UNIVERSITY OF CALIFORNIA, SAN DIEGO

Numerical Study of Reaction in Porous Catalysts under Composition Modulation

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Chemical Engineering

By

Hsu-Wen Hsiao

Committee in charge:

Professor Richard K. Herz, Chair

Professor Thomas R. Bewley

Professor Pao C. Chau

Professor Kalyanasundaram Seshadri

Professor Paul K. Yu

2010

The Dissertation of Hsu-Wen Hsiao is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2010

TABLE OF CONTENTS

LIST OF SYMBOLS

A Pre-exponential factor of Arrhenius equation

A^p Cross-sectional area of pellet face, m^2

A^s Total surface area of solid catalyst, m^2

$C^g_{b,i}$ Concentration of component i of the mass transfer boundary layer, mol/m^3

C^g_i Concentration of component i in gas phase, mol/m^3

$C^g_{\max}$ Maximum gas concentration, mol/m^3

C^s_i Concentration of component i over solid phase, mol/m^2

C^s_{tot} Total surface concentration, mol/m^2

D_{AB} Binary diffusion coefficient, mol/m^2

D_{eff} Effective diffusivity, m^2/s

$D_{eff,i}$ Effective diffusion coefficient of component i, m^2/s

D_K Knudsen diffusion coefficient, m^2/s

E_a Activation energy, J/mol

J_A Molar diffusion flux of component A, $mol/s \cdot m^2$

K_A Equilibrium constant

K_B Boltzmann constant, 1.38×10^{-23} $m^2 \cdot kg/s^{-2} \cdot K^{-1}$

L Reactor thickness, m

M Molecular weight, kg/mol

N Number of total components

N_D Number of dependent components

N_I Number of independent components

N_T Order of temperature dependence

N_v Avogadro constant, 6.22×10^{23} mol^{-1}

P Pressure, Pa

R Ideal gas constant, 8.314 J/mol·K

S_{ads} Sticking probability

S_c Surface area per unit mass of catalyst, m^2/kg

T Temperature, K

V Total volume of reactor, m^3

V^g Volume of gas phase in reactor, m^3

Z Dimensionless coordinate

c Speed of traveling wave, m/s

d Collision diameter of gas molecule, m

$f_0(t)$ Modulation function on the obundary

k Rate constant

k_{ads} Rate constant of adsorption

k_{des} Rate constant of desorption

k_{ext} External mass transfer coefficient, m/s

k_{rxn} Rate constant of surface reaction

l Mean free path, m

m Mass of gas molecule, kg

n_i Reaction order of component i

q Volume flow rate, m^3/s

$q_{in,i}$ Volume flow rate or component i in feed gas, m^3/s

r Reaction rate

r_{ads} Reaction rate of adsorption

r_{des} Reaction rate of desorption

r_i Reaction rate of component i

r_m Mean pore radius, m

r_{rxn} Reaction rate of surface reaction

$\langle u \rangle$ Average speed of gas, m/s

z_{coll} Collision frequency of gas, molecules/$m^2 \cdot s$

$\mathbf{C}$ Vector of components

$\mathbf{C}_I$ Vector of independent components

$\mathbf{C}_D$ Vector of dependent components

$\mathbf{L}$ Linear part of differential equation

$\mathbf{M}$ Formula matrix

$\mathbf{M}_D$ Formula matrix of dependent component

$\mathbf{M}_I$ Formula matrix of independent component

$\mathbf{R}$ Vector of reaction rates

$\mathbf{R}_C$ Rate constant of reaction rates

$\mathbf{R}_I$ Integer array of reaction rates

$\mathbf{R}_L$ Location array of reaction rates

$\mathbf{S}$ Stoichiometric matrix

$\mathbf{S}_\Psi$ Stoichiometric matrix of gas component

$\mathbf{S}_\Theta$ Stoichiometric matrix of solid component

$\Psi_{b,i}$ Dimensionless gas concentration in the mass transfer boundary layer

Ψ_i Dimensionless gas concentration

Θ_i Surface coverage

α Surface-to-gas capacity ratio

ε Porosity

ρ_N Number density of gas, molecule/m^3

ρ_c Density of porous catalyst, kg/m^3

ρ_i Normalized reaction part of component *i*, s^{-1}

σ Area occupied by one mole of adsorbent, m^2/mol

τ_d Diffusion time constant, s

τ_{des} Time constant of desorption, s

τ_l Space time of the CSTR, s

τ_{tor} Tortuosity factor

∇ Del operator

ϕ_{ij} Thiele modulus of gas component *i* to rate constant *j*

η Effectiveness factor

LIST OF FIGURES

LIST OF TABLES

VITA

2002 Bachelor of Science, National Tsing Hua University, Taiwan

2006 Master of Science, University of California, San Diego

2010 Doctor of Philosophy, University of California, San Diego

ABSTRACT OF THE DISSERTATION

Numerical Study of Reaction in Porous Catalysts under Composition Modulation

by

Hsu-Wen Hsiao

Doctor of Philosophy in Chemical Engineering

University of California, San Diego, 2010

Professor Richard K. Herz, Chair

A numerical simulator for modeling heterogeneous catalysis is developed to study the transient response of gas-solid kinetics in porous catalysts under composition modulation. A lumped system and a one-dimensional model are built in this numerical simulator. The numerical simulator, Transcat, is a general solver, which can handle unlimited components, arbitrary kinetics and various type of boundary conditions. An original algorithm of element constraint method is developed and also successfully integrated in Transcat to ensure the conservation of total mass and element in a numerical computation. With multiple numerical methods and the theory of stiffness-reduced modeling, Transcat can compute stiff problems efficiently. This general solver is

validated by case studies of partial-equilibrium assumption, steady-state profiles and analytical solution of Fisher-Kolmogorov equation.

Carbon monoxide oxidation over Pt/Al_2O_3 catalyst under periodic forcing is studied by numerical simulation using Transcat. Transcat is capable of fitting the experiment data and explain the behavior of CO oxidation under transient operation conditions. Frequency optimization of various input signal to maximize the turnover rate of CO is performed in a series of numerical experiments in Transcat. At optimum frequency, the characteristic spatiotemporal pattern of turnover rate is continuous, symmetric and boundary-attached, and the saturated storage of CO over the catalytic site is at minimum.

Chapter 1 Introduction

1.1 Dynamic Heterogeneous Catalysis

Heterogeneous catalysis involves mass transport and chemical reaction between a fluid and a solid phase. Engineering in heterogeneous catalysis emerged in the early twentieth century since iron and ruthenium were first used as catalysts in large-scale ammonia production [1]. Other industrial applications such as cracking, isomerization and re-forming of hydrocarbon processes in petroleum refinery also require the presence of solid catalysts [2]. Heterogeneous catalysis is also important in emission control of exhaust gas pollution. Automobile catalytic converters are usually made of precious metal catalyst like platinum, palladium and rhodium embedded in porous matrix to reduce the discharge of poisonous gas into the atmosphere [3].

Most industrial catalytic processes are made as continuous as possible in order to achieve economical large-scale production. Sometimes porous catalysts have to be operated under transient conditions such as the start-up and shutdown. Some porous catalysts are always operated under periodic feed cycling rather than steady state to increase the yield and improve the reactor performance [4]. In automobile emission control, three-way catalytic converter (TWC), the reactor to detoxification of automobile exhaust gas, always processes intermittent feed from an internal combustion engine. In these heterogeneous catalytic systems, it is important to understand the dynamic behavior of surface kinetics over the catalyst and mass transport within and between multiple phases. The subject of this research work is to develop a model of heterogeneous

catalytic reactors based on the theories of mass transport, detailed gas-solid kinetics and stoichiometric balance of chemical reactions. A general computer solver is build based on these scientific premises and it can accurately and efficiently model the transient behavior of catalytic reactors. This model can further be used to predict the reactor response to various transient operation conditions to help improve the design of heterogeneous catalytic reactors and to optimize the productivity, yield or selectivity of catalytic processes.

Modeling the heterogeneous catalysis of a fluid-solid system requires the knowledge of mass transport in all the phases involved. For the fluid phase, separated mass balance equations for each of phases are required to model both transport phenomena and catalytic reactions. In the solid phase, the chemical species are usually assumed to be stationary, where the surface concentration varies only due to the process of adsorption, desorption and reaction. In order to obtain higher active surface area for the solid catalysts, they are usually made highly porous. The large surface area increases the conversion of reactant tremendously but the pore structure slows down mass transport in the gas phase. The combination of slow diffusion and fast surface reaction usually makes the model's partial differential equations (PDEs) stiff [5,6]. Small time steps and long simulation time are required to solve these equations numerically and sometimes it is impractical to compute stiff PDEs of a diffusion-reaction problem even on a fast personal computer. For a porous catalytic system operated under transient conditions, the stiffness not only comes from the intrinsic pore structure itself but also the sudden change of the boundary condition. The key challenge of this research work is to compute the stiff

unsteady PDEs with diffusion resistance and the detailed microkinetics but still be efficient and feasible on a personal computer.

The scope of this research work is to model the heterogeneous catalytic reaction in porous solid under dynamic operation conditions while considering detailed surface kinetics and diffusion resistance. Developing a fast, accurate and comprehensive algorithm that can handle most gas-solid kinetics was a key objective. The concept of element conservation and stoichiometric balance of chemical reaction are also built into the simulator to ensure the total mass of each element is conserved between every step of solving the PDEs. The computer program that was developed is validated by existing solutions of unsteady diffusion-reaction systems and can fit experimental data based on the theoretical parameters of gas transport and surface kinetics. Finally, multiple numerical experiments are carried out to show the ability of demonstrating the behavior of a heterogeneous catalytic process and optimizing the reactor performance by numerical simulations.

1.2 Historical Review on Detailed Modeling of Catalytic Reactions

Software packages [9,10] for solving partial differential equation systems developed in the late 70's make modeling the transient behavior of detailed gas-solid kinetics inside a porous reactor feasible. Oh and Hegedus (1982) [11] developed the first model of transient adsorption and desorption kinetics with diffusion in a catalyst disc by using the ODEONE algorithm by Sincovec and Madsen (1975) [9]. In their numerical simulations, they used separate steps to model carbon monoxide adsorption and

desorption process, which can explain the experimental transient infrared spectra of adsorbed carbon monoxide on platinum catalyst. They also found that the carbon monoxide concentration is in close approach to adsorption equilibrium because changing both adsorption and desorption rate constants by same factor results in similar adsorption response.

Herz and Marin (1980) [12] developed the first steady state model with gas diffusion and detailed surface kinetics of carbon monoxide oxidation in porous catalyst. The idea of element conservation was implied in their model in computing the empty catalytic sites and their model shows good agreement to experimental data. The parameters in their model were later adopted by Cho (1983) [13] to develop the first transient model with detailed surface kinetics. He used a two-phase, non-equilibrium adsorption model to study the transient response of CO oxidation with diffusion over porous Pt/Al_2O_3 catalyst. Cho compared the numerical simulation results of both time-averaged rate and overall reaction rate to experimental measurements of average conversion during composition cycling of CO and O_2 Feed. He concluded that the cyclic operation gives better conversion than steady-feed operation under certain conditions.

Hoebink et al. (1999) [14] studied the CO oxidation over $Pt/\gamma\text{-}Al_2O_3$ in a fixed bed reactor with cycling frequencies of both CO and O_2 feed up to 10 Hz in an attached mixing chamber. They also used the kinetics of CO oxidation provided by Herz and Marin (1980) [12]. The model of the mixing chamber is an isothermal continuous stir tank reactor (CSTR) with diffusive flux through gas-solid interface and for the solid catalysts are unsteady diffusion reaction equations. The partial differential equation

system is solved by backward differential formula (BDF) using the D02NHF routine of NAG Fortran library [15]. They compared the simulation result to experiments and confirm that the feed composition modulation is affected by diffusion inside catalyst pellets if the time scale of diffusion inside the catalyst is much smaller than the time scale of the forced composition cycling. They also suggested that CO might adsorb on oxygen-covered sites and propose a more detailed mechanism of CO oxidation with second layer adsorption of CO molecules, but they need further improvement in their model to verify this hypothesis.

Hayes, Mukadi et al. (2002) modeled the detailed kinetics of a single channel monolith (SCM) in TWC [16,17]. They developed the first non-isothermal model with detailed kinetics and diffusion resistance to study the light-off curve of TWC from cold start. Most of their gas-solid kinetics is composed of elementary reaction steps and only a few rates of hydrocarbon oxidation are algebraic equations. This transient model is solved either by sixth-order explicit Runge-Kutta method or by Newton-Krylov solver, depending on the stiffness. The element conservation concept has also been used to compute the surface concentration of empty catalytic sites on alumina, ceria and other noble metals. They concluded that the concentration of gas molecules and adsorbed species undergo large changes during the light-off process and the diffusion limitation is significant in the washcoat even at low operation temperature. When modeling under composition cycling, the dynamic response of surface concentration in the washcoat can develop surprising patterns.

Marek et al. (2004) carried out experimental studies on a three-way-catalyst monolith and also modeled its behavior by numerical computation [18-21]. The microkinetics used in their model is mainly detailed elementary steps except for a few hydrocarbon oxidations on the catalytic site. They modeled the monolith channel with diffusion in thin porous washcoat layer by both non-isothermal CSTR and plug flow reactor (PFR). These models are computed numerically by the Livermore solver for ordinary differential equations (LSODE) [10]. The LSODE is an adaptive solver, which switches between the semi-implicit Adams-Moulton method and backward differential formula (BDF) depending on the stiffness. The developed model was then applied in studying the outlet hydrocarbon conversion to the modulation of inlet feed concentration or temperature.

Koci et al. (2004) [18] modeled the periodic operation of the inlet oxygen concentration over Pt-Rh/Ce/γ-Al_2O_3 catalyst in a monolith channel of TWC to help designing the control of air-fuel ratio by the lambda-sensor of automobile engine. Higher time-averaged conversion of outlet CO, HC and NO_x around the stoichiometric point was found by frequency and amplitude modulation of the inlet oxygen rather than stationary operation. They can predict the light-off behavior of TWC at low temperature with oscillating inlet O_2 concentration. They also studied the hysteresis phenomena, autocatalytic loop and chaotic behavior with period modulation of inlet concentration over Pt (110), Pt/γ-Al_2O_3 or Pt/Ce/γ-Al_2O_3 in TWC [19,20,21]. Their simulation results qualitatively agree with experiments and the computed spatiotemporal pattern within the washcoat layer can help understanding of the dynamic response of TWC monolith to

transient operation conditions.

Herz and Nett-Carrington (2002) studied the spatiotemporal pattern of CO oxidation with diffusion resistance in single-pellet reactor with Pt/Al_2O_3 catalyst [22,23]. The surface kinetics in their model is composed of elementary steps of competitive adsorption, reactant inhibition on bimodal distribution of active sites on catalytic surface. This model is computed by semi-implicit numerical method with the assumption of quasi-equilibrium adsorption and reduction of the surface-to-gas capacity ratio. The model was able to predict the intrapellet spatiotemporal patterns, which explained the transient response of concentration gradients in experimental measurement [22]. The surface adsorption, desorption and reaction rate change rapidly in different magnitude of order as the temperature increased or decreased. The light-off and quenching behavior of carbon monoxide oxidation was observed during temperature ramps and the spatiotemporal patterns are analyzed qualitatively with various layer thickness of catalyst pellet [23].

There are other notable works in modeling heterogeneous catalytic reaction under dynamic condition. Aida et al. (1997) modeled the diffusion, adsorption and desorption of NO-CO over supported noble metal catalyst under periodic operation [24,25]. A detailed Langmuir-Hinshelwood reaction mechanism of NO and CO interaction on the noble metal surface is used in their model. They find the cycling of strong adsorbed species can effectively improve the reactor performance. Peskov (2000) developed a mathematical model of CO oxidation over porous zeolite particle with embedded Pd catalyst [26]. He studied and compared the spatiotemporal pattern with or without diffusion to each other under CO pressure oscillation. Tuttlies et al. approximated the gas

concentration in the porous layer by a step-like profile to the study the NO_x storage and regeneration in nanoparticles of Barium catalyst [27].

