Appendix A

Programs

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

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 $T = 400\,\mathrm{K}$, $x_{\mathrm{CO}} = 0.35$ and $\chi = 0.10$.

```
// MEAN FIELD CO OXIDATION
// HOMOGENEOUS SURFACE MODEL

// kinetikDlg.h : Header-Datei
//

#if !defined(AFX_KINETIKDLG_H__4EDC3193_8978_11D4_8535_00B0D0159F30__INCLUDED_)
#define AFX_KINETIKDLG_H__4EDC3193_8978_11D4_8535_00B0D0159F30__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CKinetikDlg Dialogfeld
#include "math.h"


// Global constants
const double    k      = 1.380658e-23;
const double    Navo  = 6.0221367e23;
const double    R      = Navo*k;
const double    TO2      = 300.0;
const double    mo2      = 31.999e-3/Navo;
const double    mco      = 28.010e-3/Navo;
```

[1]Microsoft Visual C++, ©1994–98 Microsoft Corporation

[2]©1984–2002 The MathWorks, Inc.

```
const double    maxco = 0.5;
const double    maxo  = 0.25;
const double    M_PI  = 4.0*atan(1.0);

class CKinetikDlg : public CDialog
{
// Konstruktion
public:
    CKinetikDlg(CWnd* pParent = NULL);    // Standard-Konstruktor
    int  CKinetikDlg::Heterogeneous();
    int  CKinetikDlg::Integration();
    int  CKinetikDlg::Deriv( int n );
    void CKinetikDlg::new_T( double d );
    void CKinetikDlg::new_xco( double d );
    void CKinetikDlg::HelpVariable();
    void CKinetikDlg::FileHeader();
    void CKinetikDlg::ErrorVerbose( int n );
// Dialogfelddaten
    //{{AFX_DATA(CKinetikDlg)
    enum { IDD = IDD_KINETIK_DIALOG };
    CButton    m_BtnHomogeneous;
    CButton    m_BtnCoupling;
    double    m_TempStart;
    double    m_TempNumber;
    double    m_TempStep;
    double    m_partco0;
    int        m_xcoStep;
    CString    m_MeldungText;
    int        m_after;
    double    m_Npd;
    double    m_Ptot;
    double    m_vdes0;
    double    m_vdes1;
    double    m_vlh0;
    double    m_vlh1;
    int        m_beamon;
    BOOL    m_bCoupling;
    double    m_Edes0;
    double    m_Edes1;
    double    m_Elh0;
    double    m_Elh1;
    double    m_xcoexact;
    double    m_Texact;
    double    m_Codes;
    double    m_tStep;
    CString    m_Schritt;
    double    m_Ct;
    BOOL    m_bHomogeneous;
    int        m_exp0ad;
    double    m_sCO;
    double    m_sO2;
    double    m_Elhtheta;
    BOOL    m_bTransient;
    double    m_RDSStep;
    int        m_resolution;
    int        m_Ts;
    //}}AFX_DATA

    // Vom Klassenassistenten generierte berladungen virtueller Funktionen
    //{{AFX_VIRTUAL(CKinetikDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV-Untersttzung
    //}}AFX_VIRTUAL

// Implementierung
protected:
    HICON m_hIcon;

    // Generierte Message-Map-Funktionen
    //{{AFX_MSG(CKinetikDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnChangeEDITAnzahl();
    afx_msg void OnChangeEDITafter();
    afx_msg void OnBtnAllTransient();
    afx_msg void OnBtnSS();
    afx_msg void OnBtnDataOneTransient();
    afx_msg void OnChkCoupling();
    afx_msg void OnBtnInit();
    afx_msg void OnImgMinerva();
    afx_msg void OnBtnFit();
    afx_msg void OnChangeEditafter();
    afx_msg void OnChkTransient();
    afx_msg void OnBtnRDS();
    afx_msg void OnBtnSSCov();
    afx_msg void OnBtnRegion();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ fgt unmittelbar vor der vorhergehenden Zeile zustzliche Deklarationen ein.

#endif // !defined(AFX_KINETIKDLG_H__4EDC3193_8978_11D4_8535_00B0D0159F30__INCLUDED_)
```

```
/**************************************************************************
**                                                                        **
**    MEAN FIELD CO OXIDATION BISTABILITY                                    **
**                                                                        **
**    HETEROGENEOUS SURFACE MODEL                                            **
**                                                                        **
**    TRANSIENTS  STEADY-STATE  RAMPS  RATE-DETERMINING STEP  COVERAGES    **
**                                                                        **
**                                                                        **
**  Modified model (diffusion of oxygen)                                **
**                                                                        **
**    Author:                JENS HOFFMANN                                  **
**    Continued by:        MATHIAS LAURIN                                **
**                                                                        **
**    LAST MODIFICATION:    21.07.04    16:56:10                            **
**                                                                        **
**                                                                        **
**************************************************************************/

// kinetikDlg.cpp

#include "stdafx.h"
#include "kinetik.h"
#include "kinetikDlg.h"
#include "math.h"
#include "stdio.h"
#include "string.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////////
```

```
// CAboutDlg-Dialogfeld fr Anwendungsbefehl "Info"

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialogfelddaten
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA
//    OnBtnSS();
//    OnBtnAllTransient();
//    OnBtnDataOneTransient();
    // Vom Klassenassistenten generierte berladungen virtueller Funktionen
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV-Untersttzung
    //}}AFX_VIRTUAL

// Implementierung
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

/////////////////////////////////////////////////////////////////////////////
// CKinetikDlg Nachrichten-Handler

BOOL CKinetikDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // IDM_ABOUTBOX muss sich im Bereich der Systembefehle befinden.


    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu( false );
    if (pSysMenu != NULL) {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty()) {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }


    // Symbol fr dieses Dialogfeld festlegen. Wird automatisch erledigt
    //  wenn das Hauptfenster der Anwendung kein Dialogfeld ist
    SetIcon(m_hIcon,  true );            // Groes Symbol verwenden
    SetIcon(m_hIcon,  false );        // Kleines Symbol verwenden

    // ZU ERLEDIGEN: Hier zustzliche Initialisierung einfgen

    return  true ;  // Geben Sie  true  zurck, auer ein Steuerelement soll den Fokus erhalten
```

```
}

////////////////////////////////////////////////////////////////////////////

void CKinetikDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX) {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
        CDialog::OnSysCommand(nID, lParam);
}

////////////////////////////////////////////////////////////////////////////
//         CONSTANTS
///////////////////////////////////////////////////////////////////////////

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // Keine Nachrichten-Handler
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
//         VARIABLES
///////////////////////////////////////////////////////////////////////////

CKinetikDlg::CKinetikDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CKinetikDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CKinetikDlg)
    m_TempStart = 400.0;
    m_TempNumber = 1.0;
    m_TempStep = 15.0;
    m_partco0 = 99;
    m_xcoStep = 50;
    m_after = 20;
    m_Npd = 1.53e+19;
    m_Ptot = 1.e-4;
    m_vdes0 = 4.e+14;
    m_vdes1 =4.e+14;
    m_vlh0 = 5.e+7;
    m_vlh1 = 5.e+7;
    m_beamon = 500;
    m_bCoupling =  true ;
    m_Edes0 = 142.0;
    m_Edes1 = 117.0;
    m_Elh0 = 53.0;
    m_Elh1 = 44.0;
    m_xcoexact = 50.00;
    m_Texact = 415;
    m_Codes = 12;
    m_tStep = 0.1;
    m_Schritt = _T("0.1000");
    m_Ct = 30;
    m_exp0ad = 2;
    m_sCO = 70.0;
    m_sO2 = 100.0;
    m_Elhtheta = 0.0;
    m_bTransient =  false ;
    m_RDSStep = 1.0;
    m_resolution = 10;
    m_Ts = 0;
    //}}AFX_DATA_INIT
    // Beachten Sie, dass LoadIcon unter Win32 keinen nachfolgenden DestroyIcon-Aufruf bentigt
    m_hIcon = AfxGetApp()->LoadIcon(IDI_Kinetik);
}

// Global variables
double        ko, kco, dt;
double        kdes[2], Edes[2], vdes[2];
double        klh[2],  Elh[2],  vlh[2];
double        co[2], o[2], dO[2], dCO[2];
double        prod[2], totprod, RTs, mmax;
double        p[2];
double        sO2, sCO, ElhTheta, COdes, Ct;
int            tmax, tafter;
double        eps;

FILE        *filename;

/////////////////////////////////////////////////////////////////////////////
//        PROCEDURES DEFINITION
////////////////////////////////////////////////////////////////////////////

void CKinetikDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CKinetikDlg)
    DDX_Control(pDX, IDC_ChkCoupling, m_BtnCoupling);
    DDX_Text(pDX, IDC_EDIT_TempStart, m_TempStart);
    DDV_MinMaxDouble(pDX, m_TempStart, 300., 600.);
    DDX_Text(pDX, IDC_EDIT_TempNumber, m_TempNumber);
    DDV_MinMaxDouble(pDX, m_TempNumber, 1., 200.);
    DDX_Text(pDX, IDC_EDIT_TempStep, m_TempStep);
    DDV_MinMaxDouble(pDX, m_TempStep, 1., 50.);
    DDX_Text(pDX, IDC_EDIT_partco1, m_partco0);
    DDV_MinMaxDouble(pDX, m_partco0, 0., 100.);
    DDX_Text(pDX, IDC_EDIT_xcoStep, m_xcoStep);
    DDV_MinMaxInt(pDX, m_xcoStep, 1, 200);
    DDX_Text(pDX, IDC_EDIT_after, m_after);
    DDV_MinMaxInt(pDX, m_after, 1, 200);
    DDX_Text(pDX, IDC_EDIT_Npd, m_Npd);
    DDV_MinMaxDouble(pDX, m_Npd, 1.e+018, 3.e+019);
    DDX_Text(pDX, IDC_EDIT_Ptot, m_Ptot);
    DDV_MinMaxDouble(pDX, m_Ptot, 1.e-007, 1.);
    DDX_Text(pDX, IDC_EDIT_vdes1, m_vdes0);
    DDV_MinMaxDouble(pDX, m_vdes0, 1., 1.e+030);
    DDX_Text(pDX, IDC_EDIT_vdes2, m_vdes1);
    DDV_MinMaxDouble(pDX, m_vdes1, 1., 1.e+030);
    DDX_Text(pDX, IDC_EDIT_vlh1, m_vlh0);
    DDV_MinMaxDouble(pDX, m_vlh0, 1., 1.e+030);
    DDX_Text(pDX, IDC_EDIT_vlh2, m_vlh1);
    DDV_MinMaxDouble(pDX, m_vlh1, 1., 1.e+030);
    DDX_Text(pDX, IDC_EDIT_beamon, m_beamon);
    DDV_MinMaxInt(pDX, m_beamon, 1, 50000);
    DDX_Check(pDX, IDC_ChkCoupling, m_bCoupling);
    DDX_Text(pDX, IDC_EDIT_Edes1, m_Edes0);
    DDV_MinMaxDouble(pDX, m_Edes0, 50., 250.);
    DDX_Text(pDX, IDC_EDIT_Edes2, m_Edes1);
    DDV_MinMaxDouble(pDX, m_Edes1, 50., 250.);
    DDX_Text(pDX, IDC_EDIT_Elh1, m_Elh0);
    DDV_MinMaxDouble(pDX, m_Elh0, 30., 250.);
    DDX_Text(pDX, IDC_EDIT_Elh2, m_Elh1);
    DDV_MinMaxDouble(pDX, m_Elh1, 30., 250.);
    DDX_Text(pDX, IDC_EDIT_OneTransient_xco, m_xcoexact);
    DDV_MinMaxDouble(pDX, m_xcoexact, 0., 100.);
    DDX_Text(pDX, IDC_EDIT_OneTransient_T, m_Texact);
    DDV_MinMaxDouble(pDX, m_Texact, 350., 550.);
    DDX_Text(pDX, IDC_EDIT_COdes, m_Codes);
    DDV_MinMaxDouble(pDX, m_Codes, 0., 100.);
    DDX_Text(pDX, IDC_EDIT_dt, m_tStep);
    DDV_MinMaxDouble(pDX, m_tStep, 1.0e-006, 1.0e-001);
    DDX_Text(pDX, IDC_Schritt, m_Schritt);
    DDX_Text(pDX, IDC_EDIT_Ct, m_Ct);
    DDV_MinMaxDouble(pDX, m_Ct, 0., 100.);
    DDX_Text(pDX, IDC_EDIT_expOad, m_expOad);
    DDV_MinMaxInt(pDX, m_expOad, 1, 15);
    DDX_Text(pDX, IDC_EDIT_sCO, m_sCO);
    DDV_MinMaxDouble(pDX, m_sCO, 0., 1000.);
    DDX_Text(pDX, IDC_EDIT_sO2, m_sO2);
    DDV_MinMaxDouble(pDX, m_sO2, 0., 1000.);
    DDX_Text(pDX, IDC_EDIT_Elhtheta, m_Elhtheta);
    DDV_MinMaxDouble(pDX, m_Elhtheta, -100., 100.);
    DDX_Check(pDX, IDC_ChkTransientFit, m_bTransient);
    DDX_Text(pDX, IDC_EDIT_RDSStep, m_RDSStep);
    DDV_MinMaxDouble(pDX, m_RDSStep, 0., 100.);
    DDX_Text(pDX, IDC_EDIT_Resolution, m_resolution);
    DDV_MinMaxInt(pDX, m_resolution, 1, 1000);
    DDX_Text(pDX, IDC_EDIT_Ts, m_Ts);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CKinetikDlg, CDialog)
    //{{AFX_MSG_MAP(CKinetikDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_EN_KILLFOCUS(IDC_EDIT_Ptot, OnChangeEDITafter)
    ON_BN_CLICKED(IDC_BtnAllTransient, OnBtnAllTransient)
    ON_BN_CLICKED(IDC_BtnSS, OnBtnSS)
    ON_BN_CLICKED(IDC_BtnDataOneTransient, OnBtnDataOneTransient)
    ON_BN_CLICKED(IDC_ChkCoupling, OnChkCoupling)
    ON_BN_CLICKED(IDC_BtnInit, OnBtnInit)
    ON_BN_CLICKED(IDB_BITMAP1, OnImgMinerva)
    ON_BN_CLICKED(IDC_BtnFit, OnBtnFit)
    ON_BN_CLICKED(IDC_ChkTransientFit, OnChkTransient)
    ON_BN_CLICKED(IDC_BtnRDS, OnBtnRDS)
    ON_BN_CLICKED(IDC_BtnSSCov, OnBtnSSCov)
    ON_BN_CLICKED(IDC_BtnRegion, OnBtnRegion)
    ON_EN_KILLFOCUS(IDC_EDIT_after, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_Edes1, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_Elh1, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_vdes1, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_vdes2, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_vlh2, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_beamon, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_vlh1, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_dt, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_Edes2, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_Elh2, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_Npd, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_partco1, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_TempStart, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_TempNumber, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_TempStep, OnChangeEDITafter)
    ON_BN_KILLFOCUS(IDC_ChkCoupling, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_OneTransient_T, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_OneTransient_xco, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_COdes, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_Ct, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_sCO, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_sO2, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_expOad, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_Elhtheta, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_RDSStep, OnChangeEDITafter)
    ON_EN_KILLFOCUS(IDC_EDIT_Resolution, OnChangeEDITafter)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////

void CKinetikDlg::OnPaint()
{
    if (IsIconic()) {
        CPaintDC dc(this); // Gertekontext fr Zeichnen

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Symbol in Client-Rechteck zentrieren
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Symbol zeichnen
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
        CDialog::OnPaint();
}


HCURSOR CKinetikDlg::OnQueryDragIcon()
{ return (HCURSOR) m_hIcon; }


/////////////////////////////////////////////////////////////////////////////

void CKinetikDlg::OnImgMinerva()
{
    MessageBox("- Fritz Haber Institut der MPG - \nAbpartung Chemische Physik\nFaradayweg 4-6\n14195 Berlin\n+49 (0)30 8413
    /*
    - Fritz Haber Institut der MPG -
    Abpartung Chemische Physik
    Faradayweg 4-6
    14195 Berlin
    +49 (0)30 8413 4309

    http://www.fhi-berlin.mpg.de/cp/mb/mb.html
    */
}


////////////////////////////////////////////////////////////////////////////
// OUTPUT FILE HEADER
////////////////////////////////////////////////////////////////////////////

void CKinetikDlg::FileHeader()
{
    int temp;
    fprintf(filename,"PARAMETER:\nEdes1: %.2f",m_Edes0);
    fprintf(filename," kJ/mol\nvdes1: %e",m_vdes0);
    fprintf(filename," s-1\nEdes2: %.2f",m_Edes1);
    fprintf(filename," kJ/mol\nvdes2: %e",m_vdes1);
    fprintf(filename,"s-1\nElh1: %.2f",m_Elh0);
    fprintf(filename," kJ/mol\nvlh1: %e",m_vlh0);
    fprintf(filename," s-1\nElh2: %.2f",m_Elh1);
    fprintf(filename," kJ/mol\nvlh2: %e",m_vlh1);
    fprintf(filename," s-1\nfraction Facet 1: %.2f",m_partco0);
    fprintf(filename," percent\ncoverage Pd surface atoms: %e",m_Npd);
    fprintf(filename," m-2\ntotal pressure: %.5f",m_Ptot);
    fprintf(filename," Pa\ncoupling of facets\t");
    if (m_bCoupling)
        fprintf(filename,"on\n");
    else
        fprintf(filename,"off\n");
    fprintf(filename,"\nExp. O-Diss.: %.i",m_expOad);
    fprintf(filename,"\ninitial stick. coeff. CO: %.2f",m_sCO);
    fprintf(filename," percent\ninitial stick. coeff. O: %.2f",m_sO2);
    fprintf(filename," percent\nTheta dep. reaction: %.2f",m_Elhtheta);
    fprintf(filename," percent\nTheta dep. CO-desorption: %.2f",m_Codes);
    fprintf(filename," percent\nTheta dep. CO-adsorption: %.2f",m_Ct);
    fprintf(filename," percent\nCO-fraction in %.2f",100.0/double(m_xcoStep));
    fprintf(filename,"-percent steps\nTemperatures: ");
    for (temp=0; temp<m_TempNumber; temp++) {
        fprintf(filename,"%i",(temp)*m_TempStep+m_TempStart);
        fprintf(filename,"K  ");
    }
    fprintf(filename,"\nstep size numerical Integration: %.3f",dt);
    fprintf(filename,"\n");
}


/////////////////////////////////////////////////////////////////////////////
// THINGS TO DO IF A EDIT FIELD IS LEFT
////////////////////////////////////////////////////////////////////////////

void CKinetikDlg::OnChangeEDITafter()
{
    UpdateData( true );
    m_Schritt.Format("%.3g",m_tStep);
    SetDlgItemText(IDC_Schritt,m_Schritt);
    HelpVariable();
    UpdateData( false );
}


/////////////////////////////////////////////////////////////////////////////
// INITIALIZE ALL VARIABLES
////////////////////////////////////////////////////////////////////////////

void CKinetikDlg::OnBtnInit()
{
    m_TempStart    = 400.0;
    m_TempNumber= 1.0;
    m_TempStep    = 15.0;
    m_partco0    = 99;
    m_xcoStep    = 50;
    m_after        = 20;
    m_Npd        = 1.53e+19;
    m_Ptot        = 1.e-4;
    m_vdes0        = 4.e+14;
    m_vdes1        = 4.e+14;
    m_vlh0        = 5.e+7;
    m_vlh1        = 5.e+7;
    m_beamon    = 500;
    m_bCoupling = true;
    m_Edes0        = 142.0;
    m_Edes1        = 117.0;
    m_Elh0        = 53.0;
    m_Elh1        = 44.0;
    m_xcoexact    = 50.00;
    m_Texact    = 415;
    m_Codes        = 12;
    m_tStep        = 0.1;
    m_Schritt    = _T("0.1000");
    m_Ct        = 30;
    m_expOad    = 2;
    m_sCO        = 70.0;
    m_sO2        = 100.0;
    m_Elhtheta    = 0.0;
    m_bTransient= false;
    m_RDSStep   = 1.0;
    m_resolution= 10;

    HelpVariable();
    UpdateData( false );
}


//////////////////////////////////////////////////
// CONVERSIONS TO S.I. UNIT                        //
//////////////////////////////////////////////////

void CKinetikDlg::HelpVariable()
{
    dt      = m_tStep;

    vlh[0]    = m_vlh0;
    Elh[0]    = m_Elh0    *1.0e3;
    vdes[0]    = m_vdes0;
    Edes[0]    = m_Edes0    *1.0e3;
    sCO        = m_sCO        *1.0e-2;
    ElhTheta= m_Elhtheta*1.0e-2;
    COdes    = m_Codes    *1.0e-2;
    Ct        = m_Ct        *1.0e-2;

    vlh[1]    = m_vlh1;
    Elh[1]    = m_Elh1    *1.0e3;
    vdes[1]    = m_vdes1;
    Edes[1]    = m_Edes1    *1.0e3;
    eps        = exp( (Edes[1]-Edes[0])/RTs );

    p[0]    = m_partco0 *1.0e-2;
    p[1]    = 1.0-p[0];

    mmax    = maxo*maxco;

    tmax    = int(m_beamon/dt);
    tafter    = int(m_after/dt);
}

//////////////////////////////////////////////////

void CKinetikDlg::OnChkCoupling()
{ m_bCoupling=~m_bCoupling; }

//////////////////////////////////////////////////

void CKinetikDlg::OnChkTransient()
{
    if (m_bTransient)
        m_bTransient = false ;
    else
        m_bTransient = true ;
    UpdateData( false );
}


/************************************************
```

```
**                                              **
**                INTERNAL PROCEDURES              **
**                                              **
************************************************/

////////////////////////////////////////////////
// INTEGERATION OF THE KINETIC EQUATIONS       //
////////////////////////////////////////////////

// Driver for the integration
int CKinetikDlg::Integration()
{
    int ErrCode;
    if( eps == 0.0 ) HelpVariable();

    // Facet 'Perfect'
    if( p[0] > 0.9999 ) {
        ErrCode = Deriv(0);
        prod[0] = klh[0]*o[0]*co[0];
        totprod = prod[0];
    } else {
    // Facet 'Defect'
        Deriv(0);
        Deriv(1);
        ErrCode = Heterogeneous();
        prod[0] = klh[0]*o[0]*co[0];
        prod[1] = klh[1]*o[1]*co[1];
        totprod = p[0]*prod[0] + p[1]*prod[1];
    }
    if( totprod < 0.0 )    ErrCode = 1;

    return ErrCode;
}

// Derivation of the kinetic eq.
int CKinetikDlg::Deriv( int F )
{
    double dO, dCO;
    double Reac;
    double dum;

    //klh[F]    = vlh[F] * exp( -Elh[F]*(1.0 - ElhTheta*co[F])/RTs );
    klh[F]    = vlh[F] * exp( -Elh[F]/RTs );
    Reac    = klh[F]*o[F]*co[F];

    dum        = 1.0-co[F]-o[F];
    if( dum >= 0.0 )
        dum = dum*dum; // dum = pow( dum, m_expOad );
    else
        dum = 0.0;

    dO        = 2.0*ko/maxo*sO2*dum - Reac*maxco;
    kdes[F]    = vdes[F]*exp( -Edes[F]/RTs*( 1.0-COdes*co[F] ) );
    dum        = 1.0 - co[F] - Ct*o[F];
    if( dum < 0.0 )
        dum = 0.0;
    dCO        = sCO*dum*kco/maxco - kdes[F]*co[F] - Reac*maxo;

    o[F]    += dO*dt;
    co[F]    += dCO*dt;

    return 0;
}

// thermodynamical equilibrium for the heterogeneous model
int CKinetikDlg::Heterogeneous()
{
    double cotot, otot;
    double a, b, c;
    double meps = 1.0 - eps;

    cotot    = p[0]*co[0] + p[1]*co[1];
    otot     = p[0]*o[0]  + p[1]*o[1];

    if( p[0] < 0.0001 ) {    // strictly: exclude zero
        co[1] = cotot;
        o[1]  = otot;
        co[0] = o[0] = 0.0;
    } else if( p[1] < 0.0001 ) { // strictly: exclude zero
        co[0] = cotot;
        o[0]  = otot;
        co[1] = o[1] = 0.0;
    } else {
        a    = 0.5/(p[0]*meps);
        b    = cotot*meps + eps + p[0]*meps;
        c    = -4.0*cotot*p[0]*meps;

        c    = b*b + c;
        if( c>=0.0 )
            c = sqrt(c);
        else
            return 2;

        co[0] = a*(b-c);
        if( (co[0] < 0.0) || (co[0] > 1.0) )
            co[0] = a*(b+c);

        co[1]    = (cotot - p[0]*co[0])/p[1];

        // Oxygen //
        o[0]  = p[0]*otot;
        o[1]  = p[1]*otot;

        if( (co[1] < 0.0) || (co[0] < 0.0) )    return 4;
        if( (o[1]  < 0.0) || (o[0]  < 0.0) )    return 8;
        if( (o[1]  > 1.0) || (o[0]  > 1.0) )    return 8;
    }
    return 0;
}


////////////////////////////////////////////////////////////////////////
//               NEW T                                                //
////////////////////////////////////////////////////////////////////////

void CKinetikDlg::new_T(double Ts)
{
    RTs  = R*Ts;
    sO2  = m_sO2*1.0e-2 - 7.4e-4*Ts;
    m_Ts = int( Ts );
    UpdateData( false );
}


////////////////////////////////////////////////////////////////////////
//               NEW x(CO)                                            //
////////////////////////////////////////////////////////////////////////

void CKinetikDlg::new_xco(double xco)
{
    double Ftot;
    double dDum;
    dDum = 2.0*M_PI*k*TO2;
    Ftot = m_Ptot/( xco*sqrt(mco*dDum) + (1.0-xco)*sqrt(mo2*dDum) );
    kco  = xco*Ftot/m_Npd;
    ko   = (1.0-xco)*Ftot/m_Npd;
}


//////////////////////////////////////////////////////////////////////
//    VERBOSE    ON ERROR                            ML 20/08/03            //
//////////////////////////////////////////////////////////////////////

void CKinetikDlg::ErrorVerbose( int ErrorCode )
{
    switch( ErrorCode )
    {
    case 0:
        MessageBox("Satisfactorily finished","Done!");
        break;
    case 1:
        MessageBox("FAILED #1: Divergence - Decrease dt","Error",MB_ICONWARNING);
        break;
    case 2:
        MessageBox("FAILED #2","Error",MB_ICONWARNING);
        break;
    case 4:
        MessageBox("FAILED #4: Divergence - Decrease dt","Error",MB_ICONWARNING);
        break;
    case 8:
        MessageBox("FAILED #8: Divergence - Decrease dt","Error",MB_ICONWARNING);
        break;
    case 64:
        MessageBox("FAILED #64: eps = 0. - Retry","Error",MB_ICONWARNING);
        break;
    case 1024:    // Silent out
        break;
    default:
        MessageBox("FAILED: Unknown error","Error",MB_ICONWARNING);
    }
}


/*************************************************
**                                              **
**              BUTTONS                            **
**                                              **
*************************************************/

//////////////////////////////////////////////////
//              FIT ROUTINE                        //
//////////////////////////////////////////////////

void CKinetikDlg::OnBtnFit()
{
    int        ii, jj;
    int        nTime, n_T, n_xco, nDir;
    int        ErrCode;
    double    fp;
    int        n1,n2,n3,n4,n5,n6,n7,n8,n9,n10;
    int        nStep[11];
    double    Max, diff;
    double    xco, Ts;
    double    StartVal[11],EndVal[11];
    double    mittedata, mittess;
    double    R2rel[2], R2abs[2];
    double    sim[7][20][2];
    double    data[7][20];
    double    trans[4][1891];
    FILE    *stream;

