

# Appendix A

## Programs

The sources reproduced here are written for Visual C++<sup>®</sup> 6.0<sup>1</sup>, Fortran 90 (fixed and free forms) and Matlab<sup>®</sup> 5<sup>2</sup>.

### A.1 Heterogeneous surface mean field model

The following Microsoft Visual C++ program “kinetik.exe” was originally written by Dr. Jens Hoffmann and largely rewritten by me. The sources reproduced here need to be compiled together with other semi-automatically generated files from the Microsoft Visual C++ development environment. All output from HoMF and HeMF were generated by this program. The output filenames generally consist in a two or three letter code describing the contents followed by the relevant parameters, i.e. “SSCT400x35d10.txt” contains the steady-state coverages calculated for  $T = 400$  K,  $x_{CO} = 0.35$  and  $\chi = 0.10$ .

```
1 // MEAN FIELD CO OXIDATION
2 // HOMOGENEOUS SURFACE MODEL
3
4 // kinetikDlg.h : Header-Datei
5 //
6
7 #if !defined(AFX_KINETIKDLG_H__4EDC3193_8978_11D4_8535_00B0D0159F30__INCLUDED_)
8 #define AFX_KINETIKDLG_H__4EDC3193_8978_11D4_8535_00B0D0159F30__INCLUDED_
9
10 #if _MSC_VER > 1000
11 #pragma once
12 #endif // _MSC_VER > 1000
13
14 //////////////////////////////////////
15 // CKinetikDlg Dialogfeld
16 #include "math.h"
17
18
19 // Global constants
20 const double k = 1.380658e-23;
21 const double Navo = 6.0221367e23;
22 const double R = Navo*k;
23 const double T02 = 300.0;
24 const double mo2 = 31.999e-3/Navo;
25 const double mco = 28.010e-3/Navo;
```

---

<sup>1</sup>Microsoft Visual C++, ©1994–98 Microsoft Corporation

<sup>2</sup>©1984–2002 The MathWorks, Inc.

```

26  const double   maxco = 0.5;
27  const double   maxo  = 0.25;
28  const double   M_PI  = 4.0*atan(1.0);
29
30  class CKinetikDlg : public CDialog
31  {
32  // Konstruktion
33  public:
34      CKinetikDlg(CWnd* pParent = NULL);    // Standard-Konstruktor
35      int  CKinetikDlg::Heterogeneous();
36      int  CKinetikDlg::Integration();
37      int  CKinetikDlg::Deriv( int n );
38      void CKinetikDlg::new_T( double d );
39      void CKinetikDlg::new_xco( double d );
40      void CKinetikDlg::HelpVariable();
41      void CKinetikDlg::FileHeader();
42      void CKinetikDlg::ErrorVerbose( int n );
43  // Dialogfelddaten
44      //{AFX_DATA(CKinetikDlg)
45      enum { IDD = IDD_KINETIK_DIALOG };
46      CButton   m_BtnHomogeneous;
47      CButton   m_BtnCoupling;
48      double    m_TempStart;
49      double    m_TempNumber;
50      double    m_TempStep;
51      double    m_partco0;
52      int       m_xcoStep;
53      CString   m_MeldungText;
54      int       m_after;
55      double    m_Npd;
56      double    m_Ptot;
57      double    m_vdes0;
58      double    m_vdes1;
59      double    m_vlh0;
60      double    m_vlh1;
61      int       m_beamon;
62      BOOL      m_bCoupling;
63      double    m_Edes0;
64      double    m_Edes1;
65      double    m_Elh0;
66      double    m_Elh1;
67      double    m_xcoexact;
68      double    m_Texact;
69      double    m_Codes;
70      double    m_tStep;
71      CString   m_Schritt;
72      double    m_Ct;
73      BOOL      m_bHomogeneous;
74      int       m_exp0ad;
75      double    m_sCO;
76      double    m_sO2;
77      double    m_Elhtheta;
78      BOOL      m_bTransient;
79      double    m_RDSStep;
80      int       m_resolution;
81      int       m_Ts;
82      //}}AFX_DATA
83
84      // Vom Klassenassistenten generierte berladungen virtueller Funktionen
85      //{AFX_VIRTUAL(CKinetikDlg)
86      protected:
87      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV-Unterstützung
88      //}}AFX_VIRTUAL
89
90  // Implementierung
91  protected:
92      HICON m_hIcon;

```

```

93
94 // Generierte Message-Map-Funktionen
95 //{{AFX_MSG(CKinetikDlg)
96 virtual BOOL OnInitDialog();
97 afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
98 afx_msg void OnPaint();
99 afx_msg HCURSOR OnQueryDragIcon();
100 afx_msg void OnChangeEDITAnzahl();
101 afx_msg void OnChangeEDITafter();
102 afx_msg void OnBtnAllTransient();
103 afx_msg void OnBtnSS();
104 afx_msg void OnBtnDataOneTransient();
105 afx_msg void OnChkCoupling();
106 afx_msg void OnBtnInit();
107 afx_msg void OnImgMinerva();
108 afx_msg void OnBtnFit();
109 afx_msg void OnChangeEditafter();
110 afx_msg void OnChkTransient();
111 afx_msg void OnBtnRDS();
112 afx_msg void OnBtnSSCov();
113 afx_msg void OnBtnRegion();
114 //}}AFX_MSG
115 DECLARE_MESSAGE_MAP()
116 };
117
118 //{{AFX_INSERT_LOCATION}}
119 // Microsoft Visual C++ fgt unmittelbar vor der vorhergehenden Zeile zustzliche Deklarationen ein.
120
121 #endif // !defined(AFX_KINETIKDLG_H__4EDC3193_8978_11D4_8535_00B0D0159F30__INCLUDED_)

```

```

1  /*****
2  **
3  **   MEAN FIELD CO OXIDATION BISTABILITY           **
4  **
5  **   HETEROGENEOUS SURFACE MODEL                 **
6  **
7  **   TRANSIENTS  STEADY-STATE  RAMPS  RATE-DETERMINING STEP  COVERAGES  **
8  **
9  **
10 **   Modified model (diffusion of oxygen)         **
11 **
12 **   Author:           JENS HOFFMANN               **
13 **   Continued by:    MATHIAS LAURIN              **
14 **
15 **   LAST MODIFICATION:  21.07.04   16:56:10      **
16 **
17 **
18 *****/

```

```

19
20 // kinetikDlg.cpp
21
22 #include "stdafx.h"
23 #include "kinetik.h"
24 #include "kinetikDlg.h"
25 #include "math.h"
26 #include "stdio.h"
27 #include "string.h"
28
29 #ifdef _DEBUG
30 #define new DEBUG_NEW
31 #undef THIS_FILE
32 static char THIS_FILE[] = __FILE__;
33 #endif
34
35
36 //////////////////////////////////////

```

```

37 // CAboutDlg-Dialogfeld fr Anwendungsbefehl "Info"
38
39 class CAboutDlg : public CDialog
40 {
41 public:
42     CAboutDlg();
43
44 // Dialogfelddaten
45     //{AFX_DATA(CAboutDlg)
46     enum { IDD = IDD_ABOUTBOX };
47     //}AFX_DATA
48 //     OnBtnSS();
49 //     OnBtnAllTransient();
50 //     OnBtnDataOneTransient();
51 // Vom Klassenassistenten generierte berladungen virtueller Funktionen
52     //{AFX_VIRTUAL(CAboutDlg)
53     protected:
54     virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV-Untersttzung
55     //}AFX_VIRTUAL
56
57 // Implementierung
58     protected:
59     //{AFX_MSG(CAboutDlg)
60     //}AFX_MSG
61     DECLARE_MESSAGE_MAP()
62 };
63
64 ////////////////////////////////////////////////////////////////////
65
66 CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
67 {
68     //{AFX_DATA_INIT(CAboutDlg)
69     //}AFX_DATA_INIT
70 }
71
72 ////////////////////////////////////////////////////////////////////
73 // CKinetikDlg Nachrichten-Handler
74
75 BOOL CKinetikDlg::OnInitDialog()
76 {
77     CDialog::OnInitDialog();
78
79     // IDM_ABOUTBOX muss sich im Bereich der Systembefehle befinden.
80
81
82     ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
83     ASSERT(IDM_ABOUTBOX < 0xF000);
84
85     CMenu* pSysMenu = GetSystemMenu( false );
86     if (pSysMenu != NULL) {
87         CString strAboutMenu;
88         strAboutMenu.LoadString(IDS_ABOUTBOX);
89         if (!strAboutMenu.IsEmpty()) {
90             pSysMenu->AppendMenu(MF_SEPARATOR);
91             pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
92         }
93     }
94
95
96     // Symbol fr dieses Dialogfeld festlegen. Wird automatisch erledigt
97     // wenn das Hauptfenster der Anwendung kein Dialogfeld ist
98     SetIcon(m_hIcon, true ); // Groes Symbol verwenden
99     SetIcon(m_hIcon, false ); // Kleines Symbol verwenden
100
101     // ZU ERLEDIGEN: Hier zustzliche Initialisierung einfgn
102
103     return true ; // Geben Sie true zurck, auer ein Steuerelement soll den Fokus erhalten

```

```

104 }
105
106 ///////////////////////////////////////////////////////////////////
107
108 void CKinetikDlg::OnSysCommand(UINT nID, LPARAM lParam)
109 {
110     if ((nID & 0xFFFF) == IDM_ABOUTBOX) {
111         CAboutDlg dlgAbout;
112         dlgAbout.DoModal();
113     }
114     else
115         CDialog::OnSysCommand(nID, lParam);
116 }
117
118 ///////////////////////////////////////////////////////////////////
119 //          CONSTANTS
120 ///////////////////////////////////////////////////////////////////
121
122 void CAboutDlg::DoDataExchange(CDataExchange* pDX)
123 {
124     CDialog::DoDataExchange(pDX);
125    //{{AFX_DATA_MAP(CAboutDlg)
126    //}}AFX_DATA_MAP
127 }
128
129 BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
130    //{{AFX_MSG_MAP(CAboutDlg)
131     // Keine Nachrichten-Handler
132    //}}AFX_MSG_MAP
133 END_MESSAGE_MAP()
134
135 ///////////////////////////////////////////////////////////////////
136 //          VARIABLES
137 ///////////////////////////////////////////////////////////////////
138
139 CKinetikDlg::CKinetikDlg(CWnd* pParent /*=NULL*/)
140     : CDialog(CKinetikDlg::IDD, pParent)
141 {
142     ////{{AFX_DATA_INIT(CKinetikDlg)
143     m_TempStart = 400.0;
144     m_TempNumber = 1.0;
145     m_TempStep = 15.0;
146     m_partco0 = 99;
147     m_xcoStep = 50;
148     m_after = 20;
149     m_Npd = 1.53e+19;
150     m_Ptot = 1.e-4;
151     m_vdes0 = 4.e+14;
152     m_vdes1 = 4.e+14;
153     m_vlh0 = 5.e+7;
154     m_vlh1 = 5.e+7;
155     m_beamon = 500;
156     m_bCoupling = true ;
157     m_Edes0 = 142.0;
158     m_Edes1 = 117.0;
159     m_Elh0 = 53.0;
160     m_Elh1 = 44.0;
161     m_xcoexact = 50.00;
162     m_Texact = 415;
163     m_Codes = 12;
164     m_tStep = 0.1;
165     m_Schritt = _T("0.1000");
166     m_Ct = 30;
167     m_exp0ad = 2;
168     m_sCO = 70.0;
169     m_sO2 = 100.0;
170     m_Elhtheta = 0.0;

```

```

171     m_bTransient = false ;
172     m_RDSStep = 1.0;
173     m_resolution = 10;
174     m_Ts = 0;
175     //}}AFX_DATA_INIT
176     // Beachten Sie, dass LoadIcon unter Win32 keinen nachfolgenden DestroyIcon-Aufruf bentigt
177     m_hIcon = AfxGetApp()->LoadIcon(IDI_Kinetik);
178 }
179
180 // Global variables
181 double     ko, kco, dt;
182 double     kdes[2], Edes[2], vdes[2];
183 double     klh[2], Elh[2], vlh[2];
184 double     co[2], o[2], d0[2], dCO[2];
185 double     prod[2], totprod, RTs, mmax;
186 double     p[2];
187 double     sO2, sCO, ElhTheta, COdes, Ct;
188 int        tmax, tafter;
189 double     eps;
190
191 FILE        *filename;
192
193 ////////////////////////////////////////////////////////////////////
194 //      PROCEDURES DEFINITION
195 ////////////////////////////////////////////////////////////////////
196
197 void CKinetikDlg::DoDataExchange(CDataExchange* pDX)
198 {
199     CDialog::DoDataExchange(pDX);
200     //{{AFX_DATA_MAP(CKinetikDlg)
201     DDX_Control(pDX, IDC_ChkCoupling, m_BtnCoupling);
202     DDX_Text(pDX, IDC_EDIT_TempStart, m_TempStart);
203     DDV_MinMaxDouble(pDX, m_TempStart, 300., 600.);
204     DDX_Text(pDX, IDC_EDIT_TempNumber, m_TempNumber);
205     DDV_MinMaxDouble(pDX, m_TempNumber, 1., 200.);
206     DDX_Text(pDX, IDC_EDIT_TempStep, m_TempStep);
207     DDV_MinMaxDouble(pDX, m_TempStep, 1., 50.);
208     DDX_Text(pDX, IDC_EDIT_partco1, m_partco0);
209     DDV_MinMaxDouble(pDX, m_partco0, 0., 100.);
210     DDX_Text(pDX, IDC_EDIT_xcoStep, m_xcoStep);
211     DDV_MinMaxInt(pDX, m_xcoStep, 1, 200);
212     DDX_Text(pDX, IDC_EDIT_after, m_after);
213     DDV_MinMaxInt(pDX, m_after, 1, 200);
214     DDX_Text(pDX, IDC_EDIT_Npd, m_Npd);
215     DDV_MinMaxDouble(pDX, m_Npd, 1.e+018, 3.e+019);
216     DDX_Text(pDX, IDC_EDIT_Ptot, m_Ptot);
217     DDV_MinMaxDouble(pDX, m_Ptot, 1.e-007, 1.);
218     DDX_Text(pDX, IDC_EDIT_vdes1, m_vdes0);
219     DDV_MinMaxDouble(pDX, m_vdes0, 1., 1.e+030);
220     DDX_Text(pDX, IDC_EDIT_vdes2, m_vdes1);
221     DDV_MinMaxDouble(pDX, m_vdes1, 1., 1.e+030);
222     DDX_Text(pDX, IDC_EDIT_vlh1, m_vlh0);
223     DDV_MinMaxDouble(pDX, m_vlh0, 1., 1.e+030);
224     DDX_Text(pDX, IDC_EDIT_vlh2, m_vlh1);
225     DDV_MinMaxDouble(pDX, m_vlh1, 1., 1.e+030);
226     DDX_Text(pDX, IDC_EDIT_beamon, m_beamon);
227     DDV_MinMaxInt(pDX, m_beamon, 1, 50000);
228     DDX_Check(pDX, IDC_ChkCoupling, m_bCoupling);
229     DDX_Text(pDX, IDC_EDIT_Edes1, m_Edes0);
230     DDV_MinMaxDouble(pDX, m_Edes0, 50., 250.);
231     DDX_Text(pDX, IDC_EDIT_Edes2, m_Edes1);
232     DDV_MinMaxDouble(pDX, m_Edes1, 50., 250.);
233     DDX_Text(pDX, IDC_EDIT_Elh1, m_Elh0);
234     DDV_MinMaxDouble(pDX, m_Elh0, 30., 250.);
235     DDX_Text(pDX, IDC_EDIT_Elh2, m_Elh1);
236     DDV_MinMaxDouble(pDX, m_Elh1, 30., 250.);
237     DDX_Text(pDX, IDC_EDIT_OneTransient_xco, m_xcoexact);