Scholz et al. (2007) studied the NO_x storage and reduction (NSR) over Pt-Ba/γ-Al2O3 catalyst in a packed bed reactor (PBR) [28]. They modeled the PBR by isothermal plug flow reactor with constant pressure. The diffusion in catalyst cluster, microkinetics on the surface and element conservation on catalytic sites are considered in their mathematic model as well. The resulting partial differential system is solved by commercial software, gPROMS. There are other software packages available in modeling and simulation of dynamic heterogeneous catalysis in porous solids although these simulators may not be developed completely based on the microkinetics, pore diffusion and multiple phase transport. Research in fluid-solid kinetics has been done in computer programs such as Comsol and DetChem. Comsol is capable of modeling unsteady and non-isothermal catalysis, but the dynamics in porous layers is computed by a lumped system [29-31]. DetChem can model detailed surface kinetics in catalytic monolith. The gas transport in the monolith channels is computed either by fixed two-dimensional flow pattern or by plug flow. The gas flow near the surface is approximated by a boundary layer model instead of diffusive transport [32,33].

Most literature studies discussed above consider both diffusion and reaction behavior in modeling gas-solid catalysis under transient conditions. There are still plenty of publications in modeling porous catalysts either at steady state or in a lumped system, where the diffusion effect is ignored. However, diffusion resistance has important influence on the performance of porous catalysts, which should not be ignored in a detailed model [12-27]. Besides, modeling the dynamic response of porous catalysts is

indispensible, since a significant amount of industrial catalytic processes are always operated under periodic cycling of feed to increase the conversion [34]. Current numerical algorithms for solving the unsteady diffusion-reaction equations of porous catalysts are developed only for the surface kinetics of interest, and other commercial software packages are not as specific in either the diffusion effect or the interaction between the fluid and solid phases. The goals of this research work is to create a comprehensive algorithm can handle most complex surface kinetics, to build a general solver can efficiently compute the stiff unsteady diffusion-reaction equation, to develop a model can better explain the transient phenomena of heterogeneous catalysis, and to design an numerical simulator can effectively predict the behavior of catalytic process under dynamic operation conditions.

1.3 Advances in Modeling Porous Catalysts

The final goal of this research work is to find the best dynamic operating conditions of heterogeneous catalytic process in a porous reactor by numerical simulation. Carrying out comprehensive lab experiments searching for the best temporal contacting pattern of feed for catalytic reaction requires tremendous effort. Seeking the optimized operation condition of a catalytic process on a large-scale production plant by empirical trial-and-error procedure is expensive. If only simulations of these heterogeneous catalytic processes on a personal computer can do the job, it could be a great saving in both time and money. TransCat, which is the numerical simulator developed based on the scientific premises of this research work, can make this happen.

The origin of Transcat comes from the dynamic diffusion reactor (DDR), which

has been used to study the carbon monoxide oxidation over porous Pt/Al_2O_3 catalyst in our group [22]. An early work has been done in developing a numerical algorithm to study the kinetics of bimodal distribution of active sites on catalyst surface for carbon monoxide oxidation. The modeling equations for CO, O_2 and CO_2 in the gas phase are the unsteady diffusion-reaction equations with a surface-to-gas capacity ratio in the nonlinear reaction term to deal with the interchange of chemicals between phases. As for the solid phase, all the surface species are solved by ordinary differential equations with catalytic reactions, and the bimodal empty catalytic sites are computed by algebraic equations of stoichiometric balance. Transcat is designed to simulate not only the carbon monoxide oxidation in the DDR but also any one-dimensional unsteady diffusion-reaction system in a porous solid catalyst.

To make Transcat a general solver, it is designed to be able to handle unlimited chemicals in each phase, to compute most gas-solid kinetics and to meet the configuration of other types of reactors by various boundary conditions. Except for the one-dimensional model, a lumped system, where the diffusion in pores is ignored, is also built in Transcat. The lumped system, although not as accurate as the one-dimensional model, can still give a fast, general idea of how a surface kinetics works. Furthermore, the two models can be compared to each other to show whether the diffusion resistance is important.

The major difference between Transcat and other numerical algorithm is the ability to handle arbitrary surface kinetics. The surface kinetics in literature work is sometimes simplified, linearized, and most of them are written on a case-by-case basis [16,17,24,25,35]. Transcat can compute any combination of detailed elementary reaction

and non-elementary steps with simple algebraic rate equations, which covers most microkinetics on catalyst surface [36]. Only a simple text file is required to input the surface kinetics instead of rewriting the function of entire reaction part in the program. Even with the complexity of mixing all kinds of surface kinetics, the calculation speed can still be boosted by the sparse matrix operation techniques [37-39]. Transcat can also compute the first derivative of reaction rates in the Jacobian matrix when the Newton's method is used for solving the implicit numerical methods.

The continuity constraint method, which enforces the continuity differential constraint while solving the momentum equation, has been widely used in computational fluid dynamic [40,41]. A similar concept of element constraint method (ECM) based on the stoichiometric balance of chemical reaction and conservation of elements is applied in Transcat. The stoichiometric degree of freedom and the actual available mass balance equation has been discussed in literature for only closed systems [7,8]. An original set of element conservation equations for an open and transient heterogeneous catalysis system is derived and has been successfully implemented in the numerical algorithm of Transcat. The ECM algorithm is provided as an optional feature in Transcat, which can ensure the total mass conservation during reaction and transport processes.

To deal with stiff partial differential equations, the code of Transcat is well structured to integrate multiple numerical methods and all of them are time adaptive. User can choose from explicit, semi-implicit and implicit methods according the stiffness of the modeling problem and the desired accuracy. These methods can also be cross-compared to each other to ensure the correctness of implementation. A theory of reducing the stiffness by adjusting the relative reaction rate based on the partial equilibrium

assumption has been tested under certain circumstances. If the reaction rate constant is much larger than the diffusion time constant, the stiffness-reduced simulation can give accurate spatiotemporal patterns of both gas concentration and surface coverage and can save computing time inversely proportional to the decrease in the maximum rate constant.

Temperature affects the catalyst performance and outlet conversion dramatically especially under oscillation of inlet concentration [16]. The ability to solve the energy equation simultaneously with the mass balance equation is currently not developed and will be incorporated in the future. In order to study the performance of a spatially isothermal catalytic reactor under thermal variations, a temperature control functions is integrated to Transcat to modulate the temperature of both the reactor and boundary. Numerical experiments such as the behavior of light-off, quenching and shift of thermodynamic equilibrium of catalytic reactor can still be studied by Transcat.

The correctness of Transcat is validated by several case studies like inert gas diffusion, partial equilibrium approximation and steady state profile. The simulation result can further match the analytical solutions of Fisher-Kolmogorov equation and can fit experimental data as well. The most valuable part of Transcat is the ability to find the best dynamic operating condition of a heterogeneous catalytic process with only limited information. Based on a set of parameters fit to experimental data, a series of numerical experiments can be done in Transcat to find the best temporal operating pattern for this process. The available modulation variables include waveform, frequency, amplitude and phase shift in both concentration and temperature.

This research work advances the modeling and simulation of porous catalysts in several aspects. Transcat is a comprehensive numerical simulator, which can model any

unsteady diffusion reaction process in a porous solid catalyst. The cutting edge algorithm gives Transcat the ability to handle various types of boundary conditions, unlimited components and arbitrary kinetics. An original element constraint method can guarantee the conservation of mass for open system during numerical computation. The well-structured code of Transcat balance efficiency and accuracy by embracing multiple numerical methods and gives it the possibility to incorporate newly developed numerical methods in the future to overcome stiffness. Transcat is able to fit experimental data and is also validated to other mathematical models based on scientific premises to provide dependable simulation results. With the built-in function of modulation in both concentration and temperature, it can explain, predict and optimize the dynamic response of heterogeneous catalytic process subject to transient operation conditions.

Reference:

[1] Smil, V. (2001). "Enriching the earth: Fritz Haber, Carl Bosch, and the transformation of world food production." Cambridge, Mass., MIT Press.

[2] Meyers, R. A. (2004). Handbook of petroleum refining processes. New York, McGraw-Hill.

[3] Yamagata, H., Institute of Materials Minerals and Mining., et al. (2005). "The science and technology of materials in automotive engines." Cambridge, Woodhead Publishing and Maney Publishing, CRC Press: xii, 318 p.

[4] Silveston, P. L. (1998). "Composition modulation of catalytic reactors." Amsterdam, Gordon and Breach Science.

[5] Beers, K. J. (2007). "Numerical methods for chemical engineering: applications in Matlab®." Cambridge, UK; New York, Cambridge University Press.

[6] Hairer, E. and G. Wanner (1993). "Solving ordinary differential equations II. Stiff and differential-algebraic problems." Heidelberg; New York, Springer.

[7] Smith, W. R. and R. W. Missen (2003). "Mass conservation implications of a reaction mechanism." Journal of Chemical Education 80(7): 833-838.

[8] Silbey, R. J. and R. A. Alberty (2001). Physical chemistry. New York, Wiley.

[9] Sincovec, R. F. and N. K. Madsen (1975). "Software for Nonlinear Partial Differential Equations." ACM Trans. Math. Softw. 1(3): 232-260.

[10] Hindmarsh, A. C. (1983). "ODEPACK, A Systematized Collection of ODE Solvers. Scientific computing: applications of mathematics and computing to the physical sciences." R. S. Stepleman. Amsterdam: 55-64.

[11] Oh, S. H. and L. L. Hegedus (1982). "Dynamics of High-temperature Carbon-monoxide Chemisorption on Platinum Alumina by Fast-response IR Spectroscopy." Acs Symposium Series 178: 79-103.

[12] Herz, R. K. and S. P. Marin (1980). "Surface-chemistry Models of Carbon-Monoxide Oxidation on Supported Platinum Catalysts." Journal of Catalysis 65(2): 281-296.

[13] Cho, B. K. (1983). "Dynamic behavior of a Single Catalyst Pellet .1. Symmetric Concentration Cycling During CO Oxidation Over Pt/Al2O3." Industrial & Engineering Chemistry Fundamentals 22(4): 410-420.

[14] Hoebink, J., A. J. L. Nievergeld, et al. (1999). "CO oxidation in a fixed bed reactor with high frequency cycling of the feed." Chemical Engineering Science 54(20): 4459-4468.

[15] NAG Fortran Library Manual, Mark 22. NAG Ltd., Oxford, 2009.

[16] Mukadi, L. S. and R. E. Hayes (2002). "Modelling the three-way catalytic converter with mechanistic kinetics using the Newton-Krylov method on a parallel computer." Computers & Chemical Engineering 26(3): 439-455.

[17] Hayes, R. E., L. S. Mukadi, et al. (2004). "Three-way catalytic converter modelling with detailed kinetics and washcoat diffusion." Topics in Catalysis 30-1(1-4): 411-415.

[18] Koci, P., M. Kubicek, et al. (2004). "Periodic forcing of three-way catalyst with diffusion in the washcoat." Catalysis Today 98(3): 345-355.

[19] Koci, P., M. Kubicek, et al. (2004). "Modeling of three-way-catalyst monolith converters with microkinetics and diffusion in the washcoat." Industrial & Engineering Chemistry Research 43(16): 4503-4510.

[20] Koci, P., V. Nevoral, et al. (2004). "Nonlinear dynamics of automobile exhaust gas converters: the role of nonstationary kinetics." Chemical Engineering Science 59(22-

23): 5597-5605.

[21] Marek, M., M. Schejbal, et al. (2006). "Oscillations, period doublings, and chaos in CO oxidation and catalytic mufflers." Chaos 16(3).

[22] Nett-Carrington, L. C. and R. K. Herz (2002). "Spatiotemporal patterns within a porous catalyst: dynamic carbon monoxide oxidation in a single-pellet reactor." Chemical Engineering Science 57(8): 1459-1474.

[23] Herz, R. K. (2004). "Spatiotemporal patterns in a porous catalyst during light-off and quenching of carbon monoxide oxidation." Chemical Engineering Science 59(19): 3983-3991.

[24] Aida, T., R. Kobayashi, et al. (1997). "Periodic operation of NO-CO reaction over Pt/Al2O3 - Effect of intraparticle diffusion." Kagaku Kogaku Ronbunshu 23(6): 962-968.

[25] Aida, T., D. Na-Ranong, et al. (1999). "Effect of diffusion and adsorption-desorption on periodic operation performance of NO-CO reaction over supported noble metal catalysts." Chemical Engineering Science 54(20): 4449-4457.

[26] Peskov, N. V. (2000). "Spatiotemporal patterns in a model of heterogeneous reaction in a porous catalyst particle." Physica D 137(3-4): 316-332.

[27] Tuttlies, U., V. Schmeisser, et al. (2004). "A new simulation model for NOx storage catalyst dynamics." Topics in Catalysis 30-1(1-4): 187-192.

[28] Scholz, C. M. L., V. R. Gangwal, et al. (2007). "Model for NOx storage/reduction in the presence of CO2 on a Pt-Ba/gamma-Al2O3 catalyst." Journal of Catalysis 245(1): 215-227.

[29] More, H., R. E. Hayes, et al. (2006). "The effect of catalytic washcoat geometry on light-off in monolith reactors." Topics in Catalysis 37(2-4): 155-159.

[30] Wickman, B., A. Lundstrom, et al. (2007). "Modeling mass transport with microkinetics in monolithic NOx storage and reduction catalyst." Topics in Catalysis 42-43(1-4): 123-127.

[31] Perdana, I., D. Creaser, et al. (2007). "Modelling NOx adsorption in a thin NaZSM-5 film supported on a cordierite monolith." Chemical Engineering Science 62(15): 3882-3893.

[32] Schwiedernoch, R., S. Tischer, et al. (2003). "Experimental and numerical study on the transient behavior of partial oxidation of methane in a catalytic, monolith." Chemical Engineering Science 58(3-6): 633-642.

[33] Koop, J. and O. Deutschmann (2009). "Detailed surface reaction mechanism for Pt-

catalyzed abatement of automotive exhaust gases." Applied Catalysis B-Environmental 91(1-2): 47-58.

[34] Silveston, P. L. (1998). "Composition modulation of catalytic reactors." Amsterdam, Gordon and Breach Science.

[35] Kryl, D., P. Koci, et al. (2005). "Catalytic converters for automobile diesel engines with adsorption of hydrocarbons on zeolites." Industrial & Engineering Chemistry Research 44(25): 9524-9534.

[36] Masel, R. I. (2001). "Chemical kinetics and catalysis." New York, Wiley.

[37] Bank, R. and C. Douglas (1993). "Sparse matrix multiplication package (SMMP)." Advances in Computational Mathematics 1(1): 127-137.

[38] D'Azevedo, E. F., M. R. Fahey, et al. (2005). "Vectorized sparse matrix multiply for compressed row storage format." Computational Science - Iccs 2005, Pt 1, Proceedings. V. S. Sunderam, G. D. VanAlbada, P. M. A. Sloot and J. J. Dongarra. 3514: 99-106.

[39] Eisenstat, S. C., M. C. Gursky, et al. (1982). "YALE SPARSE-MATRIX PACKAGE .1. THE SYMMETRIC CODES." International Journal for Numerical Methods in Engineering 18(8): 1145-1151.

[40] Williams, P. T. and A. J. Baker (1996). "Incompressible computational fluid dynamics and the continuity constraint method for the three-dimensional Navier-Stokes equations." Numerical Heat Transfer Part B-Fundamentals 29(2): 137-273.

[41] Williams, P. T. and A. J. Baker (1997). "Numerical simulations of laminar flow over a 3D backward-facing step." International Journal for Numerical Methods in Fluids 24(11): 1159-1183.

Chapter 2 Modeling Reactions in Porous Catalysts

2.1 Dynamic Diffusion Reactor

The origin of the modeling problem of this research work comes from the previous experiments carried our in a dynamic diffusion reactor (DDR) which is also called a single-pellet reactor [1]. This reactor has been built to study the adsorption, desorption and surface reaction of carbon monoxide oxidation with oxygen over Pt/Al_2O_3 catalysts. Other types of solid catalytic reactions can also be studied in the DDR with packing of different catalysts and feeding of various gas reactants. The DDR and other apparatus connected to it is the role model for the numerical simulator. As the cross-section plot shows in figure 2.1, the DDR contains a porous catalyst pellet 4 mm in diameter and 0.64 mm in thickness. The pellet is also attached to a centerplane chamber with 0.044 cm^3 in volume on the left and to a bulk chamber with 0.056 cm^3 in volume on the right. The detail geometry and design of DDR is given by Cannestra et al. [1].

