WaveTrain: A Python package for numerical quantum mechanics of chain-like systems based on tensor trains

View Online   Export Citation   CrossMark

Jerome Riedel,[1,a] (iD) Patrick Gelß,[2,3,b] (iD) Rupert Klein,[2,c] (iD) and Burkhard Schmidt[2,4,d] (iD)

AFFILIATIONS

[1] Institut für Chemie, Freie Universität Berlin, Altensteinstraße 23A, D-14195 Berlin, Germany
[2] Institut für Mathematik, Freie Universität Berlin, Arnimallee 3–9, D-14195 Berlin, Germany
[3] Zuse-Institut Berlin, Takustraße 7, D-14195 Berlin, Germany
[4] Weierstraß-Institut für Angewandte Analysis und Stochastik, Mohrenstraße 39, 10117 Berlin, Germany

[a] jerome.riedel@fu-berlin.de
[b] p.gelss@fu-berlin.de
[c] rupert.klein@math.fu-berlin.de
[d] Author to whom correspondence should be addressed: burkhard.schmidt@fu-berlin.de

ABSTRACT

WAVETRAIN is an open-source software for numerical simulations of chain-like quantum systems with nearest-neighbor (NN) interactions only. The Python package is centered around tensor train (TT, or matrix product) format representations of Hamiltonian operators and (stationary or time-evolving) state vectors. It builds on the Python tensor train toolbox SCIKIT_TT, which provides efficient construction methods and storage schemes for the TT format. Its solvers for eigenvalue problems and linear differential equations are used in WAVETRAIN for the time-independent and time-dependent Schrödinger equations, respectively. Employing efficient decompositions to construct low-rank representations, the tensor-train ranks of state vectors are often found to depend only marginally on the chain length N. This results in the computational effort growing only slightly more than linearly with N, thus mitigating the curse of dimensionality. As a complement to the classes for full quantum mechanics, WAVETRAIN also contains classes for fully classical and mixed quantum–classical (Ehrenfest or mean field) dynamics of bipartite systems. The graphical capabilities allow visualization of quantum dynamics "on the fly," with a choice of several different representations based on reduced density matrices. Even though developed for treating quasi-one-dimensional excitonic energy transport in molecular solids or conjugated organic polymers, including coupling to phonons, WAVETRAIN can be used for any kind of chain-like quantum systems, with or without periodic boundary conditions and with NN interactions only. The present work describes version 1.0 of our WAVETRAIN software, based on version 1.2 of SCIKIT_TT, both of which are freely available from the GitHub platform where they will also be further developed. Moreover, WAVETRAIN is mirrored at SourceForge, within the framework of the WAVEPACKET project for numerical quantum dynamics. Worked-out demonstration examples with complete input and output, including animated graphics, are available.

I. INTRODUCTION

Progress in ultra-fast experimental techniques, in particular, the generation of ultra-short, intense laser pulses, has led to substantial advances in atomic and molecular physics, chemical reaction dynamics, material sciences, and related fields.[1] This has also motivated research in theoretical and simulation studies of quantum dynamics in recent years.[2–4] However, in marked contrast to electronic structure theory where a number of software packages have been under constant development for years or even decades and which have reached a remarkable degree of sophistication, general-purpose simulation software for quantum dynamics is relatively scarce. For example, QuTiP is an open-source Python framework for the dynamics of open quantum systems.[5,6] Another framework for closed and open quantum systems, coded in Matlab, aims at applications in quantum optics and condensed matter.[7] Furthermore, Libra offers a toolbox for quantum and classical dynamics simulations, including non-adiabatic processes in a molecular system.[8] This is

also a main feature of WAVEPACKET, a general purpose package for solving coupled Schrödinger or Liouville–von Neumann equations of closed and open quantum systems, respectively.[9–11] Additionally, it offers modules for fully classical and mixed quantum–classical dynamics on an equal footing, as well as a module for optimal control. The latter is the focus of QEngine,[12] a C++ library, and of Krotov, a Python implementation of quantum optimal control.[13]

Quantum dynamical simulations using any of the software packages mentioned above are limited to rather few degrees of freedom. This is because of the use of conventional grid techniques for the representations of quantum states and operators, thus suffering from the *curse of dimensionality*, i.e., the exponential growth of storage and CPU time with the number of dimensions. One way to overcome this problem is the Quantics general purpose program package,[14] which is based on the multi-configurational time-dependent Hartree (MCTDH) algorithm and its multi-layer (ML) extensions.[15,16] These packages are frequently used for complex quantum molecular dynamics simulation tasks, and they have evolved into a quasi-standard in the chemical physics community. From the quantum physics point of view, similar concepts are formulated in terms of tensor networks. In fact, it is well established that the (ML-)MCTDH algorithm corresponds to (hierarchical) Tucker tensor formats. For various types of tensor networks, the ITensor software library is available for practical calculations.[17] In particular, it contains the density matrix renormalization group (DMRG) algorithm for computing low-energy states of quantum systems.[18]

The present work deals with high-dimensional quantum dynamics using tensor train (TT) representations of quantum states and operators, also known as matrix product states (MPSs) and operators (MPOs).[19–21] The idea behind this format is to decompose a high-dimensional tensor into a chain-like network of lower-dimensional tensors, which enables us to simulate and analyze large-scale problems if the underlying coupling structure allows for low-rank decompositions. Several applications of tensor trains—which can be considered as a special case of the ansatz used in the multi-layer (ML) variant of MCTDH[15,16] mentioned above—and tensor-train operators have shown that it is possible to mitigate the curse of dimensionality and to tackle high-dimensional problems, which cannot be solved using conventional numerical methods, see, e.g., dynamical systems,[22,23] system identification,[24,25] quantum mechanics,[26–28] and also quantum machine learning.[29,30] Typically, the applications require the approximation of the solutions of systems of linear equations, eigenvalue problems, and ordinary/partial differential equations. For this reason, we use the open-source toolbox SCIKIT-TT (available through https://github.com/PGelss/scikit_tt), a general-purpose package for tensor trains written in Python based on NumPy and Scipy. It provides a powerful TT class as well as different modules for the automatic construction of tensor trains. Furthermore, SCIKIT-TT comprises different solvers for algebraic problems, which we need for our simulations.

Herein, we present version 1.0 of the WAVETRAIN software package, which specializes on high-dimensional quantum dynamics for systems with a chain-like topology and nearest-neighbor (NN) interactions only. Using tensor-train (TT) representations based on the so-called SLIM scheme,[31] this packet builds on SCIKIT-TT, thus providing efficient low-rank tensor approximation approaches that aim at reducing the exponential

scaling of the computational effort for solving time-independent and time-dependent Schrödinger equations in many dimensions. Being restricted to the SLIM scheme for TT representations for chain-like quantum systems with NN interactions, this approach is less general than other tensor schemes, such as the (hierarchical) Tucker format underlying the ML-MCTDH scheme, but has the advantage of very favorable scaling of the numerical effort with the chain length.

In our previous papers, the TT scheme was applied to the solution of the time-independent (TISE) and time-dependent Schrödinger equation (TDSE) for exciton–phonon systems of NN type, i.e., quasi-1D excitonic chains, ranging from few to about one hundred sites.[27,28] There, it was demonstrated that the storage consumption of the SLIM scheme scales linearly with the number of sites, and the scaling of the CPU time is only slightly less favorable. Moreover, for the case of the TISE, convergence with regard to the tensor rank was shown to be essentially independent of the system size. In another recent study, the efficiency in calculating ground states of chains of linear rotors interacting through their dipole moments was investigated. There, it was found that for these systems, a TT-based approach is less time- and memory-consuming than the state-of-the-art implementation of ML-MCTDH.[32,33] Finally, it is mentioned that the WAVETRAIN platform also contains modules for fully classical and hybrid quantum–classical dynamics, both for reference and/or for treating systems that are too complex for a fully quantum-mechanical treatment.

II. PHYSICAL SYSTEMS AND HAMILTONIANS
A. Tensor trains and the SLIM decomposition

Throughout the WAVETRAIN software package, we limit ourselves to the treatment of physical/chemical systems with a chain-like topology with NN (nearest neighbor) interactions only. For such systems, quantum-mechanical Hamiltonians H can be decomposed into operators S that act locally on single sites and products of operators L, M that couple NN pairs. Together with the identity operators I, this notation has given rise to the acronym for the so-called *SLIM decomposition*,[31] indicating the "slimness" in memory consumption. Thus, the *canonical representation* of the tensor $H \in \mathbb{R}^{(d_1 \times d_1) \times \cdots \times (d_N \times d_N)}$ for a chain with N sites consists only of elementary tensors, where at most two (adjacent) components are unequal to the identity matrix,

$$H = S_1 \otimes I_2 \otimes \cdots \otimes I_N + \cdots + I_1 \otimes \cdots \otimes I_{N-1} \otimes S_N$$

$$+ \sum_{\lambda=1}^{\xi_1} L_{1,\lambda} \otimes M_{2,\lambda} \otimes I_3 \otimes \cdots \otimes I_N$$

$$+ \cdots + \sum_{\lambda=1}^{\xi_{N-1}} I_1 \otimes \cdots \otimes I_{N-2} \otimes L_{N-1,\lambda} \otimes M_{N,\lambda}$$

$$+ \sum_{\lambda=1}^{\xi_N} M_{1,\lambda} \otimes I_2 \otimes \cdots \otimes I_{N-1} \otimes L_{N,\lambda}. \tag{1}$$

Here, all components S_i, $L_{i,\lambda}$, and $M_{i,\lambda}$ as well as the identities I_i are matrices in $\mathbb{R}^{d_i \times d_i}$ where d_i are the dimensions of the Hilbert spaces characterizing quantum states on the sites i. Note that the last line of Eq. (1) is only to comply with periodic boundary conditions of cyclic systems and can be omitted otherwise.

As shown in Ref. 31, the structure of such a Hamiltonian corresponds to the topology of a tensor train (TT, also known as matrix product) format. Gathering all components of $L_{i,\lambda}$ ($M_{i,\lambda}$) in corresponding core elements L_i (M_i) in a row-wise (column-wise) fashion (see Appendix 2 of Ref. 27) allows us to express Hamiltonian H in the following form:

$$H = \begin{bmatrix} S_1 & L_1 & I_1 & M_1 \end{bmatrix} \otimes \begin{bmatrix} I_2 & 0 & 0 & 0 \\ M_2 & 0 & 0 & 0 \\ S_2 & L_2 & I_2 & 0 \\ 0 & 0 & 0 & J_2 \end{bmatrix}$$

$$\otimes \cdots \otimes \begin{bmatrix} I_{N-1} & 0 & 0 & 0 \\ M_{N-1} & 0 & 0 & 0 \\ S_{N-1} & L_{N-1} & I_{N-1} & 0 \\ 0 & 0 & 0 & J_{N-1} \end{bmatrix} \otimes \begin{bmatrix} I_N \\ M_N \\ S_N \\ L_N \end{bmatrix}, \quad (2)$$

where J_i comprises ξ_i identity matrices I_i along the diagonal and zero matrices else. Note that the double square bracket notation does not stand for block matrices but for the compact tensor notation of Ref. 31. The Appendix of that work gives a proof of the above equation for all heterogeneous, cyclic systems. For homogeneous systems, the core elements S_i, L_i, I_i, M_i, and J_i do not depend on the site index i.

