster models as well as to flat partitional versus hierarchical clustering. It turns out that density-based as a trait alone can not provide a model that is sufficient to cluster data. It always has to be eventually combined with a notion of connectivity or prototyping.

Figure 14.14a shows a data set of points scattered in two dimensions. A density-based view on the data suggests the existence of two intertwined, sickle-shaped clusters as cohesive regions of high data point density, separated by low (zero) density. Formally, we expect the sampled data to be generated by an unknown probability density function $\rho : X \to R_{\geq 0}$. A density-based model for the data depends on an estimate of the true probability density based on the scattered samples. One way to achieve such an estimate is to impose a grid of cells with constant volume upon the data (figure 14.14b) and to count the number of points that fall into each cell, which means nothing else than to construct a histogram on the data (figure 14.14c).

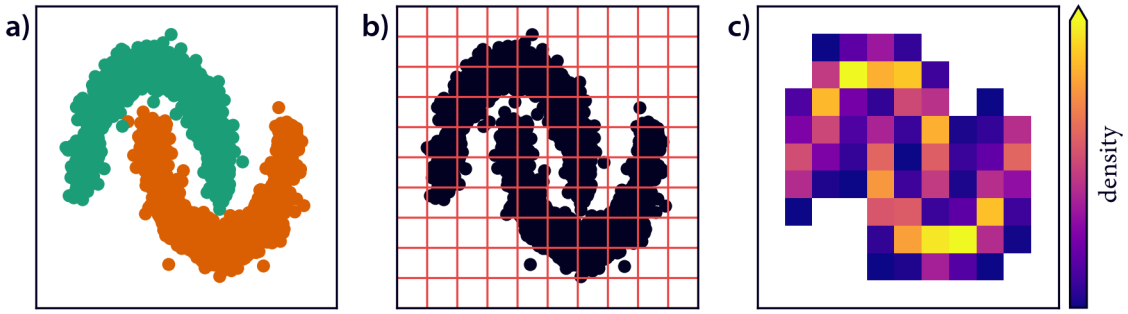


Figure 14.14 Density-based clustering using histograms
a) The scikit-learn *moons* data set (2000 points in 2D) with true cluster labels indicated by green and orange color. **b)** A regular grid with 10 bins in each dimension imposed upon the data. **c)** Cell-wise point density estimate (histogram).

The process of choosing a grid of cells that bins the data points can itself already be interpreted as a simple form of clustering in the sense of a discretisation. By setting the width of each cell to a constant ϵ and by limiting the extend of the grid in each dimension with $x_{i,\text{low}}$ and $x_{i,\text{high}}$, one obtains a number of $n_{\text{bins},i} = \text{ceil}\left((x_{i,\text{high}} - x_{i,\text{low}})/\epsilon\right)$ cells in each dimension or $\prod_i n_{\text{bins},i}$ cells in total. Alternatively, the number of bins per dimension can be set, which determines the bin width ϵ. A cell is uniquely addressed via its location on the grid by an indexing tuple, e.g. $i_{\text{cell}} = (i_1, i_2)$ in the shown 2D-case, or an indexing function $i_{\text{cell}} = i_1 n_{\text{bins},1} + i_2$, which is equivalent to a cluster label for all data points in the respective cell. The cells themselves or for example the midpoints of the cells constitute a form of prototype that individual points are identified with. There are many alternative approaches to impose a grid upon objects in a data set. Another rather sophisticated approach to tile objects recursively into bounding boxes is for example the construction of an R-tree.[435] It should be noted, though, that for grid cells of unequal volume, density can not just be estimated as the number of points per cell but has to be normalised by the cells volume.

regular grid

The discretisation of the data points into grid cells can in a way be seen as a pre-clustering of the data. We use this clustering as a support that provides a way to approximate the underlying probability density ρ. Note that the grid dimension controls the resolution at which the density is estimated. Having a density estimate for portions of the data space based on our notion of density as shown in figure 14.14c is, however, only half of what is necessary to derive density-based clusters. We also need a model for how our final clusters should be constructed, continuing on the density information. This could be done conveniently in this case by establishing a connectivity-based model.

connectivity

We could for example say that adjacent cells on the grid should be assigned to the same cluster if they both represent regions of relatively high density. 'Relatively' high implies that we could define a *density threshold* above which density is considered high. Two adjacent grid cells within which the threshold is exceeded will be regarded as connected.

threshold

An intuitive way to illustrate this, is to think of the (pre-clustered) data in terms of a graph structure in which each grid cell is a vertex and unweighted edges are drawn for pairs of adjacent cells in case their density is higher than the threshold (figure 14.15a). In this picture, clusters become immediately discernable as *connected components* of the respective graph, which is displayed in figure 14.15b for an example threshold. The cluster labels obtained for the histogram grid cells can then be transferred back to the original points. Figure 14.15c shows the final result of this, which is in agreement with the initially expected clustering (compare figure 14.14a). A particularity of the clustering is that grid cells below the density threshold that are not connected to any other cell can be labelled as noise, i.e. as not part of any cluster.

data graph

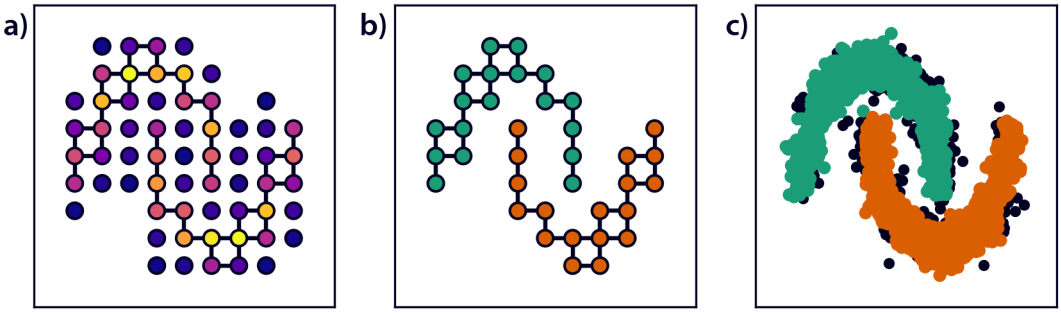


Figure 14.15 Density-based clustering using histograms and a density threshold
A density threshold is applied to identify connected grid cells. **a)** Histogrammed data represented as unweighted graph with edges connecting adjacent grid cell when both exceed the density threshold (e.g. here 15 points per cell). **b)** Clusters identified as connected components of grid cells. Grid cells with low density are not assigned to any cluster and are omitted. **c)** Cluster labels of the grid cells translated back to the original data points. Data points not assigned to any cluster (noise) are coloured in black.

How to describe this clustering?

Let's pause for a second and recapitulate how we could characterise the presented clustering approach. To begin with, we are using a type of grid to join individual data objects into groups. Based on this, we could say that our method is grid-based and in fact two typical methods doing something very similar, STING[436] and CLIQUE[437], are commonly referred to as grid-based methods. Note, however, that this naming is chosen to emphasise that the respective methods are supposed to be eligible for the clustering of large, high-dimensional data sets. The employed grids are a neat trick to condense the contained information, so that the clustering can be carried out on a pre-clustered data set, which is easier to handle. The finally produced cluster model on the other hand is not actually grid-based. The employed grid is exchangeable without altering the method in its essence, so grid-based is an implementation focused categorisation.

In contrast, it is vital to the method that the grid cells are used to estimate object density, so we can say that the method entails a density-based model. Further, the density-based model is characterised as connectivity-based because clusters of high density are identified as networks of connected (adjacent) dense cells.

Finally, we have a threshold-based criterion. Threshold-based is a categorisation focused on the inductive principle. The used threshold on the estimated density suggests what the optimal model for the considered data would be: a partitioning of the data objects according to their pairwise connectivity, where the threshold determines which connections exist and which are neglected. Each object can be reached directly or indirectly from any other point in the same cluster just by following the these connections. This objective is equivalent to a search for maximally connected components in the data graph from which low density connections where trimmed off. As was pointed out in chapter 7, finding connected components in graphs is a routine task that can be solved efficiently by well established graph traversal algorithms.

related methods

A similar threshold-based approach is used by the popular DBSCAN method (see section 14.7) in its original formulation and can also be applied to CommonNN clustering (see section 14.9). These methods differ from the presented grid-based approach primarily in the way how density is estimated and how dense objects are connected with each other. In the following section 14.6, the unifying concept of level-sets that underlies all of these methods will be discussed briefly.

similarity

The connectivity between object vertices in the implied graph structure for the data can be interpreted as a form of similarity. Two connected points are similar with respect to how the connection was defined in the connectivity-based model. When the density threshold is applied to select valid

connections, similarity can be viewed as a binary relation, that is two objects can either be similar (because they pass the threshold criterion) or not. In a relaxed view, it would also be possible to state that all points within the same cluster are similar to each other if one allows the following reasoning: if two points a and b are connected (similar) and b is connected to a third point c, then a is also (indirectly) connected to c. Note, however, that this form of similarity is detached from similarity measures with respect to the data space the clustered data points where originally embedded in. This means that two points in the same cluster can be actually very dissimilar (far away from each other), and vice versa two points in different clusters can be actually very similar (close to each other), from a different perspective on similarity.

latent space

Interestingly, the actual data space plays only a minor role for the presented grid-based approach. The decisive element for the clustering is the density estimate that does only require the grid, which in turn is in general any kind of discretisation—of not necessarily known origin. There is no direct dependence on another concept of similarity, neglecting that the needed discretisation of the space may involve one. In this sense, the clustering can be said to operate on a latent data space.

choice of threshold

For threshold-based approaches to work with a density estimate on a data set, the clustering result is obviously dependent on the choice of the threshold. For lower thresholds, the two disjoint components of the graph as shown in figure 14.15 may for instance become connected. In this case, only one cluster will be found. For even higher thresholds, the two components are expected to shrink since lower density grid cells on their outer rims will fall below the minimum density requirement. Where to put the threshold, can be understood as a tuning parameter. It is left to the user of the clustering to select an ideal value in the context of a specific application and possible expectations. There is no universal concept of what a suitable threshold would be.

cluster hierarchy

Effectively, a variation of the threshold criterion creates a hierarchy of clusterings of systematically varying granularity. From this perspective, the applied threshold acts as termination criterion on an intrinsically hierarchical cluster model. It provides a strict inductive principle that prefers a model at a specific level of the hierarchy and turns the method into a quasi-flat clustering. Yet, it is also possible to not apply a threshold on the estimated density and to obtain the actual hierarchy of clusterings instead. Let's recall that the use of a density threshold gave us a binary perspective on pairwise object similarity (connectivity) where the threshold was exceeded. To relinquish the threshold means to find a quantitative description of connections between grid cells as weighted edges in the respective data graph. In this picture, objects can be more or less similar.

A simple variant of this could be to use the minimum density of two connected cells as a weight for their connection. In this way, a pair of cells in which one of the cells represents a low density data region will consequently have a low edge weight on the connection between the pair, rendering it of low importance (note that it is in principle arbitrary whether high density equates to low or high edge weight). A hierarchy of clustering results is than basically given by reviewing all connections in the graph in the order of their importance, for example by beginning with the least important one and dropping one connection at a time. As more and more connections are ignored each level in the hierarchy represents a new clustering. Starting with all objects in the data set in one cluster when still all connections are considered, the hierarchy is ending with each data object in a separate cluster (i.e. as noise) once the last connection has been dropped. In reverse, it is also possible to begin with adding the most important connection and to proceed with lower weight edges.

Figure 14.16 shows the data graph with weighted edges and the resulting hierarchy of clustering results for our example case. From this, it is now clear at which density threshold a splitting of the data set into sub-clusters can be observed exactly. Note that here clusters need to contain more than two cells to be counted as such and are considered noise otherwise. For this demonstration, a connected component search was performed on a graph at each hierarchy level after adding all edges of a certain

weight. It should be mentioned that this is in general very inefficient and that other approaches can be used, for example by leveraging MSTs (see section 7.2 for theory). Using a MST of density weighted edges transforms the clustering into nothing else than single-linkage clustering.

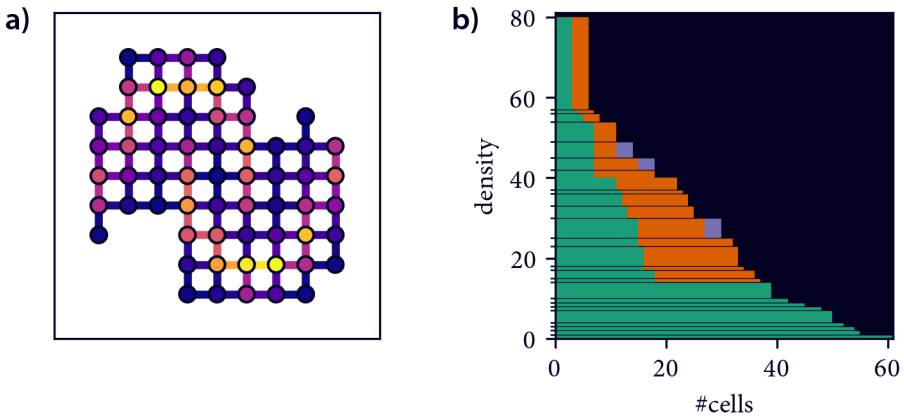


Figure 14.16 Density-based clustering using histograms hierarchically
Not applying a threshold, the full hierarchy of clusterings can be built instead. **a)** Data set represented as weighted graph with edges connecting adjacent grid cells quantitatively. **b)** Hierarchy of clusterings. Each edge weight on the y-axis gives rise to a new hierarchy level when edges of the corresponding weight are removed from the graph (or added). Starting at the bottom with all grid cells in one cluster (all connections are present), the graph becomes more and more disconnected at higher levels until all grid cells are separated at the top. The absolute number of cells assigned to each cluster at each level is represented with coloured horizontal bars where black stands in for noise.

14.6 Density-based clustering using level-sets

Connectivity-based density-based clustering like introduced in the last section can be uniformly expressed with the help of level-sets. A level-set of a function $f(x)$ is in principle just the set of points $L = \{x \,|\, f(x) = \lambda\}$ for which the function takes on a certain value λ. The use of level-sets for the definition of density-based clusters is usually attributed to Hartigan.[438] Let $\rho(x)$ be an underlying probability density on a continuous data space X (practically usually a subset of $\mathbb{R}^d$) from wich a data set $\mathcal{D} = \{x_1, ..., x_n \,|\, x_i \in X\}$ can be sampled. Then, a level-set of ρ using a density criterion λ corresponds to a density iso-surface (in 2D a contour-line) on this density function. A super level-set (also often called an upper level-set or just a level-set) is furthermore defined as $L = \{x \in X \,|\, \rho(x) \geq \lambda\}$, i.e. the set of points for which the density exceeds the specified λ-threshold. Density-based clusters are the connected components of such a super level-set. The sub level-set, the domain of the function for which the density falls below the threshold, is ignored for the cluster assignment and treated as a 'noise', outlier, or 'fluff' region.[439] A clustering based on level-sets does not yield a full partitioning of a data set.

level-sets

While a clustering can be achieved with a respective λ-value, the level-set formulation makes it obvious that there is actually a hierarchy of connected components corresponding to a continuous variation of λ in the interval $[0, \max(\rho(x))]$. It can be shown that this leads to a finite level-set tree of clustering results.[440] A property of this tree is for example that for any connected component C_{child} obtained at a density value λ, there is exactly one connected component C_{parent} with $C_{\text{child}} \subset C_{\text{parent}}$ for any $\lambda' < \lambda$, that is there is a strict child-parent relation between connected components at different levels. Furthermore, if of two connected components one is not a subset of the other, they do not overlap at all. There is a large of amount of level-set tree related theory, on how to construct and analyse them, and on how to estimate them from scattered data.[441, 442] Density-based clustering of

level-set trees

point samples is one possibility to estimate level-set trees for unknown probability density functions. Of particular interest might also be the connection between density-based clusters and single-linkage clustering (using MSTs).[439, 443, 444]

Figure 14.17 shows an example of a level-set tree on a 1-dimensional multimodal distribution for which $\rho(x)$ is known. In this case one can construct the tree of connected components quasi-analytically for different λ values just by finding the respective level-sets, i.e. the points where the threshold and ρ intersect. The plot represents the size of each component and its mean location. For an analysis aiming on the extraction of clusters, one is usually most interested in the λ-values at which the number of connected components increases.[440] In the example, this is the case for λ_1 and λ_2. Up to λ_1 there is only a single connected region under the curve of the function, which shrinks from the borders when λ is increased starting from 0. Above λ_1, the minimum between the rightmost peak and the rest of the function falls below the threshold and leaves the two respective regions as disjoint sets. Note that once λ_3 is reached, the maximum of the right component falls below the threshold and the component vanishes. One single threshold value is not able to select a partition of three components corresponding to the peaks of the distribution. The intuitively correct clustering result is a combination of the components from the orange, red, and green branch of the level-set tree.

1D example

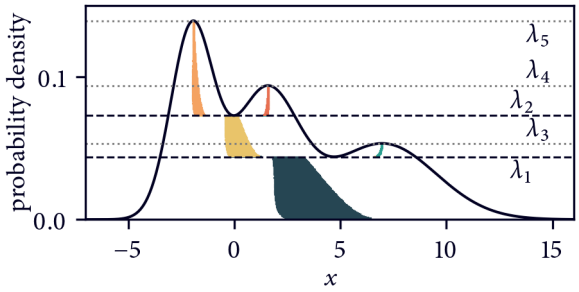


Figure 14.17 Level-set tree for a 1D multimodal Gaussian distribution Slicing the shown probability density distribution along a λ threshold value, we obtain corresponding clusters as the connected components of the respective super level-set. For each component, its size is represented by a coloured horizontal bar scaled in width by the integral $\int_{x_1}^{x_2} \rho(x)$, where x_1 and x_2 are the components bounds. The bars are centred around the mean of the component.

When density-based clusters should be discovered from discrete point samples in accordance with the level-set approach, one essentially needs a density estimate ρ' for the unknown underlying probability density ρ. The connected components of the discrete level-set $L' = \{x \in \mathcal{D} \,|\, \rho'(x) \geq \lambda\}$, that is the identified clusters, are an approximation to the corresponding components of the true density. Naturally, the validity of a clustering does therefore depend on a consistent density estimate and beyond that on a robust definition of connectivity between data points in L'.

density estimate

A generic realisation of connectivity that does in principle work with arbitrary density estimates is, to give another example, found in the following approach. Consider a Voronoi partitioning of the n points in a data set $\mathcal{D}$ into n cells, i.e. a spatial tessellation in which any other point not in $\mathcal{D}$ would fall into the same cell as its respectively closest neighbour in $\mathcal{D}$. Then take the Delaunay triangulation by connecting each data point to other data points that are in adjacent cells. Given a point-wise density estimate, for example the reciprocal k-nearest distance (with say $k = 1$), one can drop those points from the just created network that fall below a specified density threshold and reveal the clusters as the remaining connected components.[445] Note that this is very similar to the previously discussed grid-based example only that the grid served there as the density estimate as well while it does here primarily establish connectivity. A slightly different realisation using a k-nearest neighbours graph from which low density nodes are removed at each iteration of λ is available with DeBaCl clustering.[446] The following sections will discuss connectivity-based density-based clustering procedures that solve the problem of establishing a density estimate and connectivity differently.

Delaunay connectivity

14.7 DBSCAN

A VERY POPULAR DENSITY-BASED CLUSTERING method is DBSCAN,[447] which stands for *density-based spatial clustering for applications with noise*. Occasionally, DBSCAN and density-based clustering are even used as synonyms in the literature. As mentioned in section 14.5, the method is conceptually very similar to the previously described density-based clustering using grid cells. Virtually, the only difference lies in how density is estimated based on the samples in a data set. Instead of the number of objects per cell, DBSCAN takes the number of neighbouring points as a density estimate for individual points.

Typically, the neighbourhood of a point is the open- or closed-ball neighbourhood $\mathcal{B}_r$ as defined in and 13.7, using a distance metric (usually the Euclidean distance) and a fixed radius r. The cardinality of the set $\mathcal{B}_r$ provides the density estimate for a single point. Frankly speaking, the volume element corresponding to a neighbourhood can also be viewed as a special type of cell, making the relation to grid-based clustering quite obvious—only that the cells are centred around individual points and partially overlapping.

Just as in grid-based clustering, the density-estimate alone does not suffice to group data points into clusters. We still need a connectivity concept to decide when two points should be part of the same cluster. In the original formulation of DBSCAN,[447] this is defined leveraging a threshold. Points that possess a density estimate exceeding the threshold, i.e. that have at least a number of n_c neighbours with respect to a neighbour search radius r, are called *core points*. Core points that are neighbours of each other are in turn considered to be connected. Fundamentally, the set of all present connections constitutes a graph for the data and clusters are the maximally connected components of this graph. This notion is in line with the formulation of the clustering problem in terms of level-sets (compare section 14.6). Additionally, points that are not themselves core points but neighbours of a core point are termed *border points*. Connections of border points to core points can optionally be added to the graph as well, but note that this introduces some ambiguity because border points may be connected to core points in different clusters. In this case, it has to be decided to which cluster a point should be assigned, which can be done randomly or for example by choosing the closest core point or by evaluating the membership of all neighbouring core points against each other.

Figure 14.18 illustrates the DBSCAN density criterion with a threshold and a clustering result for the previously used scikit-learn *moons* data set.

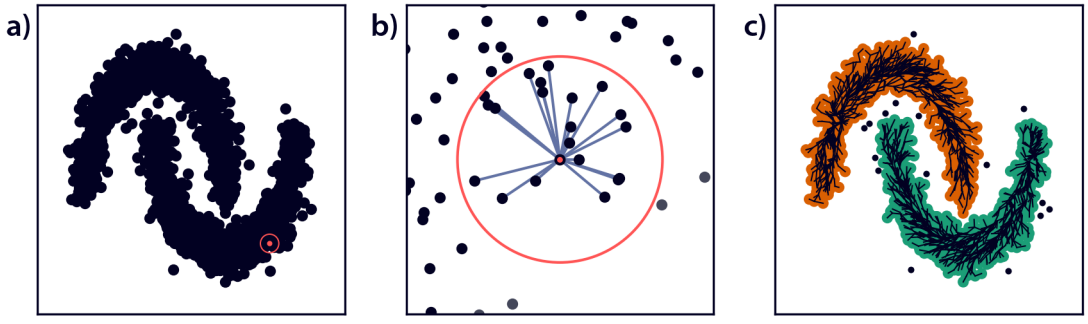


Figure 14.18 DBSCAN for the *moons* data set
a) Original data points with neighbour search radius $r = 0.15$ shown for a single point with a red circle. **b)** For a density threshold of $n_c = 10$, the point highlighted in **a)** is a core-point since it has 20 neighbours. All its neighbours are connected to it as indicated by blue lines. Border points are drawn in gray. **c)** The connections between the data points form a graph in which clusters correspond to maximally connected components. The data points are coloured by their cluster labels (noise in black). The edges of a graph build from a connected component search (see code snippet further below) are shown with black lines.

The DBSCAN publication coins the terms *density-reachable* and *density-connected*. A point b is directly density-reachable from a point a if a is a core point and b is a neighbour of a. Furthermore, a point c is indirectly density-reachable from a if there is a chain of directly density-reachable points from a to c, say there is a point b that is reachable from a while c is reachable from b. Finally, two points are density-connected if there is a point from which both are (indirectly) reachable. Density-connectivity is a symmetric definition while density-reachability is only symmetric for core points. Clusters are identified as sets of density-connected points. The paper does not identify this with the concept of connected components of a graph in which edges correspond to the direct reachability of two data points but the notion is exactly equivalent.

density-reachability and connectivity

Practically, DBSCAN can be implemented using a connected component search algorithm like breadth-first-search (BFS) (see also section 7.1). Starting with any point of the data set as the root of the first cluster, the full cluster can be explored iteratively by adding all points that are connected to this first point, then adding all points connected to the newly added points and so forth. Once no other point can be added, the next cluster can be explored by choosing a new root if there are still unassigned points left in the data set. During such a search, it is possible to either build a graph of connections explicitly or to just directly assign cluster labels to data points. The short code snippet below gives an example for how to do both at the same time. The example assumes that core points have been previously identified by checking the neighbour count against a threshold. As usual, the algorithm depends on a FIFO queueing structure to collect points from which the cluster can be grown further and an indexable indicator structure to keep track of which points have been already visited. The graph in figure 14.18c has been generated in this way. Note that this graph contains only a (minimum) set of required connections that depends on how the graph is explored. Note also that the code example will add border points to the first possible core point if the used neighbour lists contain not only core points.

implementation

```
clabel = 1
for p in core_points:
  if visited[p]:
    continue
  visited[p] = True
  labels[p] = clabel
  queue.push(p)

  while queue:
    p = queue.pop()
    for q in neighbours[p]:
      if visited[q]:
        continue
      visited[q] = True
      labels[q] = clabel
      graph.add_edge(p, q)
      queue.push(q)

  clabel += 1
```

As a side note, one of the earliest density-based clustering method, which was proposed by Wishart in 1969,[448] does something very close to DBSCAN with a different notion of how to establish connectivity. The idea is to identify core points using the same density estimate, but than using classic single-linkage clustering on these core points to find the clusters.

Wishart variant

Like for the grid-based approach, the choice of the threshold value tunes the outcome of the clustering in DBSCAN. A complete screen of the threshold results in a hierarchy where higher thresholds create splits in denser data regions while larger portions of the data fall below the threshold and are declared noise. Choosing a suitable threshold can, however, be unintuitive and cumbersome,

hierarchy

not to mention that a systematic test of many different thresholds can be fairly expensive for larger data sets. There exists a number of possible approaches to use the DBSCAN density estimate without a threshold instead, though.

The equivalent to what was described for the grid-based example, would be to define a weight for connections between two data points based on their density estimate. This could be the minimum density of the two points. An evaluation of all connections ordered by their weight will then lead to the complete hierarchy of clusterings where each hierarchy level corresponds to a specific density threshold (compare 14.16). In this case, one needs to additionally define when two points should be connected to each other in the first place. For the grid example, connections where considered only between adjacent cells. Correspondingly, connections can be considered only between neighbouring points here. Note, however, that this in turn again depends on the neighbour search radius r, which kind of acts as a resolution parameter comparable to the bin size in the grid case.

HDBSCAN

Another variation that turns the DBSCAN concept upside down was proposed with HDBSCAN.[449, 450] The idea is to transform the point-wise density estimate into a new metric called *mutual reachability distance* that can be used as a connection weight between data points. In conventional DBSCAN, the question is *'how many neighbours does a point have?'* or rather *'does a point have at least n_c neighbours?'*. In HDBSCAN, this is turned into *'how large does r need to be so that a point has at least n_c neighbours?'* The neighbour search radius r at which a point fulfils this density criterion is called the point's core distance d_{core}—the radius for which a point becomes a core point—and is equal to the k-nearest distance for $k = n_c$, i.e. the distance to its n_cth closest neighbour. The mutual reachability distance between two points a and b is then defined as

$$d_{\text{mutual}}(a, b) = \max\left(d_{\text{core}}(a), d_{\text{core}}(b), d(a, b)\right) , \tag{14.8}$$

where $d(a, b)$ is a regular (the Euclidean) distance between the points. Effectively, points that are in relatively sparse data regions and have large core distances are pushed further away from other points. Dense points with low core distances remain at their original distance to other dense points. DBSCAN can be reformulated as single-linkage clustering on this new metric. In practice, however, the set of all pairwise connections between the data points can be reduced to a minimal set of relevant connections, namely a MST. Bottom-up iteration over the edges of this MST starting with the two most closely connected points creates the hierarchy of clusters. A certain slice of the hierarchy is exactly what is achieved with conventional DBSCAN with a certain density threshold.

Unfortunately, single-linkage hierarchies can be complex for larger data sets. For n data points, the MST has $n - 1$ edges and hence the cluster hierarchy comprises $n - 1$ merges. The number of merges one eventually might be interested in, can be much smaller, though, because many of them typically correspond to a situation where two small clusters merge or where a small cluster is swallowed by a big one. To narrow the number of merges down to those where two big clusters are joined, one can define a minimum cluster size. Clusters with a member count lower than this minimum requirement can be regarded as noise. A merge of a noise cluster can be simply seen as cluster growing and can be ignored in the hierarchy of merges.

Figure 14.19 shows an example application of HDBSCAN to the *Iris* data set including the single-linkage hierarchy obtained directly from the MST of mutual reachability and a condensed hierarchy using a minimum size for relevant clusters.

analysing hierarchies

The full hierarchy of clustering results is more powerful than individual flat clusterings. For one thing, the hierarchy can guide the choice of again a simple threshold to eventually extract a certain slice of the hierarchical tree. On the other hand, the hierarchy can be processed to select a final clustering result as a combination of clusters from different branches of the tree, which means with possibly different thresholds for each cluster. This selection of child clusters can be just done rationally by the user but HDBSCAN does also provide a heuristic automatic approach for it. Starting with the

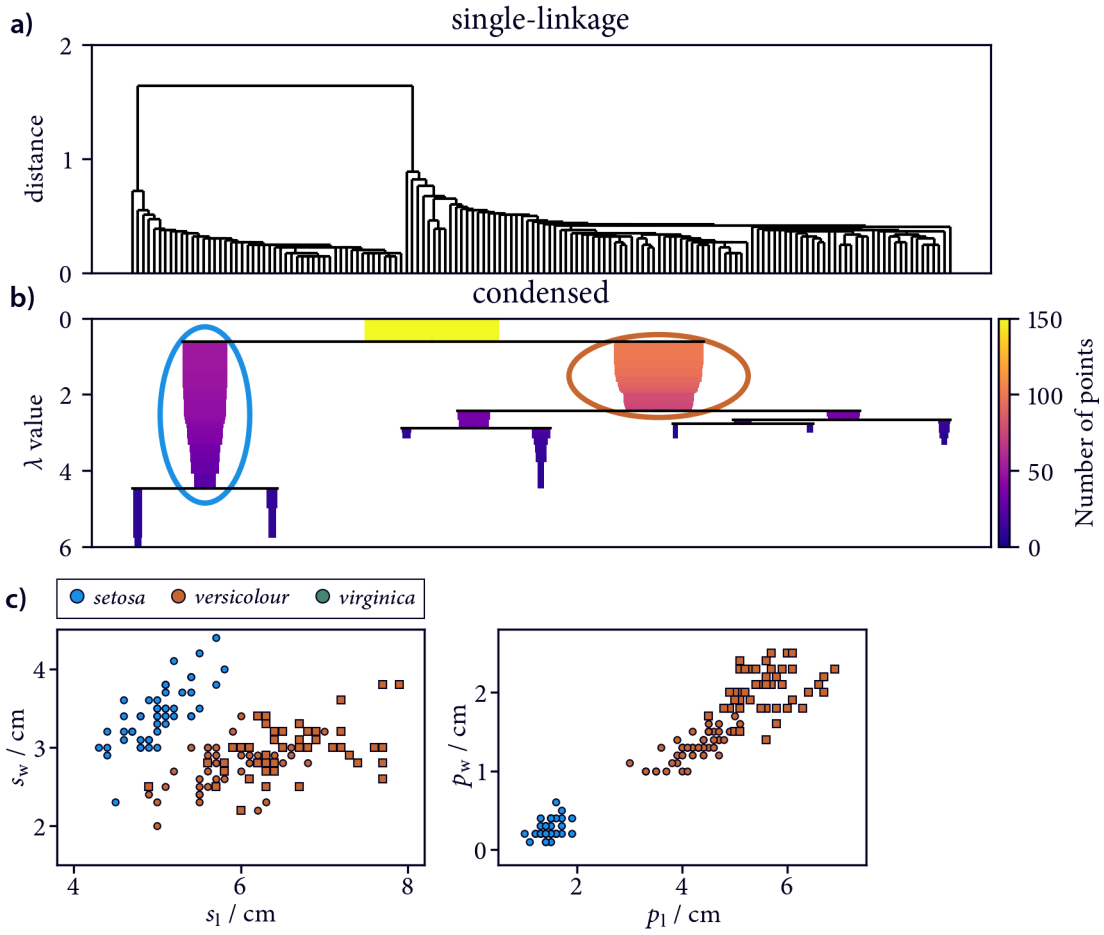


Figure 14.19 ***Iris* data HDBSCAN hierarchy and clustering result**
a) Dendrogram for a single-linkage clustering on a MST using the mutual reachability distance for $n_c = 2$ (see equation 14.8) and **b)** condensed tree applying a minimum cluster size value of 5. Note that while the single-linkage tree is labelled on the y-axis with the actual distance value at which a respective merge can be observed, the condensed tree is given in values of $\lambda = 1/d_{\text{mutual}}$, which is an efficient density estimate. Based on the relative persistence of the clusters in the condensed tree, HDBSCAN (`hdbscan.HDBSCAN`) suggests a preferred clustering result as a combination of clusters from different branches of the tree (highlighted with circles around them). **c)** Recommended clustering result based on cluster persistency. 67 % of the cluster labels match the true classification labels. Non-matching assignments are marked with squares.

outer leaf clusters, a persistence, or in other words a live time, is determined for each cluster in terms of the threshold range in which the cluster exists. If the live time of a parent cluster is longer than the summed live times of its children, the parent will be kept as the more relevant cluster. How to process cluster hierarchies programmatically will be discussed further in the context of CommonNN clustering in chapter 15.

Of course, it still depends on the application if such a processing of a hierarchy leads to a desired outcome. For the *Iris* data set, the HDBSCAN live time rational prefers two clusters, which is not well in agreement with the biological assignment (compare figure 14.19 and 13.3). Figure 14.20 shows a more agreeable clustering for which the threshold was selected based on the hierarchy as the smallest value where three clusters can be obtained. Finding this threshold without the hierarchy would necessitate a manual try-and-evaluate approach with different threshold values.

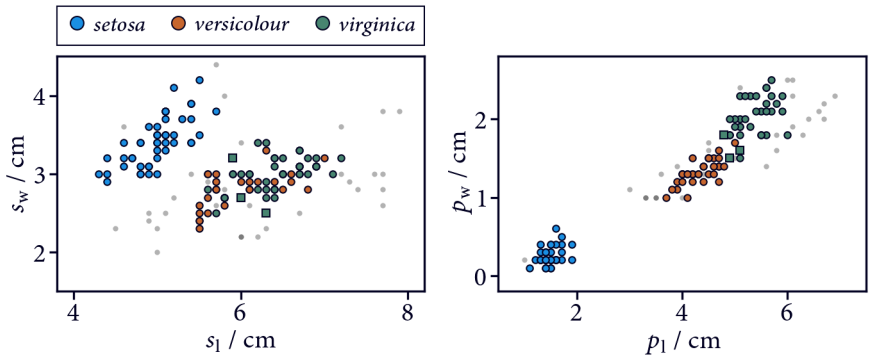


Figure 14.20 ***Iris* data DBSCAN with a threshold selected from the hierarchy**
Investigating the single-linkage hierarchy (compare figure 14.19), it is possible to select $d_{\text{mutual}} = 0.4$ as a threshold value at which the data set is partitioned into three clusters of similar size. The respective cluster labels can be extracted from the HDBSCAN hierarchy. Conventional DBSCAN (using `sklearn.cluster.DBSCAN`) with $r = d_{\text{mutual}}$ and $n_c = 2$ gives the same result. 81 % of the cluster labels match the true classification labels (98 % if noise points are neglected). Non-matching assignments are marked with squares.

14.8 Jarvis-Patrick clustering

In the previous sections, we discussed density-based clustering protocols that defined the notion of how density is estimated using either a certain kind of volume elements (grid cells) or the neighbourhoods of individual data points. For the actual identification of clusters, a separate introduction of a connectivity concept was necessary in these cases because the entities for which the density is estimated have no intrinsic density-related relationship to each other. Connectivity was derived by mixing another type of relation (the adjacency of grid cells or the neighbourhood relation of points) with the density estimate.

density estimate

The Jarvis-Patrick clustering methodology is relatively unpopular (at least in comparison to DBSCAN) but it does in contrast provide a density estimate that directly serves as a connectivity definition as well.[451] Density is taken as the number of neighbours that two points share with respect to their k-nearest neighbourhood. Since this density estimate involves two points, it establishes a connection between them. The definition is also sometimes referred to as the *shared nearest neighbour* (SNN) similarity.

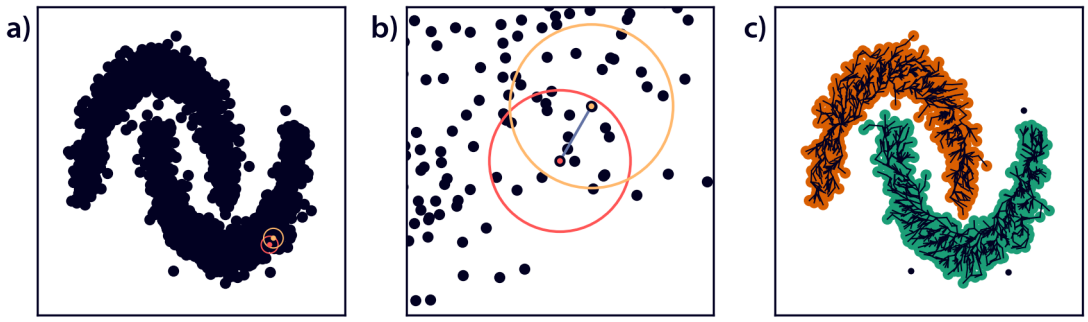


Figure 14.21 **Jarvis-Patrick for the *moons* data set**
a) Original data points with k-nearest radii ($k = 20$) shown for a data point (red) and its 15th closest neighbour (orange). **b)** For a density threshold of $n_c \leq 11$, the points highlighted in **a)** are connected since they share 11 neighbours. **c)** The connections between the data points form a graph in which clusters correspond to maximally connected components. For a clustering with $n_c = 10$, the data points are coloured by their cluster labels (noise in black). The edges of a respective graph build from a connected component search (see code snippet in section 14.7) are shown with black lines.

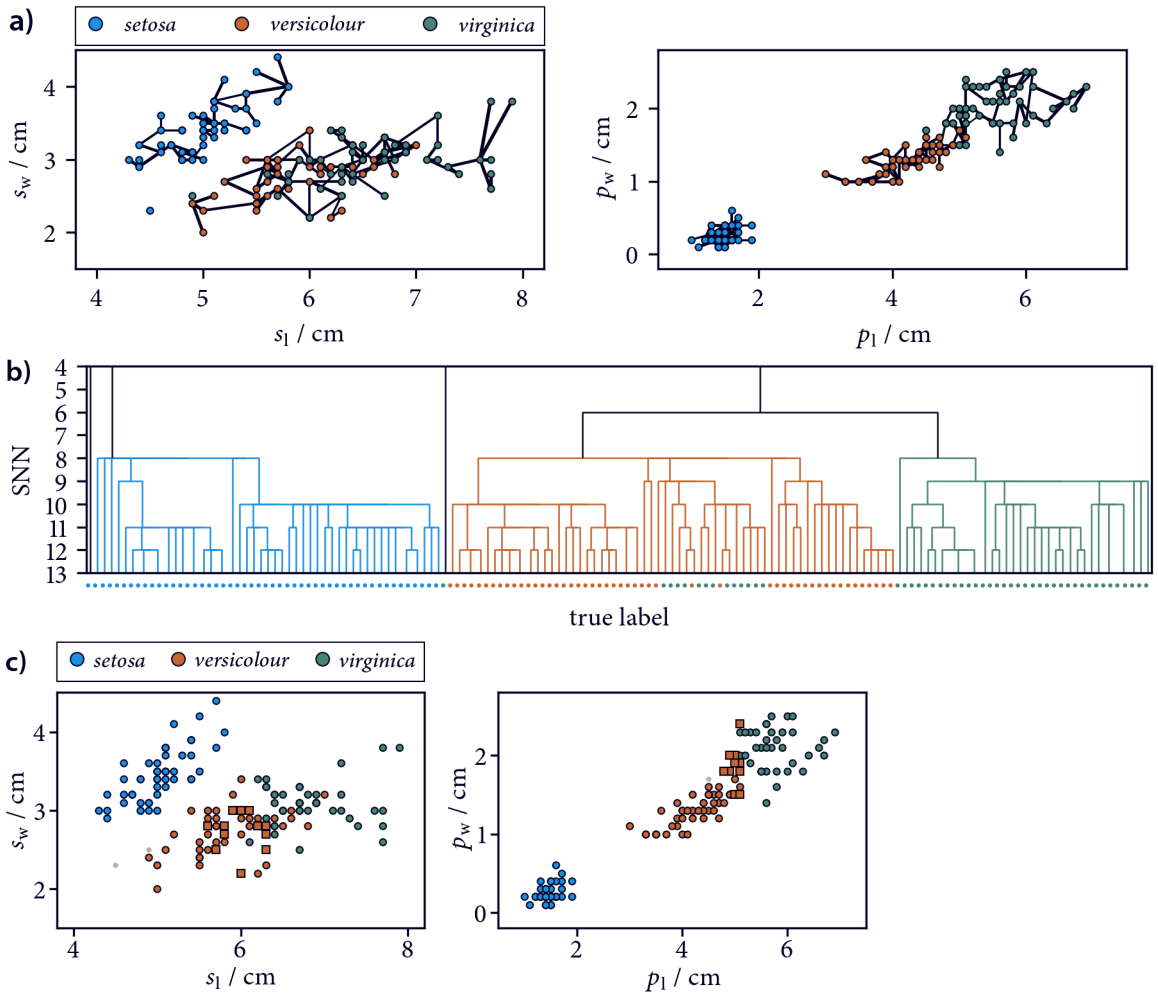


Figure 14.22 ***Iris* data Jarvis-Patrick hierarchically**

a) Original data points with true classification labels (compare figure 13.3) and MST using the Jarvis-Patrick connectivity criterion for $k = 15$. The edges of the tree are shown with black lines, scaled by their weight. **b)** Single-linkage hierarchy illustrated as a dendrogram. By investigating the hierarchy, it is possible to select three clusters, corresponding to a flat clustering with $n_c \in \{6, 7\}$. The legs of the tree are coloured by the resulting cluster labels while circles on the bottom denote the true classification of individual points. A threshold $n_c \leq 6$ would lead to a merge of the orange and green cluster. A threshold $n_c \geq 7$ creates several splits in the shown clusters. **c)** Flat clustering for $n_c = 7$ result as scatter plot. 90 % of the cluster labels match the true classification labels (91 % if noise points are neglected). Non-matching assignments are marked with squares.

The size of the k-nearest neighbourhoods to evaluate is kind of a resolution parameter comparable to the neighbour search radius r in DBSCAN. It should be noted, though, that the k-nearest neighbourhoods can not be associated with a fixed spatial volume. For data points in sparse environments, the distances to their kth neighbour are naturally much larger than for dense points. In principle, the density estimate can be done for all possible pairs of points, which essentially constructs a similarity matrix. However, a crucial additional requirement is usually made: points for which the similarity is evaluated must be a kth nearest neighbour of each other. The similarity is set to zero for all other pairs. Without this limitation, the outcome of the clustering is altered substantially. For the identification of clusters as connected components, a minimal set of local connections between points is sufficient. Consequently, what is practically dealt with in implementations of this clustering is not the full similarity matrix but rather a subset of necessary connections in terms of edges in a graph.

Similar to the already described density-based clustering schemes, Jarvis-Patrick clustering is traditionally used with a threshold to produce a flat clustering. Points that share at least n_c common neighbours with respect to their k-nearest neighbourhoods are identified to belong to the same cluster. Figure 14.21 illustrates the Jarvis-Patrick density-criterion on the scikit-learn *moons* data set.

What has been stated about hierarchical clustering in the previous sections, can also be translated almost one-to-one for Jarvis-Patrick clustering as well. The basic idea is to use the similarity between data points quantitatively instead of converting it to a binary relation using a threshold. Figure 14.22a shows a MST of edges corresponding to the Jarvis-Patrick density estimate for the *Iris* data set. By single-linkage clustering of this tree, the full hierarchy of clustering results with increasing threshold can be built and analysed as shown in figure 14.22b. Three clusters in close agreement with the expectation can be selected, which are shown in figure 14.22c. The details of this will be discussed further in the context of the very closely related CommonNN clustering in chapter 15.

It should be noted, however, that the MST of the data and consequently the hierarchy of clustering results still depends on the cluster parameter k, i.e. number of nearest neighbours to be considered in the point neighbourhoods.

14.9 Common-nearest-neighbour clustering

A variation of the Jarvis-Patrick clustering approach is found in the formulation of an independent method, referred to as common-nearest-neighbour (CommonNN) clustering. As an alternative to the use of k-nearest neighbourhoods and the SNN similarity in Jarvis-Patrick clustering, CommonNN clustering uses fixed radius neighbourhoods with an accordingly modified *shared fixed radius near neighbours* similarity (for which to my knowledge no widely accepted abbreviation exists). In other words, the similarity between a pair of data points in CommonNN clustering is defined as the number of points that can be found in both the fixed radius neighbourhoods of each point, i.e. as the number of their in this sense *common* neighbours. A differentiation between common near(est) and shared nearest neighbours in terms of a similarity definition just by the naming is admittedly a bit blurry, though. If the character of the neighbour lists is neglected, Jarvis-Patrick and CommonNN clustering are basically identical. From an objective standpoint it would make sense to treat the two clustering methods as different flavours of essentially the same method, let's say *shared neighbours*

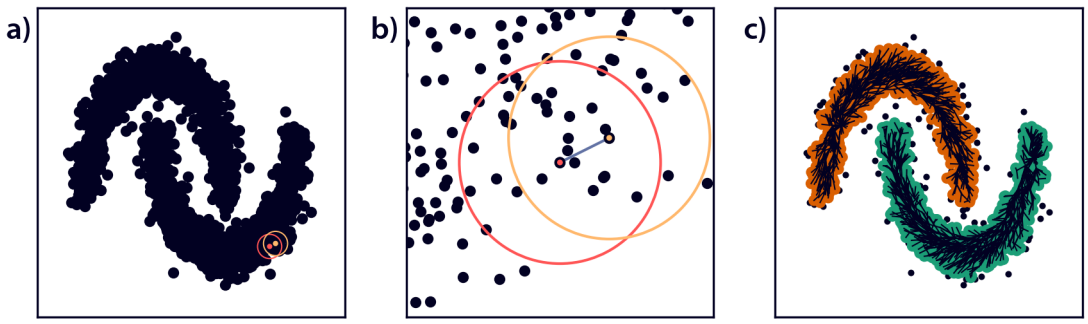


Figure 14.23 CommonNN for the *moons* data set
a) Original data points with fixed near neighbourhood radii (r = 0.2) shown for a data point (red) and one of its neighbours (orange). **b)** For a density threshold of $n_c \leq 18$, the points highlighted in **a)** are connected since they share 18 neighbours. **c)** The connections between the data points form a graph in which clusters correspond to maximally connected components. For a clustering with n_c = 10, the data points are coloured by their cluster labels (noise in black). The edges of a respective graph build from a connected component search (see code snippet in section 14.7) are shown with black lines.

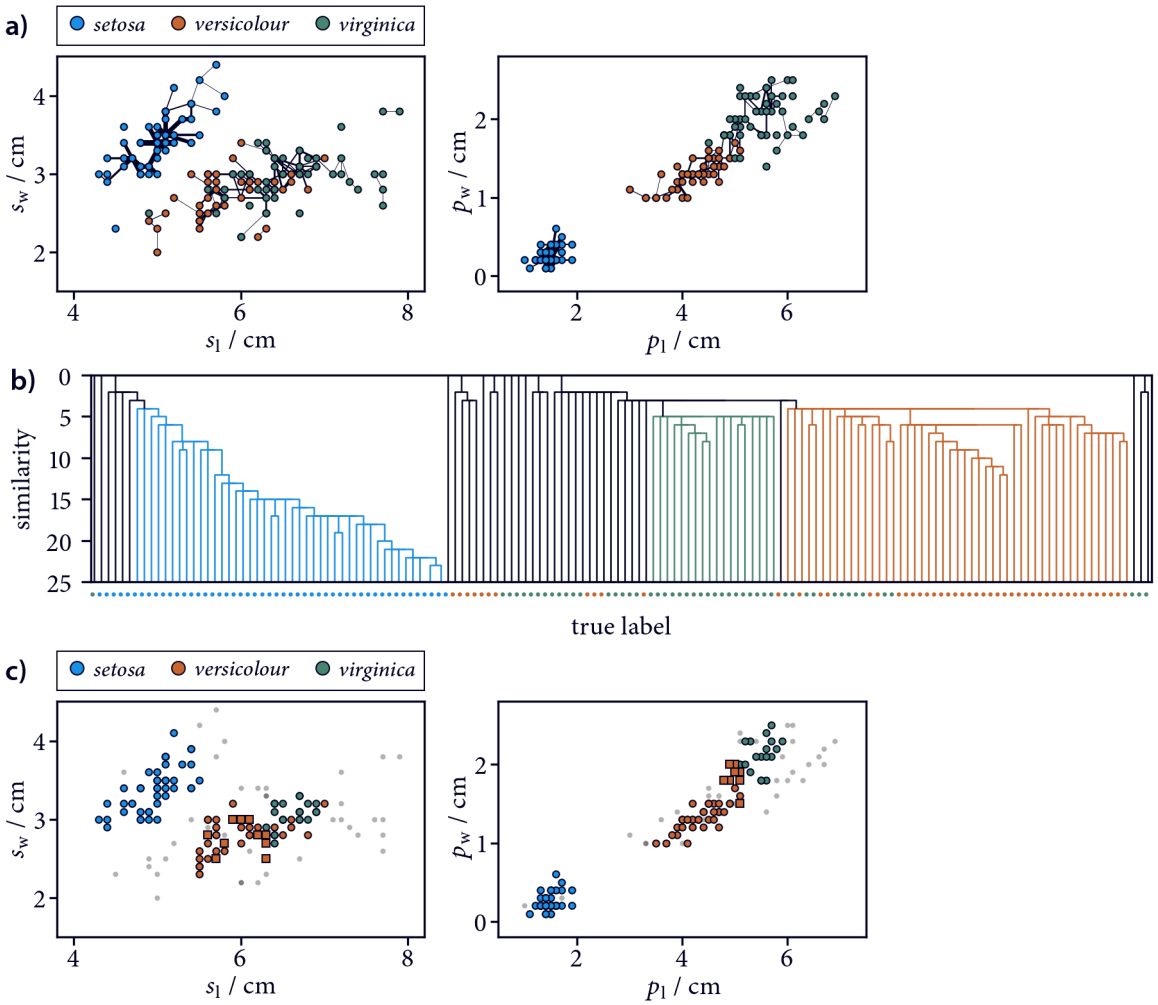


Figure 14.24 ***Iris* data CommonNN hierarchically**

a) Original data points with true classification labels (compare figure 13.3) and MST using the CommonNN connectivity criterion for $r = 0.45$. The edges of the tree are shown with black lines, scaled by their weight. **b)** Single-linkage hierarchy illustrated as a dendrogram. By investigating the hierarchy, it is possible to select three clusters, corresponding to a flat clustering with $n_c = \{4\}$ (note that this includes an offset of 2 due to neighbour self-counting). The legs of the tree are coloured by the resulting cluster labels while circles on the bottom denote the true classification of individual points. **c)** Flat clustering result for $n_c = 4$ as scatter plot. 67 % of the cluster labels match the true classification labels (90 % if noise points are neglected). Non-matching assignments are marked with squares.

shared neighbours clustering

clustering for that matter. Besides, shared neighbours clustering with fixed radius near neighbours or k-nearest neighbours, any other neighbourhood definition might be used as well—possibly, though, with drastically different outcome. Like for Jarvis-Patrick clustering, we will see for shared neighbours clustering in general that it can in turn be viewed as a form of single-linkage clustering with a density derived similarity measure.

In the original publication of 2010[452], the name for the CommonNN clustering was coined in analogy to the previously developed *neighbour* algorithm, which is conceptionally, however, quite different.[453] Later, the abbreviation CNN clustering was used.[240, 454] This is avoided here, however, because CNN is prominently occupied by the concept of *convolutional neural networks*. While this term falls into the same broader topic of machine learning like clustering does in general, it is otherwise completely unrelated and thus can be only confusing.

Figure 14.23 illustrates the CommonNN density-criterion on the scikit-learn *moons* data set. Like the other connectivity based clustering methods discussed so far, it is traditionally used with a threshold. Points that share a number of at least n_c neighbours with respect to a neighbour search radius r will be considered connected, rendering the final clusters connected components of the graph formed by these connections.

CommonNN clustering can also be done hierarchically by building a MST from the un-truncated density criterion. Figure 14.24 exemplifies this with the *Iris* data set. Note that the result may not be fully satisfactory in this case but this can be attribute to the relatively low number of samples in the data set. Density-based clustering in general depends on a sufficiently high number of data points for a robust density estimate. This is even more true for CommonNN clustering where the density estimate relies on a sufficient sampling of neighbourhood intersections. The fact that other clusterings like DBSCAN and Jarvis-Patrick clustering seem to perform better for the *Iris* data set should not be taken prematurely as a general qualitative difference.

The details of how we implemented the CommonNN clustering procedure in a convenient and efficient Python package will be lain out in the following in chapter 15.

14.10 Density-peaks

The last density-based clustering procedure I want to put some attention to is density-peaks clustering.[455] So far we discussed connectivity-based density-based clusterings that essentially aimed on the identification of connected components of a super level-set on an approximate probability density. The hierarchies (or hierarchy slices) presented by these methods can be understood in terms of level-set trees, i.e. there has been a focus on where a considered data set splits when a density tuning parameter exceed a certain threshold value. In a way we could say that this view is centred on the minima of the probability distribution underlying a data set. By excluding regions of low density, the high density regions reveal themselves as disjoint components.

prototypes

Density-peaks clustering is interesting because it can be in contrast considered a prototype-based approach. This perspective is focused on the maxima of the (approximate) probability density of a sample set. By identification of the highest density data points, the method tries to find those that could be suitable prototypes for the modes of the distribution that attract the lower density points around them. This idea is similar to what is done by mean-shift clustering,[456] where a set of test points is converged to the closest density maxima by updating their position iteratively to be the mean of their respective neighbourhoods. The way how density-peaks finds the maxima is, however, quite different. An important thing to point out is that density-peaks is able to find non-spherical clusters of arbitrary shape and form—a trait commonly only attributed to connectivity-based clustering procedures.

implementation

Density-peaks is based around two assumptions: 1) the desired cluster prototypes (the cluster centres) are points of relatively high density, surrounded by neighbouring points with lower density, and 2) they are relatively far away from other points of high density in the sense of 1). Practically, we need to assess the density of individual points and their distance to the nearest point of higher density. Local point density can be estimated around each point, like seen before in DBSCAN (section 14.7), as the number of neighbouring points with respect to a neighbour search radius r. Lets denote this number by ρ'_a, the density estimate for point a. The authors of density-peaks claim that the clustering is robust against variations in r because only the relative proportion of density differences between points is of interest, not the absolute value of the density estimate. So while r can still be seen as a resolution parameter that should be set in a reasonable range—a too low value might give a noisy

estimate, a too large value might not be able to resolve density differences—it should not have a direct effect on the clustering result in terms of a tuning parameter.

Next, let's denote the distance of each point to the closet denser point as

$$\delta_a = \min_{b \,|\, \rho'_b > \rho'_a} \left(d(a,b) \right), \tag{14.9}$$

where $d(a,b)$ is a distance function. The densest point is conventionally assigned the maximum distance found in the data set. Proper prototypes are expected to have a much larger δ_a value than other points. Points that at the same time have a very low density may be rather considered outliers, though. For the actual selection of clusters, density-peaks provides a decision heuristic in terms of a two dimensional plot of δ_a versus $\rho\prime_b$ (see figure 14.25a). The user can manually select those points that are both dense and far away from other dense points. All remaining points will be assigned recursively to the same cluster as their closest point with higher density. Note that in contrast in mean-shift clustering, points are typically assigned to their closest prototype, which tends to yield globular clusters and ignores the fact that the closest prototype may represent an actually different data region of high density.

cluster selection

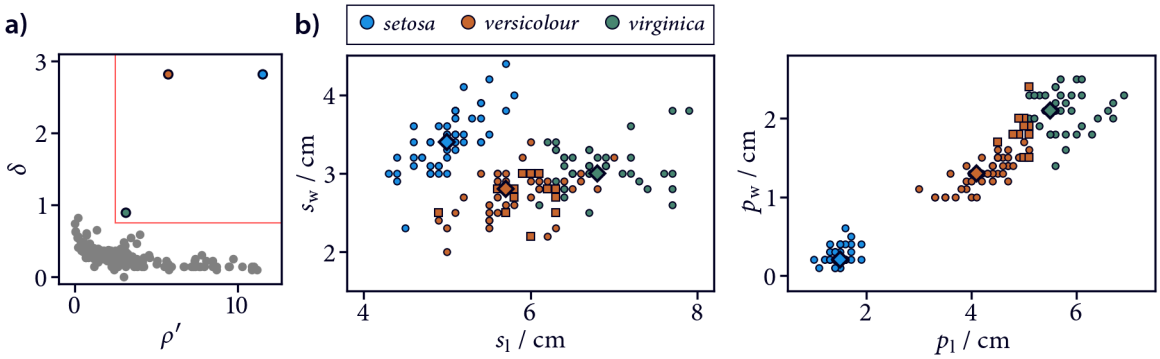


Figure 14.25 ***Iris* data set density-peaks (3 clusters)**

a) Density-peaks decision graph for cluster selection. Valid cluster prototypes are supposed to stand out in terms of a relatively high density and a relatively large distance to the next densest data point. **b)** Data points (compare figure 13.3) with cluster labels from density-peaks clustering (`pydpc.Cluster`[1]). Prototypes highlighted with diamond shapes. 91 % of the cluster labels match the true classification labels. Non-matching assignments are marked with squares.

Within the limits of the applied heuristic to select clusters, density-peaks provides kind of a hierarchical view on the data set as well, only that child-parent relationships are not made explicit. By systematically including an increasing number of cluster centres in the result, data regions that where mingled with the respectively closest denser region become succeedingly separated as their own clusters. The decision of which density-peaks are relevant in the end has to be made by the user.

hierarchy

[1]Visit the development repository on GitHub: `https://github.com/cwehmeyer/pydpc`

{ 15 }

The CommonNNClustering project

Design of a generic Python package

The density-based CommonNN clustering approach and how it works fundamentally has been addressed in section 14.9. We saw that this approach provides a pairwise similarity measure that entails a density estimate, which is suitable to be used in connectivity-based clustering. The clustering can be done using a threshold criterion or in a hierarchical fashion.

This chapter should shed some light on the design process that led to our current implementation of the procedure, which is publicly available as a Python package.[1] The project itself runs under the name CommonNNClustering while the Python package as it was published to the Python package index is currently named `cnnclustering` and after installation importable as such. An independent subset of this project, i.e. essentially the core clustering procedure, was moreover contributed to scikit-learn-extra.[2] A previous implementation of CommonNN clustering is available in terms of the CNNClustering project,[3] which was used so far in related publications.[240, 454] The presented re-implementation of the procedure aims to offer improvements with regard to usability, flexibility, and efficiency. In its core, the project is based on the idea of a generic approach to CommonNN clustering.[4]

Before the actual architecture of the provided package should be described in detail, the following section provides a short detour on the design principle of generic programming, achieved through object-orientation. This is essential to understand how the CommonNNClustering project is structured. The explanation will be focused on Python but the underlying principles are also valid for other programming languages. We use Cython to translate performance crucial elements of the implementation to C/C++.[457] The complexities introduced through this will in general not be discussed throughout this chapter with the exception of a few important details.

15.1 Primer on generic interfaces in object-oriented programming

Python OOP

Python is intrinsically an object-oriented programming language. Although this does not mean that every bit of Python code has to be aware of this nature or has to be based on object-oriented design principles, the language offers wide support of typical mechanisms used in OOP. Python (like for example C++) can still be used following procedural or functional programming paradigms and does not force the use of object-oriented features upon the user[5].

[1] Visit the development repository on GitHub (`github.com/janjoswig/CommonNNClustering`) or the latest release in the Python package index (`pypi.org/project/cnnclustering/`)

[2] Visit the development repository on GitHub: `github.com/scikit-learn-contrib/scikit-learn-extra`

[3] Visit the development repository on GitHub (`github.com/bettinakeller/CNNClustering`)

[4] Before the background that the abbreviation 'CNN' is substituted with 'CommonNN' here (see also 14.9), the package name `cnnclustering` should be probably revised. Furthermore, the whole project is not at all restricted any more to CommonNN clustering and it should be eventually renamed to reflect its generic character better.

[5] See the Python docs for more information: `docs.python.org/3/howto/functional.html`

objects

A key element for object orientation is the concept of *objects*. In general, objects are said to bundle *data* (variables) with *behaviour* (functions). The data on an object defines its current *state* and can be referred to through individual *attributes* of the object. An object is usually aware of itself and can modify its state through the use of functional attributes, also called *methods*. In object-oriented programming, problems that a certain program should address are broken down into operations on and interactions between objects that model the partaking entities.

objects example

As an example, the code that drives a laboratory inventory management system behind the scenes could make use of `User` objects to represent registered staff members that have access to the inventory. Typical data associated with such an object may be the user's name or personal ID, or the current list of items a user has borrowed at the moment to make an experiment. Certain actions that a user can perform, like taking a needed chemical from the stock, may be realized through a respective methods. A selectable article would be probably in turn modelled as a different object as well as the item list itself.

Objects can map to 'real' things that should be represented but also to all kinds of possible entities of only conceptual nature. An ideal breakdown of a problem with a suitable set of objects can be non-trivial to find and is often a core challenge in the design of an object-oriented program. A useful principle to define and grasp objects, is to think of objects as *nouns* (*'who or what is the actor? With whom or what is been interacted?'*) and of its methods (like of functions in general) as *verbs* (*'what is being done?'*). *'User x takes item i from the inventory'* could be a possible action in this particular example and it could be realized through a call of the `take_item(item)` method associated with the user object. This method is internally aware of the object it is called on, i.e. it has a direct reference to the object and can modify its state, which means to manipulate the objects attributes.

classes

Python is furthermore a *class-based*[6] language. A class in object-oriented programming can be understood as an object factory. In this sense, a class knows how to make a certain type of object following some sort of blue-print or template. Objects are called *instances* of the class they were made of. For an instance of a Python class the access of a certain attribute or method on the instance can be delegated to the respective value on the class. Individual instances are, however, usually independent of each other and each instance can have different values for attributes that are directly set on the instance, not on the class. Attributes on the instances are called *instance attributes* while attributes on the class are called *class attributes*.

Methods are fundamentally a special kind of class attribute (descriptors) that are *bound* to an instance when accessed from it. To bind a method to an instance means to insert the instance object itself as the first argument to any call of the method, so that the instance can be referred to from within the method. By convention, this first argument that represents the instance object itself is named *self* in Python. Besides regular methods, Python knows the concept of *classmethods* that are not bound to an instance but to a class instead, and *staticmethods* that are bound to neither instance nor class.