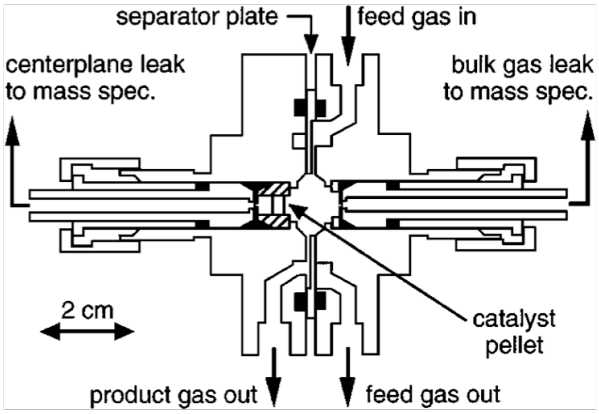

Figure 2.1 Cross-section plot of dynamic diffusion reactor (DDR) [1]

For the carbon monoxide oxidation experiment, the catalyst pellet is Pt/Al_2O_3. Its

structure is platinum particles impregnated in a porous aluminum oxide support, and it can be prepared from the method given by Racine and Herz (1992) [2]. The electron microscope image of the platinum particles in the catalyst is shown in Figure 2.2. To describe a catalyst like Pt/Al_2O_3 in the simulator, several parameters are required like the porosity, reactor weight, weight fraction of different catalyst, surface area occupied by a catalyst atom, fraction of catalytic sites on each catalyst and the dispersion of catalyst particle. Fraction of catalytic sites can represent multiple places where the surface reaction can possibly happen such as linear site, bridgebound site and triply coordinated site [3]. The dispersion is the percent of exposed catalyst atoms on the surface to the whole cluster of catalyst particle, and it usually ranges from 1% to 100%. The weight percent of Pt in the catalyst pellet of DDR, for example, is less than 2% and the surface area occupied by a Pt atom is 5.5×10^4 m^2mol^{-1} [4-6].

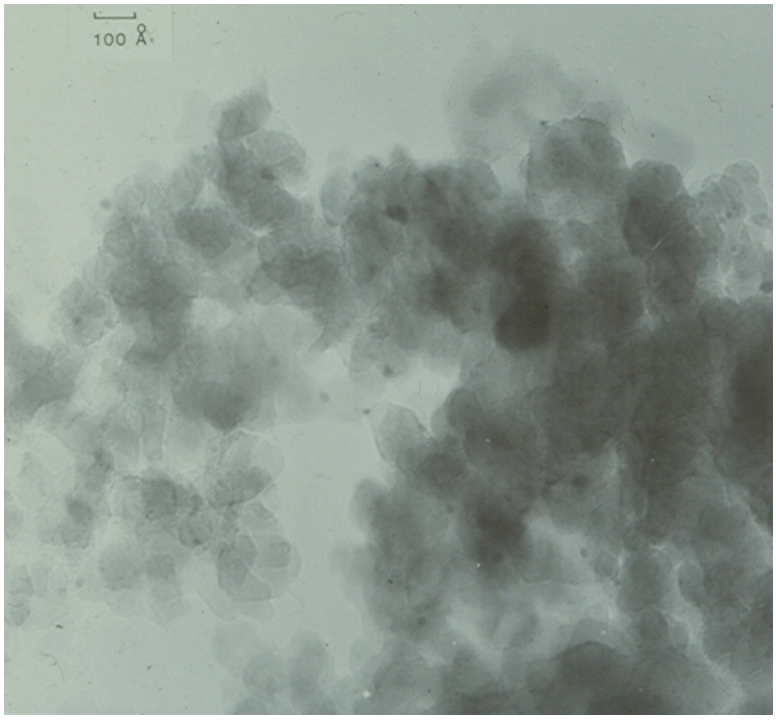

Figure 2.2 Electron microscope image of the platinum particle in Pt/Al_2O_3 catalyst

2.2 Kinetics on Catalysts Surface

Detailed surface kinetics also called microkinetics means the reaction mechanism is composed of separate adsorption, desorption and reaction processes. Major elementary steps and corresponding rate equations occurring over solid catalysts are summarized by Masel (2001) and are listed in Table 2.1 [7]. The Transcat numerical simulator is capable of computing any combination of these elementary equations. Other unlisted kinetics expressed by simple algebraic rate equation and stoichiometric balanced reactions are also included in the numerical algorithm. The general form of these rate equation computed by Transcat is

$$aA + bB \xrightarrow{k} product\ ,\ r = kC_A^{n_A} C_B^{n_B}$$

Table 2.1 Elementary reactions on metal surface

Type	Reaction	Rate Equation
Molecular Adsorption	$A_{(g)} + S_{(s)} \xrightarrow{k_{ads}} A*S_{(s)}$	$r_{ads} = k_{ads} C_A^g C_S^s$
Dissociative Adsorption	$B_{2(g)} + 2S_{(s)} \xrightarrow{k_{ads}} 2B*S_{(s)}$	$r_{ads} = k_{ads} C_{B_2}^g C_S^{s2}$
Molecular Desorption	$A*S_{(s)} \xrightarrow{k_{des}} A_{(g)} + S_{(s)}$	$r_{des} = k_{des} C_{A*S}^s$
Recombinative Desorption	$A*S_{(s)} + B*S_{(s)} \xrightarrow{k_{des}} AB_{(g)} + 2S_{(s)}$	$r_{des} = k_{des} C_{A*S}^s C_{B*S}^s$
Displacement Desorption	$A_{(g)} + B*S_{(s)} \xrightarrow{k_{des}} A*S_{(s)} + B_{(g)}$	$r_{des} = k_{des} C_A^g C_{B*S}^s$
Dissociation Reaction	$AB*S_{(s)} + S_{(s)} \xrightarrow{k_{rxn}} A*S_{(s)} + B*S_{(s)}$	$r_{rxn} = k_{rxn} C_{AB*S}^s C_S^s$
Association Reaction	$A*S_{(s)} + B*S_{(s)} \xrightarrow{k_{rxn}} AB*S_{(s)} + S_{(s)}$	$r_{rxn} = k_{rxn} C_{A*S}^s C_{B*S}^s$

The rate constant of adsorption, desorption and reaction on catalytic surface can be divided into three different categories according the kinetic theory of gas. The adsorption rate is related to the collision frequency of gas molecular onto solid surface [7,8]

$$z_{coll} = \frac{\rho_N \langle u \rangle}{4}$$

where z_{coll} is the collision frequency of adsorbate, the molecules that bind to the surface, in molecules/m^2, ρ_N is the number density in molecules/m^3 and $\langle u \rangle$ is the average speed of gas in m/s. From the kinetic theory of gases, the average speed of molecules in an ideal gas is

$$\langle u \rangle = \left(\frac{8RT}{\pi M} \right)^{0.5}$$

where R is the ideal gas constant, T is the temperature and M is the molecular weight. The rate of adsorption is given

$$r_{ads} = \left(\frac{z_{coll}}{N_v} \right) S_{ads}$$

where N_v is the Avogadro constant and S_{ads} is the sticking probability. The sticking probability is also defined as

$$S_{ads} = \frac{\text{Number of molecules that adsorb}}{\text{Number of molecules that impinge on a surface}}$$

so the rate constant of adsorption is

$$k_{ads} = \left(\frac{RT}{2\pi M} \right)^{0.5} \sigma S_{ads}$$

where σ is the area occupied by 1 mol of adsorbent, the substance that holds the adsorbate , in m^2/mol. The typical surface area of porous support of catalyst is around 10 m^2/mol to 100 m^2/mol and it can be as high as 1000 m^2/mol for porous media like activated carbon [7,9].

The desorption rate constant obeys an Arrhenius-like expression

$$k = \tau_{des}^{-1} e^{-E_a / RT}$$

where τ_{des} is time constant of desorption and E_a is the activation energy. For desorption process, the value of τ_{des} is around 10^{-12} sec, and E_a is equivalent to the enthalpy of adsorption which is typically 15-100 kcal/mol for chemisorption, a direct chemical bond between the adsorbate and the adsorbent, and 2-20 kcal/mol for physisorption, adsorbate is held on adsorbent by physical forces such as van der Waals forces. The rate constant for surface reaction also obeys the Arrhenius equation. To generalize an expression for these rate constants, the Arrhenius equation with an extra term of temperature derived from the transition-state theory is adopted in the numerical simulator [7,10]

$$k = AT^{N_T} e^{-E_a / RT}, \; 0 \le N_T \le 1$$

where A is temperature independent pre-exponential factor and N_T is the order of temperature dependent term. The summaries of these rate constants used in the numerical model are listed in Table 2.2.

Table 2.2 Summary of rate constant

Kinetic	Rate Constant	Pre-exponential Factor	Order of Temperature	Activation Energy
Adsorption	k_{ads}	$\left(\frac{8R}{2\pi M}\right)^{0.5}\sigma S_{ads}$	0.5	0
Desorption	k_{des}	τ_{des}^{-1}	0	E_a
Reaction	k_{rxn}	A	0	E_a

2.3 Gas Diffusion in Porous Layer

Diffusion is the process by which matter is transported from one part of system to another as a result of random molecular motions [11]. The Fick's law of diffusion in a binary gas mixture is

$$J_A = -D_{AB}\nabla C_A$$

where J_A is the molar diffusion flux of component A, D_{AB} is the binary diffusion coefficient, ∇ is the del operator and C_A is the concentration of A [12]. This is a result from the fluid-fluid intermolecular collision. The calculation of diffusion coefficient for rigid spheres of unequal mass and collision diameter is [13]

$$D_{AB} = \frac{2}{3}\left(\frac{\mathrm{K}_b^3}{\pi^3}\right)^{1/2}\left(\frac{1}{2m_A}+\frac{1}{2m_B}\right)^{1/2}\frac{T^{3/2}}{P\left(\frac{d_A+d_B}{2}\right)^2}$$

where K_b is the Boltzmann constant, T is the temperature, P is the pressure and m and

d are the mass and collision diameter of gas molecule A and B respectively.

When gas molecule was transported in porous channels, the fluid-wall collision is more dominant than the fluid-fluid collision. This type of diffusion is called the free-molecule flow or Knudsen flow, that is, the mean pore radius is much smaller than the mean free path [14]. From Chapman-Enskog Theory for ideal gas, the Knudsen diffusion coefficient D_K for single molecule in long straight pore with mean pore radius r_m is [9]

$$D_K = -\left(\frac{2}{3} r_m\right)\sqrt{\frac{8RT}{\pi M}}$$

The mean pore radius can be calculated by the single pore approximation of material filled with successive cylinders [9]

$$r_m = \frac{2\varepsilon}{\rho_c S_c}$$

where ε is the porosity, ρ_c is the density of porous catalyst and S_c is the surface area per unit mass of catalyst. The mean pore radius subject to the variation catalytic surface at different porosity is shown in Figure 2.3.

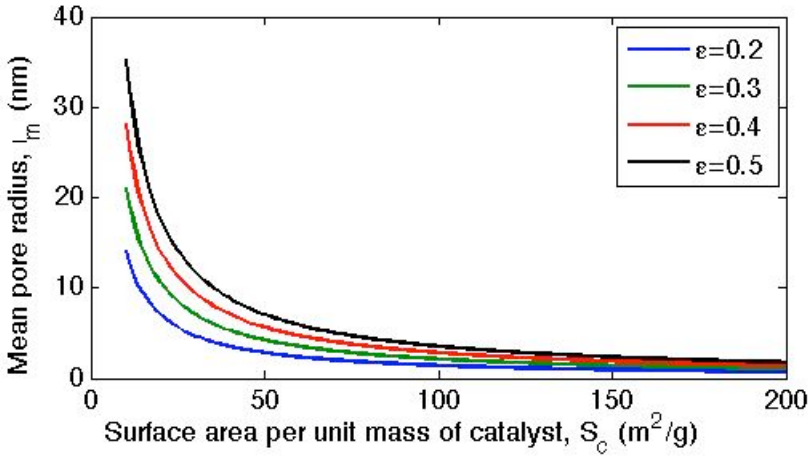

Figure 2.3 Size of mean pore radius to the variation of surface area per unit mass of catalyst. (ρ_c=2.839 g/cm^3)

The mean free path is the average distance traveled by a molecule between successive collisions and is given by [8]

$$l = \frac{1}{\sqrt{2}\pi d^2 \rho_N}$$

where l is the mean free path, d is the collision diameter of molecule treated as a hard sphere and ρ_N is the number density of gas molecule. The maximum operating pressure of Knudsen flow occurring in porous material can be obtained by assuming the mean pore radius is less than or equal to 1% of mean free path. Figure 2.4 shows the maximum operating pressure of CO in porous catalyst at different temperature, where the catalyst density is 2.839 g/cm^3 and porosity is 0.3. Previous experimental study under this criteria also show the gas diffusion in porous single pellet reactor is governed solely by Knudsen flow, even in the presence of a difference in total pressure across the reactor [1,5].

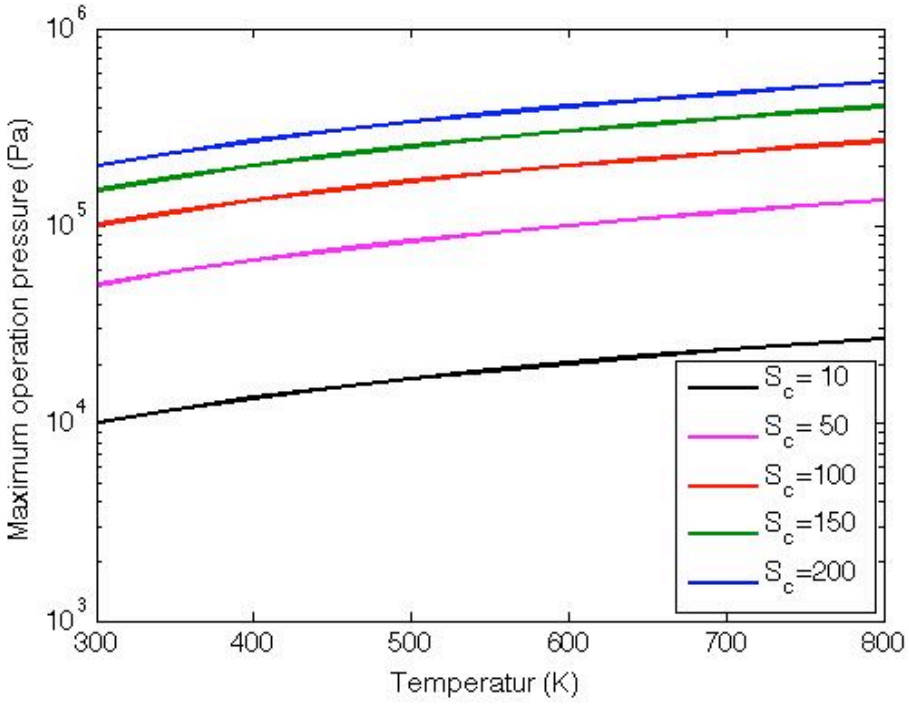

Figure 2.4 Maximum operation pressure of Knudsen flow of CO. (d_{CO} = 317 pm)

Due to the complexity geometry of porous material, the Knudsen diffusivity can further been corrected by the tortuosity factor. The effective diffusivity D_{eff} would have the form

$$D_{eff} = \frac{D_K}{\tau_{tor}}$$

where τ_{tor} is the tortuosity factor which is typically from 1.5 to 10 or higher [9].

2.4 Mass Balance in Porous Catalysts

Modeling a heterogeneous catalytic reactor requires mass balance equations in both gas and solid phases. Both a lumped and a one-dimensional distributed reactor model will be used in this work. The lumped model can compute fast to get a general idea of a surface kinetics. The one-dimensional distributed model can demonstrate the importance of diffusion resistance in porous catalysts. The lumped model can be seen as a constant volume continuous-stir-tank reactor (CSTR) over a large slab with catalyst on a non-porous surface, and the distributed one is like the dynamic diffusion reactor (DDR) with catalyst embedded in a porous layer.