The ranks of the TT operator (2) are naturally bounded due to the restriction to NN interactions only, e.g., for homogeneous and periodic systems, we have $\xi_1 = \cdots = \xi_N =: \xi$ and, thus, $R = 2 + 2\xi$ (see Ref. 31). One of the main advantages of SLIM decompositions is the linear scaling of the memory consumption with N in case that the TT ranks of the solution do not increase with the order. Similarly, this also holds for the computational effort when considering time-independent and -dependent Schrödinger equations (see Secs. III B and III C, respectively). The considered SLIM decompositions in WAVETRAIN are constructed using SCIKIT-TT.

In Subsections II C–II E we will introduce exemplarily a few simple model Hamiltonians for chain-like systems with their SLIM decompositions and a description of the Python classes used for their respective implementations. In particular, those are classes for excitons, for phonons, and for exciton–phonon coupling in quasi-1D chains. Note that all these classes inherit from a common super-class for the implementation of the chain topology (see also the class hierarchy diagram shown in Fig. 1).

B. Super class *Chain*: General setup of linear or cyclic chain systems

The properties of the quasi-1D chain-like topologies underlying all of the present work are handled in super class *Chain*. For initialization, this class uses just three parameters. In addition to `n_site` giving the number of sites, N, the two Boolean variables `periodic` and `homogen` specify whether or not periodic boundary conditions are to be used and whether the chain is homogeneous or heterogeneous, respectively. According to the latter setting, all further parameters of the respective Hamiltonians are given either as
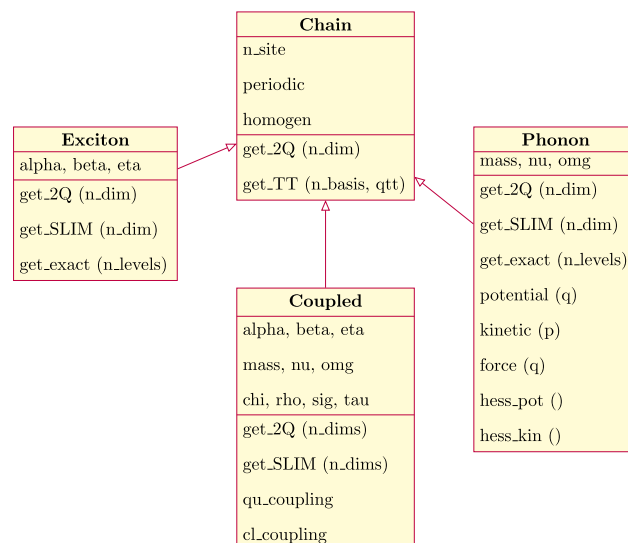

FIG. 1. Hierarchy of the Python classes representing the physical systems and Hamiltonians available as samples in WAVETRAIN. Selected attributes and methods of each class are given in the upper and lower parts, respectively, of the boxes. The corresponding Python files are located in the folder wave_train/hamilton.

scalars or as Python lists. Furthermore, the class *Chain* contains the following two methods of general use:

a. *Method get_2Q* is intended for quantum-mechanical Hamiltonians formulated in terms of the second quantization. For given dimension d (argument `n_dim`) of the local Hilbert space, which is assumed to be the same for each of the sites, this method sets up matrix representations of the raising ($a^\dagger$) and lowering (a) operators, as well as of the number operators. Where applicable, the position and momentum operators are obtained from the ladder operators.

b. *Method get_TT* is at the very heart of the SLIM formalism within our WAVETRAIN package. Given the lists of matrices $S_{i,\lambda}$, $L_{i,\lambda}$, I_i, and $M_{i,\lambda}$ (potentially independent of site index i for homogeneous chains) from one of the sub classes described below, this method serves to construct the tensor train super cores according to Eq. (2). Subsequently, an instance of class *TT* (tensor train) from the SCIKIT-TT package is created whose attributes (dimensions, ranks, and cores) are then set accordingly, depending on the initialization parameters `n_site`, `periodic`, and `homogen`.

C. Class *Exciton*: Electronic dynamics

As a first example, we introduce a simple Hamiltonian for the excitonic dynamics of atoms or molecules in a chain-like arrangement. For simplicity, we restrict ourselves here to a chain of two-state-systems, e.g., assuming only excitations of one electron from the highest occupied to the lowest unoccupied molecular orbital (HOMO–LUMO). Then, the excitonic Hamiltonian for a heterogeneous system of N sites can be given in terms of (bosonic) exciton raising, $b_i^\dagger$, and lowering, b_i, operators for site i,

$$H^{(\text{ex})} = \sum_{i=1}^{N} \alpha_i b_i^\dagger b_i + \sum_{i=1}^{N} \beta_i \left(b_i^\dagger b_{i+1} + b_i b_{i+1}^\dagger \right) + \eta, \qquad (3)$$

where α_i are local ("on site") excitation energies and η is a general offset of the energy scale. The nearest-neighbor (NN) coupling energies β_i between site i and $i + 1$, also known as "transfer integrals" or "hopping integrals," govern the delocalization and mobility of excitons within this simple model. Here and throughout the following, the last summand ($i = N$) of the NN coupling term (with indices $i + 1$ replaced by 1) is used for systems with periodic boundary conditions only and is omitted otherwise.

The most important methods in class *Exciton* are described in the following:

a. *Method __init__*: The following sample input illustrates the handling of excitons within our WAVETRAIN software package:

```
from wave_train.hamilton.exciton import Exciton
hamilton = Exciton(
    n_site = 6, periodic = True, homogen = True,
    alpha = 0.1, beta = -0.01, eta = 0.0
)
```

This creates an object of class *Exciton*, the definition of which is imported from sub folder *hamilton* in the `wave_train` source folder. Note that the first three arguments in the code above are used to initialize the super class *Chain* (see Sec. II B), whereas the remaining three arguments specify the energetic parameters α, β, η as given in Eq. (3), the values of which are taken here from our previous work in Refs. 27 and 28.

b. *Method get_SLIM*: Based on the above attributes of class *Exciton* and on the definition of the ladder operators in class *Chain*, this method provides the SLIM formulation of Eq. (3) yielding

$$
\begin{aligned}
S_i &= \alpha_i b_i^\dagger b_i + \frac{\eta}{N} I_i, \\
L_{i,1} &= \beta_i b_i^\dagger, \quad M_{i+1,1} = b_{i+1}, \\
L_{i,2} &= \beta_i b_i, \quad M_{i+1,2} = b_{i+1}^\dagger,
\end{aligned}
\qquad (4)
$$

where the dependence on the site index i is omitted for the case of a homogeneous chain. Note that method *get_SLIM* is called from within method *get_TT* in super class *Chain* to construct the tensor train supercores according to Eq. (2) (see Sec. II A). The following code line illustrates this for the case of excitons

```
hamilton.get_TT(n_basis = 2, qtt = False)
```

where the first argument gives the dimension d of the local exciton Hilbert space, i.e., the size of the electronic basis set.

c. *Method get_exact*: For the case of homogeneous excitonic chains, i.e., with all sites being equivalent, this method provides analytic/exact solutions of the time-independent Schrödinger equation (TISE) based on a Bethe ansatz as given in Ref. 27. In principle, the number of analytic solutions to be calculated can be chosen by the user (see Sec. III B). However, for linear systems, only the energy levels for the ground state and for the N states within the Fock space of singly excited states are currently

available, which are obtained in close analogy to Hückel theory. For cyclic systems, we also implemented the $N(N - 1)/2$ energy levels for states with two quanta of excitation.[34]

D. Class *Phonon*: Vibrational dynamics

As another example, we introduce a simple Hamiltonian for the vibrational (phononic) dynamics of a one-dimensional lattice model based on the harmonic approximation. In terms of site masses m_i, displacement coordinates R_i, and conjugate momenta P_i, a general Hamiltonian can be written as

$$H^{(\text{ph})} = \frac{1}{2} \sum_{i=1}^{N} \frac{P_i^2}{m_i} + \frac{1}{2} \sum_{i=1}^{N} m_i \nu_i^2 R_i^2 + \frac{1}{2} \sum_{i=1}^{N} \mu_i \omega_i^2 (R_i - R_{i+1})^2, \qquad (5)$$

where each site i is restrained around its equilibrium position by harmonic oscillators with frequencies ν_i. The NN interactions between neighboring sites i and $i + 1$ are modeled by harmonic oscillators with frequency ω_i and corresponding reduced masses $\mu_i = m_i m_{i+1}/(m_i + m_{i+1})$.

In analogy to the treatment of the excitons in Sec. II C, we re-formulate the phononic Hamiltonian of Eq. (5) using second quantization,

$$H^{(\text{ph})} = \sum_{i=1}^{N} \tilde{\nu}_i \left(c_i^\dagger c_i + \frac{1}{2} \right) - \sum_{i=1}^{N} \tilde{\omega}_i \left(c_i^\dagger + c_i \right) \left(c_{i+1}^\dagger + c_{i+1} \right), \qquad (6)$$

with raising ($c_i^\dagger$) and lowering (c_i) operators of (local) vibrations of site i. The effective frequencies of single site and NN pair vibrations are given by

$$\tilde{\nu}_i = \sqrt{\nu_i^2 + \frac{m_{i-1}}{m_i + m_{i-1}} \omega_{i-1}^2 + \frac{m_{i+1}}{m_i + m_{i+1}} \omega_i^2}, \qquad (7)$$

$$\tilde{\omega}_i = \frac{\mu_i \omega_i^2}{2\sqrt{m_i \tilde{\nu}_i m_{i+1} \tilde{\nu}_{i+1}}}, \qquad (8)$$

where for linear systems without periodic boundary conditions, the second or third term under the square root of Eq. (7) is omitted for the first ($i = 1$) or last ($i = N$) site, respectively. Note that the SLIM structure defined in Eq. (1) is apparent in our formulation (6) for the phononic Hamiltonian.