```

```

238     DDV_MinMaxDouble(pDX, m_xcoexact, 0., 100.);
239     DDX_Text(pDX, IDC_EDIT_OneTransient_T, m_Texact);
240     DDV_MinMaxDouble(pDX, m_Texact, 350., 550.);
241     DDX_Text(pDX, IDC_EDIT_C0des, m_Codes);
242     DDV_MinMaxDouble(pDX, m_Codes, 0., 100.);
243     DDX_Text(pDX, IDC_EDIT_dt, m_tStep);
244     DDV_MinMaxDouble(pDX, m_tStep, 1.0e-006, 1.0e-001);
245     DDX_Text(pDX, IDC_Schritt, m_Schritt);
246     DDX_Text(pDX, IDC_EDIT_Ct, m_Ct);
247     DDV_MinMaxDouble(pDX, m_Ct, 0., 100.);
248     DDX_Text(pDX, IDC_EDIT_exp0ad, m_exp0ad);
249     DDV_MinMaxInt(pDX, m_exp0ad, 1, 15);
250     DDX_Text(pDX, IDC_EDIT_sCO, m_sCO);
251     DDV_MinMaxDouble(pDX, m_sCO, 0., 1000.);
252     DDX_Text(pDX, IDC_EDIT_sO2, m_sO2);
253     DDV_MinMaxDouble(pDX, m_sO2, 0., 1000.);
254     DDX_Text(pDX, IDC_EDIT_Elhtheta, m_Elhtheta);
255     DDV_MinMaxDouble(pDX, m_Elhtheta, -100., 100.);
256     DDX_Check(pDX, IDC_ChkTransientFit, m_bTransient);
257     DDX_Text(pDX, IDC_EDIT_RDSStep, m_RDSStep);
258     DDV_MinMaxDouble(pDX, m_RDSStep, 0., 100.);
259     DDX_Text(pDX, IDC_EDIT_Resolution, m_resolution);
260     DDV_MinMaxInt(pDX, m_resolution, 1, 1000);
261     DDX_Text(pDX, IDC_EDIT_Ts, m_Ts);
262     //}}AFX_DATA_MAP
263 }
264
265 BEGIN_MESSAGE_MAP(CKinetikDlg, CDialog)
266    //{{AFX_MSG_MAP(CKinetikDlg)
267     ON_WM_SYSCOMMAND()
268     ON_WM_PAINT()
269     ON_WM_QUERYDRAGICON()
270     ON_EN_KILLFOCUS(IDC_EDIT_Ptot, OnChangeEDITafter)
271     ON_BN_CLICKED(IDC_BtnAllTransient, OnBtnAllTransient)
272     ON_BN_CLICKED(IDC_BtnSS, OnBtnSS)
273     ON_BN_CLICKED(IDC_BtnDataOneTransient, OnBtnDataOneTransient)
274     ON_BN_CLICKED(IDC_ChkCoupling, OnChkCoupling)
275     ON_BN_CLICKED(IDC_BtnInit, OnBtnInit)
276     ON_BN_CLICKED(IDC_BITMAP1, OnImgMinerva)
277     ON_BN_CLICKED(IDC_BtnFit, OnBtnFit)
278     ON_BN_CLICKED(IDC_ChkTransientFit, OnChkTransient)
279     ON_BN_CLICKED(IDC_BtnRDS, OnBtnRDS)
280     ON_BN_CLICKED(IDC_BtnSSCov, OnBtnSSCov)
281     ON_BN_CLICKED(IDC_BtnRegion, OnBtnRegion)
282     ON_EN_KILLFOCUS(IDC_EDIT_after, OnChangeEDITafter)
283     ON_EN_KILLFOCUS(IDC_EDIT_Edes1, OnChangeEDITafter)
284     ON_EN_KILLFOCUS(IDC_EDIT_Elh1, OnChangeEDITafter)
285     ON_EN_KILLFOCUS(IDC_EDIT_vdes1, OnChangeEDITafter)
286     ON_EN_KILLFOCUS(IDC_EDIT_vdes2, OnChangeEDITafter)
287     ON_EN_KILLFOCUS(IDC_EDIT_vlh2, OnChangeEDITafter)
288     ON_EN_KILLFOCUS(IDC_EDIT_beamon, OnChangeEDITafter)
289     ON_EN_KILLFOCUS(IDC_EDIT_vlh1, OnChangeEDITafter)
290     ON_EN_KILLFOCUS(IDC_EDIT_dt, OnChangeEDITafter)
291     ON_EN_KILLFOCUS(IDC_EDIT_Edes2, OnChangeEDITafter)
292     ON_EN_KILLFOCUS(IDC_EDIT_Elh2, OnChangeEDITafter)
293     ON_EN_KILLFOCUS(IDC_EDIT_Npd, OnChangeEDITafter)
294     ON_EN_KILLFOCUS(IDC_EDIT_partco1, OnChangeEDITafter)
295     ON_EN_KILLFOCUS(IDC_EDIT_TempStart, OnChangeEDITafter)
296     ON_EN_KILLFOCUS(IDC_EDIT_TempNumber, OnChangeEDITafter)
297     ON_EN_KILLFOCUS(IDC_EDIT_TempStep, OnChangeEDITafter)
298     ON_BN_CLICKED(IDC_ChkCoupling, OnChangeEDITafter)
299     ON_EN_KILLFOCUS(IDC_EDIT_OneTransient_T, OnChangeEDITafter)
300     ON_EN_KILLFOCUS(IDC_EDIT_OneTransient_xco, OnChangeEDITafter)
301     ON_EN_KILLFOCUS(IDC_EDIT_C0des, OnChangeEDITafter)
302     ON_EN_KILLFOCUS(IDC_EDIT_Ct, OnChangeEDITafter)
303     ON_EN_KILLFOCUS(IDC_EDIT_sCO, OnChangeEDITafter)
304     ON_EN_KILLFOCUS(IDC_EDIT_sO2, OnChangeEDITafter)

```

```

305     ON_EN_KILLFOCUS(IDC_EDIT_exp0ad, OnChangeEDITafter)
306     ON_EN_KILLFOCUS(IDC_EDIT_Eltheta, OnChangeEDITafter)
307     ON_EN_KILLFOCUS(IDC_EDIT_RDSStep, OnChangeEDITafter)
308     ON_EN_KILLFOCUS(IDC_EDIT_Resolution, OnChangeEDITafter)
309     //}}AFX_MSG_MAP
310 END_MESSAGE_MAP()
311
312
313 ///////////////////////////////////////////////////////////////////
314
315 void CKinetikDlg::OnPaint()
316 {
317     if (IsIconic()) {
318         CPaintDC dc(this); // Gertekontext fr Zeichnen
319
320         SendMessage(WM_ICONERASEBKGND, (LPARAM) dc.GetSafeHdc(), 0);
321
322         // Symbol in Client-Rechteck zentrieren
323         int cxIcon = GetSystemMetrics(SM_CXICON);
324         int cyIcon = GetSystemMetrics(SM_CYICON);
325         CRect rect;
326         GetClientRect(&rect);
327         int x = (rect.Width() - cxIcon + 1) / 2;
328         int y = (rect.Height() - cyIcon + 1) / 2;
329
330         // Symbol zeichnen
331         dc.DrawIcon(x, y, m_hIcon);
332     }
333     else
334         CDialog::OnPaint();
335 }
336
337
338 HCURSOR CKinetikDlg::OnQueryDragIcon()
339 { return (HCURSOR) m_hIcon; }
340
341
342 ///////////////////////////////////////////////////////////////////
343
344 void CKinetikDlg::OnImgMinerva()
345 {
346     MessageBox("- Fritz Haber Institut der MPG - \nAbpartung Chemische Physik\nFaradayweg 4-6\n14195 Berlin\n+49 (0)30 8413
347     /*
348     - Fritz Haber Institut der MPG -
349     Abpartung Chemische Physik
350     Faradayweg 4-6
351     14195 Berlin
352     +49 (0)30 8413 4309
353
354     http://www.fhi-berlin.mpg.de/cp/mb/mb.html
355     */
356 }
357
358
359 ///////////////////////////////////////////////////////////////////
360 // OUTPUT FILE HEADER
361 ///////////////////////////////////////////////////////////////////
362
363 void CKinetikDlg::FileHeader()
364 {
365     int temp;
366     fprintf(filename,"PARAMETER:\nEdes1: %.2f",m_Edes0);
367     fprintf(filename," kJ/mol\nvdes1: %e",m_vdes0);
368     fprintf(filename," s-1\nEdes2: %.2f",m_Edes1);
369     fprintf(filename," kJ/mol\nvdes2: %e",m_vdes1);
370     fprintf(filename,"s-1\nElh1: %.2f",m_Elh0);
371     fprintf(filename," kJ/mol\nvlh1: %e",m_vlh0);

```



```

372     fprintf(filename," s-1\nElh2: %.2f",m_Elh1);
373     fprintf(filename," kJ/mol\nvlh2: %e",m_vlh1);
374     fprintf(filename," s-1\nfraction Facet 1: %.2f",m_partco0);
375     fprintf(filename," percent\ncoverage Pd surface atoms: %e",m_Npd);
376     fprintf(filename," m-2\ntotal pressure: %.5f",m_Ptot);
377     fprintf(filename," Pa\ncoupling of facets\t");
378     if (m_bCoupling)
379         fprintf(filename,"on\n");
380     else
381         fprintf(filename,"off\n");
382     fprintf(filename,"\nExp. 0-Diss.: %.i",m_exp0ad);
383     fprintf(filename,"ninitial stick. coeff. CO: %.2f",m_sCO);
384     fprintf(filename," percent\ninitial stick. coeff. O: %.2f",m_sO2);
385     fprintf(filename," percent\nTheta dep. reaction: %.2f",m_Elhtheta);
386     fprintf(filename," percent\nTheta dep. CO-desorption: %.2f",m_Codes);
387     fprintf(filename," percent\nTheta dep. CO-adsorption: %.2f",m_Ct);
388     fprintf(filename," percent\nCO-fraction in %.2f",100.0/double(m_xcoStep));
389     fprintf(filename,"-percent steps\nTemperatures: ");
390     for (temp=0; temp<m_TempNumber; temp++) {
391         fprintf(filename,"%i", (temp)*m_TempStep+m_TempStart);
392         fprintf(filename,"K ");
393     }
394     fprintf(filename,"\nstep size numerical Integration: %.3f",dt);
395     fprintf(filename,"\n");
396 }
397
398
399 ////////////////////////////////////////////////////////////////////
400 // THINGS TO DO IF A EDIT FIELD IS LEFT
401 ////////////////////////////////////////////////////////////////////
402
403 void CKineticDlg::OnChangeEDITafter()
404 {
405     UpdateData( true );
406     m_Schritt.Format("%.3g",m_tStep);
407     SetDlgItemText(IDC_Schritt,m_Schritt);
408     HelpVariable();
409     UpdateData( false );
410 }
411
412
413 ////////////////////////////////////////////////////////////////////
414 // INITIALIZE ALL VARIABLES
415 ////////////////////////////////////////////////////////////////////
416
417 void CKineticDlg::OnBtnInit()
418 {
419     m_TempStart     = 400.0;
420     m_TempNumber= 1.0;
421     m_TempStep      = 15.0;
422     m_partco0      = 99;
423     m_xcoStep       = 50;
424     m_after         = 20;
425     m_Npd           = 1.53e+19;
426     m_Ptot          = 1.e-4;
427     m_vdes0         = 4.e+14;
428     m_vdes1         = 4.e+14;
429     m_vlh0          = 5.e+7;
430     m_vlh1          = 5.e+7;
431     m_beamon        = 500;
432     m_bCoupling     = true;
433     m_Edes0         = 142.0;
434     m_Edes1         = 117.0;
435     m_Elh0          = 53.0;
436     m_Elh1          = 44.0;
437     m_xcoexact      = 50.00;
438     m_Texact        = 415;

```

```

439     m_Codes          = 12;
440     m_tStep          = 0.1;
441     m_Schritt        = _T("0.1000");
442     m_Ct             = 30;
443     m_exp0ad         = 2;
444     m_sCO            = 70.0;
445     m_sO2            = 100.0;
446     m_Elhtheta       = 0.0;
447     m_bTransient     = false;
448     m_RDSStep        = 1.0;
449     m_resolution     = 10;
450
451     HelpVariable();
452     UpdateData( false );
453 }
454
455
456 ////////////////////////////////////////////////////
457 // CONVERSIONS TO S.I. UNIT //
458 ////////////////////////////////////////////////////
459
460 void CKinetikDlg::HelpVariable()
461 {
462     dt                = m_tStep;
463
464     vlh[0]            = m_vlh0;
465     Elh[0]            = m_Elh0 *1.0e3;
466     vdes[0]           = m_vdes0;
467     Edes[0]           = m_Edes0 *1.0e3;
468     sCO               = m_sCO *1.0e-2;
469     ElhTheta          = m_Elhtheta*1.0e-2;
470     COdes              = m_Codes *1.0e-2;
471     Ct                = m_Ct *1.0e-2;
472
473     vlh[1]            = m_vlh1;
474     Elh[1]            = m_Elh1 *1.0e3;
475     vdes[1]           = m_vdes1;
476     Edes[1]           = m_Edes1 *1.0e3;
477     eps               = exp( (Edes[1]-Edes[0])/RTs );
478
479     p[0]              = m_partco0 *1.0e-2;
480     p[1]              = 1.0-p[0];
481
482     mmax              = maxo*maxco;
483
484     tmax              = int(m_beamon/dt);
485     tafter            = int(m_after/dt);
486 }
487
488 ////////////////////////////////////////////////////
489
490 void CKinetikDlg::OnChkCoupling()
491 { m_bCoupling=~m_bCoupling; }
492
493 ////////////////////////////////////////////////////
494
495 void CKinetikDlg::OnChkTransient()
496 {
497     if (m_bTransient)
498         m_bTransient = false ;
499     else
500         m_bTransient = true ;
501     UpdateData( false );
502 }
503
504
505 /*****

```

```

506  **                                     **
507  **             INTERNAL PROCEDURES             **
508  **                                     **
509  *****/
510
511  //////////////////////////////////////
512  // INTEGRATION OF THE KINETIC EQUATIONS //
513  //////////////////////////////////////
514
515  // Driver for the integration
516  int CKinetikDlg::Integration()
517  {
518      int ErrCode;
519      if( eps == 0.0 ) HelpVariable();
520
521      // Facet 'Perfect'
522      if( p[0] > 0.9999 ) {
523          ErrCode = Deriv(0);
524          prod[0] = klh[0]*o[0]*co[0];
525          totprod = prod[0];
526      } else {
527          // Facet 'Defect'
528          Deriv(0);
529          Deriv(1);
530          ErrCode = Heterogeneous();
531          prod[0] = klh[0]*o[0]*co[0];
532          prod[1] = klh[1]*o[1]*co[1];
533          totprod = p[0]*prod[0] + p[1]*prod[1];
534      }
535      if( totprod < 0.0 ) ErrCode = 1;
536
537      return ErrCode;
538  }
539
540  // Derivation of the kinetic eq.
541  int CKinetikDlg::Deriv( int F )
542  {
543      double d0, dCO;
544      double Reac;
545      double dum;
546
547      //klh[F] = vlh[F] * exp( -Elh[F]*(1.0 - ElhTheta*co[F])/RTs );
548      klh[F] = vlh[F] * exp( -Elh[F]/RTs );
549      Reac = klh[F]*o[F]*co[F];
550
551      dum = 1.0-co[F]-o[F];
552      if( dum >= 0.0 )
553          dum = dum*dum; // dum = pow( dum, m_exp0ad );
554      else
555          dum = 0.0;
556
557      d0 = 2.0*ko/maxo*sO2*dum - Reac*maxco;
558      kdes[F] = vdes[F]*exp( -Edes[F]/RTs*( 1.0-COdes*co[F] ) );
559      dum = 1.0 - co[F] - Ct*o[F];
560      if( dum < 0.0 )
561          dum = 0.0;
562      dCO = sCO*dum*kco/maxco - kdes[F]*co[F] - Reac*maxo;
563
564      o[F] += d0*dt;
565      co[F] += dCO*dt;
566
567      return 0;
568  }
569
570  // thermodynamical equilibrium for the heterogeneous model
571  int CKinetikDlg::Heterogeneous()
572  {

```

```

573     double cotot, otot;
574     double a, b, c;
575     double meps = 1.0 - eps;
576
577     cotot   = p[0]*co[0] + p[1]*co[1];
578     otot    = p[0]*o[0]  + p[1]*o[1];
579
580     if( p[0] < 0.0001 ) { // strictly: exclude zero
581         co[1] = cotot;
582         o[1]  = otot;
583         co[0] = o[0] = 0.0;
584     } else if( p[1] < 0.0001 ) { // strictly: exclude zero
585         co[0] = cotot;
586         o[0]  = otot;
587         co[1] = o[1] = 0.0;
588     } else {
589         a   = 0.5/(p[0]*meps);
590         b   = cotot*meps + eps + p[0]*meps;
591         c   = -4.0*cotot*p[0]*meps;
592
593         c   = b*b + c;
594         if( c>=0.0 )
595             c = sqrt(c);
596         else
597             return 2;
598
599         co[0] = a*(b-c);
600         if( (co[0] < 0.0) || (co[0] > 1.0) )
601             co[0] = a*(b+c);
602
603         co[1]   = (cotot - p[0]*co[0])/p[1];
604
605         // Oxygen //
606         o[0]    = p[0]*otot;
607         o[1]    = p[1]*otot;
608
609         if( (co[1] < 0.0) || (co[0] < 0.0) ) return 4;
610         if( (o[1]  < 0.0) || (o[0]  < 0.0) ) return 8;
611         if( (o[1]  > 1.0) || (o[0]  > 1.0) ) return 8;
612     }
613     return 0;
614 }
615
616
617 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
618 //                               NEW T                               //
619 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
620
621 void CKinetikDlg::new_T(double Ts)
622 {
623     RTs = R*Ts;
624     sO2 = m_sO2*1.0e-2 - 7.4e-4*Ts;
625     m_Ts = int( Ts );
626     UpdateData( false );
627 }
628
629
630 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
631 //                               NEW x(CO)                               //
632 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
633
634 void CKinetikDlg::new_xco(double xco)
635 {
636     double Ftot;
637     double dDum;
638     dDum = 2.0*M_PI*k*T02;
639     Ftot = m_Ptot/( xco*sqrt(mco*dDum) + (1.0-xco)*sqrt(mo2*dDum) );