```

```
n1=n2=n3=n4=n5=n6=n7=n8=n9=n10=0; // to avoid error message during compilation

stream    = fopen( "exptrans.txt", "r+" );
   filename= fopen( "parameter.txt","w");

m_TempStart = 400.0;
m_TempStep  = 15.0;
m_TempNumber= 2;
m_xcoStep    = 20;
m_beamon    = 500;
m_after        = 1;
m_bCoupling = true;

UpdateData( false );

fprintf(filename,"vdes1\t vlh1\t Edes1\t Elh1\t expOad\t COdes\t Ct\t sCO\t sO2\t vdes2\t vlh2\t Edes2\t Elh[1]\t part1\
/*
vdes1    vlh1    Edes1    Elh1
expOad    COdes    Ct    sCO    sO2
vdes2    vlh2    Edes2    Elh2
part1    time    scale    R2relCO    R2relO    R2absCO    R2absO
*/
FileHeader();

///////////  EXPERIMENTAL TRANSIENT DATA TAKEN FROM EXPTRANS.TXT  //////////
for( ii = 1; ii <= 1890; ii++ ) {
    fscanf( stream, "%f", &fp );
    trans[1][ii]=fp;
    fscanf( stream, "%f", &fp );
    trans[2][ii]=fp;
}

///////////////   EXPERIMENTAL STEADY STATE DATA    ///////////////////////

///  EBL particles
/*
data[1][2]=0.05;data[2][2]=0.0041;data[3][2]=0.0034;data[4][2]=0.0034;
data[1][3]=0.1;data[2][3]=0.0069;data[3][3]=0.006;data[4][3]=0.0059;
data[1][4]=0.15;data[2][4]=0.0091;data[3][4]=0.0084;data[4][4]=0.0082;
data[1][5]=0.20;data[2][5]=0.0125;data[3][5]=0.0115;data[4][5]=0.0107;
data[1][6]=0.25;data[2][6]=0.015;data[3][6]=0.014;data[4][6]=0.0134;
data[1][7]=0.3;data[2][7]=0.0175;data[3][7]=0.017;data[4][7]=0.0157;
data[1][8]=0.35;data[2][8]=0.021;data[3][8]=0.0198;data[4][8]=0.0182;
data[1][9]=0.4;data[2][9]=0.0235;data[3][9]=0.0229;data[4][9]=0.0215;
data[1][10]=0.45;data[2][10]=0.0259;data[3][10]=0.0263;data[4][10]=0.0241;
data[1][11]=0.5;data[2][11]=0.0082;data[3][11]=0.0289;data[4][11]=0.0277;
data[1][12]=0.55;data[2][12]=0.0052;data[3][12]=0.0166;data[4][12]=0.0301;
data[1][13]=0.6;data[2][13]=0.0037;data[3][13]=0.0087;data[4][13]=0.0206;
data[1][14]=0.65;data[2][14]=0.0033;data[3][14]=0.0071;data[4][14]=0.0143;
data[1][15]=0.7;data[2][15]=0.0027;data[3][15]=0.0059;data[4][15]=0.0114;
data[1][16]=0.75;data[2][16]=0.002;data[3][16]=0.0045;data[4][16]=0.0091;
data[1][17]=0.8;data[2][17]=0.0014;data[3][17]=0.0034;data[4][17]=0.0071;
data[1][18]=0.85;data[2][18]=0.00067;data[3][18]=0.0025;data[4][18]=0.0048;
data[1][19]=0.9;data[2][19]=0.00031;data[3][19]=0.0015;data[4][19]=0.0027;
data[1][20]=0.95;data[2][20]=0.00001;data[3][20]=0.00074;data[4][20]=0.00091;
//bistability//
data[1][22]=0.05;data[2][22]=0.0037;data[3][22]=0.0029;data[4][22]=0.0034;
data[1][23]=0.1;data[2][23]=0.0058;data[3][23]=0.0051;data[4][23]=0.0059;
data[1][24]=0.15;data[2][24]=0.008;data[3][24]=0.0082;data[4][24]=0.0082;
data[1][25]=0.20;data[2][25]=0.0109;data[3][25]=0.0111;data[4][25]=0.0107;
data[1][26]=0.25;data[2][26]=0.0136;data[3][26]=0.0139;data[4][26]=0.0134;
data[1][27]=0.3;data[2][27]=0.0157;data[3][27]=0.0171;data[4][27]=0.0157;
data[1][28]=0.35;data[2][28]=0.0172;data[3][28]=0.02;data[4][28]=0.0182;
data[1][29]=0.4;data[2][29]=0.0074;data[3][29]=0.0231;data[4][29]=0.0215;
data[1][30]=0.45;data[2][30]=0.0057;data[3][30]=0.0197;data[4][30]=0.0241;
data[1][31]=0.5;data[2][31]=0.0044;data[3][31]=0.0109;data[4][31]=0.0277;
data[1][32]=0.55;data[2][32]=0.0037;data[3][32]=0.0084;data[4][32]=0.0301;
```

```
data[1][33]=0.6;data[2][33]=0.0031;data[3][33]=0.0071;data[4][33]=0.0206;
data[1][34]=0.65;data[2][34]=0.0026;data[3][34]=0.0059;data[4][34]=0.0143;
data[1][35]=0.7;data[2][35]=0.0022;data[3][35]=0.0046;data[4][35]=0.0114;
data[1][36]=0.75;data[2][36]=0.0016;data[3][36]=0.0036;data[4][36]=0.0091;
data[1][37]=0.8;data[2][37]=0.0015;data[3][37]=0.0031;data[4][37]=0.0071;
data[1][38]=0.85;data[2][38]=0.00088;data[3][38]=0.0021;data[4][38]=0.0048;
data[1][39]=0.9;data[2][39]=0.00051;data[3][39]=0.0012;data[4][39]=0.0027;
data[1][40]=0.95;data[2][40]=0.00023;data[3][40]=0.00059;data[4][40]=0.00091;

mittedata=0.0259+0.0289+0.0301;//0.0259+0.0289+0.0301;*/

///  big particles (6 nm)

data[1][2]=0.05;data[2][2]=0.0058;data[3][2]=0.0056;data[4][2]=0.0062;
data[1][3]=0.1;data[2][3]=0.0101;data[3][3]=0.0108;data[4][3]=0.0097;
data[1][4]=0.15;data[2][4]=0.0104;data[3][4]=0.0158;data[4][4]=0.0143;
data[1][5]=0.20;data[2][5]=0.006;data[3][5]=0.0196;data[4][5]=0.0197;
data[1][6]=0.25;data[2][6]=0.0042;data[3][6]=0.0137;data[4][6]=0.0241;
data[1][7]=0.3;data[2][7]=0.0034;data[3][7]=0.0084;data[4][7]=0.0292;
data[1][8]=0.35;data[2][8]=0.003;data[3][8]=0.0066;data[4][8]=0.0337;
data[1][9]=0.4;data[2][9]=0.0024;data[3][9]=0.0051;data[4][9]=0.0351;
data[1][10]=0.45;data[2][10]=0.0021;data[3][10]=0.0045;data[4][10]=0.0237;
data[1][11]=0.5;data[2][11]=0.0019;data[3][11]=0.0037;data[4][11]=0.0152;
data[1][12]=0.55;data[2][12]=0.0018;data[3][12]=0.003;data[4][12]=0.0122;
data[1][13]=0.6;data[2][13]=0.0015;data[3][13]=0.0028;data[4][13]=0.0101;
data[1][14]=0.65;data[2][14]=0.0013;data[3][14]=0.0021;data[4][14]=0.0084;
data[1][15]=0.7;data[2][15]=0.0011;data[3][15]=0.0018;data[4][15]=0.0071;
data[1][16]=0.75;data[2][16]=0.00079;data[3][16]=0.0016;data[4][16]=0.0054;
data[1][17]=0.8;data[2][17]=0.00088;data[3][17]=0.0012;data[4][17]=0.0039;
data[1][18]=0.85;data[2][18]=0.00063;data[3][18]=0.00085;data[4][18]=0.003;
data[1][19]=0.9;data[2][19]=0.00039;data[3][19]=0.0005;data[4][19]=0.0021;
data[1][20]=0.95;data[2][20]=0.00016;data[3][20]=0.0002;data[4][20]=0.00078;
//bistability//
data[1][22]=0.05;data[2][22]=0.0049;data[3][22]=0.0045;data[4][22]=0.0053;
data[1][23]=0.1;data[2][23]=0.0094;data[3][23]=0.0105;data[4][23]=0.0098;
data[1][24]=0.15;data[2][24]=0.0121;data[3][24]=0.0155;data[4][24]=0.014;
data[1][25]=0.20;data[2][25]=0.0101;data[3][25]=0.0195;data[4][25]=0.0188;
data[1][26]=0.25;data[2][26]=0.0067;data[3][26]=0.0192;data[4][26]=0.0239;
data[1][27]=0.3;data[2][27]=0.0043;data[3][27]=0.0138;data[4][27]=0.0285;
data[1][28]=0.35;data[2][28]=0.0033;data[3][28]=0.0092;data[4][28]=0.0337;
data[1][29]=0.4;data[2][29]=0.0027;data[3][29]=0.007;data[4][29]=0.036;
data[1][30]=0.45;data[2][30]=0.0023;data[3][30]=0.0051;data[4][30]=0.0286;
data[1][31]=0.5;data[2][31]=0.0018;data[3][31]=0.0041;data[4][31]=0.0168;
data[1][32]=0.55;data[2][32]=0.0019;data[3][32]=0.0035;data[4][32]=0.0123;
data[1][33]=0.6;data[2][33]=0.0015;data[3][33]=0.0025;data[4][33]=0.01;
data[1][34]=0.65;data[2][34]=0.0014;data[3][34]=0.0023;data[4][34]=0.0084;
data[1][35]=0.7;data[2][35]=0.0011;data[3][35]=0.0019;data[4][35]=0.0068;
data[1][36]=0.75;data[2][36]=0.00094;data[3][36]=0.0015;data[4][36]=0.0057;
data[1][37]=0.8;data[2][37]=0.00093;data[3][37]=0.0014;data[4][37]=0.0041;
data[1][38]=0.85;data[2][38]=0.00055;data[3][38]=0.00082;data[4][38]=0.003;
data[1][39]=0.9;data[2][39]=0.00036;data[3][39]=0.00046;data[4][39]=0.0017;
data[1][40]=0.95;data[2][40]=0.00082;data[3][40]=0.00015;data[4][40]=0.00092;

mittedata=0.0104+0.0196;//+0.0351;//0.0104+0.0196+0.0351;

/* m_vdes0 */        StartVal[1] = 0.0;        EndVal[1] = 10.0;    nStep[1] = 11;
/* m_vlh0 */        StartVal[2] = 0.0;        EndVal[2] = 5.0;    nStep[2] = 6;
/* m_Edes1 */        StartVal[3] = 120.0;    EndVal[3] = 140.0;    nStep[3] = 6;
/* m_Elh1 */        StartVal[4] = 50.0;        EndVal[4] = 70.0;    nStep[4] = 6;
/* m_exp0ad */        StartVal[5] = 2.0;        EndVal[5] = 10.0;    nStep[5] = 2;
/* m_partco0 */        StartVal[6] = 70.0;        EndVal[6] = 90.0;    nStep[6] = 3;
/* m_sCO */            StartVal[7] = 80.0;        EndVal[7] = 100.0;    nStep[7] = 5;
/* m_sO2 */            StartVal[8] = 40.0;        EndVal[8] = 80.0;    nStep[8] = 9;
/* m_Codes */        StartVal[9] = 16.0;        EndVal[9] = 20.0;    nStep[9] = 5;
/* m_Elhtheta */    StartVal[10]=-20.0;        EndVal[10]= 0.0;    nStep[10]= 3;

/*j = 0;
```

```
    for( i=1;i<=10;i++ ) {
        if( j < nStep[i] ) j=nStep[i];
    }*/

    double ValParam[11][1003];

    for( ii = 1; ii <= 10; ii++) {
        for( jj = 1; jj <= nStep[ii]; jj++) {
            ValParam[ii][jj]=StartVal[ii]+(jj-1)*(EndVal[ii]-StartVal[ii])/(nStep[ii]-1);
            if( ii == 1 ) ValParam[1][jj] = double(1.0e+11*pow(4,ValParam[1][jj]));
            if( ii == 2 ) ValParam[2][jj] = double(1.0e+7*pow(2,ValParam[2][jj]));
        }
    }

    // MAIN LOOP
    /*for (n1=1;n1<=nStep[1];n1++) {
        m_vdes0=ValParam[1][n1];
    for (n2=1;n2<=nStep[2];n2++) {
        m_vlh0=ValParam[2][n2];*/
    for (n3=1;n3<=nStep[3];n3++) {
        m_Edes1=ValParam[3][n3];
    for (n4=1;n4<=nStep[4];n4++) {
        m_Elh1=ValParam[4][n4];
    /*for (n5=1;n5<=nStep[5];n5++) {
        m_expOad=int(ValParam[5][n5]);*/
    for (n6=1;n6<=nStep[6];n6++) {
        m_partco0=ValParam[6][n6];
    for (n7=1;n7<=nStep[7];n7++) {
        m_sCO=ValParam[7][n7];
    for (n8=1;n8<=nStep[8];n8++) {
        m_sO2=ValParam[8][n8];
/*    for (n9=1;n9<=nStep[9];n9++) {
        m_Codes=ValParam[9][n9];*/
    /*for (n10=1;n10<=nStep[10];n10++) {
        m_Elhtheta=ValParam[10][n10];*/

        mittess = 0.0;

        // TEMPER LOOP
        for( n_T = 1; n_T <= m_TempNumber; n_T++ ) {
            Max = 0.0;
            Ts    = m_TempStart + n_T*m_TempStep;
            new_T( Ts );

            for( nDir = 0; nDir <= 1; nDir++ ) {
                for( n_xco = 1; n_xco <= m_xcoStep; n_xco++) {
                    if( nDir == 0 ) {
                        co[0]=co[1]=0.0; o[0]=o[1]=1.0;
                    } else {
                        co[0]=co[1]=1.0; o[0]=o[1]=0.0;
                    }
                    xco    = double(n_xco)/double(m_xcoStep);
                    new_xco( xco );
                    for (nTime=1; nTime<tmax; nTime++) {
                        ErrCode = Integration();
                        if( ErrCode != 0 )    goto Error;
                    }
                    sim[n_T][n_xco][nDir]=totprod*mmax;
                }
            }
            if( sim[n_T][n_xco][nDir] > Max ) Max = sim[n_T][n_xco][nDir];
        } // END TEMPER LOOP


        /////////   COMPARISON EXPERIMENT - SIMULATION  ///////////////////////

        // NORMALISATION
        for( n_xco = 1; n_xco <= m_xcoStep; n_xco++) {
            for( n_T = 1; n_T <= m_TempNumber; n_T++ ) {
                for( nDir = 0; nDir <= 1; nDir++ ) {
                    sim[n_T][n_xco][nDir] = sim[n_T][n_xco][nDir]/mittess*mittedata;
        }}}

        // CALC. ERRORS
        R2abs[1] = R2abs[2] = 0.0;
        R2rel[1] = R2rel[2] = 0.0;

        for( n_xco=1; n_xco <= m_xcoStep; n_xco++ ) {
            for( n_T = 1; n_T <= m_TempNumber; n_T++) {
                for( nDir=0; nDir <= 1; nDir++ ) {
                    diff   = sim[n_T][n_xco][nDir]-data[n_T+1][n_xco+1];
                    R2abs[nDir] = R2abs[nDir] + diff*diff;
                    R2rel[nDir] = R2rel[nDir] + (diff/data[n_T+1][n_xco+1])*(diff/data[n_T+1][n_xco+1]);
        }}}

        // OUTPUT
        fprintf(filename,"\n%e",m_vdes0);
        fprintf(filename,"\t%e",m_vlh0);
        fprintf(filename,"\t%e",m_Edes0);
        fprintf(filename,"\t%e",m_Elh0);
        fprintf(filename,"\t%i",m_expOad);
        fprintf(filename,"\t%e",m_Codes);
        fprintf(filename,"\t%e",m_Ct);
        fprintf(filename,"\t%e",m_sCO);
        fprintf(filename,"\t%e",m_sO2);
        fprintf(filename,"\t%e",m_vdes1);
        fprintf(filename,"\t%e",m_vlh1);
        fprintf(filename,"\t%e",m_Edes1);
        fprintf(filename,"\t%e",m_Elh1);
        fprintf(filename,"\t%e",m_partco0);
        //fprintf(filename,"\t%e",m_Elhtheta);
        fprintf(filename,"\t%e",time);
        if( mittedata/mittess > 0 )
            fprintf(filename,"\t%e",mittedata/mittess);
        else
            fprintf(filename,"\t%e",mittedata);
        fprintf(filename,"\t%e",R2rel[1]);