The unsteady mass balance equation of a constant volume CSTR in the gas phase is [10,15]

$$\varepsilon V \frac{dC_i^g}{dt} = q\left(C_{b,i}^g - C_i^g\right) + A^s r_i$$

where the superscript g denotes the gas phase, ε is the porosity, V is the total volume, C_i^g is the concentration of component i, $C_{b,i}^g$ is the concentration of component i of the

mass transfer boundary layer, q is the volume flow rate, A^s is the total surface over of solid phase and r_i is the reaction rate per unit area of component i. The surface species is assumed to be stationary and the mass balance equation over the solid phase is

$$\frac{dC_i^s}{dt} = r_i$$

where the superscript denotes for the solid phase, C_i^s is the surface concentration of component i in moles per unit area.

The unsteady diffusion-reaction behavior is assumed to obey the Fick's law in a DDR. The unsteady one dimensional diffusion reaction equation in gas phase of the DDR is in the following form [6]

$$\varepsilon V \frac{\partial C_i^g}{\partial t} = V D_{eff,i} \frac{\partial^2 C_i^g}{\partial z^2} + A^s r_i$$

where $D_{eff,i}$ is the effective diffusion coefficient of component i. The equation of the DDR over the solid phase is the same as the CSTR model but the concentration is a function of both space and time instead of time only.

A good example to show the adsorption, desorption and reaction in the modeling of a catalytic reactor is the Langmuir-Hinshelwood-Hougen-Watson (LHHW) kinetic model of carbon monoxide oxidation on platinum surface. The LHHW mechanism on catalytic site S of platinum is

$$CO_{(g)} + S_{(s)} \underset{k_2}{\overset{k_1}{\rightleftarrows}} CO*S_{(s)}$$

$$O_{2(g)} + 2S_{(s)} \xrightarrow{k_3} 2O*S_{(s)}$$

$$CO*S_{(s)} + O*S_{(s)} \xrightarrow{k_4} CO_{2(g)} + 2S_{(s)}$$

The ordinary differential equations in CSTR model are

$$\text{Gas Phase}: \begin{cases} \varepsilon V \dfrac{dC_{CO}^{g}}{dt} = q\left(C_{b,CO}^{g} - C_{CO}^{g}\right) + A^{s}\left(-k_1 C_{CO}^{g} C_{S}^{s} + k_2 C_{CO}^{s}\right) \\ \varepsilon V \dfrac{dC_{O_2}^{g}}{dt} = q\left(C_{b,O_2}^{g} - C_{O_2}^{g}\right) + A^{s}\left(-k_3 C_{O_2}^{g} C_{S}^{s2}\right) \\ \varepsilon V \dfrac{dC_{CO_2}^{g}}{dt} = q\left(C_{b,CO_2}^{g} - C_{CO_2}^{g}\right) + A^{s}\left(k_4 C_{CO}^{s} C_{O}^{s}\right) \end{cases}$$

$$\text{Solid Phase}: \begin{cases} \dfrac{dC_{CO}^{s}}{dt} = k_1 C_{CO}^{g} C_{S}^{s} - k_2 C_{CO}^{s} - k_4 C_{CO}^{s} C_{O}^{s} \\ \dfrac{dC_{O}^{s}}{dt} = 2k_3 C_{O_2}^{g} C_{S}^{s2} - k_4 C_{CO}^{s} C_{O}^{s} \\ \dfrac{dC_{S}^{s}}{dt} = -k_1 C_{CO}^{g} C_{S}^{s} + k_2 C_{CO}^{s} - 2k_3 C_{O_2}^{g} C_{S}^{s2} + 2k_4 C_{CO}^{s} C_{O}^{s} \end{cases}$$

The partial differential equations in DDR model are

$$\text{Gas Phase}: \begin{cases} \varepsilon V \dfrac{\partial C_{CO}^{g}}{\partial t} = VD_{eff,CO} \dfrac{\partial^2 C_{CO}^{g}}{\partial z^2} + A^{s}\left(-k_1 C_{CO}^{g} C_{S}^{s} + k_2 C_{CO}^{s}\right) \\ \varepsilon V \dfrac{\partial C_{O_2}^{g}}{\partial t} = VD_{eff,O_2} \dfrac{\partial^2 C_{O_2}^{g}}{\partial z^2} + A^{s}\left(-k_3 C_{O_2}^{g} C_{S}^{s2}\right) \\ \varepsilon V \dfrac{\partial C_{CO_2}^{g}}{\partial t} = VD_{eff,CO_2} \dfrac{\partial^2 C_{CO_2}^{g}}{\partial z^2} + A^{s}\left(k_4 C_{CO}^{s} C_{O}^{s}\right) \end{cases}$$

$$\text{Solid Phase}: \begin{cases} \dfrac{\partial C_{CO}^{s}}{\partial t} = k_1 C_{CO}^{g} C_{S}^{s} - k_2 C_{CO}^{s} - k_4 C_{CO}^{s} C_{O}^{s} \\ \dfrac{\partial C_{O}^{s}}{\partial t} = 2k_3 C_{O_2}^{g} C_{S}^{s2} - k_4 C_{CO}^{s} C_{O}^{s} \\ \dfrac{\partial C_{S}^{s}}{\partial t} = -k_1 C_{CO}^{g} C_{S}^{s} + k_2 C_{CO}^{s} - 2k_3 C_{O_2}^{g} C_{S}^{s2} + 2k_4 C_{CO}^{s} C_{O}^{s} \end{cases}$$

Dimensional analysis can be applied to the governing equations to get the time constants and non-dimensional variable of these equations. The following dimensionless variables are used

$$\Psi_i = \frac{C_i^g}{C_{\max}^g},\ \Theta_i = \frac{C_i^s}{C_{tot}^s},\ \rho_i = \frac{r_i}{C_{tot}^s},\ Z = \frac{z}{L}$$

where $C_{\max}^g$ is the maximum gas concentration, C_{tot}^s is the total surface concentration and L is the length in the z axis. The gas equation of a CSTR then becomes

$$\frac{d\Psi_i}{dt} = \frac{1}{\tau_l}\left(\Psi_{b,i} - \Psi_i\right) + \alpha\rho_i,\ \tau_l = \frac{V^g}{q},\ \alpha = \frac{A^s C_{tot}^s}{\varepsilon V C_{\max}^g}$$

where τ_l is the space time of the CSTR and α is the surface-to-gas capacity ratio. The gas equation in the distributed model is

$$\frac{\partial\Psi_i}{\partial t} = \frac{1}{\tau_d}\frac{\partial^2\Psi_i}{\partial Z^2} + \alpha\rho_i,\ \tau_d = \frac{\varepsilon L^2}{D_{eff,i}},\ \alpha = \frac{A^s C_{tot}^s}{\varepsilon V C_{\max}^g}$$

where τ_d is the diffusion time constant.

To be able to compare the lumped and distributed model in numerical experiments, the boundary condition with external mass transfer resistance in the mass boundary layer should be used. The equation for the gas components at the boundary of the attached mixing chamber on the right is

$$\begin{cases} V^b \dfrac{dC_{b,i}^g}{dt} = q_{in,i}\left(C_{in,i}^g - C_{b,i}^g\right) + A^p\left(-D_{eff,i}\dfrac{\partial C_i^g}{\partial Z}\bigg|_{Z=1}\right) \\ D_{eff,i}\dfrac{\partial C_i^g}{\partial Z}\bigg|_{Z=0\ or\ 1} = k_{ext}\left(C_{b,i}^g - C_i^g\right) \end{cases}$$

where V^b is the volume of the mixing chamber, $q_{in,i}$ is the volume flow rate of feed gas, $C_{in,i}^g$ is the concentration of feed gas, A^p is the cross-sectional area on the pellet face and k_{ext} is the external mass transfer coefficient. The normalized form of this equation is

$$\frac{d\Psi_{b,i}}{dt} = \frac{1}{\tau_{in,i}}\left(\Psi_{in,i} - \Psi_{b,i}\right) + \frac{1}{\tau_{b,i}}\left(\Psi_{b,i} - \Psi_{i}\right), \text{ where } \tau_{in,i} = \frac{V^b}{q_{in,i}} \text{ and } \tau_{b,i} = \frac{V^b}{A^p k_{ext}}$$

There are other important dimensionless variables; Thiele modulus and effectiveness factor should be included in this discussion. The Thiele modulus is defined as [10,15]

$$\phi_{ij} = L\sqrt{\frac{k_j}{D_{eff,i}}}$$

where ϕ_{ij} is the Thiele modulus of gas component i corresponded to rate constant j. The Thiele modulus is a useful parameter for predicting reactor behavior with diffusion resistance in a porous layer. Due to operation under transient conditions, multiple gas components and complex surface kinetics, all the Thiele moduluses with various rate constant and diffusion coefficients are required to generalize the reactor performance subject to diffusion resistance.

Another parameter is the effectiveness factor, which represents the effect on the reaction rate of diffusion resistance in the pores. The effectiveness factor η of a steady feed is defined as [10,15]

$$\eta = \frac{r_j \text{ with diffusion}}{r_j \text{ without diffusion}}$$

The effectiveness factor can be calculated from the reaction rate of both lumped and distributed model under the same operating condition. Under transient operating condition, the space and time averaged concept can be used for the dynamic effectiveness factor [17]

$$\bar{\eta} = \frac{\frac{1}{L}\int_{t}^{t+\mathrm{T}}\int_{0}^{L} r_j^{DDR}\, dz\, d\tau}{\int_{t}^{t+\mathrm{T}} r_j^{CSTR}\, d\tau}$$

where T is the period operation time and τ is a dummy index for time integration.

Reference:

[1] Cannestra, A. F., L. C. Nett, et al. (1997). "Measurement of gas composition at the center of a porous pellet during adsorption and catalytic reaction under dynamic conditions." Journal of Catalysis 172(2): 346-354.

[2] Racine, B. N. and R. K. Herz (1992). "Modeling Dynamic CO Oxidation over Pr/Al2O3 - Effects of Intrapellet Diffusion and Site Heterogeneity." Journal of Catalysis 137(1): 158-178.

[3] Masel, R. I. (1996). Principles of adsorption and reaction on solid surfaces. New York, Wiley.

[4] Racine, B. N., M. J. Sally, et al. (1991). "Dynamic CO Oxidation Over Pt/Al2O3." Journal of Catalysis 127(1): 307-331.

[5] Guinn, K. V. and R. K. Herz (1992). "Thermal-desorption in the Pressure Gap and at High-Pressure Using a Micro Knudsen Effusion Cell." Catalysis Letters 14(3-4): 251-261.

[6] Nett-Carrington, L. C. and R. K. Herz (2002). "Spatiotemporal patterns within a porous catalyst: dynamic carbon monoxide oxidation in a single-pellet reactor." Chemical Engineering Science 57(8): 1459-1474.

[7] Masel, R. I. (2001). Chemical kinetics and catalysis. New York, Wiley.

[8] McQuarrie, D. A. and J. D. Simon (1997). Physical chemistry: a molecular approach. Sausalito, Calif., University Science Books.

[9] Froment, G. F. and K. B. Bischoff (1990). Chemical reactor analysis and design. New York, Wiley.

[10] Fogler, H. S. (2006). Elements of chemical reaction engineering. Upper Saddle River, NJ, Prentice Hall PTR.

[11] Crank, J. (1979). The mathematics of diffusion. Oxford, [Eng], Oxford University Press.

[12] Bird, R. B., E. N. Lightfoot, et al. (1960). Transport phenomena. New York, Wiley.

[13] Chapman, S. and T. G. Cowling (1970). "The mathematical theory of non-uniform gases; an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases." [Cambridge, Eng.], Cambridge University Press.

[14] Mason, E. A. and A. P. Malinauskas (1983). Gas transport in porous media : the dusty-gas model. Amsterdam; New York, Elsevier Scientific Pub. Co.

[16] Levenspiel, O. (1999). Chemical reaction engineering. New York, Wiley: 1 online resource (xvi, 668 p.).

[17] Alvarez-Ramirez, J., J. A. Ochoa-Tapia, et al. (2005). "Dynamic effectiveness factor for catalyst particles." Journal of Physical Chemistry B 109(21): 11058-11064.

Chapter 3 Element Conservation in Chemical Reactors

The concept of element conservation has been used in the computation of chemical stoichiometry, equilibrium and reaction dynamics. Theories and algorithms that integrate the element balance equations into the calculation of chemical reaction have been developed completely in closed systems [1,2]. In an open system, only the stationary chemical species have been calculated based on the element balance equations in the past. In this chapter, the inconsistency of total mass of chemical reaction calculations when not using the idea of element conservation will be revealed. As a result, a generalized algorithm of element constraint method (ECM) for both closed and open system is developed to eliminate the mass loss or gain due to the error propagation in numerical computation or transient problems. The algorithm of ECM is implemented in the numerical simulator to ensure the conservation of both total mass and elemental mass.

3.1 Element Conservation in Closed System

The continuity constraint method, which enforces the continuity differential constraint while solving the momentum equation, has been used in direct numerical simulation of computational fluid dynamics [3,4]. A similar algorithm of enforcing the element balance equations in solving mass balance equation of catalytic reaction is developed in this research work and is called the element constraint method (ECM). The theory of element conservation in a closed system is described in detail by Smith and Missen (1982) and Silbey and Alberty (2001) [1,2]. They use the operation of

stoichiometric matrix and formula matrix, also called conservation matrix, to apply the element conservation concept to a multi-component chemical reaction system. The formula matrix is constructed by the molar ratio between elements in all of the components involved in the chemical reaction. The maximum independent rows in the formula matrix give the number of element constraints, N_D, for this system.

$$N_D = rank(\mathbf{M})$$

The number of element constraints represents the maximum number of element balance equations can be used in numerical computation, so N_D is also equivalent to the number of dependent components. For a multi-component system with N chemical species, the number of independent components or independent mass balance equations N_I is

$$N_I = N - N_D$$

Considering the enzyme-substrate reaction with the following mechanism for example,

$$S + E \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C \xrightarrow{k_3} E + P$$

the formula matrix $\mathbf{M}$ of this reaction is [2]

$$\begin{array}{c} \begin{matrix} S & E & C & P \end{matrix} \\ \mathbf{M} = \left[\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right] \begin{array}{l} \text{Substrate} \\ \text{Enzyme} \end{array} \end{array}$$

The proportion between the entries in this formula matrix is also called the pseudo-element balance since the elements in substrate and enzyme are always in the same ratio [2]. There are total four components in this system and the rank of $\mathbf{M}$ is two, so the number of independent component N_I is two and the number of dependent component

N_D is also two. The mass balance equations of all four components in a closed system are

$$\begin{cases} \dfrac{dC_S}{dt} = -k_1 C_E C_S + k_2 C_C \\ \dfrac{dC_E}{dt} = -k_1 C_E C_S + k_2 C_C + k_3 C_C \\ \dfrac{dC_C}{dt} = k_1 C_E C_S - k_2 C_C - k_3 C_C \\ \dfrac{dC_P}{dt} = k_3 C_C \end{cases}$$

The variables in these ordinary differential equations (ODEs) can be written in the following matrix form,

$$\mathbf{C} = \begin{bmatrix} C_S & C_E & C_C & C_P \end{bmatrix}^T,\ \mathbf{R} = \begin{bmatrix} k_1 C_E C_S & k_2 C_C & k_3 C_C \end{bmatrix}^T,\ \mathbf{S} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

where **C** is the vector of components, **R** is the vector of reaction rates and **S** is the stoichiometric matrix. The ODEs can be simplified to the following matrix form

$$\frac{d\mathbf{C}}{dt} = \mathbf{SR}$$

The element balance equations of substrate and enzyme reaction are

$$\begin{cases} \dfrac{dC_S}{dt} + \dfrac{dC_C}{dt} + \dfrac{dC_P}{dt} = 0 \\ \dfrac{dC_E}{dt} + \dfrac{dC_C}{dt} = 0 \end{cases}$$

The element balance equations can be integrated directly with the following initial conditions

$$\mathbf{C}\left(t=t_0\right)=\left[C_{S0}\;\;C_{E0}\;\;C_{C0}\;\;C_{P0}\right]^T$$

and the result is

$$\begin{cases} C_S + C_C + C_P = C_{S0} + C_{C0} + C_{P0} \\ C_E + C_C = C_{E0} + C_{C0} \end{cases}$$

The result of these two element conservation equations comes from the reconstruction of mass balance equations by the means of the stoichiometric matrix and formula matrix. These two extra equations can replace two mass balance equations in the computation of the enzyme-substrate reaction in a closed system. A generalized method of using the element balance equations together with the mass balance equations for an open system will be discussed in section 3.5 of this chapter. There is no analytical solution available for the mass balance equations of the enzyme-substrate equation, so other mathematical approaches such as steady state approximation, partial equilibrium approximation or numerical methods could be chosen in the computation. In the following sections of this chapter, the conservation of total elements of this enzyme-substrate reaction solved by various mathematical methods is studied.