The most important methods in class *Phonon* are described in the following:

a. *Method __init__*: This sample input shows the setup of the phonon dynamics using the WAVETRAIN package

```
from wave_train.hamilton.phonon import Phonon
hamilton = Phonon(
    n_site = 6, periodic = True, homogen = True,
    mass = 1, nu = 1e-3, omg = 2**(1/2)*1e-3
)
```

which creates an object of class *Phonon*, which is imported from subfolder *hamilton* in the `wave_train` source folder. Again, the first three arguments in the code above are used to initialize the super class *Chain* (see Sec. II B), whereas the other three arguments specify the masses and frequency parameters m, ν, ω, as

given in Eq. (5). The initialization method of class *Phonon* also provides the effective frequencies $\tilde{v}$ and $\tilde{\omega}$ [see Eqs. (7) and (8)].

b. *Method get_SLIM*: Based on the above attributes of class *Phonon* and on the matrix representations of the ladder operators from super class *Chain*, the SLIM formulation of Eq. (6) is straight-forwardly expressed as

$$S_i = \tilde{v}_i\left(c_i^\dagger c_i + \frac{1}{2}\right),$$
$$L_{i,1} = -\tilde{\omega}_i\left(c_i^\dagger + c_i\right), \quad M_{i+1,1} = c_{i+1}^\dagger + c_{i+1}, \tag{9}$$

where the dependence on the site index i becomes irrelevant for a homogeneous chain. Again, method *get_SLIM* is called from within method *get_TT* (super class *Chain*) to construct the tensor train supercores [see Eq. (2)]. The use of this method is illustrated here,

```
hamilton.get_TT(n_basis = 8, qtt = False)
```

where the first argument gives the dimension d of the local phonon Hilbert space, i.e., the size of the harmonic oscillator vibrational basis set. In practice, this parameter needs to be determined by convergence tests. Typically, it depends on the total energy available in the simulated system.

c. *Method get_exact*: In addition, for the one-dimensional chain of oscillators given in Eq. (5), we implemented reference solutions for homogeneous chains to check the accuracy of the numeric TISE solvers described in Sec. III B. For periodic chains, analytic (Bloch type) solutions are well known (see Ref. 27). For non-periodic systems, where fully analytic solutions are not available because of the non-uniformity of the effective frequencies in Eqs. (7) and (8), energy levels are obtained from a conventional normal mode analysis, which is considered to be quasi-exact here. Note that this requires the calculation of the Hessian matrix of the phonon potential energy function of Eq. (5), which is provided in method *hess_pot* in class *Phonon* (see also Fig. 1).

E. Class *Coupled*: Exciton–phonon coupling

Because the excitonic energy transfer is known to be affected by coupling to vibrational degrees of freedom, the study of exciton–phonon coupling (EPC) is of vital importance, e.g., for the transport of electronic energy in semiconducting materials[35–37] or the transport of amide I vibrational energy in helical proteins.[38,39] Within the Hilbert space used for EPC, which is a direct product of the Hilbert spaces for the excitonic and phononic states, the total Hamiltonian can be written as

$$H = H^{(ex)} \otimes \mathbb{I}^{(ph)} + \mathbb{I}^{(ex)} \otimes H^{(ph)} + H^{(epc)}, \tag{10}$$

where $H^{(ex)}$ and $H^{(ph)}$ are the Hamiltonians for excitons and phonons [see Eqs. (3) and (6)] and where $\mathbb{I}^{(ex)}$ and $\mathbb{I}^{(ph)}$ are identity operators on the respective Hilbert spaces. A selection of simple,

Fröhlich–Holstein type Hamiltonians $H^{(epc)}$ for the coupling of excitons and phonons is implemented in WAVETRAIN,

$$\sum_{i=1}^{N} \chi_i b_i^\dagger b_i \otimes R_i = \sum_{i=1}^{N} \bar{\chi}_i b_i^\dagger b_i \otimes \left(c_i^\dagger + c_i\right),$$

$$\sum_{i=1}^{N} \rho_i b_i^\dagger b_i \otimes (R_{i+1} - R_i)$$
$$= \sum_{i=1}^{N} b_i^\dagger b_i \otimes \left[\bar{\rho}_i\left(c_{i+1}^\dagger + c_{i+1}\right) - \bar{\bar{\rho}}_i\left(c_i^\dagger + c_i\right)\right],$$

$$\sum_{i=1}^{N} \sigma_i b_i^\dagger b_i \otimes (R_{i+1} - R_{i-1})$$
$$= \sum_{i=1}^{N} b_i^\dagger b_i \otimes \left[\bar{\sigma}_i\left(c_{i+1}^\dagger + c_{i+1}\right) - \bar{\bar{\sigma}}_i\left(c_{i-1}^\dagger + c_{i-1}\right)\right],$$

$$\sum_{i=1}^{N} \tau_i \left(b_i^\dagger b_{i+1} + b_i b_{i+1}^\dagger\right) \otimes (R_{i+1} - R_i)$$
$$= \sum_{i=1}^{N} \left(b_i^\dagger b_{i+1} + b_i b_{i+1}^\dagger\right) \otimes \left[\bar{\tau}_i\left(c_{i+1}^\dagger + c_{i+1}\right) - \bar{\bar{\tau}}_i\left(c_i^\dagger + c_i\right)\right]. \tag{11}$$

Here, the EPC constants χ, ρ, and σ give the linear dependence of the excitonic site energies α on the positions of, or distances between, nearest or second-nearest sites, respectively. In contrast, the constants τ characterize the dependence of excitonic coupling energies β on the corresponding distances, thus including also Holstein–Peierls type models. The bar notation in Eq. (11) is used to convert the EPC constants to second quantization,

$$\bar{\chi}_i = \chi_i/\sqrt{2m_i\tilde{v}_i},$$
$$\bar{\rho}_i = \rho_i/\sqrt{2m_{i+1}\tilde{v}_{i+1}}, \quad \bar{\bar{\rho}}_i = \rho_i/\sqrt{2m_i\tilde{v}_i},$$
$$\bar{\sigma}_i = \sigma_i/\sqrt{2m_{i+1}\tilde{v}_{i+1}}, \quad \bar{\bar{\sigma}}_i = \sigma_i/\sqrt{2m_{i-1}\tilde{v}_{i-1}}, \tag{12}$$
$$\bar{\tau}_i = \tau_i/\sqrt{2m_{i+1}\tilde{v}_{i+1}}, \quad \bar{\bar{\tau}}_i = \tau_i/\sqrt{2m_i\tilde{v}_i},$$

Note that in our previous work,[27] the distinction between EPC constants with bars and double bars was missing, which, however, was not required for the cyclic systems mainly investigated there.

In the following, a description of important methods comprising class *Coupled* will be given as follows:

a. *Method __init__*: The following lines of input serve to create an instance of class *Coupled*

```
from wave_train.hamilton.coupled import Coupled
hamilton = Coupled(
n_site = 5, periodic = True, homogen = True,
alpha = 0.1, beta = −0.01, eta = 0.0,
mass = 1, nu = 1e−3, omg = 1e−3*2**(1/2),
chi = 0, rho = 0, sig = 1.6e−4, tau = 0
)
```

where the first nine arguments specify the chain topology, the excitons, and the phonons (see Secs. II B–II D, respectively). The last four arguments specify the parameters χ, ρ, σ, τ required for the different types of EPC models given in Eq. (11). For simplicity, we only consider the σ-coupling mechanisms in the present work.

b. *Method get_2Q*: Unlike classes *Exciton* and *Phonon*, which essentially use the inherited method *get_2Q* from super class *Chain*, the class *Coupled* overrides the super class method *get_2Q*. Here, one object of class *Exciton* and another object of class *Phonon* are created, along with their respective matrix representations for ladder operators. This allows a convenient calculation of direct products of excitonic and phononic operators, e.g., $b_i \otimes c_{i+1}$, using the Numpy function `kron` for the Kronecker product.

c. *Method get_SLIM*: This method is intended to provide the SLIM formulation of Eq. (11), which is given by

$$S_i = (\bar{\chi}_i - \bar{\bar{\rho}}_i) b_i^\dagger b_i \otimes \left(c_i^\dagger + c_i \right),$$

$$L_{i,1} = (\bar{\rho}_i + \bar{\sigma}_i) b_i^\dagger b_i, \quad M_{i+1,1} = c_{i+1}^\dagger + c_{i+1},$$

$$L_{i,2} = -\left(c_i^\dagger + c_i \right), \quad M_{i+1,2} = \bar{\bar{\sigma}}_{i+1} b_{i+1}^\dagger b_{i+1},$$

$$L_{i,3} = \bar{\tau}_i b_i^\dagger, \quad M_{i+1,3} = b_{i+1} \otimes \left(c_{i+1}^\dagger + c_{i+1} \right), \quad (13)$$

$$L_{i,4} = -\bar{\bar{\tau}}_i b_i^\dagger \otimes \left(c_i^\dagger + c_i \right), \quad M_{i+1,4} = b_{i+1},$$

$$L_{i,5} = \bar{\tau}_i b_i, \quad M_{i+1,5} = b_{i+1}^\dagger \otimes \left(c_{i+1}^\dagger + c_{i+1} \right),$$

$$L_{i,6} = -\bar{\bar{\tau}}_i b_i \otimes \left(c_i^\dagger + c_i \right), \quad M_{i+1,6} = b_{i+1}^\dagger.$$

Also here, the method *get_SLIM* is called within method *get_TT* of super class *Chain*, which constructs the tensor train super cores (see Sec. II B). The following code line illustrates this for the case of coupled excitons and phonons

```
hamilton.get_TT(n_basis = [2, 8] qtt = False)
```

where the Python list in the first argument contains the sizes of the electronic and vibrational basis sets, respectively.

III. QUANTUM AND CLASSICAL DYNAMICS

A. Super classes for quantum and classical mechanics

This section deals with the implementation of different types of physical/chemical dynamics within WAVETRAIN. The main work horses of our software package are the classes *TISE* and *TDSE* containing numerical solvers for the time-independent and time-dependent Schrödinger equation based on the TT tensor format (see Secs. III B and III C). For completeness, we have added classes *QCMD* and *CEoM* for mixed quantum–classical molecular dynamics and fully classical dynamics (see Secs. III D and III E).

The four main classes inherit from a set of super classes for quantum mechanics, mixed quantum–classical mechanics, and classical mechanics (see Fig. 2 for a class hierarchy diagram). Upon initializing objects of any of these classes, an input argument `hamilton` is required, which has to be an object of one of the three classes for excitons, phonons, or coupled systems explained above in Sec. II. Note that quantum–classical dynamics only works for coupled exciton–phonon systems while fully classical dynamics is restricted to phonons only. Most importantly, each of the three super classes provides a method *observe*, which deals with calculating and printing expectation values of important observables, such as energy, positions, and momenta of the particles. This is complemented by utility methods, such as calculations of "braket" scalar

products, expectation values with their uncertainties, and reduced density matrices for quantum simulations.