Python makes a differentiation between the *creation* of a new object and the subsequent *initialization* through a constructor. Object creation is done through the `__new__` method of a class that is by design implemented as a staticmethod. The default constructor is defined through `__init__` which is a regular method and operates on a just created instance.

class example

Picking up the 'inventory' example from above, individual user objects could be instances of a `User` class. Every user object will be created and initialized through the `User` class and will have the same set of common attributes and methods but each instance will have individual attribute values. A minimal working example for how to define and use a custom class in Python is lain out below.

[6] The counterpart to class-based is prototype-based. Although Python is fundamentally class-based, the prototype design pattern can still be used to create objects from other objects through cloning.

```
class User:
  def __init__(self, name: str):
    """Default constructor/sets an instance attribute"""
    self.name = name
user = User("Cleese")
another_user = User("Idle")
assert user.name != another_user.name  # passes
```

Note that we did explicitly only provide an `__init__` constructor method here, and left out `__new__`. This is possible because every class (at least in Python 3) automatically *inherits* from `object`. Inheritance is the process of sharing the definitions of a class with other classes further down in the inheritance hierarchy. An inheritance relationship between two classes can be compared to that of child and parent. A child class takes over all the attributes and methods of its parent class. Another valid name for parent is *base*. Inherited attributes and methods on the child can be overridden so that the implemented functionality changes and new ones can be added to extend the child. The class `object` is Python's default class at the top of the inheritance hierarchy. It is the base class for all classes. When we instantiate a customer object in the above example through the syntax of calling the `User` class, this actually creates an object via `object.__new__` followed by a call to `__init__` on the just created object. While `object.__new__` expects the class of which a new instance should be created as the first argument and returns an instance object, `object.__init__` expects the constructor arguments and returns `None`. It is roughly equivalent to:

```
user = object.__new__(User)
user.__init__("Cleese")
```

As a twist, classes are objects themselves. Python knows the concept of *metaclasses*, which can be understood as *class factories* in analogy to classes being object factories. The default metaclass from which all other classes are created is called `type`. [7] As far as we should be concerned here, however, `type` can be also used as a function to retrieve the class of an object.

type vs. class

```
type(user)      # <class '__main__.User'>
user.__class__  # gives the same
type(User)      # <class 'type'>
```

So basically, the *type* and the *class* of an object are virtually identical here. This notion is, however, somewhat different to how types and classes are normally understood in object-oriented design. The type of an object is in general determined through the *interface* it provides. An interface is a set of public attributes and methods that can be accessed on an object to interact with it. Since a class defines such a set, it does also define a type for its objects. But while an object can be a direct instance of only one class it can provide many interfaces and therefore can fulfil the requirements for many different types.

interfaces

Let's contemplate over this with a simple example. Python's `list` class is a standard container. An instance of this class can be considered of multiple types depending on which of its characteristics is emphasized. For the role as a `Container`, it is sufficient that it carries a `__contains__` method through which it can be checked if some item is contained in it. To be called a `Sized` type, it only needs to provide a `__len__` method that returns the number of items it holds. A `list` is also a `Sequence` because it additionally can return individual items by index via its `__getitem__` method. Furthermore it is iterable, reversible, and so forth.

For many of these commonly needed types, Python provides *abstract base classes*.[8] Abstract classes are not meant to be initiated but only serve to define a certain interface. If a type is in this way

abstract base classes

[7] See the Python docs for more information: `docs.python.org/3/reference/datamodel.html`
[8] See also `docs.python.org/3/library/collections.abc.html`

documented via a class, the differentiation between types and classes becomes admittedly again somewhat blurry.

Long story short, the concept of types with certain interfaces can be extremely valuable for the design of a program architecture. Whenever a certain kind of functionality should be implemented, it makes sense to program against interfaces not against specific classes. A function should not care whether it is passed a dict, list, or set as an argument if it only needs to iterate over the contained elements. The principle of writing programs so that they can work with any type of object as long as they only provide the necessary interface is called *generic* programming. For the CommonNNClustering package, we made extensive use of generic interfaces to make the program as flexible as possible and to allow the clustering of data sets in many different concrete formats.

generic programming

15.2 Generic threshold-based CommonNN clustering

Let's recall that in connectivity-based clustering a data set of objects can be conceptually treated as a graph. The objects that should be grouped into clusters are connected by edges corresponding to some relation in the sense of a distance or similarity. Once these connections are known, the clustering problem can be solved via operations on the respective graph. In CommonNN clustering, the similarity measure is given as the number of neighbouring objects that two objects share with respect to a neighbour search radius r. In this sense, the clustering is a special type of clustering using shared neighbours with the extra specification to make use of fixed radius near neighbours. For threshold-based CommonNN clustering, the similarity measure is converted into a binary relation using a minimum number of n_c neighbours that two points need to share to be considered as connected to each other. The desired clusters are the maximally connected components of a graph with these connections. Connected components can in turn be discovered for example in a breadth-first-search (BFS) traversal of the graph.

There are basically two different strategies for the implementation of this. On the one hand, one could construct the graph completely before it is subjected to the connected component search. Then the graph itself would be a primary output of the procedure, which could be interesting if the graph should be processed also in other ways. The graph construction incorporating the CommonNN logic and the cluster extraction would be decoupled. On the other hand, one could run the search directly while exploring the connections of the graph just when they are needed. In this way, the graph is only treated implicitly. The output of the procedure are the clusters themselves without detour. In the present work, we focused so far on the latter of these strategies.

graph construction and traversal

For the actual implementation of the connected-component search and the determination of the connections, there are a lot of possible variations. They arise partly from the fact that there are many different starting points for the clustering. We could for example work for one thing with data point coordinates in a metric space in which neighbourhoods need to be computed before the similarity criterion can be checked, or we could start from already pre-computed neighbourhoods. In the former case, the point coordinates could be presented for example in a classic matrix format (e.g. a NumPy array) or in some other data table (e.g. a Pandas data frame). Different concrete data formats may require slightly different ways to retrieve the needed information from them. We could just restrict the input source for the clustering to a specific format but this would be not satisfactory in terms of the broad usability of the clustering. Besides this, we may want to be flexible with regard to the metric that is used to compute fixed radius neighbourhoods from point coordinates. Furthermore, the comparison of two neighbourhoods with respect to the similarity criterion can be technically realised in more than one way. As a last example, the queuing structure that is required to support the BFS could be implemented differently.

implementation variations

This is where generic programming comes into play. We want to have an implementation of the implicit connected-component search that is agnostic of these details. We do not want to have a whole bunch of implementations accounting for each possible variation but only one that remains unchanged if individual parts of the procedure are exchanged or extended. To make this objective more clear, the code snippet below shows the formulation of an implicit BFS for connected components with the use of abstract types rather than concrete data structures. For the sake of brevity, statements and arguments of secondary importance are omitted here, which is then marked by an ellipsis (`...`). Compare this to the more concrete code example in the DBSCAN section 14.7 and the theory in section 7.1 for more details.

generic BFS

```
# Initialise component label
current = 1
n = input_data.n_points
for init_point in range(n):
  ...
  # Get neighbours of init_point
  neighbours_getter.get(...)
  labels[init_point] = current
  while True:
    for member in neighbours:
      ...
      # Get neighbours of member
      neighbours_getter.get(...)
      # Check neighbours intersection
      if similarity_checker.check(...):
        labels[member] = current
        queue.push(member)
    if queue.is_empty():
      break
    point = queue.pop()
    # Get neighbours of point
    neighbours_getter.get(...)
  current += 1
```

Notice the use of the following generic types here: `input_data` stands in for anything that represents a data set. The algorithm does not need to be concerned with how the data is actually stored here. What matters is that the `input_data` type allows access to an `n_points` attribute that reveals how many objects are in the data set so that we can iterate over it. Next, `neighbours_getter` represents a type that can be called via its `get` method to fill up a generic container, here named `neighbours`. Again, the BFS procedure is unaware of how the `neighbours_getter` extracts the neighbours of a point from the `input_data` and how those are stored in the intermediate `neighbours` container. A `neighbours` container can be anything that allows us to retrieve individual neighbours to then call the `neighbours_getter` again and fill up another container with the neighbours of a neighbour—let's call them the neighbour neighbours. With the retrieved neighbourhoods of two points, the `check` method of a generic `similarity_checker` type can be called to test whether the similarity criterion is fulfilled. As far as the component search is concerned, it is not important how this is done exactly as long as it returns a definite `true` or `false`. During the process, a `queue` type is used to store point indices in one or the other way. Note, that whatever the queue actually does, the BFS only requires it to be called via `push`, `pop`, and `is_empty` methods.

generic types

Virtually the only entity that is not treated generically in this example, is the `labels` object. To be fully consistent, this could be indeed also substituted with a generic type, but at the time there

labels array

was no pressing need to do so. The cluster labels are basically just stored and exposed as an array, which is sufficiently flexibel for most situations. If there should be a demand for a variation of the logic associated with the labels container, it can always be converted into a generic type with which interactions flow via a certain interface later. In general, all building blocks that are used during the clustering can be discretionarily and gradually translated into generic types or treated as concrete data structures. It should be mentioned, though, that generic interfaces may come at a cost. In terms of efficiency, it is naturally almost always faster to interact with objects directly (e.g. indexing a NumPy array) instead of using an extra layer of function calls to interact with them generically (e.g. accessing points in a NumPy array in the same way as any other possible format through a universal `get` method that hides the internal logic of point retrieval).

generic vs. concrete

The next section will discuss how such a generic BFS and the respective generic types are dealt with in our CommonNNClustering project. To avoid unnecessary confusion, here is a brief note on the notation that is used to address the different objects and classes. In compliance with the conventions communicated in PEP8,[9] classes (including generic types represented by abstract classes) will be written in camel case, i.e. using no underscores and capitalised words, like in `InputData` and `NeighboursGetter`. Object instance names that appear as variables in code examples like above will in contrast be written in all lowercase with underscores, e.g. `input_data` and `neighbours_getter`. Apart from that, the different clustering building blocks that are represented by generic types will be generally referred to in text also without markup, e.g. just as input data and neighbours getter.

comment on notation

15.3 Package realisation and basic usage

Taking the generic BFS formulation from the last section one step further, we decided to provide another exchangeable type that represents the component search itself: the fitter. Within our CommonNNClustering framework, the clustering procedure can be swapped entirely against a different one just by selecting a different `Fitter` type that is essentially interacted with by the same interface (essentially a `fit` method) but that may contain a completely different implementation. As an example, one could use a depth-first-search (DFS) strategy instead or modify the algorithm to implement DBSCAN or Jarvis-Patrick clustering.

fitter

Figure 15.1 summarizes the design idea for the CommonNNClustering package with generic types. For the convenience of the user, we provide a main `Clustering` class. The API of the package contains a set of classes that define and implement the generic clustering building blocks described above. For the actual execution of a clustering, these building blocks need to be combined and called in a certain way. This provides a maximum degree of flexibility but on the downside can be fairly complex. The `Clustering` class provides a high-level user interface to allow a simple and intuitive execution of clusterings. The usage of this class is quite similar to how other popular Python packages like for example scikit-learn expose their clustering protocols and should feel familiar to users that have already some experience with those. The internal details of how a clustering is realised are hidden from the unconcerned user as much as possible. In essence, the execution of a clustering is as simple as this:

Clustering class

```
clustering = Clustering(data)
clustering.fit(**params)
clustering.labels  # array([...])
```

First, a `Clustering` instance (here named `clustering`) needs to be created with some input data. On initialisation, the correct assembly of the necessary clustering building blocks is taken care of in

[9] See `peps.python.org/pep-0008/`

the background. This follows the builder design pattern,[458] and is described in section 15.3.1. On first approximation, a `Clustering` object bundles the input data alongside a `Fitter`. By calling the objects `fit` method with suitable cluster parameters, the execution of the clustering is delegated to the `Fitter`, which in turn orchestrates the procedure and populates a labels container as the result.

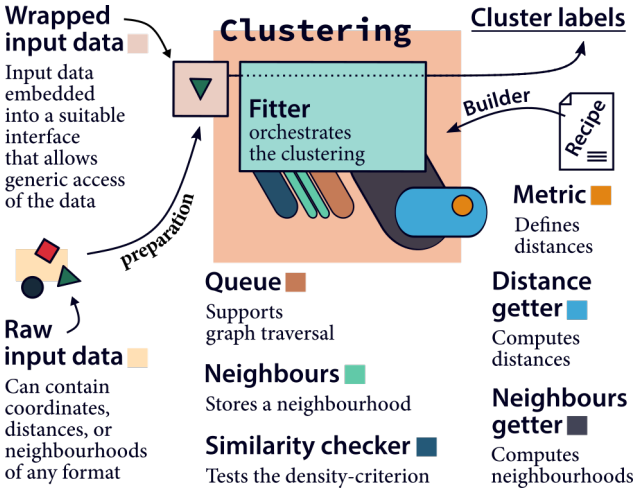


Figure 15.1 Aggregation of generic types for CommonNN clustering The `Clustering` class provides the main pathway for user interaction with the clustering functionality. Basically, this class bundles input data (i.e. a data set of arbitrary format wrapped into a suitable `InputData` type) with a `Fitter` that controls how a clustering is executed. The `Fitter` does in turn carry a set of generic building blocks that are needed during the procedure. For details on how the correct aggregation is controlled using a recipe and a `Builder` on `Clustering` initialisation see section 15.3.1.

Like a `Clustering` object is associated with a `Fitter`, the `Fitter` bundles generic types like for example a `NeighboursGetter`. As can be also seen from figure 15.1, there are still more generic types involved than the already discussed. In particular, a `NeighboursGetter` may require also a `DistanceGetter` to work properly. A `DistanceGetter` does in general depend on a `Metric`. The way how the individual building blocks are associated with each other follows the fundamental approach of object *aggregation*. This means that a certain building block possesses a number of attributes that refer to other lower-level building blocks that it depends on. The behaviour of the higher-level building block is controlled by the types it aggregates. Object attributes that refer to a generic building block are named after the respective abstract type with a leading underscore to indicate their use as private attributes. The `Fitter` type associated with a `Clustering` for example will be found under the `Clustering._fitter` attribute.

object aggregation

The flow of method calls from tip to toe does in general look like the following when a user initialised a `Clustering` object and wants to execute a clustering: 1) The user calls `clustering.fit` to trigger the procedure. 2) The call is (after a few basic checks and a general setup) delegated to `clustering._fitter.fit`, which contains the generic clustering logic. 3) During the execution, different generic building blocks may be utilised, for example via `~._fitter._neighbours_getter.get`. In this case, input data and neighbours container types are passed to the `get` call. 4) Depending on the logic of this method, this may involve calls to `~._neighbours_getter._distance_getter.get`. 5) If the input data contains point coordinates, `~._distance_getter._metric.calc_distance` will eventually return a specific distance between two points. The whole purpose of this chain of delegations is to allow the exchange of individual parts of it. While the methods of the used generic types are always called in the same way, the logic they entail and the kind of further types they call may be very different.

chain of delegations

Figure 15.2 gives a rough overview of the most important generic and non-generic types as a UML class diagram. For each of them, the diagram shows the relation to other types and it also lists a selection of attributes and methods. A `Clustering` aggregates primarily an `InputData` type and a `Fitter`. A `Clustering` is also equipped with an instance of `Labels` that holds the cluster label assignments and an instance of `Summary` that collects statistical records for past clustering executions. A `Fitter` aggregates for example a `Queue`, a `SimilarityChecker`, a `NeighboursGetter`,

relations between types

and normally two `Neighbours` containers. It should be noted though that other fitters may use a different set of building blocks. The `NeighboursGetter` fills the `Neighbours` container, so that the `SimilarityChecker` can check if they fulfil the similarity criterion. Both require access to the cluster parameters because to get the neighbours of a point we may need the neighbour search radius r and to check the similarity we need the similarity cut-off n_c. These parameters are passed around via a `ClusterParameters` object that has to be created before `Fitter.fit` is called, which will be also done internally by `Cluster.fit`.

Italic class names indicate here that the respective classes are abstract classes. These define an interface but are not to be initialised. The actually aggregated objects need to be concrete realisation of those. The standard BFS clustering procedure is for example exposed by the concrete type `FitterBFS`. For `InputData`, the interface is, however, further differentiated by inheritance into `InputDataComponents` to store point coordinates (or distances), `InputDataPairwiseDistances` and `InputDataNeighbourhoods`. The difference between the two types `InputDataComponents` and `InputDataPairwiseDistances` is that in the former case information is retrieved from it via a `get_component` method. This is expected to be called by a `Metric` type. `InputDataComponents` can still be used to store distances if a dummy metric (e.g. `MetricPrecomputed`) is used. On the contrary, `InputDataPairwiseDistances` reveals its content via a `get_distance` method, which bypasses the metric type and is supposed to be used for distances only. For distances and neighbourhoods input data types, there also exist corresponding `Computer` interfaces. These are extended by methods that can be used to compute distances and neighbourhoods in bulk. For each of these input data type, a matching `NeighboursGetter` needs to be selected that can actually deal with it and implements the necessary logic to extract neighbours from it.

Cython extension types

For `InputDataComponents`, the diagram shows an example for a concrete realisation, namely `InputDataExtComponentsMemoryview`. The `Ext` in the class name indicates that this is a Cython extension type instead of a regular Python class. These types are generally more performant and whenever possible they should be used instead of the pure Python counterparts. It should be noted that extension types can be used as building blocks aggregated by Python types but in reverse extension types may only aggregate other extension types. Instances of `InputDataExtComponentsMemoryview` use a Cython specific *typed memoryview* to store their content and allow very fast C-array-like item access and processing.

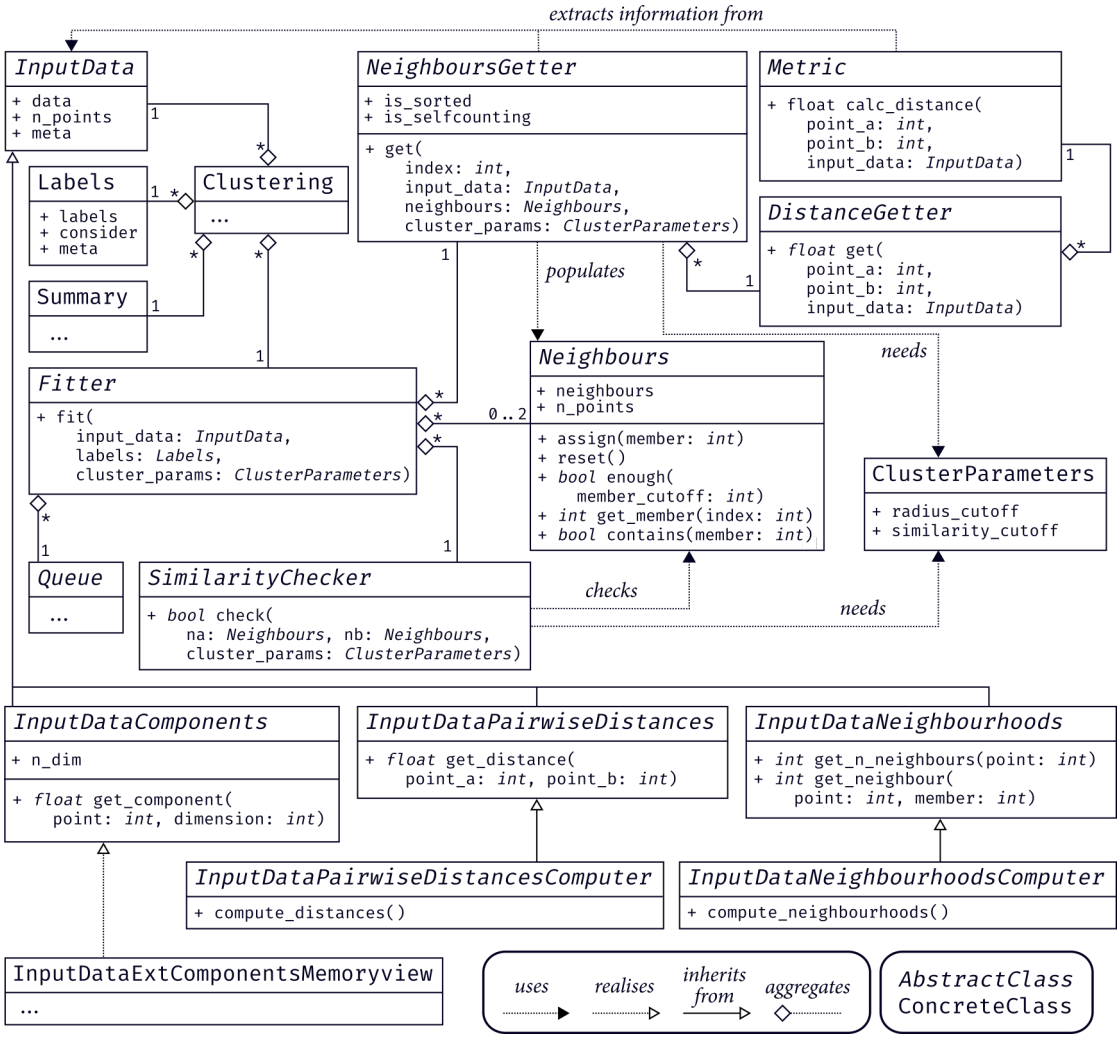


Figure 15.2 **UML class diagram for the CommonNNClustering project**

Each of the shown classes is represented by a box with sub-boxes for a header with the class name, selected attributes, and selected methods. Italic class names denote abstract classes that define the interface for a (generic) type, while concrete classes have upright names. Relations between the classes are indicated by different kinds of arrows, pointing from a class *A* to a class *B*. Dotted arrows with filled heads describe associations of the form 'class *A* uses class *B* (e.g. as an argument to one of its methods)'. Dotted arrows with empty heads say 'concrete class *A* realises abstract class *B*', while solid arrows with empty heads say '(abstract) class *A* inherits from (abstract) class *B*'. Most importantly, solid arrows with empty diamonds stand in for 'class *A* is aggregated by class *B*'. These arrows are annotated according to the ratio of the aggregation, e.g. a `Clustering` aggregates only 1 `Fitter`, but the same `Fitter` can be a part of many (`*`) `Clustering` instances. Function argument and return types are set in italic (`void`/`None` is omitted).

15.3.1 Internal aggregation

We discussed the fundamental architecture of the CommonNNClustering project and a typical set of generic clustering building blocks used in threshold-based CommonNN clustering. As already mentioned, the provided building blocks can be used directly for a clustering if they are assembled in a sensible manner. To get a better impression on how this could actually look like, let's assume we have some example input data of point coordinates stored in a 2D NumPy array. How do we proceed from here?

manual aggregation

First, it is required to choose a suitable `InputData` type to wrap our raw data in, to be generically processable (see figure 15.1). The best choice for this would be the highly performant `InputDataExtComponentsMemoryview` class (see section 15.4 for more information on where to find available types). This class can be initialised with the data right away if our NumPy array of data points is of the shape $n_{\text{points}} \times n_{\text{dimensions}}$, of `dtype=numpy.float64`, and C-order memory layout. If this was not the case, the data would need to be pre-processed accordingly (see further below and section 15.4 for more information). Next, we need to choose a metric, since we have to compute distances in order to extract neighbourhoods from the given point coordinates. We can for example settle on `MetricExtEuclideanReduced`, which needs no arguments on initialisation and computes reduced (squared) Euclidean distances as its name suggests. Now, we take this `Metric` object and initialise a `DistanceGetter` with it, for which the obvious choice would be `DistanceGetterExtMetric`. Then we continue by initialising a `NeighboursGetter`, e.g. `NeighboursGetterExtBruteForce`, which expects the distance getter on initialisation and computes neighbourhoods brute force by commanding distance calculations from a point a to all other points in the data set to identify its neighbours. We need more building blocks, though, if we want to use the fitter `FitterExtBFS` for the clustering. That would be a queue (say the first-in-first-out `QueueExtFIFOQueue` that is based around a C++ `std::queue`), a similarity checker (e.g. `SimilarityCheckerExtSwitchContains`, which will be further discussed in section 15.6.1) and two neighbourhood containers (e.g. `NeighboursExtVectorCPPUnorderedSet`, also addressed in section 15.6.1).

The full aggregation of the admittedly somewhat cryptic types above is expressed in the following code example. At the end, it is also shown how to start the clustering procedure for which we also need `Labels` and `ClusterParameters`. For the creation of the latter, it is recommended to use the `make_parameters` method of the established fitter to ensure that the parameters are consistent with what the fitter actually requires.

```
input_data = InputDataExtComponentsMemoryview(data)
metric = MetricExtEuclideanReduced()
dgetter = DistanceGetterExtMetric(metric)
ngetter = NeighboursGetterExtBruteForce(dgetter)
queue = QueueExtFIFOQueue()
checker = SimilarityCheckerExtSwitchContains()
na = NeighboursExtVectorCPPUnorderedSet()
nb = NeighboursExtVectorCPPUnorderedSet()
fitter = FitterExtBFS(
  ngetter, na, nb, checker, queue
  )
labels = Labels.from_length(data.shape[0])
cluster_params = fitter.make_parameters(
  1.5, 1, 1  # r, nc, start label
  )
fitter.fit(input_data, labels, cluster_params)
```

Builder & recipes

Such a manual setup is obviously not very convenient. To simplify these steps, we established the concept of a `Builder`,[458] which is a class that has the only purpose of reproducibly assembling above components for direct use. A builder requires a recipe that it can follow, which is basically a mapping of argument values to generic types. For the above composition, the respective recipe can be defined as such:

```
recipe = {
  "input": "components_mview",
  "prep": "points_from_parts",
  "fitter": "bfs",
  "fitter.ngetter": "brute_force",
```

```
    "fitter.na": "vuset",
    "fitter.checker": "switch",
    "fitter.queue": "fifo",
    "fitter.ngetter.dgetter": "metric",
    "fitter.ngetter.dgetter.metric": "euclidean_r",
}
```

Each key-value pair in this mapping defines which generic type should be used for which building block. For nested aggregations, the keys contain dots to indicate lower building block levels. Note, that the recipe refers to the building blocks and generic types with brief names and aliases (see the next section 15.4 for more information on where theses names and recipes in general can be found). The key `"prep"` refers to a function that will be used to prepare the passed input data in some way before it is wrapped in a specific `InputData` type (see also section 15.4). Instead of string identifiers, a recipe does also allow function and class objects directly to be used as valid values. Furthermore, values can be a tuple of the form `("value", (), {})` to allow the extra specification of initialisation arguments and keyword arguments. This recipe can now be used to initialise a builder and to create input data and fitter.

```
Builder(recipe, **recipe_kwargs)
input_data = builder.make_input_data(data)
fitter = builder.make_component("fitter")
```

Instead of passing a mapping as the first argument on `Builder` initialisation, it is also possible to refer to a pre-defined recipe with a string identifier. The example recipe is accessible through the name `"coordinates"`. Other currently available recipes are `"distances"`, `"neighbourhoods"`, and `"sorted_neighbourhoods"`. Remaining keyword arguments passed on initialisation are used to override the base recipe, e.g. like `**{"fitter.na": _types.NeighboursExtVector}` or equivalently `fitter__na="vector"` (note the use of double underscores instead of dots in this case because dots would be misunderstood by Python in keyword arguments as attribute access). When a `Clustering` is initialised, it is a builder that controls the assembly of the object behind the scenes.

15.4 Module overview

The CommonNNClustering project is organised into several sub-modules. Here is a quick overview of where to find what within the `cnnclustering` Python package. Each of the modules can be imported, e.g. by a statement of the form `from cnnclustering import cluster`. Some of the modules are Cython extensions, which can be imported from Python as regular modules. To use these extensions from Cython, it is required to import them for instance as `from cnnclustering cimport _types`. Cython extensions are marked with a superscribed C (e.g. $\texttt{\_types}^{\mathsf{C}}$).

`cluster` Provides the main high level entry point of the API through the `Clustering` class. Besides the bundling of generic clustering building blocks, it exposes convenience functions for the evaluation and post-processing of clustering results.

`recipes` Provides the `Builder` class for clustering building block aggregation. It also defines available default recipes (`REGISTERED_RECIPES`), a set of names to refer to individual building blocks (`COMPONENT_NAME_TYPE_MAP`), and available aliases (`COMPONENT_ALT_KW_MAP`). It is also the place where preparation functions can be found to convert raw input data into something that can be wrapped by an input data type (named by convention `prepare_<*>`), e.g. the function `prepare_points_from_parts` that takes data point coordinates in nested sequences and returns them ready to be wrapped by `InputDataExtComponentsMemoryview`.

plot Convenience functionality for different kinds of plots using Matplotlib. Several functions are only available if Networkx or SciPy are installed. Most of the provided aspects are either directly accessible or via respective methods of a `Clustering`.

report Functionality to collect (record and summarise) cluster results, normally done on the same `Clustering` object. Mainly provides a `Record` and a `Summary` class.

_types^{C} Crucial low level extension. Defines the interface for generic types through Python abstract base classes and in case of extension types additionally through a concrete `<*>ExtInterface` variant because Cython does not support abstract extension types in the same sense as Python does. Also provides concrete realisations for each type. This is the place where to look for if a user needs to implement a new custom type, e.g. an unsupported metric or an input data type that can deal with specific data. Provides also `Labels` and `ClusterParameters`.

_fit^{C} Defines the `Fitter` interface and provides concrete implementations. Also comprises the up to here not discussed `Predictor` and `HierarchicalFitter` interface.

_bundle^{C} As an additional layer of indirection, a `Clustering` object is not actually aggregating an `InputData` object itself but instead a `Bundle` that is defined here. A bundle brings together the input data and the cluster labels or a corresponding graph structure. Essentially it levels the field to deal with cluster hierarchies as it can be engaged in child-parent relations with other bundles. See sections 15.8 to 15.10 for more details on hierarchical clustering.

_primitive_types^{C} Lowest level definition of integer, float, and boolean values.

15.5 Technical remarks

Before we proceed further to examples of CommonNN clustering in use, this section should address a few assumptions and decisions that went into the current implementation. First of all, we should sort out a few things concerning the notation of parameter values because this was treated quite differently in the literature so far.

The core of the CommonNN concept are fixed radius ball neighbourhoods $\mathcal{B}_r$ (compare equation 13.7). Data point density is estimated from the intersection of the neighbourhoods of two points $\mathcal{B}_r(a)$ and $\mathcal{B}_r(b)$ as

$$\rho'(a,b) = n_{\mathrm{sn}}(a,b) = \mathrm{card}\big(\mathcal{B}_r(a) \bigcap \mathcal{B}_r(b)\big)\,. \tag{15.1}$$

present notation

The dash in ρ' should emphasise that this is only an estimate on the true (unknown) probability density ρ based on discrete point samples. Here, I denote the cardinality of the neighbourhoods intersection, i.e. the number of points that are in both the neighbourhoods, with n_{sn} with a subscript for *shared neighbours*. Such a density estimate is also valid for shared neighbours clustering in general with a difference in how the neighbourhoods are defined (compare for example k-nearest neighbourhoods in Jarvis-Patrick clustering discussed in equation 13.8 and section 14.8). The density estimate is considered a similarity measure that can be converted into a binary relation using a cut-off (threshold) criterion (referred to also as similarity or density criterion) n_{c} that is a minimum number of neighbours that need to be shared by two points to be considered connected

$$s(a,b) = \begin{cases} 0, & \text{for } n_{\mathrm{sn}} < n_{\mathrm{c}} \\ 1, & \text{for } n_{\mathrm{sn}} \geq n_{\mathrm{c}} \end{cases}\,. \tag{15.2}$$

Additionally, it is possible to normalise the density estimate by the volume of the neighbourhood intersection to counter the fact that this is not a constant[459]

$$\widetilde{\rho}' = \frac{\rho'}{V_{\mathrm{I}}(a,b)} \,. \tag{15.3}$$

In the original publication introducing threshold-based CommonNN clustering, the two necessary cluster parameters were called the 'nearest-neighbour-distance cut-off *nndc*' (here the neighbour search radius r) and the 'nearest-neighbour-number cut-off *nnnc*' (here n_c).[452] Subsequent publications used R (called the neighbourhood parameter) and respectively N instead.[240, 454] The corresponding implementation used the flags `-Cut`/`--Cut-off` for R and `-Sim`/`--Similarity` for N as arguments to the clustering script but used the notation of the original paper internally. In the present implementation, the standard fit-function arguments are called `radius_cutoff` and `similarity_cutoff` (or `cnn_cutoff` for backwards compatibility).[10] In the scikit-learn-extra implementation, the `CommonNNClustering` class expects the parameters as `eps` and `min_samples`, which is universal for clustering procedures part of the scikit-learn universe.

other notations

There are multiple practical decisions to make for the implementation of CommonNN clustering. For one thing, we can either use closed- or open-ball neighbourhoods. So far there seems to be a general consensus on closed-ball neighbourhoods so that points that are exactly found at a radial distance r from a reference point a will be included in the neighbour list of a. The currently available `NeighboursGetter` types adhere to this convention but note that the addition of alternative behaviour in other types is trivial. Furthermore, neighbourhoods that were pre-computed in any possible way can also be fed into the clustering procedure. The discrepancy between closed- and open-ball neighbourhoods in the clustering output is expected to be minor for most data sets.

closed- vs. open-ball neighbourhoods

Another more critical decision to make is whether neighbourhoods should be constructed in a *self-counting* or *self-exclusive* manner. Self-counting neighbourhoods treat each point as its own neighbour. The original formulation of CommonNN clustering assumes that and it is also commonly done by scikit-learn's neighbour computation functionality. As a consequence, neighbourhoods will never be empty and points that are neighbours of each other will automatically have at least two shared neighbours, namely themselves. This perspective is consistent with the formulation of the density estimate in terms of the number of points that are found in a neighbourhood intersection. Two neighbouring points lie in fact both in their neighbourhood intersection volume and consequently the density estimate can be considered higher than for point pairs that are not neighbours. On the other hand, if neighbourhoods are self-exclusive, neighbourhoods can be empty and points that are neighbours of each other have 0 shared neighbours unless there is at least a third point in their neighbourhood intersection. This perspective is arguably more intuitive in terms of the formulation of the similarity criterion as the number of 'shared' neighbours. In other words, while self-counting neighbourhoods represent an intersection focused view ('*how many points are found in a specific volume element?*'), the self-exclusive view focuses on the connectivity between two points ('*how many other neighbouring points are shared by the two considered points?*').

self-counting vs. self-exclusive neighbourhoods

Figure 15.3 illustrates the difference for the density estimate with a simple example of two points at varying distance to each other. In the first series, where only the two points themselves are considered, self-exclusive neighbourhoods always result in a zero density estimate while self-counting gives rise to a jump in the density once the two points are neighbours. Normalisation by the intersection volume dampens this jump. In the second series, where a third point is included, the self-exclusive estimate is only effected by this other point while self-counting again gives rise to a density jump once the

[10] It should be noted, though, that a `Fitter` can in principle take any arguments to its `fit` method.

two main points become neighbours. Normalisation has the somewhat counter-intuitive effect of lowering the density estimate significantly in the self-exclusive case.

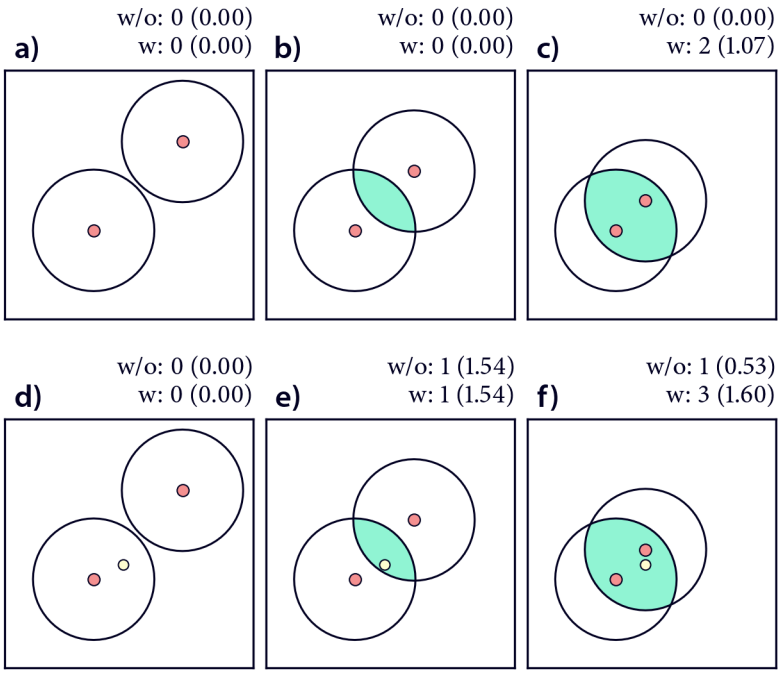


Figure 15.3 Density estimate for neighbourhoods with and without self-counting For two points coming closer together, the number n_{sn} of points in their neighbourhood intersection with and without self-counting is noted above the subfigures in absence of any other points (**a)**, **b)**, and **c)**) and in presence of a third point (**d)**, **e)**, and **f)**). The n_{sn} values normalised by the intersection area are shown in parentheses.

pair candidate selection

Which of the neighbourhood conventions is chosen, effects the values of the density estimate and the values to be set for the similarity threshold, but it is also coupled to another important decision to make: between which pairs of points do we check the similarity criterion? Figure 15.4 illustrates a few possibilities for schemes to select a pair of candidate points for which the similarity should be determined.

The naive approach would be to check all possible pairs. This is, however, very wasteful not only because there can be a lot of pairs but also because the density between points that are at least $2r$ away from each other will always be zero. The approach that is normally followed instead and which also the currently provided `Fitter` types are based on, therefore makes the assumption that it is sufficient to check only pairs of points that are neighbours of each other in the first place. This drastically limits the number of point pairs that need to be checked. It also limits the possible differences in the neighbourhood intersection volumes because two checked points are a maximum distance of one neighbour search radius r away from each other. In practice, this assumption yields satisfactory results.[240, 454] A detailed assessment of contingently resulting discrepancies between an exhaustive check of all pairs and a check of neighbours only has not been done so far, though.

Alternatives that can be considered a compromise between the two first approaches would be for example to use a buffer region ϵ around the neighbourhood of each point and check all pairs that have a distance of at most $(r + \epsilon) < 2r$ between each other. Furthermore, it would be possible to check the density criterion between a point and all its k-nearest neighbours. Especially in low density regions this is, however, expected to yield a lot of unnecessary checks because the distances to the nearest points can be fairly large. It is, however, not clear if any of these approaches could constitute a substantial improvement of the clustering that is worth the expected performance cost and the introduction of an additional ϵ or k parameter.

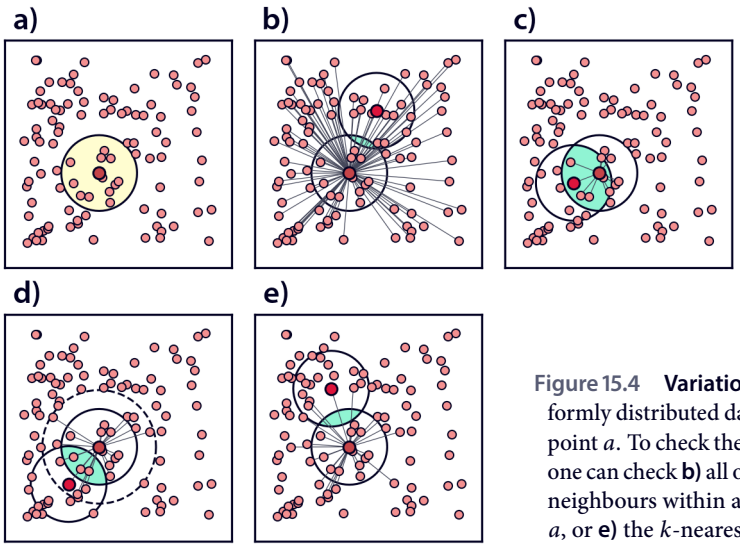


Figure 15.4 **Variation of pair candidates scheme** Within uniformly distributed data points, **a)** shows a neighbourhood for a point a. To check the similarity criterion for a and other points, one can check **b)** all other points, **c)** only the neighbours of a, **d)** neighbours within a buffer-extended neighbourhood around a, or **e)** the k-nearest neighbours of a.

Under the assumption that only pairs of neighbouring points will be subjected to a similarity check, we can come back to the last decision about self-counting versus self-exclusive neighbourhoods. In this case the two conventions are interconvertible. The current standard fitter `FitterExtBFS` is based on the self-exclusive approach, that means the density estimate has a minimum value of $n_{sn} = 0$. When the cluster parameters are prepared before the fit, the similarity cut-off n_c specified by the user needs to be adjusted in case that self-counting neighbourhoods are provided. This information can be collected from the fitter by accessing a respective `is_selfcounting` attribute on its associated neighbours getter (the attribute needs to be set when a `NeighboursGetter` type is initialised). If it finds self-counting neighbourhoods, n_c is increased by 2 to account for the fact that a neighbourhood intersection contains two neighbours when it is expected to contain none. The advantage of this is that the user can always pass the same n_c value, based on the self-exclusive convention, no matter which convention is actually used for the neighbourhoods. Additionally, the high level `Clustering.fit` method offers a `similarity_offset` (`cnn_offset` for backwards compatibility) keyword argument so that the user can pass the n_c value as if the fitter was based on the self-counting convention. When `similarity_offset=2` is given, n_c is internally decreased by 2.

n_c adjustment

One should be careful though with this conversion if the assumption that two checked points are always neighbours is not made. In this case, it has to be differentiated during the clustering if two checked points are actually neighbours (n_c may need adjustment) or not (no adjustment). As a side note, the assumption that two checked point have to be neighbours is also practical for another reason. In this case, the behaviour of the clustering $n_c = 0$ in the self-exclusive view is unambiguously defined. Points that are neighbours but have no further shared neighbours will nonetheless be part of the same cluster. For checked point pairs that could be non-neighbours, it has to be differentiated between neighbouring and non-neighbouring pairs in the situation where $n_{sn} = 0$. It might be conceptually the cleanest option to forbid $n_c = 0$ entirely then.

neighbours vs. non-neighbours

Apart from these quite essential decisions, there are also a few more technicalities that solely concern the efficiency of the clustering procedure. When during a clustering the neighbourhood of a certain point was collected, it is possible to skip the normally following similarity check if the neighbourhood does not have enough members to possibly pass the check in the first. Say we set $n_c = 10$ and a just collected neighbourhood turns out to have only nine members. Then we do not need to check the similarity criterion and can exclude the point entirely for the rest of the procedure as it will never share enough neighbours with another point. The standard fitter `FitterExtBFS` includes this short-cut.

skipping similarity checks

The check can also be executed in bulk before the clustering if neighbourhoods are pre-computed. Note that for the adjustment to self-counting neighbourhoods, the required member count for a neighbourhood to be forwarded to a similarity check is increased by 1 because one member is always the checked point itself.

cluster initialisation

The next remark concerns an assumption that was made in the previous CommonNN publications. In order to make the clustering more efficient, the exploration of a new cluster was not started with an arbitrary point but always with the one still unassigned point that has the currently highest neighbour count, i.e. that is found in the densest data region. In light of the original implementation at the time, this made sense. Here, the basic algorithm behind the clustering entailed repeated loops over all points in the data set and points that have been already assigned to the current cluster to check if more points could be added to it.[452]. In this case, it is good advice to explore the densest clusters (the ones to which the presumably largest portion of the data set can be assigned) first because it ensures that the first (and most expensive) loops over the whole data can add as many points as possible, which can than be ignored in subsequent loops. For the present BFS approach that recognises the desired clusters as connected components, it is, however, not so clear if such an approach yields a general performance improvement. It is true that in special cases, it may be beneficial to explore the densest parts of a data set first if the following reasoning holds: a cluster growth step is efficient if the respective similarity check is successful because a point will be added to the current cluster and will be only checked once again when it is considered as a potential starting point for further cluster growth. It is not efficient if the similarity check fails because in this case the checked point partner will not be added to the current cluster and remains available for further similarity checks. In the worst case, a single point will be checked many times before it is eventually merged into a cluster or chosen as the source point of a new cluster. Checks in high density regions are more likely to pass than checks in low density regions and hence it can be advantageous to prioritize points with high member counts.

It should be, however, considered that a selection of source points in dense regions is only possible if neighbourhoods are pre-computed and that it comes at the cost of finding these source nodes. A gain in clustering efficiency is only worth it if it makes up for the additional overhead. In the current default fitter `FitterExtBFS`, no such effort is implemented. Instead, source nodes are just found by iterating further trough the data set from top to bottom until an unassigned point is found. To mimic the behaviour of source selection according to neighbour counts, one may pre-compute and pre-sort the points in the data set after their neighbour count, but beware that the actual order of points in a data set may need to be remembered for later (e.g. in case of MD data). For simple toy data sets, it does indeed look like such an approach can produce very fast clusterings (see section 15.6) but for definite statements, excessive benchmarks will be necessary, and representative benchmarks are notoriously difficult to achieve (see chapter 16).

member cut-off

As a last remark, it may be desired to control the minimum size of valid clusters. While it is in principle possible to integrate this demand into the clustering procedure itself, the present approach defers it to an optional post-processing step after the clustering. The `FitterExtBFS` type numbers the clusters it yields by default with increasing cluster labels starting by default from 1 and identifies single member clusters among them as separate clusters. The resulting `Labels` container provides a `sort_by_size` method that can be used to clean-up the result by sorting the cluster labels in such a way that the largest cluster will get the lowest label. In this course, it can also declare clusters with less than `member_cutoff` members as noise (labelled with 0)[11] and remove them from the cluster label assignment. `Clustering.fit` offer keyword arguments to sort the labels, and to set the member cut-off and the starting label for newly found clusters.

[11]Note that the scikit-learn clusterings denote noise with -1 and label valid clusters starting at 0 instead.

15.6 Fast threshold-based clustering

So now that we have a first overview of the CommonNNClustering project, it is time for an actual clustering example and advice on how to run it to get the fastest result. We will consider six toy 2-dimensional toy data set of 2000 points each, which can be created using the `sklearn.datasets` module.

As has been discussed, there are many options for how to feed data into a clustering. When we have actual point coordinates available, it seems straightforward to directly use them as the input source. During the clustering, distances can be calculated from the coordinates, which can than be used to identify the neighbours of points when two points should be subjected to a similarity check. This can, however, be quite expensive because it may be necessary to calculate the same distances (and neighbourhoods) multiple times if individual data points are involved in more than one check. Note, that so far no `NeighboursGetter` or `DistanceGetter` supports caching but respective additions would be of course possible. The advantage of this direct on-the-fly approach is that the memory demand of the procedure is minimal since besides the input coordinates, only intermediately needed distances and neighbourhoods have to be stored somewhere. Note also that it can make a significant difference how distances are calculated. Brute force calculation (`NeighboursGetterExtBruteForce`) offers just a baseline. The hybrid input data type `InputDataSklearnKDTree`, storing point coordinates alongside a *kd*-tree, in combination with `NeighboursGetterRecomputeLookup`[12] that can extract neighbours from the input data (the tree) and trigger the re-building of the tree on demand, may be for example an alternative. These types are, however, only available as pure Python classes at the moment and not used in the provided default recipes.

on-the-fly approach

It might be worthwhile to instead pre-compute distances to start the clustering from here and to avoid repeated distances calculations. This offers the flexibility to leverage any possible external solution to efficiently prepare the distance input. It should be noted, though, that a bulk computation of pairwise distances usually has a high memory demand. In the worst case, a $n \times n$ square distance matrix needs to be stored, which becomes quickly infeasible. Input data types that support sparse storage and access distances can be an option but currently we do not provide any. Users are, however, encouraged to implement custom `InputData` types as needed.

bulk approach

If one is willing to pre-compute distances, why not go the full way to pre-compute neighbourhoods. The gain in performance is expected to be even greater. Again, we can take advantage of any available tool to outsource the calculation of neighbours in the most efficient way possible. Also the memory demand of a neighbourhoods calculation in bulk is usually much lower than for distances. Especially for small neighbour search radii r, the average number of neighbours that each point has is expected to be much smaller than the total number of data points. A possible disadvantage of neighbourhoods over distance could be only that they are less reusable. Once distances are available, clusterings with different parameter combinations can be run without limitation. For neighbourhoods, a variation in r necessitates a re-computation.

There exist two further optimisations when input data is provided in terms of neighbourhoods. First, it can make a big difference if each neighbourhood is sorted by the indices of the members it contains. This means for example if a neighbourhood of a point contains the points 3, 4, 1, 7, 8, and 0 of the data set, it is a good idea to store it in an equivalent of the list `[0, 1, 3, 4, 7, 8]`. See section 15.6.1 for the explanation why. For this setup, there is a recommended recipe, which will be shown further below. Furthermore, it might be also a good idea to sort all neighbourhoods relative to each other so that point 0 in the data set is the one with the highest neighbour count (see the end

[12] Note that figure 15.2 shows how the `InputData` interface is differentiated. A `NeighboursGetter` needs to be compatible with what the input data provides.

of section 15.5). The following code example illustrates how both these optimisations are practically realised, using a *kd*-tree as provided by scikit-learn to pre-compute distances:

```
from cnnclustering import cluster
import numpy as np
from sklearn.neighbors import KDTree
# Pre-compute distances (data of coordinates in NumPy array)
tree = KDTree(data)
neighbourhoods = tree.query_radius(
  data, r=params["radius_cutoff"], return_distance=False)
# Sort by neighbour count (and remember sort-order)
n_members = np.array([n.shape[0] for n in neighbourhoods])
sort_by_member_count = np.argsort(n_members)
revert_sort = np.argsort(sort_by_member_count)
neighbourhoods = neighbourhoods[sort_by_member_count]
neighbourhoods = [revert_sort[n] for n in neighbourhoods]
# Sort each neighbourhood by member indices
for n in neighbourhoods:
    n.sort()
# Clustering with recipe
clustering = cluster.Clustering(
  neighbourhoods, recipe="sorted_neighbourhoods")
clustering.fit(**params)
# Get labels in original order
clustering.labels[revert_sort]
```

The aggregation of clustering building blocks according to the specified recipe on clustering initialisation looks as follows. One can get this information by calling `Clustering.__str__`, e.g. via `print(clustering)`. Note that this recipe requires individual neighbourhoods to be sorted and will produces nonsense otherwise. Sorting of neighbourhoods by member count is optional, though.

```
Clustering(
  input_data=InputDataExtNeighbourhoodsMemoryview,
  fitter=FitterExtBFS(
    ngetter=NeighboursGetterExtLookup(
      sorted=True, selfcounting=True),
    na=NeighboursExtVector,
    nb=NeighboursExtVector,
    checker=SimilarityCheckerExtScreensorted,
    queue=QueueExtFIFOQueue))
```

clustering results

Figure 15.5 shows the clustering results for the six considered data sets. In all cases, the procedure yields an intuitive partitioning of the data into clusters. As expected, the identified groups of data points are independent of size and shape and correlate with regions of high data point density separated by low density. For more information on how to choose suitable cluster parameters to obtain these results, see 15.7.

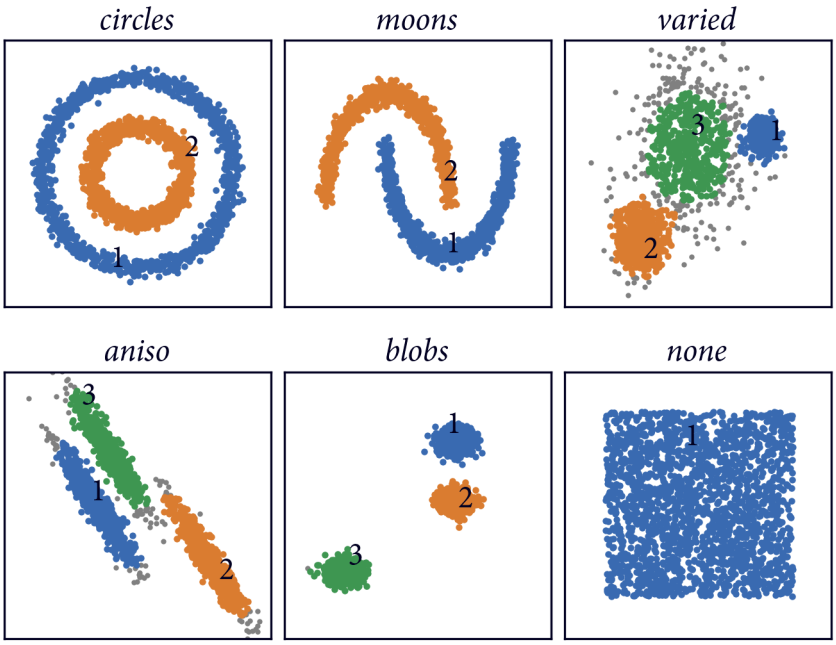


Figure 15.5 Fast CommonNN clustering of scikit-learn toy data sets Independent of the used recipe, CommonNN clustering deterministically yields the same result. For each of the shown example cases, the clustering is aligned with what would be intuitively expected. Clusters are coloured by cluster label (noise in grey).

To get an impression of how quickly clusterings for data sets of the considered size may be done, table 15.6 summarizes the execution times for the standard clustering recipes discussed above. In all cases, the procedure finishes in no time but there is a clear advantage of pre-computing distances or neighbourhoods over the brute-force on-the-fly computation from point coordinates. The best performance is achieved when neighbourhoods are sorted by member indices and by neighbour counts. *timings*

	time / ms	recipe				
		P	D	N	S	S^a
data set	*circles*	172	91	45	4	3
	moons	170	110	63	5	4
	varied	188	139	75	7	5
	aniso	147	84	38	5	4
	blobs	241	187	135	14	11
	none	124	71	30	4	3

Figure 15.6 Recorded toy set clustering execution times For the standard recipes `"points"` (P), `"distances"` (D), `"neighbourhoods"` (N), and `"sorted_neighbourhoods"` without (S) and with points additionally sorted by neighbour count (S^a). Timings where obtained on a machine with an Intel® Xeon® CPU E5-2690 v3 @ 2.60GHz and comprise the execution of `Fitter.fit` without pre- or post-processing.

15.6.1 Similarity check variants

The optimal recipe for CommonNN clustering starting from pre-computed sorted neighbourhoods discussed in the last section depends on a special implementation of the similarity check. So let's focus for a moment on how this check can in general be implemented. Let's assume we have two neighbour containers filled with the indices of neighbouring points for two points that should be subjected to a similarity check. To assess the similarity between two points, we want to get in principle the number of elements (point indices) that can be found in both the neighbours containers.

Such an operation is highly optimised for specific data structures in different programming languages. For instance, NumPy provides it for its own array type (`intersect1d`) and C++ has it for sorted ranges like `std::set` (`std::set_intersection`). In the context of our generic approach and a generic treatment of neighbours containers in terms of `Neighbours` types, we run into problems if

we like to use one of those, though. This is because, these functions explicitly demand certain data structures or respective interfaces. If we want to have similarity checkers that implement them, we would either have to write many different checkers that all require a different type of neighbours container (not very generic any more) or we would have to convert generic containers, which are passed to the checker, internally into the specifically demanded data structure (not very efficient). The only other alternative would be to settle on the one best solution for the check, and to have only one type of neighbours container (again not at all generic any more). So what we need to do instead, is to program the checks against our `Neighbours` interface.

There is another important point why we would want to implement the check without the use of specialised external functions: for threshold-based clustering, we are not actually interested in the exact number of shared neighbours but only in the fact if there are enough shared neighbours. In certain situations, the similarity check can be accelerated by aborting the check once a sufficiently high number of elements is found in the two neighbours containers. By using a function that returns the cardinality of the container intersection we deprive ourselves of this potentially performant option.

Figure 15.7 explains the different approaches to the similarity check that have been so far considered. The first variant is called the 'contains' methods here because it is based on repeated containment checks. Essentially, we loop over the point indices stored in the neighbours container for a point a and then ask the question if this element is contained also in the neighbours container of a point b. For this containment check, there are in turn three prevalent options.

First, there could be another loop over the second container to compare the current element picked from the first container to all the element stored here. This is the worst conceivable solution because it has a runtime complexity of $O(n_a n_b)$, where n_a and n_b are the lengths of the two containers, in the worst case if there are no shared neighbours—for each element in container a there will be a full loop over container b. Only in the case that the first n_c elements in container a are also the first elements in container b (the absolute best case), the runtime is reduced to $O(n_c^2/2)$, where n_c is the specified similarity threshold.

The second option is much more promising but requires the data to be stored in a data structure that allows fast element lookup via a hash function, e.g. a set or map. This is on average assumed to take constant time so that the overall runtime complexity is dominated only be the loop over container a and therefore improved to $O(n_a)$ in the worst case and $O(n_c)$ in the best case. In the default recipes, except for the one that requires sorted neighbourhoods, this approach is realised by the `NeighboursExtVectorCPPUnorderedSet` type that uses a combination of a `std::vector` (fast looping) and a `std::unordered_set` (fast lookup). The matching similarity checker would be `SimilarityCheckerExtContains` or `SimilarityCheckerExtSwitchContains` where the latter always ensures that the looping part is done over the shorter of the two containers.

sorted neighbourhoods

Last but not least, it is possible to follow a binary search scheme here that has a logarithmic runtime complexity and would lead to $O(n_a \log n_b)$ (respectively $O(n_c \log n_b)$ on early breaks) scaling. This requires, however, that the neighbourhoods are sorted, i.e. that the indices stored in the containers are found in strictly ascending order. The binary search approach was so far not implemented because for sorted neighbourhoods there is another much better way. It is called the 'screen' check here because it is based on a simultaneous screen through both containers. It works like this: for each container we initialise a pointer to its first element. The two elements are compared and if there is a match both pointers will be advanced by 1 in forward direction. If the two compared elements are not same, only the pointer pointing to the smaller of the elements is advanced before another comparison takes place. For the worst case of for example alternating and non-overlapping containers the runtime complexity is still $O(n_a + n_b - 1)$, which is not too bad. In the best case, only $O(n_c)$ comparisons are necessary to find n_c matching elements. To realise this approach, it is recommended to use `NeighboursExtVector` containers and the `SimilarityCheckerExtScreensorted` checker.

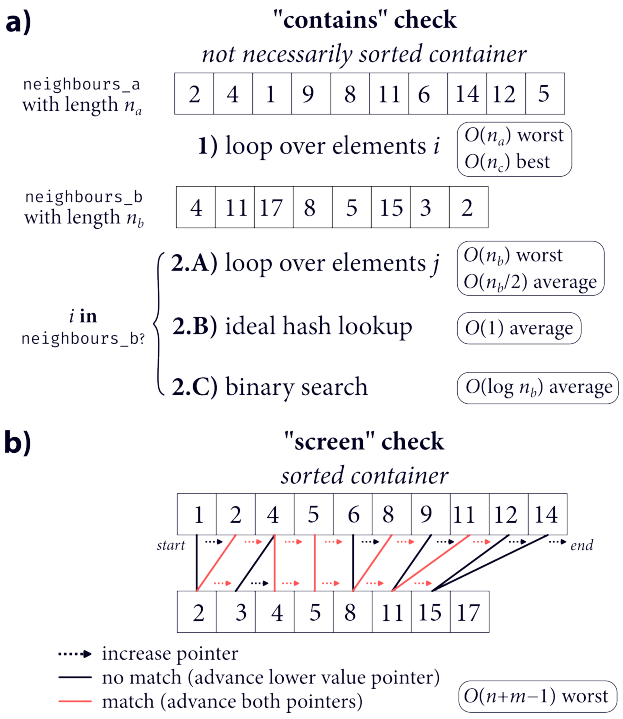


Figure 15.7 Similarity check variants For the assessment of the similarity between two data points a and b based on their respective neighbourhoods, there are two fundamentally different approaches available. **a)** The 'contains' check loops through one neighbourhood and queries from the other neighbourhood whether the current item is also contained in there. Depending on how the containment check is done, this is of varying efficiency. **b)** The 'screen' check runs trough both the neighbourhoods at the same time and evaluates their items in a linear chain of comparisons. This requires neighbourhoods sorted by member indices.

15.7 Parameter selection

Threshold-based CommonNN clustering depends on the two cluster parameters r (neighbour search radius) and n_c (similarity cut-off). The result of the clustering depends sensitively on the values that are selected here. Before the background that the two parameters represent a density-estimate, it becomes clear that the tuning of r versus n_c can be considered antipodal. The larger we set n_c and the smaller we set r, the higher is the requirement on the density estimate for two checked points to be connected. That means that an increase of n_c simultaneously with r may result in essentially the same effective threshold and qualitatively identical results for technically different parameter combinations.