```

```

640     kco = xco*Ftot/m_Npd;
641     ko  = (1.0-xco)*Ftot/m_Npd;
642 }
643
644
645 ////////////////////////////////////////////////////////////////////
646 //  VERBOSE    ON ERROR                                ML 20/08/03      //
647 ////////////////////////////////////////////////////////////////////
648
649 void CKinetikDlg::ErrorVerbose( int ErrorCode )
650 {
651     switch( ErrorCode )
652     {
653     case 0:
654         MessageBox("Satisfactorily finished","Done!");
655         break;
656     case 1:
657         MessageBox("FAILED #1: Divergence - Decrease dt","Error",MB_ICONWARNING);
658         break;
659     case 2:
660         MessageBox("FAILED #2","Error",MB_ICONWARNING);
661         break;
662     case 4:
663         MessageBox("FAILED #4: Divergence - Decrease dt","Error",MB_ICONWARNING);
664         break;
665     case 8:
666         MessageBox("FAILED #8: Divergence - Decrease dt","Error",MB_ICONWARNING);
667         break;
668     case 64:
669         MessageBox("FAILED #64: eps = 0. - Retry","Error",MB_ICONWARNING);
670         break;
671     case 1024: // Silent out
672         break;
673     default:
674         MessageBox("FAILED: Unknown error","Error",MB_ICONWARNING);
675     }
676 }
677
678
679 /*****
680 **                                     **
681 **             BUTTONS                 **
682 **                                     **
683 *****/
684
685 ////////////////////////////////////////////////////////////////////
686 //             FIT ROUTINE              //
687 ////////////////////////////////////////////////////////////////////
688
689 void CKinetikDlg::OnBtnFit()
690 {
691     int     ii, jj;
692     int     nTime, n_T, n_xco, nDir;
693     int     ErrCode;
694     double  fp;
695     int     n1,n2,n3,n4,n5,n6,n7,n8,n9,n10;
696     int     nStep[11];
697     double  Max, diff;
698     double  xco, Ts;
699     double  StartVal[11],EndVal[11];
700     double  mittedata, mittess;
701     double  R2rel[2], R2abs[2];
702     double  sim[7][20][2];
703     double  data[7][20];
704     double  trans[4][1891];
705     FILE    *stream;
706

```

```

707 n1=n2=n3=n4=n5=n6=n7=n8=n9=n10=0; // to avoid error message during compilation
708
709 stream = fopen( "exptrans.txt", "r+" );
710 filename= fopen( "parameter.txt","w");
711
712 m_TempStart = 400.0;
713 m_TempStep = 15.0;
714 m_TempNumber= 2;
715 m_xcoStep = 20;
716 m_beamon = 500;
717 m_after = 1;
718 m_bCoupling = true;
719
720 UpdateData( false );
721
722 fprintf(filename,"vdes1\t vlh1\t Edes1\t Elh1\t exp0ad\t C0des\t Ct\t sCO\t sO2\t vdes2\t vlh2\t Edes2\t Elh[1]\t part1\n");
723 /*
724 vdes1 vlh1 Edes1 Elh1
725 exp0ad C0des Ct sCO sO2
726 vdes2 vlh2 Edes2 Elh2
727 part1 time scale R2relCO R2relO R2absCO R2absO
728 */
729 FileHeader();
730
731 /////////////// EXPERIMENTAL TRANSIENT DATA TAKEN FROM EXPTRANS.TXT ///////////////
732 for( ii = 1; ii <= 1890; ii++ ) {
733 fscanf( stream, "%f", &fp );
734 trans[1][ii]=fp;
735 fscanf( stream, "%f", &fp );
736 trans[2][ii]=fp;
737 }
738
739 /////////////// EXPERIMENTAL STEADY STATE DATA ///////////////
740
741 /// EBL particles
742 /*
743 data[1][2]=0.05;data[2][2]=0.0041;data[3][2]=0.0034;data[4][2]=0.0034;
744 data[1][3]=0.1;data[2][3]=0.0069;data[3][3]=0.006;data[4][3]=0.0059;
745 data[1][4]=0.15;data[2][4]=0.0091;data[3][4]=0.0084;data[4][4]=0.0082;
746 data[1][5]=0.20;data[2][5]=0.0125;data[3][5]=0.0115;data[4][5]=0.0107;
747 data[1][6]=0.25;data[2][6]=0.015;data[3][6]=0.014;data[4][6]=0.0134;
748 data[1][7]=0.3;data[2][7]=0.0175;data[3][7]=0.017;data[4][7]=0.0157;
749 data[1][8]=0.35;data[2][8]=0.021;data[3][8]=0.0198;data[4][8]=0.0182;
750 data[1][9]=0.4;data[2][9]=0.0235;data[3][9]=0.0229;data[4][9]=0.0215;
751 data[1][10]=0.45;data[2][10]=0.0259;data[3][10]=0.0263;data[4][10]=0.0241;
752 data[1][11]=0.5;data[2][11]=0.0082;data[3][11]=0.0289;data[4][11]=0.0277;
753 data[1][12]=0.55;data[2][12]=0.0052;data[3][12]=0.0166;data[4][12]=0.0301;
754 data[1][13]=0.6;data[2][13]=0.0037;data[3][13]=0.0087;data[4][13]=0.0206;
755 data[1][14]=0.65;data[2][14]=0.0033;data[3][14]=0.0071;data[4][14]=0.0143;
756 data[1][15]=0.7;data[2][15]=0.0027;data[3][15]=0.0059;data[4][15]=0.0114;
757 data[1][16]=0.75;data[2][16]=0.002;data[3][16]=0.0045;data[4][16]=0.0091;
758 data[1][17]=0.8;data[2][17]=0.0014;data[3][17]=0.0034;data[4][17]=0.0071;
759 data[1][18]=0.85;data[2][18]=0.00067;data[3][18]=0.0025;data[4][18]=0.0048;
760 data[1][19]=0.9;data[2][19]=0.00031;data[3][19]=0.0015;data[4][19]=0.0027;
761 data[1][20]=0.95;data[2][20]=0.00001;data[3][20]=0.00074;data[4][20]=0.00091;
762 //bistability//
763 data[1][22]=0.05;data[2][22]=0.0037;data[3][22]=0.0029;data[4][22]=0.0034;
764 data[1][23]=0.1;data[2][23]=0.0058;data[3][23]=0.0051;data[4][23]=0.0059;
765 data[1][24]=0.15;data[2][24]=0.008;data[3][24]=0.0082;data[4][24]=0.0082;
766 data[1][25]=0.20;data[2][25]=0.0109;data[3][25]=0.0111;data[4][25]=0.0107;
767 data[1][26]=0.25;data[2][26]=0.0136;data[3][26]=0.0139;data[4][26]=0.0134;
768 data[1][27]=0.3;data[2][27]=0.0157;data[3][27]=0.0171;data[4][27]=0.0157;
769 data[1][28]=0.35;data[2][28]=0.0172;data[3][28]=0.02;data[4][28]=0.0182;
770 data[1][29]=0.4;data[2][29]=0.0074;data[3][29]=0.0231;data[4][29]=0.0215;
771 data[1][30]=0.45;data[2][30]=0.0057;data[3][30]=0.0197;data[4][30]=0.0241;
772 data[1][31]=0.5;data[2][31]=0.0044;data[3][31]=0.0109;data[4][31]=0.0277;
773 data[1][32]=0.55;data[2][32]=0.0037;data[3][32]=0.0084;data[4][32]=0.0301;

```

```

774 data[1][33]=0.6;data[2][33]=0.0031;data[3][33]=0.0071;data[4][33]=0.0206;
775 data[1][34]=0.65;data[2][34]=0.0026;data[3][34]=0.0059;data[4][34]=0.0143;
776 data[1][35]=0.7;data[2][35]=0.0022;data[3][35]=0.0046;data[4][35]=0.0114;
777 data[1][36]=0.75;data[2][36]=0.0016;data[3][36]=0.0036;data[4][36]=0.0091;
778 data[1][37]=0.8;data[2][37]=0.0015;data[3][37]=0.0031;data[4][37]=0.0071;
779 data[1][38]=0.85;data[2][38]=0.00088;data[3][38]=0.0021;data[4][38]=0.0048;
780 data[1][39]=0.9;data[2][39]=0.00051;data[3][39]=0.0012;data[4][39]=0.0027;
781 data[1][40]=0.95;data[2][40]=0.00023;data[3][40]=0.00059;data[4][40]=0.00091;
782
783 mittedata=0.0259+0.0289+0.0301;//0.0259+0.0289+0.0301;*/
784
785 /// big particles (6 nm)
786
787 data[1][2]=0.05;data[2][2]=0.0058;data[3][2]=0.0056;data[4][2]=0.0062;
788 data[1][3]=0.1;data[2][3]=0.0101;data[3][3]=0.0108;data[4][3]=0.0097;
789 data[1][4]=0.15;data[2][4]=0.0104;data[3][4]=0.0158;data[4][4]=0.0143;
790 data[1][5]=0.20;data[2][5]=0.006;data[3][5]=0.0196;data[4][5]=0.0197;
791 data[1][6]=0.25;data[2][6]=0.0042;data[3][6]=0.0137;data[4][6]=0.0241;
792 data[1][7]=0.3;data[2][7]=0.0034;data[3][7]=0.0084;data[4][7]=0.0292;
793 data[1][8]=0.35;data[2][8]=0.003;data[3][8]=0.0066;data[4][8]=0.0337;
794 data[1][9]=0.4;data[2][9]=0.0024;data[3][9]=0.0051;data[4][9]=0.0351;
795 data[1][10]=0.45;data[2][10]=0.0021;data[3][10]=0.0045;data[4][10]=0.0237;
796 data[1][11]=0.5;data[2][11]=0.0019;data[3][11]=0.0037;data[4][11]=0.0152;
797 data[1][12]=0.55;data[2][12]=0.0018;data[3][12]=0.003;data[4][12]=0.0122;
798 data[1][13]=0.6;data[2][13]=0.0015;data[3][13]=0.0028;data[4][13]=0.0101;
799 data[1][14]=0.65;data[2][14]=0.0013;data[3][14]=0.0021;data[4][14]=0.0084;
800 data[1][15]=0.7;data[2][15]=0.0011;data[3][15]=0.0018;data[4][15]=0.0071;
801 data[1][16]=0.75;data[2][16]=0.00079;data[3][16]=0.0016;data[4][16]=0.0054;
802 data[1][17]=0.8;data[2][17]=0.00088;data[3][17]=0.0012;data[4][17]=0.0039;
803 data[1][18]=0.85;data[2][18]=0.00063;data[3][18]=0.00085;data[4][18]=0.003;
804 data[1][19]=0.9;data[2][19]=0.00039;data[3][19]=0.0005;data[4][19]=0.0021;
805 data[1][20]=0.95;data[2][20]=0.00016;data[3][20]=0.0002;data[4][20]=0.00078;
806 //bistability//
807 data[1][22]=0.05;data[2][22]=0.0049;data[3][22]=0.0045;data[4][22]=0.0053;
808 data[1][23]=0.1;data[2][23]=0.0094;data[3][23]=0.0105;data[4][23]=0.0098;
809 data[1][24]=0.15;data[2][24]=0.0121;data[3][24]=0.0155;data[4][24]=0.014;
810 data[1][25]=0.20;data[2][25]=0.0101;data[3][25]=0.0195;data[4][25]=0.0188;
811 data[1][26]=0.25;data[2][26]=0.0067;data[3][26]=0.0192;data[4][26]=0.0239;
812 data[1][27]=0.3;data[2][27]=0.0043;data[3][27]=0.0138;data[4][27]=0.0285;
813 data[1][28]=0.35;data[2][28]=0.0033;data[3][28]=0.0092;data[4][28]=0.0337;
814 data[1][29]=0.4;data[2][29]=0.0027;data[3][29]=0.007;data[4][29]=0.036;
815 data[1][30]=0.45;data[2][30]=0.0023;data[3][30]=0.0051;data[4][30]=0.0286;
816 data[1][31]=0.5;data[2][31]=0.0018;data[3][31]=0.0041;data[4][31]=0.0168;
817 data[1][32]=0.55;data[2][32]=0.0019;data[3][32]=0.0035;data[4][32]=0.0123;
818 data[1][33]=0.6;data[2][33]=0.0015;data[3][33]=0.0025;data[4][33]=0.01;
819 data[1][34]=0.65;data[2][34]=0.0014;data[3][34]=0.0023;data[4][34]=0.0084;
820 data[1][35]=0.7;data[2][35]=0.0011;data[3][35]=0.0019;data[4][35]=0.0068;
821 data[1][36]=0.75;data[2][36]=0.00094;data[3][36]=0.0015;data[4][36]=0.0057;
822 data[1][37]=0.8;data[2][37]=0.00093;data[3][37]=0.0014;data[4][37]=0.0041;
823 data[1][38]=0.85;data[2][38]=0.00055;data[3][38]=0.00082;data[4][38]=0.003;
824 data[1][39]=0.9;data[2][39]=0.00036;data[3][39]=0.00046;data[4][39]=0.0017;
825 data[1][40]=0.95;data[2][40]=0.00082;data[3][40]=0.00015;data[4][40]=0.00092;
826
827 mittedata=0.0104+0.0196;//+0.0351;//0.0104+0.0196+0.0351;
828
829 /* m_vdes0 */ StartVal[1] = 0.0; EndVal[1] = 10.0; nStep[1] = 11;
830 /* m_vlh0 */ StartVal[2] = 0.0; EndVal[2] = 5.0; nStep[2] = 6;
831 /* m_Edes1 */ StartVal[3] = 120.0; EndVal[3] = 140.0; nStep[3] = 6;
832 /* m_Elh1 */ StartVal[4] = 50.0; EndVal[4] = 70.0; nStep[4] = 6;
833 /* m_exp0ad */ StartVal[5] = 2.0; EndVal[5] = 10.0; nStep[5] = 2;
834 /* m_partco0 */ StartVal[6] = 70.0; EndVal[6] = 90.0; nStep[6] = 3;
835 /* m_sCD */ StartVal[7] = 80.0; EndVal[7] = 100.0; nStep[7] = 5;
836 /* m_sO2 */ StartVal[8] = 40.0; EndVal[8] = 80.0; nStep[8] = 9;
837 /* m_Codes */ StartVal[9] = 16.0; EndVal[9] = 20.0; nStep[9] = 5;
838 /* m_Elthetha */ StartVal[10]= -20.0; EndVal[10]= 0.0; nStep[10]= 3;
839
840 /*j = 0;

```

```

841     for( i=1;i<=10;i++ ) {
842         if( j < nStep[i] ) j=nStep[i];
843     }*/
844
845     double ValParam[11][1003];
846
847     for( ii = 1; ii <= 10; ii++) {
848         for( jj = 1; jj <= nStep[ii]; jj++) {
849             ValParam[ii][jj]=StartVal[ii]+(jj-1)*(EndVal[ii]-StartVal[ii])/(nStep[ii]-1);
850             if( ii == 1 ) ValParam[1][jj] = double(1.0e+11*pow(4,ValParam[1][jj]));
851             if( ii == 2 ) ValParam[2][jj] = double(1.0e+7*pow(2,ValParam[2][jj]));
852         }
853     }
854
855     // MAIN LOOP
856     /*for( n1=1;n1<=nStep[1];n1++) {
857         m_vdes0=ValParam[1][n1];
858         for( n2=1;n2<=nStep[2];n2++) {
859             m_vlh0=ValParam[2][n2];*/
860         for( n3=1;n3<=nStep[3];n3++) {
861             m_Edes1=ValParam[3][n3];
862             for( n4=1;n4<=nStep[4];n4++) {
863                 m_Elh1=ValParam[4][n4];
864             /*for( n5=1;n5<=nStep[5];n5++) {
865                 m_exp0ad=int(ValParam[5][n5]);*/
866             for( n6=1;n6<=nStep[6];n6++) {
867                 m_partco0=ValParam[6][n6];
868             for( n7=1;n7<=nStep[7];n7++) {
869                 m_sC0=ValParam[7][n7];
870             for( n8=1;n8<=nStep[8];n8++) {
871                 m_sO2=ValParam[8][n8];
872             /* for( n9=1;n9<=nStep[9];n9++) {
873                 m_Codes=ValParam[9][n9];*/
874             /*for( n10=1;n10<=nStep[10];n10++) {
875                 m_Eltheta=ValParam[10][n10];*/
876
877             mittess = 0.0;
878
879             // TEMPER LOOP
880             for( n_T = 1; n_T <= m_TempNumber; n_T++ ) {
881                 Max = 0.0;
882                 Ts = m_TempStart + n_T*m_TempStep;
883                 new_T( Ts );
884
885                 for( nDir = 0; nDir <= 1; nDir++ ) {
886                     for( n_xco = 1; n_xco <= m_xcoStep; n_xco++ ) {
887                         if( nDir == 0 ) {
888                             co[0]=co[1]=0.0; o[0]=o[1]=1.0;
889                         } else {
890                             co[0]=co[1]=1.0; o[0]=o[1]=0.0;
891                         }
892                         xco = double(n_xco)/double(m_xcoStep);
893                         new_xco( xco );
894                         for( nTime=1; nTime<tmax; nTime++ ) {
895                             ErrCode = Integration();
896                             if( ErrCode != 0 ) goto Error;
897                         }
898                         sim[n_T][n_xco][nDir]=totprod*mmax;
899                     }
900                 }
901                 if( sim[n_T][n_xco][nDir] > Max ) Max = sim[n_T][n_xco][nDir];
902             } // END TEMPER LOOP
903
904
905             /////////////// COMPARISON EXPERIMENT - SIMULATION ////////////////////////
906
907             // NORMALISATION