        fprintf(filename,"\t%e",R2rel[2]);
        fprintf(filename,"\t%e",R2abs[1]);
        fprintf(filename,"\t%e",R2abs[2]);

    }    //m_vdes0
    }    //m_vlh0
    }    //m_Edes1
    }    //m_Elh1
    }    //m_expOad
    //}    //m_partco0
    //}    //m_sCO
    //}    //m_sO2
    //}    //m_Codes
    //}    //m_ElhTheta
    // END MAIN LOOP

Error:
    fclose(filename);
    UpdateData( false );

    ErrorVerbose( ErrCode );
}


/////////////////////////////////////////////////////////
//            RATE DETERMINING STEP            ML 19.07.04    //
//            w/out bistability                            //
```

```
/////////////////////////////////////////////////////////

void CKinetikDlg::OnBtnRDS()
{
    int        n_xco, n_T, nTime, n;
    int     ErrCode;
    double  Ts, xco;
    double  PctInc;
    double  RateIni;
    double  oini=0.0, coini=1.0;
    double  drc[4];        // [JCatal 204 (2001) 520]
    double  sav1, sav2;
    FILE    * pFile[4];

    PctInc = m_RDSStep * 1.0e-2;

    // open files
    pFile[0] = fopen("drc_info.txt", "w");
    fprintf( pFile[0], "TStart %e TStep %e TNumber %e \n", m_TempStart, m_TempStep, m_TempNumber);
    fprintf( pFile[0], "xCO %i, xCO step %e \n", m_xcoStep, 1.0/double(m_xcoStep));
    fclose( pFile[0] );

    pFile[0] = fopen("drc_ini.txt", "w");
    pFile[1] = fopen("drc_CO.txt", "w");
    pFile[2] = fopen("drc_O2.txt", "w");
    pFile[3] = fopen("drc_LH.txt", "w");

    for( n_T=0; n_T<=m_TempNumber; n_T++ ) {
        Ts = m_TempStart + double(n_T)*m_TempStep;
        new_T( Ts );

        for( n_xco=1; n_xco < m_xcoStep; n_xco++ ) {
            xco = double( n_xco )/double( m_xcoStep );
            new_xco( xco );

            // 0. Initial rate
            co[0]=co[1]=coini; o[0]=o[1]=oini;
            for( nTime=1; nTime <= tmax; nTime++ ) {
                ErrCode = Integration();
                if( ErrCode != 0 ) goto Error;
            }
            RateIni = drc[0] = totprod;

            //1. adsorption desorption CO
            sav1 = kco; sav2 = vdes[0];
            kco     = kco     + kco*PctInc;
            vdes[0] = vdes[0] + vdes[0]*PctInc;
            co[0]=co[1]=0.0; o[0]=o[1]=1.0;
            for( nTime=1; nTime <= tmax; nTime++ ) {
                ErrCode = Integration();
                if( ErrCode != 0 ) goto Error;
            }
            kco    = sav1; vdes[0] = sav2;
            drc[1] = (totprod-RateIni)/(RateIni*PctInc);

            //2. adsorption O2
            sav1 = ko;
            ko   = ko + ko*PctInc;
            co[0]=co[1]=coini; o[0]=o[1]=oini;
            for( nTime=1; nTime <= tmax; nTime++ ) {
                ErrCode = Integration();
                if( ErrCode != 0 ) goto Error;
            }
            ko     = sav1;
            drc[2] = (totprod-RateIni)/(RateIni*PctInc);

            //3. reaction
            sav1   = vlh[0];
            vlh[0] = vlh[0] + vlh[0]*PctInc;
            co[0]=co[1]=coini; o[0]=o[1]=oini;
            for( nTime=1; nTime <= tmax; nTime++ ) {
                ErrCode = Integration();
                if( ErrCode != 0 ) goto Error;
            }
            vlh[0] = sav1;
            drc[3] = (totprod-RateIni)/(RateIni*PctInc);

            // OUTPUT
            for( n=0; n<4; n++ )
                fprintf( pFile[n], "%e\t", drc[n]);
        } // xCO loop

        for( n=0; n<4; n++ )
            fprintf( pFile[n], "\n");
    } // Temperature loop

Error:
    for( n=0; n<4; n++ ) fclose( pFile[n] );
    ErrorVerbose( ErrCode );
}


//////////////////////////////////////////////////
// ALL TRANSIENTS                               //
//////////////////////////////////////////////////
void CKinetikDlg::OnBtnAllTransient()
{
    OnBtnDataOneTransient();
}


//////////////////////////////////////////////////////////////
// ONE TRANSIENT                              ML 18/09/03    //
//////////////////////////////////////////////////////////////

void CKinetikDlg::OnBtnDataOneTransient()
{
    int        nTime, nDir;
    int        ErrCode;
    double    kosav, kcosav;
    double    time, Ts, xco;
    char    transf[20];
    int        nStep;


    Ts        = m_Texact;
    new_T( Ts );
    xco        = m_xcoexact/100.0;
    new_xco( xco );

    kosav    = ko;
    kcosav    = kco;

    sprintf( transf, "TrsT%.3ix%.2id%.2i.txt", int(Ts), int(xco*100.0), int(100-m_partco0) );
    filename= fopen( transf,"w" );
    fprintf(filename,"\ntime\tCO2\tNtot");
    FileHeader();
    fprintf(filename,"\nHERE:\nCO-fraction:  %.2f",xco*100.0);
    fprintf(filename," percent\nTemperature: %i",Ts);
    //fprintf(filename,"\tFac.1CO2\tN1\tFac.2CO2\tN2\tFac.1CO\tNco1\tFac.2CO\tNco2\tFac1.O\tNo1\tFac.2O\tNo2\tCO1->2\tO1->2\

    for( nDir = 0; nDir <= 1; nDir++ ) {
        ko   = kosav;
        kco  = kcosav;
        time = 0.0;
        nStep = 0;

        fprintf( filename, "\n" );
        if( nDir == 0 ) {
            co[0]=co[1]=0.0; o[0]=o[1]=1.0;
        } else {
            co[0]=co[1]=1.0; o[0]=o[1]=0.0;
        }

        for( nTime = 1; nTime <= tmax+tafter; nTime++) {
            time = time + dt;
            ++nStep;

            if( nTime > tmax )    {
                kco = 0.0;
                ko  = 0.0;
            }

            ErrCode = Integration();
            if( ErrCode != 0 )    goto Error;

            if( nStep == int(1.0/dt) ) {
                nStep = 0;
                fprintf( filename, "\n%e", time );
                fprintf( filename, "\t%e", totprod*mmax );
                fprintf( filename, "\t%e", totprod*mmax*m_Npd );
                /*fprintf( filename, "\t%e", prod[0]*mmax );
                fprintf( filename, "\t%e", prod[0]*mmax*m_Npd );
                fprintf( filename, "\t%e", prod[1]*mmax );
                fprintf( filename, "\t%e", prod[1]*mmax*m_Npd );
                fprintf( filename, "\t%e", co[0] );
                fprintf( filename, "\t%e", co[0]*maxco*m_Npd );
                fprintf( filename, "\t%e", co[1] );
                fprintf( filename, "\t%e", co[1]*maxco*m_Npd );
                fprintf( filename, "\t%e", o[0] );
                fprintf( filename, "\t%e", o[0]*maxo*m_Npd );
                fprintf( filename, "\t%e", o[1] );
                fprintf( filename, "\t%e", o[1]*maxo*m_Npd );*/
            }
        }
    }

Error:
    fclose(filename);

    if( eps == 0.0 ) ErrCode = 64;
    if( ErrCode == 0 ) ErrCode = 1024;
    ErrorVerbose( ErrCode );
}


//////////////////////////////////////////////////////////////////
// STEADY STATES INCLUDING COVERAGES            ML 17/09/2003    //
//////////////////////////////////////////////////////////////////

void CKinetikDlg::OnBtnSS()
{
    int        nTime, n_xco, nDir;
    int        ErrCode;
    double  Ts, xco;
    char    ssf[20];

    Ts = m_TempStart;
    new_T( Ts );

    sprintf( ssf, "SST%.3id%.2i.txt", int(Ts), int(100-m_partco0) );
    filename = fopen(ssf,"w");
    fprintf(filename,"\nxco\tCO2\nT=%.0f\n", Ts);
    FileHeader();

    for( nDir=0; nDir <= 1; nDir++ ) {
        fprintf(filename,"\n");

        for (n_xco=1; n_xco<=m_xcoStep-1; n_xco++) {
            xco = double(n_xco)/double(m_xcoStep);
            new_xco( xco );

            // INTEGRATIONS
            if( nDir == 0 ) {
                co[0]=co[1]=1.0; o[0]=o[1]=0.0;
            } else {
                co[0]=co[1]=0.0; o[0]=o[1]=1.0;
            }

            for( nTime = 1; nTime <= tmax; nTime++) {
                ErrCode = Integration();
                if( ErrCode != 0 )    goto Error;
            }

            // OUTPUT
            fprintf(filename, "\n%e", xco);
            fprintf(filename, "\t%e", totprod*mmax);
        }
    }

Error:
    fclose(filename);

    if( eps == 0.0 ) ErrCode = 64;
    ErrorVerbose( ErrCode );
}


/*************************************************
**                                              **
**                Bistability                      **
**                                              **
*************************************************/

//////////////////////////////////////////////////////////////////////
// Steady State Coverages :                           ML 18/09/2003    //
// starting with different coverages                                //
//////////////////////////////////////////////////////////////////////

void CKinetikDlg::OnBtnSSCov()
{
    char    SSCf[20];
    int         no, nco, nTime;
    double  Ts, xco;
    int         ErrCode;
    double    dResolution;

    HelpVariable();

    dResolution = double(m_resolution);

    xco        = m_xcoexact/100.0;
    new_xco( xco );
    Ts         = m_Texact;
    new_T( Ts );

    sprintf( SSCf, "SSCT%.3ix%.2id%.2i.txt", int(Ts), int(xco*100.0), int(100-m_partco0) );
    filename = fopen(SSCf,"w");
    fprintf(filename,"CO\t O\t co[1]\nSteady State values\n");
    FileHeader();
    fprintf(filename,"file format:\n1st column: starting value theta(CO) \n2nd: starting value theta(O)\n3rd: steady state t

    /* file format:
    1st: steady state theta(CO)
    2nd: steady state theta(O)
    3th: steady state co[1] production
    */

    for( nco=0; nco <= m_resolution; nco++ ) {
        for( no=0; no <= m_resolution-nco; no++ ) {
            co[0] = co[1]    = double(nco)/dResolution;
            o[0]  = o[1]    = double(no) /dResolution;

            // INTEGRATION
            for( nTime = 0; nTime <= tmax; nTime++ ) {
                ErrCode = Integration();
                if( ErrCode != 0 )    goto Error;
            }

            // OUTPUT
            fprintf( filename,"\n%e", p[0]*co[0] + p[1]*co[1] );
            fprintf( filename,"\t%e", p[0]*o[0]  + p[1]*o[1]  );
            fprintf( filename,"\t%e", totprod*mmax            );
        }
    }

Error:
    fclose(filename);
    if( eps == 0.0 ) ErrCode = 64;

    ErrorVerbose( ErrCode );
}


//////////////////////////////////////////////////////////////////////
//    bistabiliy region                              ML 17/09/03        //
//////////////////////////////////////////////////////////////////////

void CKinetikDlg::OnBtnRegion()
{
    int        n_xco, n_T, nTime, nDir;
    int        ErrCode;
    double    CO2prod[2], Ts, xco;

    FILE    *bro;
    char    brf[25];

    sprintf( brf, "bist_regd%.2i.txt", int(100-m_partco0) );
    bro    = fopen(brf,"w");

    // FileHeader(); needs declaration of FILE *filename
    fprintf(bro,"xco\t Temperature\n");

    for( n_xco=1; n_xco <= m_xcoStep-1; n_xco++ ) {
        xco = double(n_xco)/double(m_xcoStep);
        new_xco( xco );

        for( n_T=0; n_T<=m_TempNumber; n_T++ ) {
            Ts = m_TempStart + m_TempStep*double(n_T);
            new_T( Ts );

            // INTEGRATIONS
            for( nDir = 0; nDir <= 1; nDir++ ) {
                if( nDir == 0 ) {
                    co[0]=co[1]=1.0; o[0]=o[1]=0.0;
                } else {
                    co[0]=co[1]=0.0; o[0]=o[1]=1.0;
                }
                for( nTime = 1; nTime <= tmax; nTime++ ) {
                    ErrCode = Integration();
                    if( ErrCode != 0 )    goto Error;
                }
                CO2prod[nDir] = totprod;
            }

            // OUTPUT
            if(    CO2prod[0]/CO2prod[1] < 1.02 &&
                CO2prod[0]/CO2prod[1] > 0.98 ) {
            } else {
                fprintf(bro,"\n%e",xco);
                fprintf(bro,"\t%e",Ts);
            }
        }
    }

Error:
    fclose(bro);

    if( eps == 0.0 ) ErrCode = 64;
    ErrorVerbose( ErrCode );
}
```