To choose appropriate values of r and n_c is fundamentally a difficult task. The in the first instance rigorous approach to it would entail essentially a 2-dimensional parameter scan over a broad value range and a selection of the best result from the set of all possible results. This can be, however, problematic for multiple reasons. For one thing, detailed scans using a fine enough grid of parameter combinations are relatively expensive especially for big data sets. On the other hand, a validation of the cluster results to select the best one may not be straightforward. It is possible to categorise the results in terms of statistical quantities like the number of obtained clusters, the points assigned to the biggest cluster, or the number of noise points but there is no guarantee that any of these metrics actually connects to the quality of the clustering. There is in general no reliable basis for the preference of a certain number of clusters for example. Typically used validation methods for clusterings like for example shown in section 14.3 are not very applicable for density-based clusterings either because the obtained clusters may have arbitrary shapes and sizes and their quality is hard to measure on an internal scale. Eventually, clustering results may need to be validated manually, that is visually or otherwise inspected by an expert. Inspection of a vast number of results from an extensive parameter scan may be tedious and error-prone, though. Ideally, an indirect method of validation would be used but this is closely tied to specific fields of application. In case of MD data, one option might be

2D scans

the construction of Markov-models where the best model identifies the best clustering. This may, however, only shift the problem to the question of what the best Markov-model would be, which is also not trivial.

stable parameter regions

That said, a parameter scan is still the best chance to find good cluster parameters. A principle that may help in the identification of reasonable parameter regions, is to look for those in which the cluster label assignments stay qualitatively the same. This reasoning is similar to the live-time argument made by HDBSCAN to select clustering results. Cluster assignments that can only be produced in a very narrow parameter region may likely be meaningless. Again, there is no guarantee, though, that the best result is also the most stable one.

pseudo-hierarchies

We should also recall, that a systematic variation of the cluster parameters, i.e. a screen of the density threshold, is in essence nothing else than a pseudo-hierarchical approach. Starting with a low similarity threshold, most points will be connected by a sufficiently high density threshold. Likely, all points will be part of the same one cluster (or of a low number of very well separated clusters). When we increase the threshold, the points will be split into clusters as more and more points in low density regions fall below the threshold and carve out disconnected high density regions. Typically, the present clusters will at first only shrink (i.e. split off noise points) before larger parts of them become disconnected. As the data point density can vary drastically within a data set, it may also happen that lower density clusters vanish completely into noise before other higher density clusters are split into sub-clusters. Therefore it should be kept in mind, that the best clustering result may not be achievable at all by applying a single similarity threshold. What we want to find ideally, are those threshold values where clusters split and there might be more than one of them. Before one puts too much energy into the search for an optimal parameter combination on a vast grid, the possibility should be considered that the one combination does not exist there.

basic plateau strategy

Therefore, a well-tried approach to tackle the problem of parameter optimisation, is to do the following. Starting with a low density threshold, one does increase the threshold continuously and records the number of obtained clusters. With higher thresholds, the cluster count is expected to go up. One does proceed with increasing the threshold up to the point where the cluster number goes down again. This might be a good point to stop because starting from here we loose low density clusters into noise. One may want to combine this with the notion of looking for stable parameter regions and specifically try to find the last plateau region before the cluster count decreases. This approach levels the field for manual hierarchical clustering, which will be described in the next section 15.8. Let's record for our following considerations that a cluster parameter optimisation is only really meaningful in the sense of finding the most far-reaching partition on the current hierarchy level of clusterings before information is lost when clusters vanish into noise.

resolution parameter

As a further strategy, it is possible to consider the tuning of r and n_c separately instead of a complex 2D parameter scan. In the selection of r, one should in general aim for a low values. The neighbour search radius can be understood as a resolution parameter for the density estimate. We want this resolution to be as fine as possible to get a *sufficiently local* estimate of the density. Large values for r may not be able to resolve the differences in the density that lead to a separation of clusters. Also in terms of the efficiency of the clustering, low values for r are preferable because this keeps the neighbourhoods of individual data points small. On the other hand, r can not be too small because depending on the sampling of the data set, the density estimate may become noisy. Moreover, the sensitivity of the density estimate towards a variation in n_c becomes higher for small r—one additional shared neighbour makes a bigger difference when the neighbourhood radius is small. A time-proven heuristic to select a good first guess for r based on the relative positions of data points is to plot a histogram of pairwise inter-point distances. Figure 15.8 exemplifies this for the toy data sets clustered in section 15.6. The first peak in each of the shown distributions can give an orientation for what can be considered local for the respective data set. Additionally, the better the given sampling situation,

distance distributions

the smaller we can set r. So while for scarce data a good radius might be found on the right side of the peak, it might be found on left side for dense data.

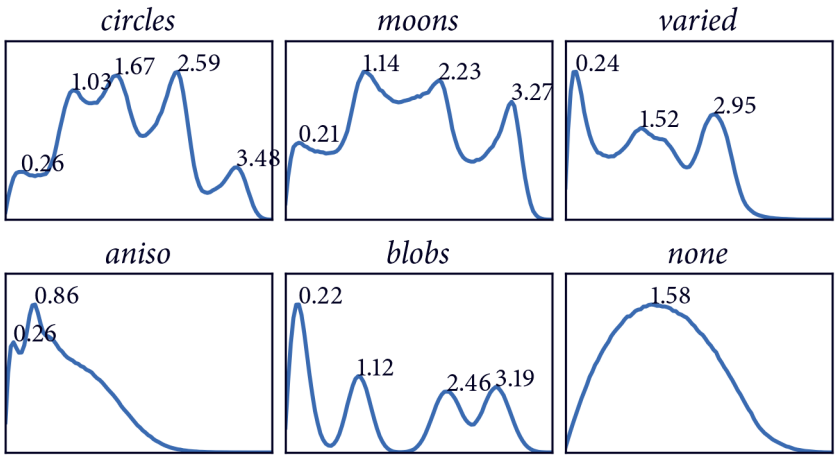


Figure 15.8 Distance histograms for scikit-learn toy data sets The distribution of pairwise distances can give on orientation for a good first guess on a suitable value for the cluster parameter r. The first maximum represents a distance that can be considered sufficiently local in terms of the specific data set.

One thing should be pointed out in connection with theses distance distributions. Although it might be tempting, it is in general not possible to deduce the optimal number of clusters that can be found in a data set from these plots. It is true that a globular cluster gives rise to one single distribution maximum. Further, there should be one additional peak for each pair of globular clusters. The total number of visible peaks may, however, be lower due to overlaps so that the plots are not a reliable way to predict how many clusters there should be. For non-globular clusters that may themselves give already rise to multiple peaks the attempt is accordingly hopeless.

distribution interpretation

After settling on a rough value for r, the cluster parameter scan can essentially be focused on n_c. This has the pleasant side effect that only the integer value n_c needs to be screened and not the floating point valued radius. Starting with a low value for the similarity cut-off (e.g. $n_c = 0$), the value can now be increased in (small) increments for successive clusterings until the desired result is found or until a plateau region is reached as described above from which an further increase of the value leads to a loss of clusters.

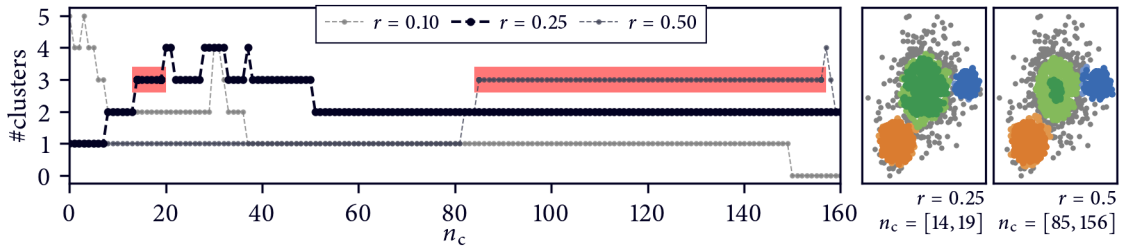


Figure 15.9 Parameter scans with fixed r for the *varied* data set
A general strategy to find suitable cluster parameters is to select a rough value for r (e.g. judging the distribution of pairwise distances) and to increase n_c until a plateau region with respect to the number of identified clusters is found after which the cluster number decreases again. For the given example, the regions selected according to this approach are highlighted in the scan with red rectangles. The corresponding cluster label assignments are shown on the right. While reasonable results are obtained with the larger radii, $r = 0.1$ is definitely too small because no traceable increase in the cluster number can be detected before the number decreases. All clusterings use `member_cutoff=20`.

Figure 15.9 illustrates this approach for one of the previously used toy data sets. The data set has a relatively low number of points so the radius is supposed to rather be larger then what is suggested by the distance distribution. The cluster results within each of the identified plateau regions are qualitatively the same—only the cluster size varies. From the standpoint of cluster quality, all of them are basically equivalent and the user has to decide in which state a particular cluster should be

obtained. It would be possible to either select the largest state (the situation right at the first threshold where a cluster emerges caused by a split) or the smallest state (the situation right before a cluster is further split). If there is a method of validation, any state in between may also be identified as the most appropriate.

15.8 Manual hierarchical clustering

We stated in the last section that an optimisation of cluster parameters can be only really valid in terms of the same hierarchy level. To indeed systematically optimise a clustering result one actually needs to do the following. Starting from a low value, n_c is increased until no further splits are achieved without loosing clusters or, more concisely, even only until the first split occurs. Then the procedure needs to be independently continued on the data sub-sets that were separated by the split, in other words a further clustering of the obtained clusters into sub-clusters has to be done. In this way, the hierarchical tree of clusterings is explored branch by branch. Doing this by the means of the provided threshold-based clustering functionality will be referred here to as *manual* hierarchical clustering.

bundle concept

The CommonNNClustering project offers the tools to support this approach. First of all, to be able to represent clustering hierarchies, we use the concept of a `Bundle`. Such a bundle aggregates for example a data set, a labels container, potentially a graph structure for the data, some meta information, and a summary of all previously done clustering attempts. It also can be associated to other bundles in a child-parent relationship. `Bundle._children` is a mapping of cluster labels to other child bundles. `Bundle._parent` is a weak reference to the parent bundle. Standard threshold-based clustering operates on one bundle at the time. The later described hierarchical clustering approaches can also operate on a hierarchy of bundles. A `Clustering` is not directly associated to a specific input data set but to a root bundle `Clustering._bundle`. One clustering object can therefore handle single bundles but also full hierarchies.

isolate & reel

For manually orchestrated hierarchical clustering this means that we start by clustering the root bundle. Once a satisfying split has been achieved, a child bundle is created for each obtained cluster. This step will be referred to the *isolation* of a clustering result. Then the clustering can be continued for each child bundle individually. In the end, cluster results from lower hierarchy levels can be integrated back into the root bundle as desired. This will be figuratively described as to *reel* in clustering results. These two steps are realised by respective methods of the `Bundle` class, which are also callable via `Clustering.isolate` and `Clustering.reel`. During isolation, child bundles are by default equipped with a subset of the input data that is relevant to it. To reel in information from child bundles, `ReferencIndices` stored on child bundles are used to map cluster label assignments for data sub-sets back to the parent or the root bundle.

Let's have a look at how the isolate-reel approach works in practice with a MD data set for alanine dipeptide (a 2-dimensional projection on backbone torsion angles of 10020 data points). In principle, we have to use the following pattern:

```
# Root data
clustering = Clustering(data)
clustering.fit(**params)  # Find first split
clustering.isolate()
# First child
b1 = clustering.get_child(1)  # Same as clustering._bundle._children[1]
clustering.fit(**params, bundle=b1) # Find next split
b1.isolate()  # To proceed to deeper levels
# Second child
```

```
...
# Once satisfied with the hierarchy
clustering.reel()
```

Figure 15.10 illustrates the result of this procedure. A proper clustering of the shown data points requires a hierarchical approach because lower density clusters will vanish at threshold values needed to split clusters in the higher density regions. Applying the isolate-reel formalism, the root data set is partitioned into three clusters. The largest cluster is split further into two cluster of which the larger one is again split. The final result is a combination of splits at different threshold values. For each hierarchy level, the first threshold at which a split is observed is chosen as the isolation point.

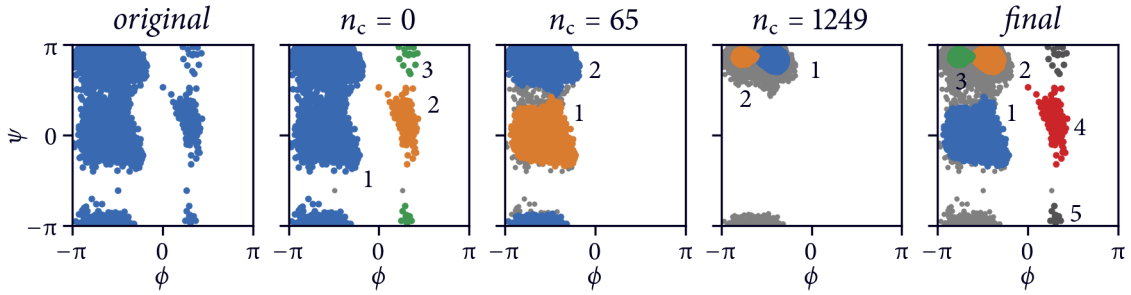


Figure 15.10 Manual hierarchical clustering of an *alanine dipeptide* data set
Cluster label assignments for clusterings at different hierarchy levels. All clusterings use a neighbour search radius of $r = 0.4$, which was derived from the respective pairwise distance distribution. On the first level, the member cut-off was set to 10 wich was then increased to 100 at deeper hierarchy levels. Note, that the data set is periodic in ϕ and ψ and requires a corresponding metric or a transformation to sine and cosine components, which was done here.

Cluster hierarchies represented by bundles associated with each other can be visualised as a tree by using `Clustering.tree`. Figure 15.11 shows the tree for the manual hierarchical clustering of the *alanine* data.

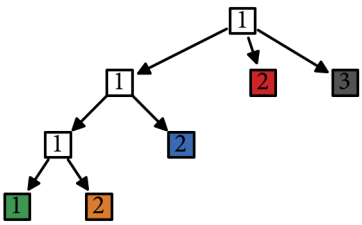


Figure 15.11 Tree of clustering results for the *alanine* data set The root bundle at the top has the label 1 because without applied clustering there will be only one cluster. In the first stage, the data is split into three clusters of which 2 and 3 constitute cluster 4 and 5 in the final result (indicated by colour). The obtained cluster 1 is split into two further clusters in the next stage where cluster 2 is the later cluster 1. In the last stage, obtained cluster 1 splits into cluster 1 and 2 that are the later clusters 3 and 2.

15.9 Semi-automatic hierarchical clustering

Manual hierarchical clustering as addressed in the last section offers full control over splits on different hierarchy levels. For large data sets that may contain many clusters appearing at varying threshold values, it can be, however, problematic. Each split does essentially require a separate parameter scan to find the next split or at least careful rationalisation of what to choose as the next threshold value. An alternative is offered by an approach that should be called *semi-automatic* hierarchical clustering here.

This approach is still pseudo-hierarchical but instead of forcing the user to orchestrate each hierarchy split manually, it only requires the specification of a range of cluster parameters. In essence, it works like a scan along this range while constructing the hierarchy of clustering result without further

user interaction. Because such a hierarchical clustering needs access to the cluster hierarchy, we introduced another generic fitter type with a slightly different interface: the `HierarchicalFitter`. Semi-automatic clustering is currently realised by the `HierarchicalFitterRepeat` class. It is called `Repeat` because it repeatedly uses a standard threshold-based `Fitter` to get the next clustering. After each clustering execution, the labels of the current run are compared with those of the previous one. Each bundle from the last step will then be extended by the children that appeared in the current step. The result is a raw hierarchy with as many levels as parameter combinations specified. This hierarchy can than be processed in multiple ways to be condensed and to find the clusters that are most relevant.

To exemplify the semi-automatic hierarchical clustering strategy, let's have a look at another data set: 37500 data points from a MD simulation of a small helical peptide (PDB-ID 6a5j) projected onto 4 time-lagged independent components using backbone and χ_1-, and χ_2 side chain torsion angle features as input. This data set is much more complex than those considered before, revealing a fairly large number of clusters. It would probably be very time consuming to find all the clusters manually but the semi-automatic approach can help with that. Hierarchical clustering from the user perspective is not much different to pure threshold-based clustering. The only thing we have to do is to equip a `Clustering` instance with a `HierarchicalFitter` and to call `Clustering.fit_hierarchical`. `HierarchicalFitterRepeat` excepts iterable collections of radii r and thresholds n_c. Here, we want to keep our previous approach of fixing r at a reasonable value and then control the density criterion with increasing values for n_c.

Figure 15.12 shows the considered data set in its original projection and the clustering result. The raw hierarchical tree was trimmed by removing child clusters that represent a shrinking of the respective clusters without splitting using `Clustering.trim_shrinking_leafs`. Before the final result was reeled back in onto the root bundle, `Clustering.trim_trivial_leafs` additionally ensured that there were no all-noise leafs in the tree. In the end, we get a 7-step hierarchical result (tree not shown) where the first split of the root cluster into 9 sub-clusters happens at $n_c = 0$ before there are multiple further binary splits at higher threshold values.

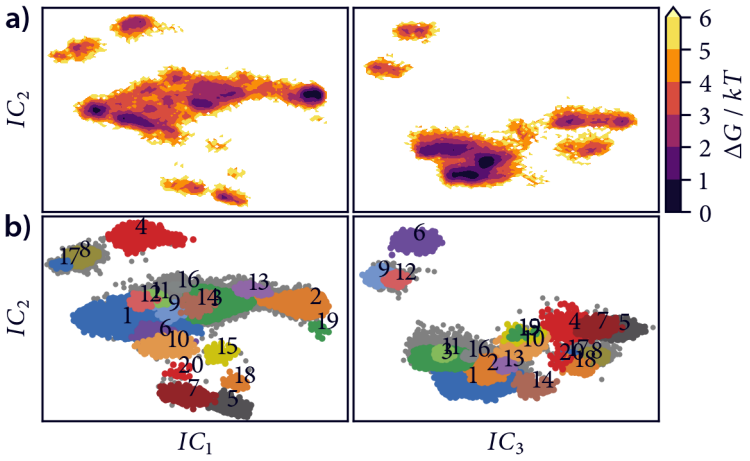


Figure 15.12 Semi-automatic hierarchical clustering of the *helix* data set a) 4-dimensional projection as pseudo free-energy surfaces. They contain multiple minima each of which corresponds to a cluster that might be desirable to find. **b)** Semi-automatic clustering with $r = 0.3$ and $n_c = [0, 600]$ (`member_cutoff=20`) presumably exposed most of them. Data points coloured by cluster label assignment (noise in grey).

As figure 15.13 confirms, the achieved separation of the data into clusters was quite successful. Most of the identified conformational states are structurally well defined. Whether this result is indeed satisfactory has to be evaluated, though, before the background of a possible application.

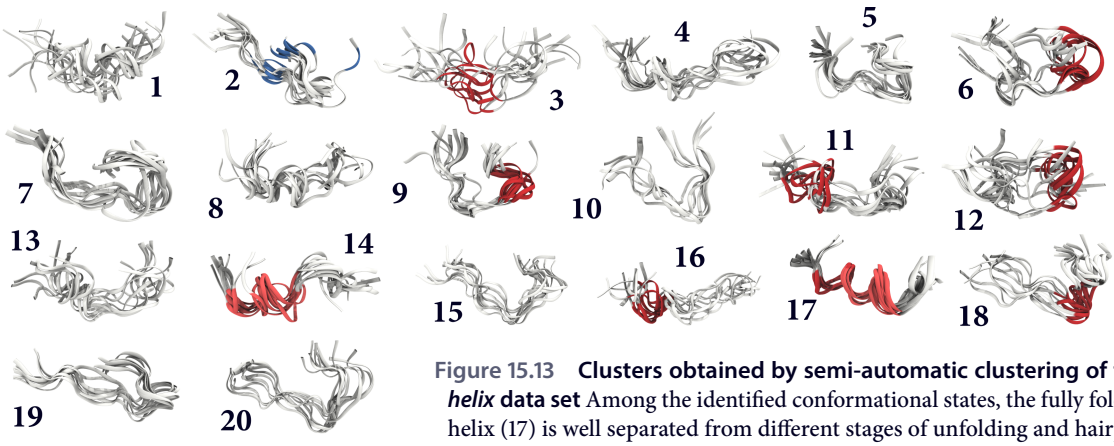


Figure 15.13 Clusters obtained by semi-automatic clustering of the *helix* data set Among the identified conformational states, the fully folded helix (17) is well separated from different stages of unfolding and hairpin-like structures (2).

Semi-automatic hierarchical clustering depends sensitively on the parameter range that is specified as the input. In principle, the grid of parameter values that is scanned here should be as fine as possible so that no threshold value that generates a split is being missed. For practical reasons, this might, however, not be possible because frequent execution of the clustering may be too costly. In this case, the accuracy of the approach has to be balanced against the computation time that should be afforded. The presented example using 601 similarity cut-off values finished in about 20 minutes but for larger data sets one may need to reduce the number of grid points or except a considerably larger time investment.

15.10 Hierarchical clustering using minimum spanning trees

The previous sections dealt with the application of threshold-based CommonNN clustering in a pseudo-hierarchical fashion. By repetition of the clustering at different thresholds, an approximate view on the underlying cluster hierarchy can be obtained, either fully rationally guided level after level (the manual approach) or more conveniently and less influenced by user decisions on a pre-defined parameter grid (the semi-automatic approach). One could ask the question now, why threshold-based clustering is used in the first place if the result we are interested in is often a hierarchy, and rightfully so.

As has been discussed in the methods chapter 14, density-based clustering is *intrinsically* hierarchical. By the choice of a threshold, which means for CommonNN clustering to truncate the pairwise density estimate and to convert it into a binary similarity measure, we simplify the clustering result and prefer a single slice of the hierarchy that is actually present. Why this is so commonly done may be due to historical reasons, because it is conceptually clean, easy to grasp, and fast, or because for a broad range of applications it is just good enough. To take the threshold-based picture and turn it back into a hierarchical formalism is, however, arguably fairly counter-intuitive. There is actually no need to convert a quantitative density and connectivity estimate as we have it for CommonNN clustering (see equation 15.1) into a cut-off criterion (see equation 15.2), so that the hierarchy can be screened by repeated searches for connected components at each level. The full hierarchy can already be directly learned from the un-truncated connectivity information. How to do it, was introduced in the last chapter in terms of single-linkage clustering using the pairwise density estimate as a quantitative similarity measure.

Practically, we want to construct a minimum spanning tree (MST) containing only the most relevant connections between the data points. Then we review the MST edges in the order of their weight (from

low to high, where a low weight equates to a high similarity). Each edge will produce a new merge of two sub-clusters (initially single data points) and we will end up with a binary tree representing the full single-linkage hierarchy of clusters. The advantages of this over pseudo-hierarchical threshold-based approaches would be that it is expected to be both more efficient and more concise. A single search for connected components may be faster than the construction of a MST plus single-linkage clustering but while the latter approach yields the full hierarchy in one go, it may require a lot (maybe thousands) of connected component searches to achieve the same result.

Let's assume we estimate a graph for a data set containing edges in accordance with the CommonNN similarity. Based on a certain neighbour search radius r, we end up with a graph of m edges (one for each neighbour pair) and n nodes (one for each data point). Neglecting the fact that in the threshold-based picture the number of explicitly considered edges may be lower (non-connecting edges can be removed), the graph will be of the same size no matter whether the density-estimate is truncated or kept as a continuous quantity. Also neglecting the fact that similarity checks with a threshold can be quicker due to early breaks once the similarity criterion is fulfilled, the construction of the graph takes about the same time for both cases. The classic BFS algorithm to find connected components in the graph has a runtime complexity of $\mathcal{O}(m+n)$. A standard approach for MST construction is Prim's algorithm that scales with $\mathcal{O}(m \log n)$. We have to add a linear $O(n-1)$ to review the $n-1$ edges of the MST. Figure 15.14 shows a comparison of the theoretically expected runtime scaling for different values of n and m. The atomic operations of both algorithms are virtually the same, namely a processing of the graph's edges, so we can take the scaling as a rough approximation to expected absolute run times. In reality, the execution prefactor for the MST algorithm might be higher, though, because edges are sorted on the way to be processed in order of their weight. Still, for the largest shown value of $n = 1,000,000$, the MST approach has diverged from the BFS scaling by only a factor of 13 and will quickly be amortised when multiple repetitions of the latter are necessary. The MST construction can furthermore be achieved in even more efficient ways by choosing a different algorithm.

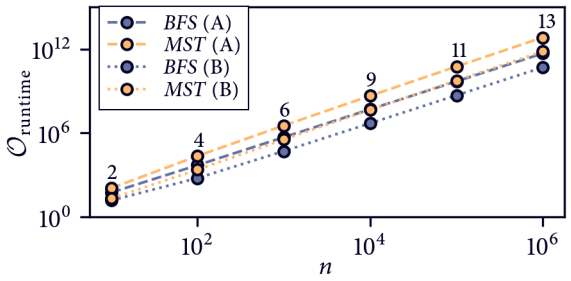


Figure 15.14 Runtime expectation of BFS vs. MST On an input graph of n nodes and m edges, compared is a connected components search ($\mathcal{O}(m+n)$) to MST construction (Prim) plus single-linkage clustering ($\mathcal{O}(m \log n + (n-1))$). The number of edges was set to either $m = (n^2-n)/2$ (A) or $m = 0.05n^2$ (B). The runtime ratio MST/BFS (marker labels) is the same for (A) and (B).

I also like to stress again that the fact that we obtain the full hierarchy of clusters in the single-linkage approach ensures that we can not 'miss' any clusters—with the limitation of how the MST is constructed of course. A threshold-based parameter scan, however, can only ensure this if a sufficiently wide range and a fine a enough grid of thresholds is used while it is not known in advance what will be the optimal choice in this regard. It should be also acknowledged, that the MST approach eliminates the cut-off parameter n_c (or rather does no introduce it). Therefore a tuning of this value will not be necessary at all.

For the MST construction taking the quantitative density-estimate (equation 15.1) saying '*how many neighbours are shared by two points with respect to the neighbour search radius?*', we have to choose a value for the cluster parameter r. Depending on this value, the MST we obtain can be a different one. In principle, the same considerations as for threshold-based clustering apply for how to settle on a value for r here. It has to be sufficiently small to allow a local density estimate but must not be too small so that the estimate becomes noisy. It is still an open question, how sensitive the quality of

the MST and consequently the clustering really is with respect to r. For a given radius, we can then determine the pairwise density estimate for pairs of points. Here, we have again the same basic choices like before in threshold-based clustering in terms of what are suitable candidates for the similarity check (compare figure 15.4). So far we stayed with the former assumption that only neighbouring pairs will be assessed.

For more details on how MSTs are found using Prim's algorithm see section 7.2 on graph theory. The following code snippet shows a scheme for a generic implementation as it is incorporated in `HierarchicalFitterMSTPrim`. Like the implicit BFS for threshold-based clustering, this does not actually construct a physical graph but only tries to find the corresponding set of edges to be directly processed further. Note the appearance of another generic type providing a data structure to which edges can be added and retrieved back in order of their weight: the `PriorityQueue` type. Such a queue will be both used to stored candidate edges (`prio_queue`) as well the final edges of the MST (`prio_queue_mst`). Also note the use of `SimilarityChecker.get` instead of `check` to get the actual number of shared neighbours not only an assessment of whether there are enough of them.

```
n = input_data.n_points
for point in range(n):
  ...
  neighbours_getter.get(...) # Get neighbours of point
  for member in neighbours:
    ...
    neighbours_getter.get(...) # Get neighbours of member
    weight = similarity_checker.get(...)
    prio_queue.push(point, member, weight)
  while not prio_queue.is_empty():
    a, b, weight = prio_queue.pop()
    ...
    prio_queue_mst.push(a, b, weight)  # New MST edge!
    neighbours_getter.get(...) # Get neighbours of b
    for member in neighbours:
      ...
      neighbours_getter.get(...)
      weight = similarity_checker.get(...)
      prio_queue.push(b, member, weight)
```

Once we have the MST edges, we can build the single-linkage hierarchy. There are multiple options for how to store and present this hierarchy. One would for example be to use the SciPy format described in section 14.1, which can be analysed directly or further condensed.[444] Here, however, we want to convert the binary hierarchy into a hierarchy of bundles where one-to-many parent-child relations are allowed. Furthermore, we want to apply a member cut-off to condense the hierarchy such that only those merges appear as relevant where large enough clusters are combined. In this hierarchy building step, we actually make use of explicit graphs for each bundle.[13] The whole process is a bit involved and not particularly well optimised at the moment, so instead of presenting it here the interested reader is invited to check the source code for `HierarchicalFitterMSTPrim.fit` directly for more information.

Figure 15.15 illustrate such a tree as obtained for the *helix* data set. The result is similar to what we had before with the semi-automatic approach (compare 15.12 and 15.13) but this time we can be sure that we found every possible cluster there is for the given radius cut-off r and after trimming using the member cut-off. Execution of this example took about 30 minutes to finish. This is comparable to the

[13] For now, the Python third party Networkx package is used for that. A generic `Graph` type may be an option for future improvements. Maybe even a generic `HierarchyBuilder` type might be in order to be able to exchange the way how the hierarchy is made and to decouple it from the MST building step.

semi-automatic approach shown before, but it should be kept in mind that while the threshold-based clustering procedure is fairly optimised already, the MST hierarchical variant should be considered a proof-of-concept implementation at the moment.

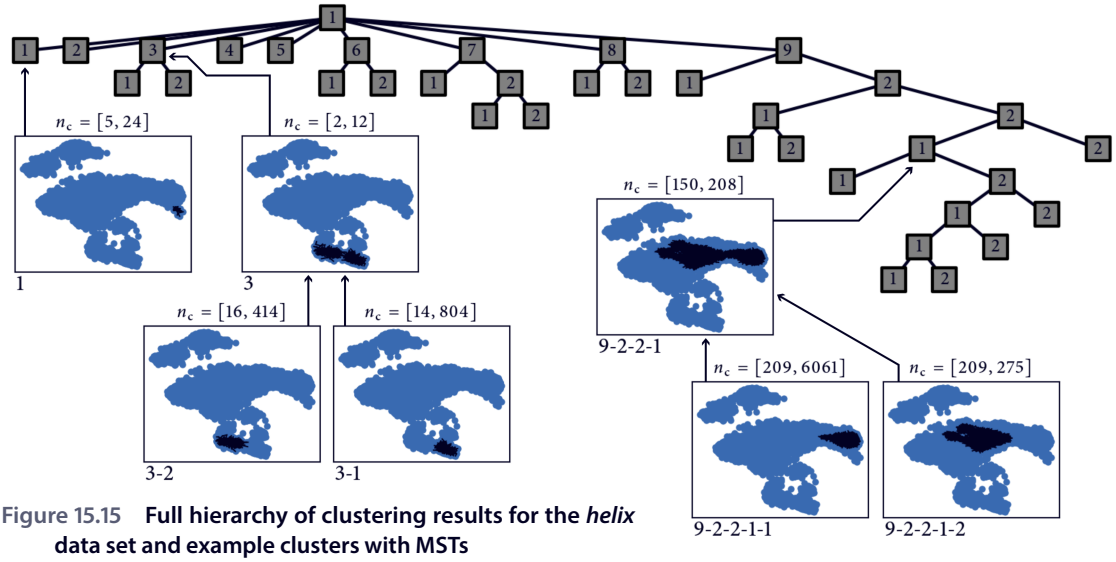


Figure 15.15 Full hierarchy of clustering results for the *helix* data set and example clusters with MSTs
The shown tree represents an 8-stage hierarchical clustering. When the MSTs were constructed, the 9 clusters on the first level ended up as disjoint, which corresponds to a threshold-based clustering with $n_c = 0$. For each split in the hierarchy, one can compare a parent to its children to decide which partitioning should be kept. Since each bundle in the hierarchy carries a weighted graph, one can furthermore fine tune clusters in the n_c range they exist. For a few example bundles, their tree is shown and the minimum/maximum edge weight is annotated above the subplots. Note, that a cluster appears in threshold-based clustering when the maximum edge weight of its parent is exceeded but its own minimum weight can be higher.

Besides the possibility to use the quantitative CommonNN density estimate at a fixed radius r as an edge weight for the construction of MSTs, there is also a second way to derive a continuous similarity from the CommonNN notion of density. This would be more similar to what is done for example in the HDBSCAN scheme (see section 14.7) under the concept of *mutual reachability distance*.[450] For this we would turn our density estimate upside down and ask the question '*How large does r need to be for two points to share a certain number of neighbours?*'. While the previously discussed approach eliminated the cluster parameter n_c for the clustering procedure, this alternative formulation eliminates the neighbour search radius r and requires us to only set n_c. Figure 15.16 illustrates this idea.

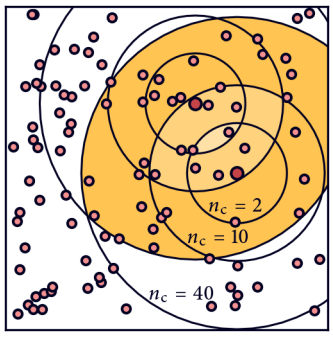


Figure 15.16 CommonNN mutual reachability distance Given two points a and b, how large does r have to be so that a and b share n_c of their neighbours? Optionally, mutual reachability can be defined as $\max(r_{\text{reach}}(a, b; n_c), d(a, b))$. The radii were found using the code snippet below.

Algorithmically, the determination of the required radius $r_{\text{reach}}(a, b; n_c)$ is a bit more complicated than a plain evaluation of neighbourhood intersections. It can for example be realised, however, by

the following approach: maintain for each of the two checked points a mapping that allows quick membership lookups of point indices and associated distances. Then do a series of alternating kth nearest neighbour queries for the two points and check after each query if the next nearest neighbour of point a was already seen as a k-nearest neighbour of point b. Track the number of neighbours that are seen in both neighbourhoods and return the distance in question once the given threshold is reached.

```
"""At which distance do point a and b share nc neighbours"""
na_distance_map = {}  # Keep track of neighbours and distances
nb_distance_map = {}
common = 0
ka_count = kb_count = 1  # Current kth neighbour for a and b
ka_index, ka_distance = get_kth_neighbour(a, ka_count)
kb_index, kb_distance = get_kth_neighbour(b, kb_count)
while True:
  if ka_distance <= kb_distance:
    if ka_index in nb_distance_map:
      common += 1
      if common >= c: return max(ka_distance, nb_distance_map[ka_index])
    na_distance_map[ka_index] = ka_distance
    ka_count += 1
    ka_index, ka_distance = get_kth_neighbour(a, ka_count)
  else:
    if kb_index in na_distance_map:
      common += 1
      if common >= c: return max(kb_distance, na_distance_map[kb_index])
    nb_distance_map[kb_index] = kb_distance
    kb_count += 1
    kb_index, kb_distance = get_kth_neighbour(a, ka_count)
```

The approach does heavily really on efficient logic in `get_kth_neighbour` to retrieve the kth nearest neighbours of the checked point in order. Practically, one may perhaps want to limit the number of k or to set a maximum distance beyond which two points a and b are not checked any more. It should also be noted that the case $n_c = 0$ becomes meaningless here because any point pair trivially fulfils this for $r_{reach} = 0$.

CommonNN clustering using MSTs weighted by mutual reachability distances is a rather theoretical possibility at the moment and there is no hierarchical fitter type in our package supporting this approach yet. It will be interesting to see if there are fundamental differences in the robustness of the two different hierarchical approaches with respect to the selection of the remaining cluster parameter r or c. Qualitatively, both approaches should still yield essentially the same results, though.

{ 16 }

Benchmarking clustering algorithms

Execution time measurements along multiple axes of variation

For the broad applicability of a clustering procedure it is not only important that it is conceptually sound and capable of yielding qualitatively good results, it does also need to be reasonably performant. How fast a useful clustering method needs to be, depends on the value it provides and on the relative performance of other similar methods. Good performance can be understood in basically two ways: in terms of the *absolute* runtime requirement (‘*How many seconds, minutes, or hours does the procedure need to complete?*’) and in terms of the runtime *scaling* (‘*How much longer does the procedure need to complete if the problem size is twice as large?*’). In general, one would ideally have satisfied both, a cheap base case and friendly linear, logarithmic, or even constant scaling. The scaling of a method becomes especially important for larger data sets. Absolute runtimes are nice for a basic orientation.

absolute runtime vs. scaling

For CommonNN clustering, in particular the threshold-based approach, quick execution is very important because in typical data exploration use-cases, the method is probably applied interactively with a large number of different cluster parameter combinations. On moderately sized data sets of 10^3 to 10^5 points, execution times longer than a few seconds will strongly hamper the practical usefulness for many workflows. From the hierarchical perspective, longer execution times are probably less of a problem because as long as a method reliably exposes the full hierarchy of clusters without much tuning and repetitions, users might except runtimes of multiple hours (e.g. a running over night). Here it becomes very important that the approach does not scale too badly to be also useful for an application to large data sets of say $\geq 10^6$ data points.

To argue about the speed and scaling of a clustering procedures, we have to measure it in the first place. Theoretical scaling analyses are very valuable and I tried to comment on expected runtime complexities in the CommonNN context whenever possible. However, it is not always easy to assess the scaling precisely here or while it can be stated for individual algorithmic aspects (for example for the BFS in general), it is sometimes not very clear how the parts add up to the full picture. To get a realistic impression of the efficiency of a complex procedure, eventually we need to run it and just see how long it takes to finish. Only that this is easier said than done: benchmarking clustering algorithms (representatively) is hard.

Mainly it is hard because the performance of a clustering procedure (that is often more than just one fundamental algorithm) can depend on a lot of factors. The biggest difference is probably made by the kind of input data that we provide. Starting from point coordinates, which necessitates (repeated) distance calculations, is likely more expensive than reading off pre-computed neighbourhoods. But how much more expensive? Will it only effect the absolute runtime or does a circumvention of brute-force distance calculations also improve the scaling? Does it make a big difference if we use one storage format over another, e.g. a memoryview instead of a NumPy array? It will probably also

varying the procedure

have a great impact if we change how the similarity check is executed. The CommonNNClustering project is very flexible in the combination of building blocks it allows. Only by measuring the impact of a certain modification we can identify possible weaknesses in our current recipes and see which difference is made by a potential optimisation.

varying the data

Apart from the fact that we can change how the clustering works and what it generally uses as the input format, we can also change the character of the data set itself. For density-based clustering it may make a substantial difference how the points of the data set are distributed. To assess the scaling of the method, we normally also need to measure clustering execution times for the 'same' set but with an increasing number of data points. Similarly, we may need to scale the number of dimensions. Last but not least, the efficiency of the clustering does heavily depend on the cluster parameters. For large neighbour search radii, neighbourhoods contain more members, which may increase the time needed to process them. Also the similarity threshold may slow down the clustering because for large numbers of shared neighbours the program has to do more exhaustive comparisons of the neighbourhoods. On the other hand, for even larger neighbour cut-offs it is expected that the clustering is trivially fast because only few neighbourhoods may be able to fulfil the criterion in the first place.

varying the parameters

With the involvement of the cluster parameters we touch upon a very sensitive aspect of the benchmark: would it be a fair comparison to use any (the same) cluster parameter combination for two different data sets—no matter whether the cluster result is actually meaningful in both cases? Should we instead compare the best parameters (whatever that might be) for each data set with each other? When scaling the number of points a data set consists of, should we adjust the parameters to maintain the cluster result or would it distort the impression we get from the scaling?

method comparison

This last and the other aspects may become even more critical if we compare different clustering procedures. Is it fairest to try two methods against each other with any, the best, the most similar, or the default parameters? Are the methods even entirely comparable? Are they not using a completely different input source? When it comes to the comparison of different methods, a fixation of all other variations so that both are tried under the exact same conditions may not always be possible.

best vs. average vs. worst result

As a short note on how to interpret obtained timings (a list of individual results from independent repetitions), the basic question is whether to report the *best*, *worst*, or *average* result? Generally, it is recommended to use the best one: this is the fastest the machine could do. Considering that the execution times under otherwise identical circumstances can vary due to various factors on the machine level—pre-dominantly other processes running simultaneously and blocking available resources—the best of several timings is normally the 'purest' one, meaning the one that is least influenced by peripheral negative effects. For the benchmarking of clustering procedures, we will adhere to that practice. Average timings may, however, be of interest if the runtime of a procedure for a given problem setup is expected to be not deterministic. Does the result contain an element of randomness? Does this influence the performance? In this case, a consideration of only the best timing may give the wrong impression on how fast the procedure is overall. Beyond that, a consideration of average or even worst timings is often only useful if the stability of a procedure should be benchmarked under 'real life' condition. They can also be specifically used to spot issues with the system on which a procedure is executed.

varying the machine

Before the background of best timings being of general interest on how to asses how fast a clustering procedure can ideally be, it is recommended to choose a benchmark context that minimises any kind of negatively influencing effects. A dedicated run on a HPC cluster for example, where an execution with specifically allocated resources in the absence of other competing processes is possible, may be universally a good idea. Besides that, a comparison of absolute runtimes on different typical machines (a standard workstation, server, or notebook) may also be interesting. It should be kept in

mind, though, that the benchmark might be influenced by whatever runs in the background when the timings are measured.

When execution times t for a series of variation along a specific quantity (e.g. different problem sizes n) are collected, one can determine an empirical growth factor b by fitting the measurements to the power function

$$t \approx an^b \,. \tag{16.1}$$

On a log-log-plot, b is the slope of the resulting line if the scaling is stable over the tested parameter range. A factor of $b = 2$ or $b = 1$ correspond for example to empirical quadratic or linear scaling. Increasing the problem size by a factor of 10, quadratically scaling applications will take 100 times longer to complete while linearly scaling ones only take 10 times longer. The pre-factor a is just a proportionality constant we do not need to put much attention to. For applications with a larger overhead, which means that for example trivial execution for $n = 0$ already takes up a substantial amount of, a fitting to the extended power function

$$t \approx an^b + c \tag{16.2}$$

can be an option, where c offers an estimate on the baseline for the execution of a procedure.

As a last side-note, empirical execution time measurements are not the only benchmark tool that might be of interest to assess the performance of clusterings. To find bottle-necks in a procedure, it might also be interesting to run them while monitoring what they are doing with a *line profiler*. Tools like that offer more or less detailed insight into how often certain functions and statements are executed during the runtime and how much time each of them takes. This can give valuable hints on which parts of a procedure take up the most computation time and can reveal where further improvement may be worthwhile. Similarly, there are also *memory profilers* that show in detail how the memory demand evolves during the execution. I will not go into the complex details of profiling here but it should be noted that profiling usually creates a big overhead. Execution time measurements and profiling can therefore usually not be done simultaneously. The CommonNNClustering package also needs to be compiled with the `TRACE_CYTHON=1` option to allow line profiling for the Cython functions, which will drastically decrease the performance of the package.

profiling

In the following section, I will discuss the benchmark setup that was used in the CommonNNClustering development. This should offer the starting point for how these benchmarks can be realised in general, before a few actual benchmarks are presented in section 16.2.

16.1 The framework

The Python standard library provides the basic means to measure the execution time of function calls through the `timeit` module. Often, the convenience function `timeit.timeit` suffices to collect a timing for a small code fragment within a Python script:

```
import timeit
x = 1
execution_time = timeit.timeit(
  stmt="a + b",
  setup="b = 1",
  globals={"a": x}
)
```

The executed code passed in as `stmt` runs in `timeit`'s own namespace which can be overridden by using the `get_kth_neighbour` keyword argument. Alternatively, setup code passed in as `setup` is

executed once prior to the measurement and can be used to set the values of needed variables. By default, code is executed 10^6 times (this can be modified by the `number` keyword) and the timings are averaged for the returned result.

Extra care needs to be taken, if variables are changed by the measured code which may effect subsequent executions. The following for example, will provoke an `IndexError` in the second repetition as popping from an empty list is not allowed:

```
timeit.timeit(stmt="a.pop()", setup="a = [1]")
```

When we benchmark clusterings, the timed functions may alter the utilised data structures. In particular, our BFS implementation of CommonNN clustering for example uses an array to track which points have been already assigned to a cluster. This array needs to be reinitialised in between clusterings and we run into problems when we are interested in measuring *pure* clustering execution times. Setup code is either only executed once by `timeit` for all measurings or included in the code portion for which a timing should be measured. To prevent distorting (or breaking) effects on the obtained results, we therefore collect only single timings for one particular setup and do the averaging over multiple repetitions separately.

Benchmarking clusterings generally can be broken down into three setup stages: 1) A data set of a particular kind, size, and dimensionality needs to be generated. 2) The data optionally needs to be transformed into something that should be used as input to the clustering. 3) A timed function needs to be defined that can be passed to `timeit`. With loose type annotations this translates into the following functional signatures:

```
def gen_func(*args, **kwargs) -> Type["RawData"]: ...
def transform_func(
  data: Type["RawData"], *args, **kwargs) -> Type["Data"]: ...
def setup_func(
  data: Type["Data"], *args, **kwargs) -> Callable: ...
```

Such a benchmark approach is realised within the `CommonNNClustering-Benchmark` framework.[1] For the organisation of setup and timing, it uses the concept of a benchmark unit (`BMUnit`) that bundles the functions used in each stage alongside proper (keyword) arguments under a unit ID. Note, that it furthermore excepts `timed_args` and `timed_kwargs` that will be used to call the function to time, which will be prepared by the setup.

```
class BMUnit:
  def __init__(
      self, id,
      gen_func=None, gen_args=None, gen_kwargs=None,
      ...
      timed_args=None, timed_kwargs=None):
    self.id = id
    ...
```

A benchmark unit can be subjected to the `time_unit` function which orchestrates repeated setups and timings for this specific unit and returns a list of timings.

An iterable collection of benchmark units can be further grouped as a `Run`. Typically a benchmark run collects units through which only a few constraints of a clustering setup are varied, e.g. an increasing data set size while dimensionality and cluster parameters are fixed. A trivial example for the execution of a benchmark run is shown below for timings of the `time.sleep` standard library function. Note, that the benchmark framework is not (yet) an installable package but only a collection

[1] Visit the repository on GitHub: https://github.com/janjoswig/CommonNNClustering-Benchmark

of functions in provisional module files (`helper_<*>`). To use them from anywhere beyond the directory they are saved to (`bm_src_dir`), one can make them discoverable from within Python by using `sys.path.insert(0, bm_src_dir)` of the standard library.

```
run_name = "sleep_example"
bm_units = [
  helper_base.BMUnit(
    id=1,
    setup_func=lambda x: time.sleep,
    timed_args=(0.1,)
    ),
  helper_base.BMUnit(
    id=2,
    setup_func=lambda x: time.sleep,
    timed_args=(0.2,)
    )
  ]
run = Run(run_name, bm_units)
run.collect()
```

The resulting timings are accessible in a mapping of unit IDs to instances of the convenience type `IPython.core.magics.execution.TimeitResult` via `Run.timings`:

```
{1: <TimeitResult : 100 ms ± 6.92 µs per loop (...)>,
 2: <TimeitResult : 200 ms ± 16.9 µs per loop (...)>}
```

For the reporting and plotting of timing results, respective functionality is provided in `helper_plot` and `helper_timeit`.

To use this framework for our benchmarks, we need to do the following: 1) Define generation functions to produce data sets with different numbers of points, dimension, etc. (we can use the `sklearn.datasets` module for that). 2) Define transform functions if necessary to pre-compute distances or (sorted) neighbourhoods. 3) Define a setup function, that returns the function we actually want to time (i.e. creating a `Clustering` instance and returning the `fit` function or a wrapper including pre-processing steps). 4) Think about cases we want to test and define them as `Run` instances. 5) Run the benchmarks.

For the final step, the actual execution of the benchmarks, there is the little `bm.py` binary that can be run from the shell, for example like

```
python bm.py -runs bm_e -m qcm07
```

This expects the name of a Python-file in which a list of `Run` objects is defined as `run_list`. It can among other things also be passed a name identifier for the machine it currently runs on. The script will take care of the execution of all runs and saves the reported timings in json-format in sensible locations. The content of `bm_e.py` could look like the following:

```
n_points = [500 * 2**x for x in range(10)]
runs_report_dir = "examples"
raw_run_list = [
  (
    f"no_structure_e_{param_letter}",
    {
      "r_list": r,
      "c_list": c,
      "d_list": 2,
      "n_list": n_points,
```

```
        "gen_func": helper_base.gen_no_structure_points,
        "transform_func": helper_base.compute_neighbours,
        "transform_args": ("<r>",),
        "transform_kwargs": {"sort": True, "sort_by_n": True},
        "setup_kwargs": {
            "recipe": "sorted_neighbourhoods"
        }
      }
    )
    for param_letter, (r, c) in [
      ("a", (0.25, 0)), ("b", (0.25, 50)), ("c", (0.25, 100)),
      ("d", (0.1, 0)), ("e", (0.1, 50)), ("f", (0.1, 100)),
      ]
    ]
  run_list = (
    helper_base.Run(
      run_name,
      cases.gen_bm_units_cnnclustering__fit(**kwargs),
    )
    for run_name, kwargs in raw_run_list
  )
```

This makes additional use of a setup specific convenience function `gen_bm_units_cnnclustering__fit` that creates a series of `BMUnits` for the `Clustering._fit` function. It expects keyword arguments that control how the number of points or the cluster parameter should be varied in this series. The `raw_run_list` provides these for multiple runs. This convenience function is defined alongside the actual setup function in the `cases` module. In this example, we prepare six runs on a data set of uniformly distributed data points in 2D where the size is increased from 500 to 256,000 points. The cluster parameters r and n_c are kept within each run as fixed values but differ between the runs (each parameter combination is given a letter code `"a"` to `"f"`). Each clustering will use the `sorted_neighbourhoods` recipe on pre-computed neighbourhoods (denoted by the letter code `"e"`). The `"<r>"` placeholder in the transform function arguments is substituted by the actual value of r when the units are created, which ensures that the transformation always uses the correct value even if r is varied within the run. This example will generate six output-files `"<bm_src_dir>/reports/qcm07/examples/no_structure_e_<param_letter>.json"` containing the specific timings.

16.2 CommonNN clustering performance

Using the benchmark facilities described in the last section, a few interesting execution timings for the CommonNN clustering procedure in the present implementation have been done and will be shown here. The shown plots deliberately violate the normally good practice of comparing only one aspect of variation at a time to present the observations in a rather condensed form.

comparing recipes

Figure 16.1a presents threshold-based clustering timings for growing data sets of uniformly distributed points in 2D using the different fundamental input data recipes that have been already mentioned in section 15.6 (table 15.6). Compared are clusterings starting from point coordinates (brute force on-the-fly distance calculation), from pre-computed distances, and from neighbourhoods that have been optionally sorted by member indices—always in conjunction with the ideal combination of generic types. The obtained numbers correspond to the execution of the fit only, excluding for example the data preparation time and the post-processing of the obtained labels. Expectedly, the absolute runtimes decrease when the input data contains pre-computed information. The biggest impact,

however, has the sorting of neighbourhoods (see section 15.6.1) which amounts to a gain in efficiency by over two orders of magnitude with respect to the brute force base case. 256k data points can be clustered in about 9 seconds. Note that for the case of pre-computed distances, 64k data points where the maximum that could be handled with the available memory.

Also clearly noticeable is an acceleration when the neighbour search radius is reduced. In all clusterings, the similarity cut-off was set to $n_c = 0$, which will essentially bypass the similarity check. Small neighbourhoods are therefore even beneficial when neighbours only have to be retrieved and their comparison with respect to the number of shared neighbours is skipped (constant time). Strikingly, the relative scaling of all approaches remains the same as obviously quadratic. This can be rationalised in terms of the fact that a doubling of the number of points for this uniform data set will also roughly double the number of points in each neighbourhood. The number of potential pairs for which neighbourhoods need to be collected in order to be compared depends in turn quadratically on this. It is still surprising that the runtime complexity of how neighbours are retrieved, i.e. either trough a quadratically scaling calculation or a linear look-up, seems to be negligible for the overall scaling.

Figure 16.1b and c puts the gain in efficiency through a pre-computing of information into perspective as it shows absolute timings including the data preparation time. The clusterings use the scikit-learn *varied* data set (compare figure 15.5). For distances, the gain is only minimal if the information is not recycled. For neighbourhoods, however, the pre-computation cost is already repaid in the first run.

The cluster parameter n_c was set to a fixed non-zero value while the radius was scaled down for the test cases with bigger problem sizes. This approach is legitimate because for denser data the resolution parameter r may usually be reduced. Note that the observerd scaling is now sub-quadratic for the neighbourhood cases.

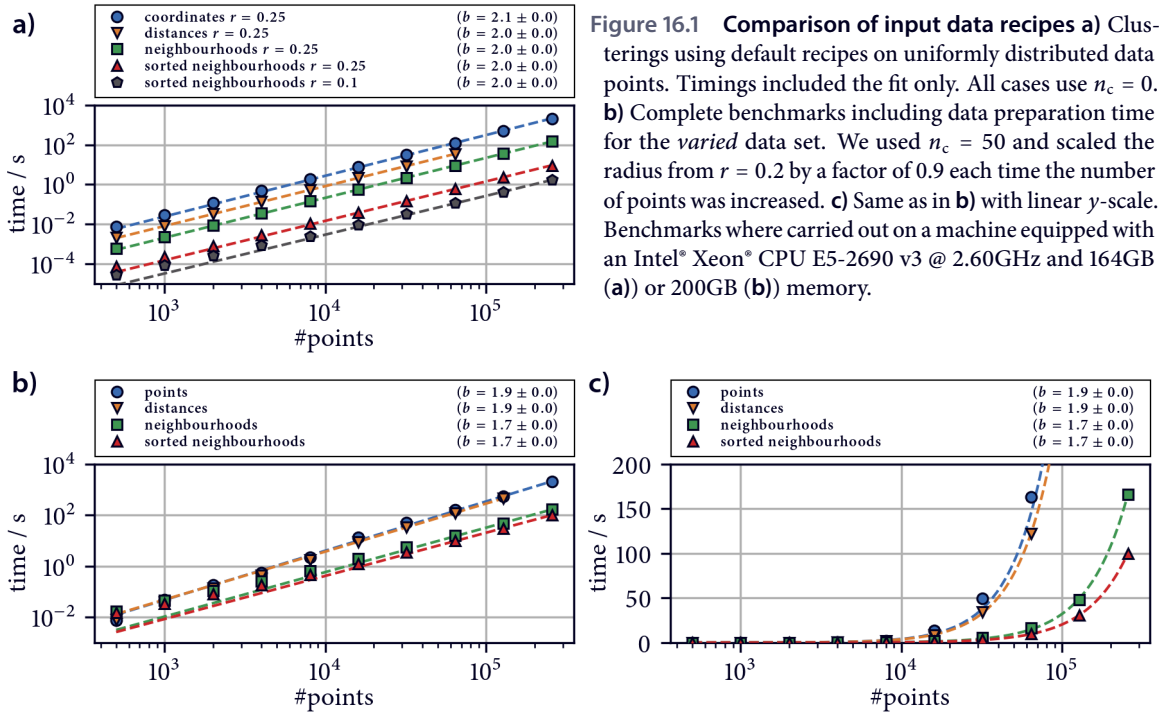


Figure 16.1 Comparison of input data recipes a) Clusterings using default recipes on uniformly distributed data points. Timings included the fit only. All cases use $n_c = 0$. **b)** Complete benchmarks including data preparation time for the *varied* data set. We used $n_c = 50$ and scaled the radius from $r = 0.2$ by a factor of 0.9 each time the number of points was increased. **c)** Same as in **b)** with linear y-scale. Benchmarks where carried out on a machine equipped with an Intel® Xeon® CPU E5-2690 v3 @ 2.60GHz and 164GB (**a)**) or 200GB (**b)**) memory.

In section 15.6, the approach was discussed to sort neighbourhoods not only by the indices of their members to allow quick similarity checks (section 15.6.1) but also in total by the number of members they contain. Figure 16.2 assesses the effect of this for timings measuring the fit only with uniformly distributed data points and $n_c = 0$ like in figure 16.1a and with complete timings for the *varied* data set

neighbourhoods sorting

as in figure 16.1b. For the uniform case, a priorisation of the point processing order by total neighbour counts has basically no effect for larger numbers of points. In the *varied* case, however, we see an efficiency gain that even counter-balances the larger preparation cost.

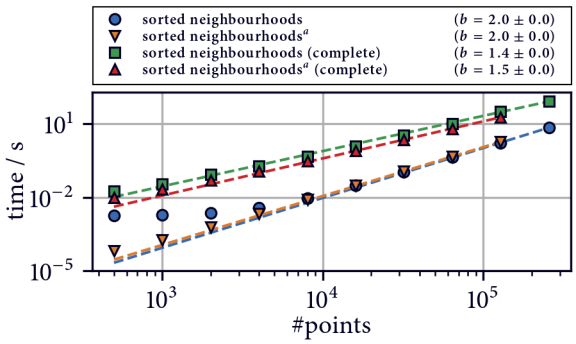


Figure 16.2 Comparison of neighbourhoods sorting by index and member count For uniformly distributed data, timings measuring the fit only ($r = 0.25$, $n_c = 0$) are shown for neighbourhoods pre-sorted by member indices and additionally sorted by absolute member counts (superscript^a). The same is done for complete timings on the *varied* data ($r = 0.2$ and scaled down by a factor 0.9 at step, $n_c = 50$).

comparing parameters

We largely ignored the effect of the cluster parameters on the actual result of the clustering and focused only on execution times here. It was already noticed, that the neighbour search radius r can have a strong effect on the clustering performance but also the similarity cut-off n_c may have a substantial impact. Figure 16.3 compares different parameter combinations on the uniform set using the `"sorted_neighbourhoods"` recipe. For non-zero similarity thresholds, there is a sudden increase in the execution timings at a certain point. The reason for that might be that the number of successful and failing similarity checks, which is a consequence of the cluster parameters relative to the size (and distribution) of the data set, influences the clustering performance. Most interestingly, very large density thresholds (low r, high n_c in the context of n) accelerate the procedure probably because a lot of points can be declared noise and do not need to be checked. In general, however, larger n_c values cause a higher cost of similarity checks. Overall the performance influence is highly unpredictable. For more details, the interested reader is refered to the benchmark repository that also contains examples for 2D parameter scans.

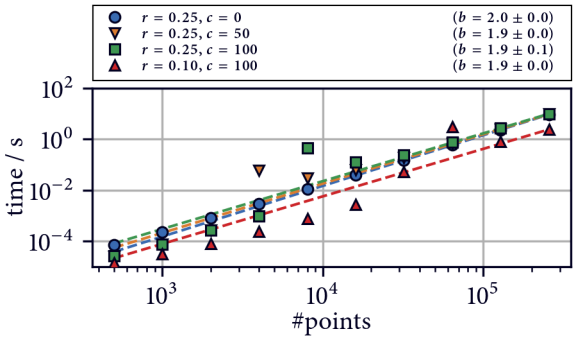


Figure 16.3 Comparison of cluster parameters with sorted neighbourhoods Execution timings, measuring the fit only, on uniformly distributed data sets. The relative settings for r and n_c with respect to the data set size n decides not only over the clustering result but also its performance.

General advice

From the shown timings, we can conclude the following universal advice. If possible, always use pre-sorted neighbourhoods with the respective recipe for the best performance and optionally consider also sorting of the data points with respect to their neighbour count. Choose a preferably small radius r. Since the other cluster parameter n_c is then responsible for tuning the cluster result, there is not much freedom on how to set it for good performance. Be aware that when tuning n_c from low to high, the computation cost is generally expected to go up. Only for very large values a drastic performance increase will be observed but this may not yield a useful result any more. If a lot of parameter combinations should be tested (e.g. in semi-automatic hierarchical clustering), try to

limit the total number of data point to < 100k so that a single clustering is not taking more time than a few seconds.[2]

Benchmarks for the more or less entire clustering procedure can deliver a good overall performance impression but the results are often not straightforward to interpret. Especially when the cluster parameters are varied it can in general only be guessed why the execution time is influenced. A crucial element of the whole procedure is the similarity check, which is done very frequently, and it can be suspected that it impacts the timings quite a lot. To assess in more detail how the performance of individual similarity checks changes depending on the strategy that is used here (compare section 15.6.1), let's consider a few separate benchmarks of only this function.

For this, we have set up a bunch of synthetic test cases to model extrem situations of neighbourhood containers with different lengths that can be subjected to a similarity check.

equal Two identical containers of length n_a with strictly increasing member indices. Each item comparison is a success.

shuffled Like *equal* but the container content was brought into no specific order. Each item comparison is a success.

mixed Like *equal* but the second container contains only every 2nd element is of length $n_b = n_a/2$. Each item comparison from b to a is a success.

different Two containers of length n_a with strictly increasing indices in non-overlapping ranges. All item comparisons fail.

different (alternating) Like *different* but with perfectly alternating indices between the two containers. All item comparisons fail.

With these cases, different flavours of the similarity check have been timed. Figure 16.4a shows the basic possibilities. For naive containment checks using the `Contains` similarity checker with two `Vector` neighbours containers on the *equal* case, we observe the expected quadratic scaling (not recommended). Leveraging of an unordered set for constant time containment checks brings this done to linear scaling. If the similarity cut-off n_c is set to lower values (33 % instead of 50 % of the neighbourhoods length n_a), it is not surprising that the check is a little bit faster because we can break out early once n_c is satisfied. A shuffling of the containers makes the check less predictable, although it is not a hundred percent clear why that is (still every item comparison should be a success). A linear screen of sorted neighbourhoods gives the absolute best performance. For $n_c = 0$, the check is bypassed entirely.

In contrast, figure 16.4b shows an assessment of the *equal* case with 625k items but with increasing values for the similarity cut-off n_c. The trends for the different approaches are the same as before. Note that the possibility of switching in the `SwitchContains` similarity checker has no impact here because the neighbourhoods are of the same lengths.

[2]It is not discussed here but it is often a possibility to cluster only a representative subset of a data set (e.g. generated by striding or k-means pre-clustering) and to assign the full set to the identified clusters in a later step. This can be referred to as cluster label *prediction* and the CommonNNClustering project provides the `Predictor` interface for it.

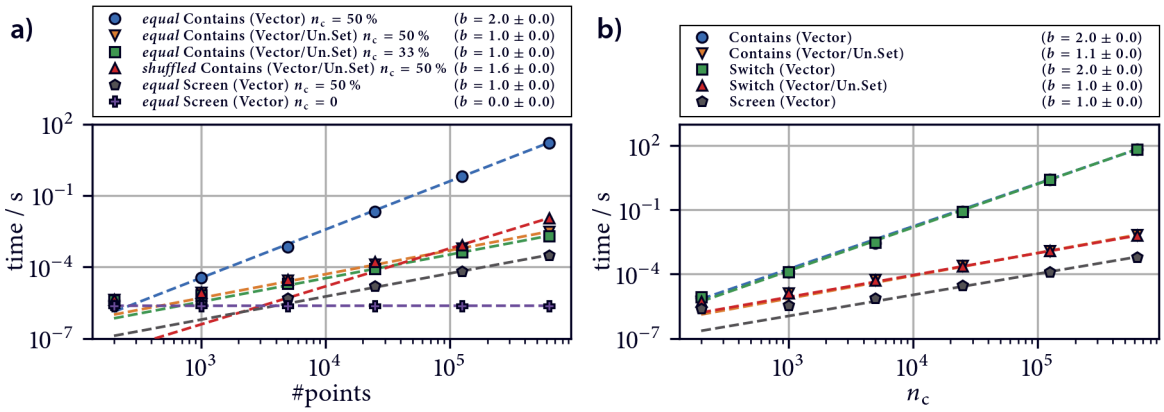


Figure 16.4 Execution timings for the similarity check
a) Compared are different check variants on neighbourhood test containers with increasing length. A linear screen on sorted containers performs best. **b)** Comparisons for the *equal* neighbourhoods case with increasing values for n_c.

In general, however, switching can indeed have a positive effect. Figure 16.5 shows timings for the *mixed* case with and without switching. While the scaling of the approaches is of course not effected, absolute runtimes are reduced. The effect is expectedly smaller for the quadratically scaling naive approach because a single containment check involves a loop over both of the containers anyway.

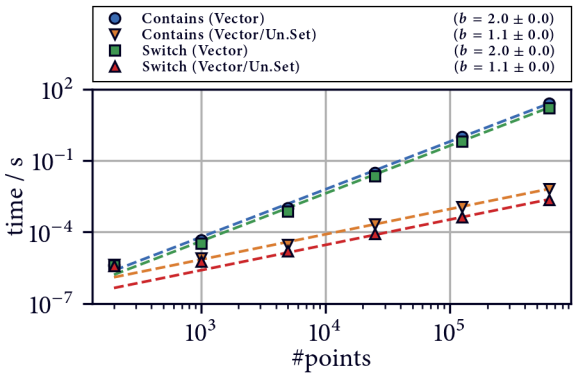


Figure 16.5 Impact of conditional switching before containment checks If two neighbourhoods of unequal length are subjected to a similarity check, it can make sense to ensure that the shorter container is looped over while the longer will be used for the `contains` check.