3.2 Steady State Approximation

The steady-state approximation, which is also called steady-state assumption or pseudo-steady-state hypothesis, is obtained when the derivative with respect to time of the concentration of reactive intermediates is small or equal to zero [5,6]. The assumption for the steady-state approximation of the exemplary enzyme-substrate reaction in the previous section is to set the derivative of component C close to zero

$$\frac{dC_C}{dt} = k_1 C_E C_S - k_2 C_C - k_3 C_C \approx 0$$

The mass balance equation of component C becomes

$$C_C = \frac{\left(C_{E0} + C_{C0}\right) C_S}{K_m + C_S} \text{ and } K_m = \frac{k_2 + k_3}{k_1}$$

The concentration of component C is expressed as a function of component S. To solve for S, substitute the algebraic equation of component C into the mass balance equation of S

$$\frac{dC_S}{dt} = -k_1 C_E C_S + k_2 C_C \approx -k_3 C_C = -\frac{k_3\left(C_{E0} + C_{C0}\right) C_S}{K_m + C_S}$$

This ODE can be integrated directly and solution is

$$K_m \ln\left(\frac{C_S}{C_{S0}}\right) + \left(C_S - C_{S0}\right) + k_3\left(C_{E0} + C_{C0}\right)\left(t - t_0\right) = 0$$

The concentration of component S can be further solved by integration methods such as bisection, secant or Newton's method. For component E, the mass balance equation is approximately zero.

$$\frac{dC_E}{dt} = -k_1 C_E C_S + k_2 C_C + k_3 C_C \approx 0$$

Therefore, the only way to computed the concentration of component E is to use the element balance equation of enzyme

$$C_E = C_{E0} + C_{C0} - C_C$$

Based on the steady-state assumption, the change of the reaction intermediate is small, so the generation of product P is approximately the consumption of substrate S

$$\frac{dC_P}{dt} = k_3 C_C \approx -\frac{dC_S}{dt}$$

The integral of this equation is

$$C_P = C_{S0} + C_{P0} - C_S$$

This result conflicts with the element balance equation of substrate in which the concentration of intermediate C disappears in the equation. The algorithm of steady-state approximation for the enzyme-substrate reaction is programmed under Matlab® and the concentration profile and error of total elements of the steady-state approximation is shown in figure 3.1. The rate constant and initial conditions used in this simulation satisfy the criteria given by Segel (1988), and all other enzyme-substrate reactions in this chapter will be computed with the same parameters [7].

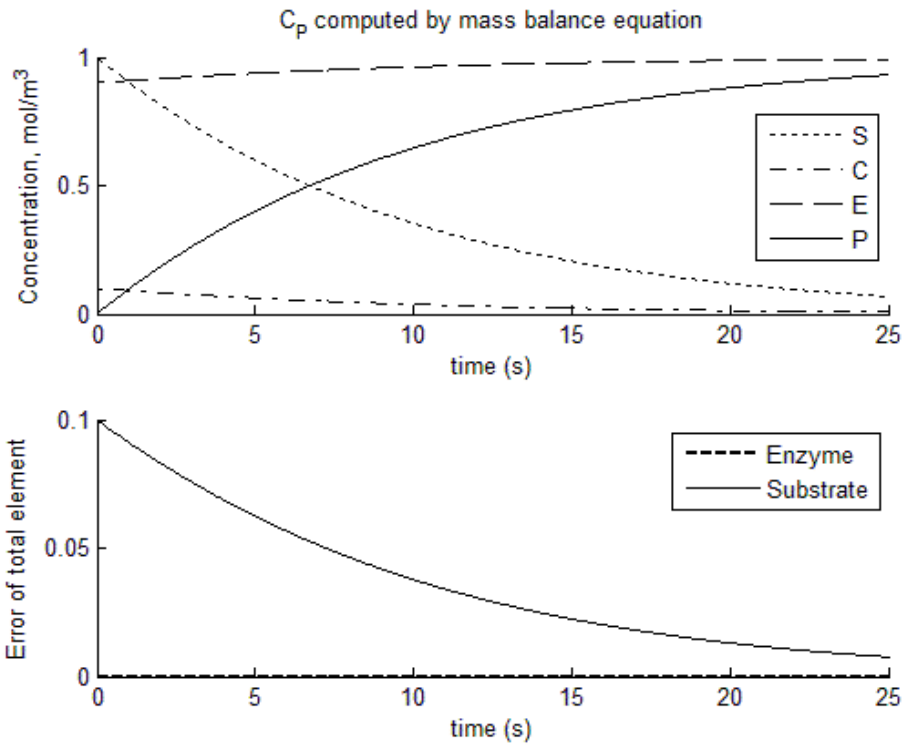

Figure 3.1 Steady-state approximation of enzyme-substrate reaction. The rate constants are $k_1 = 1$, $k_2 = 8$ and $k_3 = 1$.

The concentration of component P in figure 3.1 is computed by the mass balance equation and the dynamics of all four components shows reasonable results. However, there is 2% to 10% error in total elements of substrate, which means the element is not conserve under this approximation. If the concentration of component P is computed by the element balance equation of substrate rather than the result of the steady-state approximation, only the computer round-off error is detected in total elements and concentration profile and error are shown in figure 3.2. Unfortunately, replacing the equation for component P causes mass loss in the initial transient. As a result, the steady state approximation only shows good dynamics of concentration profiles but element conservation cannot be achieved.

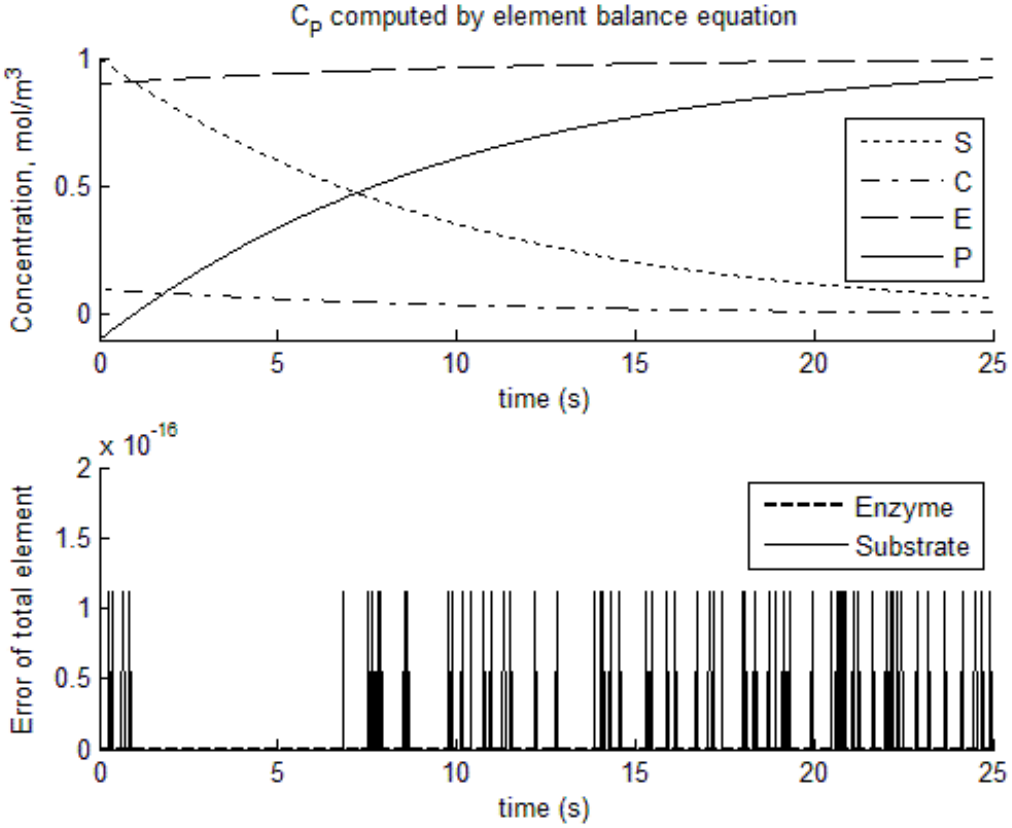

Figure 3.2 Steady state approximations with element conservation constraint

3.3 Partial Equilibrium Approximation

In the original mass balance equations of the enzyme-substrate reaction, the second step is often the rate determine step and both forward and reverse reaction rates of the first step are much larger than rate of the second step. We can apply the partial equilibrium assumption to this reaction by assuming the first step is close to equilibrium. The relation between the forward and reverse rates of the first step at equilibrium is

$$k_1 C_E C_S \approx k_2 C_C$$

By substituting the element balance equation of enzyme into this equation, the concentration of C and E are

$$C_C = \frac{\left(C_{E0} + C_{C0}\right) K_1 C_S}{1 + K_1 C_S}, \quad C_E = \frac{C_{E0} + C_{C0}}{1 + K_1 C_S}, \quad K_1 = \frac{k_1}{k_2}$$

The element balance of substrate becomes

$$\frac{dC_S}{dt} + \frac{d}{dt}\left[\frac{\left(C_{E0} + C_{C0}\right) K_1 C_S}{1 + K_1 C_S}\right] = -k_3\left[\frac{K_1\left(C_{E0} + C_{C0}\right) C_S}{1 + K_1 C_S}\right]$$

The solution to this ODE is

$$\ln\left(\frac{C_S}{C_{S0}}\right) + K\left(C_{E0} + C_{C0}\right)\ln\left(\frac{C_S}{C_{S0}}\frac{C_{S0} + 1/K}{C_S + 1/K}\right) + k_3 K\left(C_{E0} + C_{C0}\right)\left(t - t_0\right) = 0$$

The concentration of component P can also be express in terms of component S

$$C_P = C_{P0} + \left(C_{S0} - C_S\right) + \left(C_{E0} + C_{C0}\right) K_1\left(\frac{C_{S0}}{1 + K_1 C_{S0}} + \frac{C_S}{1 + K_1 C_S}\right)$$

The concentration of component S is calculated first by a root finding method, and other components are computed dependent on the variation of component S. The calculation result based on the partial equilibrium assumption is shown in figure 3.3, and

the concentration profile shows similar results to the steady-state approximation, but the error of total element in the rector is increased by more than 10% for the substrate. The total element is not change with respect to time since the equations are derived by means of the element balance equation and the initial condition of intermediate component C has been modified to $C_{C0} = K_1 C_{E0} C_{S0}$. Therefore, the partial equilibrium assumption breaks the conservation of total mass initially even though the error of total element is not affected much during a numerical computation. Due to the apparent error in total elements of the steady state assumption and partial equilibrium approximation these two methods will not be adopted in the numerical simulator.

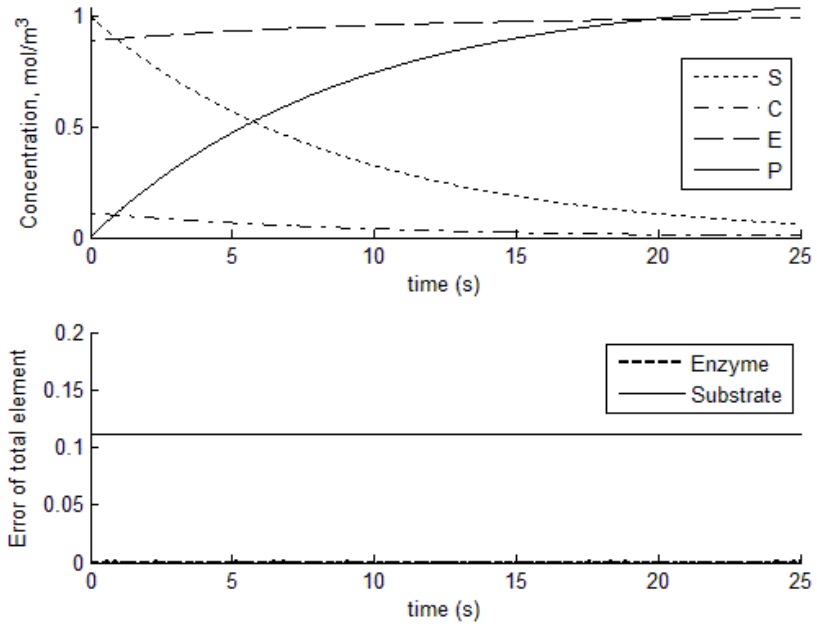

Figure 3.3 Partial equilibrium approximations

3.4 Numerical Approximation

The term numerical approximation in this section means the ODEs of the enzyme-substrate reaction are solved by numerical integration. Six different numerical methods

are used in this section, and these methods are explicit Euler (EE), second order Runge-Kutta (RK2), third order Runge-Kutta (RK3), implicit Euler (IE), second-order backward differential formula (BDF2), and third-order backward differential formula (BDF3) [8,9]. One way to compute the dynamics of this enzyme-substrate system is to solve all the mass balance equations directly by these numerical methods, the other is to integrate only the independent equations numerically and correct the error in total elements by the element conservation equations. The mass balance equations of component S and E are independent equations and C and P are dependent components which are computed by the algebraic equation of element balance. This two-step method is the element constraint method (ECM), and the generalized algorithm of ECM will be provided in the next section. These numerical algorithms are coded Matlab® and the result of RK3 and BDF3 with or without the ECM is shown in figure 3.4. The term ECM is added to the end of abbreviation of numerical method when it is used.