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
`info.txt`	A small summary of the parameters used, the output of this file is also sent to STD OUT
`gco2VSxco.dat`	Global CO_2 production vs. x_{CO}
`ThPhiR.dat`	Coordinates of each surface element on the particle, in spherical coordinates (θ, ϕ, r)
`fluxo.dat`	Local O_2 flux
`fluxco.dat`	Local CO flux
`fluxbs.dat`	Local backscattered flux
`fort.#1`[a]	Transient global CO_2 production
`fort.#2`[b]	Local oxygen coverage
`fort.#3`[c]	Local CO coverage (not generated by default)
`fort.#4`[d]	Local CO_2 production

[a] $\#1 = x_{\mathrm{CO}} \times 10^5$
[b] $\#2 = \text{integration step/parameter} \times 10^7 + \#1$
[c] $\#3 = \#2 + 1$
[d] $\#4 = \#2 + 2$

```
program diff
!
! Creation Date:        12.01.2004 15:17:36
! Last Modification:    23.03.2004 14:44:05
! Author:               Mathias Laurin
!
! include backscattering + shadow
! (Carsten Beta          5148)
!
implicit none
integer*4, parameter    :: nThetaParam=19, nPhiParam=20, nrbsParam=1000
integer*4    :: nPhi, nTheta, nrbs, nthetabs
integer*4    :: k, MaxIt
integer*4    :: savqo, nsavqo, savgco2, nsavgco2, nxco, nfile
integer*4    :: mqco, noBS, nhomo
real*8       :: rbs, rbsmax, bs(0:nthetaparam, nrbsparam), drbs
real*8       :: rx, ry, rz
real*8       :: bsx, bsy, bsvec(3), normbs, sumbso, sumbsco, bsa, sola
real*8       :: W(4)
real*8       :: Phi, Theta, radius(0:nPhiParam+1), ra, rc, ex2
real*8       :: p(3), q(3), r(3), normr
real*8       :: alpha, sina, cosa, beta, cosb, sinb, ThetaPhi
real*8       :: dq2, df1, df2
real*8       :: dq2f(nphiParam), df1f(nPhiParam), df2f(nphiParam)
real*8       :: cosg, sing, cose
real*8       :: sinfi(0:nPhiParam+1),  cosfi(0:nPhiParam+1)
real*8       :: sinq(-1:nThetaParam+1), cosq(-1:nThetaParam+1)
real*8       :: fluxo(0:nThetaParam, 0:nPhiParam)
real*8       :: fluxco(0:nThetaParam, 0:nPhiParam)
real*8       :: dfluxo(0:nThetaParam, 0:nPhiParam)
real*8       :: dfluxco(0:nThetaParam, 0:nPhiParam)
real*8       :: dq, df
real*8       :: dqodt, dqcodt, qoi, qcoi
real*8       :: qo(-1:nThetaParam+1, -1:nPhiParam+1)
real*8       :: qco(0:nThetaParam, 0:nPhiParam)
real*8       :: ro, rco
real*8       :: nu_d, en_d, nu_r, en_r, e1, a_d
real*8       :: T, rt
real*8       :: Ftot, Ptot, fco, fo2
real*8       :: kb, pi, Tvel, Rgp, Na
real*8       :: mco, mo2, nPd, xco
real*8       :: sCO, sO2, Ct
real*8       :: d1, pco, po, kr
real*8       :: en_diff, dfick
real*8       :: dt
real*8       :: dum, duma, crit, lap
real*8       :: gco2, avgqco, avgqo, lco2(0:nThetaParam, 1:nPhiParam), qcoparam

! Files
open(unit=1,file='ThPhiR.dat', status='replace', action='write')
open(unit=2,file='fluxo.dat',  status='replace', action='write')
open(unit=3,file='fluxco.dat', status='replace', action='write')
open(unit=4,file='fluxbs.dat', status='replace', action='write')
!open(unit=15,file='profile_phiVSr.dat', status='replace', action='write')
open(unit=25,file='gco2VSxco.dat', status='replace', action='write')

! General
maxIt   = 2500000
dt      = 1.0d-4           ![s]
savgco2 = 25000
savqo   = maxIt

! Constants
kb   = 1.380658d-23       ![J/K]     !! [J] == [Kg.m2.s-2]
Na   = 6.0221367d23       ![mol-1]
Rgp  = Na*kb              ![J.mol-1.K-1] !! [Kg.m2.s-2.mol-1.K-1]
pi   = 4.0d0*datan(1.0d0)

!Ptot = 1.0d-4            ![Torr]
!Ptot = Ptot/760.0d0*101.325d0 ![Pa = kg.m-1.s-2]
Ptot = 1.0d-4             ![Pa]
```

```
T    = 415.0d0            ![K]
RT   = Rgp*T              ![J.mol-1]
en_diff = 55.0d3          ![J.mol-1]
! Diffusion
Dfick = 1.0d-3                  ![cm2.s-1]
Dfick = Dfick*1.0d-4            ![m2.s-1]
Dfick = Dfick*dexp(-en_diff/RT) ![m2.s-1]

!Dfick = 0.0d0

mqco = 1              ! m = 0:    cov. dep.
                      ! m = 1: no cov. dep.
qcoparam = 2.0d0
noBS = 0              ! if noBS=1, no backscattering

nhomo = 0             ! if nhomo=1, homogeneous fluxes.

! Initial values
qoi  = 0.d0
qcoi = 0.5d0

nPd  = 1.53d15            ![cm-2]
nPd  = nPd*1.0d4          ![m-2]
mco  = 28.010d-3/Na       ![kg]
mo2  = 31.999d-3/Na       ![kg]
Tvel = 300.0d0            ![K]

! Desorption
nu_d = 4.0d14             ![s-1]
en_d = 142.0d3            ![J.mol-1]
a_d  = 0.12
! Reaction
nu_r = 5.0d7              ![s-1]
en_r = 53.0d3             ![J.mol-1]
! Adsorption CO
sCO  = 0.7d0
Ct   = 0.3d0
! Adsorption O2
sO2  = 1.0d0 - 7.4d-4*T

! ellipse
rc   = 450.0d-9           ! height     [m]
ra   = 250.0d-9           ! 1/2 width  [m]

if( ra .ge. rc ) then     ! oblate spheroid (squashed)
    ex2  = 1.0d0 - rc*rc/(ra*ra) ! eccentricity^2
else                      ! prolate spheroid (pointy)
    ex2  = 1.0d0 - ra*ra/(rc*rc)
endif
write(*,*) 'eccentricity',ex2
if( (ex2.lt.0.0d0).or.(ex2.ge.1.0d0) ) stop

!---------------------------------------------------------------------
!   ANGLES: Discretisation, calculation of cos, sin and radius
!---------------------------------------------------------------------

df              = 0.5d0*Pi/real(nPhiParam+1)
dq              = 2.0d0*Pi/real(nThetaParam+1)  ! +1 to exclude 2 Pi

alpha           = 55.0d0                        ! incidence of the O2 beam
                                                ! from the (xy) plane [deg]
alpha           = alpha/180.0d0*Pi              ! [radian]

beta            = 90.0d0                        ! incidence of the CO beam [deg]
beta            = beta/180.0d0*Pi               ! [radian]

ThetaPhi        = real((nThetaParam+1)*nPhiParam)
```

```

crit            = 1.0d-4                        ! cos / sin annihilation crit.
do nPhi=0, nPhiParam+1
   Phi = df*real(nPhi)

   sinfi(nPhi) = dsin(Phi)
   if((sinfi(nPhi) .lt. crit).and.(sinfi(nPhi) .gt. -crit))  sinfi(nPhi) = 0.0d0
   if(sinfi(nPhi)  .gt.  1.0d0-crit)  sinfi(nPhi)  =  1.0d0
   if(sinfi(nPhi)  .lt. -1.0d0+crit)  sinfi(nPhi)  = -1.0d0

   cosfi(nPhi) = dcos(Phi)
   if((cosfi(nPhi) .lt. crit).and.(cosfi(nPhi) .gt. -crit))  cosfi(nPhi) = 0.0d0
   if(cosfi(nPhi)  .gt.  1.0d0-crit)  cosfi(nPhi)  =  1.0d0
   if(cosfi(nPhi)  .lt. -1.0d0+crit)  cosfi(nPhi)  = -1.0d0

   if( rc .ge. ra ) then              ! prolate sph.d
       radius(nphi) = (1.0d0 - ex2)/(1.0d0 - ex2*cosfi(nphi)*cosfi(nphi))
       radius(nPhi) = rc*dsqrt(radius(nphi))
   else                               ! oblate sph.d
       radius(nphi) = (1.0d0 - ex2)/(1.0d0 - ex2*sinfi(nphi)*sinfi(nphi))
       radius(nphi) = ra*dsqrt(radius(nphi))
   endif
   !write(15,*) phi/pi*180.0d0, radius(nphi)
enddo
do nTheta=-1, nThetaParam+1
   Theta = dq*real(nTheta)

   sinq(nTheta) = dsin(Theta)
   if((sinq(nTheta) .lt. crit).and.(sinq(nTheta) .gt. -crit)) sinq(nTheta) = 0.0d0
   if(sinq(nTheta) .gt.  1.0d0-crit)  sinq(nTheta) =  1.0d0
   if(sinq(nTheta) .lt. -1.0d0+crit)  sinq(nTheta) = -1.0d0

   cosq(nTheta) = dcos(Theta)
   if((cosq(nTheta) .lt. crit).and.(cosq(nTheta) .gt. -crit)) cosq(nTheta) = 0.0d0
   if(cosq(nTheta) .gt.  1.0d0-crit)  cosq(nTheta) = 1.0d0
   if(cosq(nTheta) .lt. -1.0d0+crit)  cosq(nTheta) = -1.0d0
enddo

cosa = dcos(alpha)
if((cosa .lt. crit).and.(cosa .gt. -crit))       cosa = 0.0d0
if(cosa  .gt.  1.0d0-crit) cosa =  1.0d0
if(cosa  .lt. -1.0d0+crit) cosa = -1.0d0
sina = dsin(alpha)
if((sina .lt. crit).and.(sina .gt. -crit))       sina = 0.0d0
if(sina  .gt.  1.0d0-crit) sina =  1.0d0
if(sina  .lt. -1.0d0+crit) sina = -1.0d0

cosb = dcos(beta)
if((cosb .lt. crit).and.(cosb .gt. -crit))       cosb = 0.0d0
if(cosb  .gt.  1.0d0-crit) cosb =  1.0d0
if(cosb  .lt. -1.0d0+crit) cosb = -1.0d0
sinb = dsin(beta)
if((sinb .lt. crit).and.(sinb .gt. -crit))       sinb = 0.0d0
if(sinb  .gt.  1.0d0-crit) sinb =  1.0d0
if(sinb  .lt. -1.0d0+crit) sinb = -1.0d0

!----------------------------------------------------------------------
!          Factors to the LAPLACIAN (out of the main loop)
!----------------------------------------------------------------------
do nphi = 1, nphiparam
   dum        = radius(nphi)*radius(nphi)

   dq2f(nphi) = dum * sinfi(nphi)*sinfi(nphi) * dq*dq
   dq2f(nphi) = 1.0d0/dq2f(nphi)

   df1f(nphi) = dum * sinfi(nphi) * df
   df1f(nphi) = cosfi(nphi)/df1f(nphi)

   df2f(nphi) = 1.0d0/(dum * df*df)

   !write(55,'(3es20.8)') dq2f(nphi),df1f(nphi),df2f(nphi)
enddo

!----------------------------------------------------------------------
!   BACKSCATTERING from the Support:
!----------------------------------------------------------------------
rbsmax = 25.0d-6                       ! [m]
drbs   = (rbsmax - ra)/real(nrbsParam) ! [m]
write(*,*) 'dq=',dq
write(*,*) 'dr=',abs(drbs)

!----------------------------------------------------------------------
!                       FLUX DISTRIBUTION
!----------------------------------------------------------------------
dfluxo(:,:) = 0.0d0
dfluxco(:,:)= 0.0d0
do nPhi=1, nPhiParam
   do nTheta=0, nThetaParam
      ! ------------------------------------  DIRECT FLUX from the beam
      ! cart. coord.
      ! orthogonal vectors (p/Phi, q/Theta)
      p(1) = cosq(ntheta)*( radius(nphi-1)*sinfi(nphi-1) - radius(nphi+1)*sinfi(nphi+1) )
      p(2) = sinq(ntheta)*( radius(nphi-1)*sinfi(nphi-1) - radius(nphi+1)*sinfi(nphi+1) )
      p(3) =                radius(nphi-1)*cosfi(nphi-1) - radius(nphi+1)*cosfi(nphi+1)

      q(1) = radius(nphi)*sinfi(nphi)*(cosq(ntheta-1) - cosq(ntheta+1))
      q(2) = radius(nphi)*sinfi(nphi)*(sinq(ntheta-1) - sinq(ntheta+1))
      q(3) = 0.0d0

      ! r normal/part. ie: r.q = r.p = 0
      dum  =  p(2)*q(1) - p(1)*q(2)
      r(1) =  p(3)*q(2)/dum
      r(2) = -p(3)*q(1)/dum
      r(3) =  1.0d0
      normr= dsqrt( r(1)*r(1) + r(2)*r(2) + r(3)*r(3) )

      ! Flux O2
      ! scalar prod.   ie: r.Flux = normr*cosg
      cosg = r(1)*cosa + r(3)*sina
      cosg = cosg/normr
      if( cosg .lt. crit ) cosg = 0.0d0
      dfluxo(ntheta, nphi) = cosg

      ! Flux CO
      cosg = r(1)*cosb + r(3)*sinb
      cosg = cosg/normr
      if( cosg .lt. crit ) cosg = 0.0d0
      dfluxco(ntheta, nphi) = cosg

      ! ------------------------------------------  BACKSCATTERED FLUX
      if( noBS .ne. 1 ) then
         sumbso  = 0.0d0
         sumbsco = 0.0d0

         ! Origin of r, on the particle
         rx = radius(nphi)*cosq(ntheta)*sinfi(nphi)
         ry = radius(nphi)*sinq(ntheta)*sinfi(nphi)
         rz = radius(nphi)*cosfi(nphi)

         do nthetabs=0, nthetaparam
            do nrbs=1, nrbsparam
               rbs  = real(nrbs)/real(nrbsparam)*(rbsmax - ra) + ra

               ! origin of BS flux / substrate
```

```
                bsx  = rbs*cosq(nthetabs)
                bsy  = rbs*sinq(nthetabs)

                ! Area elt / substrate
                bsa  = rbs*drbs*dq

                ! BSvec is backscattered flux vector (btw pts BS and R)
                bsvec(1) = rx-bsx
                bsvec(2) = ry-bsy
                bsvec(3) = rz
                normbs   = bsvec(1)*bsvec(1) + bsvec(2)*bsvec(2) + bsvec(3)*bsvec(3)
                normbs   = dsqrt(normbs)

                ! sing is INTENSITY of BS flux (exp. vector norm)
                sing = abs(rz) / normbs
                if( sing .lt. crit )        sing = 0.0d0
                if( sing .gt. 1.0d0-crit )  sing = 1.0d0
                ! normalisation (solid angle / half sph.)
                sing = sing/pi

                ! cose/( normbs*normbs ) is solid angle
                ! scal. prod. ( BS.r )
                cose   =  bsvec(1)*r(1) + bsvec(2)*r(2) + bsvec(3)*r(3)
                cose   = -cose/(normbs*normr)
                if( cose .lt. crit )        cose = 0.0d0
                if( cose .gt. 1.0d0-crit )  cose = 1.0d0

                sola     = sing*cose/(normbs*normbs)*bsa

                ! shadow of O2 beam
                dum  = 2.0d0*rc*rc*bsx*cosa
                dum  = dum*dum

                duma = 4.0d0*(rc*rc*cosa*cosa + ra*ra*sina*sina)
                duma = duma*(rc*rc*( (bsx*bsx + bsy*bsy)-ra*ra) )

                if( (dum .lt. duma ).or.(bsx.gt.0.0d0) ) then
                    ! out of the shadow
                    sumbso = sumbso + sola
                endif

                ! shadow of CO beam
                dum  = 2.0d0*rc*rc*bsx*cosb
                dum  = dum*dum

                duma = 4.0d0*(rc*rc*cosb*cosb + ra*ra*sinb*sinb)
                duma = duma*(rc*rc*( (bsx*bsx + bsy*bsy)-ra*ra) )

                if( (dum .lt. duma ).or.(bsx.gt.0.0d0) ) then
                    ! out of the shadow
                    sumbsco = sumbsco + sola
                endif

             enddo
          enddo
          dfluxco(ntheta, nphi) = dfluxco(ntheta, nphi) + sumbsco*sinb
          dfluxo(ntheta, nphi)  = dfluxo(ntheta, nphi)  + sumbso*sina
       endif
    enddo
enddo

! Homogeneous model
if( nhomo .eq. 1 ) then
   dfluxo(:,:) = sum(dfluxo(:,:))/thetaphi
   dfluxco(:,:)= sum(dfluxco(:,:))/thetaphi
endif