```



```

908     for( n_xco = 1; n_xco <= m_xcoStep; n_xco++) {
909         for( n_T = 1; n_T <= m_TempNumber; n_T++ ) {
910             for( nDir = 0; nDir <= 1; nDir++ ) {
911                 sim[n_T][n_xco][nDir] = sim[n_T][n_xco][nDir]/mittess*mittedata;
912             }}}
913
914     // CALC. ERRORS
915     R2abs[1] = R2abs[2] = 0.0;
916     R2rel[1] = R2rel[2] = 0.0;
917
918     for( n_xco=1; n_xco <= m_xcoStep; n_xco++ ) {
919         for( n_T = 1; n_T <= m_TempNumber; n_T++ ) {
920             for( nDir=0; nDir <= 1; nDir++ ) {
921                 diff = sim[n_T][n_xco][nDir]-data[n_T+1][n_xco+1];
922                 R2abs[nDir] = R2abs[nDir] + diff*diff;
923                 R2rel[nDir] = R2rel[nDir] + (diff/data[n_T+1][n_xco+1])*(diff/data[n_T+1][n_xco+1]);
924             }}}
925
926     // OUTPUT
927     fprintf(filename, "\n%e", m_vdes0);
928     fprintf(filename, "\t%e", m_vlh0);
929     fprintf(filename, "\t%e", m_Edes0);
930     fprintf(filename, "\t%e", m_Elh0);
931     fprintf(filename, "\t%i", m_exp0ad);
932     fprintf(filename, "\t%e", m_Codes);
933     fprintf(filename, "\t%e", m_Ct);
934     fprintf(filename, "\t%e", m_sCO);
935     fprintf(filename, "\t%e", m_sO2);
936     fprintf(filename, "\t%e", m_vdes1);
937     fprintf(filename, "\t%e", m_vlh1);
938     fprintf(filename, "\t%e", m_Edes1);
939     fprintf(filename, "\t%e", m_Elh1);
940     fprintf(filename, "\t%e", m_partco0);
941     //fprintf(filename, "\t%e", m_Elhtheta);
942     fprintf(filename, "\t%e", time);
943     if( mittedata/mittess > 0 )
944         fprintf(filename, "\t%e", mittedata/mittess);
945     else
946         fprintf(filename, "\t%e", mittedata);
947     fprintf(filename, "\t%e", R2rel[1]);
948     fprintf(filename, "\t%e", R2rel[2]);
949     fprintf(filename, "\t%e", R2abs[1]);
950     fprintf(filename, "\t%e", R2abs[2]);
951
952     } //m_vdes0
953     } //m_vlh0
954     } //m_Edes1
955     } //m_Elh1
956     } //m_exp0ad
957     //} //m_partco0
958     //} //m_sCO
959     //} //m_sO2
960     //} //m_Codes
961     //} //m_ElhTheta
962     // END MAIN LOOP
963
964     Error:
965     fclose(filename);
966     UpdateData( false );
967
968     ErrorVerbose( ErrCode );
969 }
970
971
972 ///////////////////////////////////////////////////////////////////
973 //          RATE DETERMINING STEP          ML 19.07.04 //
974 //          w/out bistability                //

```

```

975 ///////////////////////////////////////////////////////////////////
976
977 void CKinetikDlg::OnBtnRDS()
978 {
979     int         n_xco, n_T, nTime, n;
980     int         ErrCode;
981     double      Ts, xco;
982     double      PctInc;
983     double      RateIni;
984     double      oini=0.0, coini=1.0;
985     double      drc[4];          // [JCatal 204 (2001) 520]
986     double      sav1, sav2;
987     FILE        * pFile[4];
988
989     PctInc = m_RDSStep * 1.0e-2;
990
991     // open files
992     pFile[0] = fopen("drc_info.txt", "w");
993     fprintf( pFile[0], "TStart %e TStep %e TNumber %e \n", m_TempStart, m_TempStep, m_TempNumber);
994     fprintf( pFile[0], "xCO %i, xCO step %e \n", m_xcoStep, 1.0/double(m_xcoStep));
995     fclose( pFile[0] );
996
997     pFile[0] = fopen("drc_ini.txt", "w");
998     pFile[1] = fopen("drc_CO.txt", "w");
999     pFile[2] = fopen("drc_O2.txt", "w");
1000    pFile[3] = fopen("drc_LH.txt", "w");
1001
1002    for( n_T=0; n_T<=m_TempNumber; n_T++ ) {
1003        Ts = m_TempStart + double(n_T)*m_TempStep;
1004        new_T( Ts );
1005
1006        for( n_xco=1; n_xco < m_xcoStep; n_xco++ ) {
1007            xco = double( n_xco )/double( m_xcoStep );
1008            new_xco( xco );
1009
1010            // 0. Initial rate
1011            co[0]=co[1]=coini; o[0]=o[1]=oini;
1012            for( nTime=1; nTime <= tmax; nTime++ ) {
1013                ErrCode = Integration();
1014                if( ErrCode != 0 ) goto Error;
1015            }
1016            RateIni = drc[0] = totprod;
1017
1018            //1. adsorption desorption CO
1019            sav1 = kco; sav2 = vdes[0];
1020            kco = kco + kco*PctInc;
1021            vdes[0] = vdes[0] + vdes[0]*PctInc;
1022            co[0]=co[1]=0.0; o[0]=o[1]=1.0;
1023            for( nTime=1; nTime <= tmax; nTime++ ) {
1024                ErrCode = Integration();
1025                if( ErrCode != 0 ) goto Error;
1026            }
1027            kco = sav1; vdes[0] = sav2;
1028            drc[1] = (totprod-RateIni)/(RateIni*PctInc);
1029
1030            //2. adsorption O2
1031            sav1 = ko;
1032            ko = ko + ko*PctInc;
1033            co[0]=co[1]=coini; o[0]=o[1]=oini;
1034            for( nTime=1; nTime <= tmax; nTime++ ) {
1035                ErrCode = Integration();
1036                if( ErrCode != 0 ) goto Error;
1037            }
1038            ko = sav1;
1039            drc[2] = (totprod-RateIni)/(RateIni*PctInc);
1040
1041            //3. reaction

```

```

1042         sav1 = vlh[0];
1043         vlh[0] = vlh[0] + vlh[0]*PctInc;
1044         co[0]=co[1]=coini; o[0]=o[1]=oini;
1045         for( nTime=1; nTime <= tmax; nTime++ ) {
1046             ErrCode = Integration();
1047             if( ErrCode != 0 ) goto Error;
1048         }
1049         vlh[0] = sav1;
1050         drc[3] = (totprod-RateIni)/(RateIni*PctInc);
1051
1052         // OUTPUT
1053         for( n=0; n<4; n++ )
1054             fprintf( pFile[n], "%e\t", drc[n]);
1055     } // xCO loop
1056
1057     for( n=0; n<4; n++ )
1058         fprintf( pFile[n], "\n");
1059 } // Temperature loop
1060
1061 Error:
1062     for( n=0; n<4; n++ ) fclose( pFile[n] );
1063     ErrorVerbose( ErrCode );
1064 }
1065
1066
1067 ////////////////////////////////////////////////////////////////////
1068 // ALL TRANSIENTS //
1069 ////////////////////////////////////////////////////////////////////
1070 void CKinetikDlg::OnBtnAllTransient()
1071 {
1072     OnBtnDataOneTransient();
1073 }
1074
1075
1076 ////////////////////////////////////////////////////////////////////
1077 // ONE TRANSIENT ML 18/09/03 //
1078 ////////////////////////////////////////////////////////////////////
1079
1080 void CKinetikDlg::OnBtnDataOneTransient()
1081 {
1082     int nTime, nDir;
1083     int ErrCode;
1084     double kosav, kcosav;
1085     double time, Ts, xco;
1086     char transf[20];
1087     int nStep;
1088
1089
1090     Ts = m_Texact;
1091     new_T( Ts );
1092     xco = m_xcoexact/100.0;
1093     new_xco( xco );
1094
1095     kosav = ko;
1096     kcosav = kco;
1097
1098     sprintf( transf, "TrsT%.3ix%.2id%.2i.txt", int(Ts), int(xco*100.0), int(100-m_partco0) );
1099     filename= fopen( transf,"w" );
1100     fprintf(filename, "\ntime\tCO2\tNtot");
1101     FileHeader();
1102     fprintf(filename, "\nHERE:\nCO-fraction: %.2f", xco*100.0);
1103     fprintf(filename, " percent\nTemperature: %i", Ts);
1104     //fprintf(filename, "\tFac.1CO2\tN1\tFac.2CO2\tN2\tFac.1CO\tNco1\tFac.2CO\tNco2\tFac.1.0\tNo1\tFac.20\tNo2\tCO1->2\tO1->2");
1105
1106     for( nDir = 0; nDir <= 1; nDir++ ) {
1107         ko = kosav;
1108         kco = kcosav;

```

```

1109     time = 0.0;
1110     nStep = 0;
1111
1112     fprintf( filename, "\n" );
1113     if( nDir == 0 ) {
1114         co[0]=co[1]=0.0; o[0]=o[1]=1.0;
1115     } else {
1116         co[0]=co[1]=1.0; o[0]=o[1]=0.0;
1117     }
1118
1119     for( nTime = 1; nTime <= tmax+tafter; nTime++ ) {
1120         time = time + dt;
1121         ++nStep;
1122
1123         if( nTime > tmax ) {
1124             kco = 0.0;
1125             ko = 0.0;
1126         }
1127
1128         ErrCode = Integration();
1129         if( ErrCode != 0 ) goto Error;
1130
1131         if( nStep == int(1.0/dt) ) {
1132             nStep = 0;
1133             fprintf( filename, "\n%e", time );
1134             fprintf( filename, "\t%e", totprod*mmax );
1135             fprintf( filename, "\t%e", totprod*mmax*m_Npd );
1136             /*fprintf( filename, "\t%e", prod[0]*mmax );
1137             fprintf( filename, "\t%e", prod[0]*mmax*m_Npd );
1138             fprintf( filename, "\t%e", prod[1]*mmax );
1139             fprintf( filename, "\t%e", prod[1]*mmax*m_Npd );
1140             fprintf( filename, "\t%e", co[0] );
1141             fprintf( filename, "\t%e", co[0]*maxco*m_Npd );
1142             fprintf( filename, "\t%e", co[1] );
1143             fprintf( filename, "\t%e", co[1]*maxco*m_Npd );
1144             fprintf( filename, "\t%e", o[0] );
1145             fprintf( filename, "\t%e", o[0]*maxo*m_Npd );
1146             fprintf( filename, "\t%e", o[1] );
1147             fprintf( filename, "\t%e", o[1]*maxo*m_Npd );*/
1148         }
1149     }
1150 }
1151
1152 Error:
1153     fclose(filename);
1154
1155     if( eps == 0.0 ) ErrCode = 64;
1156     if( ErrCode == 0 ) ErrCode = 1024;
1157     ErrorVerbose( ErrCode );
1158 }
1159
1160
1161 ////////////////////////////////////////////////////////////////////
1162 // STEADY STATES INCLUDING COVERAGES ML 17/09/2003 //
1163 ////////////////////////////////////////////////////////////////////
1164
1165 void CKinetikDlg::OnBtnSS()
1166 {
1167     int nTime, n_xco, nDir;
1168     int ErrCode;
1169     double Ts, xco;
1170     char ssf[20];
1171
1172     Ts = m_TempStart;
1173     new_T( Ts );
1174
1175     sprintf( ssf, "SST%.3id%.2i.txt", int(Ts), int(100-m_partco0) );

```

```

1176     filename = fopen(ssf,"w");
1177     fprintf(filename,"\nxc0\tCO2\nT=%.0f\n", Ts);
1178     FileHeader();
1179
1180     for( nDir=0; nDir <= 1; nDir++ ) {
1181         fprintf(filename,"\n");
1182
1183         for (n_xco=1; n_xco<=m_xcoStep-1; n_xco++) {
1184             xco = double(n_xco)/double(m_xcoStep);
1185             new_xco( xco );
1186
1187             // INTEGRATIONS
1188             if( nDir == 0 ) {
1189                 co[0]=co[1]=1.0; o[0]=o[1]=0.0;
1190             } else {
1191                 co[0]=co[1]=0.0; o[0]=o[1]=1.0;
1192             }
1193
1194             for( nTime = 1; nTime <= tmax; nTime++) {
1195                 ErrCode = Integration();
1196                 if( ErrCode != 0 ) goto Error;
1197             }
1198
1199             // OUTPUT
1200             fprintf(filename, "\n%e", xco);
1201             fprintf(filename, "\t%e", totprod*mmax);
1202         }
1203     }
1204
1205 Error:
1206     fclose(filename);
1207
1208     if( eps == 0.0 ) ErrCode = 64;
1209     ErrorVerbose( ErrCode );
1210 }
1211
1212
1213 /*****
1214 **                                     **
1215 **             Bistability             **
1216 **                                     **
1217 *****/
1218
1219 ////////////////////////////////////////////////////////////////////
1220 // Steady State Coverages :                               ML 18/09/2003 //
1221 // starting with different coverages                       //
1222 ////////////////////////////////////////////////////////////////////
1223
1224 void CKinetikDlg::OnBtnSSCov()
1225 {
1226     char    SSCf[20];
1227     int     no, nco, nTime;
1228     double  Ts, xco;
1229     int     ErrCode;
1230     double  dResolution;
1231
1232     HelpVariable();
1233
1234     dResolution = double(m_resolution);
1235
1236     xco        = m_xcoexact/100.0;
1237     new_xco( xco );
1238     Ts         = m_Texact;
1239     new_T( Ts );
1240
1241     sprintf( SSCf, "SSCT%.3ix%.2id%.2i.txt", int(Ts), int(xco*100.0), int(100-m_partco0) );
1242     filename = fopen(SSCf,"w");

```

```

1243     fprintf(filename,"CO\t 0\t co[1]\nSteady State values\n");
1244     FileHeader();
1245     fprintf(filename,"file format:\n1st column: starting value theta(CO) \n2nd: starting value theta(0)\n3rd: steady state t
1246
1247     /* file format:
1248     1st: steady state theta(CO)
1249     2nd: steady state theta(0)
1250     3th: steady state co[1] production
1251     */
1252
1253     for( nco=0; nco <= m_resolution; nco++ ) {
1254         for( no=0; no <= m_resolution-nco; no++ ) {
1255             co[0] = co[1] = double(nco)/dResolution;
1256             o[0] = o[1] = double(no) /dResolution;
1257
1258             // INTEGRATION
1259             for( nTime = 0; nTime <= tmax; nTime++ ) {
1260                 ErrCode = Integration();
1261                 if( ErrCode != 0 ) goto Error;
1262             }
1263
1264             // OUTPUT
1265             fprintf( filename,"\n%e", p[0]*co[0] + p[1]*co[1] );
1266             fprintf( filename,"\t%e", p[0]*o[0] + p[1]*o[1] );
1267             fprintf( filename,"\t%e", totprod*mmax );
1268         }
1269     }
1270
1271 Error:
1272     fclose(filename);
1273     if( eps == 0.0 ) ErrCode = 64;
1274
1275     ErrorVerbose( ErrCode );
1276 }
1277
1278
1279 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
1280 //      bistabiliy region                                     ML 17/09/03      //
1281 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
1282
1283 void CKinetikDlg::OnBtnRegion()
1284 {
1285     int      n_xco, n_T, nTime, nDir;
1286     int      ErrCode;
1287     double   CO2prod[2], Ts, xco;
1288
1289     FILE     *bro;
1290     char     brf[25];
1291
1292     sprintf( brf, "bist_regd%.2i.txt", int(100-m_partco0) );
1293     bro     = fopen(brf,"w");
1294
1295     // FileHeader(); needs declaration of FILE *filename
1296     fprintf(bro,"xco\t Temperature\n");
1297
1298     for( n_xco=1; n_xco <= m_xcoStep-1; n_xco++ ) {
1299         xco = double(n_xco)/double(m_xcoStep);
1300         new_xco( xco );
1301
1302         for( n_T=0; n_T<=m_TempNumber; n_T++ ) {
1303             Ts = m_TempStart + m_TempStep*double(n_T);
1304             new_T( Ts );
1305
1306             // INTEGRATIONS
1307             for( nDir = 0; nDir <= 1; nDir++ ) {
1308                 if( nDir == 0 ) {
1309                     co[0]=co[1]=1.0; o[0]=o[1]=0.0;

```

```

1310         } else {
1311             co[0]=co[1]=0.0; o[0]=o[1]=1.0;
1312         }
1313         for( nTime = 1; nTime <= tmax; nTime++ ) {
1314             ErrCode = Integration();
1315             if( ErrCode != 0 ) goto Error;
1316         }
1317         CO2prod[nDir] = totprod;
1318     }
1319
1320     // OUTPUT
1321     if( CO2prod[0]/CO2prod[1] < 1.02 &&
1322         CO2prod[0]/CO2prod[1] > 0.98 ) {
1323     } else {
1324         fprintf(bro, "\n%e", xco);
1325         fprintf(bro, "\t%e", Ts);
1326     }
1327 }
1328 }
1329
1330 Error:
1331     fclose(bro);
1332
1333     if( eps == 0.0 ) ErrCode = 64;
1334     ErrorVerbose( ErrCode );
1335 }

```

## A.2 Reaction-diffusion model

### A.2.1 Reaction-diffusion model

This Fortran 90 program `diff.f90` produces the output for RD.