Finally, let us get an idea about the worst case performance of the similarity check. The worst case here means that the similarity cut-off is not satisfied by the containers and that there can be no early break. For test cases of non-overlapping containers, figure 16.6 shows the timings for all considered approaches. Generally, the naive approach suffers the most while the linear screen is almost not effected by the *different* case. The latter check only has a hard time with perfectly alternating containers. This case can be, however, assumed to be quite rare in reality.

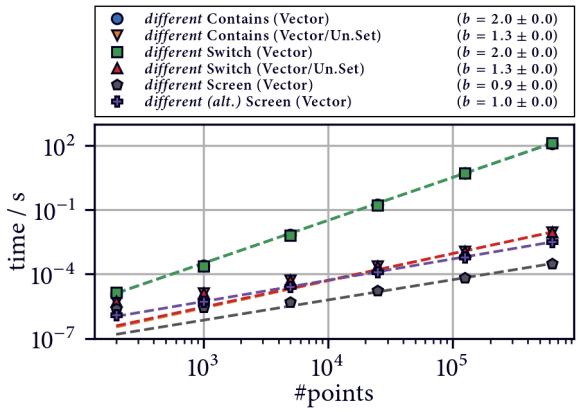


Figure 16.6 **Worst case scaling of similarity checks** On non-overlapping neighbourhoods, the performance of a single similarity check can be slow. We observe about one order of magnitude for the naive case and a factor of two for the set-lookup variants. Linear screens are not necessarily affected by segmentally differing containers. Their worst case are perfectly alternating non-overlapping containers where we see one order of magnitude loss in performance.

Appendix

References

[1] R. Frigg, J. Reiss, *Synthese* **2009**, *169*, 593–613.

[2] I. Mills, W. Metanomski, *Quantities, Units and Symbols in Physical Chemistry*, (Eds.: E. R. Cohen, T. Cvitas, J. G. Frey, B. Holström, K. Kuchitsu, R. Marquardt, I. Mills, F. Pavese, M. Quack, J. Stohner, H. L. Strauss, M. Takami, A. J. Thor), Royal Society of Chemistry, Cambridge, **2007**.

[3] J. Burnet, *Early greek philosophy*, 3rd ed., A & C Black, London, **1920**.

[4] J. H. Holden, *A History of Horoscopic Astrology*, 2nd ed., AFA, **1996**.

[5] G. Smith, 'Newton's Philosophiae Naturalis Principia Mathematica' in *Stanford Encycl. Philos.* (Ed.: E. N. Zalta), Metaphysics Research Lab, Stanford University, **2008**.

[6] S. Strogatz, New York Times Opinionator Guest Column: Loves Me, Loves Me Not (Do the Math), **2009**.

[7] H. Goldstine, A. Goldstine, *IEEE Ann. Hist. Comput.* **1996**, *18*, 10–16.

[8] W. T. Moye, *ENIAC: The Army-Sponsored Revolution*, ARL Historian, **1996**.

[9] H. L. Anderson, *Los Alamos Sci.* **1986**, *14*, 104–105.

[10] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, *J. Chem. Phys.* **1953**, 1087–1092.

[11] E. Fermi, J. G. Pasta, S. M. Ulam, *LASL Rep. LA-1940* **1955**.

[12] T. Dauxois, *Phys. Today* **2008**, *61*, 55–57.

[13] B. J. Alder, T. E. Wainwright, *J. Chem. Phys.* **1957**, *27*, 1208–1209.

[14] G. Battimelli, G. Ciccotti, *Eur. Phys. J. H.* **2018**, *43*, 303–335.

[15] B. J. Alder, T. E. Wainwright, *J. Chem. Phys.* **1959**, *31*, 459–466.

[16] J. B. Gibson, A. N. Goland, M. Milgram, G. H. Vineyard, *Phys. Rev.* **1960**, *120*, 1229–1253.

[17] A. Rahman, *Phys. Rev.* **1964**, *136*, A405–A411.

[18] A. Rahman, F. H. Stillinger, *J. Chem. Phys.* **1971**, *55*, 3336–3359.

[19] F. H. Stillinger, A. Rahman, *J. Chem. Phys.* **1974**, *60*, 1545–1557.

[20] J. A. McCammon, B. R. Gelin, M. Karplus, *Nature* **1977**, *267*, 585–590.

[21] M. Levitt, A. Warshel, *Nature* **1975**, *253*, 694–698.

[22] C. Levinthal, 'Levinthal's Paradox' in *Mossbauer Spectrosc. Biol. Syst. Proc. a Meet. held Allert. House, Monticello, Illinois*, (Eds.: J. T. P. DeBrunner, E. Munck), University of Illinois Press, **1969**, pp. 22–24.

[23] A. Warshel, *Nature* **1976**, *260*, 679–683.

[24] P. Nogly, T. Weinert, D. James, S. Carbajo, D. Ozerov, A. Furrer, D. Gashi, V. Borin, P. Skopintsev, K. Jaeger, K. Nass, P. Båth, R. Bosman, J. Koglin, M. Seaberg, T. Lane, D. Kekilli, S. Brünle, T. Tanaka, W. Wu, C. Milne, T. White, A. Barty, U. Weierstall, V. Panneels, E. Nango, S. Iwata, M. Hunter, I. Schapiro, G. Schertler, R. Neutze, J. Standfuss, *Science* **2018**, *361*, DOI 10.1126/science.aat0094.

[25] L. Verlet, *Phys. Rev.* **1967**, *159*, 98–103.

[26] S. Lifson, A. Warshel, *J. Chem. Phys.* **1968**, *49*, 5116–5129.

[27] M. Levitt, S. Lifson, *J. Mol. Biol.* **1969**, *46*, 269–279.

[28] J.-P. Ryckaert, G. Ciccotti, H. J. Berendsen, *J. Comput. Phys.* **1977**, *23*, 327–341.

[29] W. van Gunsteren, H. Berendsen, *Mol. Phys.* **1977**, *34*, 1311–1327.

[30] H. C. Andersen, *J. Chem. Phys.* **1980**, *72*, 2384–2393.

[31] M. Parrinello, A. Rahman, *Phys. Rev. Lett.* **1980**, *45*, 1196–1199.

[32] M. Parrinello, A. Rahman, *J. Appl. Phys.* **1981**, *52*, 7182–7190.

[33] W. C. Swope, H. C. Andersen, P. H. Berens, K. R. Wilson, *J. Chem. Phys.* **1982**, *76*, 637–649.

[34] J. P. M. Postma, H. J. C. Berendsen, J. R. Haak, *Faraday Symp. Chem. Soc.* **1982**, *17*, 55.

[35] U. C. Singh, F. K. Brown, P. A. Bash, P. A. Kollman, *J. Am. Chem. Soc.* **1987**, *109*, 1607–1614.

[36] M. P. Allen, D. J. Tildesley, *Computer Simulation of Liquids*, *Vol. 1*, 2nd ed., Oxford University Press, **2017**.

[37] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, M. L. Klein, *J. Chem. Phys.* **1983**, *79*, 926–935.

[38] W. L. Jorgensen, J. Tirado-Rives, *J. Am. Chem. Soc.* **1988**, *110*, 1657–1666.

[39] A. K. Mazur, R. A. Abagyan, *J. Biomol. Struct. Dyn.* **1989**, *6*, 815–832.

[40] R. Abagyan, P. Argos, *J. Mol. Biol.* **1992**, *225*, 519–532.

[41] K. Kremer, G. S. Grest, *J. Chem. Phys.* **1990**, *92*, 5057–5086.

[42] D. C. Rapaport, *J. Phys. A. Math. Gen.* **1978**, *11*, L213–L217.

[43] H. Grubmüller, B. Heymann, P. Tavan, *Science* **1996**, *271*, 997–999.

[44] C. Jarzynski, *Phys. Rev. Lett.* **1997**, *78*, 2690–2693.

[45] M. J. Dudek, J. W. Ponder, *J. Comput. Chem.* **1995**, *16*, 791–816.

[46] A. Warshel, M. Kato, A. V. Pisliakov, *J. Chem. Theory Comput.* **2007**, *3*, 2034–2045.

[47] C. Schütte, A. Fischer, W. Huisinga, P. Deuflhard, *J. Comput. Phys.* **1999**, *151*, 146–168.

[48] Y. Duan, P. A. Kollman, *Science* **1998**, *282*, 740–744.

[49] S. Plimpton, *Comput. Mater. Sci.* **1995**, *4*, 361–364.

[50] H. J. C. Berendsen, D. van der Spoel, R. van Drunen, *Comput. Phys. Commun.* **1995**, *91*, 43–56.

[51] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham, S. DeBolt, D. Ferguson, G. Seibel, P. Kollman, *Comput. Phys. Commun.* **1995**, *91*, 1–41.

[52] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, K. Schulten, *J. Comput. Chem.* **2007**, *28*, 2618–2640.

[53] J. A. Anderson, C. D. Lorenz, A. Travesset, *J. Comput. Phys.* **2008**, *227*, 5342–5359.

[54] P. L. Freddolino, A. S. Arkhipov, S. B. Larson, A. McPherson, K. Schulten, *Structure* **2006**, *14*, 437–449.

[55] D. E. Shaw, P. J. Adams, A. Azaria, J. A. Bank, B. Batson, A. Bell, M. Bergdorf, J. Bhatt, J. A. Butts, T. Correia, R. M. Dirks, R. O. Dror, M. P. Eastwood, B. Edwards, A. Even, P. Feldmann, M. Fenn, C. H. Fenton, A. Forte, J. Gagliardo, G. Gill, M. Gorlatova, B. Greskamp, J. Grossman, J. Gullingsrud, A. Harper, W. Hasenplaugh, M. Heily, B. C. Heshmat, J. Hunt, D. J. Ierardi, L. Iserovich, B. L. Jackson, N. P. Johnson, M. M. Kirk, J. L. Klepeis, J. S. Kuskin, K. M. Mackenzie, R. J. Mader, R. McGowen, A. McLaughlin, M. A. Moraes, M. H. Nasr, L. J. Nociolo, L. O'Donnell, A. Parker, J. L. Peticolas, G. Pocina, C. Predescu, T. Quan, J. K. Salmon, C. Schwink, K. S. Shim, N. Siddique, J. Spengler, T. Szalay, R. Tabladillo, R. Tartler, A. G. Taube, M. Theobald, B. Towles, W. Vick, S. C. Wang, M. Wazlowski, M. J. Weingarten, J. M. Williams, K. A. Yuh in Proc. Int. Conf. High Perform. Comput. Networking, Storage Anal. ACM, New York, NY, USA, **2021**, pp. 1–11.

[56] J. R. Perilla, K. Schulten, *Nat. Commun.* **2017**, *8*, 15959.

[57] J. Jung, C. Kobayashi, K. Kasahara, C. Tan, A. Kuroda, K. Minami, S. Ishiduki, T. Nishiki, H. Inoue, Y. Ishikawa, M. Feig, Y. Sugita, *J. Comput. Chem.* **2021**, *42*, 231–241.

[58] L. Casalino, Z. Gaieb, J. A. Goldsmith, C. K. Hjorth, A. C. Dommer, A. M. Harbison, C. A. Fogarty, E. P. Barros, B. C. Taylor, J. S. McLellan, E. Fadda, R. E. Amaro, *ACS Cent. Sci.* **2020**, *6*, 1722–1734.

[59] A. Dommer, L. Casalino, F. Kearns, M. Rosenfeld, N. Wauer, S.-H. Ahn, J. Russo, S. Oliveira, C. Morris, A. Bogetti, A. Trifan, A. Brace, T. Sztain, A. Clyde, H. Ma, C. Chennubhotla, H. Lee, M. Turilli, S. Khalid, T. Tamayo-Mendoza, M. Welborn, A. Christensen, D. G. A. Smith, Z. Qiao, S. K. Sirumalla, M. O'Connor, F. Manby, A. Anandkumar, D. Hardy, J. Phillips, A. Stern, J. Romero, D. Clark, M. Dorrell, T. Maiden, L. Huang, J. McCalpin, C. Woods, A. Gray, M. Williams, B. Barker, H. Rajapaksha, R. Pitts, T. Gibbs, J. Stone, D. Zuckerman, A. Mulholland, T. Miller, S. Jha, A. Ramanathan, L. Chong, R. Amaro, *bioRxiv* **2021**, DOI 10.1101/2021.11.12.468428.

[60] J.-M. Chia, Accelerating Drug Discovery with Supercomputing Scale Biomolecular Simulations on Azure, **2020**.

[61] M. S. Islam, S. L. Junod, S. Zhang, Z. Y. Buuh, Y. Guan, M. Zhao, K. H. Kaneria, P. Kafley, C. Cohen, R. Maloney, Z. Lyu, V. A. Voelz, W. Yang, R. E. Wang, *Nat. Commun.* **2022**, *13*, 350.

[62] E. J. Maginn, J. R. Elliott, *Ind. Eng. Chem. Res.* **2010**, *49*, 3059–3078.

[63] F. Fratev, S. Sirimulla, *Sci. Rep.* **2019**, *9*, 16829.

[64] L. Carvalho Martins, E. A. Cino, R. S. Ferreira, *J. Chem. Theory Comput.* **2021**, *17*, 4262–4273.

[65] C. Tian, K. Kasavajhala, K. A. A. Belfon, L. Raguette, H. Huang, A. N. Migues, J. Bickel, Y. Wang, J. Pincay, Q. Wu, C. Simmerling, *J. Chem. Theory Comput.* **2020**, *16*, 528–552.

[66] C. J. Dickson, R. C. Walker, I. R. Gould, *J. Chem. Theory Comput.* **2022**, *18*, 1726–1736.

[67] G. Balogh, T. Gyöngyösi, I. Timári, M. Herczeg, A. Borbás, K. Fehér, K. E. Kövér, *J. Chem. Inf. Model.* **2019**, *59*, 4855–4867.

[68] A. M. Salsbury, J. A. Lemkul, *Curr. Opin. Struct. Biol.* **2021**, *67*, 9–17.

[69] Z. Li, L. F. Song, P. Li, K. M. Merz, *J. Chem. Theory Comput.* **2020**, *16*, 4429–4442.

[70] X. He, V. H. Man, W. Yang, T.-S. Lee, J. Wang, *J. Chem. Phys.* **2020**, *153*, 114502.

[71] R. Galvelis, S. Doerr, J. M. Damas, M. J. Harvey, G. De Fabritiis, *J. Chem. Inf. Model.* **2019**, *59*, 3485–3493.

[72] M. Chen, *Eur. Phys. J. B* **2021**, *94*, 211.

[73] R. Menichetti, M. Giulini, R. Potestio, *Eur. Phys. J. B* **2021**, *94*, 204.

[74] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer,

S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, D. Hassabis, *Nature* **2021**, *596*, 583–589.

[75] F. Noé, S. Olsson, J. Köhler, H. Wu, *Science* **2019**, *365*, DOI 10.1126/science.aaw1147.

[76] A. Glielmo, B. E. Husic, A. Rodriguez, C. Clementi, F. Noé, A. Laio, *Chem. Rev.* **2021**, *121*, 9722–9758.

[77] G. Ciccotti, C. Dellago, M. Ferrario, E. R. Hernández, M. E. Tuckerman, *Eur. Phys. J. B* **2022**, *95*, 3.

[78] L. Donati, M. Weber, B. G. Keller, *J. Phys. Condens. Matter* **2021**, *33*, 115902.

[79] S. Kieninger, B. G. Keller, *J. Chem. Phys.* **2021**, *154*, 094102.

[80] President's Information Technology Advisor Committee, *Rep. to Pres.* **2005**.

[81] P. W. Anderson, 'Local moments and localized states (1977)' in *Nobel Lect. Phys. 1971-1980*, (Ed.: S. Lundqvist), World Scientific, Singapore, **1992**.

[82] S. Y. Diallo, J. J. Padilla, I. Bozkurt, A. Tolk, 'Modeling and Simulation as a Theory Building Paradigm' in **2013**, pp. 193–206.

[83] M. Born, R. Oppenheimer, *Ann. Phys.* **1927**, *389*, 457–484.

[84] P. Ehrenfest, *Zeitschrift für Phys.* **1927**, *45*, 455–457.

[85] J. Fass, D. Sivak, G. Crooks, K. Beauchamp, B. Leimkuhler, J. Chodera, *Entropy* **2018**, *20*, 318.

[86] W. B. Hayes, *Phys. Rev. Lett.* **2003**, *90*, 054104.

[87] W. F. van Gunsteren, D. Bakowies, R. Baron, I. Chandrasekhar, M. Christen, X. Daura, P. Gee, D. P. Geerke, A. Glättli, P. H. Hünenberger, M. A. Kastenholz, C. Oostenbrink, M. Schenk, D. Trzesniak, N. F. A. van der Vegt, H. B. Yu, *Angew. Chemie Int. Ed.* **2006**, *45*, 4064–4092.

[88] R. D. Skeel, *SIAM J. Sci. Comput.* **2009**, *31*, 1363–1378.

[89] E. Braun, J. Gilmer, H. B. Mayes, D. L. Mobley, J. I. Monroe, S. Prasad, D. M. Zuckerman, *Living J. Comput. Mol. Sci.* **2019**, *1*, DOI 10.33011/livecoms.1.1.5957.

[90] K. Johannessen, *Eur. J. Phys.* **2014**, *35*, 035014.

[91] M. E. Tuckerman, *Statistical Mechanics: Theory and Molecular Simulation*, Oxford University Press, **2010**.

[92] J. K. Vandiver, D. Gossard, 'Lecture 15: Introduction to Lagrange with examples' in *Struct. Mech. Course 2.003S No.~1.053J*, Cambridge~MA, **2011**.

[93] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, P. A. Kollman, *J. Am. Chem. Soc.* **1995**, *117*, 5179–5197.

[94] S. Riniker, *J. Chem. Inf. Model.* **2018**, *58*, 565–578.

[95] K. Lindorff-Larsen, S. Piana, K. Palmo, P. Maragakis, J. L. Klepeis, R. O. Dror, D. E. Shaw, *Proteins Struct. Funct. Bioinforma.* **2010**, *78*, 1950–1958.

[96] P. M. Morse, *Phys. Rev.* **1929**, *34*, 57–64.

[97] K. Farah, F. Müller-Plathe, M. C. Böhm, *ChemPhysChem* **2012**, *13*, 1127–1151.

[98] D. M. Ferguson, *J. Comput. Chem.* **1995**, *16*, 501–511.

[99] B. R. Brooks, C. L. Brooks, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, M. Karplus, *J. Comput. Chem.* **2009**, *30*, 1545–1614.

[100] A. Kania, K. Sarapata, M. Gucwa, A. Wójcik-Augustyn, *J. Phys. Chem. A* **2021**, *125*, 2673–2681.

[101] G. A. Kaminski, R. A. Friesner, J. Tirado-Rives, W. L. Jorgensen, *J. Phys. Chem. B* **2001**, *105*, 6474–6487.

[102] P. Li, L. F. Song, K. M. Merz, *J. Phys. Chem. B* **2015**, *119*, 883–895.

[103] P. Li, *Front. Chem.* **2021**, *9*, DOI 10.3389/fchem.2021.721960.

[104] R. A. Buckingham, *Proc. R. Soc. London A* **1938**, *168*, 264–283.

[105] J. Delaye, V. Louis-Achille, D. Ghaleb, *J. Non. Cryst. Solids* **1997**, *210*, 232–242.

[106] M. Levitt, M. Hirshberg, R. Sharon, V. Daggett, *Comput. Phys. Commun.* **1995**, *91*, 215–231.

[107] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, E. Lindahl, *SoftwareX* **2015**, *1-2*, 19–25.

[108] D. Frenkel, B. Smit, *Understanding Molecular Simulation*, 2nd ed., (Eds.: D. Frenkel, B. Smit), Academic Press, San Diego, **2002**, pp. 167–200.

[109] K. M. Jablonka, D. Ongari, B. Smit, *J. Chem. Theory Comput.* **2019**, *15*, 5635–5641.

[110] A. Kubincová, S. Riniker, P. H. Hünenberger, *Phys. Chem. Chem. Phys.* **2020**, *22*, 26419–26437.

[111] H. Wang, P. Zhang, C. Schütte, *J. Chem. Theory Comput.* **2012**, *8*, 3243–3256.

[112] C. Predescu, A. K. Lerer, R. A. Lippert, B. Towles, J. Grossman, R. M. Dirks, D. E. Shaw, *J. Chem. Phys.* **2020**, *152*, 084113.

[113] V. S. Inakollu, D. P. Geerke, C. N. Rowley, H. Yu, *Curr. Opin. Struct. Biol.* **2020**, *61*, 182–190.

[114] F.-Y. Lin, P. E. M. Lopes, E. Harder, B. Roux, A. D. MacKerell, *J. Chem. Inf. Model.* **2018**, *58*, 993–1004.

[115] J. Huang, A. D. MacKerell, *Biophys. J.* **2014**, *107*, 991–997.

[116] J. Huang, A. D. MacKerell, *Curr. Opin. Struct. Biol.* **2018**, *48*, 40–48.

[117] H. Torabifard, G. A. Cisneros, *Chem. Sci.* **2017**, *8*, 6230–6238.

[118] Š. Timr, J. Kadlec, P. Srb, O. H. S. Ollila, P. Jungwirth, *J. Phys. Chem. Lett.* **2018**, *9*, 1613–1619.

[119] T. Martinek, E. Duboué-Dijon, Š. Timr, P. E. Mason, K. Baxová, H. E. Fischer, B. Schmidt, E. Pluhařová, P. Jungwirth, *J. Chem. Phys.* **2018**, *148*, 222813.

[120] P. Li, K. M. Merz, *Chem. Rev.* **2017**, *117*, 1564–1686.
[121] A. Sengupta, A. Seitz, K. M. Merz, *J. Am. Chem. Soc.* **2018**, *140*, 15166–15169.
[122] L. F. Song, A. Sengupta, K. M. Merz, *J. Am. Chem. Soc.* **2020**, *142*, 6365–6374.
[123] P. Kantakevičius, C. Mathiah, L. O. Johannissen, S. Hay, *J. Chem. Theory Comput.* **2022**, *18*, 2367–2374.
[124] F. Duarte, P. Bauer, A. Barrozo, A. Amrein, M. Purg, J. Åqvist, S. Caroline, L. Kamerlin, *J. Phys. Chem. B* **2014**, *118*, 4351–4362.
[125] M. B. Peters, Y. Yang, B. Wang, L. Füsti-Molnár, M. N. Weaver, K. M. Merz, *J. Chem. Theory Comput.* **2010**, *6*, 2935–2947.
[126] P. E. M. Lopes, O. Guvench, A. D. MacKerell, 'Current Status of Protein Force Fields for Molecular Dynamics Simulations' in **2015**, pp. 47–71.
[127] S. Patel, C. L. Brooks, *Mol. Simul.* **2006**, *32*, 231–249.
[128] G. Lamoureux, A. D. MacKerell, B. Roux, *J. Chem. Phys.* **2003**, *119*, 5185–5197.
[129] J. W. Ponder, C. Wu, P. Ren, V. S. Pande, J. D. Chodera, M. J. Schnieders, I. Haque, D. L. Mobley, D. S. Lambrecht, R. A. DiStasio, M. Head-Gordon, G. N. I. Clark, M. E. Johnson, T. Head-Gordon, *J. Phys. Chem. B* **2010**, *114*, 2549–2564.
[130] P. E. M. Lopes, J. Huang, J. Shim, Y. Luo, H. Li, B. Roux, A. D. MacKerell, *J. Chem. Theory Comput.* **2013**, *9*, 5430–5449.
[131] F.-Y. Lin, J. Huang, P. Pandey, C. Rupakheti, J. Li, B. Roux, A. D. MacKerell, *J. Chem. Theory Comput.* **2020**, *16*, 3221–3239.
[132] Y. Shi, Z. Xia, J. Zhang, R. Best, C. Wu, J. W. Ponder, P. Ren, *J. Chem. Theory Comput.* **2013**, *9*, 4046–4063.
[133] C. Zhang, C. Lu, Z. Jing, C. Wu, J.-P. Piquemal, J. W. Ponder, P. Ren, *J. Chem. Theory Comput.* **2018**, *14*, 2084–2108.
[134] J. Huang, A. C. Simmonett, F. C. Pickard, A. D. MacKerell, B. R. Brooks, *J. Chem. Phys.* **2017**, *147*, 161702.
[135] C. Liu, J.-P. Piquemal, P. Ren, *J. Chem. Theory Comput.* **2019**, *15*, 4122–4139.
[136] F. Vitalini, A. S. J. S. Mey, F. Noé, B. G. Keller, *J. Chem. Phys.* **2015**, *142*, 084101.
[137] S. Jephthah, F. Pesce, K. Lindorff-Larsen, M. Skepö, *J. Chem. Theory Comput.* **2021**, *17*, 6634–6646.
[138] P. Robustelli, S. Piana, D. E. Shaw, *Proc. Natl. Acad. Sci.* **2018**, *115*, DOI 10.1073/pnas.1800690115.
[139] A. Kuzmanic, R. B. Pritchard, D. F. Hansen, F. L. Gervasio, *J. Phys. Chem. Lett.* **2019**, *10*, 1928–1934.
[140] K. K. Patapati, N. M. Glykos, *Biophys. J.* **2011**, *101*, 1766–1771.
[141] A. Wang, Z. Zhang, G. Li, *J. Phys. Chem. Lett.* **2018**, *9*, 7110–7116.
[142] N. Panel, F. Villa, E. J. Fuentes, T. Simonson, *Biophys. J.* **2018**, *114*, 1091–1102.
[143] A. S. Kamenik, P. H. Handle, F. Hofer, U. Kahler, J. Kraml, K. R. Liedl, *J. Chem. Phys.* **2020**, *153*, 185102.
[144] A. J. Hazel, E. T. Walters, C. N. Rowley, J. C. Gumbart, *J. Chem. Phys.* **2018**, *149*, 072317.
[145] N. Li, Y. Gao, F. Qiu, T. Zhu, *Molecules* **2021**, *26*, 5379.
[146] Z. Jing, C. Liu, R. Qi, P. Ren, *Proc. Natl. Acad. Sci.* **2018**, *115*, DOI 10.1073/pnas.1805049115.
[147] J. Litman, A. C. Thiel, M. J. Schnieders, *J. Chem. Theory Comput.* **2019**, *15*, 4602–4614.
[148] R. Qi, B. Walker, Z. Jing, M. Yu, G. Stancu, R. Edupuganti, K. N. Dalby, P. Ren, *J. Phys. Chem. B* **2019**, *123*, 6034–6041.
[149] V. V. Welborn, T. Head-Gordon, *J. Am. Chem. Soc.* **2019**, *141*, 12487–12492.
[150] T. Lewis-Atwell, P. A. Townsend, M. N. Grayson, *Tetrahedron* **2021**, *79*, 131865.
[151] Z. Jing, C. Liu, S. Y. Cheng, R. Qi, B. D. Walker, J.-P. Piquemal, P. Ren, *Annu. Rev. Biophys.* **2019**, *48*, 371–394.
[152] T. A. Wassenaar, PhD thesis, **2006**.
[153] K. Lindorff-Larsen, P. Maragakis, S. Piana, M. P. Eastwood, R. O. Dror, D. E. Shaw, *PLoS One* **2012**, *7*, e32131.
[154] M. C. Bellissent-Funel, *J. Mol. Liq.* **1998**, *78*, 19–28.
[155] V. Gapsys, B. L. de Groot, *Elife* **2020**, *9*, 9:e57589.
[156] H. Bekker, *J. Comput. Chem.* **1997**, *18*, 1930–1942.
[157] U. K. Deiters, *Zeitschrift für Phys. Chemie* **2013**, *227*, 345–352.
[158] R. Weber, H.-J. Schek, S. Blott in Proc. 24rd Int. Conf. Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, **1998**, pp. 194–205.
[159] J. Groß, M. Köster, A. Krüger, *Comput. Graph. & Vis. Comput.* **2019**, 55–63.
[160] J. L. Bentley, *Commun. ACM* **1975**, *18*, 509–517.
[161] F. Gieseke, J. Heinermann, C. Oancea, C. Igel in Proc. 31st Int. Conf. Int. Conf. Mach. Learn. - Vol. 32, JMLR.org, **2014**, pp. I–172–I–180.
[162] S. Green, *NVIDIA whitepaper* **2010**, *6*, 121–128.
[163] R. Hoetzlein, *GPU Technol. Conf.* **2014**.
[164] S. Páll, B. Hess, *Comput. Phys. Commun.* **2013**, *184*, 2641–2650.
[165] H. J. C. Berendsen, W. F. van Gunsteren, 'Practical Algorithms for Dynamics Simulation' in *Mol. Simulations Stat. Syst. (Enrico Fermi Summer Sch.* Varenna, **1986**, pp. 43–65.
[166] H. Gould, J. Tobochnik, W. Christian, *An Introduction to Computer Simulation Methods*, 3rd ed., **2007**.
[167] J. Wu, R. O. Watts, *J. Chem. Phys.* **1995**, *103*, 3718–3732.
[168] A. Cromer, *Am. J. Phys.* **1981**, *49*, 455–459.
[169] E. Hairer, C. Lubich, G. Wanner, *Acta Numer.* **2003**, *12*, 399–450.
[170] R. Hockney, S. Goel, J. Eastwood, *J. Comput. Phys.* **1974**, *14*, 148–158.

[171] R. A. Lippert, K. J. Bowers, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, D. E. Shaw, *J. Chem. Phys.* **2007**, *126*, 046101.

[172] D. S. Lemons, A. Gythiel, *Am. J. Phys.* **1997**, *65*, 1079–1081.

[173] N. Bou-Rabee, *Entropy* **2013**, *16*, 138–162.

[174] M. Hutzenthaler, A. Jentzen, P. E. Kloeden, *Proc. R. Soc. A Math. Phys. Eng. Sci.* **2011**, *467*, 1563–1576.

[175] J. Finkelstein, G. Fiorin, B. Seibold, *Mol. Phys.* **2020**, *118*, DOI 10.1080/00268976.2019.1649493.

[176] S. Kieninger, B. G. Keller, **2022**, DOI 2204.02105.

[177] J. E. Basconi, M. R. Shirts, *J. Chem. Theory Comput.* **2013**, *9*, 2887–2899.

[178] Z. Zhang, X. Liu, K. Yan, M. E. Tuckerman, J. Liu, *J. Phys. Chem. A* **2019**, *123*, 6056–6079.

[179] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak, *J. Chem. Phys.* **1984**, *81*, 3684–3690.

[180] G. Bussi, D. Donadio, M. Parrinello, *J. Chem. Phys.* **2007**, *126*, 014101.

[181] B. Israelewitz, J. Baudry, J. Gullingsrud, D. Kosztin, K. Schulten, *J. Mol. Graph. Model.* **2001**, *19*, 13–25.

[182] B. Israelewitz, M. Gao, K. Schulten, *Curr. Opin. Struct. Biol.* **2001**, *11*, 224–230.

[183] Y. Wang, S. A. Shaikh, E. Tajkhorshid, *Physiology* **2010**, *25*, 142–154.

[184] T. Giorgino, G. De Fabritiis, *J. Chem. Theory Comput.* **2011**, *7*, 1943–1950.

[185] W. Zhang, T. Yang, S. Zhou, J. Cheng, S. Yuan, G. V. Lo, Y. Dou, *Biomolecules* **2019**, *9*, 852.

[186] C. F. Wong, *J. Comput. Chem.* **2018**, *39*, 1307–1318.

[187] H. Grubmüller, *Methods Mol. Biol.* **2005**, *305*, 493–515.

[188] S. Park, F. Khalili-Araghi, E. Tajkhorshid, K. Schulten, *J. Chem. Phys.* **2003**, *119*, 3559–3566.

[189] G. S. Hammond, *J. Am. Chem. Soc.* **1955**, *77*, 334–338.

[190] A. Pohorille, C. Jarzynski, C. Chipot, *J. Phys. Chem. B* **2010**, *114*, 10235–10253.

[191] O. K. Dudko, G. Hummer, A. Szabo, *Proc. Natl. Acad. Sci.* **2008**, *105*, 15755–15760.

[192] F. M. Boubeta, R. M. Contestín García, E. N. Lorenzo, L. Boechi, D. Estrin, M. Sued, M. Arrar, *Chem. Biol. Drug Des.* **2019**, *93*, 1129–1138.

[193] C. C. Moore, *Proc. Natl. Acad. Sci.* **2015**, *112*, 1907–1911.

[194] A. D. McNaught, A. Wilkinson, *The IUPAC Compendium of Chemical Terminology*, 2nd ed., (Ed.: V. Gold), International Union of Pure and Applied Chemistry (IUPAC), Research Triangle Park, NC, **2019**.

[195] S. Ali, M. Hassan, A. Islam, F. Ahmad, *Curr. Protein Pept. Sci.* **2014**, *15*, 456–476.

[196] J. D. Durrant, L. Votapka, J. Sørensen, R. E. Amaro, *J. Chem. Theory Comput.* **2014**, *10*, 5047–5056.

[197] W. Kabsch, C. Sander, *Biopolymers* **1983**, *22*, 2577–2637.

[198] W. G. Touw, C. Baakman, J. Black, T. A. H. te Beek, E. Krieger, R. P. Joosten, G. Vriend, *Nucleic Acids Res.* **2015**, *43*, D364–D368.

[199] S. C. C. van der Lubbe, C. Fonseca Guerra, *Chem. – An Asian J.* **2019**, asia.201900717.

[200] I. Y. Torshin, I. T. Weber, R. W. Harrison, *Protein Eng. Des. Sel.* **2002**, *15*, 359–363.

[201] E. Baker, R. Hubbard, *Prog. Biophys. Mol. Biol.* **1984**, *44*, 97–179.

[202] P. Wernet, D. Nordlund, U. Bergmann, M. Cavalleri, M. Odelius, H. Ogasawara, L. Å. Näslund, T. K. Hirsch, L. Ojamäe, P. Glatzel, L. G. M. Pettersson, A. Nilsson, *Science* **2004**, *304*, 995–999.

[203] A. Voet, X. Qing, X. Y. Lee, J. De Raeymaecker, J. Tame, K. Zhang, M. De Maeyer, *J. Receptor. Ligand Channel Res.* **2014**, 81.

[204] M. Janežič, K. Valjavec, K. B. Loboda, B. Herlah, I. Ogris, M. Kozorog, M. Podobnik, S. G. Grdadolnik, G. Wolber, A. Perdih, *Int. J. Mol. Sci.* **2021**, *22*, 13474.

[205] O. Fleetwood, M. A. Kasimova, A. M. Westerlund, L. Delemotte, *Biophys. J.* **2020**, *118*, 765–780.

[206] D. Freedman, P. Diaconis, *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* **1981**, *57*, 453–476.

[207] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, **1986**.

[208] Z. I. Botev, J. F. Grotowski, D. P. Kroese, *Ann. Stat.* **2010**, *38*, DOI 10.1214/10-AOS799.

[209] J. M. Joyce, 'Kullback-Leibler Divergence' in *Int. Encycl. Stat. Sci.* Springer Berlin Heidelberg, Berlin, Heidelberg, **2011**, pp. 720–722.

[210] J. Gorodkin, *Comput. Biol. Chem.* **2004**, *28*, 367–374.

[211] Y. Wang, Y. Li, H. Cao, M. Xiong, Y. Y. Shugart, L. Jin, *BMC Bioinformatics* **2015**, *16*, 260.

[212] B. Efron, R. J. Tibshirani, *An Introduction to the Bootstrap*, Chapman & Hall, Boca Raton, **1993**.

[213] A. Altis, P. H. Nguyen, R. Hegger, G. Stock, *J. Chem. Phys.* **2007**, *126*, 244111.

[214] M. K. Scherer, B. Trendelkamp-Schroer, F. Paul, G. Pérez-Hernández, M. Hoffmann, N. Plattner, C. Wehmeyer, J. H. Prinz, F. Noé, *J. Chem. Theory Comput.* **2015**, *11*, 5525–5542.

[215] L. Molgedey, H. G. Schuster, *Phys. Rev. Lett.* **1994**, *72*, 3634–3637.

[216] G. Pérez-Hernández, F. Paul, T. Giorgino, G. De Fabritiis, F. Noé, *J. Chem. Phys.* **2013**, *139*, 15102.

[217] C. R. Schwantes, V. S. Pande, *J. Chem. Theory Comput.* **2013**, *9*, 2000–2009.

[218] F. Noé, C. Clementi, *J. Chem. Theory Comput.* **2015**, *11*, 5002–5011.

[219] H. Wu, F. Noé, *J. Nonlinear Sci.* **2020**, *30*, 23–66.

[220] S. Schultze, H. Grubmüller, *J. Chem. Theory Comput.* **2021**, *17*, 5766–5776.

[221] C. R. Schwantes, V. S. Pande, *J. Chem. Theory Comput.* **2015**, *11*, 600–608.

[222] L. van der Maaten, G. Hinton, *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.

[223] V. Spiwok, P. Kříž, *Front. Mol. Biosci.* **2020**, *7*, DOI 10.3389/fmolb.2020.00132.

[224] C. E. Shannon, *Bell Syst. Tech. J.* **1948**, *27*, 379–423.

[225] J. Duda, K. Tahboub, N. J. Gadgil, E. J. Delp in 2015 Pict. Coding Symp. IEEE, **2015**, pp. 65–69.

[226] E. T. Jaynes, *Am. J. Phys.* **1965**, *33*, 391–398.

[227] G. R. Bowman, P. L. Geissler, *Proc. Natl. Acad. Sci.* **2012**, *109*, 11681–11686.

[228] M. S. Roulston, *Phys. D Nonlinear Phenom.* **1999**, *125*, 285–294.

[229] K. H. DuBay, J. P. Bothma, P. L. Geissler, *PLoS Comput. Biol.* **2011**, *7*, (Ed.: E. I. Shakhnovich), e1002168.

[230] J. Bernoulli, *Wahrscheinlichkeitsrechnung (Ars conjectandi, 1713)*, (Ed.: R. Haussner), Wilhelm Engelmann, **1899**.

[231] A. A. Markov, *Sci. Context* **2006**, *19*, 591–600.

[232] P. A. Gagniuc, *Markov Chains: From Theory to Implementation and Experimentation*, 1st ed., John Wiley & Sons, Hoboken, **2017**.

[233] W. C. Swope, J. W. Pitera, F. Suits, *J. Phys. Chem. B* **2004**, *108*, 6571–6581.

[234] N. Singhal, C. D. Snow, V. S. Pande, *J. Chem. Phys.* **2004**, *121*, 415.

[235] M. Sarich, J.-H. Prinz, C. Schütte, 'Markov Model Theory' in **2014**, pp. 23–44.

[236] F. Noé, F. Nüske, *Multiscale Model. Simul.* **2013**, *11*, 635–655.

[237] F. Nüske, B. G. Keller, G. Pérez-Hernández, A. S. J. S. Mey, F. Noé, *J. Chem. Theory Comput.* **2014**, *10*, 1739–1752.

[238] G. R. Bowman, V. S. Pande, F. Noé, *An Introduction to Markov State Models and Their Application to Long Timescale Molecular Simulation*, (Eds.: G. R. Bowman, V. S. Pande, F. Noé), Springer Netherlands, Dordrecht, **2014**.

[239] P. G. Bolhuis, D. Chandler, C. Dellago, P. L. Geissler, *Annu. Rev. Phys. Chem.* **2002**, *53*, 291–318.

[240] O. Lemke, B. G. Keller, *J. Chem. Phys.* **2016**, *145*, 164104.

[241] M. Sarich, C. Schütte, *Commun. Math. Sci.* **2012**, *10*, 1001–1013.

[242] C. Schütte, F. Noé, J. Lu, M. Sarich, E. Vanden-Eijnden, C. Schütte, F. Noé, J. Lu, M. Sarich, E. Vanden-Eijnden, *J. Chem. Phys.* **2011**, *134*, DOI 10.1063/1.3590108.

[243] J. H. Peng, W. Wang, Y. Q. Yu, H. L. Gu, X. Huang, *Chinese J. Chem. Phys.* **2018**, *31*, 404–420.

[244] D. K. Wolfe, J. R. Persichetti, A. K. Sharma, P. S. Hudson, H. L. Woodcock, E. P. O'Brien, *J. Chem. Theory Comput.* **2020**, *16*, 1816–1826.

[245] J.-H. Prinz, J. D. Chodera, F. Noé, 'Estimation and Validation of Markov Models' in **2014**, pp. 45–60.

[246] F. Nüske, H. Wu, J.-H. Prinz, C. Wehmeyer, C. Clementi, F. Noé, *J. Chem. Phys.* **2017**, *146*, 094104.

[247] B. Cooke, S. C. Schmidler, *J. Chem. Phys.* **2008**, *129*, 164112.

[248] B. E. Husic, V. S. Pande, *J. Am. Chem. Soc.* **2018**, *140*, 2386–2396.

[249] F. Noé, E. Rosta, *J. Chem. Phys.* **2019**, *151*, 190401.

[250] R. Shields, *Theory Cult. Soc.* **2012**, *29*, 43–57.

[251] E. Cayley, *Berichte der Dtsch. Chem. Gesellschaft* **1875**, *8*, 1056–1059.

[252] *Network Analysis, Methodological Foundations*, (Eds.: U. Brandes, T. Erlebach), Springer, **2005**.

[253] M. Needham, A. E. Hodler, *Graph algorithms, Practical Examples in Apache Spark & Neo4j*, 1st ed., O'Reilly, Boston, **2019**.

[254] R. Sedgewick, K. Wayne, *Algorithms*, 4th ed., Addison-Wesley Professional, **2011**.

[255] G. Zanotti, L. Falcigno, M. Saviano, G. D. Auria, B. M. Bruno, T. Campanile, L. Paolillo, *Chem. Eur. J.* **2001**, *7*, 1479–1485.

[256] F. Lynen, U. Wieland, *Justus Liebig's Ann. der Chemie* **1938**, *533*, 93–117.

[257] J. A. Swain, S. R. Walker, M. B. Calvert, M. A. Brimble, *Nat. Prod. Rep.* **2022**, *39*, 410–443.

[258] T. Wieland, *Peptides of Poisonous Amanita Mushrooms*, De Gruyter, **1987**.

[259] B. Le Daré, P.-J. Ferron, A. Couette, C. Ribault, I. Morel, T. Gicquel, *Toxicol. Lett.* **2021**, *346*, 1–6.

[260] B. Le Daré, P.-J. Ferron, T. Gicquel, *Toxins (Basel).* **2021**, *13*, 417.

[261] J. Vetter, *Toxicon* **1998**, *36*, 13–24.

[262] T. Wieland, *Int. J. Pept. Protein Res.* **2009**, *22*, 257–276.

[263] A. Kumari, S. Kesarwani, M. G. Javoor, K. R. Vinothkumar, M. Sirajuddin, *EMBO J.* **2020**, *39*, DOI 10.15252/embj.2019104006.

[264] G. H. Kim, T. A. Klotchkova, B.-C. Lee, S. H. Kim, *Bot. Mar.* **2001**, *44*, 501–508.

[265] C. H. Scott Chialvo, L. H. Griffin, L. K. Reed, L. Ciesla, *Ecol. Evol.* **2020**, *10*, 4233–4240.

[266] L. Wang, N. Wang, W. Zhang, X. Cheng, Z. Yan, G. Shao, X. Wang, R. Wang, C. Fu, *Signal Transduct. Target. Ther.* **2022**, *7*, 48.

[267] I. Petta, S. Lievens, C. Libert, J. Tavernier, K. De Bosscher, *Mol. Ther.* **2016**, *24*, 707–718.

[268] J.-S. Choi, S. H. Joo, *Biomol. Ther. (Seoul).* **2020**, *28*, 18–24.

[269] A. F. B. Räder, M. Weinmüller, F. Reichart, A. Schumacher-Klinger, S. Merzbach, C. Gilon, A. Hoffman, H. Kessler, *Angew. Chemie Int. Ed.* **2018**, *57*, 14414–14438.

[270] E. H. M. Mohammed, D. Mandal, S. Mozaffari, M. Abdel-Hamied Zahran, A. Mostafa Osman, R. Kumar Tiwari, K. Parang, *Molecules* **2020**, *25*, 2581.

[271] A. G. Jamieson, N. Boutard, D. Sabatino, W. D. Lubell, *Chem. Biol. Drug Des.* **2013**, *81*, 148–165.

[272] J. Witek, B. G. Keller, M. Blatter, A. Meissner, T. Wagner, S. Riniker, *J. Chem. Inf. Model.* **2016**, *56*, 1547–1562.

[273] J. Witek, M. Mühlbauer, B. G. Keller, M. Blatter, A. Meissner, T. Wagner, S. Riniker, *ChemPhysChem* **2017**, *18*, 3309–3314.

[274] J. Witek, S. Wang, B. Schroeder, R. Lingwood, A. Dounas, H.-J. Roth, M. Fouché, M. Blatter, O. Lemke, B. Keller, S. Riniker, *J. Chem. Inf. Model.* **2019**, *59*, 294–308.

[275] M. O. Anderson, A. A. Shelat, R. Kiplin Guy, *J. Org. Chem.* **2005**, *70*, 4578–4584.

[276] G. Yao, J.-O. Joswig, B. G. Keller, R. D. Süssmuth, *Chem. Eur. J.* **2019**, *25*, 8030–8034.

[277] L. A. Schuresko, R. S. Lokey, *Angew. Chemie - Int. Ed.* **2007**, *46*, 3547–3549.

[278] T. Wieland, T. Miura, A. Seeliger, *Int. J. Pept. Protein Res.* **1983**, *21*, 3–10.

[279] L. Falcigno, S. Costantini, G. D'Auria, B. M. Bruno, S. Zobeley, G. Zanotti, L. Paolillo, *Chem. - A Eur. J.* **2001**, *7*, 4665–4673.

[280] A. Blanc, M. Todorovic, D. M. Perrin, *Chem. Commun.* **2019**, *55*, 385–388.

[281] M. Döntgen, M.-D. Przybylski-Freund, L. C. Kröger, W. A. Kopp, A. E. Ismail, K. Leonhard, *J. Chem. Theory Comput.* **2015**, *11*, 2517–2524.

[282] R. Iftimie, P. Minary, M. E. Tuckerman, *Proc. Natl. Acad. Sci.* **2005**, *102*, 6654–6659.

[283] R. Brückner, *Reaktionsmechanismen*, 3rd ed., Spektrum, **2004**.

[284] G. Yao, C. H. Knittel, S. Kosol, M. T. Wenz, B. G. Keller, H. Gruß, A. C. Braun, C. Lutz, T. Hechler, A. Pahl, R. D. Süssmuth, *J. Am. Chem. Soc.* **2021**, *143*, 14322–14331.

[285] J. Banchereau, R. M. Steinman, *Nature* **1998**, *392*, 245–252.

[286] R. M. Steinman, *Annu. Rev. Immunol.* **2012**, *30*, 1–22.

[287] T. A. Patente, M. P. Pinho, A. A. Oliveira, G. C. M. Evangelista, P. C. Bergami-Santos, J. A. M. Barbuto, *Front. Immunol.* **2019**, *9*, DOI 10.3389/fimmu.2018.03176.

[288] B. Alberts, J. Johnson, Alexander Lewis, M. Raff, K. Roberts, P. Walter, *Molecular Biology of the Cell*, 4th ed., Garland Science, **2002**.

[289] T. B. H. Geijtenbeek, S. I. Gringhuis, *Nat. Rev. Immunol.* **2009**, *9*, 465–479.

[290] M. Bermejo-Jambrina, J. Eder, L. C. Helgers, N. Hertoghs, B. M. Nijmeijer, M. Stunnenberg, T. B. H. Geijtenbeek, *Front. Immunol.* **2018**, *9*, DOI 10.3389/fimmu.2018.00590.

[291] P. Langerhans, *Arch. für Pathol. Anat. und Physiol. und für Klin. Med.* **1868**, *44*, 325–337.

[292] T. Doebel, B. Voisin, K. Nagao, *Trends Immunol.* **2017**, *38*, 817–828.

[293] K. L. McClain, C. Bigenwald, M. Collin, J. Haroche, R. A. Marsh, M. Merad, J. Picarsic, K. B. Ribeiro, C. E. Allen, *Nat. Rev. Dis. Prim.* **2021**, *7*, 73.

[294] A. N. Zelensky, J. E. Gready, *Proteins Struct. Funct. Genet.* **2003**, *52*, 466–477.

[295] A. N. Zelensky, J. E. Gready, *FEBS J.* **2005**, *272*, 6179–6217.

[296] S. A. McMahon, J. L. Miller, J. A. Lawton, D. E. Kerkow, A. Hodes, M. A. Marti-Renom, S. Doulatov, E. Narayanan, A. Sali, J. F. Miller, P. Ghosh, *Nat. Struct. Mol. Biol.* **2005**, *12*, 886–892.

[297] J. Aretz, E.-C. Wamhoff, J. Hanske, D. Heymann, C. Rademacher, *Front. Immunol.* **2014**, *5*, DOI 10.3389/fimmu.2014.00323.

[298] J. Valladeau, V. Duvert-Frances, J.-J. J. Pin, C. Dezutter-Dambuyant, C. Vincent, C. Massacrier, J. Vincent, K. Yoneda, J. Banchereau, C. Caux, J. Davoust, S. Saeland, *Eur. J. Immunol.* **1999**, *29*, 2695–2704.

[299] J. Valladeau, O. Ravel, C. Dezutter-Dambuyant, K. Moore, M. Kleijmeer, Y. Liu, V. Duvert-Frances, C. Vincent, D. Schmitt, J. Davoust, C. Caux, S. Lebecque, S. Saeland, *Immunity* **2000**, *12*, 71–81.

[300] A. B. Palmos, V. Millischer, D. K. Menon, T. R. Nicholson, L. S. Taams, B. Michael, G. Sunderland, M. J. Griffiths, C. Hübel, G. Breen, *PLOS Genet.* **2022**, *18*, (Ed.: C. Cotsapas), e1010042.

[301] I. Trbojević-Akmačić, T. Petrović, G. Lauc, *Glycoconj. J.* **2021**, *38*, 611–623.

[302] M. Thépaut, J. Luczkowiak, C. Vivès, N. Labiod, I. Bally, F. Lasala, Y. Grimoire, D. Fenel, S. Sattin, N. Thielens, G. Schoehn, A. Bernardi, R. Delgado, F. Fieschi, *PLOS Pathog.* **2021**, *17*, (Ed.: A. Pekosz), e1009576.

[303] M. Bermejo-Jambrina, J. Eder, T. M. Kaptein, J. L. Hamme, L. C. Helgers, K. E. Vlaming, P. J. M. Brouwer, A. C. Nuenen, M. Spaargaren, G. J. Bree, B. M. Nijmeijer, N. A. Kootstra, M. J. Gils, R. W. Sanders, T. B. H. Geijtenbeek, *EMBO J.* **2021**, *40*, DOI 10.15252/embj.2020106765.

[304] P. Verdecchia, C. Cavallini, A. Spanevello, F. Angeli, *Eur. J. Intern. Med.* **2020**, *76*, 14–20.

[305] R. A. Botting, H. Rana, K. M. Bertram, J. W. Rhodes, H. Baharlou, N. Nasr, A. L. Cunningham, A. N. Harman, *Rev. Med. Virol.* **2017**, *27*, e1923.

[306] L. de Witte, A. Nabatov, M. Pion, D. Fluitsma, M. A. W. P. de Jong, T. de Gruijl, V. Piguet, Y. van Kooyk, T. B. H. Geijtenbeek, *Nat. Med.* **2007**, *13*, 367–371.

[307] R. Mc Dermott, U. Ziylan, D. Spehner, H. Bausinger, D. Lipsker, M. Mommaas, J.-P. Cazenave, G. Raposo, B. Goud, H. de la Salle, J. Salamero, D. Hanau, *Mol. Biol. Cell* **2002**, *13*, (Ed.: S. R. Pfeffer), 317–335.

[308] C. M. S. Ribeiro, R. Sarrami-Forooshani, L. C. Setiawan, E. M. Zijlstra-Willems, J. L. van Hamme, W. Tigchelaar, N. N. van der Wel, N. A. Kootstra,

S. I. Gringhuis, T. B. H. Geijtenbeek, *Nature* **2016**, *540*, 448–452.

[309] C. B. Wilen, J. C. Tilton, R. W. Doms, *Cold Spring Harb. Perspect. Med.* **2012**, *2*, a006866–a006866.

[310] N. Nasr, J. Lai, R. A. Botting, S. K. Mercier, A. N. Harman, M. Kim, S. Turville, R. J. Center, T. Domagala, P. R. Gorry, N. Olbourne, A. L. Cunningham, *J. Immunol.* **2014**, *193*, 2554–2564.

[311] C. M. Ribeiro, R. Sarrami-Forooshani, T. B. Geijtenbeek, *Future Virol.* **2015**, *10*, 1231–1243.

[312] K. M. Bertram, O. Tong, C. Royle, S. G. Turville, N. Nasr, A. L. Cunningham, A. N. Harman, *Front. Immunol.* **2019**, *10*, DOI 10.3389/fimmu.2019.02263.

[313] L. M. van den Berg, S. Cardinaud, A. M. G. van der Aar, J. K. Sprokholt, M. A. W. P. de Jong, E. M. Zijlstra-Willems, A. Moris, T. B. H. Geijtenbeek, *J. Immunol.* **2015**, *195*, 1763–1773.

[314] B. M. Nijmeijer, M. Bermejo-Jambrina, T. M. Kaptein, C. M. S. Ribeiro, D. Wilflingseder, T. B. H. Geijtenbeek, *Mucosal Immunol.* **2021**, *14*, 743–750.

[315] E. E. Vine, J. W. Rhodes, F. A. Warner van Dijk, S. N. Byrne, K. M. Bertram, A. L. Cunningham, A. N. Harman, *Mucosal Immunol.* **2022**, DOI 10.1038/s41385-022-00492-0.

[316] W. C. Ng, S. L. Londrigan, N. Nasr, A. L. Cunningham, S. Turville, A. G. Brooks, P. C. Reading, *J. Virol.* **2015**, *90*, 206–221.

[317] J.-O. Joswig, Master Thesis, Freie Universität Berlin, **2017**.

[318] J. Cramer, *RSC Med. Chem.* **2021**, *12*, 1985–2000.

[319] V. Porkolab, E. Chabrol, N. Varga, S. Ordanini, I. Sutkevičiute, M. Thépaut, M. J. García-Jiménez, E. Girard, P. M. Nieto, A. Bernardi, F. Fieschi, *ACS Chem. Biol.* **2018**, *13*, 600–608.

[320] R.-J. E. Li, T. P. Hogervorst, S. Achilli, S. C. M. Bruijns, S. Spiekstra, C. Vivès, M. Thépaut, D. V. Filippov, G. A. van der Marel, S. J. van Vliet, F. Fieschi, J. D. C. Codée, Y. van Kooyk, *Front. Cell Dev. Biol.* **2020**, *8*, DOI 10.3389/fcell.2020.00556.

[321] M. Rentzsch, R. Wawrzinek, C. Zelle-Rieser, H. Strandt, L. Bellmann, F. F. Fuchsberger, J. Schulze, J. Busmann, J. Rademacher, S. Sigl, B. Del Frari, P. Stoitzner, C. Rademacher, *Front. Immunol.* **2021**, *12*, DOI 10.3389/fimmu.2021.732298.

[322] J. Aretz, H. Baukmann, E. Shanina, J. Hanske, R. Wawrzinek, V. A. Zapol'skii, P. H. Seeberger, D. E. Kaufmann, C. Rademacher, *Angew. Chemie Int. Ed.* **2017**, *56*, 7292–7296.

[323] R. Wawrzinek, E.-C. Wamhoff, J. Lefebre, M. Rentzsch, G. Bachem, G. Domeniconi, J. Schulze, F. F. Fuchsberger, H. Zhang, C. Modenutti, L. Schnirch, M. A. Marti, O. Schwardt, M. Bräutigam, M. Guberman, D. Hauck, P. H. Seeberger, O. Seitz, A. Titz, B. Ernst, C. Rademacher, *J. Am. Chem. Soc.* **2021**, *143*, 18977–18988.

[324] J. Aretz, U. R. Anumala, F. F. Fuchsberger, N. Molavi, N. Ziebart, H. Zhang, M. Nazaré, C. Rademacher, *J. Am. Chem. Soc.* **2018**, *140*, 14915–14925.

[325] B. G. Keller, C. Rademacher, *Curr. Opin. Struct. Biol.* **2020**, *62*, 31–38.

[326] S. Hertig, N. R. Latorraca, R. O. Dror, *PLOS Comput. Biol.* **2016**, *12*, 1–16.

[327] G. B. Cohen, R. Ren, D. Baltimore, *Cell* **1995**, *80*, 237–248.

[328] H. Feinberg, A. S. Powlesland, M. E. Taylor, W. I. Weis, *J. Biol. Chem.* **2010**, *285*, 13285–13293.

[329] N. S. Stambach, M. E. Taylor, *Glycobiology* **2003**, *13*, 401–410.

[330] M. Thépaut, J. Valladeau, A. Nurisso, R. Kahn, B. Arnou, C. Vivès, S. Saeland, C. Ebel, C. Monnier, C. Dezutter-Dambuyant, A. Imberty, F. Fieschi, *Biochemistry* **2009**, *48*, 2684–2698.

[331] W. I. Weis, K. Drickamer, W. A. Hendrickson, *Nature* **1992**, *360*, 127–134.

[332] H. Feinberg, M. E. Taylor, N. Razi, R. McBride, Y. A. Knirel, S. A. Graham, K. Drickamer, W. I. Weis, *J. Mol. Biol.* **2011**, *405*, 1027–1039.

[333] A. Holla, A. Skerra, *Protein Eng. Des. Sel.* **2011**, *24*, 659–669.

[334] J. C. Muñoz-García, E. Chabrol, R. R. Vivès, A. Thomas, J. L. de Paz, J. Rojo, A. Imberty, F. Fieschi, P. M. Nieto, J. Angulo, *J. Am. Chem. Soc.* **2015**, *137*, 4100–4110.

[335] J. Zhao, X. Liu, C. Kao, E. Zhang, Q. Li, F. Zhang, R. J. Linhardt, *Biochemistry* **2016**, *55*, 4552–4559.

[336] J. Hanske, J. Schulze, J. Aretz, R. McBride, B. Loll, H. Schmidt, Y. Knirel, W. Rabsch, M. C. Wahl, J. C. Paulson, C. Rademacher, *J. Biol. Chem.* **2017**, *292*, 862–871.

[337] J. Hanske, S. Aleksić, M. Ballaschk, M. Jurk, E. Shanina, M. Beerbaum, P. Schmieder, B. G. Keller, C. Rademacher, *J. Am. Chem. Soc.* **2016**, *138*, 12176–12186.

[338] T. Onizuka, H. Shimizu, Y. Moriwaki, T. Nakano, S. Kanai, I. Shimada, H. Takahashi, *FEBS J.* **2012**, *279*, 2645–2656.

[339] *Cell Physiology Source Book*, (Ed.: N. Sperelakis), Elsevier, **2012**.

[340] J. L. Goldstein, M. S. Brown, R. G. W. Anderson, D. W. Russell, W. J. Schneider, *Annu. Rev. Cell Biol.* **1985**, *1*, 1–39.

[341] A. Sorkin, M. von Zastrow, *Nat. Rev. Mol. Cell Biol.* **2002**, *3*, 600–614.

[342] J. V. Gerasimenko, A. V. Tepikin, O. H. Petersen, O. V. Gerasimenko, *Curr. Biol.* **1998**, *8*, 1335–1338.

[343] T. Gramberg, E. Soilleux, T. Fisch, P. F. Lalor, H. Hofmann, S. Wheeldon, A. Cotterill, A. Wegele, T. Winkler, D. H. Adams, S. Pöhlmann, *Virology* **2008**, *373*, 189–201.

[344] B. Garcia-Moreno, *J. Biol.* **2009**, *8*, 98.

[345] N. Romani, B. E. Clausen, P. Stoitzner, *Immunol. Rev.* **2010**, *234*, 120–141.

[346] P. J. Cullen, F. Steinberg, *Nat. Rev. Mol. Cell Biol.* **2018**, *19*, 679–696.

[347] J.-O. Joswig, J. Anders, H. Zhang, C. Rademacher, B. G. Keller, *J. Biol. Chem.* **2021**, *296*, 100718.

[348] N. V. Di Russo, M. A. Martí, A. E. Roitberg, *J. Phys. Chem. B* **2014**, *118*, 12818–12826.

[349] E. D. Kim, C. D. Kim, J. Chaney, S. Kim, 'Protein Function | Allostery in Proteins: Canonical Models and New Insights' in *Encycl. Biol. Chem. III*, Elsevier, **2021**, pp. 27–43.

[350] A. Cooper, D. T. F. Dryden, *Eur. Biophys. J.* **1984**, *11*, 103–109.

[351] C.-j. Tsai, A. del Sol, R. Nussinov, *J. Mol. Biol.* **2008**, *378*, 1–11.

[352] R. Nussinov, C. J. Tsai, Allostery without a conformational change? Revisiting the paradigm, **2015**.

[353] A. Basit, R. K. Mishra, P. Bandyopadhyay, *J. Biomol. Struct. Dyn.* **2021**, *39*, 7213–7222.

[354] Q. Tan, Y. Ding, Z. Qiu, J. Huang, *ACS Phys. Chem. Au* **2022**, *2*, 143–155.

[355] C. R. Søndergaard, M. H. M. Olsson, M. Rostkowski, J. H. Jensen, *J. Chem. Theory Comput.* **2011**, *7*, 2284–2295.

[356] M. H. M. Olsson, C. R. Søndergaard, M. Rostkowski, J. H. Jensen, *J. Chem. Theory Comput.* **2011**, *7*, 525–537.

[357] P. Dobrev, S. P. B. Vemulapalli, N. Nath, C. Griesinger, H. Grubmüller, *J. Chem. Theory Comput.* **2020**, *16*, 2561–2569.

[358] N. Pinotsis, K. Zielinska, M. Babuta, J. L. Arolas, J. Kostan, M. B. Khan, C. Schreiner, A. Salmazo, L. Ciccarelli, M. Puchinger, E. A. Gkougkoulia, E. d. A. Ribeiro, T. C. Marlovits, A. Bhattacharya, K. Djinovic-Carugo, *Proc. Natl. Acad. Sci.* **2020**, *117*, 22101–22112.

[359] V. J. Hilser, J. O. Wrabl, H. N. Motlagh, *Annu. Rev. Biophys.* **2012**, *41*, 585–609.

[360] C. J. Tsai, R. Nussinov, *PLOS Comput. Biol.* **2014**, *10*, e1003394.