```

```
! save flux distributions
do nphi=1, nPhiParam
   do nTheta=0, nThetaParam
      phi   = real(nPhi)*df
      theta = real(nTheta)*dq
      write(1,'(3es20.8)') theta, phi, radius(nphi)   !ThPhiR.dat
      write(2,'(es20.8)')  dfluxo(ntheta,nphi)        !fluxo.dat
      write(3,'(es20.8)')  dfluxco(nTheta, nPhi)      !fluxco.dat
      write(4,'(es20.8)')  sumbsco                    ! contrib. BS
   enddo
enddo

close(1); close(2); close(3);close(4)


!---------------------------------------------------------- XCO  LOOP --
fco = dsqrt( 2.0d0*pi*mco*kb*Tvel )      ![Kg.m.s-1]
fo2 = dsqrt( 2.0d0*pi*mo2*kb*Tvel )
do nxco = 0, 40
!xco = real(nxco)/20.0d0             ! nxco = 1,19
xco = real(nxco)/200.0d0 + .34d0     ! nxco = 0,199
!if( nxco .eq. 1 ) xco = 0.34d0
!if( nxco .eq. 2 ) xco = 0.67d0

Ftot = xco*fco
Ftot = Ftot + (1.0d0-xco)*fo2
Ftot = Ptot/Ftot                         ![m-2.s-1]

! Rate constants
kr   = nu_r*dexp(-en_r/rt)    ![s-1]
pco  = xco*Ftot/nPd           ![s-1]
po   = (1.0d0-xco)*Ftot/nPd   ![s-1]

fluxo(:,:)  = dfluxo(:,:) *po
fluxco(:,:) = dfluxco(:,:)*pco

! Initial values
qo(:,:)  = qoi
qco(:,:) = qcoi

nsavqo   = 0
nsavgco2 = 0

!--------------------------------------------------------- TIME  LOOP --
write(*,*) 'start integration', xco
do k=1, maxit
   nsavqo   = nsavqo + 1
   nsavgco2 = nsavgco2 + 1

   gco2   = 0.0d0
   avgqco = 0.0d0
   avgqo  = 0.0d0

   !!!            INTEGRATION
   do nPhi = 1, nPhiParam
      do nTheta = 0, nThetaParam
         ! relative coverages
         rco    = 2.0d0*qco(nTheta,nPhi)
         ro     = 4.0d0*qo(nTheta,nPhi)

         ! Adsorption CO
         dum = 1.0d0 - rco - Ct *ro
         if( dum.gt.0.0d0 ) then
            W(1) = fluxco(nTheta, nPhi)*sCO*dum
         else
            W(1) = 0.0d0
         endif

         ! Desorption CO
         e1  = en_d*(1.0d0 - a_d*(rco))
         d1  = nu_d*dexp( -e1/rt )
         W(2) = d1*qco(nTheta,nPhi)

         ! Reaction
         W(3) = Kr*qco(nTheta,nPhi)*qo(nTheta,nPhi)
         gco2 = gco2 + w(3)
         lco2(nTheta, nPhi) = w(3)

         ! Adsorption O2
         dum = 1.0d0 - rco - ro
         if( dum.gt.0.0d0 ) then
             W(4) = fluxo(ntheta, nphi)*sO2* dum*dum
         else
             W(4) = 0.0d0
         endif

         ! Diffusion
         ! BC: Theta: PBC; Phi: no flux
         if(nTheta .eq. 0)           qo(-1, nphi)            = qo(nThetaParam, nphi)
         if(nTheta .eq. nThetaParam) qo(ntheta+1, nphi)      = qo(0, nphi)
         if(nPhi   .eq. nPhiParam)   qo(ntheta, nPhiParam+1) = qo(ntheta, nPhiParam)
         if(nPhi   .eq. 1)           qo(nTheta, 0)           = qo(nTheta, 1)

         dq2 = qo(ntheta-1, nPhi) + qo(ntheta+1, nPhi) - 2.0d0*qo(ntheta, nPhi)
         dq2 = dq2f(nphi)*dq2

         df1 = qo(ntheta, nPhi+1) - qo(ntheta, nPhi)
         df1 = df1f(nphi)*df1

         df2 = qo(ntheta, nPhi-1) + qo(ntheta, nPhi+1) - 2.0d0*qo(ntheta, nPhi)
         df2 = df2f(nphi)*df2

         lap = dq2 + df1 + df2

         ! Integration
         dqodt  = 2.0d0*W(4)  - W(3) + Dfick*lap
         dqcodt = W(1) - W(2) - W(3)

         qo(nTheta, nPhi) = qo(nTheta,nPhi)  + dqodt  * dt
         qco(nTheta,nPhi) = qco(nTheta,nPhi) + dqcodt * dt

         avgqo  = avgqo  + qo(nTheta, nPhi)
         avgqco = avgqco + qco(nTheta,nPhi)

         ! Divergence
         if(     qo(nTheta, nPhi) .lt. 0.0d0 ) then
            write(*,*) 'DIV: qo .lt. 0', Dfick*lap
            stop
          elseif( qo(nTheta, nPhi) .gt. 0.25d0 ) then
            write(*,*) 'DIV: qo .gt. 0.25', Dfick*lap
            stop
          elseif(qco(nTheta, nPhi) .lt. 0.0d0 ) then
            write(*,*) 'DIV: qco .lt. 0', Dfick*lap
            stop
          elseif(qco(nTheta, nPhi) .gt. 0.5d0 ) then
            write(*,*) 'DIV: qo .gt. 0.5', Dfick*lap
            stop
         endif
      enddo ! nTheta
   enddo ! nPhi

   gco2 = gco2 / thetaphi

   ! diff CO infinitly fast but fct of Qo
   avgqo  = avgqo  / thetaphi
   avgqco = avgqco / thetaphi

   if( (mqco .eq. 1).or.(avgqo .ge. 0.25d0) ) then    ! CO equaly shared
       qco(:,:) = avgqco
   else                      ! mqco .eq. 0
      dum = avgqco/(0.25d0 - avgqo)
      do nphi = 1, nphiparam
         do ntheta = 0, nthetaparam
            qco(ntheta, nphi) = ( 0.25d0 - qo(ntheta, nphi) )*dum
            qco(ntheta, nphi) = qco(ntheta, nphi) - avgqco
            qco(ntheta, nphi) = qco(ntheta, nphi)*qcoparam
            qco(ntheta, nphi) = qco(ntheta, nphi) + avgqco
         enddo
      enddo
   endif
   if( (k.eq.1).and.(mqco.eq.0) ) then
      write(*,*) sum(qco(:,:)), avgqco*thetaphi
   endif

   ! save global co2 production
   if( nsavgco2 .eq. savgco2 ) then
       write(int(xco*1.0d5),'(2es20.8)') real(k)*dt, gco2
       nsavgco2 = 0
   endif

   ! save cov distributions and local CO2 prod.
   if( nsavqo .eq. savqo ) then
      nsavqo = 0
      nfile = int( real(k)/real(savqo))
      if( k.eq. maxit-1000 ) nfile = 99
      nfile = int( real(nfile)*1.0d7 )
      nfile = nfile + int(xco*1.0d5)
      do nPhi=1,nPhiParam
         do nTheta=0,nThetaParam
            phi   = real(nPhi)*df
            theta = real(nTheta)*dq

            write(nfile,  '(es20.8)') qo(ntheta,nphi)
            write(nfile+2,'(es20.8)') lco2(ntheta, nphi)
            if( mqco .ne. 1 ) then
                write(nfile+1,'(es20.8)') qco(ntheta,nphi)
            endif
         enddo
      enddo
   endif
   close(nfile)
   if( mqco .ne. 1 ) close(nfile+1)
   close(nfile+2)

enddo !------------------------------------------------ END TIME  LOOP --

write(25,'(2es20.8)') xco, gco2    ! gco2VSxco.dat
close(int(xco*1.0d5))

enddo !------------------------------------------------- END XCO  LOOP --

write(*,*) 'diff Done'

open(unit=33,file='info.txt', status='replace', action='write')
write(33,'(a8,es20.8)') 'T=',     T
write(33,'(a8,es20.8)') 'E=',     en_diff
write(33,'(a8,es20.8)') 'D=',     Dfick
write(33,'(a8,es20.8)') 'alpha=',Alpha
write(33,'(a8,es20.8)') 'beta=', Beta
write(33,'(a8,i8)')     'mqco=', mqco
write(33,'(a8,i8)')     'noBS=', noBS
write(33,'(a8,i8)')     'nhomo=',nhomo
close(33)

write(*,'(a8,es20.8)') 'T=',    T
write(*,'(a8,es20.8)') 'E=',    en_diff
write(*,'(a8,es20.8)') 'D=',    Dfick
write(*,'(a8,es20.8)') 'alpha=',Alpha
write(*,'(a8,es20.8)') 'beta=', Beta
write(*,'(a8,i8)')     'mqco=', mqco
write(*,'(a8,i8)')     'noBS=', noBS
write(*,'(a8,i8)')     'nhomo=',nhomo

stop
end