Output file	Description
<code>info.txt</code>	A small summary of the parameters used, the output of this file is also sent to STD OUT
<code>gco2VSxco.dat</code>	Global CO <sub>2</sub> production vs. $x_{CO}$
<code>ThPhiR.dat</code>	Coordinates of each surface element on the particle, in spherical coordinates ( $\theta, \phi, r$ )
<code>fluxo.dat</code>	Local O <sub>2</sub> flux
<code>fluxco.dat</code>	Local CO flux
<code>fluxbs.dat</code>	Local backscattered flux
<code>fort.#1<sup>a</sup></code>	Transient global CO <sub>2</sub> production
<code>fort.#2<sup>b</sup></code>	Local oxygen coverage
<code>fort.#3<sup>c</sup></code>	Local CO coverage (not generated by default)
<code>fort.#4<sup>d</sup></code>	Local CO <sub>2</sub> production

$$^a \#1 = x_{CO} \times 10^5$$

$$^b \#2 = \text{integration step/parameter} \times 10^7 + \#1$$

$$^c \#3 = \#2 + 1$$

$$^d \#4 = \#2 + 2$$

```

1 program diff
2 !
3 ! Creation Date:      12.01.2004 15:17:36
4 ! Last Modification: 23.03.2004 14:44:05

```

```

5 ! Author:           Mathias Laurin
6 !
7 ! include backscattering + shadow
8 ! (Carsten Beta     5148)
9 !
10 implicit none
11 integer*4, parameter :: nThetaParam=19, nPhiParam=20, nrbsParam=1000
12 integer*4 :: nPhi, nTheta, nrbs, nthetabs
13 integer*4 :: k, MaxIt
14 integer*4 :: savqo, nsavqo, savgco2, nsavgco2, nxco, nfile
15 integer*4 :: mqco, noBS, nhomo
16 real*8 :: rbs, rbsmax, bs(0:nThetaParam, nrbsParam), drbs
17 real*8 :: rx, ry, rz
18 real*8 :: bsx, bsy, bsvect(3), normbs, sumbsco, sumbsa, sola
19 real*8 :: W(4)
20 real*8 :: Phi, Theta, radius(0:nPhiParam+1), ra, rc, ex2
21 real*8 :: p(3), q(3), r(3), normr
22 real*8 :: alpha, sina, cosa, beta, cosb, sinb, ThetaPhi
23 real*8 :: dq2, df1, df2
24 real*8 :: dq2f(nPhiParam), df1f(nPhiParam), df2f(nPhiParam)
25 real*8 :: cosg, sing, cose
26 real*8 :: sinfi(0:nPhiParam+1), cosfi(0:nPhiParam+1)
27 real*8 :: sinq(-1:nThetaParam+1), cosq(-1:nThetaParam+1)
28 real*8 :: fluxo(0:nThetaParam, 0:nPhiParam)
29 real*8 :: fluxco(0:nThetaParam, 0:nPhiParam)
30 real*8 :: dfluxo(0:nThetaParam, 0:nPhiParam)
31 real*8 :: dfluxco(0:nThetaParam, 0:nPhiParam)
32 real*8 :: dq, df
33 real*8 :: dqodt, dqcodt, qoi, qcoi
34 real*8 :: qo(-1:nThetaParam+1, -1:nPhiParam+1)
35 real*8 :: qco(0:nThetaParam, 0:nPhiParam)
36 real*8 :: ro, rco
37 real*8 :: nu_d, en_d, nu_r, en_r, e1, a_d
38 real*8 :: T, rt
39 real*8 :: Ftot, Ptot, fco, fo2
40 real*8 :: kb, pi, Tvel, Rgp, Na
41 real*8 :: mco, mo2, nPd, xco
42 real*8 :: sCO, sO2, Ct
43 real*8 :: d1, pco, po, kr
44 real*8 :: en_diff, dfick
45 real*8 :: dt
46 real*8 :: dum, duma, crit, lap
47 real*8 :: gco2, avgqco, avgqo, lco2(0:nThetaParam, 1:nPhiParam), qcoparam
48
49 ! Files
50 open(unit=1,file='ThPhiR.dat', status='replace', action='write')
51 open(unit=2,file='fluxo.dat', status='replace', action='write')
52 open(unit=3,file='fluxco.dat', status='replace', action='write')
53 open(unit=4,file='fluxbs.dat', status='replace', action='write')
54 !open(unit=15,file='profile_phiVSr.dat', status='replace', action='write')
55 open(unit=25,file='gco2VSxco.dat', status='replace', action='write')
56
57 ! General
58 maxIt = 2500000
59 dt = 1.0d-4 ! [s]
60 savgco2 = 25000
61 savqo = maxIt
62
63 ! Constants
64 kb = 1.380658d-23 ! [J/K] !! [J] == [Kg.m2.s-2]
65 Na = 6.0221367d23 ! [mol-1]
66 Rgp = Na*kb ! [J.mol-1.K-1] !! [Kg.m2.s-2.mol-1.K-1]
67 pi = 4.0d0*atan(1.0d0)
68
69 !Ptot = 1.0d-4 ! [Torr]
70 !Ptot = Ptot/760.0d0*101.325d0 ! [Pa = kg.m-1.s-2]
71 Ptot = 1.0d-4 ! [Pa]

```



```

72 T = 415.0d0 ! [K]
73 RT = Rgp*T ! [J.mol-1]
74 en_diff = 55.0d3 ! [J.mol-1]
75 ! Diffusion
76 Dfick = 1.0d-3 ! [cm2.s-1]
77 Dfick = Dfick*1.0d-4 ! [m2.s-1]
78 Dfick = Dfick*dexp(-en_diff/RT) ! [m2.s-1]
79
80 !Dfick = 0.0d0
81
82 mqco = 1 ! m = 0: cov. dep.
83 ! m = 1: no cov. dep.
84 qcoparam = 2.0d0
85 noBS = 0 ! if noBS=1, no backscattering
86
87 nhomo = 0 ! if nhomo=1, homogeneous fluxes.
88
89 ! Initial values
90 qoi = 0.d0
91 qcoi = 0.5d0
92
93 nPd = 1.53d15 ! [cm-2]
94 nPd = nPd*1.0d4 ! [m-2]
95 mco = 28.010d-3/Na ! [kg]
96 mo2 = 31.999d-3/Na ! [kg]
97 Tvel = 300.0d0 ! [K]
98
99 ! Desorption
100 nu_d = 4.0d14 ! [s-1]
101 en_d = 142.0d3 ! [J.mol-1]
102 a_d = 0.12
103 ! Reaction
104 nu_r = 5.0d7 ! [s-1]
105 en_r = 53.0d3 ! [J.mol-1]
106 ! Adsorption CO
107 sCO = 0.7d0
108 Ct = 0.3d0
109 ! Adsorption O2
110 sO2 = 1.0d0 - 7.4d-4*T
111
112 ! ellipse
113 rc = 450.0d-9 ! height [m]
114 ra = 250.0d-9 ! 1/2 width [m]
115
116 if( ra .ge. rc ) then ! oblate spheroid (squashed)
117 ex2 = 1.0d0 - rc*rc/(ra*ra) ! eccentricity^2
118 else ! prolate spheroid (pointy)
119 ex2 = 1.0d0 - ra*ra/(rc*rc)
120 endif
121 write(*,*) 'eccentricity',ex2
122 if( (ex2.lt.0.0d0).or.(ex2.ge.1.0d0) ) stop
123
124 !-----
125 ! ANGLES: Discretisation, calculation of cos, sin and radius
126 !-----
127
128 df = 0.5d0*Pi/real(nPhiParam+1)
129 dq = 2.0d0*Pi/real(nThetaParam+1) ! +1 to exclude 2 Pi
130
131 alpha = 55.0d0 ! incidence of the O2 beam
132 ! from the (xy) plane [deg]
133 alpha = alpha/180.0d0*Pi ! [radian]
134
135 beta = 90.0d0 ! incidence of the CO beam [deg]
136 beta = beta/180.0d0*Pi ! [radian]
137
138 ThetaPhi = real((nThetaParam+1)*nPhiParam)

```

```

139
140 crit          = 1.0d-4                ! cos / sin annihilation crit.
141 do nPhi=0, nPhiParam+1
142   Phi = df*real(nPhi)
143
144   sinfi(nPhi) = dsin(Phi)
145   if((sinfi(nPhi) .lt. crit).and.(sinfi(nPhi) .gt. -crit)) sinfi(nPhi) = 0.0d0
146   if(sinfi(nPhi) .gt. 1.0d0-crit) sinfi(nPhi) = 1.0d0
147   if(sinfi(nPhi) .lt. -1.0d0+crit) sinfi(nPhi) = -1.0d0
148
149   cosfi(nPhi) = dcos(Phi)
150   if((cosfi(nPhi) .lt. crit).and.(cosfi(nPhi) .gt. -crit)) cosfi(nPhi) = 0.0d0
151   if(cosfi(nPhi) .gt. 1.0d0-crit) cosfi(nPhi) = 1.0d0
152   if(cosfi(nPhi) .lt. -1.0d0+crit) cosfi(nPhi) = -1.0d0
153
154   if( rc .ge. ra ) then                ! prolate sph.d
155     radius(nphi) = (1.0d0 - ex2)/(1.0d0 - ex2*cosfi(nphi)*cosfi(nphi))
156     radius(nPhi) = rc*dsqrt(radius(nphi))
157   else                                  ! oblate sph.d
158     radius(nphi) = (1.0d0 - ex2)/(1.0d0 - ex2*sinfi(nphi)*sinfi(nphi))
159     radius(nPhi) = ra*dsqrt(radius(nphi))
160   endif
161   !write(15,*) phi/pi*180.0d0, radius(nphi)
162 enddo
163 do nTheta=-1, nThetaParam+1
164   Theta = dq*real(nTheta)
165
166   sinq(nTheta) = dsin(Theta)
167   if((sinq(nTheta) .lt. crit).and.(sinq(nTheta) .gt. -crit)) sinq(nTheta) = 0.0d0
168   if(sinq(nTheta) .gt. 1.0d0-crit) sinq(nTheta) = 1.0d0
169   if(sinq(nTheta) .lt. -1.0d0+crit) sinq(nTheta) = -1.0d0
170
171   cosq(nTheta) = dcos(Theta)
172   if((cosq(nTheta) .lt. crit).and.(cosq(nTheta) .gt. -crit)) cosq(nTheta) = 0.0d0
173   if(cosq(nTheta) .gt. 1.0d0-crit) cosq(nTheta) = 1.0d0
174   if(cosq(nTheta) .lt. -1.0d0+crit) cosq(nTheta) = -1.0d0
175 enddo
176
177 cosa = dcos(alpha)
178 if((cosa .lt. crit).and.(cosa .gt. -crit)) cosa = 0.0d0
179 if(cosa .gt. 1.0d0-crit) cosa = 1.0d0
180 if(cosa .lt. -1.0d0+crit) cosa = -1.0d0
181 sina = dsin(alpha)
182 if((sina .lt. crit).and.(sina .gt. -crit)) sina = 0.0d0
183 if(sina .gt. 1.0d0-crit) sina = 1.0d0
184 if(sina .lt. -1.0d0+crit) sina = -1.0d0
185
186 cosb = dcos(beta)
187 if((cosb .lt. crit).and.(cosb .gt. -crit)) cosb = 0.0d0
188 if(cosb .gt. 1.0d0-crit) cosb = 1.0d0
189 if(cosb .lt. -1.0d0+crit) cosb = -1.0d0
190 sinb = dsin(beta)
191 if((sinb .lt. crit).and.(sinb .gt. -crit)) sinb = 0.0d0
192 if(sinb .gt. 1.0d0-crit) sinb = 1.0d0
193 if(sinb .lt. -1.0d0+crit) sinb = -1.0d0
194
195 !-----
196 !           Factors to the LAPLACIAN (out of the main loop)
197 !-----
198 do nphi = 1, nhiparam
199   dum          = radius(nphi)*radius(nphi)
200
201   dq2f(nphi) = dum * sinfi(nphi)*sinfi(nphi) * dq*dq
202   dq2f(nphi) = 1.0d0/dq2f(nphi)
203
204   df1f(nphi) = dum * sinfi(nphi) * df
205   df1f(nphi) = cosfi(nphi)/df1f(nphi)

```

```

206
207     df2f(nphi) = 1.0d0/(dum * df*df)
208
209     !write(55,'(3es20.8)') dq2f(nphi),df1f(nphi),df2f(nphi)
210 enddo
211
212 !-----
213 !   BACKSCATTERING from the Support:
214 !-----
215 rbsmax = 25.0d-6           ! [m]
216 drbs   = (rbsmax - ra)/real(nrbsParam) ! [m]
217 write(*,*) 'dq=',dq
218 write(*,*) 'dr=',abs(drbs)
219
220 !-----
221 !                               FLUX DISTRIBUTION
222 !-----
223 dfluxo(:, :) = 0.0d0
224 dfluxco(:, :) = 0.0d0
225 do nPhi=1, nPhiParam
226     do nTheta=0, nThetaParam
227         ! ----- DIRECT FLUX from the beam
228         ! cart. coord.
229         ! orthogonal vectors (p/Phi, q/Theta)
230         p(1) = cosq(ntheta)*( radius(nphi-1)*sinfi(nphi-1) - radius(nphi+1)*sinfi(nphi+1) )
231         p(2) = sinq(ntheta)*( radius(nphi-1)*sinfi(nphi-1) - radius(nphi+1)*sinfi(nphi+1) )
232         p(3) =                radius(nphi-1)*cosfi(nphi-1) - radius(nphi+1)*cosfi(nphi+1)
233
234         q(1) = radius(nphi)*sinfi(nphi)*(cosq(ntheta-1) - cosq(ntheta+1))
235         q(2) = radius(nphi)*sinfi(nphi)*(sinq(ntheta-1) - sinq(ntheta+1))
236         q(3) = 0.0d0
237
238         ! r normal/part. ie: r.q = r.p = 0
239         dum = p(2)*q(1) - p(1)*q(2)
240         r(1) = p(3)*q(2)/dum
241         r(2) = -p(3)*q(1)/dum
242         r(3) = 1.0d0
243         normr= dsqrt( r(1)*r(1) + r(2)*r(2) + r(3)*r(3) )
244
245         ! Flux O2
246         ! scalar prod. ie: r.Flux = normr*cosg
247         cosg = r(1)*cosa + r(3)*sina
248         cosg = cosg/normr
249         if( cosg .lt. crit ) cosg = 0.0d0
250         dfluxo(ntheta, nphi) = cosg
251
252         ! Flux CO
253         cosg = r(1)*cosb + r(3)*sinb
254         cosg = cosg/normr
255         if( cosg .lt. crit ) cosg = 0.0d0
256         dfluxco(ntheta, nphi) = cosg
257
258         ! ----- BACKSCATTERED FLUX
259         if( noBS .ne. 1 ) then
260             sumbs0 = 0.0d0
261             sumbsc0 = 0.0d0
262
263             ! Origin of r, on the particle
264             rx = radius(nphi)*cosq(ntheta)*sinfi(nphi)
265             ry = radius(nphi)*sinq(ntheta)*sinfi(nphi)
266             rz = radius(nphi)*cosfi(nphi)
267
268             do nthetabs=0, nthetaparam
269                 do nrbs=1, nrbsparam
270                     rbs = real(nrbs)/real(nrbsparam)*(rbsmax - ra) + ra
271
272                     ! origin of BS flux / substrate