[361] M. A. Cuendet, H. Weinstein, M. V. LeVine, *J. Chem. Theory Comput.* **2016**, *12*, 5758–5767.

[362] P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern, R. P. Wiewiora, B. R. Brooks, V. S. Pande, *PLOS Comput. Biol.* **2017**, *13*, (Ed.: R. Gentleman), e1005659.

[363] J. Huang, S. Rauscher, G. Nawrocki, T. Ran, M. Feig, B. L. de Groot, H. Grubmüller, A. D. MacKerell, *Nat. Methods* **2017**, *14*, 71–73.

[364] Z. Jing, R. Qi, C. Liu, P. Ren, *J. Chem. Phys.* **2017**, *147*, 161733.

[365] D. Jiao, C. King, A. Grossfield, T. A. Darden, P. Ren, *J. Phys. Chem. B* **2006**, *110*, 18553–18559.

[366] S. Izadi, A. V. Onufriev, *J. Chem. Phys.* **2016**, *145*, 074501.

[367] P. Ren, J. W. Ponder, *J. Phys. Chem.* **2003**, *107*, 5933–5947.

[368] M. L. Laury, L.-P. Wang, V. S. Pande, T. Head-Gordon, J. W. Ponder, *J. Phys. Chem. B* **2015**, *119*, 9423–9437.

[369] A. K. Das, O. N. Demerdash, T. Head-Gordon, *J. Chem. Theory Comput.* **2018**, *14*, 6722–6733.

[370] E. Lambros, F. Paesani, *J. Chem. Phys.* **2020**, *153*, 060901.

[371] N. Prabhu, K. Sharp, *Chem. Rev.* **2006**, *106*, 1616–1623.

[372] E. Lindahl, B. Hess, D. van der Spoel, *J. Mol. Model.* **2001**, *7*, 306–317.

[373] D. van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, H. J. Berendsen, *J. Comput. Chem.* **2005**, *26*, 1701–1718.

[374] B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, *J. Chem. Theory Comput.* **2008**, *4*, 435–447.

[375] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, E. Lindahl, *Bioinformatics* **2013**, *29*, 845–854.

[376] S. Páll, M. J. Abraham, C. Kutzner, B. Hess, E. Lindahl, 'Tackling Exascale Software Challenges in Molecular Dynamics Simulations with GROMACS' in *Solving Softw. Challenges Exascale. EASC 2014. Lect. Notes Comput. Sci. Vol. 8759*, (Eds.: S. Markidis, E. Laure), Springer, Cham, **2015**, pp. 3–27.

[377] W. Humphrey, A. Dalke, K. Schulten, *J. Mol. Graph.* **1996**, *14*, 33–38.

[378] H. Nguyen, D. A. Case, A. S. Rose, *Bioinformatics* **2018**, *34*, (Ed.: A. Valencia), 1241–1242.

[379] C. A. Terry, M.-J. Fernández, L. Gude, A. Lorente, K. B. Grant, *Biochemistry* **2011**, *50*, 10375–10389.

[380] D. C. Liu, J. Nocedal, *Math. Program.* **1989**, *45*, 503–528.

[381] M. Bernetti, G. Bussi, *J. Chem. Phys.* **2020**, *153*, 114107.

[382] M. Gaertler, 'Clustering' in *Netw. Anal. Methodol. Found.* (Eds.: U. Brandes, T. Erlebach), Springer-Verlag, **2005**, pp. 178–215.

[383] *Handbook of Cluster Analysis*, (Eds.: C. Henning, M. Meila, F. Murtagh, R. Rocci), CRC Press, **2016**.

[384] V. Estivill-Castro, *ACM SIGKDD Explor. Newsl.* **2002**, *4*, 65–75.

[385] M. Aldenderfer, R. Blashfield, *Cluster Analysis*, SAGE Publications, Inc., Thousand Oaks, CA, **1984**.

[386] M. Sips, 'Visual Clustering' in *Encycl. Database Syst.* (Eds.: L. Liu, M. T. Özsu), Springer, Boston, MA, **2009**.

[387] J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, San Mateo, CA, **2000**.

[388] J. Snow, *On the mode of communication of cholera*, 2nd ed., John Churchill, London, **1855**.

[389] R. R. Sokal, *Taxon* **1963**, *12*, 190–199.

[390] M. E. Karpen, D. J. Tobias, C. L. Brooks, *Biochemistry* **1993**, *32*, 412–420.

[391] P. S. Shenkin, D. Q. McDonald, *J. Comput. Chem.* **1994**, *15*, 899–916.

[392] A. E. Torda, W. F. van Gunsteren, *J. Comput. Chem.* **1994**, *15*, 1331–1340.

[393] R. M. Cormack, *J. R. Stat. Soc. Ser. A* **1971**, *134*, 321.

[394] A. K. Jain, *Pattern Recognit. Lett.* **2010**, *31*, 651–666.

[395] C. C. Aggarwal, C. K. Reddy, *Data Clustering Algorithms and Applications*, CRC Press, **2014**.

[396] T. Finley, T. Joachims in Proc. 22nd Int. Conf. Mach. Learn. - ICML '05, ACM Press, New York, New York, USA, **2005**, pp. 217–224.

[397] OECD, *OECD Glossary of Statistical Terms*, OECD Publishing, **2008**.

[398] UNECE, 'Terminology on Statistical Metadata' in *Conf. Eur. Stat. Stat. Stand. Stud.* No. 53, **2000**.

[399] IBM Corporation, z/OS Basic Skills, https://www.ibm.com/docs/en/zos-basic-skills?topic=more-what-is-data-set.

[400] R. A. Fisher, *Annu. Eugen.* **1936**, *7*, 179–188.

[401] J. C. Gower, *Biometrics* **1971**, *27*, 857–871.

[402] S. Bishnoi, B. Hooda, *Int. J. Chem. Stud.* **2020**, *8*, 338–343.

[403] A. Jain, A. Topchy, M. Law, J. Buhmann in Proc. 17th Int. Conf. Pattern Recognition, 2004. ICPR 2004. IEEE, **2004**, 260–263 Vol.1.

[404] J. W. Carmichael, R. S. Julius, *Syst. Biol.* **1968**, *17*, 144–150.

[405] H.-P. Kriegel, E. Schubert, A. Zimek, *Knowl. Inf. Syst.* **2017**, *52*, 341–378.

[406] R. Graham, P. Hell, *IEEE Ann. Hist. Comput.* **1985**, *7*, 43–57.

[407] G. N. Lance, W. T. Williams, *Comput. J.* **1967**, *9*, 373–380.

[408] G. N. Lance, W. T. Williams, *Comput. J.* **1967**, *10*, 271–277.

[409] D. Krznaric, C. Levcopoulos, *Theor. Comput. Sci.* **2002**, *286*, 139–149.

[410] R. Sibson, *Comput. J.* **1973**, *16*, 30–34.

[411] J. C. Gower, G. J. S. Ross, *Appl. Stat.* **1969**, *18*, 54.

[412] L. Kaufman, P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, **2009**.

[413] M. Stoer, F. Wagner, *J. ACM* **1997**, *44*, 585–591.

[414] J. Shi, J. Malik, *IEEE Trans. Pattern Anal. Mach. Intell.* **2000**, *22*, 888–905.

[415] U. von Luxburg, *Stat. Comput.* **2007**, *17*, 395–416.

[416] S. Arora, S. Rao, U. Vazirani, *J. ACM* **2009**, *56*, 1–37.

[417] M. Weber, A. Rungsarityotin, Wasinee Schliep, *ZIB-Report* **2004**.

[418] P. Deuflhard, M. Weber, *Linear Algebra Appl.* **2005**, *398*, 161–184.

[419] S. Röblitz, M. Weber, *Adv. Data Anal. Classif.* **2013**, *7*, 147–179.

[420] H.-H. Bock, *J. Électronique d'Histoire des Probab. la Stat. [electronic only]* **2008**, *4*, Article 14, 18 p., electronic only–Article 14, 18.

[421] E. Schubert, P. J. Rousseeuw, *Inf. Syst.* **2021**, *101*, 101804.

[422] M. Mahajan, P. Nimbhorkar, K. Varadarajan, *Theor. Comput. Sci.* **2012**, *442*, 13–21.

[423] S. Lloyd, *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137.

[424] A. Tarsitano, *Pattern Recognit.* **2003**, *36*, 2955–2966.

[425] M. E. Celebi, H. A. Kingravi, P. A. Vela, *Expert Syst. Appl.* **2013**, *40*, 200–210.

[426] D. Arthur, S. Vassilvitskii, *Proc. 18th Annu. ACM-SIAM Symp. Discret. algorithms* **2007**, *8*, 1027–1035.

[427] P. J. Rousseeuw, *J. Comput. Appl. Math.* **1987**, *20*, 53–65.

[428] T. Calinski, J. Harabasz, *Commun. Stat. - Theory Methods* **1974**, *3*, 1–27.

[429] A. Rosenberg, J. Hirschberg in Proc. 2007 Jt. Conf. Empir. Methods Nat. Lang. Process. Comput. Nat. Lang. Learn. Association for Computational Linguistics, Prague, Czech Republic, **2007**, pp. 410–420.

[430] D. Steinley, *Psychol. Methods* **2004**, *9*, 386–396.

[431] N. X. Vinh, J. Epps, J. Bailey, *J. Mach. Learn. Res.* **2010**, *11*, 2837–2854.

[432] G. McLachlan, D. Peel, *Finite Mixture Models*, John Wiley & Sons, Hoboken, **2000**.

[433] A. P. Dempster, N. M. Laird, D. B. Rubin, *J. R. Stat. Soc. Ser. B* **1977**, *39*, 1–38.

[434] L. Scrucca, M. Fop, T. B. Murphy, A. E. Raftery, *R J.* **2016**, *8*, 289–317.

[435] A. Guttman, *ACM SIGMOD Rec.* **1984**, *14*, 47–57.

[436] W. Wang, J. Yang, R. R. Muntz in VLDB, **1997**.

[437] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, *ACM SIGMOD Rec.* **1998**, *27*, 94–105.

[438] J. A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, New York, **1975**.

[439] W. Stuetzle, R. Nugent, *J. Comput. Graph. Stat.* **2010**, *19*, 397–418.

[440] I. Steinwart in Proc. 24th Annu. Conf. Learn. Theory, (Eds.: S. M. Kakade, U. von Luxburg), PMLR, Budapest, Hungary, **2011**, pp. 703–738.

[441] K. Chaudhuri, S. Dasgupta in Adv. Neural Inf. Process. Syst. *Vol. 23*, (Eds.: J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, A. Culotta), Curran Associates, Inc., **2010**.

[442] J. Klemelä, *WIREs Comput. Stat.* **2018**, *10*, DOI 10.1002/wics.1436.

[443] J. A. Hartigan, *J. Am. Stat. Assoc.* **1981**, *76*, 388–394.

[444] W. Stuetzle, *J. Classif.* **2003**, *20*, 25–47.

[445] A. Azzalini, G. Menardi, *J. Stat. Softw.* **2014**, *57*, DOI 10.18637/jss.v057.i11.

[446] B. P. Kent, A. Rinaldo, T. Verstynen, **2013**, DOI arXiv:1.

[447] M. Ester, H.-P. Kriegel, J. Sander, X. Xu in Proc. Second Int. Conf. Knowl. Discov. Data Min. AAAI Press, **1996**, pp. 226–231.

[448] D. Wishart, 'Mode analysis: A generalization of nearest neighbor which reduces chaining effects' in *Proc. Colloq. Numer. Taxon.* (Ed.: A. J. Cole), St. Andrews, Scotland, **1969**.

[449] R. J. G. B. Campello, D. Moulavi, J. Sander, 'Density-Based Clustering Based on Hierarchical Density Estimates' in **2013**, pp. 160–172.

[450] L. McInnes, J. Healy, S. Astels, *J. Open Source Softw.* **2017**, *2*, 205.

[451] R. Jarvis, E. Patrick, *IEEE Trans. Comput.* **1973**, *C-22*, 1025–1034.

[452] B. G. Keller, X. Daura, W. F. Van Gunsteren, *J. Chem. Phys.* **2010**, *132*, 074110.

[453] X. Daura, W. F. van Gunsteren, A. E. Mark, *Proteins Struct. Funct. Genet.* **1999**, *34*, 269–280.

[454] O. Lemke, B. Keller, *Algorithms* **2018**, *11*, 19.

[455] A. Rodriguez, A. Laio, *Science* **2014**, *344*, 1492–1496.

[456] D. Comaniciu, P. Meer, *IEEE Trans. Pattern Anal. Mach. Intell.* **2002**, *24*, 603–619.

[457] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, K. Smith, *Comput. Sci. & Eng.* **2011**, *13*, 31–39.

[458] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed., Addison-Wesley, **1994**.

[459] R. G. Weiß, B. Ries, S. Wang, S. Riniker, *J. Chem. Phys.* **2021**, *154*, 084106.

Publications

The following publications relevant to this thesis are attached in the order they are listed below. Author contributions are reported as accepted by the publishing journal or in accordance with the *ICMJE Recommendations for the Conduct, Reporting, Editing, and Publication of Scholarly Work in Medical Journals* (May 2022).

'Total Synthesis of the Death Cap Toxin Phalloidin: Atropoisomer Selectivity Explained by Molecular-Dynamics Simulations' G. Yao, J.-O. Joswig, B. G. Keller, R. D. Süssmuth, *Chem. Eur. J.* **2019**, *25*, 8030–8034 (permission granted by John Wiley and Sons under the license number 5371821147027). Supporting information are included. Author contributions as published with the article: *G. Y.* and *R. D. S.* designed the experiments. *G. Y.* performed the synthesis of all shown compounds. *J.-O. J.* and *B. G. K.* performed the theoretical calculations of the precursors and phalloidin analogues. *G. Y.*, *J.-O. J.*, *B. G. K.*, and *R. D. S.* wrote the manuscript. All authors read, discussed, and approved the manuscript.

'The molecular basis for the pH-dependent calcium affinity of the pattern recognition receptor langerin' J.-O. Joswig, J. Anders, H. Zhang, C. Rademacher, B. G. Keller, *J. Biol. Chem.* **2021**, *296*, 100718 (open access under the Creative Commons CC-BY license). Supporting information are included. Author contributions as published with the article: *J.-O. J.* performed the computer experiments, except for the simulations of the long-loop unfolding, analysed and interpreted the data, made all figures, and drafted and revised the manuscript. *J. A.* performed the computer experiments for the long-loop unfolding and analysed the data. *H. Z.* performed the laboratory experiments and analysed the data. *C. R.* interpreted the data of the laboratory experiments and drafted and revised the corresponding part of the manuscript. *B. G. K.* designed the study, interpreted the data, and drafted and revised the manuscript.

'CommonNNClustering—A Python package for generic common-nearest-neighbour clustering' J.-O. Joswig, B. G. Keller *drafted manuscript*. Author contributions: *J.-O. J.* designed and developed the presented program, performed the experiments, analysed and interpreted the data, prepared all figures, and drafted and revised the manuscript. *B. G. K.* designed the study, interpreted the data, and drafted and revised the manuscript.

Total Synthesis of the Death Cap Toxin Phalloidin: Atropoisomer Selectivity Explained by Molecular-Dynamics Simulations

Chemistry : A European Journal
Volume 25, Issue 34
June 18, 2019
Pages 8030-8034
https://doi.org/10.1002/chem.201901888

JBC RESEARCH ARTICLE EDITORS' PICK

The molecular basis for the pH-dependent calcium affinity of the pattern recognition receptor langerin

Received for publication, February 16, 2021, and in revised form, April 12, 2021 Published, Papers in Press, May 12, 2021,
https://doi.org/10.1016/j.jbc.2021.100718

Jan-O. Joswig[1], **Jennifer Anders**[1], **Hengxi Zhang**[1,2,3,4], **Christoph Rademacher**[1,2,3,4], **and Bettina G. Keller**[1,*]

From the [1]*Department of Biology, Chemistry, and Pharmacy, Freie Universität Berlin, Berlin, Germany;* [2]*Department of Biomolecular Systems, Max Planck Institute of Colloids and Interfaces, Potsdam, Germany;* [3]*Department of Pharmaceutical Chemistry,* [4]*Max F. Perutz Laboratories, Department of Microbiology and Immunobiology, University of Vienna, Vienna, Austria*

Edited by Roger Colbran

The C-type lectin receptor langerin plays a vital role in the mammalian defense against invading pathogens. Langerin requires a Ca^{2+} cofactor, the binding affinity of which is regulated by pH. Thus, Ca^{2+} is bound when langerin is on the membrane but released when langerin and its pathogen substrate traffic to the acidic endosome, allowing the substrate to be degraded. The change in pH is sensed by protonation of the allosteric pH sensor histidine H294. However, the mechanism by which Ca^{2+} is released from the buried binding site is not clear. We studied the structural consequences of protonating H294 by molecular dynamics simulations (total simulation time: about 120 μs) and Markov models. We discovered a relay mechanism in which a proton is moved into the vicinity of the Ca^{2+}-binding site without transferring the initial proton from H294. Protonation of H294 unlocks a conformation in which a protonated lysine side chain forms a hydrogen bond with a Ca^{2+}-coordinating aspartic acid. This destabilizes Ca^{2+} in the binding pocket, which we probed by steered molecular dynamics. After Ca^{2+} release, the proton is likely transferred to the aspartic acid and stabilized by a dyad with a nearby glutamic acid, triggering a conformational transition and thus preventing Ca^{2+} rebinding. These results show how pH regulation of a buried orthosteric binding site from a solvent-exposed allosteric pH sensor can be realized by information transfer through a specific chain of conformational arrangements.

When pathogens invade a mammal (or more specifically: a human), Langerhans cells capture some of the pathogens, process them, and present antigens to the adaptive immune system. The swift activation of the adaptive immune system is critical for the survival of the mammal, and langerin plays a vital role in this process. Langerin is a transmembrane carbohydrate receptor, which is expressed by Langerhans cells of mammalian skin and mucosa (1, 2). It belongs to the class of type II C-type lectin receptors (3, 4). It detects pathogens such as influenza virus (5), measles virus (6), HIV (7), fungi (8), mycobacteria (9), and bacteria (10).

Langerin recognizes these pathogens by binding to carbohydrates on the pathogen surface. Its carbohydrate-binding pocket contains a Ca^{2+} cation as cofactor that is essential for carbohydrate binding, and thus for the capture of pathogens. After the initial binding event, the pathogen is captured in an endocytic vesicle, and langerin releases the pathogen into the endosome (Fig. 1*A*) (1, 2, 7, 11). This cargo release is triggered by a drop of pH from 7 in the extracellular medium to 5.5 to 6 in the early endosome (12) and by a substantial drop in the Ca^{2+} concentration from about 1 to 2 mM to a value in the micromolar range (13–15).

The pH-dependent cargo release is accomplished by a fascinating mechanism in which various chemical equilibria are carefully balanced. To be able to release the cargo into the more acidic endosome, the carbohydrate affinity of langerin needs to be pH dependent. However, the change in pH does not affect the carbohydrate binding itself. Instead, langerin depends on a Ca^{2+} cofactor for carbohydrate binding, and the observed pH dependence of the carbohydrate affinity is caused by an underlying pH dependence of the Ca^{2+} affinity (17). We previously showed that the Ca^{2+} affinity is lower at pH 6 than at pH 7. The pH sensitivity, measured as the difference in the Ca^{2+} binding free energies, is $\Delta\Delta G$ = 5.1 kJ mol^{-1} (17). At high Ca^{2+} concentrations (10 mM) the carbohydrate affinity ceases to be pH dependent, because the excess in Ca^{2+} outweighs any change in Ca^{2+} affinity due to a change in pH. However, in the endosome the Ca^{2+} concentration is low. Thus, the drop in pH from the extracellular medium to the endosome causes a decrease in Ca^{2+} affinity, and the unbinding of the Ca^{2+} cofactor leads to the dissociation of the carbohydrate ligand and to the release of the pathogen. Similarly, pH sensitivities of either ligand or Ca^{2+} affinities have been observed for several other C-type lectins (18), including ASGPR (14, 19, 20) the macrophage mannose receptor (21), DC-SIGN and DC-SIGNR (22–25), and LSECtin (26) (example structures in Fig. S32). In DC-SIGNR and LSECtin, which have a different biological role than langerin, a drop in pH causes an increase in ligand affinity. The mechanisms underlying the regulation by the pH in C-type lectins are highly diverse and not yet studied in detail.

The observation that the Ca^{2+} affinity in C-type lectins is pH dependent is surprising. First, when a carbohydrate (and attached to it an entire virus) is bound to a C-type lectin, the

* For correspondence: Bettina G. Keller, bettina.keller@fu-berlin.de.

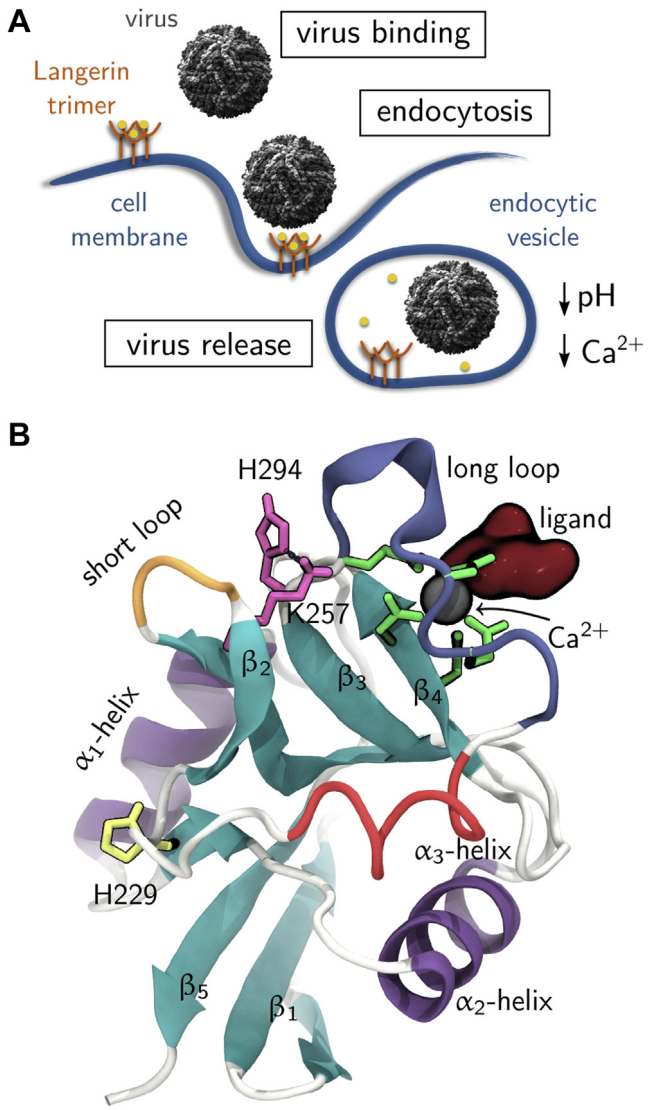


Figure 1. ***C-type lectin langerin.*** *A,* langerin's function as an endocytic pattern recognition receptor (www.scistyle.com; https://creativecommons.org/licenses/by-sa/4.0/). *B,* langerin carbohydrate recognition domain (Protein Data Bank ID 3p5g (16)).

Ca^{2+}-binding site is almost certainly not solvent exposed. Second, the Ca^{2+} in C-type lectins is coordinated by either aspartate or glutamate side chains, whose reference pK_a values (27) (in water at 25 °C) are 3.71 (aspartate) and 4.15 (glutamate). By themselves, these residues are not sensitive to a change in pH from 7 to 6. Pairs of acidic residues can in principle form a protonated dyad, which is the close arrangement of two residues with acidic side chains such that protonation of their carboxyl groups is coupled. This results in an increased pK_a of the protonated residue, stabilized by the unprotonated form of the other group. Prominent examples of this effect are found in the proteins HIV-1 protease (28, 29), BACE-1 (30), BACE-2, and CatD, where it can increase the pK_a of aspartic acid from its reference value to 5.2 (31). The presence of organic ligands can increase these values further (32). However, a protonated dyad can only form if Ca^{2+} has already left the binding pocket. So, the question arises: how do C-type lectins sense a change in pH, and how does this lead to the release of Ca^{2+}?

For langerin we previously identified the histidine residue H294 as a partial pH sensor that regulates the Ca^{2+} affinity (17). The reference pK_a of histidine is 6.04 (in water at 25 °C) (27), which makes it sensitive to a pH change from 7 to 6. When H294 is mutated to A294, the pH sensitivity is about 40% smaller than in the wildtype ($\Delta\Delta G$ = 3.1 kJ mol^{-1} upon a change in pH from 7 to 6). Because the histidine side chain points away from the Ca^{2+}-binding site, it is unlikely that the decrease in Ca^{2+} affinity is caused by electrostatic repulsion between the protonated histidine and the Ca^{2+}. This mechanism has been suggested for the C-type lectin ASGPR, in which, however, the histidine pH sensor is located directly underneath the Ca^{2+}-binding pocket (Fig. S32*D*) (20). Instead, we showed—by combining NMR experiments, site-directed point mutations, and molecular dynamics simulations—that H294 is at the center of an allosteric network that contains the Ca^{2+}-binding site. More specifically, in its unprotonated form H294 forms a hydrogen bond with lysine K257, which is also present in the known crystal structures of langerin (16). This hydrogen bond cannot be formed if H294 is protonated, and the allosteric mechanism that regulates the Ca^{2+} affinity likely hinges on this hydrogen bond.

Yet, protonation of H294 is only the initial detection that the surrounding medium has changed. Even though we identified the residues that are involved in the allosteric network, we do not yet understand how the protonation of H294 could ultimately affect the Ca^{2+}-binding pocket. Several allosteric effects have been reported for C-type lectins (see ref. (33) for a recent review), but little is known about their underlying molecular mechanisms that could be applied to the situation in langerin. The goal of this study is to elucidate how the protonation of H294 changes the conformational ensemble of langerin and to investigate the effect these conformational changes have on the Ca^{2+}-binding pocket. A model of how the signal, that the pH has changed, traverses the allosteric network to the buried Ca^{2+}-binding site and triggers the Ca^{2+} release might serve as a blueprint for understanding how pH-sensitive ligand binding is achieved in C-type lectins and other proteins.

Results and discussion

Structure of the langerin carbohydrate recognition domain

Langerin forms a homotrimer. The monomers consist of a short cytoplasmic tail, a transmembrane region, and a long alpha-helical neck (residues 56–197) extending into the extracellular milieu, which carries the C-terminal carbohydrate recognition domain (16, 18). The carbohydrate recognition domain has the typical C-type lectin domain fold (Fig. 1*B*) (4), which consists of two extended β sheets (turquoise), each composed of three single strands. The two β sheets are flanked by three α helices (purple, α_3 in red). The carbohydrate-binding pocket, which contains the Ca^{2+}-binding site, is located on top of the β_4 strand. One residue from this β sheet directly binds to the Ca^{2+}: D308. In addition, the Ca^{2+} is held in place by E293 and E285 in the long-loop (blue), which coordinate to Ca^{2+} from the side. E285 is part of a conserved

EPN-motif (E285, P286, N287 in langerin), which determines the selectivity for mannose, fucose, and glucose over galactose (18, 34, 35). The pH sensor H294 (pink) is located at the end of the long-loop. If its side chain is unprotonated, it forms a hydrogen bond to K257 (also pink) in the short-loop (orange). The allosteric network that regulates the Ca^{2+} affinity comprises the long- and the short-loop (17). H229 (yellow) is the only other histidine residue in the langerin carbohydrate recognition domain. A pathogen would bind *via* a carbohydrate ligand (dark red) to langerin and would be separated from the pH sensor by the long-loop. If Ca^{2+} is bound to langerin, we will call the system holo-langerin, otherwise apo-langerin.

The effect of H294 protonation on the conformational ensemble

We conducted 31 μs of molecular dynamics simulations of holo-langerin, in which all residues were protonated according to their default protonation state at pH 7, *i.e.*, H294 was unprotonated, and the overall protein was neutral (neutral state). We compare these simulations with 27 μs of holo-langerin, in which H294 was protonated (protonated state). Protonation of H294 has no influence on the secondary structure of langerin (Fig. 2*A*, Fig. S1). Thus, any conformational change due to the protonation of H294 affects the side chains, or those residues that are not assigned to a specific secondary structure, *i.e.*, the loop regions.

One way a conformational change in the loop regions could manifest itself, is by a change of the loop flexibility. This is, however, not corroborated by the root-mean-square fluctuations of the individual residues (Fig. 2*B*). The short-loop (sharp peak around residue 260) and the α_3 helix (broad peak around residue 275) are more rigid in the protonated state, but the difference is very small. The flexibility in all other regions of the protein, and in particular the long-loop region, does not change upon protonation.

To gauge whether protonation of H294 has an influence on the conformation of the Ca^{2+}-binding site, we measured the distance distribution between the carboxyl group of the Ca^{2+}-coordinating residues—E285, E293, and D308—and the Ca^{2+} (Fig. 2, *C* and *D*). For E293 and D308 the differences are too minor to explain the observed difference in Ca^{2+} affinity. For E285 the distribution shifts slightly to lower distances and thus to a potentially tightly bound Ca^{2+}, not explaining it either. The distance difference between the two populated states is about 0.05 nm.

Yet, we know from our previous analyses (17) that protonation of H294 causes a significant shift in the conformational ensemble, and this is again confirmed by the distance distributions between the H294 side chain and the Ca^{2+} in the neutral and the protonated state (Fig. 2*E*). In the protonated state the distribution shifts to larger distances, well beyond 1 nm. At this distance, we do not expect a significant influence of the positively charged H294 side chain on the Ca^{2+}, considering that H294 is located on the protein surface and that the dielectric constant between the two interacting groups is relatively high (see Figs. S20 and S21 for an assessment of the Coulomb interaction) (36). Thus, we can rule out that the decrease in Ca^{2+} affinity is caused by direct Coulomb repulsion between the protonated H294 and the Ca^{2+}.

To uncover which residues besides H294 are involved in the conformational shift, one needs to compare the two conformational ensembles. This cannot be accomplished in the full high-dimensional conformational space. Instead, one needs to project the two ensembles into a low-dimensional space that is representative of both systems. Principal component analysis

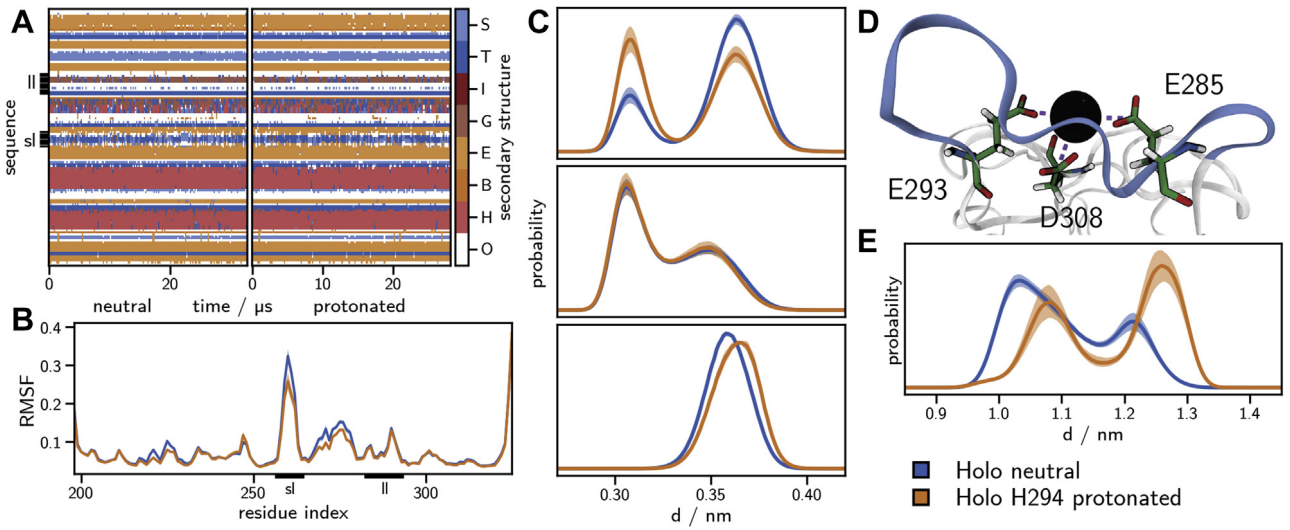


Figure 2. Structural consequences of H294 protonation. *A*, analysis by the hydrogen bond estimation algorithm DSSP of the secondary structure in the neutral (*left*) and the protonated holo-state (*right*). Legend: S, bend; T, hydrogen bonded turn; I, 5-helix; G, 3-helix; E, extended strand, part of β-ladder; B, isolated β-bridge; H, α helix; O, unassigned. *B*, C_α-root-mean-square fluctuation (RMSF). *C*, carboxyl carbon–Ca^{2+} distance histograms for E285 (*upper graph*), E293 (*middle graph*), and D308 (*lower graph*). *D*, structure of the Ca^{2+}-binding site showing the distances plotted in *C* with *dashed lines*. *E*, histogram of the minimum distance between Ca^{2+} and the side-chain N atoms (N_δ and N_ε) of H294. *Solid lines*, mean of the histograms calculated for each simulation replica. *Shaded area*, 95% confidence interval of the mean obtained by bootstrapping (1000 samples). ll, long-loop; sl, short-loop.

(37) identifies low-dimensional spaces that preserve the directions of the largest conformational variance (38). To be able to directly compare the neutral and the protonated ensemble, we combined the simulations in the two protonation states to obtain a joint principal component space. The principal component with the largest variance represents the opening and closing of the gap between the short-loop and long-loop (blue sequence of structures in Fig. 3*A*). The second principal component represents a sideways shear motion of the short-loop (orange sequence of structures in Fig. 3*A*). This is in line with our previous finding that the allosteric network is centered on these two loops (17). Even though the two principal components cover only about 28% of the total structural variance (Fig. 3*B*), they represent the conformational fluctuations that are most sensitive to a protonation of H294. Separate principal component analyses of the two protonation states yielded principal components that were almost identical, indicating that the joint principal components are a faithful representation of the largest variances for both protonation states. Figure 3*C* shows the free energy surface of the two systems in the space of the first two joint principal components. The free-energy surface of the unprotonated system is shallow with two minima corresponding to the open and closed states of the short- and long-loop. Upon protonation, the free energy surface becomes much steeper and more structured. One can discern at least three minima. The difference plot of the probability densities in the neutral and protonated states (Fig. 3*C* to the right) shows these emerging conformations in red.

We extracted the highly populated regions by clustering in the space of the first two principal components using the density-based common-nearest-neighbors cluster algorithm (39–41) and characterized the hydrogen bond pattern of the short- and long-loop residues in each of the clusters (Fig. 4). Figure 4, *C* and *D* show a subset of the full analysis (see Fig. S10) focusing on fluctuating hydrogen bonds. In the neutral state, the clusters have essentially the same hydrogen bond populations as the total ensemble, which is consistent with the shallow free-energy surface in Figure 3*C*.

The situation is different in the protonated state. Here, each of the four clusters is stabilized by a hydrogen bond pattern that is distinctively different from the hydrogen bond pattern of the total distribution (Fig. 4*B*). This indicates that, upon protonation of H294 several distinct short-loop/long-loop conformations emerge.

The most striking change arises in the green (G) cluster: the hydrogen bond between the side chain of K257 and the side chain of D308, which is barely populated in the unprotonated state (4.2%), is populated to 65.4% in this cluster and 12.9% in the ensemble. In parallel, the side chain of the now protonated H294 forms a hydrogen bond with the carboxyl group of E261. The structure is further stabilized by a hydrogen bond between the side chain of S265 and the main chain of T256. Note the significance of this finding: the K257 side chain, which is no

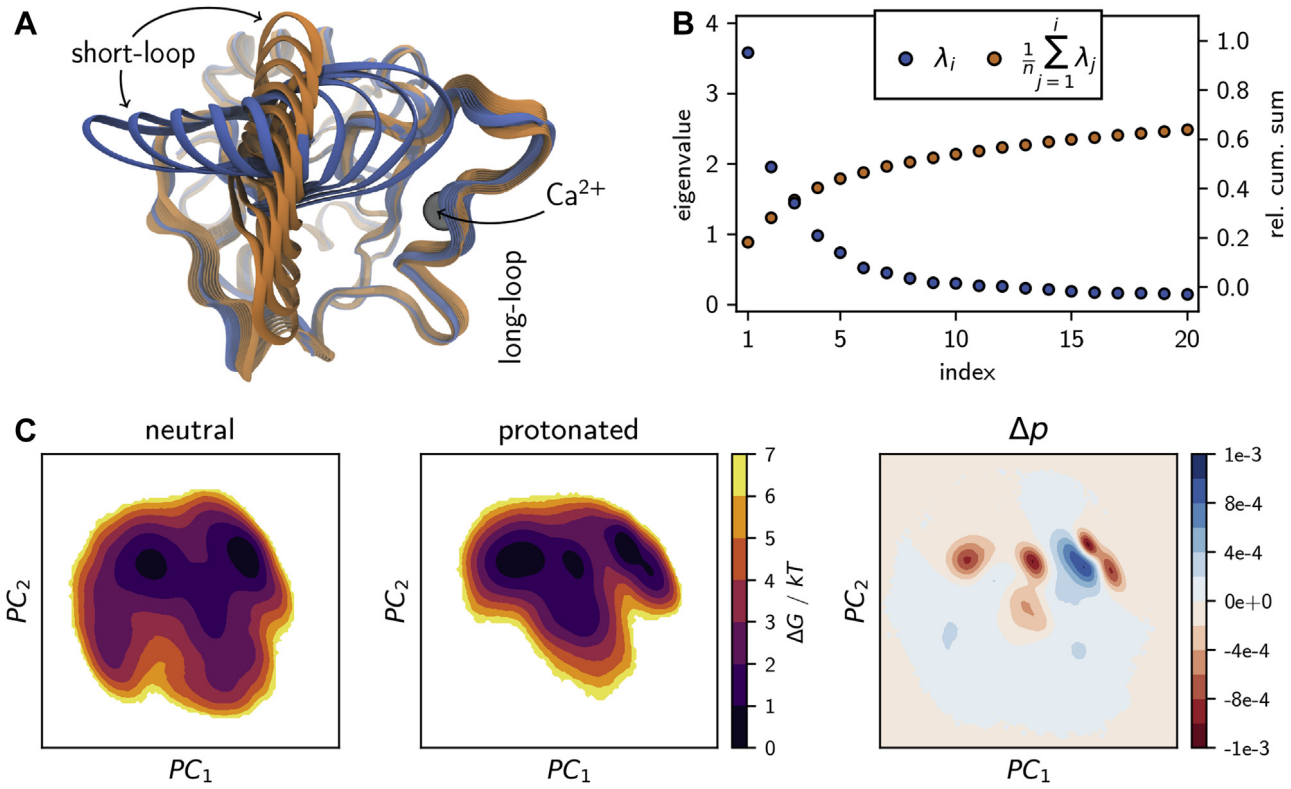


Figure 3. Principle component analysis. *A*, structural interpolations along the first two principal components. *B*, Eigenvalue spectrum of the principal component analysis (*blue dots*) and the cumulative sum normalized by the total sum of all *N* eigenvalues $n = \sum_{i=1}^{N} \lambda_i$ (*orange dots*). *C*, free-energy surfaces from the 2D projections of the individual holo-langerin trajectories onto principal components 1 and 2 and difference plot of the underlying probability distributions (neutral – protonated).

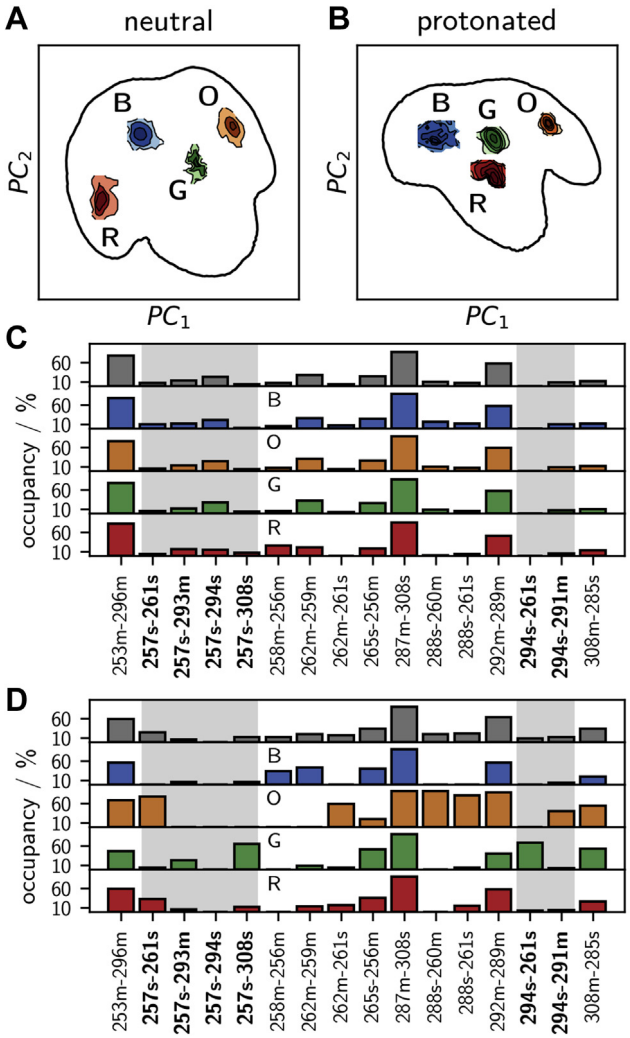


Figure 4. Characterization of conformational states *via* hydrogen bonds. Four most populated clusters in the principal-component free-energy surface of *A*, the neutral and *B*, the protonated holo-langerin. Per cluster hydrogen bond occupancy in *C*, neutral and *D*, protonated holo-langerin (populations in the full ensemble in *gray*). Hydrogen bonds involving K257 or H294 are highlighted by a *gray background*.

longer engaged in a hydrogen bond with H294, forms a new hydrogen bond with the Ca^{2+}-coordinating residue D308 and thereby moves a proton into the vicinity of the Ca^{2+}-binding pocket.

The conformation of the orange (O) cluster is complementary to that of the green cluster. The side chain of K257 forms a hydrogen bond with the carboxyl group of E261, while H294 engages in a hydrogen bond to the backbone carbonyl oxygen of N291. The conformation is stabilized by hydrogen bonds between the side chain of N288 and the backbone carbonyl oxygen of M260 and the side chain of E261. N288 is located in the center of the long-loop, and E261 is located in the center of the short-loop. Thus, these two hydrogen bonds closely connect the two loops explaining why this structure appears in the closed-loop region of the free-energy surface. The main chain–main chain hydrogen bond between N292 and A289 additionally stabilizes this structure.

The blue (B) cluster is an open-loop structure in which neither K257 nor H294 is engaged in one of the considered hydrogen bonds. It features the 258m–256m and 262m–259m hydrogen bonds within the short-loop. The red (R) cluster is a slightly sheared structure in which the K257 side chain partly forms a hydrogen bond to the carboxyl group of E261 and partly to the carboxyl group of D308.

Three hydrogen bonds in Figure 4 directly involve Ca^{2+}-coordinating residues. First, we already discussed the hydrogen bond K257–D308. Second, the hydrogen bond between the main chain of N287 and the side chain of D308 is important for the stability of the long-loop fold. It is occupied to about 90% in both protonation states. Third, population of the hydrogen bond between the main chain amid group of D308 and the carboxyl group of E285 is increased in the protonated state. This is particularly true for cluster G (green) and O (orange). This hydrogen bond might compete with the coordination of E285 to Ca^{2+} and thereby might contribute to the observed decrease in Ca^{2+} affinity. In both the neutral and the protonated systems, the bonds N288s–M260m, N288s–E261s, K257s–E261s, and G262m–E261s are strongly correlated (see Fig. S10). In the protonated state a strong correlation between K257s–D308s and H294s–E261s arises, indicating that these two hydrogen bonds are formed and broken simultaneously.

A mechanism for the pH-sensitive Ca^{2+} affinity in langerin

We are now ready to propose a mechanism that explains how protonation of H294 can lead to a decrease in Ca^{2+} affinity. In the neutral state, K257 and H294 form a hydrogen bond that is populated over a wide range of conformations. We also observe a weak hydrogen bond of the K257 side chain to the main chain of the Ca^{2+}-coordinating residue E293, but direct hydrogen bonds to the Ca^{2+}-coordinating carboxyl groups are hardly ever formed (Fig. 5*A*). Upon a drop of pH from 7 to 6, the side chain of H294 is protonated in accordance with its pK_a: H294 is the initial pH sensor. The protonation of H294 changes the hydrogen bond pattern between the short- and the long-loops. In particular, the side chains of H294 and K257 form new contacts, which gives rise to previously inaccessible conformations. Cluster O (orange) and cluster G (green) exhibit mutually exclusive hydrogen bond patterns. In cluster O, multiple hydrogen bonds connect the short- and the long-loops causing a closed loop conformation. The positively charged side chain of K257 forms a hydrogen bond to the negatively charged side chain of E261. But similar to the neutral state, there is no direct hydrogen bond to the Ca^{2+}-coordinating carboxyl groups (Fig. 5*C*). This is different in cluster G. Here the positively charged side chain of H294 forms a hydrogen bond with the negatively charged carboxyl group of E261. Simultaneously, the positively charged side chain of K257 forms a hydrogen bond with the carboxyl group of D308 (Fig. 5*B*). This hydrogen bond withdraws electron density from the coordinative bond between D308 and Ca^{2+} and thereby reduces the Ca^{2+} affinity. It is even conceivable that the proton is transferred entirely to the carboxyl group of D308 (42). We thus propose that cluster G (green) is responsible for the decrease in Ca^{2+} affinity at pH 6.

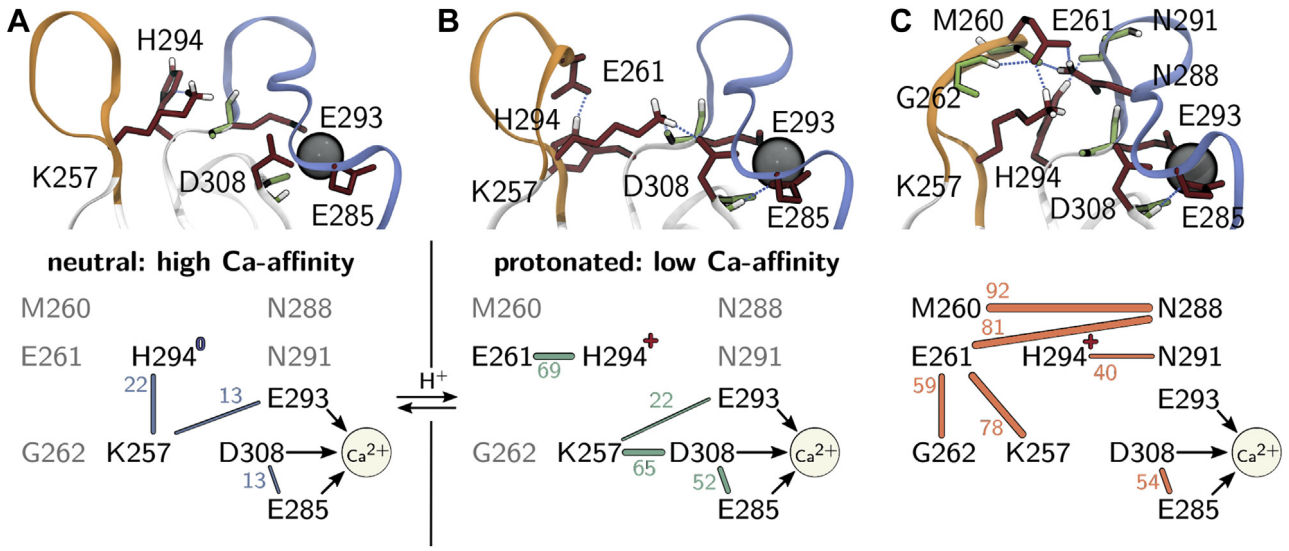


Figure 5. Allosteric mechanism for the pH-sensitive Ca^{2+} affinity in langerin. *A*, neutral state, *B*, cluster G (*green*) in the protonated state, and *C*, cluster O (*orange*) in the protonated state. *Lines*, hydrogen bonds with population in percent. *Arrows*, coordination between carboxyl groups and Ca^{2+}.

In this mechanism, K257 acts as a proton reservoir. The initial detection of a pH change *via* protonation of H294 leads to the cluster G, in which K257 moves a proton into the vicinity of the Ca^{2+}-binding site and locally increases the proton concentration. Thus, the signal that the pH has changed is allosterically transferred to the Ca^{2+}-binding pocket without transferring the actual proton that triggered the mechanism.

A crucial assertion in the proposed mechanism is that the life time of cluster G (green) represents a distinct conformation that is stable enough for the Ca^{2+} to leave the binding pocket. The fact that cluster G corresponds to a free-energy minimum in the space of the principal components hints at a stable conformation. But because the principal components maximize the spatial variance and not the variance in time, this is not sufficient to be certain.

Figure 6*A* shows the distance distribution between the K257 and D308 side chain for the neutral and the protonated states. In both protonation states, the maximum at short distances around 0.2 nm is well separated from the maximum at larger distances.

In the neutral state, the short distances are populated only in 4.3% of all simulated conformations, which increases to 13.2% when H294 is protonated. This is in line with the increase of population in the K257–D308 hydrogen bond from 4.2% to 12.9%. We obtain the same results, when plotting the distance between the K257 side-chain amine and the Ca^{2+} in Figure 6*B*. Thus, cluster G (green) indeed represents a distinct conformation.

To assess the stability of conformations in cluster G (green), and to relate its formation to other dynamic processes in the protein, we constructed a core-set Markov model of the conformational dynamics (43–45). In Markov models, the conformational space is discretized into states and the conformational dynamics are modeled as Markov transitions within a lag time τ between pairs of these states, where the transition probabilities are obtained from molecular dynamics simulations. From the eigenvectors and eigenvalues of the Markov-model transition matrix one obtains long-lived conformations as well as the hierarchy of the free-energy barriers separating them. The special feature of core-set Markov models is that the states are confined to the regions close to the minima of the free-energy surface, *i.e.*, so-called core sets, whereas the regions between these minima are modeled by committor functions. This reduces the discretization error of the model considerably.

The Markov model construction is preceded by a dimensionality reduction of the conformational space using the time-independent component analysis (46, 47). Time-independent components (tICs) maximize the variance within lag time τ rather than the instantaneous variance maximized by principal

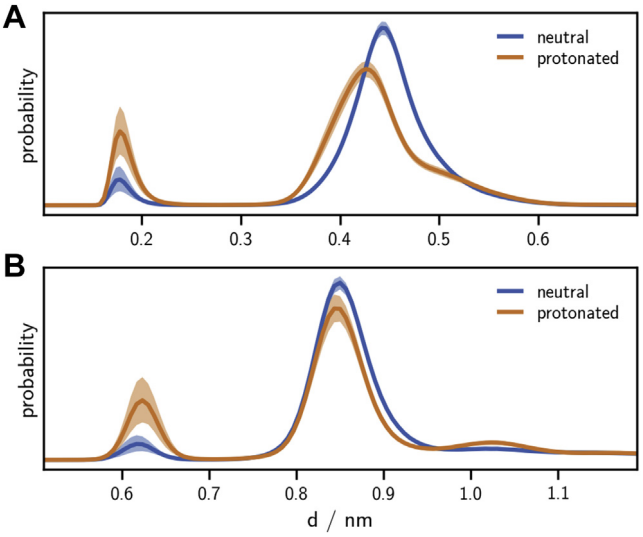


Figure 6. Frequency of K257 interaction with the Ca^{2+}-binding site. *A*, K257–D308 side-chain distance distribution. *B*, K257 side-chain amine – Ca^{2+}distance distribution. *Solid lines*, mean of the histograms calculated for each simulation replica. *Shaded area*, 95% confidence interval of the mean obtained by bootstrapping (1000 samples).

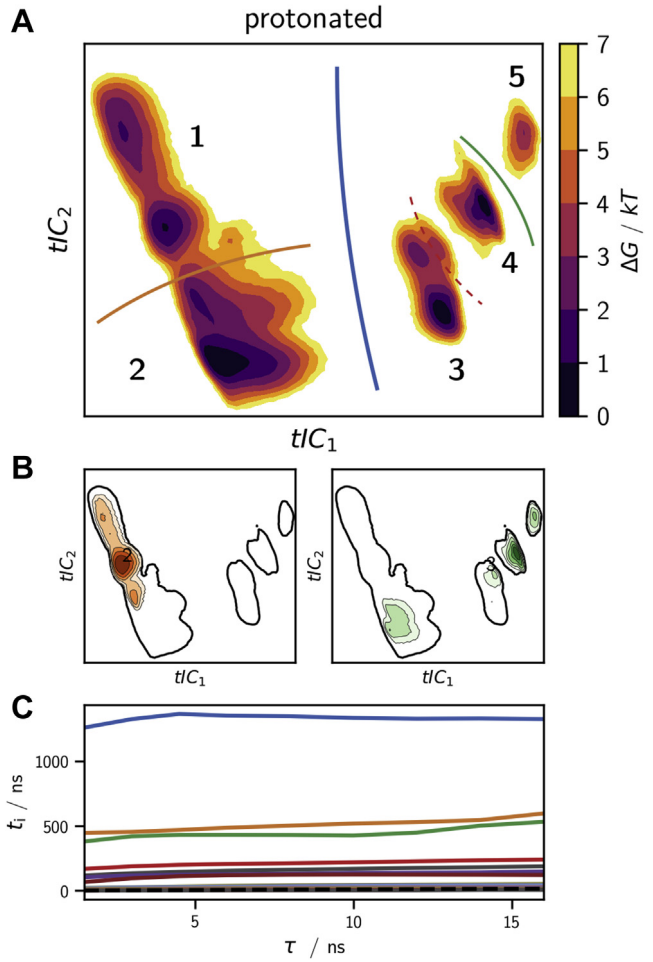


Figure 7. Core-set Markov model of the conformational dynamics of protonated holo-langerin. *A*, free energy surfaces from the 2D projections of protonated holo-langerin trajectories onto the first two time-independent components (tICs). *Solid lines*, transition regions between the five metastable states connected by the four slowest dynamic processes. *B*, projections of cluster G (*green*) and O (*orange*) into the space of the first two tICs. *C*, implied time scales of the core-set Markov model. The colors of the processes match the transition regions drawn into (*A*).

components. A projection into a low-dimensional tIC space can thus be interpreted as projection into the space of the slowly varying coordinates of the system. Figure 7*A* shows the free-energy surface of the protonated system projected into the space of the first and the second tICs (see Supporting information for other projections), and Figure 7*B* shows the projection of cluster G (green) and O (orange) into this space.

We then identified 22 core sets in the space of the first six tICs using common-nearest-neighbors clustering (39, 41) and used them to construct a core-set Markov model. The implied timescale test shows that the timescales of our core-set Markov model are independent of the lag time τ indicating a very small discretization error and thus a high-quality Markov model (Fig. 7*C*). The slowest dynamic process occurs on a timescale of about 1.3 μs and corresponds to changes in the local conformations of E261 and its hydrogen bond pattern. It thus separates the conformations of cluster G (green) and cluster O (orange) along the blue barrier in Figure 7*A*. Note that all conformations in which the K257s–D308s hydrogen bond is formed alongside H294s–E261s are located on the right-hand side of this barrier (see Supporting information). The fact that we find some structures that have originally been assigned to the G (green) conformation on the left-hand side of the barrier is likely due to the insufficient separation of long-lived conformations in the principal component space (Fig. 7*B*). Next, protonated langerin has two slow timescales that occur at about 500 ns. One process describes transitions between the closed loop conformations in region 1 and conformations in which the distance between the long- and the short-loop is larger in region 2. The other process represents a transition between conformations in which the backbone orientation of N291 forbids the N292m–A289m hydrogen bond giving rise to a distortion of the long-loop (region 5) and the conformations in which the N292m–A289m hydrogen bond is possible (regions 3 and 4). The dashed barrier marks transitions to more open short-loop forms occurring on a timescale of 210 ns.

In summary, conformations in which the K257–D308 hydrogen bond is formed are separated from the alternative O (orange) conformation by a rare transition that occurs on a timescale of 1.3 μs. Within the right-hand side of the barrier in Figure 7*A* the G (green) conformation is at least stable on a timescale of 200 ns. This is likely sufficient to enable the escape of the Ca^{2+} from the binding pocket. A core-set Markov model of neutral holo-langerin is reported in the Supporting information.

To directly probe how the stability of the Ca^{2+}-bound state of the protein depends on the protonation state and on the conformation of langerin, we used constant-velocity steered–molecular dynamics (MD) experiments (48–50). In these simulations, a force that increases linearly with time is applied to the Ca^{2+} atom (Fig. 8*A*), and the opposing force (*i.e.*, the resistance against this pulling force) is measured. At a certain maximum force the ionic bonds between the Ca^{2+} atom and the coordinating residues rupture and the Ca^{2+} leaves the binding pocket. In the computer experiment, this is marked by a sudden drop in the opposing force (Fig. 8*C*). The rupture force is a rough measure for the free-energy difference to the transition state $\Delta G^{\ddagger}$. The rational is that a deeper free-energy minimum of the Ca^{2+}-bound state is associated with a steeper slope to the transition state, and the rupture force, reflecting the maximal slope, reports on the stability of the Ca^{2+}-bound state (51, 52). We chose the pulling rate such that the rupture events are observed after several nanoseconds. This ensures that the system has enough time to adjust to the pulling and also that the initial starting conformation is preserved to some degree.

For each system, we conducted 40 steered-MD simulations and report the data as notched boxplot in Figure 8*B*. Overall, the plot shows that we could determine the median of rupture force with high confidence and hardly any outliers. The rupture force decreases from the neutral to the protonated system ($H294^{+}$) and then further to simulations of the protonated system started in the G (green) conformation, in which the K257 amine forms a hydrogen bond with the D308 carboxyl group. This decrease is predicted by our mechanism. Note that classical force fields cannot model instantaneous

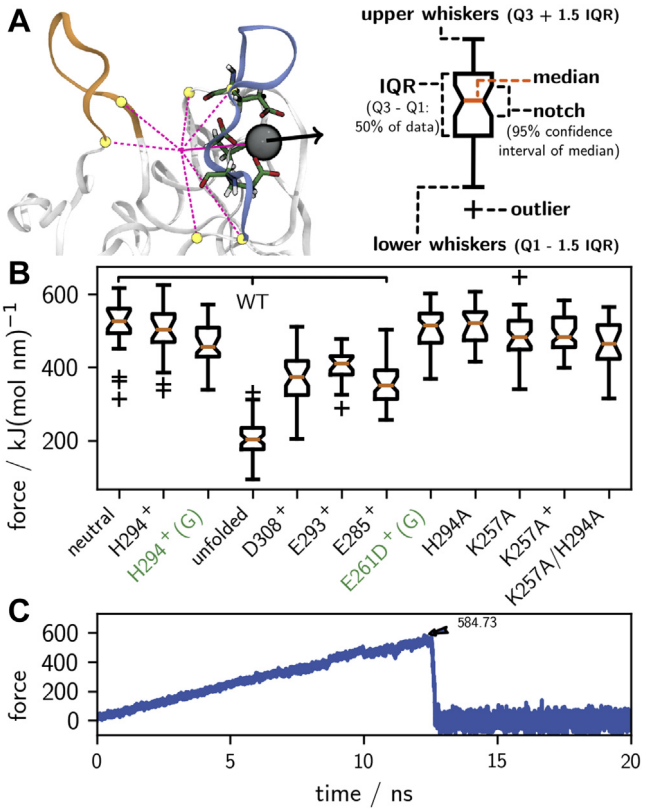


Figure 8. Constant-velocity steered-MD. *A*, pull coordinate defined as the distance vector between Ca^{2+} and the center of mass of the C_α-atoms of residues 257, 264, 281, 282, 293, and 294. *B*, maximal pulling force observed acting on Ca^{2+} during simulations of langerin in various states as *notched box* representation. The *orange line* represents the median, while the box enframes the interquartile range. The box notches indicate the 95% confidence interval on the median. Points lying beyond 1.5 times the edges of the box are regarded as outliers (+), and the whiskers mark the data range without outliers. *C*, example for a force trajectory with a rupture event at about 12 ns. Maximum force indicated by an *arrow*.

shifts in the electron density due to the formation of hydrogen bonds. Thus, the rupture force in the G (green) conformation might actually be somewhat lower. If the Ca^{2+}-coordinating residue D308 is protonated, corresponding to a situation in which the proton is transferred from K257 to D308, the rupture force is about 150 kJ/(mol nm) lower than in the neutral system.

The same is observed when one of the other two Ca^{2+}-coordinating residues is protonated. A drastic reduction in the rupture force is observed, when the experiment is started from a state where the long-loop is unfolded. This is expected, as one of the Ca^{2+}-coordinating residues E285 is removed from the cage of the binding site in this arrangement. The rupture force for the mutant E261D (started from an analogon of the G conformation) and the mutant H294A are in the same range as for the neutral wild-type langerin.

Of note, the rupture forces for K257A mutants are insensitive toward the modeled state of H294. The binding capability is virtually the same, no matter if H294 is neutral, protonated, or mutated. This substantiates the importance of K257 to transport a protonation signal to the Ca^{2+}-binding site.

Comparison with experimental data

The Ca^{2+}-dissociation constant of wildtype langerin at pH 7 is K_d = 105 ± 15 μM and increases to K_d = 800 ± 150 μM at pH 6 (17), as determined by isothermal titration calorimetry (ITC). These dissociation constants correspond to binding free energies of $\Delta G_{pH\ 7}$ = −22.9 kJ/mol, and $\Delta G_{pH\ 6}$ = −17.8 kJ/mol at T = 300 K, yielding a pH sensitivity of $\Delta\Delta G = \Delta G_{pH\ 6} - \Delta G_{pH\ 7}$ = 5.1 kJ/mol (Fig. 9). By contrast the dissociation constants of the H294A mutant, in which the pH sensor H294 is removed, are K_d = 35 ± 15 μM at pH 7 ($\Delta G_{pH\ 7}$ = −25.6 kJ/mol) and K_d = 125 ± 5 μM at pH 6 ($\Delta G_{pH\ 6}$ = −22.4 kJ/mol), corresponding to a reduced pH sensitivity of $\Delta\Delta G$ = 3.2 kJ/mol (17) (Fig. 9). Our mechanism so far explains the pH sensitivity due to the pH sensor H294. The fact that the H294A mutant exhibits a residual pH sensitivity indicates that langerin has a second pH sensor.