In turn, the three super classes inherit from the more fundamental class *Mechanics* for general mechanical systems. This class contains method *save*, which writes important quantities into binary data files, which can be either of Python "*pickle*" or of Matlab "*mat*" type. Those file types can also be read by method *load*, which thus serves to obtain deviations between the results of two simulations, e.g., for the case of different dynamic or different numerical schemes applied to the same physical problem and the same time discretization. Note that such a comparison is based on root mean squared deviations (RMSD), either for the quantum state vectors themselves, for populations, or for expectation values of observables, such as positions or momenta. The corresponding file names for such a comparison are set as properties `save_file` and `load_file`, and the type of comparison is set by the string `compare`. Moreover, class *Mechanics* also contains a method for linear regressions of conserved quantities, such as energy or norm of state vectors, or of the mentioned RMSDs. Finally, methods `gaussian` and `sec_hyp` can be used to set up wave packets with Gaussian or hyperbolic secant envelope, respectively (see the description of the eligible sub classes in Secs. III C and III D).

### B. Class *TISE*: Time-independent Schrödinger equation

Solving the time-independent Schrödinger equation (TISE),

$$\hat{H}|\Psi_n\rangle = E_n|\Psi_n\rangle, \quad n = 0, 1, \ldots, \quad (14)$$

yields a set of stationary quantum states $|\Psi_n\rangle$ along with their corresponding energies E_n, where $\hat{H}$ is one of the (time-independent) Hamiltonians presented in Sec. II or another one provided by the user. To beat the curse of dimensionality, the strategy followed in the WAVETRAIN software builds on low-rank tensor approximations for the state vectors, in analogy to the TT representation of the Hamiltonian given in Sec. II A. In practice, the eigenvalue problem is solved numerically using the alternating linear scheme (ALS), which is an iterative algorithm based on sequential contractions of the TT cores of $\hat{H}$ and $|\Psi_n\rangle$ to construct low-dimensional eigenvalue problems.[40] A key feature of the WAVETRAIN implementation is that not only ground states but also higher excited states can be obtained in an efficient way by means of integrated Wielandt deflation, which enables us to displace previously computed eigenvalues while keeping all other eigenvalues unchanged (see Ref. 27). To avoid an explosion of the computational costs for higher excited states, which would arise in a straight-forward application of the Wielandt deflation, the computation of the deflated Hamiltonians is implicitly incorporated into the ALS routine of SCIKIT-TT.

The use of class *TISE* is illustrated here by the following sample input:

```
from wave_train.dynamics.tise import TISE
dynamics = TISE(hamilton=hamilton, n_levels=10,
    solver = 'als', eigen = 'eigs',
    ranks = 15, repeats = 20, conv_eps = 1e-8,
    e_est = 0.08)
dynamics.solve()
```
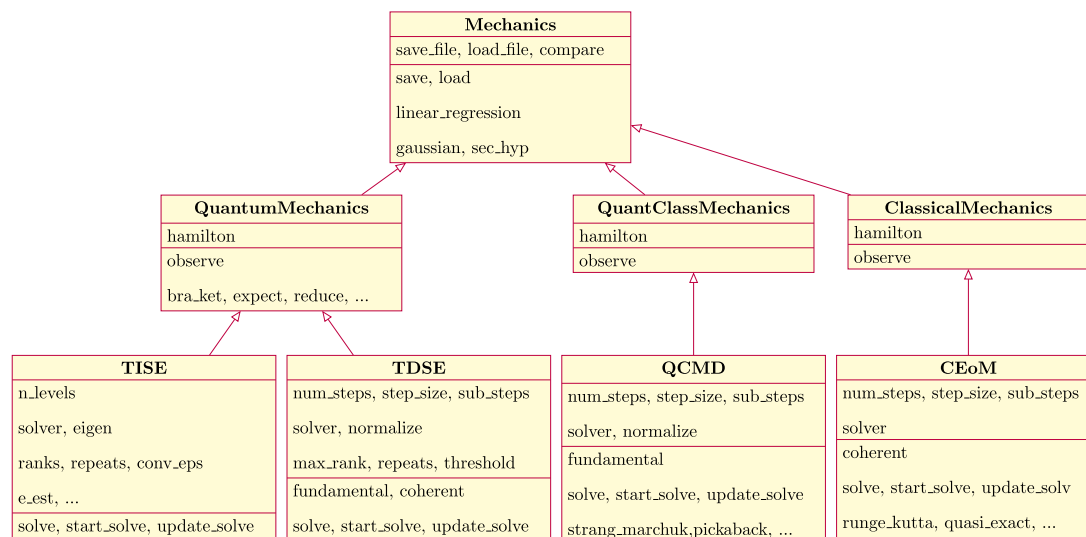
FIG. 2. Hierarchy of the Python classes for quantum and classical dynamics available in WAVETRAIN. Selected attributes and methods of each class are given in the upper and lower parts, respectively, of the boxes. The corresponding Python files are located in folder wave_train/dynamics.

where object `hamilton` pertains to one of the classes described in Sec. II and `n_levels` gives the desired number of eigenvalues to be calculated. The argument `solver` serves to choose the scheme to solve the full eigenproblem, by default the above-mentioned ALS algorithm, which is one of the key components of the `scikit_tt` package. The next argument, `eigen`, specifies the solver used for the micro-problems within each of the ALS iterations, in this case the sparse matrix eigensolver `"eigs"` from the SciPy package. Alternative choices are `"eig"` or `"eigh"`. The subsequent arguments serve to specify the ALS parameters, most importantly the number (`ranks`) of maximal ranks of the solutions. In all cases, ALS iterations are terminated once the estimated eigenvalues do not change by more than a certain threshold (`conv_eps`) in the last three ALS sweeps or when the number of sweeps reaches the limit given by attribute `repeats`. Finally, the parameter `e_est` gives an estimated energy (here, $\alpha - 2|\beta|$) close to which the energy levels are to be searched. If `eigen` is set to `"eig"`, eigenvalues closest to `e_est` are chosen from the list of all computed eigenvalues. Otherwise, Scipy's `"eigs"` uses the shift-invert mode to find the desired eigenvalues. This is of importance, e.g., when calculating the stabilization gained from mutual trapping of phonons and excitons from the lowest eigenvalue within the $\mathcal{N}^{(ex)} = 1$ manifold.[27] Typically, below that energy, there is a huge number of eigenvalues in the $\mathcal{N}^{(ex)} = 0$ manifold, which are not of interest and which can thus be excluded.

As an alternative, the WAVETRAIN package offers quasi-exact solutions, provided that the dimension of the full Hibert space, d^N, is not too large (typically 4096 for a standard PC). In that case, tensor train methods are bypassed and the eigenproblem for a matricized version of H is solved directly. This is invoked by setting `solver` = `"qe"` where the parameter `eigen` again specifies the choice of the numeric solver. While this option is clearly not eligible for

longer chains, it serves the purpose of creating reference solutions for shorter chains.

The resulting energy levels will also be compared against analytic (or semi-analytic) solutions, which are available only for the Hamiltonians (3) for uncoupled excitons and (6) for uncoupled phonons (see also Ref. 27).

C. Class *TDSE*: Time-dependent Schrödinger equation

The evolution of quantum states, $\Psi(t)$, is obtained as a solution of the time-dependent Schrödinger equation (TDSE) for one of the Hamiltonians $\hat{H}$ of Sec. II,

$$i\frac{d}{dt}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle, \quad |\Psi(t=0)\rangle = |\Psi_0\rangle, \quad (15)$$

where atomic units with $\hbar = 1$ are used. Again, the problem of high dimensionality is tackled by strategies building on low-rank tensor representations of the state vectors. Our implementation of class *TDSE* within WAVETRAIN builds on the choice of numeric propagators for tensor trains available within the `scikit_tt` software package. Restricting ourselves to explicit, reversible, and symplectic schemes, the most obvious choice is a symmetric, second order Euler (S2) method. This method has been routinely used in the quantum dynamics community for several decades, where it is also known as the second-order differencing scheme.[41,42] Within WAVETRAIN, also higher order variants, e.g., fourth (S4) and sixth (S6) order differencing methods are available. The former one has been shown to offer a good compromise between efficiency and accuracy.[28]

Frequently used alternatives are based on operator splitting originally developed for cases where Hamiltonians consist of kinetic and potential energy, which are treated separately in momentum and position representation, respectively.[43,44] In the present work, however, we resort to the Hamiltonians of Sec. II B for systems

with a chain-like topology and NN interactions only. For such cases, various novel splitting schemes are available in SCIKIT-TT, which are based on separating the interlacing pairs of NN sites.[18,28,45] Not only the classical first order Lie–Trotter (LT) and second order Strang–Marchuk (SM) schemes but also higher-order compositions of the basic methods are available, namely, the 4th order Yoshida–Neri (YN) and the 8th order Kahane–Li (KL) method, which have displayed an excellent accuracy in our test calculations.[28] For more information, see Ref. 46, where an overview of splitting methods with different order is given.

Finally, note that implicit schemes, such as the trapezoidal rule or the midpoint rule, are available within `scikit_tt` too. However, in our quantum dynamics test simulations, they have displayed a very unfavorable numeric effort because they involve the solutions of large-scale linear systems of equations. While the use of ALS[40] is an integral part of the *TISE* class, doing this at each time step results in an unfavorable numerical effort. Therefore, the *TDSE* class is solely based on explicit integration schemes.

To demonstrate the use of the *TDSE* class, we consider the following sample code lines:

```
from wave_train.dynamics.tdse import TDSE
dynamics = TDSE(hamilton = hamilton,
 num_steps = 50, step_size = 20, sub_steps = 5,
 solver = 's2', normalize = 0,
 max_rank = 8, threshold = 1e-12)
```

where the object `hamilton` refers to one of the Hamiltonian classes of Sec. II. Here, we propagate for 1000 (atomic) units of time, divided into 50 main time steps with a (constant) length of 20 units. After each of the main steps, expectation values of important observables are calculated and printed, and a frame is added to the (optionally generated) animated visualization (see Sec. IV). Internally, each of the main steps can be divided into a (constant) number of sub steps (here 5). The arguments `solver` and `normalize` for the initialization of class *TDSE* specify the choice of the numeric solver (two-letter codes explained above) as well as whether normalization of the state vector after every sub step is to be enforced or not. The remaining arguments are `max_rank`, the maximal rank in the decomposition of solutions $\Psi(t)$, and `threshold` the value of which is used for the rank truncation within the splitting schemes (LT, SM, YN, and KL) and the symmetric Euler (S2, S4, and S6) schemes. In both cases, an orthonormalization scheme called *higher-order singular value decomposition* (HOSVD)[47] with absolute as well as relative cut-off criteria for singular values is applied to keep the TT ranks of our solutions bounded by `max_rank`.