```

A.2.2 Simulation of the experimental data

This Matlab function `SimAR.m` simulates the AR CO_2 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 ouput `out` = [`phiar`, `tot`, `totn`] consists of three vectors: `totn` is the normalized CO_2 production `tot` at position `phiar`. The angle distributions are also plotted in Matlab.

```
function out = SimAR(qo, cosn);

% Simulation of the AR distribution
% for Matlab
%
% Creation Date:       26.02.2004   14:34
% Last Modification:   03.05.2004   16:25
%

if nargin == 1,
  cosn = 1;
end

nq   = 20;
nf   = 20;
nth  = 1;
nfi  = nf;
nmax = nq*nf;
us   = 40;

% load coordinates of pts on spheroid
coord = load('ThPhiR.dat');
th    = coord(:,1)';
phi   = coord(:,2)';
radius= coord(:,3)';
phi2  = 0.5*pi-phi;
[sx, sy, sz] = sph2cart(th,phi2,radius);
tri   = delaunay(sx,sy);

% trigo fct
sfi   = sin(phi);
cfi   = cos(phi);
sq    = sin(th);
cq    = cos(th);
% boundary cond.
sfi(nmax+1)    = 0.0;
sfi(nmax+2)    = 1.0;
```

```
cfi(nmax+1)    = 1.0;
cfi(nmax+2)    = 0.0;
radius(nmax+1) = 450.0e-9;
radius(nmax+2) = 250.0e-9;

% area on the sph.d
dq    = abs(th(1)-th(2));
df    = abs(phi(1)-phi(1+nf));
dA    = radius.*radius.*sfi *dq*df;
dA    = dA(1:400);

% vect normal to the particle
for n = 1:nmax
  % // (z)
  m = n-nfi;
  if m < 1,    m = nmax+1; end
  M = n+nfi;
  if M > nmax, M = nmax+2; end

  p(n,1) = cq(n)*( radius(m)*sfi(m) - radius(M)*sfi(M) );
  p(n,2) = sq(n)*( radius(m)*sfi(m) - radius(M)*sfi(M) );
  p(n,3) =         radius(m)*cfi(m) - radius(M)*cfi(M);

  % // (xy)
  m = n-nth;
  if m < 1,    m = m+nmax; end
  M = n+nth;
  if M > nmax, M = M-nmax; end

  q(n,1) = radius(n)*sfi(n)*(cq(m) - cq(M));
  q(n,2) = radius(n)*sfi(n)*(sq(m) - sq(M));
  q(n,3) = 0.;
end
dum =  p(:,2).*q(:,1) - p(:,1).*q(:,2);
rx  =  p(:,3).*q(:,2)./dum;
ry  = -p(:,3).*q(:,1)./dum;
rz  =  ones(nmax,1);
normr  = sqrt( rx.*rx + ry.*ry + rz.*rz );
rx  = rx./normr;
ry  = ry./normr;
rz  = rz./normr;

% unit sphere (position detector)
[usx, usy, usz] = sphere(us);
[usq, usf, usr] = cart2sph(usx,usy,usz);
sus             = size(usx);
dqus = 2.0 * pi / us;
dfus = pi / us;
dAus = cos(usf)*dqus * dfus;

% Direct flux
tof  = zeros(sus);
for n=1:nmax
  % Cosine Distribution
  dvx = usx - sx(n);
  dvy = usy - sy(n);
  dvz = usz - sz(n);
  normdv = sqrt(dvx.*dvx + dvy.*dvy + dvz.*dvz);

  ca = dvx*rx(n) + dvy*ry(n) + dvz*rz(n);
  ca = ca./normdv;
  ca = ca.*(ca > 0.0);
  ca = ca.^(cosn+1); % cosn+1 for solid angle

  % Flux = N Sum( qo(q,f) dA(f) cos(f) )
  tof  = tof + qo(n)*ca*dA(n);
end
```

```

% back-scattering: integration
bs  = (tof.*(usz<0.0)) + ((0.5*tof).*(usz==0));
bs  = bs.*dAus;
bsi = sum(sum(bs));

% cosine distribution to higher half sphR
bsf = bsi/pi * sin(usf.*(usz > 0.0 ));

% cut
mus = us/2+1;
phiar  = [-usf(mus:sus(1),1)+pi;  usf(mus:sus(1),mus)];
dirar  = [ tof(mus:sus(1),1);     tof(mus:sus(1),mus)];
bsfar  = [ bsf(mus:sus(1),1);     bsf(mus:sus(1),mus)];
tot    = dirar + bsfar;
totn   = tot./max(tot);

figure(2)
hold off
polar(phiar, tot, 'g');
hold on
polar(phiar2,arall, 'g');
polar(phiar, dirar,  'k');
polar(phiar, bsfar, 'r');
hold off

% figure(3)
% surf(Dx, Dy, Dz, tof) %sph
% surf(Dx(21:41,:), Dy(21:41,:), tof(21:41,:))
% shading interp, axis square

out = [phiar, tot, totn];
out = sortrows(out,1);
% 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
`info.txt`	Summary of the parameters used
`E_Prob_xy.txt`	Table of coordinates in coverage space
`CO_cov_p_facet.txt`	CO coverage per facet
`E_CO2.txt`	Transient global CO_2 production
`fort.1000001`	Probability distribution in coverage space for the O-precovered case
`fort.2000001`	Probability distribution in coverage space for the CO-precovered case