To convince ourselves of the robustness of these results, we remeasured the dissociation constants of wildtype langerin (see Supporting information). We obtained K_d = 113 ± 14 μM at pH 7 ($\Delta G_{pH\ 7}$ = −22.6 kJ/mol) and K_d = 802 ± 150 μM at pH 6 ($\Delta G_{pH\ 6}$ = −17.8 kJ/mol), yielding a pH sensitivity of $\Delta\Delta G$ = 4.8 kJ/mol (Fig. 9). This is in excellent agreement with our previous results.

Four residues are central to our mechanism: H294, K257, D308, and E261. D308 directly coordinates to $Ca^{2+}$and is therefore not a suitable candidate for site-directed mutagenesis. In contrast to H294A, the pH sensitivity of K257A could not be determined because the protein precipitated at pH 7. However, both mutants have a higher Ca^{2+} affinity than wildtype langerin at pH 6, which previously could not be explained. The overall higher Ca^{2+} affinity in the K257A mutant is predicted by our mechanism, because the K257–D308 hydrogen bond that destabilizes the Ca^{2+} coordination cannot be formed in the absence of the K257 side chain. The H294A mutant has the K257 side chain, and the conformation in which K257 is in the vicinity of D308 (Fig. 6) can in principle be formed. However, in our simulations of H294A we find that

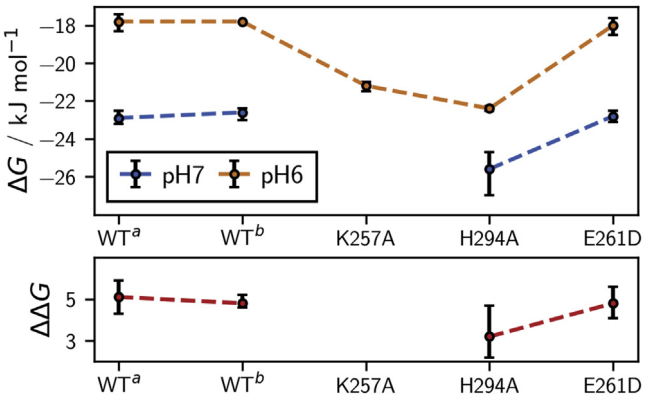


Figure 9. Ca^{2+} binding free energies under standard conditions in kJ/mol. Calculated as $\Delta G = -RT\ln(K_d)$, where R = 8.314 J/(K mol) is the gas constant, T = 300 K is the temperature, and K_d in units of mol/l are the experimentally determined dissociation constants. Measurements at pH 6 (*blue*) and pH 7 (*orange*) with experimental uncertainties indicated with error bars and pH sensitivities in kJ/mol calculated as $\Delta\Delta G = \Delta G_{pH\ 6} - \Delta G_{pH\ 7}$ (*red*).

the K257 side chain is in the vicinity of the D308 side chain in only 1.7% of the simulated structures, which might explain the higher Ca^{2+} affinity of the H294A mutant (see Supporting information).

Besides H294 and K257, residue E261 is important for the stabilization of the G (green) conformation, which is responsible for lowering the Ca^{2+} affinity. However, it also stabilizes the cluster O (orange), which is not expected to increase the Ca^{2+} affinity, because K257 forms a hydrogen bond with E261 rather than with D308 in this conformation. We therefore predicted that mutating E261 has little effect on the pH sensitivity. We measured the Ca^{2+}-dissociation constants for the E261D mutant at pH 6 and pH 7 by ITC (see Supporting information), and the results confirm our prediction. The dissociation constants of the E261D mutant are K_d = 108 ± 11 μM at pH 7 ($\Delta G_{pH\ 7}$ = −22.8 kJ/mol) and K_d = 742 ± 141 μM at pH 6 ($\Delta G_{pH\ 6}$ = −18.0 kJ/mol), yielding a pH sensitivity of $\Delta\Delta G$ = 4.8 kJ/mol (Fig. 9).

Long-loop unfolding

So far, our mechanism explains how Ca^{2+} is destabilized in the binding pocket of holo-langerin. However, if the proton is transferred from K257 to D308, the mechanism also has profound effects on apo-langerin. In holo-langerin the long-loop is stabilized in a well-defined conformation (folded long-loop conformation) by E285, which coordinates to Ca^{2+}. In apo-langerin this interaction is not possible, and the long-loop spontaneously unfolds in our simulations as shown by the RMSD evolution in Figure 10*B*. Similar long-loop unfolding has been observed in the crystal structures of other C-type lectins, like tetranectin (53), TC14 (54), or MBP (55). To estimate the unfolding rate, we conducted 30 to 60 simulations (see Supporting information) for each of the following protonation states of apo-langerin: neutral, H294 protonated, H294 and E285 protonated, H294 and E293 protonated, and H294 and D308 protonated, each of them started in the folded conformation. In four of the five protonation states 44% to 54% of all trajectories unfold within 220 ns simulation time, as determined by visual inspection (Fig. 10*C*, blue dots). The carboxyl group D308 is critical for the stabilization of the folded loop conformation in the absence of Ca^{2+} by forming hydrogen bonds with the N287 side chain, as well as with the backbone amide-hydrogen of N287 and N288 (Fig. 10*A*). If D308 is protonated, all three hydrogen bonds are much weaker, and consequently the long-loop unfolds at a higher rate (75% within 220 ns).

Long-loop unfolding often occurs *via* an intermediate conformation, in which the hydrogen bonds with the backbone amides of N287 and N288 are broken, while the hydrogen bond to the N287 side chain is still possible. In this intermediate form the loop is more flexible than in the fully folded state, but the characteristic turns in the loop backbone are still largely present, and we observe refolding to the fully folded state in some of the trajectories. The transition to the fully unfolded conformation occurs when one or more of the backbone torsion angles in the long-loop rotate and the hydrogen bond between the side chains of D308 and N287 breaks. This transition is irreversible on the timescale of our simulations.

To corroborate our visual analysis of the simulation end points, we determined the time of the unfolding event by four additional criteria: the mean between last fully folded frame and first fully unfolded frame determined by visual inspection, the C_α-RMSD of the long-loop residues exceeds 0.2 nm, and breaking of the hydrogen bonds between the D308 carboxyl group or the backbone amide-hydrogen of N287 and N288. All four criteria confirm the first analysis (Fig. 10*C*).

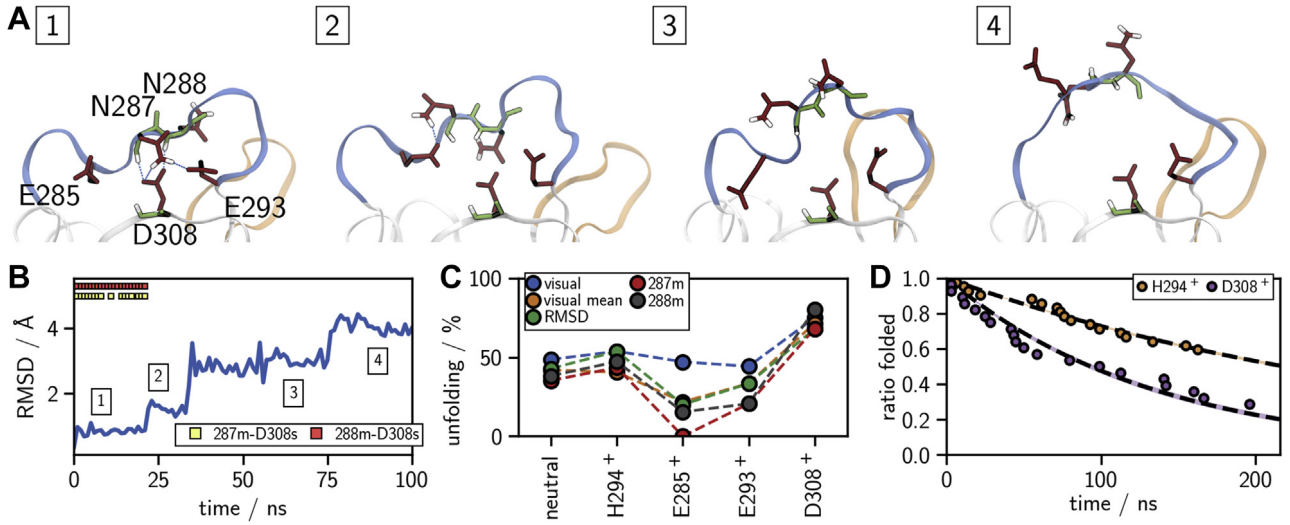


Figure 10. Long-loop unfolding in apo-langerin. *A*, example structures for a fully folded (A1), intermediate (A2), partially unfolded (A3), and fully unfolded (A4) state. *B*, example trajectory of the long-loop C_α-RMSD; 22 ns, intermediate state; 30 ns, unfolding event. *C*, percentage of unfolded trajectories within 220 ns determined by: last folded frame (visual), mean of last folded and first unfolded frame (visual mean), RMSD > 0.2 nm, and hydrogen bonds N287m–D308s (287m) and N288m–D308s (288m). *D*, decay plot of folded trajectories (last folded frame) and exponential fit (*dashed line* ±σ), $H294^+$: H294 protonated, $D308^+$: H294 and D308 protonated.

If E285 is protonated, a hydrogen bond between the protonated carboxyl group of E285 and the unprotonated carboxyl group of D308 stabilizes a partially folded loop structure, such that for some criteria we observe even fewer unfolding events than by the simple visual analysis for this system. We determined the half-life periods $t_{1/2}$ of the folded states from the decay plots of the folded trajectories (see Supporting information). Independent of the criterion, the decay is fastest, when D308 is protonated. In particular, unfolding is over twice as fast if D308 is protonated than if only H294 is protonated ($t_{1/2}$ = 218 *versus* 93 ns, Fig. 10*D*). Some of the decays deviate from a single-exponential decay, hinting at a more complex underlying unfolding mechanism.

Since the folded conformation binds Ca^{2+} much more strongly than the unfolded conformation (Fig. 8), the equilibrium between folded and unfolded long-loop is critical for the overall Ca^{2+} affinity. Thus, the protonation of D308 has a 2-fold effect: First, it destabilizes the Ca^{2+} in the binding pocket. Second, after the Ca^{2+} has left the binding pocket, it destabilizes the folded loop conformation and thereby reduces the likelihood of Ca^{2+} rebinding.

The second pH sensor

In the ITC experiments the H294A mutant exhibits a pH sensitivity of ΔG = 3.2 kJ/mol, even though the pH sensor H294 is missing (17). This suggests that langerin has a second pH sensor. To convince ourselves that this residual pH sensitivity is indeed due to a second pH sensor, we checked whether K257 forms another potentially pH-sensitive hydrogen bond in the H294A mutant that could replace the pH-sensitive K257–H294 hydrogen bond and explain the residual pH sensitivity. In our simulations of the H294A mutant, K257 does not form any highly populated hydrogen bond. With 13% population the hydrogen bond between the side chain of K257 and the main-chain carbonyl group of E293 is the most frequently formed hydrogen bond. However, in wildtype langerin it is formed with the same frequency. All other hydrogen bonds of K257 are populated with less than 5%. Thus, the experimentally determined pH sensitivity in the H294A mutant does indeed indicate that wildtype langerin has a second pH sensor.

There are two possible mechanisms to explain the residual pH sensitivity. First, langerin could have a second allosteric pH sensor that, similar to H294, is activated by protonation from the surrounding solvent prior to the dissociation of Ca^{2+}. Second, the carboxyl groups of the Ca^{2+}-coordinating residues E285, E293, and D308 could form a dyad with an effective pK_a that makes it sensitive to a pH change from 7 to 6. That is, after initial dissociation of Ca^{2+}, one of the coordinating residues (Fig. 2*D*) is protonated and the protonated state is stabilized as a hydrogen bond to an unprotonated carboxyl group (56). We first discuss the possibility of a second allosteric pH sensor before investigating whether a dyad is possible.

H229 is the only other histidine residue in langerin. It is solvent exposed and will indeed be protonated when the pH changes from 7 to 6. However, H229 is located far away from the Ca^{2+}-binding site, which makes an allosteric influence on the Ca^{2+}-binding affinity unlikely (Fig. 1). This is further corroborated by the previously published mutual information analysis of the allosteric network in langerin and by chemical shift perturbation experiments (17). In the extended simulation data set used for this study, protonation of H229 has a local effect on the lower protein region including the α_1 helix, but these conformational shifts are well separated from the Ca^{2+}-binding site. We therefore exclude H229 as a potential pH sensor.

Other candidates for allosteric pH sensors are aspartic and glutamic acids, whose pK_a (in water at 25 °C 4.15 for E and 3.71 for D) (27) can be shifted by several units by the local environment in the protein, such that their carboxyl groups could become sensitive to a pH change from 7 to 6 (57). Apart from the Ca^{2+}-coordinating residues E285, E293, and D308, langerin has nine aspartic or glutamic acids. Using PROPKA 3.1 (58, 59), we calculated the distribution of the pK_a values for these residues in holo-langerin in the neutral and the H294-protonated state, as well as for apo-langerin in the neutral and the H294-protonated state. The distributions are based on 10,000 to 30,000 structures extracted from the simulations of the corresponding systems and are reported along with the mean and the standard deviation in the Supporting information. The mean pK_a value of all tested residues is below 5.0, and none of the distributions reaches into the critical region between pH 6 and 7, indicating that none of them acts as pH sensor. We therefore conclude that the residual pH sensitivity in langerin is not generated by a second allosteric pH sensor.

PROPKA 3.1 can detect the coupling between two carboxyl groups that are in close vicinity. It returns two alternative pK_a values. In alternative *a*, one carboxyl group is protonated first and stabilized by the second (unprotonated) carboxyl group, in alternative *b* the situation is reversed. Figure 11*A* shows the pK_a distribution of the Ca^{2+}-coordinating residues E285, E293, and D308 as well as the pK_a distribution of H294 for apo-langerin in the neutral and the H294-protonated state. No coupling between E285, E293, and D308 was detected by PROPKA 3.1. Their mean pK_a value is below 5.0, and none of the distributions reaches into the critical region between pH 6 and 7. By contrast, the mean pK_a values of H294 are about 6 in the neutral and the H294 protonated states, and the pK_a distributions have a large overlap with the critical region between pH 6 and 7. Thus, from these simulations one would conclude that langerin does not have a protonatable dyad in the Ca^{2+}-binding pocket and that only H294 is sensitive to a pH change from 7 to 6.

However, in the neutral and the H294-protonated states, the carboxyl group of the Ca^{2+}-coordinating residues are negatively charged and repel each other, making structures in which the two carboxyl groups are close enough to potentially stabilize a protonation unlikely. We therefore also calculated the pK_a distribution for the following protonation states of apo-langerin: H294 and E285 protonated (Fig. 11*B*), H294 and E293 protonated (Fig. 11*C*), and H294 and D308 protonated (Fig. 11*D*). For these protonation states, substantial coupling

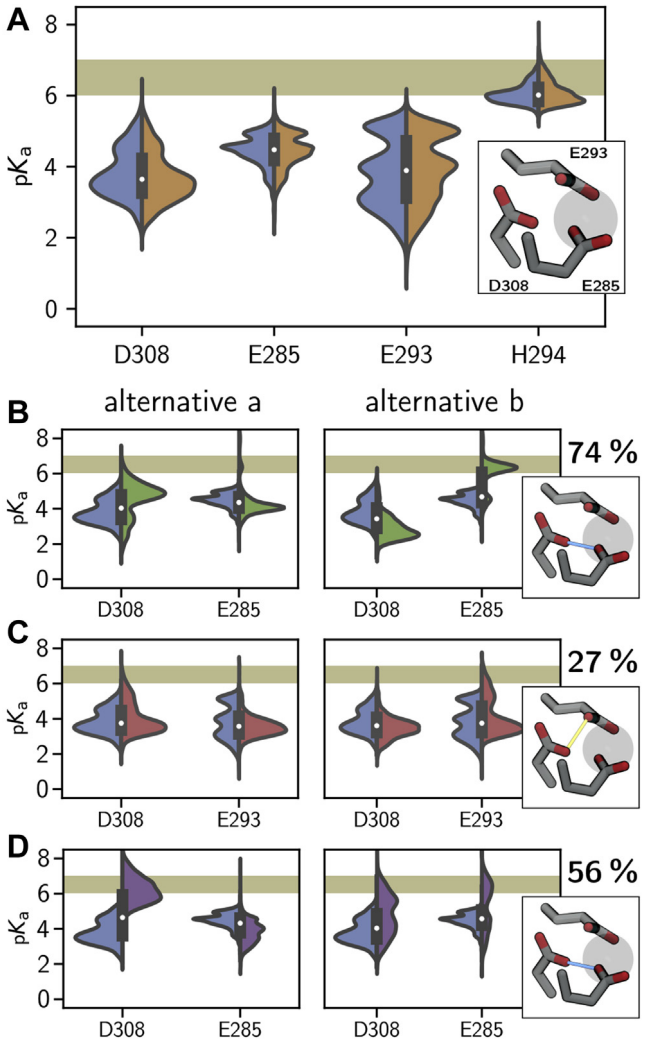


Figure 11. Calculation of pK_a values with PROPKA 3.1. *A*, pK_a distributions for the neutral (*blue*) and the H294 protonated (*orange*) apo-systems. *B*, distributions for residues involved in coupling, neutral *versus* E285 protonated (*green*), *C*, neutral *versus* E293 protonated (*red*), and *D*, neutral *versus* D308 protonated (*purple*). Alternative distributions due to the coupling *left* and *right*. Percentages of coupling frames are placed over the binding site illustrations.

between the Ca^{2+}-coordinating residues is detected. D308 and E285 couple in 74% of all structure if E285 is protonated and in 56% of all structures if D308 is protonated. When E293 is protonated, E293 and D308 couple in 27% of all structures.

These couplings give rise to a strong shift of the pK_a distributions compared with neutral apo-langerin. We report the distributions of both pK_a estimates, which should be interpreted as limiting cases of the true distribution. If D308 is protonated, the pK_a distributions of D308 for both limiting cases reach well into the critical region between pH 6 and 7, and for alternative *a* we obtain a mean pK_a value in coupling frames of 6.4 ± 0.7 (Fig. 11*D*). If E285 is protonated, the coupling to D308 in alternative *b* yields a mean pK_a value of 6.2 ± 0.6 for E285, and the corresponding distribution of all frames is almost centered on the critical region between pH 6 and 7 (Fig. 11*B*). The effect is not as strong, if E293 is protonated (Fig. 11*B*). For alternative *a* the pK_a distribution of D308 reaches slightly into the region between pH 6 and 7, and for alternative *b* the pK_a distribution of E293 reaches into this region. However, the corresponding pK_a values, 5.2 ± 0.7 and 5.5 ± 0.7, are clearly lower than those for the coupling between D308 and E285.

These results show that, in the absence of Ca^{2+}, D308 and E285 can form a protonated dyad with an effective pK_a that is likely high enough to sense a pH change from 6 to 7. We therefore believe that the second pH sensor that is active in the H294A mutant is the dyad between D308 and E285. In wild-type langerin the pH sensor H294 and this dyad would amplify each other: the K257–D308 hydrogen bond increases the probability that D308 is protonated, and, after Ca^{2+} has escaped, the protonated D308 is stabilized by the D308–E285 dyad. Constant-pH simulations (60–62) or mixed quantum mechanics/molecular mechanics simulations (63, 64) could be used to verify whether D308 and E285 indeed form a dyad and constitute the second pH sensor.

Note that the conformational fluctuations in the Ca^{2+}-binding pocket give rise to large fluctuations in the instantaneous pK_a value (Fig. 11) with some distributions covering more than six pK_a units. Thus, knowing the underlying conformational distribution is essential for a reliable estimate of the overall pK_a value.

Comparison with other C-type lectins

To gain insight into whether the proposed mechanism for the pH sensitivity is found in other C-type lectins, we compared the sequences of human langerin with mouse langerin and with human variants of 15 related C-type lectins (Fig. S31). All 16 proteins exhibit the typical C-type-lectin fold, as evidenced by crystal structures (Fig. S32). The residues D308 and E285, which form the proposed second pH sensor, are highly conserved. However, one should be careful to interpret this as evidence for a conserved second pH sensor, because these residues are also essential for the coordination of Ca^{2+} and might be conserved for this reason.

The H294–K257 motif, the primary pH sensor in langerin, is not conserved in our sequence alignment. Thus, the proposed mechanism for the pH sensitivity of the Ca^{2+} affinity *via* H294 protonation does not seem to be the most widespread mechanism to sense a change in the environment in C-type lectins. But the sequence alignment points to possible other mechanisms for sensing a change in the environment.

The selectines P-, E-, and LSECtin share the lysine K257 with langerin in the same position. In addition, the preceding threonine T256 in langerin is replaced by an arginine in these three proteins, while H294 is replaced by an aspartic acid. Note that, in LSECtin Ca^{2+} affinity increases if the pH decreases (26). It is, however, unclear whether this reversed pH sensitivity is brought about by the change of the H294–K257 motif. Other lysine residues in the short-loop in comparable positions as K257 in langerin can also be found in the macrophage C-type lectin, lung surfactant protein (SP), CD23a, Endo180, and the macrophage mannose receptor.

H294 only appears in human and mouse langerin, and is replaced by aspartic acid in most of the other C-type lectins. Instead, we find a Ca^{2+} in the position where langerin has the

H294–K257 hydrogen bond in 6 of 15 lectins in our analysis (ASGPR, MBP, DCSIGN, DCSIGNR, SP, SR). This Ca^{2+} would be partially solvent exposed even when a large entity (such as a pathogen) is bound to the C-type lectin (Fig. S32). One therefore might speculate that these lectins do not sense a change in pH but rather a change in Ca^{2+} concentration.

Several C-type lectins have histidines in other positions close to the Ca^{2+}-binding site, which might act as pH sensors *via* a different mechanism. As already mentioned, ASGPR has a histidine residue that is close to the Ca^{2+} in the primary Ca^{2+}-binding site and thereby acts as pH sensor. Furthermore, dectin-2 and the macrophage mannose receptor have a histidine residue as a neighbor to a Ca^{2+}-coordinating residue, and Endo180 and the macrophage C-type lectin have histidines at the beginning of the long-loop. Whether these histidines act as pH sensors can be tested by mutating the histidine residue and measuring the pH sensitivity of the Ca^{2+} affinity and of the carbohydrate affinity. Once a residue is confirmed as a pH sensor, the approach presented in this contribution can be used to propose a molecular mechanism for the pH sensitivity.

Conclusion

We have described the consequences of a H294 protonation in langerin and its implications for its biological function as an endocytic pattern recognition receptor. When langerin enters the acidic environment of an endosome, it releases its Ca^{2+} cofactor and subsequently its pathogenic cargo, triggered by a moderate change in pH. The Ca^{2+}-binding site is blocked from direct solvent access by the pathogen, and additionally, the Ca^{2+}-coordinating residues have low protonation probabilities in the presence of calcium. Instead, H294 acts as an accessible site, sensing already a change in pH from 7 to 6 (17).

In this contribution, we have uncovered a mechanism in which protonation of H294 perturbs the hydrogen-bonded network of the surrounding residues and alters the conformational ensemble of langerin. A new conformation becomes accessible, in which the protonated K257 side chain forms a hydrogen bond with the Ca^{2+}-coordinating D308, thereby moving a positive charge into the vicinity of the Ca^{2+}-binding site. This alone can facilitate the Ca^{2+} release as shown by the reduction in the required force to pull out the ion from its binding site in our steered-MD experiments.

The close availability of K257 as a proton source next to the Ca^{2+}-binding site possibly results in a proton transfer to the side chain of D308. At least it has been shown in a theoretical model that the neutral form of a lysine–aspartate pair can be favored over the salt bridge, if the dielectric constant of the medium is low as it can be the case in the environment of a protein (42). Thus, protonation of the initial pH sensor H294 likely triggers a cascade of events that ensures the unbinding of Ca^{2+}: K257 transfers a proton to D308, protonation of D308 competes drastically with Ca^{2+} binding and, after Ca^{2+} is expelled, the protonation of D308 is stabilized by a dyad with E285. Protonation of D308 additionally accelerates the unfolding of the long-loop, preventing Ca^{2+} from rebinding.

For langerin's role as endocytic pattern recognition receptor a fast and irreversible Ca^{2+} release is essential. On the cell surface, Ca^{2+} needs to be tightly bound such that the receptor is continuously ready to bind to pathogens. Yet, after endocytosis langerin is probably recycled within minutes (13, 65). This leaves little time for the release of the pathogen, which must be preceded by the unbinding of Ca^{2+}. The mechanism that we proposed is an elegant solution to these contradicting requirements: the Ca^{2+}-unbinding rate is increased by the K257–D308 hydrogen bond, and after the initial Ca^{2+} release, a transfer of the proton from K257 to D308 triggers a transition to a conformation to which Ca^{2+} cannot rebind.

Note that, although our results show that the K257–D308 interaction decreases the stability of Ca^{2+} in the binding pocket and that the protonation of D308 triggers the long-loop unfolding, the transfer of a proton from K257 to D308 is currently an assumption. More work is needed to study the equilibrium between the initial and the end states of the proton transfer. Computationally, this could be tackled by mixed quantum mechanics/molecular mechanics calculations (63, 64), free-energy calculations with classical force fields, or by constant pH simulations (60–62).

Another concern is that the point charge Ca^{2+} model might not capture the energetics of Ca^{2+} binding accurately enough, because the point charge model does not enforce coordination and neglects polarization effects. In our study, we tried to minimize the influence of these force-field effects by analyzing the differences between two protonation states. However, more elaborate Ca^{2+} models such as reparametrized point-charge models (66, 67), multisite models (68), or polarizable models (69) are available and should be used, for example, for the computation of state-specific Ca^{2+} binding free energies.

Our close atomistic inspection of langerin and its conformational shift upon protonation gives insight into how pH sensitivity can be incorporated in biological systems. What seemed like a general conformational shift upon protonation in Figure 3 could be focused to a specific rearrangement of a side chain (K257) to transport the information from the primary pH sensor (H294) to the allosterically regulated site (Ca^{2+}-binding site). Even though the H294–K257 motif is not typical for C-type lectins, many of these proteins exhibit a highly specific pH sensitivity and have potential pH sensors in the vicinity of the primary Ca^{2+}-binding site. Our approach can serve as a road map to elucidate the mechanism of pH sensitivity in these systems.

Experimental procedures

Molecular dynamics simulations

We used the software package GROMACS (70–76) in setup and production to simulate the considered systems in the *NPT*-ensemble (1 bar, 300 K) with AMBER99SB-ILDN force-field parameters (77) and the TIP3P water model (78). Prior to production, starting structures were put into a sufficiently large simulation box, solvated, neutralized, and equilibrated for several hundred picoseconds. For further details refer to the Supporting information.

Protein expression and purification

All standard chemicals and buffers used within this work were purchased from Sigma-Aldrich or Carl Roth if not indicated otherwise.

Human langerin CRD WT and all mutants (amino acids 193–328) were cloned from a codon-optimized langerin gene for bacterial expression (GenScript) into a pET-28a vector (GenScript) with His-tag, T7 promoter, and Kanamycin resistance. Insoluble expression was performed in *E. coli* BL21 (Thermo Fisher Scientific) in LB medium or in isotope-labeled M9 medium at 37 °C. Protein expression was induced by adding 1 mM IPTG. Bacteria were harvested 3 to 4 h after induction by centrifugation at 4000*g* for 30 min. Cell pellets were lysed in lysis buffer (50 mM Tris, 150 mM NaCl, 10 mM $MgCl_2$, 0.1% Tween-20, pH 8) with 1 mg ml^{-1} lysozyme and 100 µg ml^{-1} DNase I (Applichem) for at least 3 h at RT. Inclusion bodies were washed twice with 20 to 30 ml lysis buffer and twice with water to remove soluble proteins. Inclusion bodies were denatured in 20 ml of denaturation buffer (6 M guanidinium hydrochloride in 100 mM Tris, pH 8) with 0.01% β-mercaptoethanol for at least 1 h at 37 °C by shaking or overnight at 4 °C by rotating. After centrifuging (15,000*g*, 90 min, 4 °C), the supernatant was slowly diluted 1:10 with langerin refolding buffer (0.4 M L-arginine in 50 mM Tris, 20 mM NaCl, 0.8 mM KCl, pH 7.5) with 1 mM reduced glutathione (GSH) and 0.1 mM oxidized glutathione (GSSG) while stirring at 4 °C for at least 24 h. The refolded protein solution was dialyzed against 2 l TBS buffer (50 mM Tris, 150 mM NaCl, 5 mM CaCl2) and subsequently centrifuged to remove precipitated protein (15,000*g*, 90 min, 4 °C). The supernatant was purified *via* Ni-NTA agarose affinity chromatography and the elution fractions were pooled and dialyzed against MES (25 mM MES, 40 mM NaCl, pH 6) or HBS (25 mM Hepes, 150 mM NaCl, pH 7) buffer. Precipitated protein was removed by centrifugation (15,000*g*, 90 min, 4 °C), and the supernatant was used for experiments. Note that this procedure varies slightly from the one in our previous paper (17).

ITC measurements

ITC experiments were performed using a MicroCal iTC200 (Malvern Instruments) using either chelex-filtered HBS (25 mM Hepes, 150 mM NaCl, pH 7) or low-salt MES buffer (25 mM MES, 40 mM NaCl, pH 6). The titrant was dissolved in the same buffer as was used for dialysis of the protein sample. Using the iTC200, the titrant $CaCl_2$ (15 mM stock) was added in defined steps of 1 to 2.5 µl to 80 µl protein solution at 298 K while stirring at 750 rpm. The differential heat of each injection was measured and plotted against the molar ratio. The data were fitted to a one-set of sites binding model assuming a Hill coefficient of 1. Owing to the low c-values of the measurements ($c < 5$), the enthalpy could not be determined reliably. See also Figs. S29 and S30.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request. All remaining data are contained within the article and its supporting information. The software used for common-nearest-neighbor clustering and core-set Markov-state model estimation is publicly available on GitHub (https://github.com/janjoswig/CommonNNClustering).

Supporting information—This article contains supporting information (16, 36, 38, 58, 59, 70–90).

Acknowledgments—Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2008 – 390540038 – UniSysCat. The authors thank the North-German Supercomputing Alliance (HLRN), the Paderborn Center for Parallel Computing PC^2, and the ZEDAT of the FU Berlin for computing time. Also funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through CRC 765, and DFG RA1944/6-1. We thank the Max Planck Society for support.

Author contributions—J.-O. J. performed the computer experiments, except for the simulations of the long-loop unfolding, analyzed and interpreted the data, made all figures, and drafted and revised the manuscript. J. A. performed the computer experiments for the long-loop unfolding and analyzed the data. H. Z. performed the laboratory experiments and analyzed the data. C. R. interpreted the data of the laboratory experiments and drafted and revised the corresponding part of the manuscript. B. G. K. designed the study, interpreted the data, and drafted and revised the manuscript.

Conflict of interest—The authors declare that they have no conflicts of interest with the contents of this article.

Abbreviations—The abbreviations used are: ITC, isothermal titration calorimetry; MD, molecular dynamics; tIC, time-independent component.

References

1. Valladeau, J., Duvert-Frances, V., Pin, J. J., Dezutter- Dambuyant, C., Vincent, C., Massacrier, C., Vincent, J., Yoneda, K., Banchereau, J., Caux, C., Davoust, J., and Saeland, S. (1999) The monoclonal antibody DCGM4 recognizes Langerin, a protein specific of Langerhans cells, and is rapidly internalized from the cell surface. *Eur. J. Immunol.* **29**, 2695–2704
2. Valladeau, J., Ravel, O., Dezutter-Dambuyant, C., Moore, K., Kleijmeer, M., Liu, Y., Duvert-Frances, V., Vincent, C., Schmitt, D., Davoust, J., Caux, C., Lebecque, S., and Saeland, S. (2000) Langerin, a novel C-type lectin specific to Langerhans cells, is an endocytic receptor that induces the formation of birbeck granules. *Immunity* **12**, 71–81
3. Zelensky, A. N., and Gready, J. E. (2003) Comparative analysis of structural properties of the C-type- lectin-like domain (CTLD). *Proteins* **52**, 466–477
4. Zelensky, A. N., and Gready, J. E. (2005) The C-type lectin-like domain superfamily. *FEBS J.* **272**, 6179–6217
5. Ng, W. C., Londrigan, S. L., Nasr, N., Cunningham, A. L., Turville, S., Brooks, A. G., and Reading, P. C. (2016) The C-type lectin Langerin functions as a receptor for attachment and infectious entry of influenza A virus. *J. Virol.* **90**, 206–221
6. van der Vlist, M., deWitte, L., de Vries, R. D., Litjens, M., de Jong, M. A. W. P., Fluitsma, D., de Swart, R. L., and Geijtenbeek, T. B. H. (2011) Human Langerhans cells capture measles virus through Langerin and

present viral antigens to CD4+ T cells but are incapable of cross-presentation. *Eur. J. Immunol.* **41**, 2619–2631
7. de Witte, L., Nabatov, A., Pion, M., Fluitsma, D., de Jong, M. A. W. P., de Gruijl, T., Piguet, V., van Kooyk, Y., and Geijtenbeek, T. B. H. (2007) Langerin is a natural barrier to HIV-1 transmission by Langerhans cells. *Nat. Med.* **13**, 367–371
8. de Jong, M. A., Vriend, L. E., Theelen, B., Taylor, M. E., Fluitsma, D., Boekhout, T., and Geijtenbeek, T. B. (2010) C-type lectin Langerin is a β-glucan receptor on human Langerhans cells that recognizes opportunistic and pathogenic fungi. *Mol. Immunol.* **47**, 1216–1225
9. Hunger, R. E., Sieling, P. A., Ochoa, M. T., Sugaya, M., Burdick, A. E., Rea, T. H., Brennan, P. J., Belisle, J. T., Blauvelt, A., Porcelli, S. A., and Modlin, R. L. (2004) Langerhans cells utilize CD1a and Langerin to efficiently present nonpeptide antigens to T cells. *J. Clin. Invest.* **113**, 701–708
10. van Dalen, R., Fuchsberger, F. F., Rademacher, C., van Strijp, J. A., and van Sorge, N. M. (2020) A common genetic variation in langerin (CD207) compromises cellular uptake of Staphylococcus aureus. *J. Innate Immun.* **12**, 191–200
11. Ribeiro, C. M. S., Sarrami-Forooshani, R., Setiawan, L. C., Zijlstra-Willems, E. M., van Hamme, J. L., Tigchelaar, W., van der Wel, N. N., Kootstra, N. A., Gringhuis, S. I., and Geijtenbeek, T. B. H. (2016) Receptor usage dictates HIV-1 restriction by human TRIM5α in dendritic cell subsets. *Nature* **540**, 448–452
12. Sorkin, A., and von Zastrow, M. (2002) Signal transduction and endocytosis: Close encounters of many kinds. *Nat. Rev. Mol. Cell Biol.* **3**, 600–614
13. Cote, R., Lynn Eggink, L., and Kenneth Hoober, J. (2017) CLEC receptors, endocytosis and calcium signaling. *AIMS Allergy Immunol.* **1**, 207–231
14. Onizuka, T., Shimizu, H., Moriwaki, Y., Nakano, T., Kanai, S., Shimada, I., and Takahashi, H. (2012) NMR study of ligand release from asialoglycoprotein receptor under solution conditions in early endosomes. *FEBS J.* **279**, 2645–2656
15. Gerasimenko, J. V., Tepikin, A. V., Petersen, O. H., and Gerasimenko, O. V. (1998) Calcium uptake via endocytosis with rapid release from acidifying endosomes. *Curr. Biol.* **8**, 1335–1338
16. Feinberg, H., Powlesland, A. S., Taylor, M. E., and Weis, W. I. (2010) Trimeric structure of langerin. *J. Biol. Chem.* **285**, 13285–13293
17. Hanske, J., Aleksic, S., Ballaschk, M., Jurk, M., Shanina, E., Beerbaum, M., Schmieder, P., Keller, B. G., and Rademacher, C. (2016) Intradomain allosteric network modulates calcium affinity of the C-type lectin receptor langerin. *J. Am. Chem. Soc.* **138**, 12176–12186
18. Stambach, N. S., and Taylor, M. E. (2003) Characterization of carbohydrate recognition by langerin, a C-type lectin of Langerhans cells. *Glycobiology* **13**, 401–410
19. Loeb, J. A., and Drickamer, K. (1988) Conformational changes in the chicken receptor for endocytosis of glycoproteins. *J. Biol. Chem.* **263**, 9752–9760
20. Wragg, S., and Drickamer, K. (1999) Identification of amino acid residues that determine pH dependence of ligand binding to the asialoglycoprotein receptor during endocytosis. *J. Biol. Chem.* **274**, 35400–35406
21. Mullin, N. P., Hall, K. T., and Taylor, M. E. (1994) Characterization of ligand binding to a carbohydraterecognition domain of the macrophage mannose receptor. *J. Biol. Chem.* **269**, 28405–28413
22. Guo, Y., Feinberg, H., Conroy, E., Mitchell, D. A., Alvarez, R., Blixt, O., Taylor, M. E., Weis, W. I., and Drickamer, K. (2004) Structural basis for distinct ligand-binding and targeting properties of the receptors DC-SIGN and DC-SIGNR. *Nat. Struct. Mol. Biol.* **11**, 591–598
23. Tabarani, G., Thépaut, M., Stroebel, D., Ebei, C., Vivès, C., Vachette, P., Durand, D., and Fieschi, F. (2009) DC-SIGN neck domain is a pH-sensor controlling oligomerization. SAXS and hydrodynamic studies of extracellular domain. *J. Biol. Chem.* **284**, 21229–21240
24. Probert, F., Mitchell, D. A., and Dixon, A. M. (2014) NMR evidence for oligosaccharide release from the dendritic-cell specific intercellular adhesion molecule 3-grabbing non-integrin-related (CLEC4M) carbohydrate recognition domain at low pH. *FEBS J.* **281**, 3739–3750
25. Mitchell, D. A., Fadden, A. J., and Drickamer, K. (2001) A novel mechanism of carbohydrate recognition by the C-type lectins DC-SIGN and DC-SIGNR. Subunit organisation and binding to multivalent ligands. *J. Biol. Chem.* **276**, 28939–28945
26. Powlesland, A. S., Fisch, T., Taylor, M. E., Smith, D. F., Tissot, B., Dell, A., Pöhlmann, S., and Drickamer, K. (2008) A novel mechanism for LSECtin binding to Ebola virus surface glycoprotein through truncated glycans. *J. Biol. Chem.* **283**, 593–602
27. Lide, R. D., ed. (2006) *CRC Handbook of Chemistry and Physics*, 87th Ed, CRC Press, West Palm Beach, FL
28. Hyland, L. J., Tomaszek, T. A., and Meek, T. D. (1991) Human immunodeficiency virus-1 protease. 2. Use of pH rate studies and solvent kinetic isotope effects to elucidate details of chemical mechanism. *Biochemistry* **30**, 8454–8463
29. Torbeev, V. Y., and Kent, S. B. H. (2012) Ionization state of the catalytic dyad asp25/250 in the HIV-1 protease: NMR studies of site-specifically 13C labelled HIV-1 protease prepared by total chemical synthesis. *Org. Biomol. Chem.* **10**, 5887
30. Toulokhonova, L., Metzler, W. J., Witmer, M. R., Copeland, R. A., and Marcinkeviciene, J. (2003) Kinetic studies on β-site amyloid precursor proteincleaving enzyme (BACE). *J. Biol. Chem.* **278**, 4582–4589
31. Huang, Y., Yue, Z., Tsai, C.-C., Henderson, J. A., and Shen, J. (2018) Predicting catalytic proton donors and nucleophiles in enzymes: How adding dynamics helps elucidate the structure–function relationships. *J. Phys. Chem. Lett.* **9**, 1179–1184
32. Yamazaki, T., Nicholson, L. K., Torchia, D. A., Wingfield, P., Stahl, S. J., Kaufman, J. D., Eyermann, C. J., Hodge, N. C., Lam, P. Y. S., Ru, Y., Jadhav, P. K., Chang, C. H., and Weber, P. C. (1994) NMR and X-ray evidence that the HIV protease catalytic aspartyl groups are protonated in the complex formed by the protease and a non-peptide cyclic urea-based inhibitor. *J. Am. Chem. Soc.* **116**, 10791–10792
33. Keller, B. G., and Rademacher, C. (2020) Allostery in C-type lectins. *Curr. Opin. Struct. Biol.* **62**, 31–38
34. Drickamer, K. (1992) Engineering galactose-binding activity into a C-type mannose-binding protein. *Nature* **360**, 183–186
35. Drickamer, K., and Taylor, M. E. (2015) Recent insights into structures and functions of C-type lectins in the immune system. *Curr. Opin. Struct. Biol.* **34**, 26–34
36. Li, L., Li, C., Zhang, Z., and Alexov, E. (2013) On the dielectric "constant" of proteins: Smooth dielectric function for macromolecular modeling and its implementation in DelPhi. *J. Chem. Theory Comput.* **9**, 2126–2136
37. Jolliffe, I. T. (2002) *Principal Component Analysis*, 2nd Ed, Springer, New York, NY
38. Scherer, M. K., Trendelkamp-Schroer, B., Paul, F., Pérez-Hernández, G., Hoffmann, M., Plattner, N., Wehmeyer, C., Prinz, J.-H., and Noé, F. (2015) PyEMMA 2: A software package for estimation, validation, and analysis of Markov models. *J. Chem. Theory Comput.* **11**, 5525–5542
39. Keller, B. G., Daura, X., and van Gunsteren, W. F. (2010) Comparing geometric and kinetic cluster algorithms for molecular simulation data. *J. Chem. Phys.* **132**, 074110
40. Lemke, O., and Keller, B. G. (2016) Density-based cluster algorithms for the identification of core sets. *J. Chem. Phys.* **145**, 164104
41. Lemke, O., and Keller, B. G. (2018) Common nearest neighbor clustering – a benchmark. *Algorithms* **11**, 19
42. Nagy, P. I., and Erhardt, P. W. (2010) Theoretical studies of salt-bridge formation by amino acid side chains in low and medium polarity environments. *J. Phys. Chem. B* **114**, 16436–16442
43. Schütte, C., Noé, F., Lu, J., Sarich, M., and Vanden- Eijnden, E. (2011) Markov state models based on milestoning. *J. Chem. Phys.* **134**, 204105
44. Prinz, J.-H., Wu, H., Sarich, M., Keller, B., Senne, M., Held, M., Chodera, J. D., Schütte, C., and Noé, F. (2011) Markov models of molecular kinetics: Generation and validation. *J. Chem. Phys.* **134**, 1–23
45. Prinz, J.-H., Keller, B., and Noé, F. (2011) Probing molecular kinetics with Markov models: Metastable states, transition pathways and spectroscopic observables. *Phys. Chem. Chem. Phys.* **13**, 16912
46. Schwantes, C. R., and Pande, V. S. (2013) Improvements in Markov state model construction reveal many non-native interactions in the folding of NTL9. *J. Chem. Theory Comput.* **9**, 2000–2009

47. Pérez-Hernández, G., Paul, F., Giorgino, T., De Fabritiis, G., and Noé, F. (2013) Identification of slow molecular order parameters for Markov model construction. *J. Chem. Phys.* **139**, 15102
48. Izrailev, S., Stepaniants, S., Isralewitz, B., Kosztin, D., Lu, H., Molnar, F., Wriggers, W., and Schulten, K. (1998). In: Deuflhard, P., Hermans, J., Leimkuhler, B., Mark, A. E., Reich, S., Skeel, R. D., eds. *Computational Molecular Dynamics: Challenges, Methods, Ideas*, Springer-Verlag, Berlin: 39–65
49. Dudko, O. K., Hummer, G., and Szabo, A. (2008) Theory, analysis, and interpretation of single-molecule force spectroscopy experiments. *Proc. Natl. Acad. Sci. U. S. A.* **105**, 15755–15760
50. Rico, F., Russek, A., González, L., Grubmüller, H., and Scheuring, S. (2019) Heterogeneous and ratedependent streptavidin–biotin unbinding revealed by high-speed force spectroscopy and atomistic simulations. *Proc. Natl. Acad. Sci. U. S. A.* **116**, 6594–6601
51. Cheng, F., Shen, J., Luo, X., Jiang, H., and Chen, K. (2002) Steered molecular dynamics simulations on the "tail helix latch" hypothesis in the gelsolin activation process. *Biophys. J.* **83**, 753–762
52. Guzmán, D. L., Roland, J. T., Keer, H., Kong, Y. P., Ritz, T., Yee, A., and Guan, Z. (2008) Using steered molecular dynamics simulations and single-molecule force spectroscopy to guide the rational design of biomimetic modular polymeric materials. *Polymer* **49**, 3892–3901
53. Nielbo, S., Thomsen, J. K., Graversen, J. H., Jensen, P. H., Etzerodt, M., Poulsen, F. M., and Thøgersen, H. C. (2004) Structure of the plasminogen kringle 4 binding calcium-free form of the C-type lectin-like domain of tetranectin. *Biochemistry* **43**, 8636–8643
54. Poget, S. F., Freund, S. M. V. V., Howard, M. J., and Bycroft, M. (2001) The ligand-binding loops in the tunicate C-type lectin TC14 are rigid. *Biochemistry* **40**, 10966–10972
55. Ng, K. K. S., Park-Snyder, S., and Weis, W. I. (1998) Ca2+-dependent structural changes in C-type mannosebinding proteins. *Biochemistry* **37**, 17965–17976
56. Kim, M. O., Blachly, P. G., and McCammon, J. A. (2015) Conformational dynamics and binding free energies of inhibitors of BACE-1: From the perspective of protonation equilibria. *PLoS Comput. Biol.* **11**, 1–28
57. Pace, C. N., Grimsley, G. R., and Scholtz, J. M. (2009) Protein ionizable groups: pK values and their contribution to protein stability and solubility. *J. Biol. Chem.* **284**, 13285–13289
58. Søndergaard, C. R., Olsson, M. H. M., Rostkowski, M., and Jensen, J. H. (2011) Improved treatment of ligands and coupling effects in empirical calculation and rationalization of pKa values. *J. Chem. Theory Comput.* **7**, 2284–2295
59. Olsson, M. H. M., Søndergaard, C. R., Rostkowski, M., and Jensen, J. H. (2011) PROPKA3: Consistent treatment of internal and surface residues in empirical pKa predictions. *J. Chem. Theory Comput.* **7**, 525–537
60. Khandogin, J., and Brooks, C. L. (2005) Constant pH molecular dynamics with proton tautomerism. *Biophys. J.* **89**, 141–157
61. Lee, J., Miller, B. T., Damjanovic, A., and Brooks, B. R. (2015) Enhancing constant-pH simulation in explicit solvent with a two-dimensional replica exchange method. *J. Chem. Theory Comput.* **11**, 2560–2574
62. Radak, B. K., Chipot, C., Suh, D., Jo, S., Jiang, W., Phillips, J. C., Schulten, K., and Roux, B. (2017) Constant-pH molecular dynamics simulations for large biomolecular systems. *J. Chem. Theory Comput.* **13**, 5933–5944
63. Paasche, A., Schirmeister, T., and Engels, B. (2013) Benchmark study for the cysteine–histidine proton transfer reaction in a protein environment: Gas phase, COSMO, QM/MM approaches. *J. Chem. Theory Comput.* **9**, 1765–1777
64. Duster, A. W., and Lin, H. (2019) Tracking proton transfer through titratable amino acid side chains in adaptive QM/MM simulations. *J. Chem. Theory Comput.* **15**, 5794–5809
65. Jonker, C. T. H., Deo, C., Zager, P. J., Tkachuk, A. N., Weinstein, A. M., Rodriguez-Boulan, E., Lavis, L. D., and Schreiner, R. (2020) Accurate measurement of fast endocytic recycling kinetics in real time. *J. Cell Sci.* **133**, jcs231225
66. Yoo, J., Wilson, J., and Aksimentiev, A. (2016) Improved model of hydrated calcium ion for molecular dynamics simulations using classical biomolecular force fields. *Biopolymers* **105**, 752–763
67. Timr, S., Kadlec, J., Srb, P., Ollila, O. H. S., and Jungwirth, P. (2018) Calcium sensing by recoverin: Effect of protein conformation on ion affinity. *J. Phys. Chem. Lett.* **9**, 1613–1619
68. Saxena, A., and Sept, D. (2013) Multisite ion models that improve coordination and free energy calculations in molecular dynamics simulations. *J. Chem. Theory Comput.* **9**, 3538–3542
69. Jing, Z., Liu, C., Cheng, S. Y., Qi, R., Walker, B. D., Piquemal, J.-P., and Ren, P. (2019) Polarizable force fields for biomolecular simulations: Recent advances and applications. *Annu. Rev. Biophys.* **48**, 371–394
70. Berendsen, H. J. C., van der Spoel, D., and vanDrunen, R. (1995) GROMACS: A message-passing parallel molecular dynamics implementation. *Comput. Phys. Commun.* **91**, 43–56
71. Lindahl, E., Hess, B., and van der Spoel, D. (2001) GROMACS 3.0: A package for molecular simulation and trajectory analysis. *J. Mol. Model.* **7**, 306–317
72. van der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A. E., and Berendsen, H. J. (2005) GROMACS: Fast, flexible, and free. *J. Comput. Chem.* **26**, 1701–1718
73. Hess, B., Kutzner, C., van der Spoel, D., and Lindahl, E. (2008) GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput.* **4**, 435–447
74. Pronk, S., Páll, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., Shirts, M. R., Smith, J. C., Kasson, P. M., van der Spoel, D., Hess, B., and Lindahl, E. (2013) GROMACS 4.5: A high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* **29**, 845–854
75. Páll, S., Abraham, M. J., Kutzner, C., Hess, B., and Lindahl, E. (2015) Tackling Exascale software challenges in molecular dynamics simulations with GROMACS. In: Markidis, S., Laure, E., eds. *Solving Software Challenges for Exascale. EASC 2014. Lecture Notes in Computer Science*, **vol 8759**, Springer, Cham
76. Abraham, M. J., Murtola, T., Schulz, R., Páll, S., Smith, J. C., Hess, B., and Lindahl, E. (2015) GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* **1-2**, 19–25
77. Lindorff-Larsen, K., Piana, S., Palmo, K., Maragakis, P., Klepeis, J. L., Dror, R. O., and Shaw, D. E. (2010) Improved side-chain torsion potentials for the Amber ff99SB protein force field. *Proteins* **78**, 1950–1958
78. Jorgensen, W. L., Chandrasekhar, J., Madura, J. D., Impey, R. W., and Klein, M. L. (1983) Comparison of simple potential functions for simulating liquid water. *J. Chem. Phys.* **79**, 926–935
79. Feinberg, H., Taylor, M. E., Razi, N., McBride, R., Knirel, Y. A., Graham, S. A., Drickamer, K., and Weis, W. I. (2011) Structural basis for langerin recognition of diverse pathogen and mammalian glycans through a single binding site. *J. Mol. Biol.* **405**, 1027–1039
80. Bussi, G., Donadio, D., and Parrinello, M. (2007) Canonical sampling through velocity rescaling. *J. Chem. Phys.* **126**, 014101
81. Parrinello, M., and Rahman, A. (1981) Polymorphic transitions in single crystals: A new molecular dynamics method. *J. Appl. Phy.* **52**, 7182–7190
82. Hess, B. (2008) P-LINCS: A parallel linear constraint solver for molecular simulation. *J. Chem. Theory Comput.* **4**, 116–122
83. van Gunsteren, W. F., and Berendsen, H. J. C. (1988) A leap-frog algorithm for stochastic dynamics. *Mol. Simulat.* **1**, 173–185
84. Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., and Pedersen, L. G. (1995) A smooth particle mesh Ewald method. *J. Chem. Phys.* **103**, 8577–8593
85. Blomberg, F., Maurer, W., and Rüterjans, H. (1977) Nuclear magnetic resonance investigation of 15N-labeled histidine in aqueous solution. *J. Am. Chem. Soc.* **99**, 8149–8159
86. Hass, M. A. S., Hansen, D. F., Christensen, H. E. M., Led, J. J., and Kay, L. E. (2008) Characterization of conformational exchange of a histidine side chain: Protonation, rotamerization, and tautomerization of His61 in plastocyanin from Anabaena variabilis. *J. Am. Chem. Soc.* **130**, 8460–8470
87. Hansen, A. L., and Kay, L. E. (2014) Measurement of histidine pKa values and tautomer populations in invisible protein states. *Proc. Natl. Acad. Sci. U. S. A.* **111**, E1705–E1712

88. Humphrey, W., Dalke, A., and Schulten, K. (1996) VMD: Visual molecular dynamics. *J. Mol. Graphics* **14**, 33–38
89. Berendsen, H. J. C., Postma, J. P. M., van Gunsteren, W. F., DiNola, A., and Haak, J. R. (1984) Molecular dynamics with coupling to an external bath. *J. Chem. Phys.* **81**, 3684–3690
90. Dogan, T., MacDougall, A., Saidi, R., Poggioli, D., Bateman, A., O'Donovan, C., and Martin, M. J. (2016) UniProt-DAAC: domain architecture alignment and classification, a new method for automatic functional annotation in UniProtKB. *Bioinformatics* **32**, 2264–2271

Jan-Oliver Joswig is a doctoral student at Freie Universität Berlin in the Molecular Dynamics Group led by Prof. Dr Bettina Keller. In his research he focuses on the application of molecular simulations to biomolecular systems with complex dynamics. In addition, he develops software solutions for advanced analyses methods, *e.g.*, density-based clustering for the construction of kinetic models. GitHub: https://github.com/janjoswig.

The molecular basis for the pH-dependent calcium affinity of the pattern recognition receptor langerin – Supporting information

Jan-O. Joswig[1], Jennifer Anders[1], Hengxi Zhang[1234], Christoph Rademacher[1234], Bettina G. Keller[1*]

[1]Department of Biology, Chemistry, and Pharmacy, Freie Universität Berlin, Arnimallee 22, 14195 Berlin, Germany

[2]Department of Biomolecular Systems, Max Planck Institute of Colloids and Interfaces, Am Mühlenberg 1, 14424 Potsdam, Germany

[3]Department of Pharmaceutical Chemistry, University of Vienna, Althanstraße 14, 1090 Vienna, Austria

[4]Department of Microbiology and Immunobiology, Max F. Perutz Laboratories, University of Vienna, Campus Vienna Biocenter 5, 1030 Vienna, Austria

[*] Corresponding author: bettina.keller@fu-berlin.de

Molecular dynamics simulations

With small deviations in the individual setup procedure, all considered systems have been prepared according to the same general protocol. Langerins carbohydrate recognition domain (CRD) consisting of the residues G198 to P325 is the basis for all simulations. Initial starting conformations were derived from PDB crystal structures of holo-langerin (3p5h, 3p5g or 3kqg) (1, 2) by removing crystal water, ligands and excessive residues and ions as needed. Further starting structures were manually selected from completed simulations focusing on rarely visited conformations. GROMACS 2016+ (3–9) was used for the subsequent tasks. Desired protonation states of the protein were automatically attained during topology preparation. As force field for the protein and the Ca^{2+}-cofactor AMBER99SB-ILDN (10) was chosen. The system was put into a dodecahedral box at a distance of at least 1 nm to the box borders, solvated in explicit TIP3P water (11) and minimized (steepest decent, eps $< 1000\,\mathrm{kJ\,mol^{-1}\,nm^{-1}}$) after the replacement of water by chloride to neutralize contingent charges. Equilibration was done with position restraints on all heavy atoms of the solute in the *NVT*- (300 K, V-rescale thermostat (12), coupling rate at 0.1 ps, coupling groups protein and non-protein, 100 ps length) and *NPT*-ensemble (1 bar, Parrinello-Rahman barostat (13), isotropic, coupling rate at 2 ps, 150 ps length). For both, equilibration and production, the LINCS (14) (order 4, 1 iteration) algorithm was applied to constrain all bonds. The leap-frog integrator (15) was used at a time step of 2 fs. For Lennard-Jones (cut-off) and electrostatic (PME (16), order 4) interactions the cut-off was set to at least 1 nm, while the Verlet cut-off scheme was used to create the neighbor lists. Periodic boundary conditions were imposed in all three dimensions. Protein/calcium coordinates were written to a compressed trajectory file at a time step of 1 ps. The individual length of the production simulations varies between 0.15 and 1 µs as listed in the respective tables below (see Tab. 2 to 12). In cases where the total number of replicas and the number of starting structures do not match, multiple simulations have been started from the same structure with different initial velocities.

System nomenclature

To distinguish the langerin systems under consideration we denote the cofactor bound state with one letter: a – apo, h – holo. For the protonation state we use a summation scheme in which each protonation site contributes a specific increment. For example, the H229 protonation adds the value 1, while the H294 protonation adds the value 2. If both protonations are present, this is indicated by the sum (1 + 2, e.g. in h3). Refer to table 1 for an overview. Finally we numerate the different mutations, e.g. m1 for H294A. Later we also differentiate the folding state of the long-loop in italic: *f* – folded, *uf* – unfolded. Similarly we mark the key conformation of the H294 protonated systems where the K257–D308 and the H294–E261 hydrogen bonds are formed (quoted as the "green" PCA-cluster in the main document) with an asterisk (*).

Table 1: Langerin system state notation scheme.

bound	protonation	mutation	conformation
h: holo	H229: 1	m1: H294A	*f*: folded
a: apo	H294: 2	m2: E261D	*uf*: unfolded
	E285: 4	m3: K257A	*: K257s–D308s/H294s–E261s
	E293: 8		
	D308: 16		

Neutral holo-langerin (h0)

In the neutral protein state, the side-chains of the acidic residues aspartic and glutamic acid are modeled as deprotonated (charge −1) and those of the basic residues arginine and lysine as protonated (charge +1). All other residues including histidine are kept neutral. The presence of the bound calcium ion results in a net-charge of the system of +2, which was neutralized by two chloride atoms in solution. For neutral histidine two different tautomers, the δ- and ϵ-state exist. In high pH solutions of the free acid the ratio of the tautomers is ∼1 : 4 in favor of the ϵ-form. In a protein environment, however, the ratio depends strongly on stabilization factors like possible hydrogen bond formation (17–19). For each simulation starting structure the corresponding state was automatically determined during topology preparation, always resulting in the ϵ-form for H294. For H229 the used form is denoted in table 2.

Table 2: Simulation overview for h0.

#Replica	#Starting structures		#δ	#ϵ	$t/\mu s$	$t_{tot}/\mu s$
1		1	0	1	1	1
1		1	0	1	0.58	0.58
5[a]		5	5	0	0.4	2
5		1	5	0	0.25	1.25
119		14	43	76	0.22	26.18
2		1	1	1	0.15	0.3
133		23	54	79		31[b]

[a] Simulations reused from our previous paper (20), which uses a slightly different preparation protocol. [b] About 1.2 µs of the simulation data contain no calcium output coordinates. Calcium in the binding pocket is present during the simulation but the trajectories cannot be used for analyses that include the calcium positions.

Histidine protonated holo-langerin (h3)

In the histidine protonated protein state, the side-chains of H229 and H294 are both modeled as protonated. All other protonation states are as in h0. The presence of the bound calcium ion and the two side-chain protonations results in a net-charge of the system of +4, which was neutralized by four chloride atoms in solution.

Table 3: Simulation overview for h3.

#Replica	#Starting structures	$t/\mu s$	$t_{tot}/\mu s$
1	1	1	1
1	1	0.63	0.63
7	1	0.25	1.75
104	15	0.22	22.88
2	1	0.2	0.4
2	1	0.15	0.3
117	20		27[b]

[b] About 6.3 µs of the simulation data contain no calcium output coordinates. Calcium in the binding pocket is present during the simulation but the trajectories cannot be used for analyses that include the calcium positions.

Neutral holo-langerin H294A mutant (h0m1)

The H294A mutant was created in silico using the VMD (21) molefacture plugin from the crystal structure. The presence of the bound calcium ion results in a net-charge of the system of +2, which was neutralized by two chloride atoms in solution.

Table 4: Simulation overview for h0m1.

#Replica	#Starting structures	$t/\mu s$	$t_{tot}/\mu s$
10	10	0.25	2.5[a]
20	1	0.25	5
30	11		7.5

[a] Simulations reused from our previous paper (20), which uses a slightly different preparation protocol.

Neutral apo-langerin (a0)

The neutral apo-state was prepared in the same way as the neutral holo-state after manual deletion of the Ca^{2+}-ion from the structure. Note that in all simulations H229 is modeled in its δ-form, while H294 always remains in its ϵ-form.

Table 5: Simulation overview for a0.

#Replica	#Starting structures	$t/\mu s$	$t_{tot}/\mu s$
1	1	1	1
6	2	0.5	3.0
4	1	0.25	1
22	2	0.22	4.84
4	1	0.1	0.4
37	7		10

Histidine protonated apo-langerin (a3)

The protonated apo-state was prepared in the same way as the protonated holo-state after manual deletion of the Ca^{2+}-ion from the structure. The presence of two protonations results in a net-charge of the system of +2, which was neutralized by two chloride atoms in solution.

Table 6: Simulation overview for a3.

#Replica	#Starting structures	$t/\mu s$	$t_{tot}/\mu s$
1	1	1	1
1	1	0.5	0.5
6	1	0.25	1.5
51	8	0.22	11.22
1	1	0.15	0.15
60	12		14

Histidine and glutamic acid 285 protonated apo-langerin (a6, a7)

The apo-state a6 is protonated at H294 and E285. The presence of two protonations results in a net-charge of the system of +2, which was neutralized by two chloride atoms in solution.

Table 7: Simulation overview for a6.

#Replica	#Starting structures	$t/\mu s$	$t_{tot}/\mu s$
4	3	0.25	1

The apo-state a7 is protonated at H229, H294 and E285. The presence of three protonations results in a net-charge of the system of +3, which was neutralized by three chloride atoms in solution.

Table 8: Simulation overview for a7.

#Replica	#Starting structures	$t/\mu s$	$t_{tot}/\mu s$
22	6	0.22	4.84
4	1	0.21	0.84
1	1	0.18	0.18
3	2	0.17	0.51
30	6		6.37

Histidine and glutamic acid 293 protonated apo-langerin (a10, a11)

The apo-state a10 is protonated at H294 and E293. The presence of two protonations results in a net-charge of the system of +2, which was neutralized by two chloride atoms in solution.

Table 9: Simulation overview for a10.

#Replica	#Starting structures	$t/\mu s$	$t_{tot}/\mu s$
4	4	0.25	1

The apo-state a11 is protonated at H229, H294 and E293. The presence of three protonations results in a net-charge of the system of +3, which was neutralized by three chloride atoms in solution.

Table 10: Simulation overview for a11.

#Replica	#Starting structures	$t/\mu s$	$t_{tot}/\mu s$
1	1	0.50	0.50
28	6	0.22	6.16
1	1	0.21	0.21
1	1	0.17	0.17
3	3	0.16	0.48
2	2	0.15	0.30
5	2	0.14	0.70
1	1	0.13	0.13
1	1	0.10	0.10
43	6		8.75

Histidine and aspartic acid 308 protonated apo-langerin (a18, a19)

The apo-state a18 is protonated at H294 and D308. The presence of two protonations results in a net-charge of the system of +2, which was neutralized by two chloride atoms in solution.