Before actually solving the TDSE, it is necessary to specify the initial state $|\Psi(t = 0)\rangle = |\Psi_0\rangle$. To that end, the class *TDSE* contains method `fundamental` to set up an initial state where one (or more) sites are fundamentally ($0 \to 1$) excited, while all others are prepared in their ground state. The resulting quantum state is constructed as a tensor train using the TT class from the SCIKIT-TT toolbox. That is, depending on a given vector of coefficients `coeffs`, the canonical representation of $|\Psi_0\rangle$ is given by the sum over tensor products of the form $\texttt{coeff}[j] \cdot \otimes_{k=1}^{n} v_j^{(k)}$ for non-zero coefficients, where

$v_j^{(k)} = [0, 1]^\top$ if $k = j$ and otherwise $[1, 0]^\top$. The created instance of the `TT` class then stores the cores of the corresponding TT representation of $|\Psi_0\rangle$.

While this method works in an analogous way for excitons and phonons, we note that for coupled systems, only the electronic parts are fundamentally excited, whereas the vibrational parts are in their ground states. In the following Python example

```
dynamics.fundamental()
dynamics.solve()
```

the default behavior is to return a state with a single excitation localized at the central site of the chain, which then serves as an initial state for solving the TDSE. It is also possible to give a vector of coefficients as input for method *fundamental*, in which case a weighted sum of products, each with a single site excitation, is returned. This feature of WAVETRAIN can be used, e.g., to construct bell-shaped wave packets with Gaussian or hyperbolic secant (sech) envelope with settable mean position, mean momentum, and width. The Gaussian shape is typically used to describe a free particle, whereas the sech shape typically occurs as a solution of nonlinear cubic TDSEs (see, e.g., Davydov's soliton theory).[39,48]

As an alternative to the use of fundamentally excited states, class *TDSE* also contains method `coherent`, which is meant only for vibrational systems (see our description of class *Phonons* in Sec. II D). That method serves to set up coherent states of the i-th site, which are eigenstates of the lowering operator c_i, defined as $c_i|\zeta\rangle_i = \zeta_i|\zeta\rangle_i$ with

$$|\zeta\rangle_i = e^{-\frac{|\zeta_i|^2}{2}} \sum_{k=0}^{\infty} \frac{\zeta_i^k}{\sqrt{k!}} |k\rangle_i. \quad (16)$$

Here, $|k\rangle_i$ stands for the kth harmonic oscillator eigenstate of the ith site, and

$$\langle R_i \rangle = \sqrt{\frac{2}{m_i \tilde{v}_i}} \zeta_i \quad (17)$$

gives the mean value of the displacement coordinate, R_i, of the respective quantum harmonic oscillator with mass m_i and effective frequency $\tilde{v}_i$. In analogy to method `fundamental`, also method `coherent` allows for the possibility of a combination of excitations of several sites.

In close analogy to class *TISE* described in Sec. III B, also class *TDSE* offers quasi-exact solutions for simulations where the full Hilbert space dimension is not too large. In that case, the matricized Hamiltonian is exponentiated, yielding a direct way to calculate the time evolution operator. This can be useful when benchmarking the accuracy of different propagation schemes and/or different time steps (see, e.g., our results in Ref. 28). Moreover, for two-state systems, class *TDSE* calculates analytic Bessel function solutions of the time evolution,[49] e.g., for class *Exciton* explained in Sec. II C. However, their use for benchmarking TT-based solutions is limited because they build on the assumption of non-periodic, infinitely-long chains.

D. Class *QCMD*: Quantum–classical molecular dynamics

The above-mentioned TT-based approaches implemented in WAVETRAIN can be very helpful instruments in tackling problems in quantum dynamics of bipartite systems, such as the example of coupled excitons and phonons mentioned in Sec. II E. On the one hand, we have shown that the computational effort is almost linear in N, which allows for treating long chains.[27,28] On the other hand, these methods can mitigate the *curse of dimensionality* only as long as the problem at hand allows for an acceptable accuracy of the approximate solution when we restrict ourselves to TT cores with ranks of manageable size. However, the computational effort for solving the TDSE scales at least with d^2 (symmetric Euler) where d is the dimension of the local Hilbert space. Hence, there are still simulation scenarios where a fully quantum-mechanical treatment is out of reach with the computational resources of today and probably also in the foreseeable future.

In many simulation scenarios, a clear separation of time and/or energy scales is found. In the above example of coupled excitons and phonons, the NN excitonic coupling energies β typically exceed the vibrational energies ν, ω, which is due to the disparity of electronic and nuclear masses.[50] In such cases, a promising way to overcome the *curse of dimensionality* is to resort to hybrid quantum–classical molecular dynamics where only the light (fast) subsystem is treated quantum-mechanically, while the classical approximation for the heavy (slow) subsystem is used. Such approaches appear especially suitable for problems where a large local Hilbert space dimension d is due to the latter subsystems being more complicated than those of Eq. (5). An example is conjugated polymer chains where the chromophoric sub-units are typically connected by a chain segment of several chemical bonds featuring a number of stretching, bending, and torsional degrees of freedom.[51,52]

The simplest quantum–classical approach is given by mean field or Ehrenfest dynamics, which rests on a separability *ansatz*. There, the state vector of the coupled system is assumed to be a single product of the two subsystem states, which is also known as the time-dependent Hartree method. Moreover, the quantum (excitonic) states can be restricted to the Fock space of singly excited states $\sum_{i=1}^{N} a_i(t) b_i^\dagger |0\rangle$ with time-dependent, complex coefficients $a_i(t)$ and with $|0\rangle$ standing for the electronic ground state. While this assumption neglects couplings to states bearing two or more excitons, it renders a TT-based approach for the excitons unnecessary.

For the example of the Hamiltonians of coupled excitons and phonons introduced in Sec. II, the evolution of the quantum sub-system (excitons) is governed by a Schrödinger-type equation,

$$i\frac{\mathrm{d}a_i}{\mathrm{d}t} = \left[\alpha_i + \sigma_i(R_{i+1} - R_{i-1}) + W\right]a_i + \beta_{i-1}a_{i-1} + \beta_i a_{i+1}, \quad (18)$$

where $a_i(t)$ are the expansion coefficients of the excitonic state and where W stands for the (classical) energy of the phonons. Furthermore, the dynamics of the classical sub-system (phonons) is described in terms of a classical trajectory, $R(t)$, which is governed by a Newton-type equation,

$$m_i\frac{\mathrm{d}^2 R_i}{\mathrm{d}t^2} = -m_i\nu_i^2 R_i - \mu_{j-1}\omega_{j-1}^2(R_i - R_{i-1}) - \sigma_{i-1}|a_{i-1}|^2$$
$$+ \mu_j\omega_j^2(R_{i+1} - R_i) + \sigma_{i+1}|a_{i+1}|^2. \quad (19)$$

Note that here the two sub-systems given in Eqs. (18) and (19) are coupled to each other through terms proportional to the EPC constants σ_i defined in the third row of Eq. (11). For detailed discussions of the asymptotics and error estimates of the separability ansatz and/or the classical approximation (see, e.g., Refs. 39, 48, and 53–55).

The use of class *QCMD* is shown in the following sample code lines:

```
from wave_train.dynamics.qcmd import QCMD
dynamics = QCMD(hamilton = hamilton,
  num_steps = 50, step_size = 20, sub_steps = 5,
  solver = 'sm', normalize = 0)
dynamics.fundamental()
dynamics.solve()
```

where `hamilton` has to be an object of class *Coupled* (see Sec. II E) or another class for bipartite systems provided by the user. Note that in order to be used for Ehrenfest quantum–classical mechanics simulations, such classes have to provide methods *qu_coupling* and *cl_coupling* returning the couplings of one sub-system to the respective other one (see also Fig. 1). For numerically solving the QCMD scheme, there is a choice of numerical propagators implemented within the *QCMD* class, such as a generalized Lie–Trotter (`"lt"`) and Strang–Marchuk method (option `"sm"` in the example code above), as well as the symplectic pickaback (`"pb"`) propagator.[56]

The choice of initial conditions for the quantum sub-system (e.g., excitons) is the same as in Sec. III C for class *TDSE*, i.e., fundamental electronic excitations, with the possibility for Gaussian bell-shaped and sech-shaped superpositions thereof. Note that initial excitations of the classical sub-system (e.g., phonons) are at present not yet implemented.

E. Class *CEoM*: Classical equations of motion

Moreover, we have added a class for solving classical (Newton's or Hamilton's) equations of motion to the WAVETRAIN package. The motivation for this is to generate reference solutions for systems where a classical analog to the quantum-mechanical Hamiltonian exists. Hence, this class works, e.g., with objects of class *Phonon*. According to the Ehrenfest theorem, quantum-mechanical expectation values of observables, such as positions and momenta, coincide with results from classical trajectories, as long as the vibrational Hamiltonian is a polynomial of order not higher than two, which is indeed the case for our harmonic model Hamiltonian (5). There, the positions are governed by the Newton-type Eq. (19) but without the σ term for the EPC.