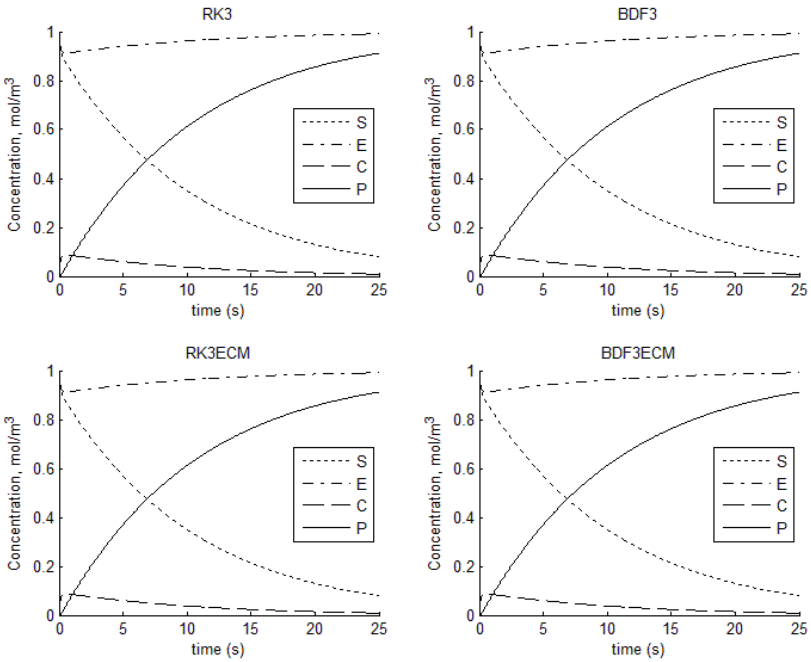

Figure 3.4 Numerical approximation of enzyme-substrate reaction

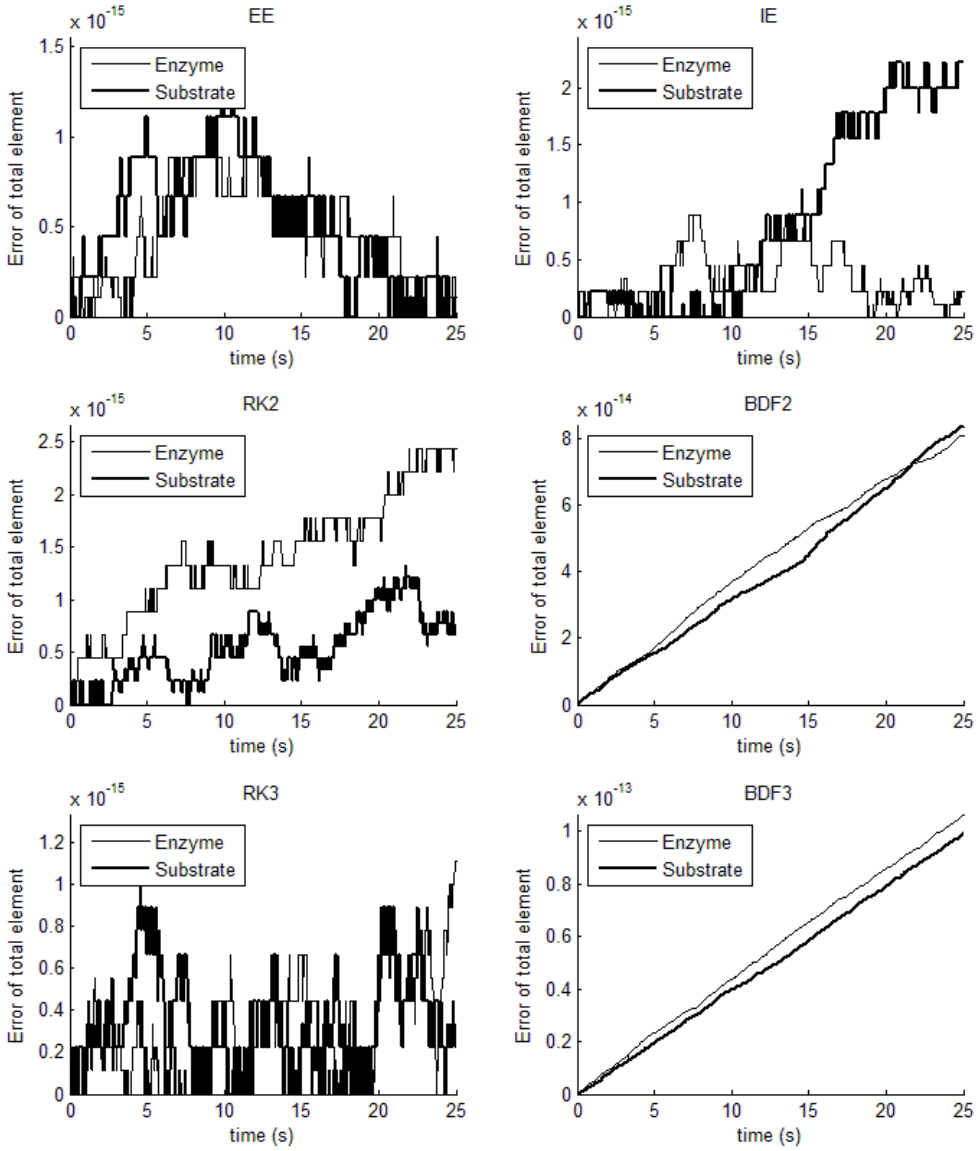

Figure 3.5 Error of elements of numerical approximation

The concentration of both RK3 and BDF3 methods shows good agreement to the steady-state and partial equilibrium approximations. No mass loss or gain is observed in the initial transient. The errors in total elements of these six methods are shown in figure 3.5. The error of explicit methods are small regardless the numerical method used, and there is no apparent trend of how the error is accumulated. However, the phenomenon of

error propagation is evident in the implicit methods and the high-order method gives more error than the low-order method under the same condition. The error propagation in both the BDF2 and BDF3 algorithm is almost linearly proportioned to the time step of integration, so when the problem is operated under transient conditions, the error might continue to reside in the system and could affect the accuracy of computation significantly.

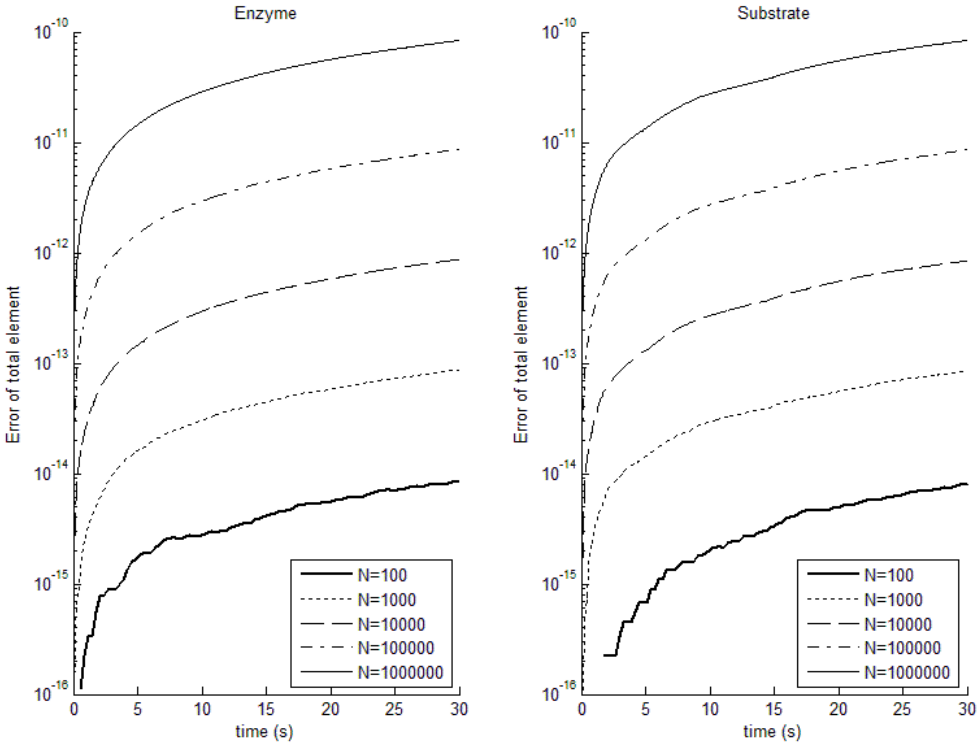

Figure 3.6 Error propagation at different number of steps

The phenomenon of error propagation is also studied at various sizes of time step. The same time span is divided into steps of various magnitude of order and the result of numerical integration is given in figure 3.6. The error of element in both enzyme and substrate changes with increasing the number of the steps within the same time span. When the same condition is solved numerically with the ECM algorithm, the error

propagation is eliminated and the computer round-off error is not even found in most of the cases. To prevent the error propagation in the simulator of porous catalyst, the ECM algorithm is added to Transcat. This gives the user an extra option for the numerical simulation of porous heterogeneous catalysts.

3.5 Element Constraint Method

The element constraint method (ECM) is an algorithm to coordinate the element balance equations into the mass balance equations in a numerical computation of a chemical reaction. The algorithm of ECM can be united into both explicit and implicit numerical methods to form either a one-step or a two-step method during calculation. The advantage of applying ECM to a numerical calculation is to ensure the conservation of elements no matter what the accuracy of a numerical method is in solving the nonlinear partial differential equation system.

The most common way to solve a multi-component reaction system numerically is to integrate all the nonlinear mass conservation equations directly, but this may cause error propagation of elements in transient problems as shown in previous sections. Methods to maintain the conservation of element or stoichiometry balance in a closed system have been developed and discussed in literature. The concept of element balance is originally used to obtain the concentration of chemical equilibrium and steady state in a closed system [1,2]. For transient problems, the existence of time-dependent stoichiometry of chemical reactions is first discussed by Laidler (1987) [10]. He concluded that the overall stoichiometrically balanced reaction is insufficient in modeling the behavior when the reaction is proceeding, so a detailed mechanism composed of

elementary steps is required to represent the kinetics. Ball (1998) shows the initial reactant concentration sets a limit to a total product formed under transient conditions [11]. Toby et al. (2000) derived the time-dependent stoichiometric relationship by summing and integrating rate equations and he also obtained the same result alternatively by using the method of element material balance [12,13]. Lee (2001) used the concept of element conservation and the extent of reaction to derive the relation between stoichiometry and chemical kinetics [14].

A more generalized method of using formula matrix and stoichiometric matrix in the computation of closed system to ensure the conservation of element is given by Smith and Missen (1982) [1]. For numerical computation of catalytic reaction in an open system, the concept of the element conservation has only been applied for absorbed species on the catalyst since the solid phase is not transported [2]. Using the concept of element conservation can eliminate error propagation in various numerical approximations as shown in previous section of this chapter. Therefore, a generalized element constraint method (ECM) is developed in this research work for the computation of chemical kinetics in both closed and open system.

The first step of ECM is to identify the number of independent and dependent chemical species in the system. The number of dependent component or mass balance equations in a chemical reaction equals to the rank of formula matrix. The relation between dependent, independent and total component is

$$N = N_I + N_D \text{ and } N_D = \text{rank}\left(\mathbf{M}\right)$$

where N is the number of total components, N_I is the number of independent components, N_D is the number of dependent components and $\mathbf{M}$ is the formula matrix.

Most mass balance equations in a chemical reactor can be written in the following form

$$\frac{\partial \mathbf{C}}{\partial t} = \mathbf{L} + \mathbf{SR}$$

, where $\mathbf{C}$ is the vector of concentration, $\mathbf{L}$ is the linear part, $\mathbf{S}$ is the stoichiometric matrix and $\mathbf{R}$ is the nonlinear reaction part. To relate the mass balance to the conservation of element, multiply the mass balance equation by the formula matrix

$$\mathbf{M}\left(\frac{\partial \mathbf{C}}{\partial t} - \mathbf{L}\right) = \mathbf{MSR}$$

The product of the formula matrix and stoichiometric matrix on the right hand side is zero, since the elements are only transported and reacted in the reactor but neither generated nor destroyed [1]. The element conservation in a chemical reactor becomes

$$\mathbf{M}\left(\frac{\partial \mathbf{C}}{\partial t} - \mathbf{L}\right) = \mathbf{0}$$

Then the independent and dependent component can be separated in this equation

$$\begin{bmatrix}\mathbf{M}_I & \mathbf{M}_D\end{bmatrix}\begin{bmatrix}\dfrac{\partial \mathbf{C}_I}{\partial t} - \mathbf{L}_I \\ \dfrac{\partial \mathbf{C}_D}{\partial t} - \mathbf{L}_D\end{bmatrix} = \mathbf{0}$$

, where the subscripts I and D denote the independent and dependent components respectively. The equation of dependent components can be expressed by the linear relation of independent components. The dependent equation is given as following

$$\frac{\partial \mathbf{C}_D}{\partial t} = \mathbf{L}_D - \mathbf{M}_D^{-1}\mathbf{M}_I\left(\frac{\partial \mathbf{C}_I}{\partial t} - \mathbf{L}_I\right)$$

The algorithm of ECM uses the combination of both the independent equations and dependent equations, equations of independent and dependent components respectively, to solve the partial differential equation system.

$$\begin{cases} \dfrac{\partial \mathbf{C}_I}{\partial t} = \mathbf{L}_I + \mathbf{S}_I\mathbf{R} \\ \dfrac{\partial \mathbf{C}_D}{\partial t} = \mathbf{L}_D - \mathbf{M}_D^{-1}\mathbf{M}_I\left(\dfrac{\partial \mathbf{C}_I}{\partial t} - \mathbf{L}_I\right) \end{cases}$$

The algorithm can be either a one-step or two-step algorithm depending on the numerical method used for the time integration. The one-step algorithm is to solve both the independent and dependent equations simultaneous in a numerical method such as the implicit Euler or backward differential formula. In a two-step algorithm, the independent equations are first integrated directly by a numerical method like explicit Euler or Runge-Kutta and then the dependent equations are solved linearly from the independent components derived in the first step. The criterion for the ECM algorithm is that the formula matrix of the dependent matrix $\mathbf{M}_D$ cannot be singular and the reaction part is the only nonlinear part in the differential equations.

The benefit of integrating the algorithm of ECM into a numerical computation is to guarantee the error induced by numerical approximation is minimized down to the computer round-off error. The computation cost of ECM compared to the numerical algorithm without ECM is around ±10% in an optimized code. Factors such as complexity of rate equation, number of time steps, sequence of chemical species and choice of the independent and dependent components have been inspected as to how they

affect the computing speed in a two-step ECM algorithm. For an implicit method, both the dimension and variable entries of Jacobian matrix in Newton's method are reduced, but the number of iterations may increase tremendously. As a result, the structure of the Jacobian matrix is the most important factor, in the effect of ECM on the speed of an implicit ECM algorithm. The ECM algorithm is added to Transcat as an option, since the efficiency could be different with various kinetics, numerical methods and operating conditions. The algorithm of element constraint method is an original concept from this research work and it has been tested in the numerical simulation of porous catalysts. More future research is required in finding the best strategy of integrating ECM into a numerical method to reduce the computation cost.

Reference:

[1] Smith, W. R. and R. W. Missen (1982). Chemical reaction equilibrium analysis: theory and algorithms. New York, Wiley.

[2] Silbey, R. J. and R. A. Alberty (2001). Physical chemistry. New York, Wiley.

[3] Williams, P. T. and A. J. Baker (1996). "Incompressible computational fluid dynamics and the continuity constraint method for the three-dimensional Navier-Stokes equations." Numerical Heat Transfer Part B-Fundamentals 29(2): 137-273.

[4] Williams, P. T. and A. J. Baker (1997). "Numerical simulations of laminar flow over a 3D backward-facing step." International Journal for Numerical Methods in Fluids 24(11): 1159-1183.

[5] Boudart, M. (1968). Kinetics of chemical processes. Englewood Cliffs, N.J.,, Prentice-Hall.

[6] Tamaru, K. (1978). Dynamic heterogeneous catalysis. London ; New York, Academic Press.

[7] Segel, L. A. (1988). "On the Validity of the Steady-state Assumption of Enzyme-kinetics." Bulletin of Mathematical Biology 50(6): 579-593.

[8] Press, W. H. (1996). Numerical recipes in fortran 90 : the art of parallel scientific computing. Cambridge [England] ; New York, Cambridge University Press.

[9] Hairer, E. and G. Wanner Solving ordinary differential equations II. Stiff and differential-algebraic problems. Heidelberg ; New York, Springer: 1 online resource (xv, 614 p.).

[10] Laidler, K. J. (1987). Chemical kinetics. New York, Harper & Row.

[11] Ball, D. W. (1998). "Kinetics of consecutive reactions: First reaction, first-order; second reaction, zeroth-order." Journal of Chemical Education 75(7): 917-919.

[12] Toby, S. (2000). "The relationship between stoichiometry and kinetics." Journal of Chemical Education 77(2): 188-190.

[13] Toby, S. and I. Tobias (2003). "What is the overall stoichiometry of a complex reaction?" Journal of Chemical Education 80(5): 520-523.

[14] Lee, J. Y. (2001). "The relationship between stoichiometry and kinetics revisited." Journal of Chemical Education 78(9): 1283-1284.

Chapter 4 Numerical Methods for Modeling Porous Catalysts

In previous chapters, the modeling equations for the porous catalysts and the element constraint method have been discussed. In this chapter, the numerical algorithm of Transcat, the simulator of heterogeneous catalysis in porous reactor, is developed. Both a lumped system and a one-dimensional model are built in Transcat. Transcat is designed to handle most surface kinetics, unlimited components and various boundary conditions. Multiple numerical methods are integrated into this simulator and each of them can be chosen by the user according to the condition and stiffness of the modeling problem. This simulator is also the first generalized computer program for porous catalysts using complete element constraint in an open system. More features are described in detail in the following sections.

4.1 Numerical Modeling for Porous Catalysts

To solve the unsteady diffusion-reaction equations in a porous catalyst, many finite difference methods can be used and they can be sorted into five major categories. These categories are explicit, implicit, semi-implicit, splitting step and other high order methods. To name a few, the explicit methods are explicit Euler (EE), Runge-Kutta (RK) and Adams-Bashforth (AB) method. The implicit methods are implicit Euler (IE), backward differential formula (BDF) and Adams-Moulton method (AM). An example of the semi-implicit method, which solves the linear part by an implicit method and the nonlinear part, is the combination of Crank-Nicholson method and third order Runge-Kutta method (CNRK3). A representative of the splitting methods is the alternating

direction implicit (ADI) method and other high order methods can be used in solve unsteady diffusion reaction equations are semi-spectral method, integrating factor method and exponential time differencing (ETD) method. Not all numerical mentioned here are implemented in Transcat, but the code will be well structured so any of these method can be added in the future when needed [1-7].