Table 11: Simulation overview for a18.

#Replica	#Starting structures	t/µs	t_{tot}/µs
4	4	0.25	1

The apo-state a19 is protonated at H229, H294 and D308. The presence of three protonations results in a net-charge of the system of +3, which was neutralized by three chloride atoms in solution.

Table 12: Simulation overview for a19.

#Replica	#Starting structures	t/µs	t_{tot}/µs
2	2	0.55	1.10
2	2	0.51	1.02
17	6	0.22	3.74
2	2	0.21	0.42
2	2	0.16	0.32
4	2	0.15	0.60
1	1	0.10	0.10
30	6		7.30

DSSP analysis

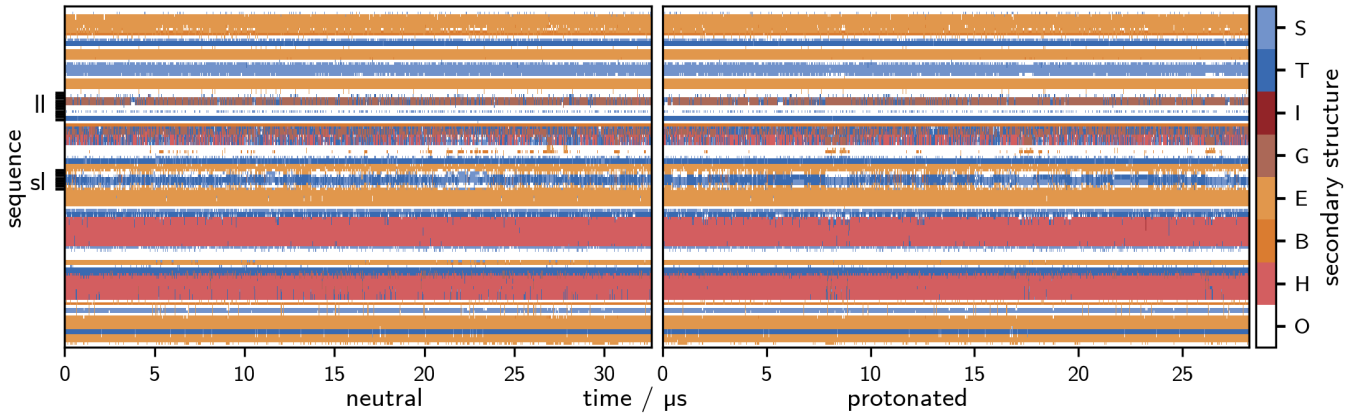


Figure 1: Analysis by the hydrogen bond estimation algorithm DSSP of the secondary structure in the neutral (left) and the protonated holo-state (right). Legend: S – bend, T – hydrogen bonded turn, I – 5-helix, G – 3-helix, E – extended strand, part of β-ladder, B: isolated β-bridge, H: α-helix, O: unassigned.

Calculation of pK_a-values with PROPKA

For selected conformational snapshots from simulations with a reduced time resolution of 1 ns (table 13), pK_a-values of titrable amino acids have been estimated using PROPKA 3.1 (22, 23), allowing to treat groups automatically as coupled where appropriate. D308 was detected in some frames non-covalently coupled to either E285 or E293 or both. In frames with no coupling, there is only one pK_a-value per residue. For frames with two coupling residues (dyad), PROPKA provides a pair of pK_a-values per residue, a higher value for the situation as proton acceptor and a lower one for the role as supporting nucleophile. We report these pairs in the way that the higher pK_a for D308 and the corresponding lower values for E285 and E293 are treated as alternative. For frames in which E285 and E293 couple to D308 at the same time (triad), PROPKA returns a set of different pK_a-values per residue based on interaction permutations. We report the minimum and the maximum of these values as a pair for each residue.

Table 13: Overview pK_a-value calculation.

			#coupling conformations		
System	#conformations	#folded	D308–E285	D308–E293	E285–D308–E293
h0	30,192	30,192	10,684	4,003	11,175
h3	20,795	20,795	7,341	2,094	8,842
a0	10,363	5,310	20	4	–
a3	14,029	7,127	26	–	–
a7	6,368	4,435	4,744	–	1
a11	8,702	6,315	22	2,333	11
a19	7,289	2,193	4,062	72	26

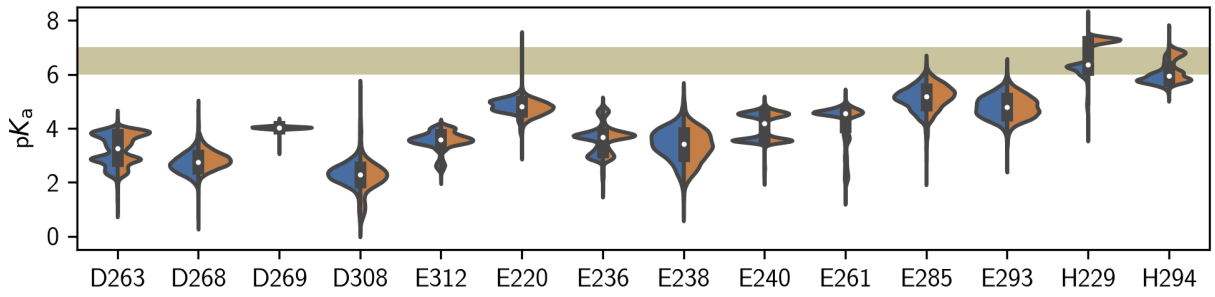


Figure 2: Calculated pK_a-values for the neutral (blue, h0) compared to the H294 protonated (orange, h3) holo-system.

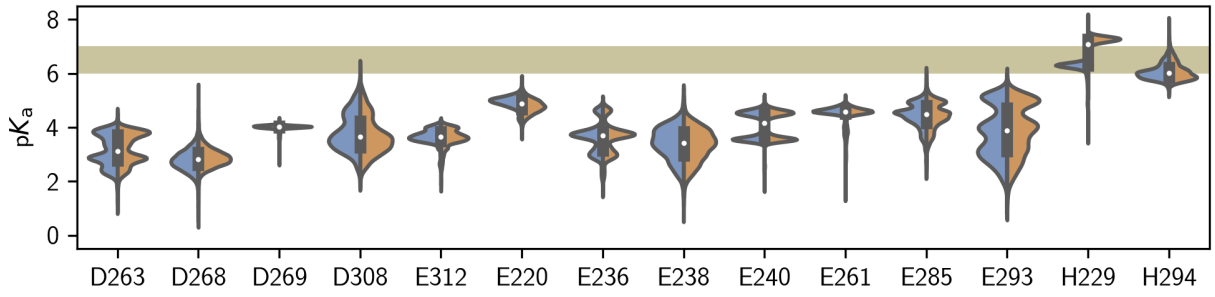


Figure 3: Calculated pK_a-values for the neutral (blue, a0) compared to the H294 protonated (orange, a3) apo-system.

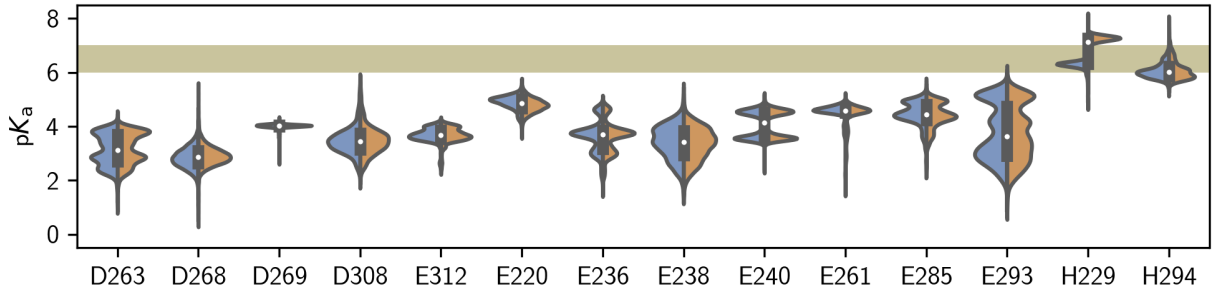


Figure 4: Calculated pK_a-values for the neutral (blue, a0) compared to the H294 protonated (orange, a3) apo-system where only folded structures are considered.

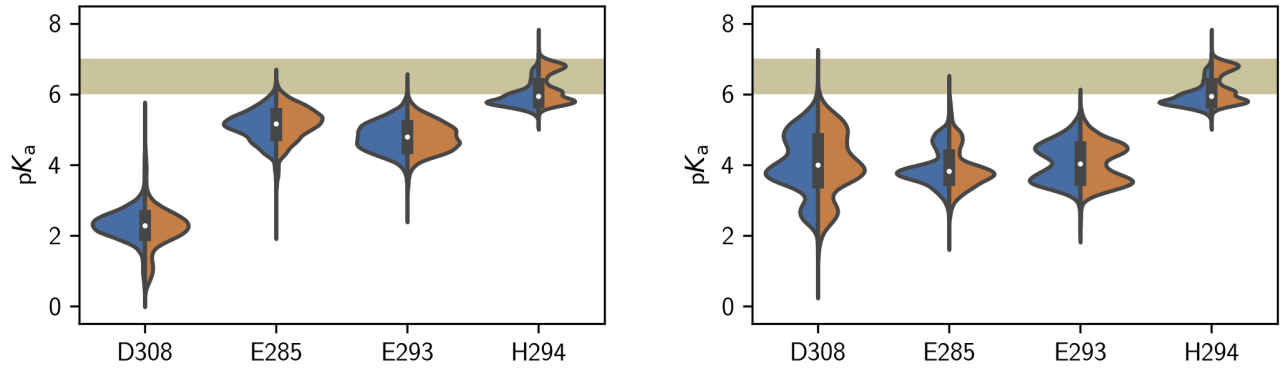


Figure 5: Calculated pK_a-values for the neutral (blue, h0) compared to the H294 protonated (orange, h3) holo-system. Alternative pK_a-values due to coupling on the right.

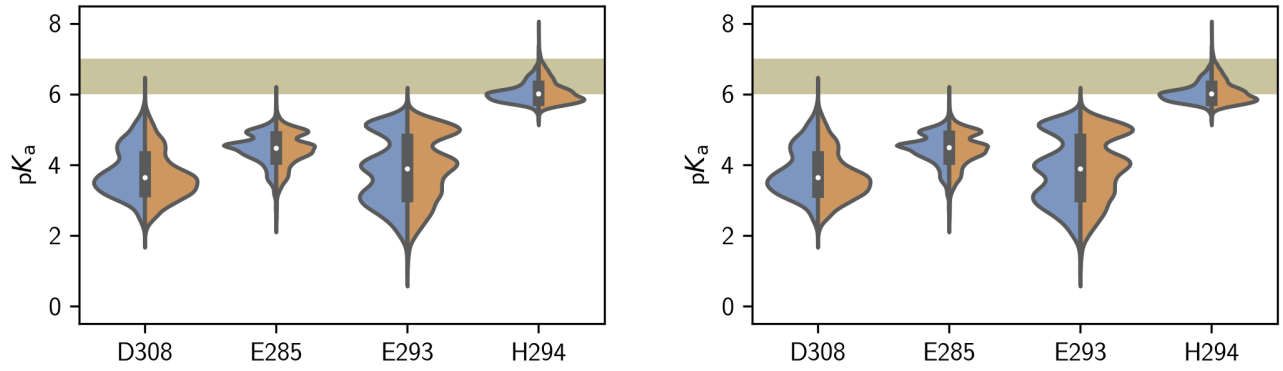


Figure 6: Calculated pK_a-values for the neutral (blue, a0) compared to the H294 protonated (orange, a3) apo-system. Alternative pK_a-values due to coupling on the right. Note, that due to the fact that there is almost no coupling of the considered residues observed in these two systems (compare table 13), the two alternative distributions shown left and right are essentially the same.

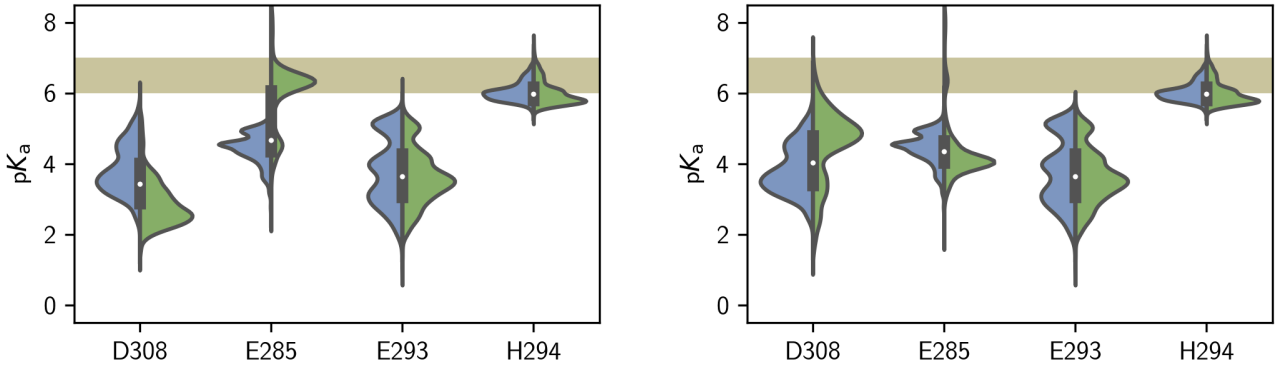


Figure 7: Calculated pK_a-values for the neutral (blue, a0) compared to the E285 protonated (green, a7) apo-system. Alternative pK_a-values due to coupling on the right.

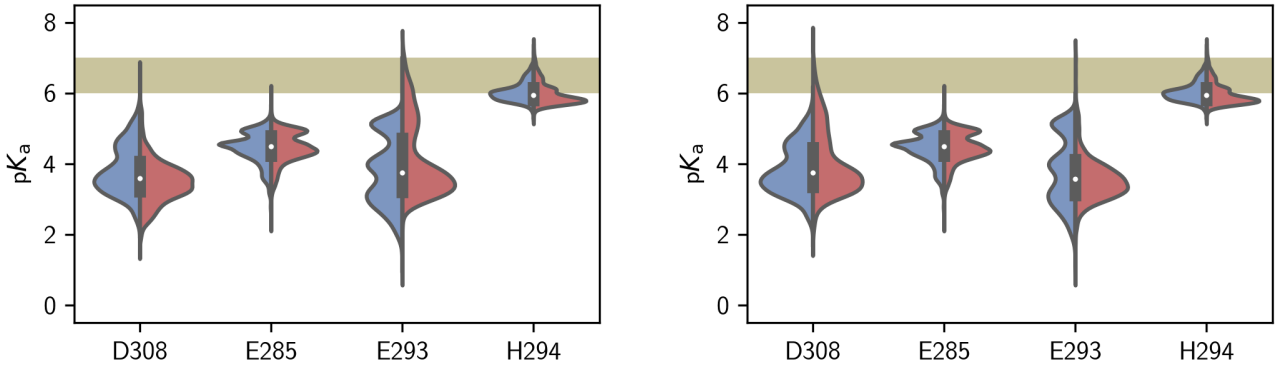


Figure 8: Calculated pK_a-values for the neutral (blue, a0) compared to the E293 protonated (red, a11) apo-system. Alternative pK_a-values due to coupling on the right.

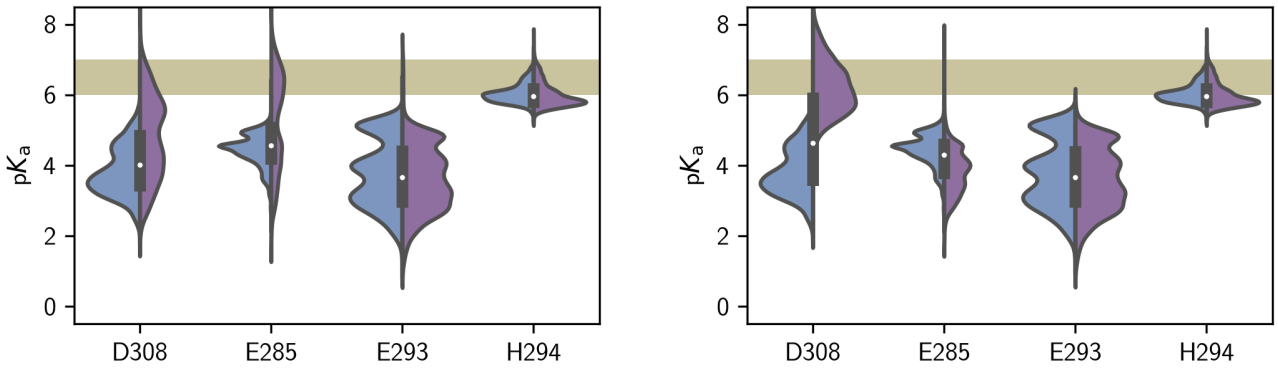


Figure 9: Calculated pK_a-values for the neutral (blue, a0) compared to the D308 protonated (purple, a19) apo-system. Alternative pK_a-values due to coupling on the right.

Table 14: Mean and standard deviation of calculated pK_a-values using PROPKA 3.1 for each ns of simulation time (all frames considered).

	pK_a $\mu \pm \sigma$						
residue	h0	h3	a0	a3	a7	a11	a19
D263	3.2 ± 0.6	3.3 ± 0.6	3.1 ± 0.6	3.2 ± 0.6	3.1 ± 0.6	3.1 ± 0.6	3.2 ± 0.6
D268	2.7 ± 0.4	2.9 ± 0.3	2.8 ± 0.4	2.9 ± 0.4	2.9 ± 0.3	2.9 ± 0.3	2.9 ± 0.3
D269	4.0 ± 0.1	4.0 ± 0.0	4.0 ± 0.1	4.0 ± 0.1	4.0 ± 0.0	4.0 ± 0.0	4.0 ± 0.0
D308	2.3 ± 0.5	2.2 ± 0.6	3.8 ± 0.7	3.7 ± 0.7	2.9 ± 0.7	3.5 ± 0.6	4.8 ± 1.2
E312	3.5 ± 0.4	3.6 ± 0.4	3.7 ± 0.3	3.6 ± 0.3	3.6 ± 0.3	3.6 ± 0.3	3.6 ± 0.3
E220	4.8 ± 0.3	4.7 ± 0.3	4.9 ± 0.3	4.8 ± 0.3	4.8 ± 0.3	4.8 ± 0.3	4.8 ± 0.3
E236	3.5 ± 0.5	3.6 ± 0.5	3.6 ± 0.6	3.6 ± 0.6	3.6 ± 0.5	3.6 ± 0.5	3.6 ± 0.5
E238	3.4 ± 0.6	3.3 ± 0.6	3.4 ± 0.6	3.3 ± 0.6	3.4 ± 0.6	3.4 ± 0.6	3.4 ± 0.6
E240	4.1 ± 0.5	4.1 ± 0.5	4.1 ± 0.5	4.1 ± 0.5	4.1 ± 0.5	4.1 ± 0.5	4.1 ± 0.5
E261	4.4 ± 0.6	4.0 ± 0.9	4.5 ± 0.4	4.4 ± 0.5	4.5 ± 0.5	4.4 ± 0.5	4.5 ± 0.3
E285	5.1 ± 0.4	5.2 ± 0.5	4.4 ± 0.5	4.4 ± 0.5	6.0 ± 0.9	4.5 ± 0.4	5.2 ± 1.3
E293	4.8 ± 0.4	4.8 ± 0.4	3.8 ± 0.9	3.9 ± 0.9	3.6 ± 0.8	4.1 ± 1.0	3.6 ± 0.9
H229	6.1 ± 0.4	7.1 ± 0.3	6.2 ± 0.4	7.2 ± 0.3	7.2 ± 0.2	7.2 ± 0.3	7.2 ± 0.3
H294	5.9 ± 0.3	6.2 ± 0.4	6.0 ± 0.3	6.1 ± 0.3	6.0 ± 0.3	6.0 ± 0.3	6.0 ± 0.3
	Alternative pK_a $\mu \pm \sigma$						
residue	h0	h3	a0	a3	a7	a11	a19
D308	4.1 ± 0.9	4.0 ± 0.9	3.8 ± 0.7	3.7 ± 0.7	4.5 ± 1.0	4.0 ± 0.9	6.2 ± 0.8
E285	3.9 ± 0.5	3.9 ± 0.6	4.4 ± 0.5	4.4 ± 0.5	4.4 ± 0.9	4.5 ± 0.4	3.8 ± 0.6
E293	4.1 ± 0.6	4.0 ± 0.6	3.8 ± 0.9	3.9 ± 0.9	3.6 ± 0.8	3.6 ± 0.6	3.6 ± 0.9

Table 15: Mean and standard deviation of calculated burying-values using PROPKA 3.1 for each ns of simulation time.

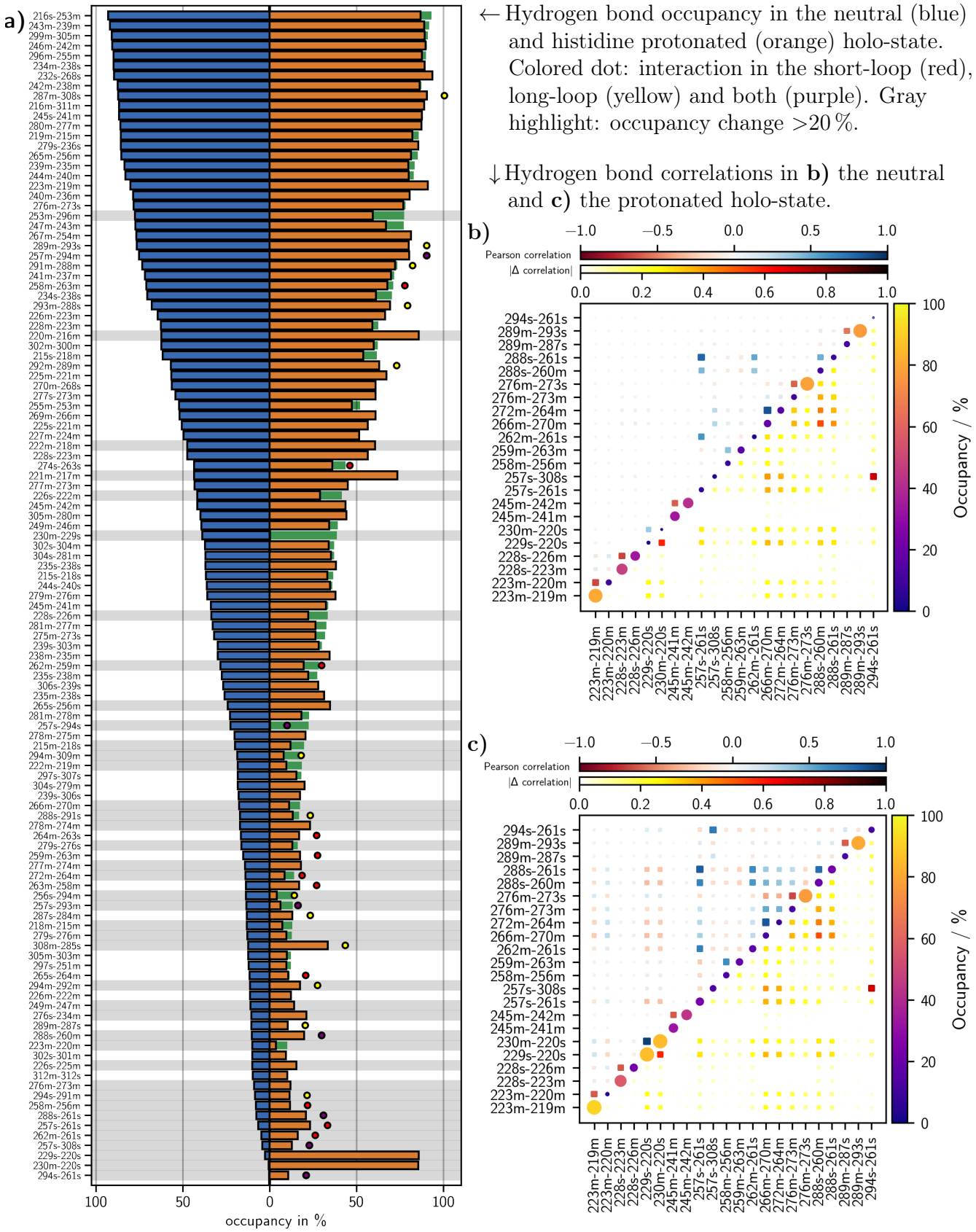
	%buried $\mu \pm \sigma$						
residue	h0	h3	a0	a3	a7	a11	a19
D263	1 ± 2	0 ± 1	1 ± 2	1 ± 3	1 ± 2	1 ± 2	1 ± 2
D268	37 ± 13	45 ± 11	42 ± 10	48 ± 8	49 ± 6	49 ± 6	48 ± 8
D269	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
D308	38 ± 4	38 ± 4	35 ± 6	34 ± 5	34 ± 5	33 ± 5	41 ± 7
E312	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
E220	16 ± 7	41 ± 10	18 ± 7	42 ± 10	44 ± 7	43 ± 9	43 ± 9
E236	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
E238	44 ± 10	46 ± 8	47 ± 8	50 ± 8	49 ± 7	53 ± 10	48 ± 8
E240	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
E261	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
E285	10 ± 7	13 ± 8	4 ± 5	4 ± 6	11 ± 11	3 ± 5	22 ± 12
E293	15 ± 4	14 ± 4	1 ± 3	1 ± 3	1 ± 3	1 ± 3	1 ± 3
H229	30 ± 13	23 ± 5	24 ± 12	24 ± 5	24 ± 4	24 ± 4	23 ± 4
H294	8 ± 9	12 ± 9	4 ± 6	13 ± 9	14 ± 9	16 ± 8	16 ± 8

Hydrogen bond analysis

We investigated the occurrence of hydrogen bonds in the langerin holo-systems (h0, h3) as we suspected that a change in the protonation state may (transforming a potential hydrogen bond acceptor into a donor) also perturb the observed hydrogen bonded interactions. To get a neutral impression of the present interactions we extracted time-resolved information about all possible bonds, disregarding only those bonds involving the termini (residues 198 to 214 and 322 to 325) and applying an occupancy filter (10 % < occupancy < 90 % in at least one of the systems). The hydrogen bond existence information was collected using the GROMACS tool gmx hbond using the default criteria (hydrogen–donor–acceptor angle $\leq 30°$ and donor-acceptor distance ≤ 3.5 Å). Figure 10 visualises the resulting subset of interactions sorted by occupancy in h0. Bonds that undergo a large change in occupancy upon H294/H229 protonation (gray highlighting for changes >20 %) can be considered important under the reservation of a potential sampling bias. When a residue of the short- or long-loop is involved, this is indicated by colored circles above the bars. Absolute occupancy is of secondary importance since neither a particularly high nor low value allows a conclusion regarding a contingent function of the structural element.

As a complementary selection criteria we looked for pairwise correlations in the existence of hydrogen bonds, since it can be a hint towards allosteric communication over a hydrogen bonded network. We calculated the Pearson-correlation coefficient for all found (and filtered) hydrogen bonds and refined the result by only looking at those bonds above a correlation cutoff (bond is involved in a correlation >0.6 in at least one of the systems). In figure 10 the resulting correlation matrix is illustrated as a heatmap. The upper left triangle shows the correlation in the individual system while the lower right triangle shows the absolute difference in the correlation between the two systems. The diagonal elements represent again the occupancy of the bonds. Highly correlated bonds are of special interest, in particular if the degree of correlation changes upon protonation.

Combining the decision criteria, occupancy and correlation, and focusing on those bonds formed by residues of the short- and long-loop and respectively the interconnecting segment, we settled on the following set of important hydrogen bonds as structurally interesting main factors effected by histidine H294 protonation: 253m-296m, 256s-294m, 257s-261s, 257s-293m, 257s-294s, 257s-308s, 258m-256m, 262m-259m, 262m-261s, 265s-256m, 266m-270m, 272m-264m, 274s-263s, 276m-273m, 287m-308s, 288s-260m, 288s-261s, 288s-291s, 292m-289m, 294m-292m, 294m-309m, 294s-261s, 294s-291m, 297s-307s, 308m-285s.



←Hydrogen bond occupancy in the neutral (blue) and histidine protonated (orange) holo-state. Colored dot: interaction in the short-loop (red), long-loop (yellow) and both (purple). Gray highlight: occupancy change >20 %.

↓Hydrogen bond correlations in **b)** the neutral and **c)** the protonated holo-state.

Figure 10: Hydrogen bond analysis.

Principle component analysis

For the principle component analysis (PCA) we used the GROMACS tool gmx covar in a joint fashion on a concatenated trajectory of h0 and h3, including only non-hydrogen protein atoms of the residues 201 to 322 (989 atoms), with an increased time step of 1 ns (57481 frames). All structures have been fitted using the backbone atoms of residues 201 to 256 and 295 to 322 (252 atoms) as a reference. Replicawise projections of the individual systems (31091247 frames for h0, 26389202 frames for h3) onto the resulting eigenvectors and structure interpolations have been obtained with the GROMACS tool gmx anaeig.

Time-lagged independent component analysis

We used the PyEmma (24) package version 2.5.7 for the time-lagged independent component analysis (TICA). The analysis was done for the neutral (h0) and the protonated (h3) holo-system separately on the same set of input features, which are backbone dihedrals of the residues 253 to 312, χ_1 and χ_2 side-chain dihedrals of the residues 285, 293, 294 and 308 and existence-functions for the set of hydrogen bonds we settled on previously (see section *Hydrogen bond analysis*): 253m-296m, 256s-294m, 257s-261s, 257s-293m, 257s-294s, 257s-308s, 258m-256m, 262m-259m, 262m-261s, 265s-256m, 266m-270m, 272m-264m, 274s-263s, 276m-273m, 287m-308s, 288s-260m, 288s-261s, 288s-291s, 292m-289m, 294m-292m, 294m-309m, 294s-261s, 294s-291m, 297s-307s, 308m-285s. Dihedral angles were decomposed into sine and cosine components. Hydrogen bond existences were converted to trajectories of 1 (present) and -1 (not present). We included 128 replica for the unprotonated and 116 replica for the protonated system with a time-step of 5 ps. Lag times between 1 and 30 ns were tested, giving the most satisfying result with respect to eigenvalue decay and state separation in the projections at 20 ns. Projections in 6D are shown in figure 11.

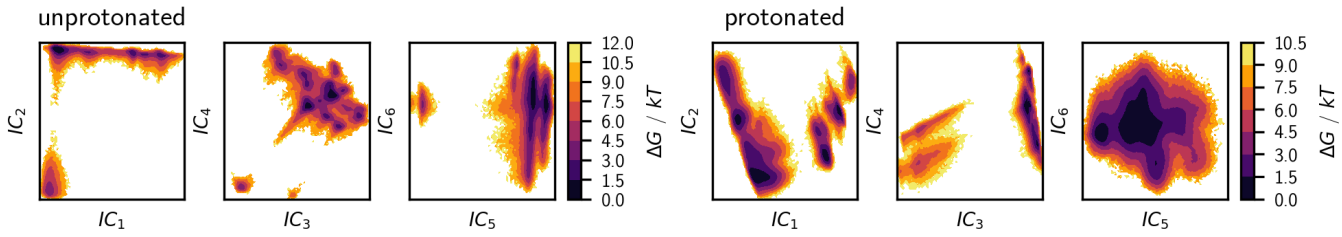


Figure 11: 6D projections of the unprotonated and protonated system onto the first 6 independent components of the TICA.

Figure 12 shows correlations of the input coordinates with components obtained from the TICA to get an impression on what degrees of freedom contribute most to the transformed coordinates.

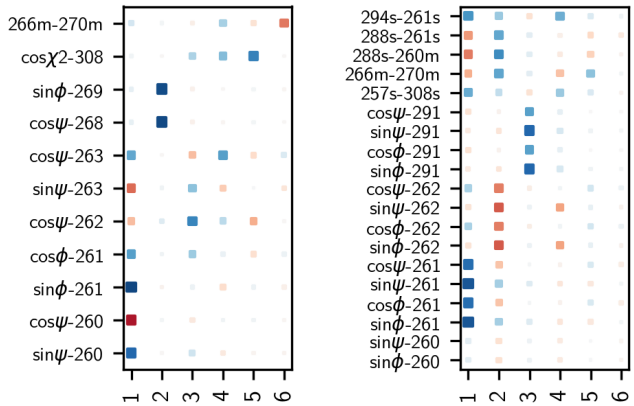


Figure 12: Correlation of input features to the transformed coordinates of the TICA (cutoff >0.5).

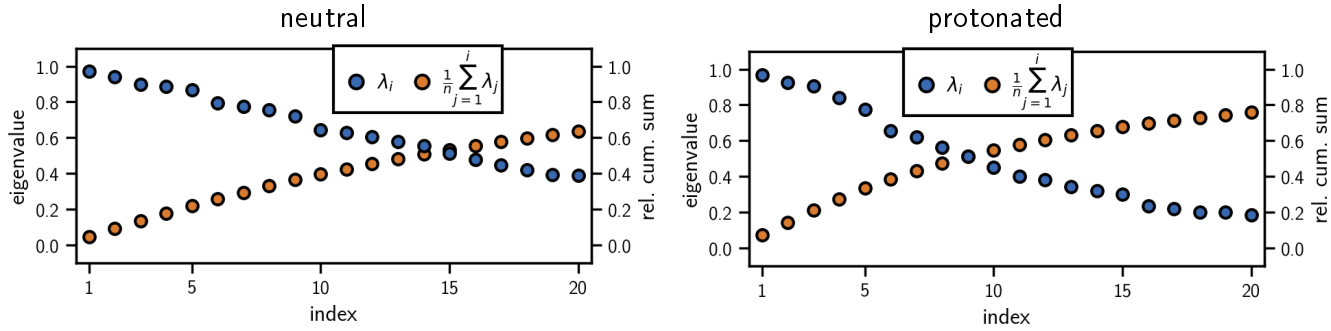


Figure 13: Eigenvalue spectra of the TICA for the neutral and the protonated holo-system.

Clustering and core-set Markov-state model construction

For the clustering of data points in the PCA- and TICA-space we used the density-based common-nearest neighbors (CNN) cluster-algorithm in an implementation that is currently under development. The source is available on GitHub: `git@github.com:janjoswig/CNN.git`. The package includes also functionalities for the estimation of core-set Markov-state models on top of a discretization obtained from such a clustering.

In figure 14 we show the final cluster results for the TICA-projections of the neutral and the protonated system. In the unprotonated case we isolated 25 clusters in a 5-step hierarchical procedure. For the protonated system we found 22 clusters in 6 steps. Clustering was done on a reduced data set with a time step of 1 ns. Cluster assignments were subsequently predicted for a data set with a time step of 100 ps.

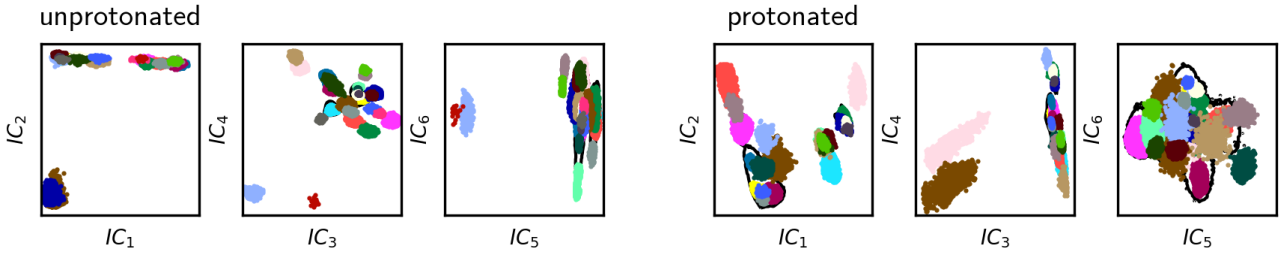


Figure 14: 6D projections of the unprotonated and protonated system onto the first 6 independent components of the TICA clustered into 25 and 22 core-sets, respectively.

We estimated MSMs for lag-times between 1 and 16 ns. Figure 15 shows the implied time-scale teste for these models.

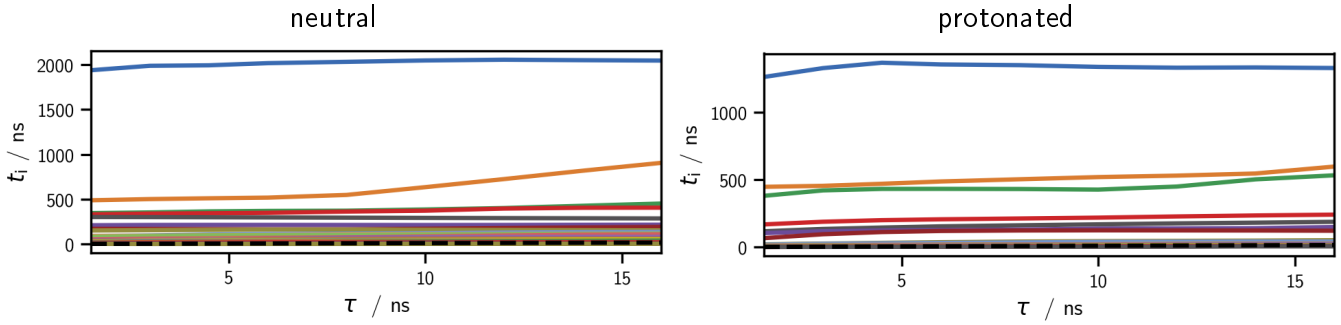


Figure 15: Implied time scales for MSMs of the neutral and the protonated system.

The core-sets were lumped by Perron-cluster cluster analysis (PCCA) on the basis of a MSM with 7 ns lag-time, giving the meta-stable sets of clusters connected by the first 5 transition processes shown in figure 16. Also shown are representative structures for each set of long-lived conformations in figure 17.

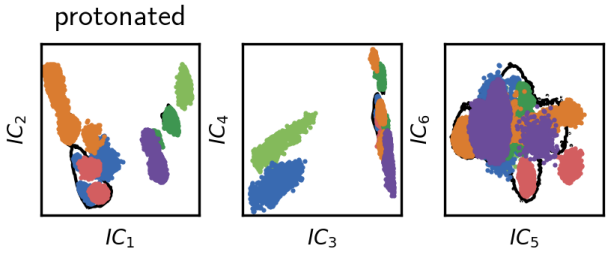


Figure 16: Core-sets from figure 14 grouped to meta-stable sets by PCCA incorporating the 5 slowest processes.

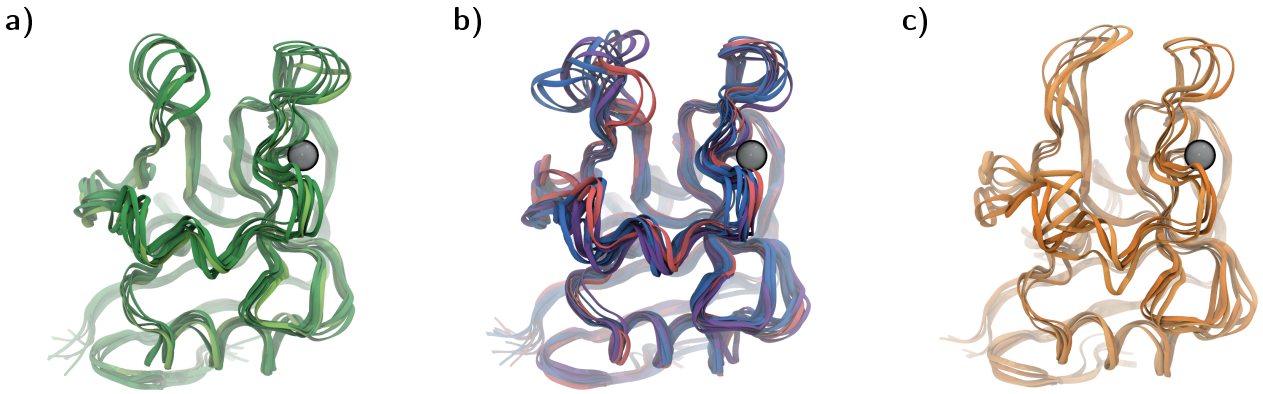


Figure 17: Example structures for meta-stable conformations from figure 16 in the H294-protonated holo-system. **a)** K257s–D308s bonded conformations referred to as the "green" cluster. **b)** Open loop forms with distinct short-loop conformations. **c)** Closed loop-forms referred to as the "orange" cluster. Within this set, α_3-helix conformations exchange on a time scale of 130 ns (lighter and deeper orange; not shown in the 2D projection).

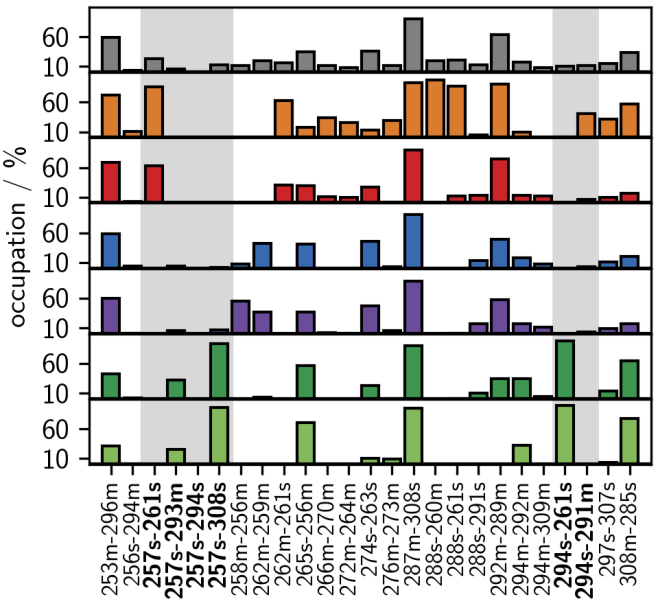


Figure 18: Hydrogen bond occupancy in meta-stable sets of the H294-protonated holo-system from figure 16 (ensemble in gray).

K257–D308 distance

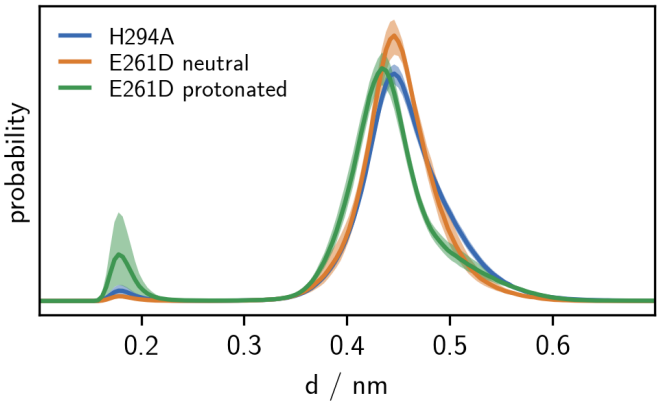


Figure 19: K257–D308 distance distribution for the H294A mutant and E261D with protonated and neutral H294. 95 % confidence interval on the mean over replica-wise probability densities obtained by bootstrapping (1000 samples).

Coulomb interactions

Fig. 20 shows the Coulomb force between two positive elementary point-charges as a function of the distance for several typical relative dielectric constants. In our MD simulations with explicit water, the pairwise Coulomb interactions are not truncated and are calculated at a relative dielectric constant of one, i.e following the blue curve in Fig. 20. But the atoms in the space between Ca^{2+} and H294 can rearrange according to the surrounding electrostatic field thereby causing an overall shielding effect, such that the effective force between two residues on the protein surface, e.g. H294 and K257, will be similar to the green curve. Thus, at distances of $d > 1.0$ nm, the effective Coulomb forces between residues on the protein surface are close to zero.

Fig. 21 shows histograms of the pairwise Coulomb-energies in vacuum between H294, K257 and Ca^{2+} of the neutral and H294 protonated holo-state. As expected, the Coulomb energy between Ca^{2+} and the overall neutral H294 is approximately zero, and increases to about 30 kJ mol^{-1} when H294 is protonated. Note that in the presence of water, the corresponding repulsive force is dampened by a shift of the charge distribution in the space between Ca^{2+} and $H294^{+}$. The most striking result of this analysis is the change of interaction energies between Ca^{2+} and K257 upon protonation of H294. In the protonated state the interaction energies reach values beyond 100 kJ mol^{-1}. We confirmed that these high energy values correspond to conformations in which the K257s–D308s hydrogen bond is formed, i.e. to those conformations that, according to our model, are responsible for the regulation of the Ca^{2+}-affinity. When the K257s–D308s hydrogen bond is formed, the repulsion between the positively charged K257 and Ca^{2+} cannot be mitigated by the rearrangement of atoms in the intervening space. Thus, in these conformations the Coulomb repulsion between K257 and Ca^{2+} is strong enough to explain the observed decrease in Ca^{2+} affinity. Finally, the Coulomb energy between K257 and H294 shifts from negative energy values to positive values upon protonation of H294, in accordance with the observed opening of the hydrogen bond (dotted lines in Fig. 21).

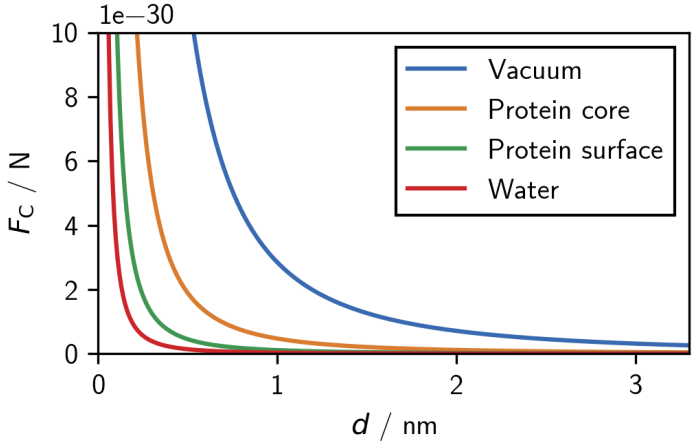


Figure 20: Expected distance dependent Coulomb-force between two positive elementary charges for assumed relative dielectric constants in vacuum ($\epsilon_r = 1$), within a protein ($\epsilon_r = 6$ (25)), on a protein surface ($\epsilon_r = 25$ (25)) and in water ($\epsilon_r = 80$). For media with higher dielectric constants, the interaction is very small beyond distances of 1 nm.

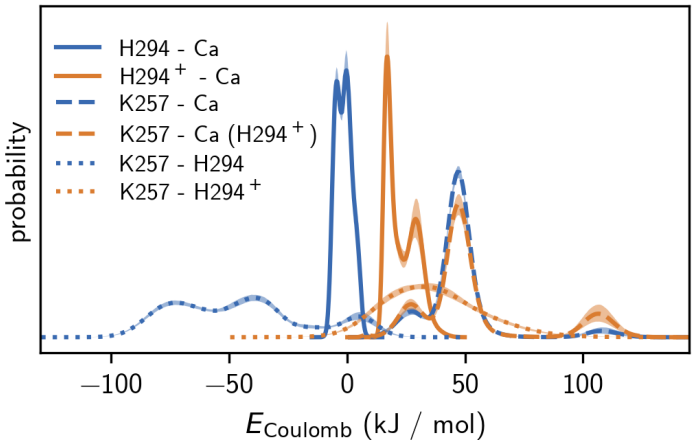


Figure 21: Coulomb contributions during MD runs between H294, K257 and calcium. The interaction energies were obtained from a GROMACS rerun with energy groups on the full trajectory data of neutral (blue) and H294 protonated (orange) holo langerin. 95 % confidence interval on the mean over replica-wise probability densities obtained by bootstrapping (1000 samples).

Steered molecular dynamics simulations

All systems simulated in a steered MD setup were prepared following the same general procedure that is very similar to the one used for the conventional MD simulations, but more concise. For a selected starting structure of holo-langerin (as denoted in table 16) without crystal waters, ligands and excessive residues and ions, a topology in the desired protonation state was automatically generated using GROMACS 2019 (3–9) and the AMBER99SB-ILDN force field (10). The molecule was put into a cubic box at a distance of at least 1.5 nm to the box borders and minimized in vacuum (steepest decent, eps $< 100\,\mathrm{kJ\,mol^{-1}\,nm^{-1}}$) before it was solvated in explicit TIP3P water (11) and the replacement of water by chloride to neutralize contingent charges. It was minimized again in solution twice (1. steepest decent, eps $< 100\,\mathrm{kJ\,mol^{-1}\,nm^{-1}}$, 2. conjugate gradient, eps $< 1\,\mathrm{kJ\,mol^{-1}\,nm^{-1}}$) and then equilibrated without any position restraints in the *NVT*- (300 K, V-rescale thermostat (12), coupling rate at 0.1 ps, coupling groups protein and non-protein, 200 ps length) and *NPT*-ensemble (1. 1 bar, Berendsen barostat (26), isotropic, coupling rate at 1 ps, 200 ps length, 2. 1 bar, Parrinello-Rahman barostat (13), isotropic, coupling rate at 2 ps, 400 ps length). For both, equilibration and production, the LINCS (14) (order 6, 2 iterations in equilibration and order 4, 1 iteration in production) algorithm was applied to constrain bonds to hydrogen. The leap-frog integrator (15) was used at a time step of 1 fs during equilibration and 2 fs in production. For Lennard-Jones (cut-off) and electrostatic (PME (16), order 6) interactions the cut-off was set to 1 nm, while the Verlet cut-off scheme was used to create the neighbour lists. Periodic boundary conditions were imposed in all three dimensions. The vacuum minimisation differs substantially from these settings as the group cut-off scheme was utilised, for Lennard-Jones (cut-off) and electrostatic (cut-off) interactions the cut-off was set to infinity (0), and no periodic boundary conditions were applied. In production a force was exerted in form of a harmonic (umbrella) potential along a single coordinate, defined as the distance between the Ca^{2+}-ion in the binding pocket and the center of mass of the "upper" protein region represented by the C_α-atoms of residues 257, 264, 281, 282, 293 and 294 (see figure 22). The harmonic spring constant was set to $k = 500\,\mathrm{kJ\,mol^{-1}\,nm^{-2}}$ and the pulling-force acting on the pull-coordinate was constantly increased at a velocity of $1 \times 10^{-4}\,\mathrm{nm\,ps^{-1}}$. Protein/calcium coordinates were written to a compressed trajectory file at a time step of 1 ps as well as the instantaneous pull-force. The individual simulations were prolonged to a maximal length of 20 ns or until the pulling-distance exceeded the length of the boxvector. For each system (starting structure) the pull experiment was repeated 40 times. As rupture force, the force needed to remove the Ca^{2+}-ion from the binding pocket, we took the maximum pull-force from the force trajectories smoothed by the running mean within a window of 10 ps.

Table 16: Overview steered MD.

System	Starting structure
h0	crystal structure PDB-ID 3p5g (1)
h0*uf*	unfolded conformation selected from a0, calcium inserted manually
h0m1	H294A mutant created with the VMD (21) molefacture plugin, crystal structure analogue
h0m3	K257A mutant created with the VMD (21) molefacture plugin, crystal structure analogue
h0m1m3	K257A/H294A double mutant created with the VMD (21) molefacture plugin, crystal structure analogue
h3	crystal structure analogue
h3*	H294–E261/K257–D308 hydrogen bonded conformation selected from h3 ("green" PCA cluster)
h3m2*	E261D mutant created with the VMD (21) molefacture plugin, H294–D261/K257–D308 hydrogen bonded conformation
h3m3	K257A mutant created with the VMD (21) molefacture plugin, crystal structure analogue
h7	crystal structure analogue
h11	crystal structure analogue
h19	crystal structure analogue

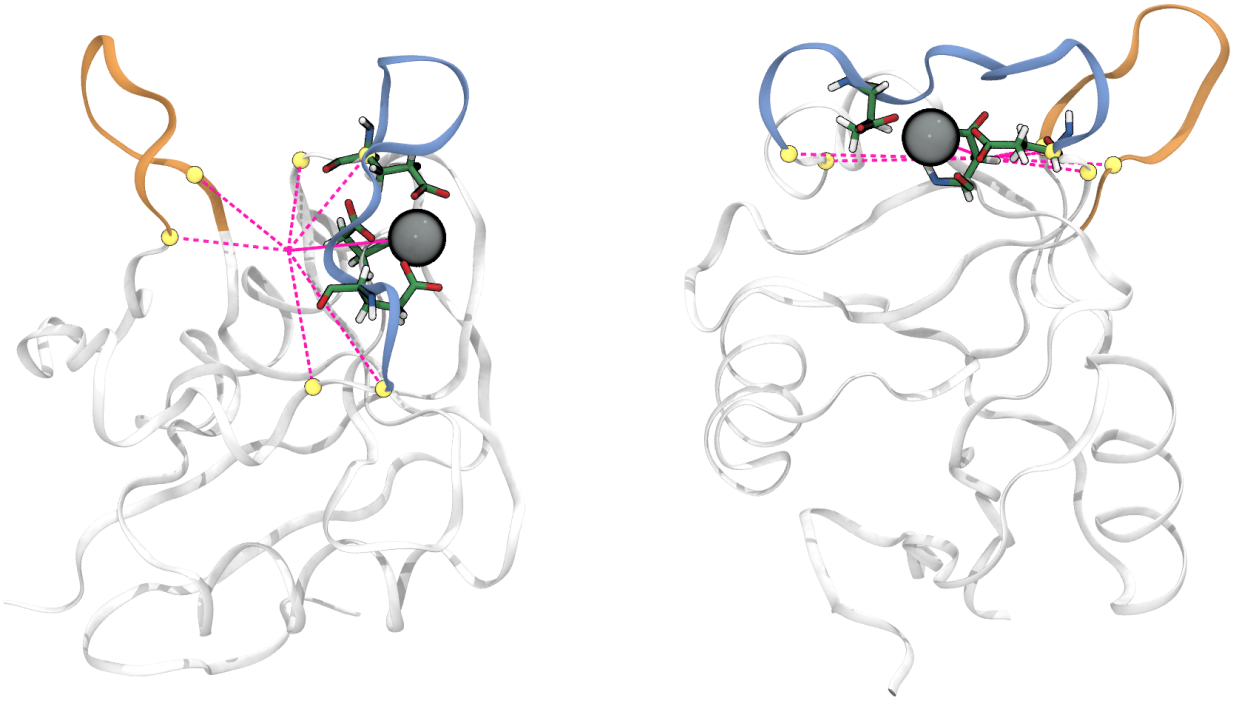


Figure 22: The pull coordinate (solid magenta line) is defined as the distance between the center of mass of the C_α-atoms of residues 257, 264, 281, 282, 293 and 294 (yellow spheres, dashed magenta lines), representing the rigid "upper" protein, and the Ca^{2+}-ion (gray Van der Waals sphere).

Rate determination of apo-langerins long-loop unfolding

Unfolding of apo-langerins long-loop was determined by following five different criteria applied to trajectories with a time step of 1 ns: a) We determined – by visual inspection in VMD (21) – the last frame in which the long-loop appears to be clearly folded. b) If applicable, we also determined the first frame from which on the loop is clearly unfolded irreversibly on the time scale of the simulation. The mean between "last folded" and "first unfolded" point was taken as the unfolding point. c) We tracked the long-loop (residues P283 to E293) C_α root-mean-square deviation (RMSD) with the crystal structure as reference (PDB-ID 3p5g (2)) and detected the first frame in which a value higher than 0.2 nm was reached. d) We tracked the existence of the N287m–D308s hydrogen bond and determined the first frame from which on the bond is not constantly occupied anymore. e) We did the same for N288m–D308s. The hydrogen bond existence functions were obtained as described in section *Hydrogen bond analysis* on page 13. RMSDs were calculated using the GROMACS tool rms. The RMSD time series, smoothed by a running average window of 5 ns, were evaluated by a change-point detection algorithm (mean shift clustering using a bandwidth of 0.035).

Table 17: Trajectories showing an event of unfolding or a clearly folded structure throughout the simulation time of 220 ns by a) visual inspection (last folded frame), b) visual inspection (mean of last folded and first unfolded frame), c) RMSD cut-off (>0.2 nm), d) breaking of N287m–D308s H-bond, e) breaking of N288m–D308s H-bond.

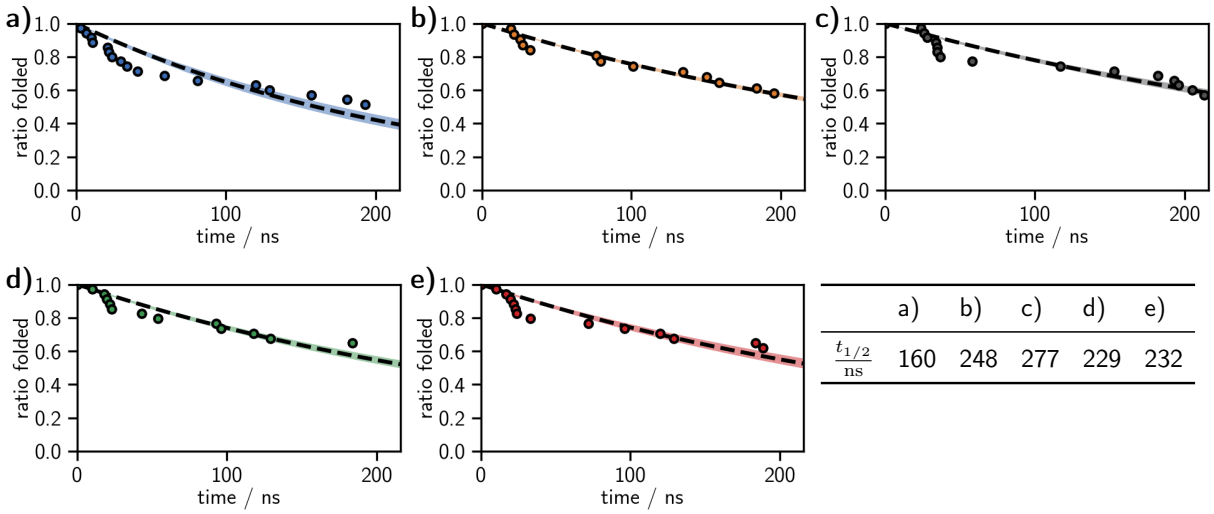
a0	a)	b)	c)	d)	e)
unfold	17	13	15	12	13
stay folded	18	18	20	22	21
sum	35	31	35	34	34

a3					
unfold	29	17	29	23	25
stay folded	25	25	25	30	28
sum	54	42	54	53	53

a6+a7					
unfold	16	5	6	0	4
stay folded	18	18	24	26	22
sum	34	23	30	26	26

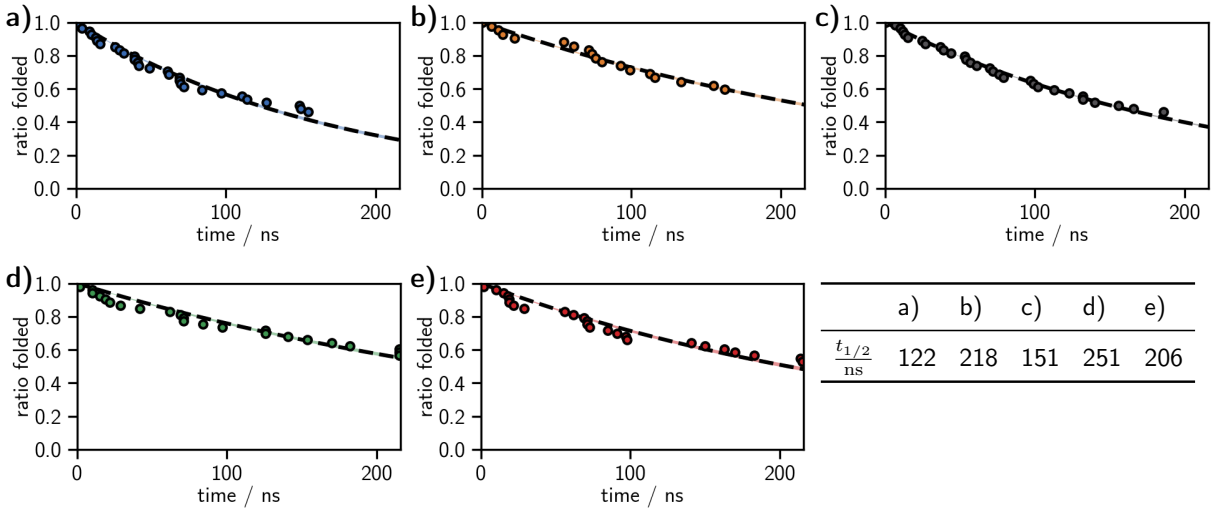
a10+a11	a)	b)	c)	d)	e)
unfold	19	12	13	7	7
stay folded	24	24	26	27	27
sum	43	36	39	34	34

a18+a19					
unfold	24	20	19	17	20
stay folded	8	8	9	8	5
sum	32	28	28	25	25



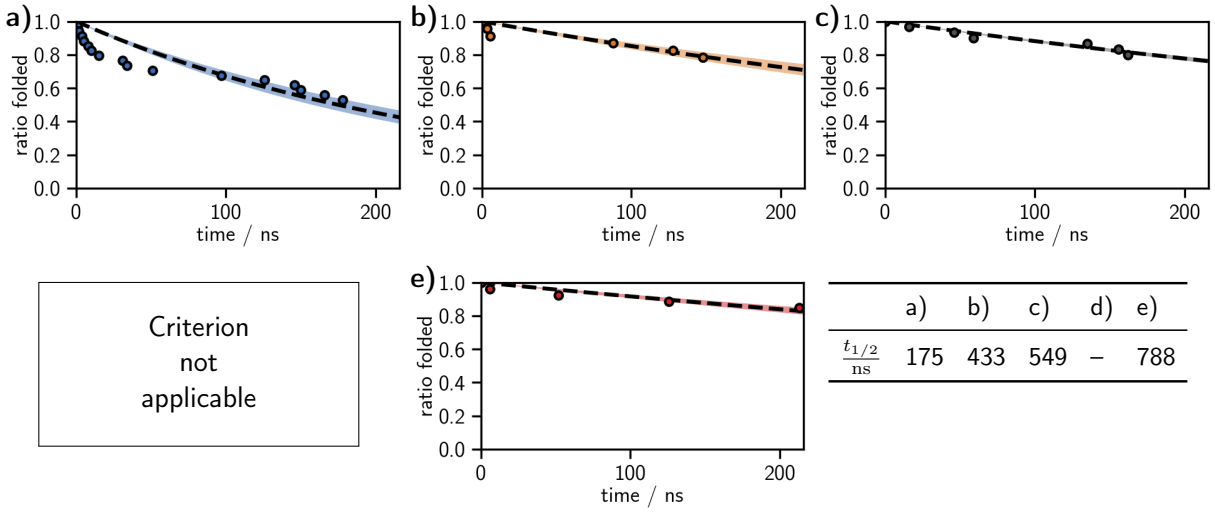
	a)	b)	c)	d)	e)
$\frac{t_{1/2}}{\text{ns}}$	160	248	277	229	232

Figure 23: Unfolding events in neutral apo-langerin (a0) and exponential fits measured by **a)** visual inspection (last folded frame), **b)** visual inspection (mean of last folded and first unfolded frame), **c)** RMSD cut-off (>0.2 nm, **d)** breaking of N287m–D308s H-bond, **e)** breaking of N288m–D308s H-bond.