The use of class *CEoM* is illustrated in the following code lines:

```
from wave_train.dynamics.ceom import CEoM
dynamics = CEoM(hamilton = hamilton,
  num_steps = 50, step_size = 20, sub_steps = 5,
  solver = 'rk', normalize = 0)
dynamics.coherent(displace=[1.0 if i = =
hamilton.n_site//2 else 0.0 for i in
range(hamilton.n_site)])
dynamics.solve()
```
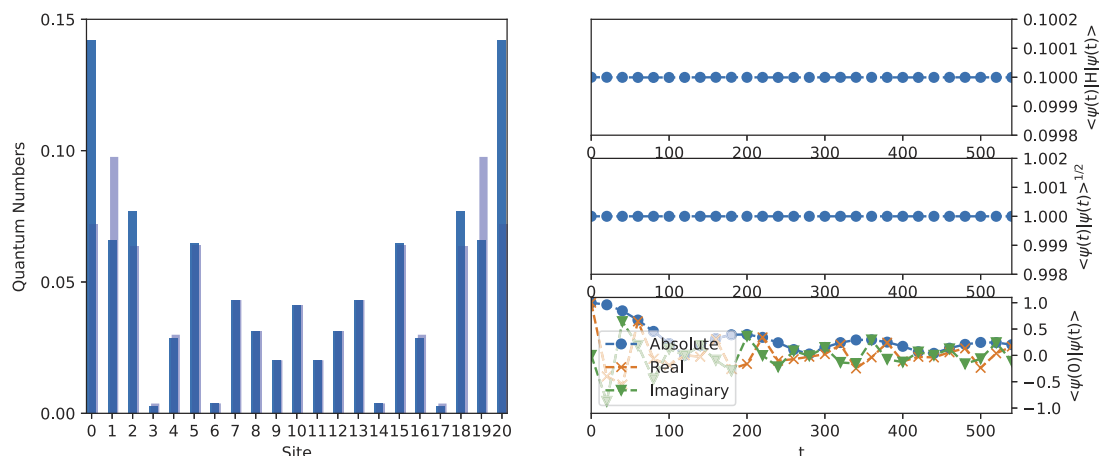
FIG. 3. Quantum dynamics of excitons on a linear chain. Left panel: snapshot of quantum numbers for each of the sites. Right panel: evolution of the mean energy and the norm of the state vector vs time, as well as the autocorrelation function.

TABLE I. Overview of the different visualization services and their cross-dependencies regarding systems and dynamics. Upper four options for simple systems, lower three options for bipartite systems.

| Service	System	Dynamics	Description
QuantNumbers	Exciton, phonon	TISE, TDSE	Mean quantum numbers
Populations	Exciton, phonon	TISE, TDSE	Populations of quantum states
DensityMat	Exciton, phonon	TISE, TDSE	Reduced density matrices
PhaseSpace	Phonon	TISE, TDSE, CEoM	Mean trajectories in phase space
Positions2	Coupled	TISE, TDSE, QCMD	Excitonic quantum numbers and lattice distortions as line plots
QuantDisplace2	Coupled	TISE, TDSE, QCMD	Excitonic quantum numbers and lattice distortions as bar plots
QuantNumbers2	Coupled	TISE, TDSE,	Excitonic and phononic quantum numbers as bar plots

where `hamilton` is an object of class *Phonon* (for a description, see Sec. II D) or another class provided by the user for a system for which the use of the classical approximation is justifiable. Note that for use in classical mechanics simulations, such classes have to encompass additional methods for the calculations of forces and of classical potential and kinetic energy (see also Fig. 1). For numerically solving the classical equations of motion, there is a choice of propagators implemented in class *CEoM*, such as the Runge–Kutta (option `"rk"` in the example above) and the Velocity-Verlet (`"vv"`) scheme. In addition, quasi-exact solutions for the harmonic vibrations are available, which require additional Python methods to calculate the Hessian matrices of the potential and kinetic energy functions [see also Eq. (25) of Ref. 28]. In the sample code above, method *coherent* of class *CEoM* is used to provide classical initial conditions equivalent to those of a coherent state of quantum harmonic oscillators with the displacement of the classical particles given by Eq. (17), here with $\langle R \rangle = 1$ at the central site and $\langle R \rangle = 0$ everywhere else.

### F. Class *load*: Loading data from a previous simulation

In addition to generating solutions of (stationary or dynamical) equations of motion as described in Subsections III B-III E, WAVETRAIN also offers the possibility of loading previously generated solutions, If, for example, a TDSE simulation is run with option `load_file=tdse_1.pic`, essential data are stored in a Python pickle file by virtue of method *save* in class *Mechanics* (see Sec. III A). Subsequently, this information is easily retrieved using class *Load*

```
from wave_train.io.load import Load
dynamics = Load('tdse_1', 'pic')
```

The created object contains not only expectation values of important physical observables, which can be used for automated analysis of series of runs, but also the TT representation of the last bound state (TISE) or the state at the last time step (TDSE), which
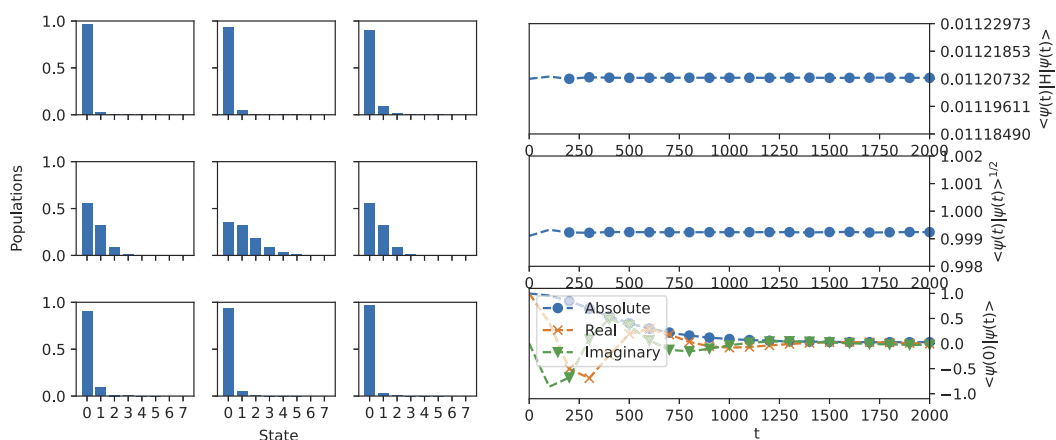
FIG. 4. Quantum dynamics of phonons on a linear chain. Left panel: snapshots of populations of harmonic oscillator states for each of the sites, arranged in a row-wise manner. Right panel: same as in Fig. 3.
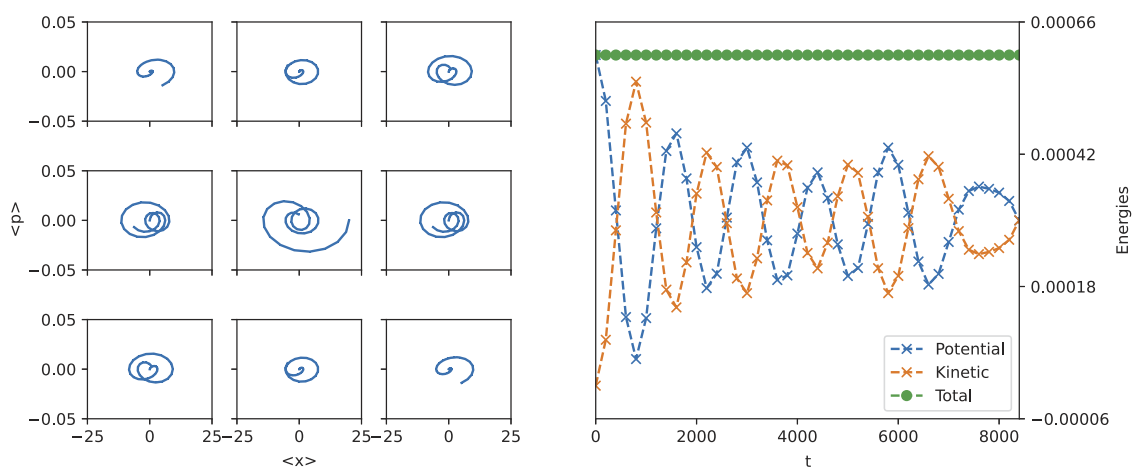
FIG. 5. Classical dynamics of phonons on a linear chain. Left panel: trajectories in phase space for each of the sites, arranged. Right panel: total energy vs time, along with its decomposition in kinetic and potential contributions.

allows for an easy restart of a simulation. Finally, objects of class *Load* contain also reduced density information, which serve the purpose of creating a new (or different) animated visualization without having to perform another full simulation (see also Sec. IV).

IV. GRAPHICAL OUTPUT

The ability to create rich graphical output is one of the hallmarks of simulations with the WAVETRAIN software. To meet the demand of users for rich and insightful graphical representations, the software package provides a set of default visualization classes. They allow the user to track the progress and stability of computations at run time or to create graphical output of previously generated results by utilizing the *Load* class (see Sec. III F). After

completion of a simulation, the plots are available not only as images (png file format) but also as animations (mp4 file format), which are created using the *ffmpeg* tool.[57]

Classes for visualization are created based on a Dependency Injection (DI) scheme, with different visualization services being injected into the main class *Visual*, that handles the execution order of the respective services. Generally, visualizing the results of solving the equations of motion introduced in Sec. III clusters into two independent services that can be separately added to the pipeline for creating visual output. In the main service step, the current quantum or classical state is visualized in a collection of subplots for each of the sites or in a single view along a discretized axis of site indices, in both cases shown in the left half of the generated figures. Additionally, a second service can be added to monitor system properties, i.e., energy (TISE, TDSE, QCMD, and CEoM), norm (TISE, TDSE, and
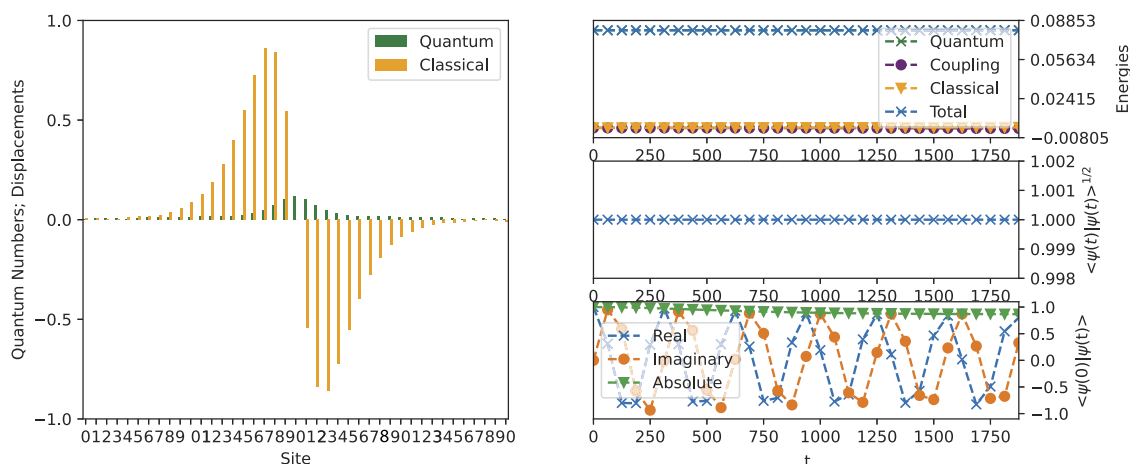
FIG. 6. Quantum–classical dynamics of coupled excitons and phonons on a linear chain. Left panel: snapshots of mean quantum numbers of the excitons (green bars) and vibrational displacements (scaled by 0.1) of the sites (orange bars). Upper right panel: total energy vs time, along with its decomposition in contributions of the quantum and the classical subsystem, as well as the quantum–classical coupling. Middle and lower right: norm and autocorrelation function of the quantum subsystem only.