```

```

273      bsx = rbs*cosq(nthetabs)
274      bsy = rbs*sinq(nthetabs)
275
276      ! Area elt / substrate
277      bsa = rbs*drbs*dq
278
279      ! BSvec is backscattered flux vector (btw pts BS and R)
280      bsvec(1) = rx-bsx
281      bsvec(2) = ry-bsy
282      bsvec(3) = rz
283      normbs = bsvec(1)*bsvec(1) + bsvec(2)*bsvec(2) + bsvec(3)*bsvec(3)
284      normbs = dsqrt(normbs)
285
286      ! sing is INTENSITY of BS flux (exp. vector norm)
287      sing = abs(rz) / normbs
288      if( sing .lt. crit )      sing = 0.0d0
289      if( sing .gt. 1.0d0-crit ) sing = 1.0d0
290      ! normalisation (solid angle / half sph.)
291      sing = sing/pi
292
293      ! cose/( normbs*normbs ) is solid angle
294      ! scal. prod. ( BS.r )
295      cose = bsvec(1)*r(1) + bsvec(2)*r(2) + bsvec(3)*r(3)
296      cose = -cose/(normbs*normr)
297      if( cose .lt. crit )      cose = 0.0d0
298      if( cose .gt. 1.0d0-crit ) cose = 1.0d0
299
300      sola = sing*cose/(normbs*normbs)*bsa
301
302      ! shadow of O2 beam
303      dum = 2.0d0*rc*rc*bsx*cosa
304      dum = dum*dum
305
306      duma = 4.0d0*(rc*rc*cosa*cosa + ra*ra*sina*sina)
307      duma = duma*(rc*rc*( bsx*bsx + bsy*bsy)-ra*ra )
308
309      if( (dum .lt. dum) .or. (bsx.gt.0.0d0) ) then
310          ! out of the shadow
311          sumbso = sumbso + sola
312      endif
313
314      ! shadow of CO beam
315      dum = 2.0d0*rc*rc*bsx*cosb
316      dum = dum*dum
317
318      duma = 4.0d0*(rc*rc*cosb*cosb + ra*ra*sinb*sinb)
319      duma = duma*(rc*rc*( bsx*bsx + bsy*bsy)-ra*ra )
320
321      if( (dum .lt. dum) .or. (bsx.gt.0.0d0) ) then
322          ! out of the shadow
323          sumbsco = sumbsco + sola
324      endif
325
326      enddo
327      enddo
328      dfluxco(ntheta, nphi) = dfluxco(ntheta, nphi) + sumbsco*sinb
329      dfluxo(ntheta, nphi) = dfluxo(ntheta, nphi) + sumbso*sina
330  endif
331  enddo
332  enddo
333
334  ! Homogeneous model
335  if( nhomo .eq. 1 ) then
336      dfluxo(:, :) = sum(dfluxo(:, :))/thetaphi
337      dfluxco(:, :) = sum(dfluxco(:, :))/thetaphi
338  endif
339

```

```

340 ! save flux distributions
341 do nphi=1, nPhiParam
342   do nTheta=0, nThetaParam
343     phi = real(nPhi)*df
344     theta = real(nTheta)*dq
345     write(1,'(3es20.8)') theta, phi, radius(nphi) !ThPhiR.dat
346     write(2,'(es20.8)') dfluxo(ntheta,nphi) !fluxo.dat
347     write(3,'(es20.8)') dfluxco(nTheta, nPhi) !fluxco.dat
348     write(4,'(es20.8)') sumbsco ! contrib. BS
349   enddo
350 enddo
351
352 close(1); close(2); close(3);close(4)
353
354
355 !----- XCO LOOP --
356 fco = dsqrt( 2.0d0*pi*mco*kb*Tvel ) ! [Kg.m.s-1]
357 fo2 = dsqrt( 2.0d0*pi*mo2*kb*Tvel )
358 do nxco = 0, 40
359   !xco = real(nxco)/20.0d0 ! nxco = 1,19
360   xco = real(nxco)/200.0d0 + .34d0 ! nxco = 0,199
361   !if( nxco .eq. 1 ) xco = 0.34d0
362   !if( nxco .eq. 2 ) xco = 0.67d0
363
364   Ftot = xco*fco
365   Ftot = Ftot + (1.0d0-xco)*fo2
366   Ftot = Ftot/Ptot ! [m-2.s-1]
367
368   ! Rate constants
369   kr = nu_r*dexp(-en_r/rt) ! [s-1]
370   pco = xco*Ftot/nPd ! [s-1]
371   po = (1.0d0-xco)*Ftot/nPd ! [s-1]
372
373   fluxo(:, :) = dfluxo(:, :) *po
374   fluxco(:, :) = dfluxco(:, :) *pco
375
376   ! Initial values
377   qo(:, :) = qoi
378   qco(:, :) = qcoi
379
380   nsavqo = 0
381   nsavgco2 = 0
382
383 !----- TIME LOOP --
384 write(*,*) 'start integration', xco
385 do k=1, maxit
386   nsavqo = nsavqo + 1
387   nsavgco2 = nsavgco2 + 1
388
389   gco2 = 0.0d0
390   avgqco = 0.0d0
391   avgqo = 0.0d0
392
393   !!! INTEGRATION
394   do nPhi = 1, nPhiParam
395     do nTheta = 0, nThetaParam
396       ! relative coverages
397       rco = 2.0d0*qco(nTheta,nPhi)
398       ro = 4.0d0*qo(nTheta,nPhi)
399
400       ! Adsorption CO
401       dum = 1.0d0 - rco - Ct *ro
402       if( dum.gt.0.0d0 ) then
403         W(1) = fluxco(nTheta, nPhi)*sCO*dum
404       else
405         W(1) = 0.0d0
406       endif

```

```

407
408      ! Desorption CO
409      e1 = en_d*(1.0d0 - a_d*(rco))
410      d1 = nu_d*dexp( -e1/rt )
411      W(2) = d1*qco(nTheta,nPhi)
412
413      ! Reaction
414      W(3) = Kr*qco(nTheta,nPhi)*qo(nTheta,nPhi)
415      gco2 = gco2 + w(3)
416      lco2(nTheta, nPhi) = w(3)
417
418      ! Adsorption O2
419      dum = 1.0d0 - rco - ro
420      if( dum.gt.0.0d0 ) then
421          W(4) = fluxo(ntheta, nphi)*sO2* dum*dum
422      else
423          W(4) = 0.0d0
424      endif
425
426      ! Diffusion
427      ! BC: Theta: PBC; Phi: no flux
428      if(nTheta .eq. 0)      qo(-1, nphi)      = qo(nThetaParam, nphi)
429      if(nTheta .eq. nThetaParam) qo(ntheta+1, nphi) = qo(0, nphi)
430      if(nPhi .eq. nPhiParam)   qo(ntheta, nPhiParam+1) = qo(ntheta, nPhiParam)
431      if(nPhi .eq. 1)          qo(nTheta, 0)      = qo(nTheta, 1)
432
433      dq2 = qo(ntheta-1, nPhi) + qo(ntheta+1, nPhi) - 2.0d0*qo(ntheta, nPhi)
434      dq2 = dq2f(nphi)*dq2
435
436      df1 = qo(ntheta, nPhi+1) - qo(ntheta, nPhi)
437      df1 = df1f(nphi)*df1
438
439      df2 = qo(ntheta, nPhi-1) + qo(ntheta, nPhi+1) - 2.0d0*qo(ntheta, nPhi)
440      df2 = df2f(nphi)*df2
441
442      lap = dq2 + df1 + df2
443
444      ! Integration
445      dqodt = 2.0d0*W(4) - W(3) + Dfick*lap
446      dqcodt = W(1) - W(2) - W(3)
447
448      qo(nTheta, nPhi) = qo(nTheta,nPhi) + dqodt * dt
449      qco(nTheta,nPhi) = qco(nTheta,nPhi) + dqcodt * dt
450
451      avgqo = avgqo + qo(nTheta, nPhi)
452      avgqco = avgqco + qco(nTheta,nPhi)
453
454      ! Divergence
455      if(      qo(nTheta, nPhi) .lt. 0.0d0 ) then
456          write(*,*) 'DIV: qo .lt. 0', Dfick*lap
457          stop
458      elseif( qo(nTheta, nPhi) .gt. 0.25d0 ) then
459          write(*,*) 'DIV: qo .gt. 0.25', Dfick*lap
460          stop
461      elseif(qco(nTheta, nPhi) .lt. 0.0d0 ) then
462          write(*,*) 'DIV: qco .lt. 0', Dfick*lap
463          stop
464      elseif(qco(nTheta, nPhi) .gt. 0.5d0 ) then
465          write(*,*) 'DIV: qo .gt. 0.5', Dfick*lap
466          stop
467      endif
468      enddo ! nTheta
469      enddo ! nPhi
470
471      gco2 = gco2 / thetaphi
472
473      ! diff CO infinitely fast but fct of Qo

```

```

474     avgqo = avgqo / thetaphi
475     avgqco = avgqco / thetaphi
476
477     if( (mqco .eq. 1).or.(avgqo .ge. 0.25d0) ) then      ! CO equally shared
478         qco(:, :) = avgqco
479     else          ! mqco .eq. 0
480         dum = avgqco/(0.25d0 - avgqo)
481         do nphi = 1, nhiparam
482             do ntheta = 0, nthetaparam
483                 qco(ntheta, nphi) = ( 0.25d0 - qo(ntheta, nphi) ) * dum
484                 qco(ntheta, nphi) = qco(ntheta, nphi) - avgqco
485                 qco(ntheta, nphi) = qco(ntheta, nphi) * qcoparam
486                 qco(ntheta, nphi) = qco(ntheta, nphi) + avgqco
487             enddo
488         enddo
489     endif
490     if( (k.eq.1).and.(mqco.eq.0) ) then
491         write(*,*) sum(qco(:, :)), avgqco * thetaphi
492     endif
493
494     ! save global co2 production
495     if( nsavgco2 .eq. savgco2 ) then
496         write(int(xco*1.0d5), '(2es20.8)') real(k)*dt, gco2
497         nsavgco2 = 0
498     endif
499
500     ! save cov distributions and local CO2 prod.
501     if( nsavqo .eq. savqo ) then
502         nsavqo = 0
503         nfile = int( real(k)/real(savqo))
504         if( k.eq. maxit-1000 ) nfile = 99
505         nfile = int( real(nfile)*1.0d7 )
506         nfile = nfile + int(xco*1.0d5)
507         do nPhi=1, nPhiParam
508             do nTheta=0, nThetaParam
509                 phi = real(nPhi)*df
510                 theta = real(nTheta)*dq
511
512                 write(nfile, '(es20.8)') qo(ntheta, nphi)
513                 write(nfile+2, '(es20.8)') lco2(ntheta, nphi)
514                 if( mqco .ne. 1 ) then
515                     write(nfile+1, '(es20.8)') qco(ntheta, nphi)
516                 endif
517             enddo
518         enddo
519     endif
520     close(nfile)
521     if( mqco .ne. 1 ) close(nfile+1)
522     close(nfile+2)
523
524     enddo !----- END TIME LOOP --
525
526     write(25, '(2es20.8)') xco, gco2      ! gco2VSxco.dat
527     close(int(xco*1.0d5))
528
529     enddo !----- END XCO LOOP --
530
531     write(*,*) 'diff Done'
532
533     open(unit=33, file='info.txt', status='replace', action='write')
534     write(33, '(a8, es20.8)') 'T=',      T
535     write(33, '(a8, es20.8)') 'E=',      en_diff
536     write(33, '(a8, es20.8)') 'D=',      Dfick
537     write(33, '(a8, es20.8)') 'alpha=', Alpha
538     write(33, '(a8, es20.8)') 'beta=', Beta
539     write(33, '(a8, i8)')      'mqco=', mqco
540     write(33, '(a8, i8)')      'noBS=', noBS

```

```

541 write(33,'(a8,i8)')      'nhomo=',nhomo
542 close(33)
543
544 write(*,'(a8,es20.8)') 'T=',    T
545 write(*,'(a8,es20.8)') 'E=',    en_diff
546 write(*,'(a8,es20.8)') 'D=',    Dfick
547 write(*,'(a8,es20.8)') 'alpha=',Alpha
548 write(*,'(a8,es20.8)') 'beta=',Beta
549 write(*,'(a8,i8)')      'mqco=',mqco
550 write(*,'(a8,i8)')      'noBS=',noBS
551 write(*,'(a8,i8)')      'nhomo=',nhomo
552
553 stop
554 end
555

```

## A.2.2 Simulation of the experimental data

This Matlab function `SimAR.m` simulates the AR CO<sub>2</sub> distribution from the oxygen coverage distributions. The coordinates to each surface elements are loaded from the file `ThPhiR.dat` generated by `diff.f90`. This file must be in the current directory.

The function takes two parameters: `qo`, oxygen coverage as generated by `diff.f90` and `cosn`, order of the cosine distribution from the particle. The output `out = [phiar,tot,totn]` consists of three vectors: `totn` is the normalized CO<sub>2</sub> production `tot` at position `phiar`. The angle distributions are also plotted in Matlab.

```

1  function out = SimAR(qo, cosn);
2
3  % Simulation of the AR distribution
4  % for Matlab
5  %
6  % Creation Date:      26.02.2004   14:34
7  % Last Modification:  03.05.2004   16:25
8  %
9
10 if nargin == 1,
11     cosn = 1;
12 end
13
14 nq  = 20;
15 nf  = 20;
16 nth = 1;
17 nfi = nf;
18 nmax = nq*nf;
19 us  = 40;
20
21 % load coordinates of pts on spheroid
22 coord = load('ThPhiR.dat');
23 th    = coord(:,1)';
24 phi   = coord(:,2)';
25 radius= coord(:,3)';
26 phi2  = 0.5*pi-phi;
27 [sx, sy, sz] = sph2cart(th,phi2,radius);
28 tri   = delaunay(sx,sy);
29
30 % trigo fct
31 sfi   = sin(phi);
32 cfi   = cos(phi);
33 sq    = sin(th);
34 cq    = cos(th);
35 % boundary cond.
36 sfi(nmax+1) = 0.0;
37 sfi(nmax+2) = 1.0;

```



```

38 cfi(nmax+1) = 1.0;
39 cfi(nmax+2) = 0.0;
40 radius(nmax+1) = 450.0e-9;
41 radius(nmax+2) = 250.0e-9;
42
43 % area on the sph.d
44 dq = abs(th(1)-th(2));
45 df = abs(phi(1)-phi(1+nf));
46 dA = radius.*radius.*sfi *dq*df;
47 dA = dA(1:400);
48
49 % vect normal to the particle
50 for n = 1:nmax
51     % // (z)
52     m = n-nfi;
53     if m < 1, m = nmax+1; end
54     M = n+nfi;
55     if M > nmax, M = nmax+2; end
56
57     p(n,1) = cq(n)*( radius(m)*sfi(m) - radius(M)*sfi(M) );
58     p(n,2) = sq(n)*( radius(m)*sfi(m) - radius(M)*sfi(M) );
59     p(n,3) = radius(m)*cfi(m) - radius(M)*cfi(M);
60
61     % // (xy)
62     m = n-nth;
63     if m < 1, m = m+nmax; end
64     M = n+nth;
65     if M > nmax, M = M-nmax; end
66
67     q(n,1) = radius(n)*sfi(n)*(cq(m) - cq(M));
68     q(n,2) = radius(n)*sfi(n)*(sq(m) - sq(M));
69     q(n,3) = 0.;
70 end
71 dum = p(:,2).*q(:,1) - p(:,1).*q(:,2);
72 rx = p(:,3).*q(:,2)./dum;
73 ry = -p(:,3).*q(:,1)./dum;
74 rz = ones(nmax,1);
75 normr = sqrt( rx.*rx + ry.*ry + rz.*rz );
76 rx = rx./normr;
77 ry = ry./normr;
78 rz = rz./normr;
79
80 % unit sphere (position detector)
81 [usx, usy, usz] = sphere(us);
82 [usq, usf, usr] = cart2sph(usx,usy,usz);
83 sus = size(usx);
84 dqus = 2.0 * pi / us;
85 dfus = pi / us;
86 dAus = cos(usf)*dqus * dfus;
87
88 % Direct flux
89 tof = zeros(sus);
90 for n=1:nmax
91     % Cosine Distribution
92     dvx = usx - sx(n);
93     dvy = usy - sy(n);
94     dvz = usz - sz(n);
95     normdv = sqrt(dvx.*dvx + dvy.*dvy + dvz.*dvz);
96
97     ca = dvx*rx(n) + dvy*ry(n) + dvz*rz(n);
98     ca = ca./normdv;
99     ca = ca.*(ca > 0.0);
100    ca = ca.^(cosn+1); % cosn+1 for solid angle
101
102    % Flux = N Sum( qo(q,f) dA(f) cos(f) )
103    tof = tof + qo(n)*ca*dA(n);
104 end

```

```

105
106 % back-scattering: integration
107 bs = (tof.*(usz<0.0)) + ((0.5*tof).*(usz==0));
108 bs = bs.*dAus;
109 bsi = sum(sum(bs));
110
111 % cosine distribution to higher half sphR
112 bsf = bsi/pi * sin(usf.*(usz > 0.0 ));
113
114 % cut
115 mus = us/2+1;
116 phiar = [-usf(mus:sus(1),1)+pi; usf(mus:sus(1),mus)];
117 dirar = [ tof(mus:sus(1),1); tof(mus:sus(1),mus)];
118 bsfar = [ bsf(mus:sus(1),1); bsf(mus:sus(1),mus)];
119 tot = dirar + bsfar;
120 totn = tot./max(tot);
121
122 figure(2)
123 hold off
124 polar(phiar, tot, 'g');
125 hold on
126 polar(phiar2,arall, 'g');
127 polar(phiar, dirar, 'k');
128 polar(phiar, bsfar, 'r');
129 hold off
130
131 % figure(3)
132 % surf(Dx, Dy, Dz, tof) %sph
133 % surf(Dx(21:41,:), Dy(21:41,:), tof(21:41,:))
134 % shading interp, axis square
135
136 out = [phiar, tot, totn];
137 out = sortrows(out,1);
138 % EOF

```

## A.3 Stochastic model

Object oriented Fortran 90 free and fixed forms has been chosen for SM. Three different programs are used to integrate the master equation (ME) using different algorithms.