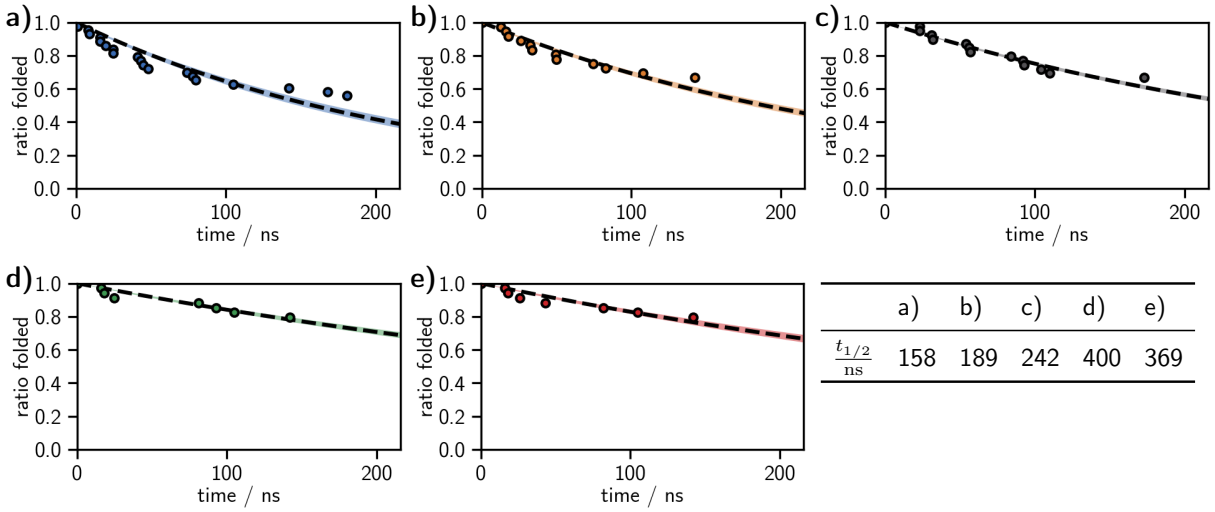
	a)	b)	c)	d)	e)
$\frac{t_{1/2}}{\text{ns}}$	122	218	151	251	206

Figure 24: Unfolding events in H294-protonated apo-langerin (a3) and exponential fits measured by **a)** visual inspection (last folded frame), **b)** visual inspection (mean of last folded and first unfolded frame), **c)** RMSD cut-off (>0.2 nm, **d)** breaking of N287m–D308s H-bond, **e)** breaking of N288m–D308s H-bond.



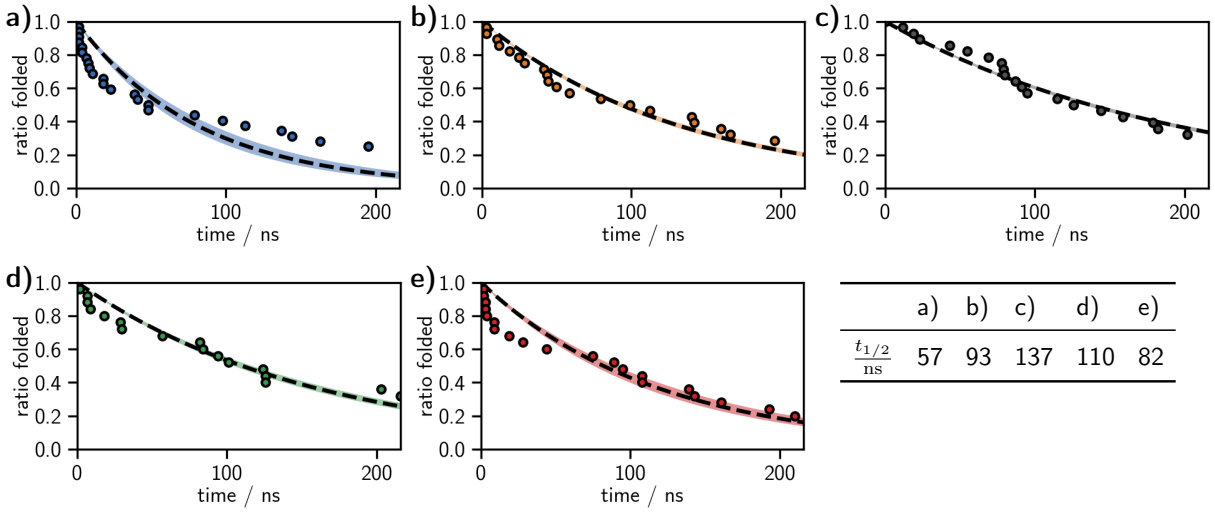
	a)	b)	c)	d)	e)
$\frac{t_{1/2}}{\text{ns}}$	175	433	549	–	788

Figure 25: Unfolding events in E285-protonated apo-langerin (a6/a7) and exponential fits measured by **a)** visual inspection (last folded frame), **b)** visual inspection (mean of last folded and first unfolded frame), **c)** RMSD cut-off (>0.2 nm, **d)** breaking of N287m–D308s H-bond (never observed), **e)** breaking of N288m–D308s H-bond.



	a)	b)	c)	d)	e)
$\frac{t_{1/2}}{\text{ns}}$	158	189	242	400	369

Figure 26: Unfolding events in E293-protonated apo-langerin (a10/a11) and exponential fits measured by **a)** visual inspection (last folded frame), **b)** visual inspection (mean of last folded and first unfolded frame), **c)** RMSD cut-off (>0.2 nm, **d)** breaking of N287m–D308s H-bond, **e)** breaking of N288m–D308s H-bond.



	a)	b)	c)	d)	e)
$\frac{t_{1/2}}{\text{ns}}$	57	93	137	110	82

Figure 27: Unfolding events in D308-protonated apo-langerin (a18/a19) and exponential fits measured by **a)** visual inspection (last folded frame), **b)** visual inspection (mean of last folded and first unfolded frame), **c)** RMSD cut-off (>0.2 nm, **d)** breaking of N287m–D308s H-bond, **e)** breaking of N288m–D308s H-bond.

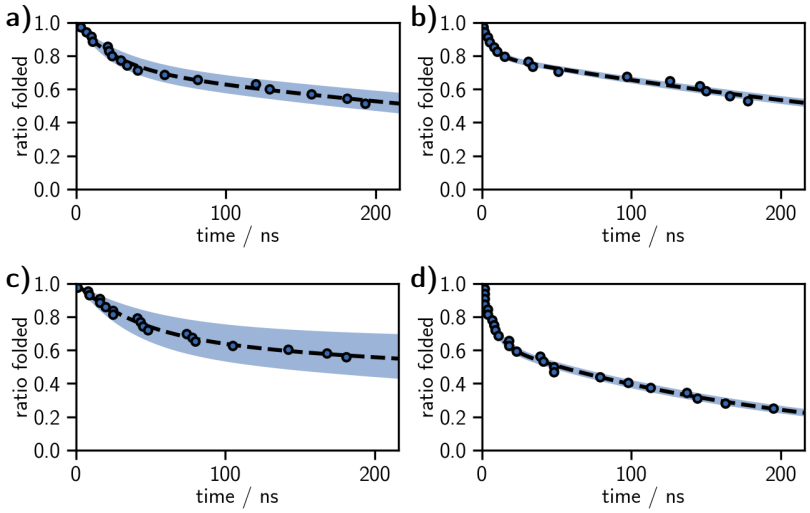


Figure 28: Unfolding events by visual inspection (last folded frame) and double exponential fits in **a)** neutral apo- (a0), **b)** E285 protonated (a6/a7), **c)** E293 protonated (a10/a11), and **d)** D308 protonated (a18/a19) langerin. The single exponential fit for the H294 protonated case is already sufficiently good (see fig. 24).

ITC measurements

Data analysis, plotting and curve fitting was performed with OriginPro 2019 (OriginLab, Northampton, MA).

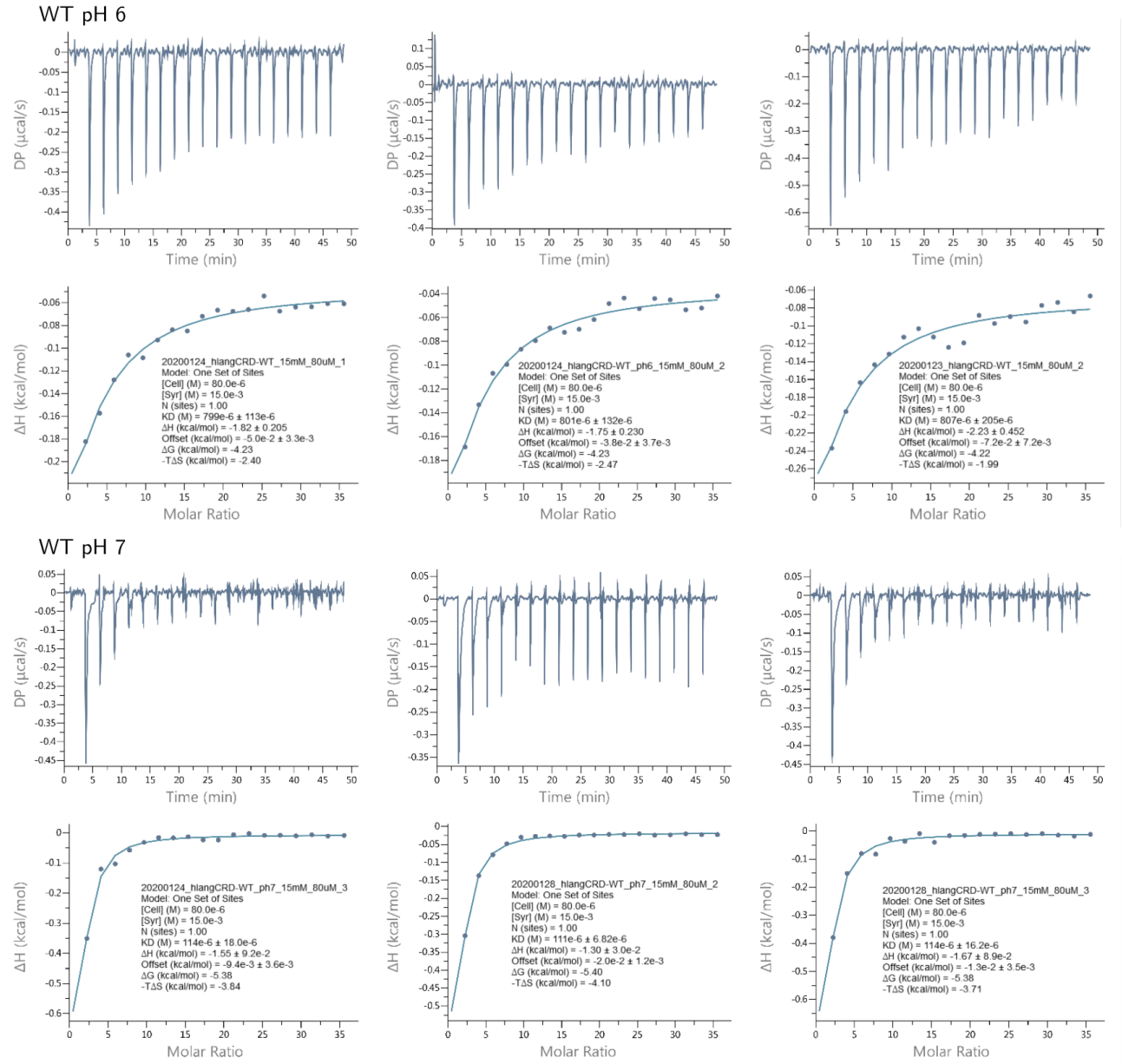


Figure 29: ITC measurements for wild type langerin.

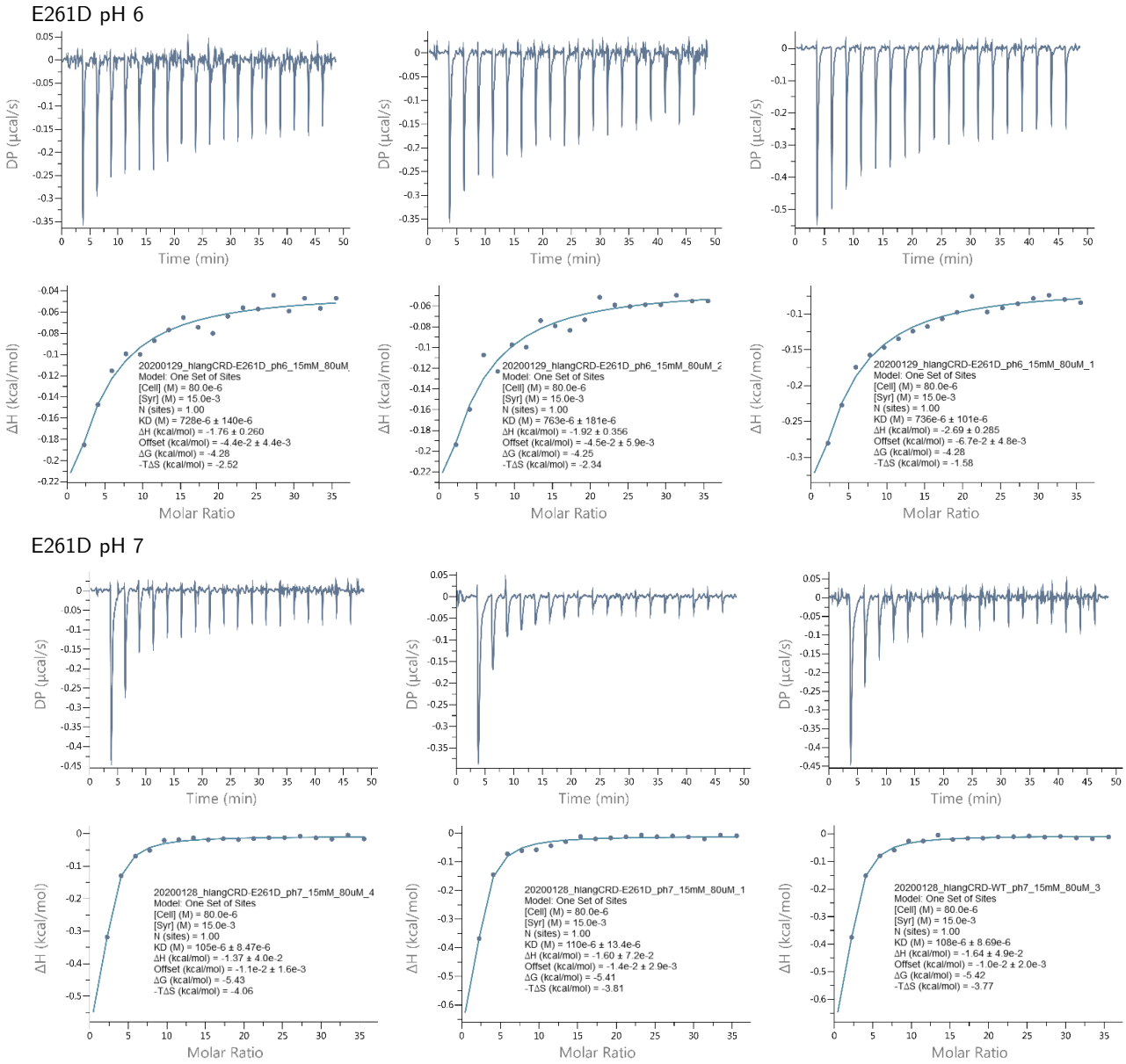


Figure 30: ITC measurements for the E261D langerin mutant.

Comparison of langerin to other C-type lectins

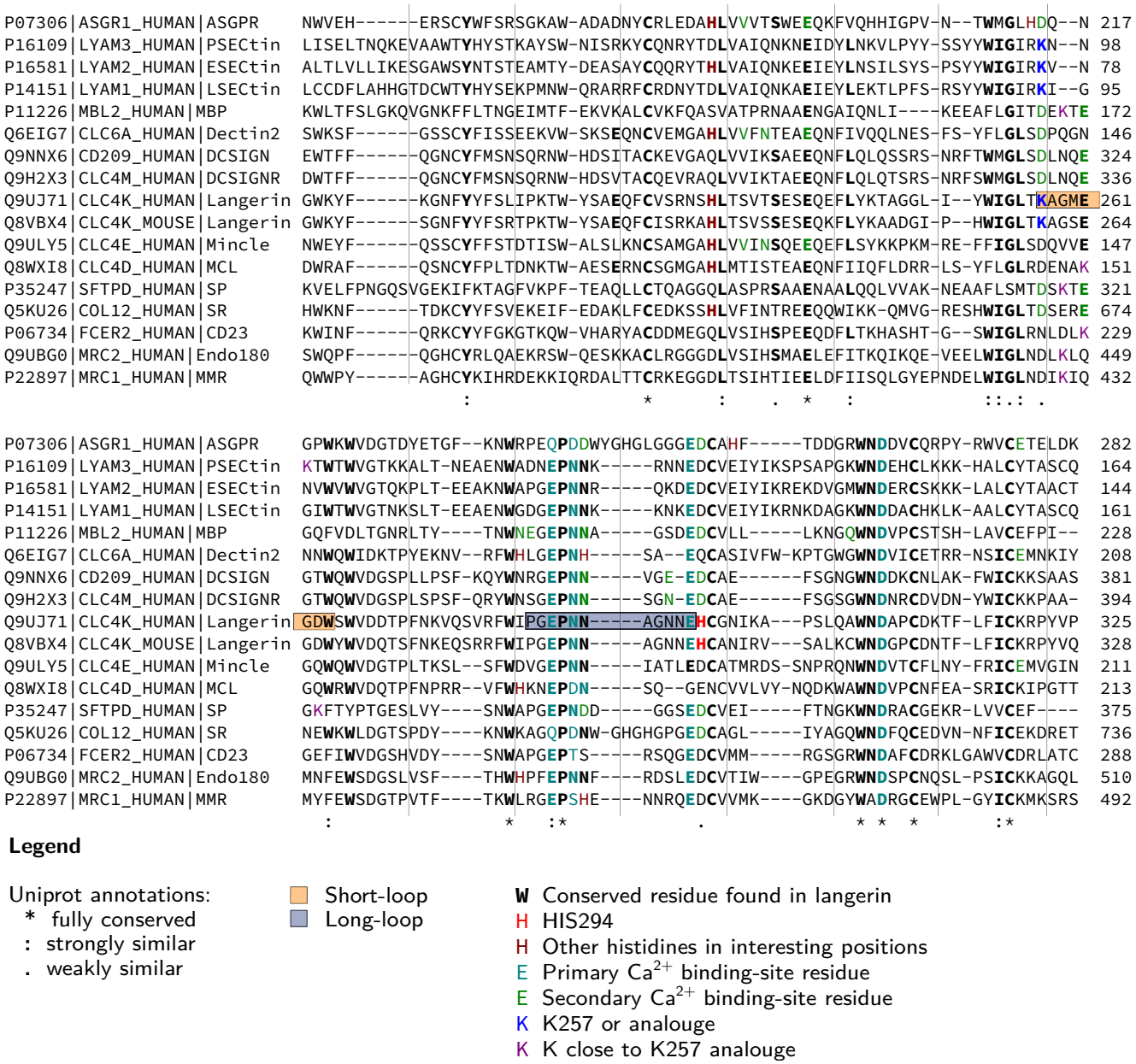


Figure 31: Sequence alignment using UniProtKB (27) for langerin's carbohydrate recognition domain (residues 198 to 325) with selected C-type lectins. Annotations have been added by comparing available crystal structures of ASGPR (PDB 5JPV), PSECtin (PDB 1G1S), ESECtin (PDB 1G1T), LSECtin (PDB 5VC1), MBP (PDB 1HUP), dectin-2 (PDB 5VYB), DC-SIGN (PDB 1SL4), DC-SIGNR (PDB 1XPH), Mincle (PDB 3WH2), MCL (PDB 3WHD), SP (PDB 1PWD), SR (PDB 2OX8), CD23a (PDB 4G9A), Endo180 (PDB 5AO5), and MMR (PDB 5XTS).

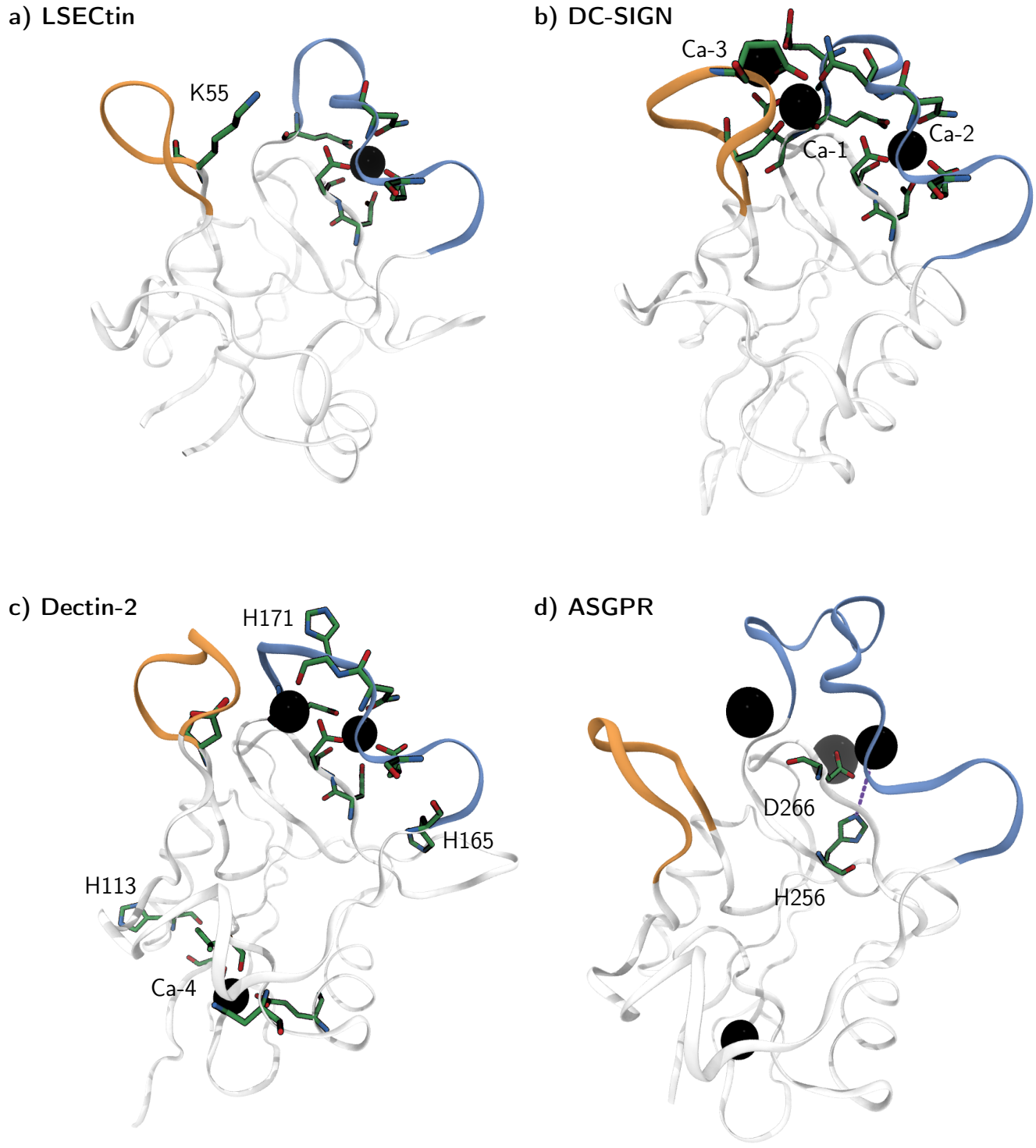


Figure 32: **a)** LSECtin (PDB 5VC1) has a lysine residue (K55) at the same position as langerin (K257). **b)** The carbohydrate recognition domains of C-type lectins may contain multiple secondary Ca^{2+} binding-sites (Ca-1, Ca-3) like it is the case for DC-SIGN (PDB 1SL4). **c)** Potentially pH-sensitive histidine residues are found for example in dectin-2 (PDB 5VYB) in two positions close to the primary Ca^{2+} binding site. A moderately conserved histidine H113 (H229 in langerin) can be found close to a secondary bindig-site (Ca-4). **d)** ASGPR (PDB 5JPV) with pH sensing residue H256. The distance between the histidine side-chain and the Ca^{2+}-ion in the binding pocket is about 0.6 nm in this structure and potentially lower in the protonated state due to a likely interaction with D266. This close proximity suggests a pH-switching mechanism for the Ca^{2+}-affinity through direct interaction.

References

1. Feinberg, H., Powlesland, A. S., Taylor, M. E., and Weis, W. I. (2010). Trimeric structure of Langerin. *Journal of Biological Chemistry* **285**, 13285–13293.
2. Feinberg, H., Taylor, M. E., Razi, N., McBride, R., Knirel, Y. A., Graham, S. A., Drickamer, K., and Weis, W. I. (2011). Structural Basis for Langerin Recognition of Diverse Pathogen and Mammalian Glycans through a Single Binding Site. *Journal of Molecular Biology* **405**, 1027–1039.
3. Berendsen, H. J. C., van der Spoel, D., and van Drunen, R. (1995). GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications* **91**, 43–56.
4. Lindahl, E., Hess, B., and van der Spoel, D. (2001). GROMACS 3.0: A package for molecular simulation and trajectory analysis. *Journal of Molecular Modeling* **7**, 306–317.
5. van der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A. E., and Berendsen, H. J. (2005). GROMACS: Fast, flexible, and free. *Journal of Computational Chemistry* **26**, 1701–1718.
6. Hess, B., Kutzner, C., van der Spoel, D., and Lindahl, E. (2008). GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation* **4**, 435–447.
7. Pronk, S., Páll, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., Shirts, M. R., Smith, J. C., Kasson, P. M., van der Spoel, D., Hess, B., and Lindahl, E. (2013). GROMACS 4.5: A high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* **29**, 845–854.
8. Páll, S., Abraham, M. J., Kutzner, C., Hess, B., and Lindahl, E., *Tackling exascale software challenges in molecular dynamics simulations with GROMACS*; Markidis, S., and Laure, E., Eds.; Springer: 2015; Vol. 8759.
9. Abraham, M. J., Murtola, T., Schulz, R., Páll, S., Smith, J. C., Hess, B., and Lindahl, E. (2015). GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* **1-2**, 19–25.
10. Lindorff-Larsen, K., Piana, S., Palmo, K., Maragakis, P., Klepeis, J. L., Dror, R. O., and Shaw, D. E. (2010). Improved side-chain torsion potentials for the Amber ff99SB protein force field. *Proteins: Structure, Function, and Bioinformatics* **78**, 1950–1958.
11. Jorgensen, W. L., Chandrasekhar, J., Madura, J. D., Impey, R. W., and Klein, M. L. (1983). Comparison of simple potential functions for simulating liquid water. *The Journal of Chemical Physics* **79**, 926–935.
12. Bussi, G., Donadio, D., and Parrinello, M. (2007). Canonical sampling through velocity rescaling. *The Journal of Chemical Physics* **126**, 014101.
13. Parrinello, M., and Rahman, A. (1981). Polymorphic transitions in single crystals: A new molecular dynamics method. *Journal of Applied Physics* **52**, 7182–7190.
14. Hess, B. (2008). P-LINCS: A Parallel Linear Constraint Solver for Molecular Simulation. *Journal of Chemical Theory and Computation* **4**, 116–122.
15. van Gunsteren, W. F., and Berendsen, H. J. C. (1988). A Leap-frog Algorithm for Stochastic Dynamics. *Molecular Simulation* **1**, 173–185.
16. Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., and Pedersen, L. G. (1995). A smooth particle mesh Ewald method. *The Journal of Chemical Physics* **103**, 8577–8593.
17. Blomberg, F., Maurer, W., and Rüterjans, H. (1977). Nuclear magnetic resonance investigation of 15N-labeled histidine in aqueous solution. *Journal of the American Chemical Society* **99**, 8149–8159.
18. Hass, M. A. S., Hansen, D. F., Christensen, H. E. M., Led, J. J., and Kay, L. E. (2008). Characterization of conformational exchange of a histidine side chain: Protonation, rotamerization, and tautomerization of His61 in plastocyanin from Anabaena variabilis. *Journal of the American Chemical Society* **130**, 8460–8470.
19. Hansen, A. L., and Kay, L. E. (2014). Measurement of histidine pK_a values and tautomer populations in invisible protein states. *Proceedings of the National Academy of Sciences* **111**, E1705–E1712.

20. Hanske, J., Aleksić, S., Ballaschk, M., Jurk, M., Shanina, E., Beerbaum, M., Schmieder, P., Keller, B. G., and Rademacher, C. (2016). Intradomain Allosteric Network Modulates Calcium Affinity of the C-Type Lectin Receptor Langerin. *Journal of the American Chemical Society* **138**, 12176–12186.

21. Humphrey, W., Dalke, A., and Schulten, K. (1996). VMD: Visual molecular dynamics. *Journal of Molecular Graphics* **14**, 33–38.

22. Søndergaard, C. R., Olsson, M. H. M., Rostkowski, M., and Jensen, J. H. (2011). Improved Treatment of Ligands and Coupling Effects in Empirical Calculation and Rationalization of pK_a Values. *Journal of Chemical Theory and Computation* **7**, 2284–2295.

23. Olsson, M. H. M., Søndergaard, C. R., Rostkowski, M., and Jensen, J. H. (2011). PROPKA3: Consistent Treatment of Internal and Surface Residues in Empirical pK_a Predictions. *Journal of Chemical Theory and Computation* **7**, 525–537.

24. Scherer, M. K., Trendelkamp-Schroer, B., Paul, F., Pérez-Hernández, G., Hoffmann, M., Plattner, N., Wehmeyer, C., Prinz, J.-H., and Noé, F. (2015). PyEMMA 2: A Software Package for Estimation, Validation, and Analysis of Markov Models. *Journal of Chemical Theory and Computation* **11**, 5525–5542.

25. Li, L., Li, C., Zhang, Z., and Alexov, E. (2013). On the Dielectric "Constant" of Proteins: Smooth Dielectric Function for Macromolecular Modeling and Its Implementation in DelPhi. *Journal of Chemical Theory and Computation* **9**, 2126–2136.

26. Berendsen, H. J. C., M., P. J. P., van Gunsteren, W. F., DiNola, A., and Haak, J. R. (1984). Molecular dynamics with coupling to an external bath. *Journal of Chemical Physics* **81**, 3684–3690.

27. Dogan, T., MacDougall, A., Saidi, R., Poggioli, D., Bateman, A., O'Donovan, C., and Martin, M. J. (2016). UniProt-DAAC: domain architecture alignment and classification, a new method for automatic functional annotation in UniProtKB. *Bioinformatics* **32**, 2264–2271.

CommonNNClustering—A Python package for generic common-nearest-neighbour clustering

Jan-Oliver Joswig and Bettina G. Keller*

Department of Theoretical Chemistry, Freie Universität Berlin, Arnimallee 22, 14195 Berlin, Germany

E-mail: bettina.keller@fu-berlin.de

Abstract

Density-based cluster algorithms are widely used in a variety of data science applications. Their advantage lies in the capability to find arbitrarily shaped and sized clusters and the robustness against outliers. In particular they proved effective in the analysis of Molecular Dynamics (MD) simulations, where they serve to identify relevant, low energetic molecular conformations. As such they can provide a convenient basis for the construction of kinetic (coreset) Markov-state models (MSMs). Here we present the open source Python project CommonNNClustering, which provides an easy-to-use and efficient re-implementation of the common-nearest-neighbour (CommonNN) cluster algorithm. Included are tools for its integration into the workflow of MD analysis and MSM estimation covering rationally guided hierarchical clustering and parameter selection. We put our emphasis on a generic API design to keep the implementation flexible and open for customisation.

Introduction

Density-based clustering procedures—like CommonNN clustering—identify clusters in general as data regions of high sample density separated by sparse, low density regions[1] and have interesting properties for a wide range of applications. In particular, they are useful in the classification of molecular structures because clusters identified by density-based clustering methods tend to have a natural correspondence to what is understood as a molecular *conformation*: an ensemble of structures with relatively high observation probability associated with the same potential energy minimum or separated by sufficiently small energetic barriers (see figure 1 for an illustrative example).

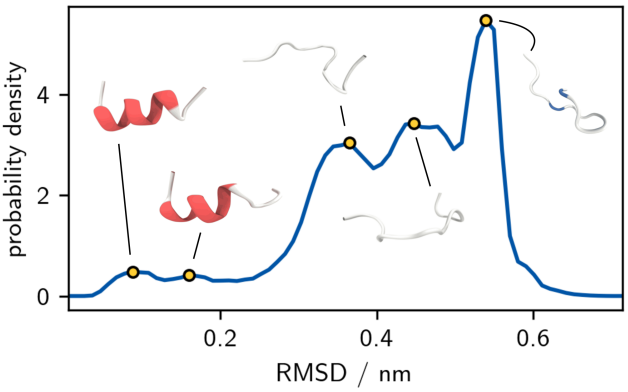


Figure 1: Molecular conformations with low potential energy (high observation probability) identified in a MD simulation of a small helical peptide (PDB ID 6A5J), shown here projected onto the root-mean-square deviation (RMSD) of backbone atom coordinates with respect to the starting structure.

Three points make density-based clustering methods exceptionally suitable in this situation: 1) a conformational cluster is not constrained to a particular shape or layout. Neither is it restricted in its size or extent. CommonNN clustering makes no assumptions in this

regard. 2) Not every molecular structure is a good representative for a stable conformation. This means, it is usually beneficial if a clustering can treat individual data points as outliers (noise), which is the case for the CommonNN method. 3) The representations of molecules, i.e. the data space in which they are clustered, can be high dimensional and complex. In general, it is not possible for example to know the correct number of conformational clusters that are to be found beforehand. The clustering should not require any prior knowledge about the data, or should allow easy data exploration. CommonNN offers systematic parameter screening and optimisation.

CommonNN clustering has proven to be a viable density-based clustering scheme. It yields intuitively correct clustering results in a wide range of challenging test data cases, as we showed previously in comparative benchmarks.[2,3] In application, CommonNN clustering has been for example successfully used to characterise the rich conformational ensemble of a foldamer and a tandem WW domain—two rigid protein domains connected by a flexible linker that sample a huge variety of relative orientations and domain-domain interfaces.[4] In other instances, the conformational clustering of small organic molecules has been applied in the context of ligand-protein interactions and pharmacophore modelling.[5,6] As a discretisation scheme for very well converged core-set Markov models,[7,8] CommonNN clustering is capable of detecting subtle changes in conformational equilibria and has been useful to explain differences in the membrane permeability of cyclic peptides[9] and to describe regulative allosteric processes.[10] Meanwhile, the CommonNN scheme found adaptation in a volume-scaled variant vs-CNN,[11] used in very recent research[12,13].

In this work, we present a revisited implementation of density-based CommonNN clustering and we provide an accessible Python package to make its use easier and more efficient for a broader audience.

We will quickly summarise the theoretical idea underlying the clustering method and how we approach its algorithmic realisation in section *Theoretical background*. In the sections "Basic usage", *Advanced usage*, and *Practical advice* we describe the usage of the package and some of the program design decisions. Section *Benchmark* provides a small benchmark of the new implementation.

Theoretical background

Consider a data set of points that should be clustered as a set of samples from an underlying probability density $p : \Omega \to \mathbb{R}_{\geq 0}$ with respect to a d-dimensional feature space $\Omega \subset \mathbb{R}^d$. Density-based CommonNN clustering rests on the idea of applying a density threshold λ to p that separates Ω into regions of high and low density like shown in figure 2 for 1D. Clusters are the resulting isolated, continuous regions of high density while everything below λ is noise. The set of possible clustering results is systematically gathered through variation of λ.

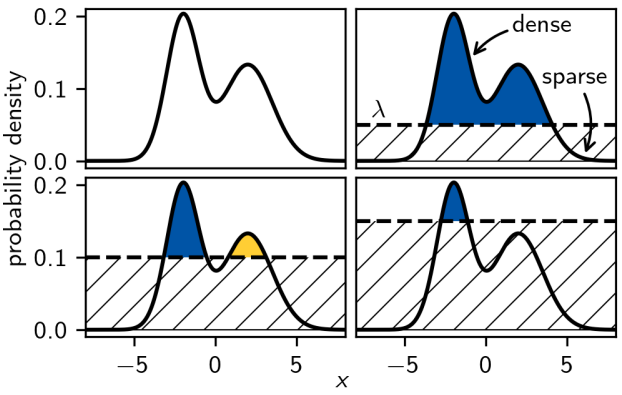


Figure 2: Example probability distribution in 1D with two maxima. Applying different density-thresholds λ splits the distribution into isolated high-density regions (clusters) separated by low density (noise).

Of course, the probability density p for an arbitrary data set is usually not known and difficult if not impossible to obtain. CommonNN clustering employs therefore a local estimate of the density, based on discrete samples: the proxy for the density is the number of data points within the neighbourhood intersection of two points. A point pair is identified to be part

of the same cluster if the density estimate exceeds a defined threshold (compare λ), i.e. if the two points share at least a number of c common neighbours with respect to their neighbourhoods circumscribed by a neighbour search radius r (see figure 3). The two points are in this case said to "fulfil the density-criterion" or "pass the similarity check". The density estimate requires the definition of a metric on the feature space that allows the calculation of pairwise distances and the determination of neighbourhoods.

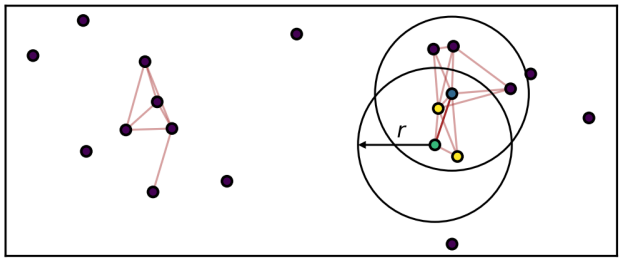


Figure 3: Illustration of the density-criterion in the CommonNN scheme for random points in 2D. The green and the blue data point share two of their neighbours with respect to the search radius r (yellow points). For a set value of $c \leq 2$, the two points are considered part of the same dense region, indicated by a red edge between them. A network of points connected in this way constitutes a cluster.

Programmatically, it is convenient to think of the clustered data set as a graph $G(V, E)$ in which each sample is represented by a node (vertex) v_i. Edges e_{ij} indicate pairwise relationships between points. If the edges correspond to whether two points v_i and v_j fulfil the density-criterion, the connected components—sub-networks of nodes that are disjoint from the rest—of the graph are the clusters that we want to find. Hence, the main task of the clustering can be solved by leveraging well established graph traversal algorithms, for example a breadth-first-search approach like shown in code snippet 1.

```
Init cluster label (to 1)

for node in graph:
  If node is visited:
    continue
  # New component
  Pick source node
  Mark node visited
  Push node to queue

  # Start breadth-first-search
  while queue is not empty:
    Pop node off of queue
    for node in connected_nodes:
      If node is visited:
        continue
      Assign label to node
      Mark node visited
      Push node to queue

  Increment cluster label
```

Snippet 1: Pseudo code for a breadth-first-search graph traversal to identify connected components (clusters) in a data graph. The CommonNN density-criterion determines if two nodes are connected.

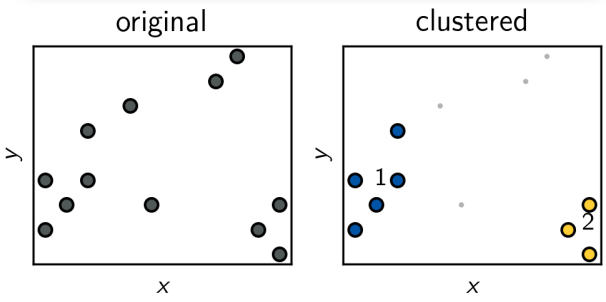
Basic usage

The CommonNNClustering package requires Python $\geq$ 3.6. It can be installed from PyPi (`pip install cnnclustering`) or from the development repository on GitHub (`https://github.com/janjoswig/CommonNNClustering`). The installation requires Cython, which is used to implement core functionalities efficiently. At runtime, NumPy is mandatory as well. Optionally, Matplotlib, Networkx, Pandas, scikit-learn, and scipy are leveraged for additional functionality. Documentation is available under `https://janjoswig.github.io/CommonNNClustering`.

Getting started with the clustering of any data set using the CommonNNClustering package is easy and should feel familiar to the use of similar available object-oriented Python APIs like that used by scikit-learn.[14] Code snippet 2

illustrates the four essential steps of a clustering: 1) how to import the main `cluster` module (line 1), 2) prepare a clustering object as an instance of the `Clustering` class from the data (line 5), 3) trigger the clustering (`fit`) itself with specified parameters (line 6), and 4) access the resulting cluster label assignments for further analysis (line 7). As a general design principle, we settled on a data oriented approach for the whole clustering procedure, which means that a created clustering object will be always associated with exactly one data set of some form. This data set can be clustered based on different combinations of cluster parameters, re-using the same clustering object.

```
1 from cnnclustering import cluster
2
3 # NumPy array (n_points, n_dim)
4 data = ...
5 default_cl = cluster.Clustering(data)
6 default_cl.fit(**params)
7 default_cl.labels  # array([...])
```



Snippet 2: Default cluster object creation and data point clustering. The scatter plots below are created using the convenience method `default_cl.evaluate()`.

The example assumes the presumably most frequent use-case of having the data presented as a NumPy array of shape (#points, #dimensions) or something equivalent for that matter, i.e. the data contains information on feature space coordinates for each sample point. The program can use these for the calculation of distances and effectively neighbourhoods to perform the clustering. This is, however, by far not the only possible scenario. In general, input data can be fed into a clustering in one of three fundamentally different formats: 1) point coordinates, 2) pairwise inter-point distances, or 3) fixed radius neighbourhoods. Each of these basic types of information can eventually be materialised in a multitude of different data structures. In particular, it is allowed and encouraged to leverage other specialised programs to take over the distance or neighbourhood calculation, e.g. with *kd*-trees as provided by scikit-learn.[14]

Advanced usage

Figure 4 gives a rough overview of how we achieve it that our package stays flexible with regard to different input formats and variations in other constraints like for example the used distance metric.

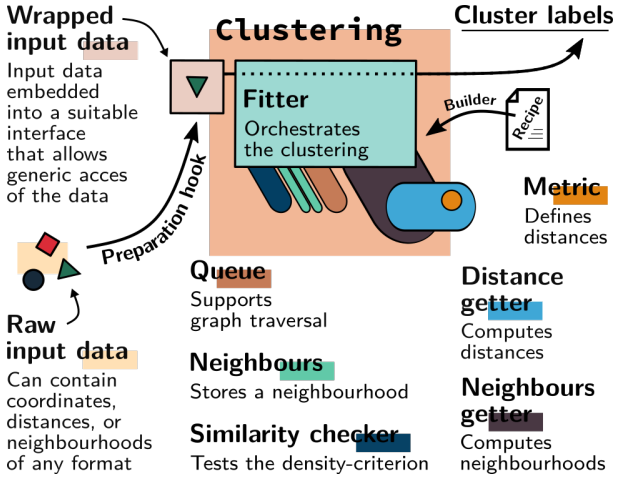


Figure 4: Aggregation of a clustering object from generic types representing exchangeable clustering components.

The central idea is the following: when the clustering algorithm is executed (e.g. as laid out in code snippet 1), it has to loop over input data points and query their neighbourhoods. Instead of accessing whatever input data structure may have been presented directly, the raw input data is wrapped within one of several *input data* objects that all can be worked with through a common generic interface, i.e. which are of a certain defined type. The task of acquiring neighbourhood information is delegated

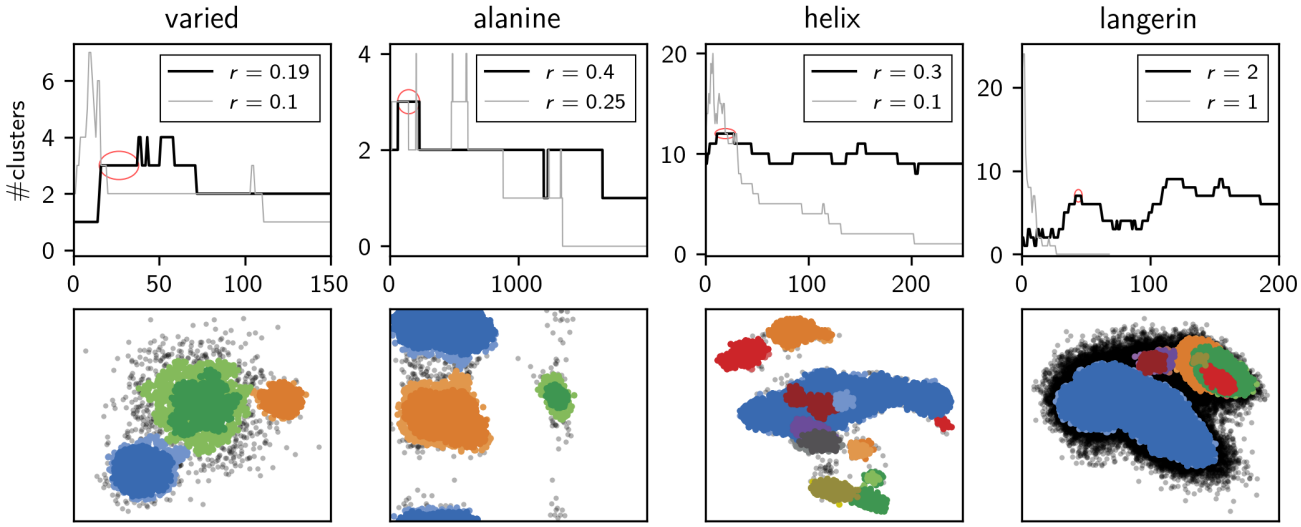


Figure 5: 1D cluster parameter scans: number of clusters vs. c for fixed radii r (proposed initial guess and much smaller). The most promising regions (stable, high cluster number) are marked with red. Clustering examples for parameters in the highlighted ranges below. Data points are coloured by cluster label in two shades of the same colour for the lowest and highest c value, respectively.

to one of many possible *neighbours getter* objects. In this way, the clustering that is itself implemented in a *fitter* object does not have to be concerned with how needed information is stored in and retrieved from the input data. In the same way, other important components as the testing of the density-criterion (*similarity checker* type), intermediate storage of retrieved neighbourhoods (*neighbours* type), or the metric (*metric* type) used for distance calculations (*distance getter* type) are represented by exchangeable objects adhering to generic interfaces. A clustering object as initialised in code example 2 aggregates all the objects needed for a clustering and is assembled in the background according to a *recipe*.

Please refer to the documentation for details on how the different interfaces are defined exactly, which generic types are available already, and how custom types and recipes can be defined and invoked.

Practical advice

Let's recall that the outcome of a CommonNN-clustering depends on the two cluster parameters r (neighbour-search radius) and c (CommonNN-cutoff). The higher the value of c compared to r, the higher is the estimated density required to be for two points ending up in the same dense cluster. Which values are eventually to be chosen for r and c depends strongly on the (subjectively) expected clustering result and on the nature of the data set (its distribution and sampling), and is in general not possible to decide *a priori*.

In selecting a suitable radius r, the aim is to choose a value that allows for a sufficiently local density estimate. In this sense, r functions as a kind of "resolution" for the clustering procedure. With a low resolution (a large radius r), local differences in the point density can not be detected, and hence no splitting of the data into clusters is achieved. On the other hand, if r is very small, fluctuations in the local point density that may originate from insufficient sampling, can lead to meaningless splittings of points into undesired clusters. Furthermore, with a high resolution (small values of r), the sensitivity of the density threshold towards the neighbour cutoff c is increased. As a heuristic for a good first guess on a neighbour search radius r, which allows an appropriately local density estimate, it has proven useful to take the distance value at which the distribution of pairwise inter-point distances has its first maximum. There is no such heuristic for c.

We propose now as a general strategy, to set r according to said heuristic and to screen c from small to large values, to obtain clusterings at increasingly high density thresholds. This allows to systematically select the favoured clustering result. For comparison, r may be adjusted slightly in both directions—in particular for high sampling rates, the resolution can usually be increased. Figure 5 illustrates this strategy with four representative data sets. When settling on a specific clustering result, it is advisable to look for a parameter range in which the clustering is qualitatively stable, meaning where only the size of individual clusters varies (shrinks for increased c) but not their number. The general observation is that while the density-criterion goes up, the number of clusters increases as more and more splittings occur. On the other hand the number will eventually go down again as more and more low density clusters fall below the density threshold and vanish into noise.

```
hierarch_cl.fit(**params)
hierarch_cl.isolate()

recluster_list = [(1, params_a),
                  (2, params_b)]
for subcl, params in recluster_list:
  hierarch_cl.children[subcl].fit(
    **params
  )

hierarch_cl.reel()
```

Snippet 3: Manual hierarchical clustering involves the isolation of clustering results and a re-clustering of child clusters.

It is often the case that the finally desired clustering result can not be achieved with a single parameter combination. The variation of the density-criterion essentially leads to a hierarchy of clusterings and individual clusters may be extracted at different levels of this hierarchy. We support currently two ways of doing this. Full user-control on each hierarchy level is offered by the manual approach as illustrated with code snippet 3 and figure 6. The idea is to follow the strategy of increasing c with fixed r to a point where the number of isolated clusters is locally maximised without to many low density clusters being lost into noise. Than this clustering result is isolated—i.e. frozen or saved—and the clustering is continued only on a subset of child clusters. Finally, the resulting hierarchy can be reeled—wrapped up—back into a single partitioning.

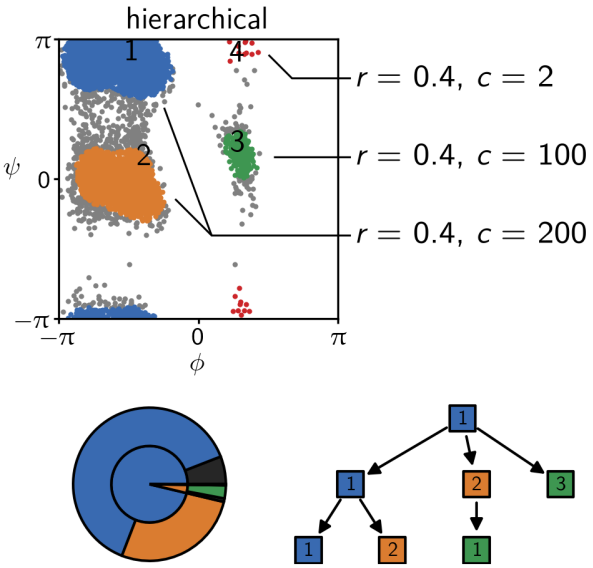


Figure 6: Manual hierarchical clustering of the *alanine* data set. Upper: Data points coloured by cluster label assignment after 2-step clustering (re-clustering of cluster 1 and 2 obtained in a first step). Lower: A hierarchy of clustering objects can be visualised as pie-chart showing the hierarchy levels going outward from high (root) to low (children) where the size of the pieces represents the amount of data points in a certain cluster. Alternatively, a Sugiyama tree-diagram can be drawn showing the splittings at individual hierarchy levels from top to bottom.

The second approach, is based on the idea to semi-automatically build the hierarchy of clusterings at certain levels by specifying a list of parameter combinations. This necessitates that in a second step, the resulting hierarchy needs to be screened according to some criteria for which child clusters should be kept as the final

result. Code snippet 4 and figure 7 illustrate this approach employing a hierarchy screen that simply avoids that clusters vanish completely. Again, please refer to the documentation for more details on hierarchical clustering and the different approaches.

```
r = 0.3
c = [1, 50, 100, 200, 500]

hierarch_cl.fit_hierarchical(
  r, c, member_cutoff=20
)
hierarch_cl.trim_trivial_leafs()
hierarch_cl.reel()
```

Snippet 4: Semi-automatic hierarchical clustering is based on the specification of a list of cluster parameters and selection of child clusters according to user-defined criteria.

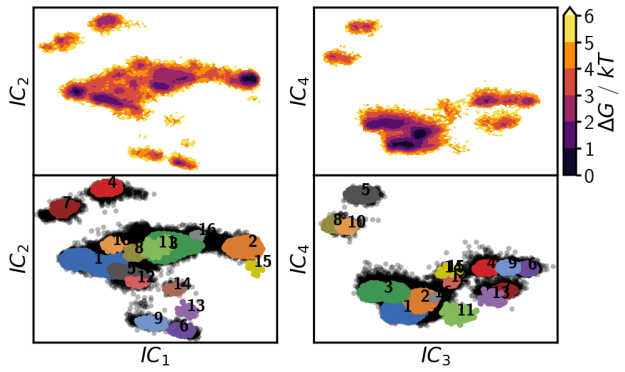


Figure 7: Semi-automatic hierarchical clustering on the *helix* data set. Upper: Pseudo free energy surface plot of the original distribution. Lower: Data points coloured by cluster label assignment after 5-step clustering and trimming of trivial leafs.

Benchmark

If large data sets are clustered or many parameter combinations are used, it becomes important that the clustering procedure can be executed relatively fast, and we will present a rough assessment of its performance. We would like to begin, though, with a word of caution. Accurate and fully representative benchmarks of the clustering are essentially hard to achieve, since the overall performance of the procedure, i.e. the total execution time needed to obtain cluster label assignments for the given input data, depends on many factors, for example: Which input source is used? Distance and neighbourhood calculation can be very costly. Which distance metric is used? Which data structures are utilised during the clustering? How is information retrieved and stored? How is the density-criterion tested? As the present generic implementation in the CommonNNClustering package is open to a customisation of the various components that play a role during the clustering, we would like to stress that the user is encouraged to mix and match the provided building blocks as needed, to make use of external methods e.g. by precomputing neighbourhoods, or even to implement their own specialised types with a matching interface.

Additionally, the observed clustering timings depend non-trivially on the cluster parameters and the structure of the data set. This is because the values for r and c define how many neighbours are retrieved or checked, and how quickly points can be assigned to clusters. It should be mentioned that besides computational efficiency in terms of speed, also the memory requirement of the employed data structures can be important.

That said, figure 8 shows the impact of data set size on the clustering performance with collected timings for clusterings under varying constraints using qualitatively the same data set but an increasing number of points. Empirical scaling was determined by fitting the measured execution times t versus problem size n to the power function $t \approx an^b$ where a is a proportionality constant and b is the growth factor of interest. The presented timings have been measured on a Debian 10 operating system, equipped with an Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz and 200 GB RAM. Note, that all implementations are serial at the moment. Par-

allel fitter schemes are planned for future releases. Timings are reported as the best out of ten repetitions. The used benchmark framework is organised under `https://github.com/janjoswig/CommonNNClustering-Benchmark`.

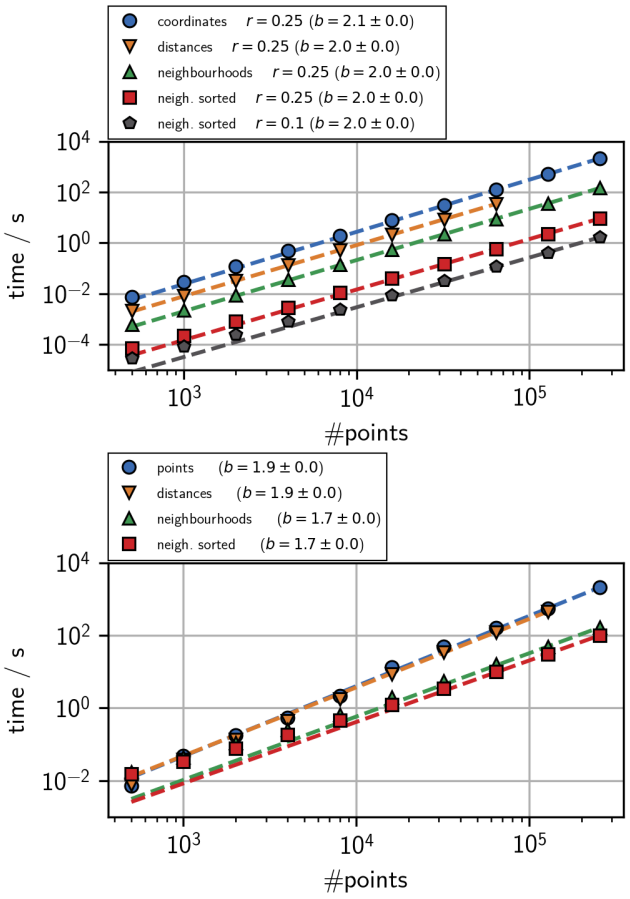


Figure 8: Clustering benchmarks for data sets with an increasing number of data points. **A** Comparison of different input data formats and corresponding default recipes measuring clustering execution times only, disregarding input data preparation. Based on uniformly distributed data points in 2D. The similarity cutoff was set to $c = 0$ in all runs. **B** Full clustering benchmarks including input data preparation time. Based on the *varied* data set. The similarity cutoff was set to $c = 50$ while the neighbour search radius was set to $r = 0.2$ initially and scaled down by a factor of 0.9 each time the number of points was increased. Note, that for the "distance" input the stored dense matrix eventually exceeded the available memory (164 GB for **A** and 200 GB for **B**).

Figure 8 **A** shows benchmarks on uniformly distributed points with fixed cluster parameters. The CommonNN-cutoff is set to $c = 0$, which means that the similarity criterion check is essentially skipped and the timings reflect only the general breadth-first search clustering procedure, including the construction of the intermediate neighbour lists.

We compare the clustering execution time for different input data formats and the corresponding default recipes (see the documentation for details on what these recipes entail). In all cases, we observe an empirically quadratic scaling of the computation time with respect to the number of points in the data set. It is obvious, however, that computing and sorting neighbourhood information prior to the clustering gives absolutely the best performance: 256,000 data points can be clustered on the order of seconds. We see that it is beneficial to choose a rather small radius r. This will keep the number of neighbours per point small, which leads to faster filling of the neighbours containers and also to a lower memory demand especially if neighbourhoods are pre-calculated.

Figure 8 **B** shows benchmarks on the *Varied* set for, which in contrast to **A** the time needed to prepare the input data—i.e. the pre-calculation of distances or neighbourhoods—is included in the measurements. The advantage of using pre-calculated distances on the clustering execution time is essentially nullified if the preparation time has to be considered for the overall clustering performance. Clustering from pre-computed neighbourhoods, however, clearly outperforms a clustering starting from point coordinates even if the preparation time is considered. Here, putting in the effort of sorting the neighbourhoods still offers a little edge.

Note that for the benchmarks in **B** the cluster parameter c was fixed but r was scaled down progressively, which can be justified for large (well sampled) data sets. This has the effect of producing sub-quadratic scaling.

Conclusion

We demonstrated the `cnnclustering` Python package that provides a convenient user in-

terface to threshold-based, density-based CommonNN clustering. The presented revised implementation rigorously improves our previous one in terms of clustering performance, usability and flexibility. We have described only a subset of the currently available functionality. Its generic design allows the application of the clustering procedure in a wide range of situations. The package is open to be extended with specialised types to cover additional use-cases—for example other forms of input data. Future work will be dedicated to broaden the array of available types and on incorporating computational parallelisation schemes into their design. Furthermore, automatic hierarchical clustering is explored.

References

(1) Sander, J. In *Encyclopedia of Machine Learning*; Sammut, C., Webb, G. I., Eds.; Springer US: Boston, MA, 2010; pp 270–273.

(2) Keller, B.; Daura, X.; van Gunsteren, W. F. Comparing geometric and kinetic cluster algorithms for molecular simulation data. *The Journal of Chemical Physics* **2010**, *132*, 074110.

(3) Lemke, O.; Keller, B. G. Common Nearest Neighbor Clustering—A Benchmark. *Algorithms* **2018**, *11*.

(4) Wenz, M.; Keller, B. G. *in preparation* **2021**,

(5) Lemke, O.; Götze, J. P. On the Stability of the Water-Soluble Chlorophyll-Binding Protein (WSCP) Studied by Molecular Dynamics Simulations. *The Journal of Physical Chemistry B* **2019**, *123*, 10594–10604, PMID: 31702165.

(6) Mortier, J.; Dhakal, P.; Volkamer, A. Truly Target-Focused Pharmacophore Modeling: A Novel Tool for Mapping Intermolecular Surfaces. *Molecules* **2018**, *23*.

(7) Schütte, C.; Noé, F.; Lu, J.; Sarich, M.; Vanden-Eijnden, E. Markov state models based on milestoning. *The Journal of Chemical Physics* **2011**, *134*, 204105.

(8) Lemke, O.; Keller, B. G. Density-based cluster algorithms for the identification of core sets. *The Journal of chemical physics 145*, 164104.

(9) Witek, J.; Wang, S.; Schroeder, B.; Lingwood, R.; Dounas, A.; Roth, H.-J.; Fouché, M.; Blatter, M.; Lemke, O.; Keller, B. G.; Riniker, S. Rationalization of the Membrane Permeability Differences in a Series of Analogue Cyclic Decapeptides. *J. Chem. Inf. Model.* **2019**, *59*, 294–308.

(10) Joswig, J.-O.; Anders, J.; Zhang, H.; Rademacher, C.; Keller, B. G. The molecular basis for the pH-dependent calcium affinity of the pattern recognition receptor langerin. *Journal of Biological Chemistry* **2021**, *296*, 100718.

(11) Weiß, R. G.; Ries, B.; Wang, S.; Riniker, S. Volume-scaled common nearest neighbor clustering algorithm with free-energy hierarchy. *The Journal of Chemical Physics* **2021**, *154*, 084106.

(12) Weiß, R. G.; Losfeld, M.-E.; Aebi, M.; Riniker, S. N-Glycosylation Enhances Conformational Flexibility of Protein Disulfide Isomerase Revealed by Microsecond Molecular Dynamics and Markov State Modeling. *The Journal of Physical Chemistry B* **2021**, *125*, 9467–9479, PMID: 34379416.

(13) Mathew, C.; Weiß, R. G.; Giese, C.; Lin, C.-w.; Losfeld, M.-E.; Glockshuber, R.; Riniker, S.; Aebi, M. Glycan–protein interactions determine kinetics of N-glycan remodeling. *RSC Chem. Biol.* **2021**, *2*, 917–931.

(14) Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.