QCMD), and autocorrelation function (ACF), $C(t) = \langle \psi(0)|\psi(t)\rangle$ (TDSE and QCMD), optionally displayed in the right half of the generated figures.

The following code snippet illustrates the setup of an animation for visualizing the quantum dynamics of a single system, e.g., a chain of excitons, as shown in Fig. 3:

```
from wave_train.graphics.factory import VisualTDSE

graphics = VisualTDSE(
    dynamics = dynamics,
    plot_type = 'QuantNumbers',
    plot_expect = True,
    movie_file = 'tdse.mp4').create()
graphics.solve()
```

Here, it is assumed that *dynamics* is a previously created object of class TDSE, as described in Sec. III C. This instance is then inserted into the factory constructor, which becomes responsible for internal logical checks, e.g., whether plot type, *hamilton* instance, and *dynamics* instance are compatible. The *VisualTDSE* factory returns an instance of the *Visual* class after a call to the *create* method, which will inject the respective services. Visualizations of different *dynamics* instances follows the same logic, with equivalent factory classes being provided for TISE, QCMD, and CEoM. In the above code snippet, the service *QuantNumbers* for displaying average quantum numbers for each of the sites has been selected, and the toggle (*plot_expect*) for the visualization of system properties (expectation values of norm and energy, ACF) has been activated. The setup for high-level visualization of these observables is routed through the factory interface, which provides the factory classes for the different dynamics implemented in the WAVETRAIN software (i.e., TISE,

TDSE, QCMD, and CEoM). Note that the new instance *graphics* provides a proxy to start solving the Schrödinger equation, thus replacing the calls to *dynamics.solve* in the code snippets given previously in Sec. III. Finally, specifying the *movie_file* keyword argument allows us to create animated output in mp4 file format.

The visualization of the system state in the left half of the figures is based on the reduced density formalism. Once calculated, the reduced density matrices for each site can be shown directly or in the form of populations or averaged quantum numbers, positions, and/or momenta. For an overview of the different visualization options, see Table I. Table I also lists the special graphics services designed for use with bipartite systems, e.g., the coupled exciton–phonon systems described in Sec. II E.

Optionally, the system properties can be visualized in the right half of the figures. These properties are directly calculated as overlaps or as expectation values by utilizing the tensor product as provided by *scikit_tt*. Where possible, system properties are always separated into their individual contributions, e.g., for bipartite systems, the state space is visualized for the two sub-systems separately. Furthermore, for CEoM simulations, the system energy is split into kinetic and potential energy contributions, whereas for QCMD, energy contributions are decomposed into the contributions from the quantum and classical subsystem, as well as the energy pertaining to the quantum–classical coupling.

Typical graphical output from WAVETRAIN is illustrated and discussed for four selected cases.

- Figure 3 is a visualization of the quantum dynamics of excitons on a linear chain of length $N = 21$, with parameters from Sec. II C. The left half is showing a snapshot for $t = 540$ after an initial excitation of the central site ($i = 11$) only. The semi-transparent bars show analytic solutions, which are available for infinitely long chains of two-level systems with

NN coupling only.[49] While analytic and numerical results agree well in the middle of the chain, there are considerable discrepancies near the edges of the chain, as expected.

- Figure 4 shows the quantum dynamics of phonons on a linear chain of length $N = 9$, with parameters from Sec. II D. The left panel represents snapshots for $t = 2000$ after an initial excitation of the central site ($i = 4$) to a coherent state with $\langle R \rangle = 50$. For this value of the initial displacement, the representation of quantum state vectors in terms of eight basis functions per site is almost large enough, with tiny deficiencies still visible in the deviation of the norm of the state vectors from unity.

- Figure 5 visualizes the dynamics of phonons on a linear chain of length $N = 9$, with parameters from Sec. II D. The left panel shows phase-space portraits for $0 \leq t \leq 8400$ starting from an initial displacement with $\langle R \rangle = 20$ of the central site ($i = 4$) only. Note that for the quadratic Hamiltonian of Eq. (5), resulting expectation values from quantum and classical dynamics coincide by virtue of the Ehrenfest theorem.

- Figure 6 shows the quantum-classical dynamics of coupled excitons and phonons on a linear chain, with parameters from Sec. II E. The left part of the figure shows a snapshot at $t = 1875$ after preparing an initial state with a sech-like distribution of an exciton peaked around the central site ($i = 20$) (see Ref. 27) but without vibrational excitation. Hence, this simulation shows the formation of a soliton or, more precisely, the onset of the dressing of an exciton with phonons in real time.

V. DOWNLOAD AND INSTALLATION

The WAVETRAIN software is a pure Python3 package and can be readily installed from the PyPI package index using pip. A command line installation of the WAVETRAIN software can be achieved by issuing the following command in a terminal environment:

```
$ pip install wave_train
```

where pip installs into a Python3 installation with minimum version requirement 3.7.0. The source code is publicly available in the Github repository *PGelss/wave_train* under the GNU General Public License v3.0. For a developer installation of WAVETRAIN, a specific version of *sckit_tt* may be required, which can be readily installed from the Github repository *PGelss/scikit_tt*. By default, WAVETRAIN installs with the latest *scikit_tt* version.

VI. CONCLUSIONS AND PROSPECT

In the present work, we have illustrated the use of WAVETRAIN for rather simple models of excitons and phonons from our previous works.[27,28] However, it is straight-forward to apply our software to a variety of other quantum systems, as long as they are of a linear or cyclic chain-like topology with on-site and NN interactions only. Obvious extensions of the models given above include exciton dynamics with more than two electronic states per site (e.g., singlet and triplet states) and/or anharmonic description of phonons.

Note that in the latter case, one does not necessarily have to use the second quantization introduced in Eq. (6). It is also possible to use, e.g., pseudo-spectral representations in coordinate space to discretize the vibrational degrees of freedom.[9,58] The flexible structure of the WAVETRAIN package also allows for easy implementation of other types of quantum systems, such as chains of spin systems (Ising or Heisenberg models), chains of molecular rotors,[33] or polarons in one-dimensional lattices.[59] In all those cases, one would have to design a new Python class for the underlying Hamiltonian, which inherits from the super-class *Chain*. It is recommended for such a class to have a method *__init__* dealing with the physical parameters of the Hamiltonian (class attributes) and a method *__str__* generating a string for print output. A mandatory ingredient of such a class is a method *get_SLIM*, providing the S, L, I, M matrices from which to construct the tensor cores[31] (see also Eqs. (1) and (2) in Sec. II A).

Moreover, the object-oriented architecture of the WAVETRAIN package also supports a straight-forward addition of Python classes for further types of equations of motion. An obvious choice is the Liouville-von Neumann equation (LvNE) adding dissipation and decoherence to quantum dynamics. In that case, tensor trains will be used for the representations of the density matrices, and numerical solution of the LvNE will rest on the efficient ordinary differential equation (ODE) solvers available in the scikit_tt toolbox, similar to our implementation of class *TDSE* described in Sec. III C. A frequently used alternative to the Ehrenfest or mean field quantum–classical approach implemented in class *QCMD* is the surface hopping trajectory method, featuring stochastic hopping between different electronic states.[60] In such a case, TT representations are not required, and a corresponding class should contain its own propagation methods, as is also the case for class *QCMD* described in Sec. III D. Yet another option could be diffusive Langevin dynamics adding friction and stochastic forces to classical dynamics, thus extending the class *CEoM* (see Sec. III E). Note that the classes for these three examples will inherit from the respective super classes for fully quantum, mixed quantum–classical, and purely classic dynamics, as described in Sec. III A. Moreover, when writing a new Python class for another type of dynamics, the following methods will have to be implemented: In addition to a method *__init__* for initialization and a method *__str__* for print output, it is mandatory for such a class to encompass a method *solve*. That method calls *start_solve* used for initialization of the numerical solvers, e.g., propagation one step backward in time, which is required for the symmetric Euler scheme to solve the TDSE. Subsequently, for every time step, a method *update_solve* is called that actually carries out the propagation. Finally, it should be mentioned that each of the dynamics classes needs to have (one or several) method(s) to generate an initial system state.

The WAVETRAIN software package is hosted and further developed at the Github platform, along with the scikit_tt toolbox for tensor train computations on which it is based. Moreover, WAVETRAIN is mirrored at the SourceForge platform, as a part of the WAVEPACKET project for numerical quantum dynamics, which is already in use for a number of years in several labs.[9–11] That MATLAB software package also features quantum and mixed quantum–classical dynamics, but for general Hamiltonians, i.e., without the restriction to chain-like topologies. The recently published version 7.0 of WAVEPACKET contains a MATLAB class definition for the Hamiltonians of Eqs. (3), (5), and (11). Hence,

integration of WAVETRAIN into the WAVEPACKET project allows for a simple and direct comparability of results, thus allowing to benefit from the easy usability and the advanced graphical capabilities of the latter one. However, such comparisons will have to be limited to short chains up to $N \approx 3$ for TISE or $N \approx 6$ for TDSE because—without the use of tensor train methods—WAVEPACKET suffers from the *curse of dimensionality*.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Jerome Riedel: Investigation (equal). **Patrick Gelß**: Investigation (equal). **Rupert Klein**: Investigation (equal). **Burkhard Schmidt**: Investigation (equal).

DATA AVAILABILITY

The Python scripts used to generate the results shown in Figs. 3–6 are openly available in the ZENODO repository at https://doi.org/10.5281/zenodo.7354077.

REFERENCES

[1] *Femtochemistry: With the Noble Lecture of A. Zewail*, edited by F. C. De Schryver, S. De Feyter, and G. Schweitzer (Wiley, Weinheim/Germany, 2001).

[2] V. May and O. Kühn, *Charge and Energy Transfer Dynamics in Molecular Systems* (Wiley, Berlin, 2000).

[3] D. Tannor, *Introduction to Quantum Mechanics: A Time-dependent Perspective* (University Science Books, Sausalito, 2004).

[4] F. Großmann, *Theoretical Femtosecond Physics* (Springer, Berlin Heidelberg, 2008).

[5] J. R. Johansson, P. D. Nation, and F. Nori, "QuTiP: An open-source Python framework for the dynamics of open quantum systems," Comput. Phys. Commun. **183**, 1760–1772 (2012).

[6] J. R. Johansson, P. D. Nation, and F. Nori, "QuTiP 2: A Python framework for the dynamics of open quantum systems," Comput. Phys. Commun. **184**, 1234–1240 (2013).