### A.3.1 Euler algorithm

EulerDRV.f90, which contains the subroutine `TransRate` and the subroutines `Equilibrium.f90` and `Euler.f90` perform an integration of the master equation in time using the Euler forward method. The main parameters are LL, the total number of sites (2000 for Sample C, 540 for Sample B and 60 for Sample A); `c1` and `c2`, CO and O coverages, respectively.

Output file	Description
<code>info.txt</code>	Summary of the parameters used
<code>E_Prob_xy.txt</code>	Table of coordinates in coverage space
<code>CO_cov_p_facet.txt</code>	CO coverage per facet
<code>E_CO2.txt</code>	Transient global CO <sub>2</sub> production
<code>fort.1000001</code>	Probability distribution in coverage space for the O-precovered case
<code>fort.2000001</code>	Probability distribution in coverage space for the CO-precovered case

```

1 program EulerDrv
2 !
3 ! Creation Date: 12.08.03 11:13:59

```

```

4  ! Last Modification: 16.09.2003 16:25:51
5  !
6  ! Author:           Mathias Laurin
7  !
8  implicit none
9  integer*4, parameter  :: LL=60
10 ! big part  LL = 540
11 ! small part LL = 60
12 integer*4, parameter  :: LL1=LL/2, LL2=LL/4
13 real*8                :: rLL, rLL1, rLL2
14 integer*4             :: k, kkk, MaxIt
15 integer*4             :: n1, n2
16 real*8                :: c1, c1a, c1b, c2, c2a, c2b
17 real*8                :: dt, t1, t2
18 real*8                :: ProdCO2, eps
19 real*8                :: Ftot, Ptot
20 real*8                :: kb, pi, Tvel, R, Na
21 real*8                :: mco, mo2, nPd
22 real*8                :: sCO, sO2, Ct, a_d
23 real*8                :: p1, p2
24 real*8                :: e, d1, kr
25 real*8                :: T, xco, RT
26 real*8, dimension(2) :: nu_d, nu_r, en_d, en_r
27
28 integer*4 :: savco2, savproba, co2step, probastep
29 integer*4 :: co2file, probafile
30
31 real*8, dimension(4)          :: W1, W2
32 real*8, dimension(4, 0:LL1, 0:LL2) :: W
33 real*8, dimension(0:LL1, 0:LL2)  :: P
34
35 !!! Parameters
36 rLL = real(LL)
37 rLL1 = real(LL1)
38 rLL2 = real(LL2)
39
40 ! General
41 Ptot = 1.0d-4      ! [Pa = kg.m-1.s-2]
42 T    = 415.0d0    ! [K]
43 xco  = 52.0d-2
44 t2   = 20.0d-2    ! % defects
45 dt   = 1.0d-3
46 MaxIt = 95000
47 ! 1200000 @ dt=1.0d-3 == 20 min
48 ! CPU == 750 min on Marvin (LL=540, kkk=2)
49 !savco2 = 500      ! int(1.0d0/dt)
50 !savproba = 500
51 !!if( savproba .gt. MaxIt ) savproba = MaxIt
52
53 t1 = 1.0d0-t2
54 ! Constants
55 kb = 1.380658d-23 ! [J/K]    !! [J] == [Kg.m2.s-2]
56 Na = 6.0221367d23 ! [mol-1]
57 R  = Na*kb        ! [J.mol-1.K-1] !! [Kg.m2.s-2.mol-1.K-1]
58 pi = 4.0d0*datan(1.0d0)
59
60 nPd = 1.53d15      ! [cm-2]
61 nPd = nPd*1.0d4    ! [m-2]
62 mco = 28.010d-3/Na ! [kg]
63 mo2 = 31.999d-3/Na ! [kg]
64 Tvel = 300.0d0     ! [K]
65
66 RT = R*T           ! [J.mol-1]
67
68 ! Desorption
69 nu_d(1) = 4.0d14   ! [s-1]
70 nu_d(2) = 4.0d14

```

```

71  en_d(1) = 142.0d3      ![J.mol-1]
72  en_d(2) = 117.0d3
73  a_d      = 0.12d0
74  ! Reaction
75  nu_r(1) = 5.0d7      ![s-1]
76  nu_r(2) = 5.0d7
77  en_r(1) = 53.0d3      ![J.mol-1]
78  en_r(2) = 44.0d3
79  ! Adsorption CO
80  sCO      = 0.7d0
81  Ct       = 0.3d0
82  ! Adsorption O2
83  sO2      = 1.0d0 - 7.4d-4*T
84
85  Ftot = xco*dsqrt( 2.0d0*pi*mco*kb*Tvel ) ![Kg.m.s-1]
86  Ftot = Ftot + (1.0d0-xco)*dsqrt( 2.0d0*pi*mo2*kb*Tvel )
87  Ftot = Ptot/Ftot      ![m-2.s-1]
88
89  ! Rate constants
90  p1      = xco*Ftot/nPd      ![s-1]
91  p2      = (1.0d0-xco)*Ftot/nPd  ![s-1]
92
93  !!!          Transition Rates (W)
94  eps = dexp( (en_d(2)-en_d(1))/RT )
95
96  open( unit=1, file='CO_cov_p_facet.txt', status='replace', action='write', form='formatted')
97  do n1 = 0,LL1
98      c1 = real(n1)/rLL
99      call equilibrium(c1, t1, eps, c1a, c1b)
100     write( 1,'(2es20.8e3)' ) c1a, c1b
101     do n2 = 0,LL2
102         c2 = real(n2)/rLL
103         c2a = t1*c2
104         c2b = t2*c2
105
106         call TransRate(c1a, c2a, 1, W1) !perfect
107         call TransRate(c1b, c2b, 2, W2) !defect
108
109         W(:, n1, n2) = t1*W1(:) + t2*W2(:)
110     enddo
111 enddo
112 close(1)
113
114 !!!          Files
115 open( unit=1, file='E_Prob_xy.txt', status='replace', action='write', form='formatted')
116 do n2 = 0,LL2
117     c2 = real(n2)/rLL2
118     do n1 = 0,LL1
119         c1 = real(n1)/rLL1
120         write( 1,'(2f20.6)' ) c1, c2
121     enddo
122 enddo
123 close(1)
124
125 open( unit=21, file='E_CO2.txt', status='replace', action='write', form='formatted')
126 write(21,'(2a8)' ) 'Time', 'ME'
127 write(21,'(2i8)' ) 0, 0
128
129 !!!! Info file
130 open( unit=1, file='Info.txt', status='replace', action='write', form='formatted')
131 write(1,'(a)')      'Parameters:'
132 write(1,'(i4,a20)') LL, ' adsorption sites'
133 write(1,'(a)')      'Constants:'
134 write(1,'(a8,es10.4)') 'Ptot ', Ptot
135 write(1,'(a8,f6.0)')  'T ', T
136 write(1,'(a8,es10.4)') 'Ftot ', Ftot
137 write(1,'(a8,f6.3)')  'x(CO)', xco

```

```

138 write(1,'(a)')          'Desorption'
139 write(1,'(a8,2es15.4)') 'nu  ', nu_d
140 write(1,'(a8,2es15.4)') 'en  ', en_d
141 write(1,'(a8,f6.4)')    'cov dep', a_d
142 write(1,'(a)')          'Reaction'
143 write(1,'(a8,2es15.4)') 'nu  ', nu_r
144 write(1,'(a8,2es15.4)') 'en  ', en_r
145 write(1,'(a)')          'Adsorption CO'
146 write(1,'(a8,es10.4)') 'nPd  ', nPd
147 write(1,'(a8,f6.4)')    'C(T) ', Ct
148 write(1,'(a8,f6.4)')    'S(CO)', sCO
149 write(1,'(a)')          'Adsorption O2'
150 write(1,'(a8,f6.4)')    'S(O2)', sO2
151 write(1,*)
152 write(1,'(a8,es12.4e2)') 'p(CO)', p1
153 write(1,'(a8,es12.4e2)') 'p(O2)', p2
154 write(1,'(a8,es12.4e2)') 'Kr   ', Kr
155 write(1,*)
156 write(1,'(a8,es12.4e2)') 'eps  ', eps
157 write(1,'(a)')          'Time'
158 write(1,'(a8,f10.6)')   'dt   ', dt
159 write(1,'(a8,i10)')     'Nb It', MaxIt
160 write(1,'(a8,f10.1)')   't max', real(MaxIt)*dt
161 write(1,'(a, f15.3)')   'Proba saved every [s]', dt*savproba
162 write(1,'(a, f15.3)')   'CO2 prod saved every [s]', dt*savco2
163 close(1)
164
165 do kkk = 1,1
166   !!!          Init Proba
167   co2file = 21
168   if( kkk .eq. 1 ) then
169     ! O2 first
170     P(:, :) = 0.0d0
171     P(0,LL2) = 1.0d0
172   else          ! kkk .eq. 2
173     ! CO first
174     p(:, :) = 0.0d0
175     p(LL1,0) = 1.0d0
176   endif
177
178   !!!          Nullify
179   probastep = 0
180   co2step = 0
181
182   do k=1, MaxIt
183
184     if( k .le. 2000 ) then
185       savco2 = 150
186       savproba = savco2
187     else
188       savco2 = 150
189       savproba = savco2
190     endif
191
192     probastep = probastep + 1
193     co2step = co2step + 1
194
195     call Euler(LL1, LL2, dt, W, P, ProdCO2)
196
197     !!! OUPUT
198     ! CO2 Prod
199     if( co2step .eq. savco2 ) then
200       write( co2file, '(f20.6,es20.6e4)' ) &
201         real(k)*dt, ProdCO2/rLL
202       co2step = 0
203     endif
204     ! proba distribution

```

```

205     if( probastep .eq. savproba ) then
206         probafile = k*10
207         do n2=0,LL2
208             do n1=0,LL1
209                 write( probafile,'(es20.6e4)' ) P(n1,n2)
210             enddo
211         enddo
212         close( probafile )
213         probastep = 0
214     endif
215     !!! END OUTPUT
216 enddo
217 enddo ! kkk
218
219 !!!     EOF
220 write(*,*) 'Euler DONE'
221 stop
222 contains
223 subroutine TransRate(c1, c2, fac, W)
224 !
225 ! Returns the matrices of transition rates
226 !
227 !         Transition Rates (W)
228 !     from J Chem Phys 114 4669 + Jens' Thesis
229 !
230 ! Creation Date      28.07.03
231 ! Last modification  19.08.03 16:34:36
232 ! Author             Mathias Laurin
233 !
234 implicit none
235 integer*4  :: n1, n2, k, fac
236 real*8     :: r1, r2, c1, c2
237 real*8     :: dum
238 real*8, dimension(4)  :: W
239
240 ! relative coverages
241 r1  = c1*2.0d0
242 r2  = c2*4.0d0
243
244 ! Adsorption CO
245 dum = 1.0d0 - r1 - Ct *r2
246 if( dum .ge. 0.0d0 ) then
247     W(1) = p1*sCO*dum
248 else
249     W(1) = 0.0d0
250 endif
251
252 ! Desorption CO
253 e  = en_d(fac)*( 1.0d0 - a_d*(r1) )
254 d1 = nu_d(fac)*dexp( -e/rt )
255 W(2) = d1*c1
256
257 ! Reaction
258 kr  = nu_r(fac)*dexp( -en_r(fac)/rt ) ![s-1]
259 W(3) = kr*c1*c2
260
261 ! Adsorption O2
262 dum = 1.0d0 - r1 - r2
263 if( dum.gt.0.0d0 ) then
264     W(4) = p2*sO2* dum*dum
265 else
266     W(4) = 0.0d0
267 endif
268
269 W(:) = rLL*W(:)
270
271 end subroutine TransRate

```

272 end program EulerDrv  
273

```
1  subroutine equilibrium(ctot, t1, eps, co1, co2)
2  !
3  ! Creation Date:      14.08.03 10:55:47
4  ! Last Modification: 18.08.03 16:52:54
5  ! Author:            Mathias Laurin
6  !
7  implicit none
8  real*8, intent(in)  :: ctot      ! total CO coverage
9  real*8, intent(in)  :: t1, eps   ! t1: fraction of facet 1 (non defect)
10 real*8, intent(out) :: co1, co2  ! CO cov on facet 1, 2
11
12 real*8                :: meps, comax, t1meps
13 real*8                :: a, b, c
14
15 comax = 0.5d0
16 meps  = 1.0d0-eps
17 t1meps = t1*meps
18
19 if( t1 .eq. 1.0d0 ) then
20   co1 = ctot
21   co2 = 0.0d0
22 else
23   if( ctot .eq. 0.0d0 ) then
24     co1 = 0.0d0
25     co2 = 0.0d0
26   elseif( ctot .eq. comax ) then
27     co1 = comax
28     co2 = comax
29   else
30     !!! J Catal 204 (2001) 378, Eq. [23]
31     a  = 0.5d0/t1meps
32     b  = ctot*meps + comax*( eps + t1meps )
33     c  = -4.0d0*comax*ctot*t1meps
34
35     c  = c + b*b
36     if( c .lt. 0.0d0 ) then
37       write(*,*) 'ERR: routine equilibrium'; stop
38     endif
39     c  = dsqrt(c)
40
41     co1 = a*( b-c )
42     if( (co1 .lt. 0.0d0) .or. (co1 .gt. comax) ) then
43       co1 = a*( b+c )
44     endif
45
46     co2 = (ctot - t1*co1)/(1.0d0-t1)
47   endif
48 endif
49
50 end subroutine equilibrium
```

```
1  subroutine Euler(LL1, LL2, dt, W, P, ProdCO2)
2  !
3  ! Numerical integration of the Master Equation 2D
4  ! Euler method
5  !
6  ! Creation Date      16.07.03 15:07:17
7  ! Last Modification  12.08.03 11:36:00
8  ! Author            Mathias Laurin
9  !
10 implicit none
```

```

11 integer*4, intent(in) :: LL1, LL2
12 real*8, intent(in) :: dt
13 real*8, intent(out) :: ProdCO2
14 real*8, dimension(4, 0:LL1, 0:LL2), intent(in) :: W
15 real*8, dimension(0:LL1, 0:LL2), intent(inout) :: P
16
17 integer*4 :: n1, n2
18 real*8 :: dum, sp, dpi
19 real*8, dimension(0:LL1, 0:LL2) :: dP
20
21 ! Nullify
22 ProdCO2 = 0.0d0
23 sp = 0.0d0
24
25 do n2 = 0, LL2
26   do n1 = 0, LL1
27     ! Adsorption CO
28     if( n1.ge.1 ) then
29       dpi = P(n1-1,n2)*w(1, n1-1,n2)
30     else
31       dpi = 0.0d0
32     endif
33
34     ! Desorption CO
35     if( n1.lt.LL1 ) then
36       dpi = dpi + P(n1+1,n2)*w(2, n1+1,n2)
37     endif
38
39     ! Reaction
40     if( (n1.lt.LL1) .and. (n2.lt.LL2) ) then
41       dum = P(n1+1, n2+1)*w(3,n1+1, n2+1)
42       ProdCO2 = ProdCO2 + dum
43       dpi = dpi + dum
44     endif
45
46     ! Adsorption O2
47     if( n2.ge.2 ) then
48       dpi = dpi + P(n1,n2-2)*w(4,n1,n2-2)
49     endif
50
51     ! dP/dt - Master Equation
52     dP(n1,n2) = dpi - P(n1,n2)*( w(1,n1,n2) + w(2,n1,n2) + w(3,n1,n2) + w(4,n1,n2) )
53   enddo
54 enddo
55
56 ! Update Proba
57 do n2=0,LL2
58   do n1=0,LL1
59     P(n1,n2) = P(n1,n2) + dt*dP(n1,n2)
60     if( P(n1,n2) .lt. 0.0d0 ) then
61       write(*,*) 'ERR: divergence'
62       stop
63     endif
64
65     sp = sp + P(n1,n2)
66   enddo
67 enddo
68 ! Normalisation
69 P(:, :) = P(:, :)/sp
70
71 end subroutine Euler