Both a lumped system ignoring diffusion effects and a one-dimensional model considering diffusion resistance are solved by these numerical methods. The element constraint method (ECM), which can reduce the error propagation in total elements and is also a possibility of improve the computing speed, are integrated into the algorithm of each numerical method. The numerical algorithm of ECM, which is determined by the discretization of the linear part of the mass balance equation, is provided in this section. An exemplary case of Langmuir-Hinshelwood mechanism of carbon monoxide oxidation in porous catalyst is served as a role model to illustration the algorithm.

The Langmuir-Hinshelwood mechanism of carbon monoxide oxidation on the catalytic site S of platinum is [8]

$$\begin{cases} CO_{(g)} + S_{(s)} \underset{k_2}{\overset{k_1}{\rightleftarrows}} CO*S_{(s)} \\ O_{2(g)} + 2S_{(s)} \xrightarrow{k_3} 2O*S_{(s)} \\ CO*S_{(s)} + O*S_{(s)} \xrightarrow{k_4} CO_{2(g)} + 2S_{(s)} \end{cases}$$

There are total six components involved in this surface kinetic. The formula matrix is

$$\mathbf{M} = \begin{array}{c} \begin{matrix} \Psi_{CO} & \Psi_{O_2} & \Psi_{CO_2} & \Theta_{CO} & \Theta_O & \Theta_s \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 2 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{matrix}\right] \end{array} \begin{matrix} \\ Carbon \\ Oxygen \\ S\ on\ Platinum \end{matrix}$$

and the rank of $\mathbf{M}$ is three, which mean there are three independent and three dependent components. If we choose the three gas species to be the independent components and all the others over the solid phase to be dependent, the formula matrix and vector of components become

$$\mathbf{C}=\begin{bmatrix}\mathbf{C}_I\\ \mathbf{C}_D\end{bmatrix},\ \mathbf{C}_I=\begin{bmatrix}\Psi_{CO}\\ \Psi_{O_2}\\ \Psi_{CO_2}\end{bmatrix},\ \mathbf{C}_D=\begin{bmatrix}\Theta_{CO}\\ \Theta_O\\ \Theta_s\end{bmatrix},\ \mathbf{M}_I=\begin{bmatrix}1&0&1\\ 1&2&2\\ 0&0&0\end{bmatrix},\ \mathbf{M}_D=\begin{bmatrix}1&0&0\\ 1&1&0\\ 1&1&1\end{bmatrix}$$

The singularity of the formula matrix of the dependent components can be confirmed by the calculation of the determinant. The determinant of matrix $\mathbf{M}_D$ is one, so there is a unique solution to this linear system of element conservation. From previous chapters, the lumped model is a constant volume continuous-stir-tank reactor (CSTR) over a slab of nonporous catalyst on the surface, and the distributed one is a porous pellet, which resembles the dynamic diffusion reactor (DDR). The two-phase mass balance equations with normalized concentration, dimensionless parameters and modified rate constant of dimension s^{-1} are

$$\text{CSTR}:\ \frac{d\mathbf{C}}{dt}=\begin{bmatrix}\dfrac{d\Psi_{CO}}{dt}=\tau_l^{-1}\left(\Psi_{b,CO}-\Psi_{CO}\right)+\alpha\left(-k_1\Psi_{CO}\Theta_S+k_2\Theta_{CO}\right)\\ \dfrac{d\Psi_{O_2}}{dt}=\tau_l^{-1}\left(\Psi_{b,O_2}-\Psi_{O_2}\right)+\alpha\left(-k_3\Psi_{O_2}\Theta_S^2\right)\\ \dfrac{d\Psi_{CO_2}}{dt}=\tau_l^{-1}\left(\Psi_{b,CO_2}-\Psi_{CO_2}\right)+\alpha\left(k_4\Theta_{CO}\Theta_O\right)\\ \dfrac{d\Theta_{CO}}{dt}=k_1\Psi_{CO}\Theta_S-k_2\Theta_{CO}-k_4\Theta_{CO}\Theta_O\\ \dfrac{d\Theta_O}{dt}=2k_3\Psi_{O_2}\Theta_S^2-k_4\Theta_{CO}\Theta_O\\ \dfrac{d\Theta_s}{dt}=-k_1\Psi_{CO}\Theta_S+k_2\Theta_{CO}-2k_3\Psi_{O_2}\Theta_S^2+2k_4\Theta_{CO}\Theta_O\end{bmatrix}$$

$$\text{DDR}: \ \frac{\partial \mathbf{C}}{\partial t} = \begin{bmatrix} \dfrac{\partial \Psi_{CO}}{\partial t} = \dfrac{1}{\tau_d}\dfrac{\partial^2 \Psi_{CO}}{\partial Z^2} + \alpha\left(-k_1 \Psi_{CO}\Theta_S + k_2 \Theta_{CO}\right) \\ \dfrac{\partial \Psi_{O_2}}{\partial t} = \dfrac{1}{\tau_d}\dfrac{\partial^2 \Psi_{O_2}}{\partial Z^2} + \alpha\left(-k_3 \Psi_{O_2}\Theta_S^2\right) \\ \dfrac{d\Psi_{CO_2}}{dt} = \dfrac{1}{\tau_d}\dfrac{\partial^2 \Psi_{CO_2}}{\partial Z^2} + \alpha\left(k_4 \Theta_{CO}\Theta_O\right) \\ \dfrac{d\Theta_{CO}}{dt} = k_1 \Psi_{CO}\Theta_S - k_2 \Theta_{CO} - k_4 \Theta_{CO}\Theta_O \\ \dfrac{d\Theta_O}{dt} = 2k_3 \Psi_{O_2}\Theta_S^2 - k_4 \Theta_{CO}\Theta_O \\ \dfrac{d\Theta_S}{dt} = -k_1 \Psi_{CO}\Theta_S + k_2 \Theta_{CO} - 2k_3 \Psi_{O_2}\Theta_S^2 + 2k_4 \Theta_{CO}\Theta_O \end{bmatrix}$$

The rate equation is assumed to be elementary according to the mechanism and the matrix form of these differential equation systems is

$$\text{CSTR}: \ \frac{d\mathbf{C}}{dt} = \frac{d}{dt}\begin{bmatrix} \Psi \\ \Theta \end{bmatrix} = \begin{bmatrix} \dfrac{1}{\tau_l}\left(\Psi_b - \Psi\right) + \alpha \mathbf{S}_\Psi \mathbf{R} \\ \mathbf{S}_\Theta \mathbf{R} \end{bmatrix}$$

$$\text{DDR}: \ \frac{\partial \mathbf{C}}{\partial t} = \frac{\partial}{\partial t}\begin{bmatrix} \Psi \\ \Theta \end{bmatrix} = \begin{bmatrix} \dfrac{1}{\tau_d}\dfrac{\partial^2 \Psi}{\partial Z^2} + \alpha \mathbf{S}_\Psi \mathbf{R} \\ \mathbf{S}_\Theta \mathbf{R} \end{bmatrix}$$

The stoichiometric matrix is divided into two parts for either gas or solid components due to the parameter of the surface-to-gas capacity ratio, so the matrix of reaction rate **R** does not have to be changed. These matrices are

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_\Psi \\ \mathbf{S}_\Theta \end{bmatrix}, \ \mathbf{S}_\Psi = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \ \mathbf{S}_\Theta = \begin{bmatrix} 1 & -1 & 0 & -1 \\ 0 & 0 & 2 & -1 \\ -1 & 1 & -2 & 2 \end{bmatrix}, \ \mathbf{R} = \begin{bmatrix} k_1 \Psi_{CO}\Theta_S \\ k_2 \Theta_{CO} \\ k_3 \Psi_{O_2}\Theta_S^2 \\ k_4 \Theta_{CO}\Theta_O \end{bmatrix}$$

If the element constraint method is used, equations for independent and dependent components are therefore

$$\text{CSTR}: \begin{cases} \dfrac{d\mathbf{C}_I}{dt} = \dfrac{d}{dt}\begin{bmatrix} \Psi_{CO} \\ \Psi_{O_2} \\ \Psi_{CO_2} \end{bmatrix} = \dfrac{1}{\tau_l}\begin{bmatrix} \Psi_{b,CO} - \Psi_{CO} \\ \Psi_{b,O_2} - \Psi_{O_2} \\ \Psi_{b,CO_2} - \Psi_{CO_2} \end{bmatrix} + \alpha\begin{bmatrix} -k_1\Psi_{CO}\Theta_S + k_2\Theta_{CO} \\ -k_3\Psi_{O_2}\Theta_S^2 \\ k_4\Theta_{CO}\Theta_O \end{bmatrix} \\ \dfrac{d\mathbf{C}_D}{dt} = \dfrac{d}{dt}\begin{bmatrix} \Theta_{CO} \\ \Theta_O \\ \Theta_S \end{bmatrix} = -\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 1 & 0 & 1 \\ 1 & 2 & 2 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} \dfrac{d\Psi_{CO}}{dt} - \dfrac{1}{\tau_l}\left(\Psi_{in,CO} - \Psi_{CO}\right) \\ \dfrac{d\Psi_{O_2}}{dt} - \dfrac{1}{\tau_l}\left(\Psi_{in,O_2} - \Psi_{O_2}\right) \\ \dfrac{d\Psi_{CO_2}}{dt} - \dfrac{1}{\tau_l}\left(\Psi_{in,CO_2} - \Psi_{CO_2}\right) \end{bmatrix} \end{cases}$$

$$\text{DDR}: \begin{cases} \dfrac{\partial\mathbf{C}_I}{\partial t} = \dfrac{\partial}{\partial t}\begin{bmatrix} \Psi_{CO} \\ \Psi_{O_2} \\ \Psi_{CO_2} \end{bmatrix} = \dfrac{1}{\tau_d}\dfrac{\partial^2}{\partial z^2}\begin{bmatrix} \Psi_{CO} \\ \Psi_{O_2} \\ \Psi_{CO_2} \end{bmatrix} + \alpha\begin{bmatrix} -k_1\Psi_{CO}\Theta_S + k_2\Theta_{CO} \\ -k_3\Psi_{O_2}\Theta_S^2 \\ k_4\Theta_{CO}\Theta_O \end{bmatrix} \\ \dfrac{\partial\mathbf{C}_D}{\partial t} = \dfrac{\partial}{\partial t}\begin{bmatrix} \Theta_{CO} \\ \Theta_O \\ \Theta_S \end{bmatrix} = -\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 1 & 0 & 1 \\ 1 & 2 & 2 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} \dfrac{\partial\Psi_{CO}}{\partial t} - \dfrac{1}{\tau_d}\dfrac{\partial^2\Psi_{CO}}{\partial Z^2} \\ \dfrac{\partial\Psi_{O_2}}{\partial t} - \dfrac{1}{\tau_d}\dfrac{\partial^2\Psi_{O_2}}{\partial Z^2} \\ \dfrac{\partial\Psi_{CO_2}}{\partial t} - \dfrac{1}{\tau_d}\dfrac{\partial^2\Psi_{CO_2}}{\partial Z^2} \end{bmatrix} \end{cases}$$

The matrix form of the these equations are

$$\text{CSTR}: \quad \frac{d\mathbf{C}}{dt} = \frac{d}{dt}\begin{bmatrix} \mathbf{C}_I \\ \mathbf{C}_D \end{bmatrix} = \begin{bmatrix} \dfrac{1}{\tau_l}\left(\mathbf{C}_{I,in} - \mathbf{C}_I\right) + \alpha\mathbf{S}_I\mathbf{R} \\ -\mathbf{M}_D^{-1}\mathbf{M}_I\left[\dfrac{d\mathbf{C}_I}{dt} - \dfrac{1}{\tau_l}\left(\mathbf{C}_{I,in} - \mathbf{C}_I\right)\right] \end{bmatrix}$$

$$\text{DDR}: \quad \frac{\partial \mathbf{C}}{\partial t} = \frac{\partial}{\partial t}\begin{bmatrix}\mathbf{C}_I \\ \mathbf{C}_D\end{bmatrix} = \begin{bmatrix}\dfrac{1}{\tau_d}\dfrac{\partial^2 \mathbf{C}_I}{\partial Z^2} + \alpha \mathbf{S}_I \mathbf{R} \\ -\mathbf{M}_D^{-1}\mathbf{M}_I\left(\dfrac{\partial \mathbf{C}_I}{\partial t} - \dfrac{1}{\tau_d}\dfrac{\partial^2 \mathbf{C}_I}{\partial Z^2}\right)\end{bmatrix}$$

In the following sections of this chapter, the detail of solving these equations numerically will be introduced, including boundary conditions, grid definition of distributed model, algorithm of computing reaction rate and the numerical methods built into Transcat.

4.2 Efficient Computation of Reaction Rate

The numerical simulator is design to handle most surface kinetics and can also be customized by the user, so an efficient and generalized algorithm is required to improve the speed of computing the reaction rate. The reaction rate equation can be sorted into three categories, elementary rate, simple algebraic rate and others. Take the oxygen adsorption on a catalytic site *S* for example. The possible surface kinetics and corresponding rate equations are shown in table 4.1.

Table 4.1 Three type of oxygen adsorption kinetics and rate equation on metal [8]

Type	Surface Kinetics	Rate Equation
Elementary Rate (Two-step adsorption)	$\begin{cases} O_{2(g)} + S_{(s)} \xrightarrow{k_1^e} O_2 * S_{(s)} \\ O_2 * S_{(s)} + S_{(s)} \xrightarrow{k_2^e} 2O * S_{(s)} \end{cases}$	$r_{1ads,O_2}^e = k_1^e \Psi_{O_2} \Theta_S$ $r_{2ads,O_2}^e = k_2^e \Theta_{O_2 * S} \Theta_S$
Algebraic Rate (One-step adsorption)	$O_{2(g)} + 2S_{(s)} \xrightarrow{k_1^a} 2O * S_{(s)}$	$r_{1ads,O_2}^a = k_1^a \Psi_{O_2} \Theta_S^2$

Table 4.1 Continued

Other (Inhibition by CO)	$O_{2(g)} + 2S_{(s)} \xrightarrow{k_1^o} 2O*S_{(s)}$	$r^o_{ads,O_2} = k_1^o \Psi_{O_2} \left(1 - 2\Theta_O - \kappa\Theta_{CO}\right)^2$, where $\kappa = \left(1 - 2\Theta_O\right) / \left(1 - \Theta_O\right)$

To calculate these reaction rates, the elementary rate and algebraic rate can be stored by the Yale sparse matrix like format given by Eisenstat et al. (1982), which can be computed efficiently by the parallel algorithm from Bank and Douglas (2001) and D'Azevedo et al (2005) [9-11]. Other types of rate equations need to be hand coded by the user, and the space for programming is reserved in advance in the Fortran 90 subroutine. The Yale sparse matrix format for the Langmuir-Hinshelwood kinetics of carbon monoxide oxidation is listed in table 4.2. The stoichiometric matrices can also be stored in this type of format and the original and the Yale's format are listed in table 4.3

Table 4.2 Yale sparse matrix storage format of reaction rate parameters

Vector	Elementary rate	Algebraic rate with simple order
Rate Constant Array	$\mathbf{R}_C = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 \end{bmatrix}$	$\mathbf{R}_C = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 \end{bmatrix}$
Integer Array	$\mathbf{R}_I^E = \begin{bmatrix} 1 & 3 & 4 & 7 & 9 \end{bmatrix}$	$\mathbf{R}_I^A = \begin{bmatrix} 1 & 3 & 4 & 6 & 8 \end{bmatrix}$
Location Array	$\mathbf{R}_L^E = \begin{bmatrix} 1 & 6 & 4 & 2 & 6 & 6 & 4 & 5 \end{bmatrix}$	$\mathbf{R}_L^A = \begin{bmatrix} 1 & 6 & 4 & 2 & 6 & 4 & 5 \end{bmatrix}$
Power Array	Neglected (all orders are one)	$\mathbf{R}_P^A = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 1 & 1 \end{bmatrix}$