[7] A. Norambuena, D. Tancara, and R. Coto, "Coding closed and open quantum systems in MATLAB: Applications in quantum optics and condensed matter," Eur. J. Phys. **41**, 045404 (2020).

[8] A. V. Akimov, "Libra: An open-source 'methodology discovery' library for quantum and classical dynamics simulations," J. Comput. Chem. **37**, 1626–1649 (2016).

[9] B. Schmidt and U. Lorenz, "WavePacket: A Matlab package for numerical quantum dynamics. I: Closed quantum systems and discrete variable representations," Comput. Phys. Commun. **213**, 223–234 (2017).

[10] B. Schmidt and C. Hartmann, "WavePacket: A Matlab package for numerical quantum dynamics. II: Open quantum systems, optimal control, and model reduction," Comput. Phys. Commun. **228**, 229–244 (2018).

[11] B. Schmidt, R. Klein, and L. Cancissu Araujo, "WavePacket: A Matlab package for numerical quantum dynamics. III. Quantum–classical simulations and surface hopping trajectories," J. Comput. Chem. **40**, 2677–2688 (2019).

[12] J. J. Sørensen, J. H. M. Jensen, T. Heinzel, and J. F. Sherson, "QEngine: A C++ library for quantum optimal control of ultracold atoms," Comput. Phys. Commun. **243**, 135–150 (2019).

[13] M. Goerz, D. Basilewitsch, F. Gago-Encinas, M. G. Krauss, K. P. Horn, D. M. Reich, and C. Koch, "Krotov: A Python implementation of Krotov's method for quantum optimal control," SciPost Phys. **7**, 080 (2019).

[14] G. A. Worth, "Quantics: A general purpose package for quantum molecular dynamics simulations," Comput. Phys. Commun. **248**, 107040 (2020).

[15] M. Beck, A. Jäckle, G. A. Worth, and H.-D. Meyer, "The multiconfiguration time-dependent Hartree (MCTDH) method: A highly efficient algorithm for propagating wavepackets," Phys. Rep. **324**, 1–105 (2000).

[16] *Multidimensional Quantum Dynamics: MCTDH Theory and Applications*, edited by H. D. Meyer, F. Gatti, and G. A. Worth (Wiley VCH, 2009).

[17] M. Fishman, S. R. White, and E. M. Stoudenmire, "The ITensor software library for tensor network calculations," SciPost Phys. Codebases **2022**, 4.

[18] S. Paeckel, T. Köhler, A. Swoboda, S. R. Manmana, U. Schollwöck, and C. Hubig, "Time-evolution methods for matrix-product states," Ann. Phys. **411**, 167998 (2019).

[19] I. Affleck, T. Kennedy, E. H. Lieb, and H. Tasaki, "Rigorous results on valence-bond ground states in antiferromagnets," Phys. Rev. Lett. **59**, 799–802 (1987).

[20] I. V. Oseledets, "A new tensor decomposition," Dokl. Math. **80**, 495–496 (2009).

[21] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use SVD in many dimensions," SIAM J. Sci. Comput. **31**, 3744–3759 (2009).

[22] S. Klus, P. Gelß, S. Peitz, and C. Schütte, "Tensor-based dynamic mode decomposition," Nonlinearity **31**, 3359–3380 (2018).

[23] M. Lücke and F. Nüske, "tgEDMD: Approximation of the Kolmogorov operator in tensor train format," J. Nonlinear Sci. **32**, 44 (2022).

[24] P. Gelß, S. Klus, J. Eisert, and C. Schütte, "Multidimensional approximation of nonlinear dynamical systems," J. Comput. Nonlinear Dyn. **14**, 061006 (2019).

[25] A. Goeßmann, M. Götte, I. Roth, R. Sweke, G. Kutyniok, and J. Eisert, "Tensor network approaches for learning non-linear dynamical laws," arXiv:2002.12388 (2020).

[26] A. Veit and L. R. Scott, "Using the tensor-train approach to solve the ground-state eigenproblem for hydrogen molecules," SIAM J. Sci. Comput. **39**, B190–B220 (2017).

[27] P. Gelß, R. Klein, S. Matera, and B. Schmidt, "Solving the time-independent Schrödinger equation for chains of coupled excitons and phonons using tensor trains," J. Chem. Phys. **156**, 024109 (2022).

[28] P. Gelß, R. Klein, S. Matera, and B. Schmidt, "Quantum dynamics of coupled excitons and phonons in chain-like systems: Tensor train approaches and higher-order propagators," J. Chem. Phys. (submitted); arXiv:2302.03568 (2023).

[29] S. Klus and P. Gelß, "Tensor-based algorithms for image classification," Algorithms **12**, 240 (2019).

[30] W. Huggins, P. Patil, B. Mitchell, K. B. Whaley, and E. M. Stoudenmire, "Towards quantum machine learning with tensor networks," Quantum Sci. Technol. **4**, 024001 (2019).

[31] P. Gelß, S. Klus, S. Matera, and C. Schütte, "Nearest-neighbor interaction systems in the tensor-train format," J. Comput. Phys. **341**, 140–162 (2017).

[32] S. Mainali, F. Gatti, D. Iouchtchenko, P.-N. Roy, and H.-D. Meyer, "Comparison of the multi-layer multi-configuration time-dependent Hartree (ML-MCTDH) method and the density matrix renormalization group (DMRG) for ground state properties of linear rotor chains," J. Chem. Phys. **154**, 174106 (2021).

[33] T. Serwatka and P.-N. Roy, "Ground state of asymmetric tops with DMRG: Water in one dimension," J. Chem. Phys. **156**, 044116 (2022).

[34] Z. Hu, G. S. Engel, and S. Kais, "Connecting bright and dark states through accidental degeneracy caused by lack of symmetry," J. Chem. Phys. **148**, 204307 (2018).

[35] O. V. Mikhnenko, P. W. M. Blom, and T.-Q. Nguyen, "Exciton diffusion in organic semiconductors," Energy Environ. Sci. **8**, 1867–1888 (2015).

[36] M. Schröter, S. D. Ivanov, J. Schulze, S. P. Polyutov, Y. Yan, T. Pullerits, and O. Kühn, "Exciton-vibrational coupling in the dynamics and spectroscopy of Frenkel excitons in molecular aggregates," Phys. Rep. **567**, 1–78 (2015).

[37] A. Zhugayevych and S. Tretiak, "Theoretical description of structural and electronic properties of organic photovoltaic materials," Annu. Rev. Phys. Chem. **66**, 305–330 (2015).

[38] A. C. Scott, "Davydov's soliton revisited," Physica D **51**, 333–342 (1991).

[39] D. D. Georgiev and J. F. Glazebrook, "On the quantum dynamics of Davydov solitons in protein α-helices," Physica A **517**, 257–269 (2019).

[40] S. Holtz, T. Rohwedder, and R. Schneider, "The alternating linear scheme for tensor optimization in the tensor train format," SIAM J. Sci. Comput. **34**, A683–A713 (2012).

[41] A. Askar and A. S. Cakmak, "Explicit integration method for the time-dependent Schrodinger equation for collision problems," J. Chem. Phys. **68**, 2794 (1978).

[42] C. Leforestier, R. H. Bisseling, C. Cerjan, M. D. Feit, R. Friesner, A. Guldberg, A. Hammerich, G. Jolicard, W. Karrlein, H.-D. Meyer, N. Lipkin, O. Roncero, and R. Kosloff, "A comparison of different propagation schemes for the time dependent Schrödinger equation," J. Comput. Phys. **94**, 59–80 (1991).

[43] J. A. Fleck, J. R. Morris, and M. D. Feit, "Time-dependent propagation of high energy laser beams through the atmosphere," Appl. Phys. **10**, 129–160 (1976).

[44] M. D. Feit, J. A. Fleck, and A. Steiger, "Solution of the Schrödinger equation by a spectral method," J. Comput. Phys. **47**, 412–433 (1982).

[45] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," Ann. Phys. **349**, 117–158 (2014).

[46] C. Lubich, *From Quantum to Classical Molecular Dynamics: Reduced Models and Numerical Analysis* (European Mathematical Society, Zürich, 2008).

[47] I. V. Oseledets, "Tensor-train decomposition," SIAM J. Sci. Comput. **33**, 2295–2317 (2011).

[48] A. S. Davydov, *Solitons in Molecular Systems* (Reidel, 1985).

[49] V. M. Kenkre and S. M. Phatak, "Exact probability propagators for motion with arbitrary degree of transport coherence," Phys. Lett. A **100**, 101–104 (1984).

[50] F. Lenz, "The ratio of proton and electron masses," Phys. Rev. **82**, 554 (1951).

[51] R. Binder, D. Lauvergnat, and I. Burghardt, "Conformational dynamics guides coherent exciton migration in conjugated polymer materials: First-principles quantum dynamical study," Phys. Rev. Lett. **120**, 227401 (2018).

[52] F. Di Maiolo, D. Brey, R. Binder, and I. Burghardt, "Quantum dynamical simulations of intra-chain exciton diffusion in an oligo (*para*-phenylene vinylene) chain at finite temperature," J. Chem. Phys. **153**, 184107 (2020).

[53] F. A. Bornemann, P. Nettesheim, and C. Schütte, "Quantum–classical molecular dynamics as an approximation to full quantum dynamics," J. Chem. Phys. **105**, 1074 (1996).

[54] I. Burghardt, R. Carles, C. F. Kammerer, B. Lasorne, and C. Lasser, "Separation of scales: Dynamical approximations for composite quantum systems," J. Phys. A: Math. Theor. **54**, 414002 (2021).

[55] I. Burghardt, R. Carles, C. F. Kammerer, B. Lasorne, and C. Lasser, "Dynamical approximations for composite quantum systems: Assessment of error estimates for a separable ansatz," J. Phys. A: Math. Theor. **55**, 224010 (2022).

[56] P. Nettesheim, F. A. Bornemann, B. Schmidt, and C. Schütte, "An explicit and symplectic integrator for quantum–classical molecular dynamics," Chem. Phys. Lett. **256**, 581–588 (1996).

[57] S. Tomar, "Converting video formats with FFmpeg," Linux J. **2006**, 10.

[58] J. C. Light and T. Carrington, "Discrete-variable representations and their utilization," Adv. Chem. Phys. **114**, 263–310 (2000).

[59] J. T. Devreese and A. S. Alexandrov, "Fröhlich polaron and bipolaron: Recent developments," Rep. Prog. Phys. **72**, 066501 (2009).

[60] J. C. Tully, "Molecular dynamics with electronic transitions," J. Chem. Phys. **93**, 1061 (1990).

21 December 2023 10:07:42