```



## A.3.2 Dickman algorithm

dm.f and the subroutine TransRate (in rc.f) allow fast convergence to the probability distribution at steady state using the algorithm proposed by Dickman in [113]. The output is given in coverage space in SDM\_Prob.txt.

```

1      program SDickman
2      !
3      ! Simplified Dickman 1st order - Pd
4      !
5      ! Numerical integration of the Master Equation 2D
6      !           gives the Steady State distribution
7      !
8      ! Creation Date      21.07.03 12:59:22
9      ! Last Modification  08.08.03 16:41:03
10     ! Author             Mathias Laurin
11     !
12     implicit none
13     integer*4, parameter :: LL = 2000
14     integer*4, parameter :: LL1=LL/2, LL2=LL/4
15     integer*4             :: k
16     integer*4            :: n1, n2
17     real*8               :: c1, c2
18     real*8               :: rLL, rLL1, rLL2
19     real*8               :: sp, err, sw
20     real*8, dimension(4, 0:LL1, 0:LL2) :: W
21     real*8, dimension(0:LL1, 0:LL2)   :: Proba, r
22
23     !!!           Parameters
24     rLL = real(LL)
25     rLL1 = real(LL1)
26     rLL2 = real(LL2)
27
28     !!!           Init Proba
29     proba(:, :) = 0.0d0
30     proba(LL1/2, LL2/2) = 1.0d0
31
32     !!!           Transition Rate
33     call TransRate(LL, LL1, LL2, 0.54d0, 415d0, W)
34     k = 0
35
36     !!! MAIN
37     do
38         do n2=0, LL2
39             do n1=0, LL1
40                 if( n1.ge.1 ) then
41                     r(n1, n2) = proba(n1-1, n2)*W(1, n1-1, n2)
42                 else
43                     r(n1, n2) = 0.0d0
44                 endif
45                 if( n1.lt.LL1 )
46 +                 r(n1, n2) = r(n1, n2) + proba(n1+1, n2)*W(2, n1+1, n2)
47                 if( (n1.lt.LL1) .and. (n2.lt.LL2) )
48 +                 r(n1, n2) = r(n1, n2) + proba(n1+1, n2+1)*W(3, n1+1, n2+1)
49                 if( n2.ge.2 )
50 +                 r(n1, n2) = r(n1, n2) + proba(n1, n2-2)*W(4, n1, n2-2)
51
52                 sw = w(1, n1, n2)+w(2, n1, n2)+w(3, n1, n2)+w(4, n1, n2)
53                 if( sw .gt. 0.0d0 ) r(n1, n2) = r(n1, n2)/sw
54             enddo
55         enddo
56
57         ! Exit
58         err = sum( sum( (r-proba), dim=2), dim=1 )
59         err = err*err
60         k = k+1

```

```

61         if( k .eq. 100 ) then
62             write(99,*) err
63             k=0
64         endif
65         if(err .lt. 1.0d-7 ) exit
66
67         ! Update proba
68         proba(:, :) = r
69     enddo
70     !!! END MAIN
71
72     ! Normalisation
73     !Proba(:, :) = Proba / sum( sum(proba, dim=2), dim=1 )
74
75     open(unit=1, file='SDM_Prob.txt', status='replace',
76     +action='write', form='formatted')
77     do n2=0,LL2
78         do n1=0,LL1
79             c1 = real(n1)/rLL1
80             c2 = real(n2)/rLL2
81             write(1,'(2f20.6,es20.8e4)') c1, c2, Proba(n1,n2)    ! P_Prob.txt
82         enddo
83     enddo
84     close(1)
85
86     !!!      EOF
87     write(*,*) 'Simplified Dickman DONE', err
88     stop
89     end program SDickman
90

```

```

1  subroutine TransRate(LL, LL1, LL2, xco, T, W)
2  !
3  ! Returns the matrices of transition rates
4  ! to work with KMC.f, Euler.f, Dickman.f
5  !
6  ! f90 -O3 -c TransRate.f
7  ! f90 -O3 -o prog prog.f TransRate.o
8  !
9  ! Creation Date      28.07.03
10 ! Last modification  30.07.03 18:28:04
11 ! Author             Mathias Laurin
12 !
13 implicit none
14 integer*4 :: n1, n2, LL, LL1, LL2
15 real*8    :: r1, r2, c1, c2, rLL, rLL1, rLL2
16 real*8    :: nu_d, en_d, nu_r, en_r, e1, a_d
17 real*8    :: T, rt
18 real*8    :: Ftot, Ptot
19 real*8    :: kb, pi, Tvel, R, Na
20 real*8    :: mco, mo2, nPd, xco
21 real*8    :: sCO, sO2, Ct
22 real*8    :: d1, p1, p2, kr
23 real*8    :: dum
24 real*8, dimension(4, 0:LL1, 0:LL2) :: W
25
26 rLL = real(LL)
27 rLL1 = real(LL1)
28 rLL2 = real(LL2)
29
30 ! General
31 !Ptot = 1.0d-4          ![Torr]
32 !Ptot = Ptot/760.0d0*101.325d0 ![Pa = kg.m-1.s-2]
33 Ptot = 1.0d-4          ![Pa]
34 ! T = 400.0d0          ![K]
35 ! xco = 0.47d0

```

```

36
37      ! Constants
38      kb = 1.380658d-23      ![J/K]      !! [J] == [Kg.m2.s-2]
39      Na = 6.0221367d23      ![mol-1]
40      R = Na*kb              ![J.mol-1.K-1] !! [Kg.m2.s-2.mol-1.K-1]
41      pi = 4.0d0*datan(1.0d0)
42
43      nPd = 1.53d15           ![cm-2]
44      nPd = nPd*1.0d4         ![m-2]
45      mco = 28.010d-3/Na      ![kg]
46      mo2 = 31.999d-3/Na     ![kg]
47      Tvel = 300.0d0         ![K]
48
49      RT = R*T                ![J.mol-1]
50
51      ! Desorption
52      nu_d = 4.0d14           ![s-1]
53      en_d = 142.0d3         ![J.mol-1]
54      a_d = 0.12
55      ! Reaction
56      nu_r = 5.0d7           ![s-1]
57      en_r = 53.0d3         ![J.mol-1]
58      ! Adsorption CO
59      sCO = 0.7d0
60      Ct = 0.3d0
61      ! Adsorption O2
62      sO2 = 1.0d0 - 7.4d-4*T
63
64      Ftot = xco*dsqrt( 2.0d0*pi*mco*kb*Tvel ) ![Kg.m.s-1]
65      Ftot = Ftot + (1.0d0-xco)*dsqrt( 2.0d0*pi*mo2*kb*Tvel )
66      Ftot = Ptot/Ftot      ![m-2.s-1]
67
68      ! Rate constants
69      kr = nu_r*dexp(-en_r/rt) ![s-1]
70      p1 = xco*Ftot/nPd      ![s-1]
71      p2 = (1.0d0-xco)*Ftot/nPd ![s-1]
72
73      !!!      Transition Rates (W)
74      !      from J Chem Phys 114 4669 + Jens' Thesis
75      do n2 = 0, LL2
76          do n1 = 0,LL1
77              ! absolute coverages
78              c1 = real(n1)/rLL
79              c2 = real(n2)/rLL
80              ! relative coverages
81              r1 = real(n1)/rLL1
82              r2 = real(n2)/rLL2
83
84              ! Adsorption CO
85              dum = 1.0d0 - r1 - Ct *r2
86              if( dum.ge.0.0d0 ) then
87                  W(1,n1,n2) = p1*sCO*dum
88              else
89                  W(1,n1,n2) = 0.0d0
90              endif
91
92              ! Desorption CO
93              e1 = en_d*(1.0d0 - a_d*(r1))
94              d1 = nu_d*dexp( -e1/rt )
95              W(2,n1,n2) = d1*c1
96
97              ! Reaction
98              W(3,n1,n2) = Kr*c1*c2
99
100             ! Adsorption O2
101             dum = 1.0d0 - r1 - r2
102             if( dum.gt.0.0d0 ) then

```

```

103         W(4,n1,n2) = p2*sO2* dum*dum
104     else
105         W(4,n1,n2) = 0.0d0
106     endif
107     enddo
108 enddo
109
110 W(:, :, :) = rLL*W(:, :, :)
111
112     !!! Info file
113     open(unit=1, file='Info.txt', status='replace',
114 +action='write', form='formatted')
115     write(1, '(a)')           'Parameters:'
116     write(1, '(i4,a20)') LL, ' adsorption sites'
117     write(1, '(a)')           'Constants:'
118     write(1, '(a8,es10.4)') 'Ptot ', Ptot
119     write(1, '(a8,f6.0)')   'T ', T
120     write(1, '(a8,es10.4)') 'Ftot ', Ftot
121     write(1, '(a8,f6.3)')   'x(CO)', xco
122     write(1, '(a)')           'Desorption'
123     write(1, '(a8,es10.4)') 'nu ', nu_d
124     write(1, '(a8,es10.4)') 'en ', en_d
125     write(1, '(a8,f6.4)')   'cov dep', a_d
126     write(1, '(a)')           'Reaction'
127     write(1, '(a8,es10.4)') 'nu ', nu_r
128     write(1, '(a8,es10.4)') 'en ', en_r
129     write(1, '(a)')           'Adsorption CO'
130     write(1, '(a8,es10.4)') 'nPd ', nPd
131     write(1, '(a8,f6.4)')   'C(T) ', Ct
132     write(1, '(a8,f6.4)')   'S(CO)', sCO
133     write(1, '(a)')           'Adsorption O2'
134     write(1, '(a8,f6.4)')   'S(O2)', sO2
135     write(1, *)
136     write(1, '(a8,es12.4e2)') 'p(CO)', p1
137     write(1, '(a8,es12.4e2)') 'p(O2)', p2
138     write(1, '(a8,es12.4e2)') 'Kr ', Kr
139
140     close(1)
141     end subroutine TransRate

```

### A.3.3 Monte Carlo algorithm

KMC.f90 and the subroutine TransRate (in rc.f) perform an integration of the master equation in time using a Monte Carlo algorithm.

Output file	Description
info.txt	A small summary of the parameters used
KMC.Pop.txt	CO and O population
KMC.ProbaCO.txt	Probability distribution in CO coverage space
KMC.ProbaO.txt	Probability distribution in O coverage space
KMC.CO2.txt	CO <sub>2</sub> production

```

1  program MonteCarlo
2  !
3  ! Creation Date      02.07.03
4  ! Last Modification  24.09.2003 12:26:15
5  ! Author            Mathias Laurin
6  !
7  implicit none
8  integer*4, parameter :: LL = 2000

```

```

9  ! big part  LL = 540
10 ! small part LL = 60
11 integer*4, parameter :: LL1=LL/2, LL2=LL/4
12 integer*4           :: i, j, k, m, n
13 integer*4           :: n1, n2, k1, k2
14 real*8              :: rLL, rLL1, rLL2
15 real*8              :: c1, c2, r1, r2
16 real*8              :: c1a, c1b
17 real*8              :: c2a, c2b, eps, def, mdef
18 real*8              :: T, xco, RT
19 real*8              :: rand, rn, sp, dt
20 real*8              :: Time, AbsTime, TimeMx, dt
21 real*8              :: CO2
22 real*8, dimension(4, 0:LL1, 0:LL2) :: S
23 real*8, dimension(4)                :: s1, s2
24 real*8, dimension(0:LL1, 0:LL2)    :: W3
25 real*8, dimension(0:LL1)           :: pr1
26 real*8, dimension(0:LL2)           :: pr2
27
28 real*8 :: dum
29
30 def = 00.0d-2
31 mdef = 1.0d0 - def
32 T = 415.0d0
33 xco = 0.48d0
34
35 RT = (1.380658d-23*6.0221367d23)*T
36 eps = dexp( (117.0d3 - 142.0d3) /RT )
37
38 !!!           Parameters
39 if( LL .eq. 540 ) then
40     timemx = 250.0d3
41     dt = 1.0d2
42 elseif ( LL .eq. 60 ) then
43     timemx = 300.0d0
44     dt = 1.0d-2
45 elseif( LL .eq. 2000 ) then
46     timemx = 500.0d3
47     dt = 1.0d2
48 else
49     stop
50 endif
51
52 rLL = real(LL)
53 rLL1 = real(LL1)
54 rLL2 = real(LL2)
55
56 ! Initial Coverages
57 n1 = LL1
58 n2 = 0
59 if( n1.gt.LL1 ) write(*,*) 'ERR: Coverage of CO too large'
60 if( n2.gt.LL2 ) write(*,*) 'ERR: Coverage of O2 too large'
61
62 ! nullify
63 Pr1(:) = 0.0d0
64 Pr2(:) = 0.0d0
65 Time = 0.0d0
66 AbsTime = 0.0d0
67 CO2 = 0.0d0
68
69 do k1 = 0, LL1
70     c1 = real(k1)/rLL
71     call equilibrium(c1, mdef, eps, c1a, c1b)
72     do k2 = 0, LL2
73         c2 = real(k2)/rLL
74         c2a = mdef*c2
75         c2b = def*c2

```

```

76
77     call TransRate(LL, c1a, c2a, xco, T, 1, s1)
78     call TransRate(LL, c1b, c2b, xco, T, 2, s2)
79
80     S(:,k1,k2) = mdef*S1(:) + def*S2(:)
81     enddo
82 enddo
83
84 W3(:,:) = s(3,,:)
85
86 ! Summation
87 s(2,,:) = s(1,,:) + s(2,,:)
88 s(3,,:) = s(2,,:) + s(3,,:)
89 s(4,,:) = s(3,,:) + s(4,,:)
90
91 do j=0,LL2
92     do i=0,LL1
93         if( s(4,i,j) .eq. 0.0d0 ) then
94             write(*,*) 'KMC FAILED'
95             stop
96         endif
97     enddo
98 enddo
99
100 ! Normalisation
101 s(4,,:) = 1.0d0/s(4,,:)
102 s(1,,:) = s(1,,:)*s(4,,:)
103 s(2,,:) = s(2,,:)*s(4,,:)
104 s(3,,:) = s(3,,:)*s(4,,:)
105
106 !!!           Files
107 open(unit=1, file='KMC_Pop.txt', status='replace', action='write', form='formatted')
108 open(unit=2, file='KMC_ProbaCO.txt', status='replace', action='write', form='formatted')
109 open(unit=3, file='KMC_ProbaO.txt', status='replace', action='write', form='formatted')
110 open(unit=4, file='KMC_CO2.txt', status='replace', action='write', form='formatted')
111
112 write(1,'(3a8)') 'Time','CO','O'
113
114 open(unit=33, file='Info.txt', status='replace', action='write', form='formatted', position='append')
115 write(33,*)
116 write(33,'(a8)')      'Ini.Cov.'
117 write(33,'(a8,f6.4)') 'CO ',real(n1)/rLL
118 write(33,'(a8,f6.4)') 'O ',real(n2)/rLL
119 write(33,'(a8)')      'Time'
120 write(33,'(a8,f12.0)') 'TimeMax',timemx
121 write(33,'(a8,f12.0)') 'TimeStep',dt
122 write(33,'(a8,f12.0)') 'T',T
123 write(33,'(a8,f6.4)') 'xco',xco
124 write(33,'(a8,f6.4)') 'def',def
125 close(33)
126
127 !!! MAIN
128 do
129     Time = Time + S(4,n1,n2)
130     if( Time .ge. dt ) then
131         AbsTime = AbsTime + Time
132         Time     = 0.0d0
133         write(1, '(es12.6,2f12.8)') AbsTime, real(n1)/rLL, real(n2)/rLL
134         write(4, '(2es20.6)')      AbsTime, CO2
135         CO2 = 0.0d0
136     endif
137     if( AbsTime .ge. TimeMx ) exit
138
139     rn = rand()
140
141     ! Adsorption CO
142     if( rn .le. S(1,n1,n2) ) then

```

```

143     n1      = n1+1
144     Pr1(n1) = pr1(n1) + S(4,n1,n2)
145     Pr2(n2) = pr2(n2) + S(4,n1,n2)
146
147     ! Desorption CO
148     elseif( rn .le. S(2,n1,n2) ) then
149         n1      = n1-1
150         Pr1(n1) = pr1(n1) + S(4,n1,n2)
151         Pr2(n2) = pr2(n2) + S(4,n1,n2)
152
153     ! Reaction
154     elseif( rn .le. S(3,n1,n2) ) then
155         n1      = n1-1
156         n2      = n2-1
157         Pr1(n1) = pr1(n1) + S(4,n1,n2)
158         Pr2(n2) = pr2(n2) + S(4,n1,n2)
159
160     ! Adsorption O2
161     else
162         n2      = n2+2
163         Pr1(n1) = pr1(n1) + S(4,n1,n2)
164         Pr2(n2) = pr2(n2) + S(4,n1,n2)
165     endif
166     CO2      = CO2 + W3(n1,n2)
167 enddo
168 !!! END MAIN
169
170 ! Normalisation
171 pr1(:) = pr1/sum(pr1)
172 pr2(:) = pr2/sum(pr2)
173
174 ! Save probabilities
175 write(2,'(2a8)') 'CO', 'P.CO'
176 write(3,'(2a8)') 'O', 'P.O'
177 do i=0,LL1
178     c1 = real(i)/rLL1
179     ! KMC_ProbaCO.txt
180     write(2,'(f12.8, es20.6e4)') c1, Pr1(i)
181 enddo
182 do j=0,LL2
183     c2 = real(j)/rLL2
184     ! KMC_ProbaO.txt
185     write(3,'(f12.8, es20.6e4)') c2, Pr2(j)
186 enddo
187
188 !!!      EOF
189 write(*,*) 'KMC DONE'
190 close(1)
191 close(2)
192 close(3)
193 stop
194 end
195

```