```
program EulerDrv
!
!  Creation Date:     12.08.03 11:13:59
!  Last Modification: 16.09.2003 16:25:51
!
!  Author:            Mathias Laurin
!
implicit none
integer*4, parameter    :: LL=60
! big part   LL = 540
! small part LL = 60
integer*4, parameter    :: LL1=LL/2, LL2=LL/4
real*8                  :: rLL, rLL1, rLL2
integer*4               :: k, kkk, MaxIt
integer*4               :: n1, n2
real*8                  :: c1, c1a, c1b, c2, c2a, c2b
real*8                  :: dt, t1, t2
real*8                  :: ProdCO2, eps
real*8                  :: Ftot, Ptot
real*8                  :: kb, pi, Tvel, R, Na
real*8                  :: mco, mo2, nPd
real*8                  :: sCO, sO2, Ct, a_d
real*8                  :: p1, p2
real*8                  :: e, d1, kr
real*8                  :: T, xco, RT
real*8, dimension(2)  :: nu_d, nu_r, en_d, en_r

integer*4  ::  savco2, savproba, co2step, probastep
integer*4  ::  co2file, probafile

real*8, dimension(4)               :: W1, W2
real*8, dimension(4, 0:LL1, 0:LL2) :: W
real*8, dimension(0:LL1, 0:LL2)    :: P

!!!     Parameters
rLL  = real(LL)
rLL1 = real(LL1)
rLL2 = real(LL2)

! General
Ptot     = 1.0d-4         ![Pa = kg.m-1.s-2]
T        = 415.0d0        ![K]
xco      = 52.0d-2
t2       = 20.0d-2        ! % defects
dt       = 1.0d-3
MaxIt    = 95000
! 1200000 @ dt=1.0d-3 == 20 min
! CPU == 750 min on Marvin (LL=540, kkk=2)
!savco2   = 500           ! int(1.0d0/dt)
!savproba = 500
!!if( savproba .gt. MaxIt ) savproba = MaxIt

t1       = 1.0d0-t2
! Constants
kb   = 1.380658d-23       ![J/K]     !! [J] == [Kg.m2.s-2]
Na   = 6.0221367d23       ![mol-1]
R    = Na*kb              ![J.mol-1.K-1] !! [Kg.m2.s-2.mol-1.K-1]
pi   = 4.0d0*datan(1.0d0)

nPd  = 1.53d15            ![cm-2]
nPd  = nPd*1.0d4          ![m-2]
mco  = 28.010d-3/Na       ![kg]
mo2  = 31.999d-3/Na       ![kg]
Tvel = 300.0d0            ![K]

RT   = R*T                ![J.mol-1]

! Desorption
nu_d(1) = 4.0d14          ![s-1]
nu_d(2) = 4.0d14
```

```
en_d(1) = 142.0d3         ![J.mol-1]
en_d(2) = 117.0d3
a_d     = 0.12d0
! Reaction
nu_r(1) = 5.0d7           ![s-1]
nu_r(2) = 5.0d7
en_r(1) = 53.0d3          ![J.mol-1]
en_r(2) = 44.0d3
! Adsorption CO
sCO  = 0.7d0
Ct   = 0.3d0
! Adsorption O2
sO2  = 1.0d0 - 7.4d-4*T

Ftot = xco*dsqrt( 2.0d0*pi*mco*kb*Tvel ) ![Kg.m.s-1]
Ftot = Ftot + (1.0d0-xco)*dsqrt( 2.0d0*pi*mo2*kb*Tvel )
Ftot = Ptot/Ftot                          ![m-2.s-1]

! Rate constants
p1   = xco*Ftot/nPd           ![s-1]
p2   = (1.0d0-xco)*Ftot/nPd   ![s-1]

!!!            Transition Rates (W)
eps = dexp( (en_d(2)-en_d(1))/RT )

open( unit=1, file='CO_cov_p_facet.txt', status='replace', action='write', form='formatted')
do n1 = 0,LL1
   c1 = real(n1)/rLL
   call equilibrium(c1, t1, eps, c1a, c1b)
   write( 1,'(2es20.8e3)' ) c1a, c1b
   do n2 = 0,LL2
      c2  = real(n2)/rLL
      c2a = t1*c2
      c2b = t2*c2

      call TransRate(c1a, c2a, 1, W1) !perfect
      call TransRate(c1b, c2b, 2, W2) !defect

      W(:, n1, n2) = t1*W1(:) + t2*W2(:)
   enddo
enddo
close(1)

!!!            Files
open( unit=1, file='E_Prob_xy.txt',  status='replace', action='write', form='formatted')
do n2 = 0,LL2
   c2 = real(n2)/rLL2
   do n1 = 0,LL1
      c1 = real(n1)/rLL1
      write( 1,'(2f20.6)' ) c1, c2
   enddo
enddo
close(1)

open( unit=21, file='E_CO2.txt', status='replace', action='write', form='formatted')
write(21,'(2a8)') 'Time', 'ME'
write(21,'(2i8)')  0,      0

!!!! Info file
open( unit=1, file='Info.txt', status='replace', action='write', form='formatted')
   write(1,'(a)')           'Parameters:'
   write(1,'(i4,a20)') LL,' adsorption sites'
   write(1,'(a)')           'Constants:'
   write(1,'(a8,es10.4)')   'Ptot ', Ptot
   write(1,'(a8,f6.0)')     'T    ', T
   write(1,'(a8,es10.4)')   'Ftot ', Ftot
   write(1,'(a8,f6.3)')     'x(CO)', xco
   write(1,'(a)')           'Desorption'
   write(1,'(a8,2es15.4)')   'nu   ', nu_d
   write(1,'(a8,2es15.4)')   'en   ', en_d
   write(1,'(a8,f6.4)')     'cov dep', a_d
   write(1,'(a)')           'Reaction'
   write(1,'(a8,2es15.4)')   'nu   ', nu_r
   write(1,'(a8,2es15.4)')   'en   ', en_r
   write(1,'(a)')           'Adsorption CO'
   write(1,'(a8,es10.4)')   'nPd  ', nPd
   write(1,'(a8,f6.4)')     'C(T) ', Ct
   write(1,'(a8,f6.4)')     'S(CO)', sCO
   write(1,'(a)')           'Adsorption O2'
   write(1,'(a8,f6.4)')     'S(O2)', sO2
   write(1,*)
   write(1,'(a8,es12.4e2)') 'p(CO)', p1
   write(1,'(a8,es12.4e2)') 'p(O2)', p2
   write(1,'(a8,es12.4e2)') 'Kr   ', Kr
   write(1,*)
   write(1,'(a8,es12.4e2)') 'eps  ', eps
   write(1,'(a)')        'Time'
   write(1,'(a8,f10.6)') 'dt   ', dt
   write(1,'(a8,i10)')   'Nb It', MaxIt
   write(1,'(a8,f10.1)') 't max', real(MaxIt)*dt
   write(1,'(a, f15.3)') 'Proba saved every [s]', dt*savproba
   write(1,'(a, f15.3)') 'CO2 prod saved every [s]', dt*savco2
close(1)

do kkk = 1,1
   !!!            Init Proba
   co2file   = 21
   if( kkk .eq. 1 ) then
       ! O2 first
       P(:,:)    = 0.0d0
       P(0,LL2)  = 1.0d0
   else         ! kkk .eq. 2
       ! CO first
       p(:,:)    = 0.0d0
       p(LL1,0)  = 1.0d0
   endif

   !!!           Nullify
   probastep = 0
   co2step   = 0

   do k=1, MaxIt

      if( k .le. 2000 ) then
          savco2   = 150
          savproba = savco2
      else
          savco2   = 150
          savproba = savco2
      endif

      probastep  = probastep + 1
      co2step    = co2step   + 1

      call Euler(LL1, LL2, dt, W, P, ProdCO2)

      !!! OUPUT
      ! CO2 Prod
      if( co2step .eq. savco2 ) then
          write( co2file,'(f20.6,es20.6e4)' ) &
                 real(k)*dt, ProdCO2/rLL
          co2step = 0
      endif
      ! proba distribution
```

```fortran
      if( probastep .eq. savproba ) then
          probafile = k*10
          do n2=0,LL2
             do n1=0,LL1
                write( probafile,'(es20.6e4)' ) P(n1,n2)
             enddo
          enddo
          close( probafile )
          probastep = 0
      endif
      !!! END OUTPUT
   enddo
enddo ! kkk

!!!        EOF
write(*,*) 'Euler DONE'
stop
contains
subroutine TransRate(c1, c2, fac, W)
!
!  Returns the matrices of transition rates
!
!            Transition Rates (W)
!    from J Chem Phys 114 4669 + Jens' Thesis
!
!  Creation Date       28.07.03
!  Last modification   19.08.03 16:34:36
!  Author              Mathias Laurin
!
implicit none
integer*4  :: n1, n2, k, fac
real*8     :: r1, r2, c1, c2
real*8     :: dum
real*8, dimension(4)  :: W

! relative coverages
r1    = c1*2.0d0
r2    = c2*4.0d0

! Adsorption CO
dum = 1.0d0 - r1 - Ct *r2
if( dum .ge. 0.0d0 ) then
    W(1) = p1*sCO*dum
else
    W(1) = 0.0d0
endif

! Desorption CO
e    = en_d(fac)*( 1.0d0 - a_d*(r1) )
d1   = nu_d(fac)*dexp( -e/rt )
W(2) = d1*c1

! Reaction
kr   = nu_r(fac)*dexp( -en_r(fac)/rt ) ![s-1]
W(3) = kr*c1*c2

! Adsorption O2
dum = 1.0d0 - r1 - r2
if( dum.gt.0.0d0 ) then
    W(4) = p2*sO2* dum*dum
else
    W(4) = 0.0d0
endif

W(:) = rLL*W(:)

end subroutine TransRate
end program EulerDrv

```

```
subroutine equilibrium(ctot, t1, eps, co1, co2)
!
! Creation Date:     14.08.03 10:55:47
! Last Modification: 18.08.03 16:52:54
! Author:            Mathias Laurin
!
implicit none
real*8, intent(in)   :: ctot       ! total CO coverage
real*8, intent(in)   :: t1, eps    ! t1: fraction of facet 1 (non deffect)
real*8, intent(out)  :: co1, co2   ! CO cov on facet 1, 2

real*8               :: meps, comax, t1meps
real*8               :: a, b, c

comax  = 0.5d0
meps   = 1.0d0-eps
t1meps = t1*meps

if( t1 .eq. 1.0d0 ) then
    co1 = ctot
    co2 = 0.0d0
else
   if( ctot .eq. 0.0d0 ) then
       co1 = 0.0d0
       co2 = 0.0d0
   elseif( ctot .eq. comax ) then
       co1 = comax
       co2 = comax
   else
       !!! J Catal 204 (2001) 378, Eq. [23]
       a    = 0.5d0/t1meps
       b    = ctot*meps + comax*( eps + t1meps )
       c    = -4.0d0*comax*ctot*t1meps

       c    = c + b*b
       if( c .lt. 0.0d0 ) then
           write(*,*) 'ERR: routine equilibrium'; stop
       endif
       c    = dsqrt(c)

       co1   = a*( b-c )
       if( (co1 .lt. 0.0d0) .or. (co1 .gt. comax) ) then
           co1 = a*( b+c )
       endif

       co2   = (ctot - t1*co1)/(1.0d0-t1)
   endif
endif

end subroutine equilibrium
```

```
subroutine Euler(LL1, LL2, dt, W, P, ProdCO2)
   !
   ! Numerical integration of the Master Equation 2D
   ! Euler method
   !
   ! Creation Date       16.07.03 15:07:17
   ! Last Modification   12.08.03 11:36:00
   ! Author              Mathias Laurin
   !
   implicit none
   integer*4, intent(in)  ::  LL1, LL2
   real*8,    intent(in)  ::  dt
   real*8,    intent(out) ::  ProdCO2
   real*8, dimension(4, 0:LL1, 0:LL2), intent(in) :: W
   real*8, dimension(0:LL1, 0:LL2), intent(inout) :: P

   integer*4  ::  n1, n2
   real*8     ::  dum, sp, dpi
   real*8, dimension(0:LL1, 0:LL2)    :: dP

   ! Nullify
   ProdCO2 = 0.0d0
   sp      = 0.0d0

   do n2 = 0,LL2
      do n1 = 0,LL1
         ! Adsorption CO
         if( n1.ge.1 ) then
             dpi = P(n1-1,n2)*w(1, n1-1,n2)
         else
             dpi = 0.0d0
         endif

         ! Desorption CO
         if( n1.lt.LL1 ) then
             dpi = dpi + P(n1+1,n2)*w(2, n1+1,n2)
         endif

         ! Reaction
         if( (n1.lt.LL1) .and. (n2.lt.LL2) ) then
             dum = P(n1+1, n2+1)*w(3,n1+1, n2+1)
             ProdCO2 = ProdCO2 + dum
             dpi     = dpi + dum
         endif

         ! Adsorption O2
         if( n2.ge.2 ) then
             dpi = dpi + P(n1,n2-2)*w(4,n1,n2-2)
         endif

         ! dP/dt  -  Master Equation
         dP(n1,n2) = dpi - P(n1,n2)*( w(1,n1,n2) + w(2,n1,n2) + w(3,n1,n2) + w(4,n1,n2) )
      enddo
   enddo

   ! Update Proba
   do n2=0,LL2
      do n1=0,LL1
         P(n1,n2)    = P(n1,n2) + dt*dP(n1,n2)
         if( P(n1,n2) .lt. 0.0d0 ) then
             write(*,*) 'ERR: divergence'
             stop
         endif

         sp = sp + P(n1,n2)
      enddo
   enddo
   ! Normalisation
   P(:,:) = P(:,:)/sp

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.

```
      program SDickman
      !
      ! Simplified Dickman 1st order - Pd
      !
      ! Numerical integration of the Master Equation 2D
      !            gives the Steady State distribution
      !
      ! Creation Date       21.07.03 12:59:22
      ! Last Modification   08.08.03 16:41:03
      ! Author              Mathias Laurin
      !
      implicit none
      integer*4, parameter  ::  LL = 2000
      integer*4, parameter  ::  LL1=LL/2, LL2=LL/4
      integer*4             ::  k
      integer*4             ::  n1, n2
      real*8                ::  c1, c2
      real*8                ::  rLL, rLL1, rLL2
      real*8                ::  sp, err, sw
      real*8, dimension(4, 0:LL1, 0:LL2) :: W
      real*8, dimension(0:LL1, 0:LL2)    :: Proba, r

      !!!            Parameters
      rLL   = real(LL)
      rLL1  = real(LL1)
      rLL2  = real(LL2)

      !!!            Init Proba
      proba(:,:)          = 0.0d0
      proba(LL1/2,LL2/2)  = 1.0d0

      !!!            Transition Rate
      call TransRate(LL, LL1, LL2, 0.54d0, 415d0, W)
      k = 0

      !!! MAIN
      do
        do n2=0,LL2
          do n1=0,LL1
            if( n1.ge.1 ) then
               r(n1,n2) = proba(n1-1,n2)*W(1,n1-1,n2)
            else
               r(n1,n2) = 0.0d0
            endif
            if( n1.lt.LL1 )
     +         r(n1,n2) = r(n1,n2) + proba(n1+1,n2)*W(2,n1+1,n2)
            if( (n1.lt.LL1) .and. (n2.lt.LL2) )
     +         r(n1,n2) = r(n1,n2) + proba(n1+1,n2+1)*W(3,n1+1,n2+1)
            if( n2.ge.2 )
     +         r(n1,n2) = r(n1,n2) + proba(n1,n2-2)*W(4,n1,n2-2)

            sw = w(1,n1,n2)+w(2,n1,n2)+w(3,n1,n2)+w(4,n1,n2)
            if( sw .gt. 0.0d0 ) r(n1,n2) = r(n1,n2)/sw
          enddo
        enddo

        ! Exit
        err  = sum( sum( (r-proba), dim=2), dim=1 )
        err  = err*err
        k = k+1
```

```
          if( k .eq. 100 ) then
              write(99,*) err
              k=0
          endif
          if(err .lt. 1.0d-7 ) exit

          ! Update proba
          proba(:,:) = r
        enddo
        !!! END MAIN

        ! Normalisation
        !Proba(:,:) = Proba / sum( sum(proba, dim=2), dim=1 )

        open(unit=1, file='SDM_Prob.txt',  status='replace',
     +action='write', form='formatted')
        do n2=0,LL2
           do n1=0,LL1
              c1 = real(n1)/rLL1
              c2 = real(n2)/rLL2
              write(1,'(2f20.6,es20.8e4)') c1, c2, Proba(n1,n2)   ! P_Prob.txt
           enddo
        enddo
        close(1)

        !!!        EOF
        write(*,*) 'Simplified Dickman DONE', err
        stop
        end program SDickman

```

```
        subroutine TransRate(LL, LL1, LL2, xco, T, W)
        !
        !  Returns the matrices of transition rates
        !  to work with KMC.f, Euler.f, Dickman.f
        !
        !  f90 -O3 -c TransRate.f
        !  f90 -O3 -o prog prog.f TransRate.o
        !
        !  Creation Date       28.07.03
        !  Last modification   30.07.03 18:28:04
        !  Author              Mathias Laurin
        !
        implicit none
        integer*4  :: n1, n2, LL, LL1, LL2
        real*8     :: r1, r2, c1, c2, rLL, rLL1, rLL2
        real*8     :: nu_d, en_d, nu_r, en_r, e1, a_d
        real*8     :: T, rt
        real*8     :: Ftot, Ptot
        real*8     :: kb, pi, Tvel, R, Na
        real*8     :: mco, mo2, nPd, xco
        real*8     :: sCO, sO2, Ct
        real*8     :: d1, p1, p2, kr
        real*8     :: dum
        real*8, dimension(4, 0:LL1, 0:LL2)   :: W

        rLL  = real(LL)
        rLL1 = real(LL1)
        rLL2 = real(LL2)

        ! General
        !Ptot = 1.0d-4            ![Torr]
        !Ptot = Ptot/760.0d0*101.325d0 ![Pa = kg.m-1.s-2]
        Ptot = 1.0d-4             ![Pa]
!        T    = 400.0d0            ![K]
!       xco  = 0.47d0
```

```

        ! Constants
        kb   = 1.380658d-23       ![J/K]     !! [J] == [Kg.m2.s-2]
        Na   = 6.0221367d23       ![mol-1]
        R    = Na*kb              ![J.mol-1.K-1] !! [Kg.m2.s-2.mol-1.K-1]
        pi   = 4.0d0*datan(1.0d0)

        nPd  = 1.53d15            ![cm-2]
        nPd  = nPd*1.0d4          ![m-2]
        mco  = 28.010d-3/Na       ![kg]
        mo2  = 31.999d-3/Na       ![kg]
        Tvel = 300.0d0            ![K]

        RT   = R*T                ![J.mol-1]

        ! Desorption
        nu_d = 4.0d14             ![s-1]
        en_d = 142.0d3            ![J.mol-1]
        a_d  = 0.12
        ! Reaction
        nu_r = 5.0d7              ![s-1]
        en_r = 53.0d3             ![J.mol-1]
        ! Adsorption CO
        sCO  = 0.7d0
        Ct   = 0.3d0
        ! Adsorption O2
        sO2  = 1.0d0 - 7.4d-4*T

        Ftot = xco*dsqrt( 2.0d0*pi*mco*kb*Tvel ) ![Kg.m.s-1]
        Ftot = Ftot + (1.0d0-xco)*dsqrt( 2.0d0*pi*mo2*kb*Tvel )
        Ftot = Ptot/Ftot                         ![m-2.s-1]

        ! Rate constants
        kr   = nu_r*dexp(-en_r/rt)    ![s-1]
        p1   = xco*Ftot/nPd           ![s-1]
        p2   = (1.0d0-xco)*Ftot/nPd   ![s-1]

        !!!            Transition Rates (W)
        !    from J Chem Phys 114 4669 + Jens' Thesis
        do n2 = 0, LL2
           do n1 = 0,LL1
              ! absolute coverages
              c1    = real(n1)/rLL
              c2    = real(n2)/rLL
              ! relative coverages
              r1    = real(n1)/rLL1
              r2    = real(n2)/rLL2

              ! Adsorption CO
              dum = 1.0d0 - r1 - Ct *r2
              if( dum.ge.0.0d0 ) then
                  W(1,n1,n2) = p1*sCO*dum
              else
                  W(1,n1,n2) = 0.0d0
              endif

              ! Desorption CO
              e1  = en_d*(1.0d0 - a_d*(r1))
              d1  = nu_d*dexp( -e1/rt )
              W(2,n1,n2) = d1*c1

              ! Reaction
              W(3,n1,n2) = Kr*c1*c2

              ! Adsorption O2
              dum = 1.0d0 - r1 - r2
              if( dum.gt.0.0d0 ) then
```

```
                W(4,n1,n2) = p2*sO2* dum*dum
            else
                W(4,n1,n2) = 0.0d0
            endif
         enddo
      enddo

      W(:,:,:) = rLL*W(:,:,:)

      !!! Info file
      open(unit=1, file='Info.txt',     status='replace',
     +action='write', form='formatted')
      write(1,'(a)')           'Parameters:'
      write(1,'(i4,a20)') LL,' adsorption sites'
      write(1,'(a)')           'Constants:'
      write(1,'(a8,es10.4)')   'Ptot ', Ptot
      write(1,'(a8,f6.0)')     'T    ', T
      write(1,'(a8,es10.4)')   'Ftot ', Ftot
      write(1,'(a8,f6.3)')     'x(CO)', xco
      write(1,'(a)')           'Desorption'
      write(1,'(a8,es10.4)')   'nu   ', nu_d
      write(1,'(a8,es10.4)')   'en   ', en_d
      write(1,'(a8,f6.4)')     'cov dep', a_d
      write(1,'(a)')           'Reaction'
      write(1,'(a8,es10.4)')   'nu   ', nu_r
      write(1,'(a8,es10.4)')   'en   ', en_r
      write(1,'(a)')           'Adsorption CO'
      write(1,'(a8,es10.4)')   'nPd  ', nPd
      write(1,'(a8,f6.4)')     'C(T) ', Ct
      write(1,'(a8,f6.4)')     'S(CO)', sCO
      write(1,'(a)')           'Adsorption O2'
      write(1,'(a8,f6.4)')     'S(O2)', sO2
      write(1,*)
      write(1,'(a8,es12.4e2)') 'p(CO)', p1
      write(1,'(a8,es12.4e2)') 'p(O2)', p2
      write(1,'(a8,es12.4e2)') 'Kr   ', Kr

      close(1)
      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_2 production

```
program MonteCarlo
!
! Creation Date       02.07.03
! Last Modification   24.09.2003 12:26:15
! Author              Mathias Laurin
!
implicit none
integer*4, parameter  ::  LL = 2000
```

```
! big part   LL = 540
! small part LL = 60
integer*4, parameter  ::  LL1=LL/2, LL2=LL/4
integer*4             ::  i, j, k, m, n
integer*4             ::  n1, n2, k1, k2
real*8                ::  rLL, rLL1, rLL2
real*8                ::  c1, c2, r1, r2
real*8                ::  c1a, c1b
real*8                ::  c2a, c2b, eps, def, mdef
real*8                ::  T, xco, RT
real*8                ::  rand, rn, sp, dt
real*8                ::  Time, AbsTime, TimeMx, dt
real*8                ::  CO2
real*8, dimension(4, 0:LL1, 0:LL2) :: S
real*8, dimension(4)               :: s1, s2
real*8, dimension(0:LL1, 0:LL2)    :: W3
real*8, dimension(0:LL1)           :: pr1
real*8, dimension(0:LL2)           :: pr2

real*8  :: dum

def  = 00.0d-2
mdef = 1.0d0 - def
T    = 415.0d0
xco  = 0.48d0

RT   = (1.380658d-23*6.0221367d23)*T
eps  = dexp( (117.0d3 - 142.0d3) /RT )

!!!            Parameters
if( LL .eq. 540 ) then
    timemx    = 250.0d3
    dt        = 1.0d2
elseif ( LL .eq. 60 ) then
    timemx    = 300.0d0
    dt        = 1.0d-2
elseif( LL .eq. 2000 ) then
    timemx    = 500.0d3
    dt        = 1.0d2
else
    stop
endif

rLL       = real(LL)
rLL1      = real(LL1)
rLL2      = real(LL2)

! Initial Coverages
n1    = LL1
n2    = 0
if( n1.gt.LL1 ) write(*,*) 'ERR: Coverage of CO too large'
if( n2.gt.LL2 ) write(*,*) 'ERR: Coverage of O2 too large'

! nullify
Pr1(:)   = 0.0d0
Pr2(:)   = 0.0d0
Time     = 0.0d0
AbsTime  = 0.0d0
CO2      = 0.0d0

do k1 = 0, LL1
   c1 = real(k1)/rLL
   call equilibrium(c1, mdef, eps, c1a, c1b)
   do k2 = 0,LL2
      c2 = real(k2)/rLL
      c2a = mdef*c2
      c2b = def*c2

      call TransRate(LL, c1a, c2a, xco, T, 1, s1)
      call TransRate(LL, c1b, c2b, xco, T, 2, s2)

      S(:,k1,k2) = mdef*S1(:) + def*S2(:)
   enddo
enddo

W3(:,:) = s(3,:,:)

! Summation
s(2,:,:) = s(1,:,:) + s(2,:,:)
s(3,:,:) = s(2,:,:) + s(3,:,:)
s(4,:,:) = s(3,:,:) + s(4,:,:)

do j=0,LL2
   do i=0,LL1
      if( s(4,i,j) .eq. 0.0d0 ) then
          write(*,*) 'KMC FAILED'
          stop
      endif
   enddo
enddo

! Normalisation
s(4,:,:) = 1.0d0/s(4,:,:)
s(1,:,:) = s(1,:,:)*s(4,:,:)
s(2,:,:) = s(2,:,:)*s(4,:,:)
s(3,:,:) = s(3,:,:)*s(4,:,:)

!!!           Files
open(unit=1, file='KMC_Pop.txt',   status='replace', action='write', form='formatted')
open(unit=2, file='KMC_ProbaCO.txt',  status='replace', action='write', form='formatted')
open(unit=3, file='KMC_ProbaO.txt',  status='replace', action='write', form='formatted')
open(unit=4, file='KMC_CO2.txt',  status='replace', action='write', form='formatted')

write(1,'(3a8)') 'Time','CO','O'

open(unit=33, file='Info.txt', status='replace', action='write', form='formatted', position='append')
write(33,*)
write(33,'(a8)')       'Ini.Cov.'
write(33,'(a8,f6.4)')  'CO   ',real(n1)/rLL
write(33,'(a8,f6.4)')  'O    ',real(n2)/rLL
write(33,'(a8)')       'Time'
write(33,'(a8,f12.0)') 'TimeMax',timemx
write(33,'(a8,f12.0)') 'TimeStep',dt
write(33,'(a8,f12.0)') 'T',T
write(33,'(a8,f6.4)')  'xco',xco
write(33,'(a8,f6.4)')  'def',def
close(33)

!!! MAIN
do
  Time  = Time  + S(4,n1,n2)
  if( Time .ge. dt ) then
     AbsTime = AbsTime + Time
     Time    = 0.0d0
     write(1, '(es12.6,2f12.8)' ) AbsTime, real(n1)/rLL, real(n2)/rLL
     write(4, '(2es20.6)' )       AbsTime, CO2
     CO2 = 0.0d0
  endif
  if( AbsTime .ge. TimeMx ) exit

  rn    = rand()

  ! Adsorption CO
  if( rn .le. S(1,n1,n2) ) then
```

```
     n1      = n1+1
     Pr1(n1) = pr1(n1) + S(4,n1,n2)
     Pr2(n2) = pr2(n2) + S(4,n1,n2)

  ! Desorption CO
  elseif( rn .le. S(2,n1,n2) ) then
      n1      = n1-1
      Pr1(n1) = pr1(n1) + S(4,n1,n2)
      Pr2(n2) = pr2(n2) + S(4,n1,n2)

  ! Reaction
  elseif( rn .le. S(3,n1,n2) ) then
      n1      = n1-1
      n2      = n2-1
      Pr1(n1) = pr1(n1) + S(4,n1,n2)
      Pr2(n2) = pr2(n2) + S(4,n1,n2)

  ! Adsorption O2
  else
      n2      = n2+2
      Pr1(n1) = pr1(n1) + S(4,n1,n2)
      Pr2(n2) = pr2(n2) + S(4,n1,n2)
  endif
  CO2     = CO2 + W3(n1,n2)
enddo
!!! END MAIN

! Normalisation
pr1(:) = pr1/sum(pr1)
pr2(:) = pr2/sum(pr2)

! Save probabilities
write(2,'(2a8)') 'CO','P.CO'
write(3,'(2a8)') 'O', 'P.O'
do i=0,LL1
   c1 = real(i)/rLL1
   ! KMC_ProbaCO.txt
   write(2,'(f12.8, es20.6e4)') c1, Pr1(i)
enddo
do j=0,LL2
   c2 = real(j)/rLL2
   ! KMC_ProbaO.txt
   write(3,'(f12.8, es20.6e4)') c2, Pr2(j)
enddo

!!!        EOF
write(*,*) 'KMC DONE'
close(1)
close(2)
close(3)
stop
end
```