Table 4.3 Yale sparse matrix storage format of stoichiometric matrix

Matrix	Original Format	Yale's Sparse Format
$\mathbf{S}$	$\begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 \\ 0 & 0 & 2 & -1 \\ -1 & 1 & -2 & 2 \end{bmatrix}$	Integer Array: $\mathbf{S}^{YI} = \begin{bmatrix} 1 & 3 & 4 & 5 & 8 & 10 & 14 \end{bmatrix}$ Location Array: $\mathbf{S}^{YL} = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 2 & 4 & 3 & 4 & 1 & 2 & 3 & 4 \end{bmatrix}$ Yale's Storage of $\mathbf{S}$: $\mathbf{S}^{Y} = \begin{bmatrix} -1 & 1 & -1 & 1 & 1 & -1 & -1 & 2 & -1 & -1 & 1 & -2 & 2 \end{bmatrix}$
$\mathbf{S}_I$ and $\mathbf{S}_\Psi$	$\begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Integer Array: $\mathbf{S}_I^{YI} = \begin{bmatrix} 1 & 3 & 4 & 5 \end{bmatrix}$ Location Array: $\mathbf{S}_I^{YL} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$ Yale's Storage of $\mathbf{S}$: $\mathbf{S}_I^{Y} = \begin{bmatrix} -1 & 1 & -1 & 1 \end{bmatrix}$
$\mathbf{S}_D$ and $\mathbf{S}_\Theta$	$\begin{bmatrix} 1 & -1 & 0 & -1 \\ 0 & 0 & 2 & -1 \\ -1 & 1 & -2 & 2 \end{bmatrix}$	Integer Array: $\mathbf{S}_D^{YI} = \begin{bmatrix} 1 & 4 & 6 & 10 \end{bmatrix}$ Location Array: $\mathbf{S}_D^{YL} = \begin{bmatrix} 1 & 2 & 4 & 3 & 4 & 1 & 2 & 3 & 4 \end{bmatrix}$ Yale's Storage of $\mathbf{S}$: $\mathbf{S}_D^{Y} = \begin{bmatrix} 1 & -1 & -1 & 2 & -1 & -1 & 1 & -2 & 2 \end{bmatrix}$

The Yale sparse matrix format only stores the nonzero entries of the stoichiometric matrix, so the unnecessary calculation of multiplication can be saved during the computing of the reaction part. When the ECM algorithm is used, only the stoichiometric matrix of the independent component $\mathbf{S}_I$ is needed. The storage and calculating of the stoichiometric matrix is reduced from thirteen entries to the minimum, four entries corresponding to four reactions rates. A Fortran 90 routine is developed to compare these algorithms of computing the nonlinear reaction part, and calculations of six different algorithms from 1 to 128 grid points after 100,000 iterations is shown in figure 4.1.

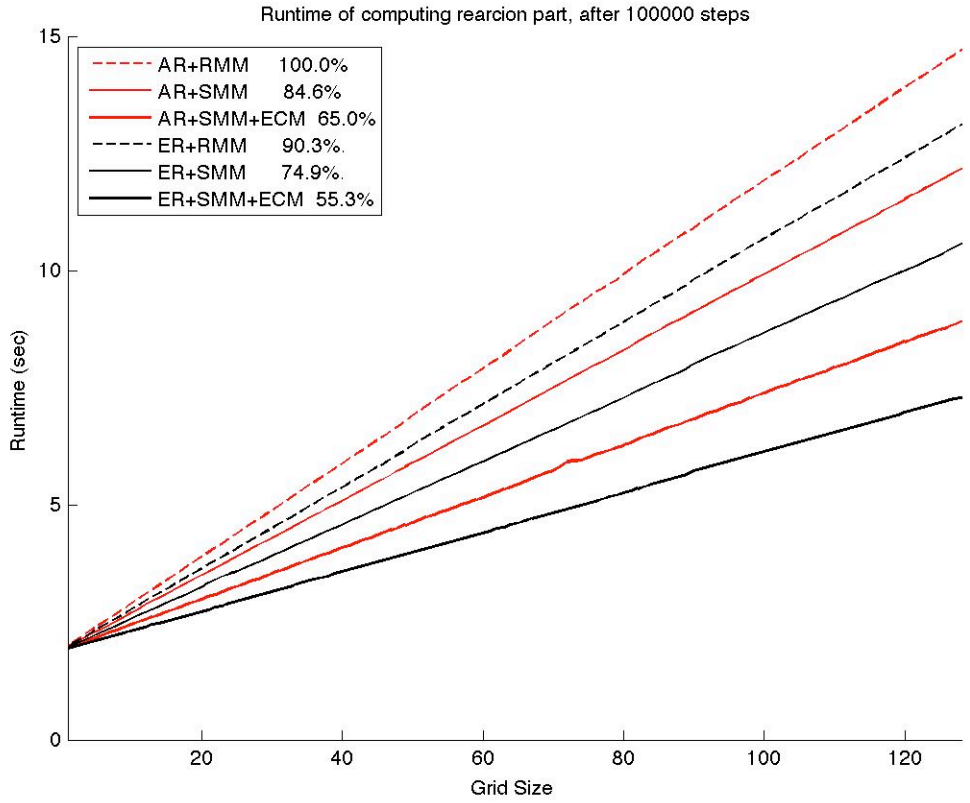

Figure 4.1 Runtime of regular and sparse matrix multiplication. (AR = algebraic rate, ER = elementary rate, RMM = regular matrix multiplication, SMM = sparse matrix multiplication, and ECM = element constraint method)

In every single step of computation shown in figure 4.1, the concentration is first random generated by a Fortran intrinsic routine, then the elementary or algebraic rate equation is calculated and finally the reaction part of each component is computed by different storage formats of stoichiometric matrix. The fastest algorithm is to compute the elementary rate (ER) by the sparse matrix multiplication (SMM) and element constraint method (ECM). This algorithm only takes 55.3% the cost of regular matrix multiplication (RMM) of an algebraic rate (AR). For both algebraic and elementary rate equation, the SSM algorithm can save around 15% of the computation cost and another 20% can be improved by the ECM algorithm, which explain the potential of ECM algorithm of

saving 10% in total runtime.

4.3 Grid Definition and Boundary Condition for Catalytic Reactor

Numerical simulations of heterogeneous catalytic reactions have been done by many researchers. The most common method that has been used for the mesh generation of partial differential equation system (PDEs) is the method of line (MOL) [12,13]. The MOL usually refers to a finite difference method, which discretizes the space derivatives in PDEs but leaves the time variable continuous [14]. The resulting ordinary differential equation system (ODEs) can be solved by various numerical methods and software packages [15,16]. If the mass balance equations for the dynamic diffusion reactor from the previous sections are discretize by MOL, they become

$$\frac{\partial \mathbf{C}}{\partial t} = \begin{bmatrix} \frac{1}{\tau_d}\left(\frac{C_{j+1} - 2C_j + C_{j+1}}{\Delta Z^2}\right) + \alpha \mathbf{S}_\Psi \mathbf{R} \\ \mathbf{S}_\Theta \mathbf{R} \end{bmatrix}$$

, where the subscript j denotes for evenly spaced grid points in the z-direction. Only the diffusion term of gas equation is modified by the center difference formula, but the solid equations are left intact.

When using MOL to discretize a variable in space in other scientific problems such as temperature distribution in heat transfer or velocity profile in viscous flow, the value of a variable usually represents for only the specific position on that grid. For the diffusion-reaction problem, the concentration actually occupies a volume in the reactor, so each grid or node on the mesh refers to a small volume surrounding this node, not just

the value right on that node. When using MOL to discrete the derivative of space in a mass balance equation, it generates unequal volume on the boundary. The boundary nodes represent half unit volume inside reactor and half unit volume outside. The grid and unit volume generated by MOL on a one-dimensional cylindrical coordinate is shown in the upper plot of figure 4.2.

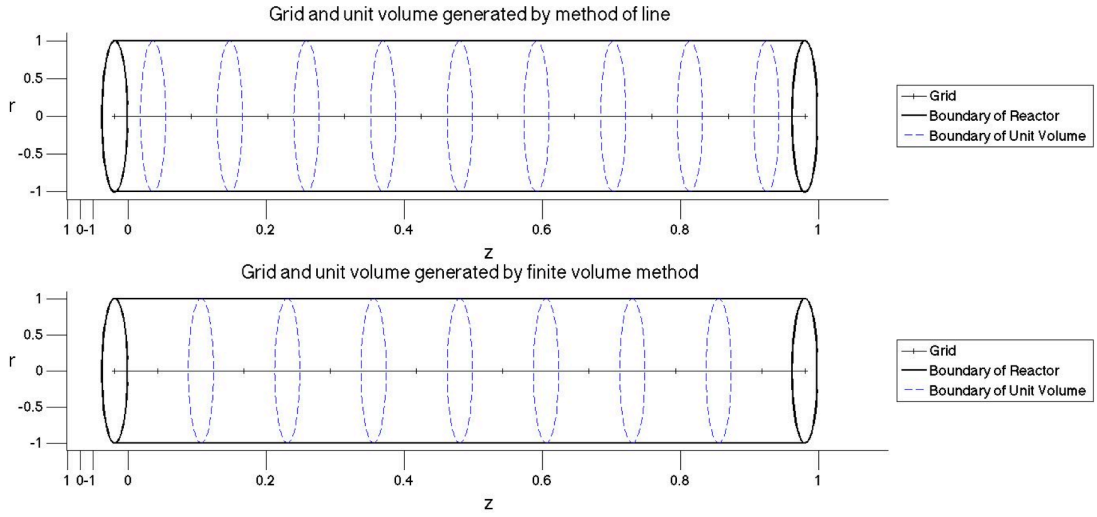

Figure 4.2 Grid definitions of MOL and FVM

Another method to generate the grid is to divide the whole reactor into equally spaced small volumes by positioning the first and last node half unit distance to the boundary. This idea of grid definition is similar to the finite volume method (FVM) and is shown in the lower plot of figure 4.2 [17]. The inner nodes on the grid occupy the same size of volume, so the finite difference equations derived from MOL are good to use for all inner nodes. For the two outer nodes with half distance to the boundary, the second derivative should be modified to a four-point differential formula in order to maintain the same second order accuracy as the inner nodes, when a computation is needed. The detailed consistency and accuracy analysis of this four-point formula is attached in Appendix A, and the four-point differential formulae for the two outer nodes are

$$\left.\frac{\partial^2 C}{\partial Z^2}\right|_{z=0} = \frac{\frac{16}{5}C_0 - 5C_1 + 2C_2 - \frac{1}{5}C_3}{\Delta Z^2},\ \varepsilon_{z=0} = \frac{\Delta z^2}{24}\left.\frac{\partial^4 C}{\partial Z^4}\right|_{z=0}$$

$$\left.\frac{\partial^2 C}{\partial Z^2}\right|_{z=1} = \frac{-\frac{1}{5}C_{Nz-2} + 2C_{Nz-1} - 5C_{Nz} + \frac{16}{5}C_{Nz+1}}{\Delta Z^2},\ \varepsilon_{z=1} = \frac{\Delta z^2}{24}\left.\frac{\partial^4 C}{\partial Z^4}\right|_{z=1}$$

, where the subscript from 2 and $Nz-1$ denote for the inner nodes of the reactor, the subscript 1 and Nz are for the two outer nodes, the subscript 0 and $Nz+1$ are for the boundary, and ε is the leading order error of the four-point formula. Since the first and last nodes on FVM grid are half a unit distance to the boundary, the derivative in a boundary condition must also be modified to maintain the consistency. The FVM grid fits better to the catalytic reactor so it is chosen in Transcat with modified discretization of the boundary conditions.

Three common types of boundary conditions, Dirichlet-type, von Neumann-type and Danckwerts'-type, are integrated in to Transcat to model the catalytic reactor [1]. The first one is the modulated concentration, which is the Dirichlet-type boundary condition. The gas concentration on the boundary is modulated as a function of time, and the reactor responds to the modulation directly. On a FVM grid, the modulated concentration is positioned right on the boundary, so the four-point differential formulae is required to evaluate the second derivative in the equation. Figure 4.3 shows the Dirichlet-type boundary condition on the FVM grid and corresponding modulation concentration on the left boundary C_0 as a function of time $f_0(t)$.

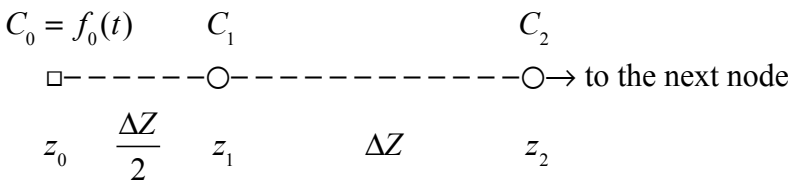

Figure 4.3 Dirichlet-type boundary condition on FVM grid

The second type of the boundary condition is the zero mass flux on the boundary, which is the von Neumann-type boundary condition. In order to get a zero mass flux right on the position of the boundary, but neither inside or outside the reactor, an extra imaginary node C_{-1} is added to the FVM grid. The value of C_{-1} is updated every step of computation as the same value of C_1, so the first derivative at the position of boundary Z_0 is always zero. The von Neumann-type boundary condition on the FVM grid is show in figure 4.4, and the finite difference equation of zero mass flux on the left boundary is provided as following.

$$J = -D\left.\frac{dC}{dZ}\right|_{Z=Z_0} = -D\left(\frac{C_1 - C_{-1}}{\Delta Z}\right) = 0 \text{ , since } C_{-1} = C_1$$

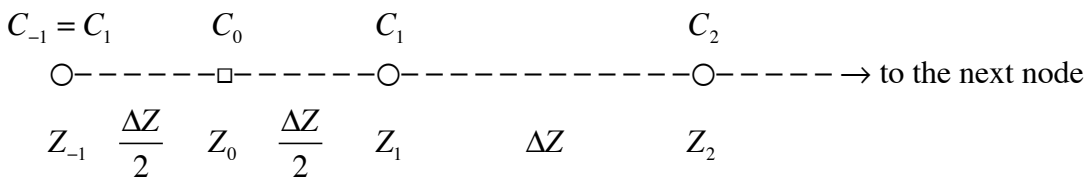

Figure 4.4 von Neumann-type boundary condition on FVM grid

The last type of boundary condition is the balanced mass flux, which is also called the Danckwerts' type boundary condition. The equation for the balanced mass flux on the left boundary is

$$k_{ext}(C_b - C_{z=0}) = D_{eff}\left.\frac{\partial C}{\partial Z}\right|_{Z=0}$$

,where C_b is the bulk concentration in a well mixed chamber attached to the reactor and k_{ext} is the external mass transfer coefficient in the mixing chamber. The left hand side of this equation is the mass flux due to the concentration gradient between the reactor and the bulk concentration, and the right hand side is the diffusive molar flux of the catalytic reactor on the boundary. The Danckwerts' type boundary condition on a FVM grid is shown in figure 4.4 and the finite difference formula for balance mass flux is given as following

$$k_{ext}(C_b - C_0) = D\left(\frac{C_1 - C_0}{\Delta Z}\right)$$

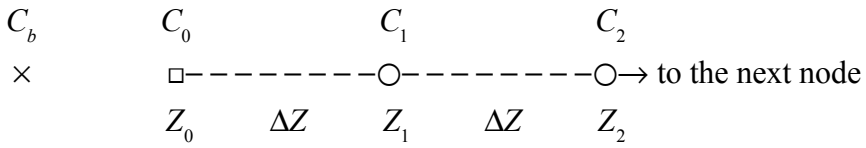

Figure 4.5 Danckwerts' type boundary condition on FVM grid

The grid definition from the finite volume method provides sufficient and reasonable ways to handle different types of concentration and mass flux on the boundary of a catalytic reactor. Based on this grid definition, the node coupling of Langmuir-Hinshelwood kinetics of carbon monoxide oxidation in porous solid catalyst on FVM grid is shown in figure 4.5. The lower plane in the plot is the solid phase and the upper one is the gas phase. Each unit volume is separated by the black solid line on the plan of the solid phase. The circle, triangle, square and x-make nodes represent the concentration in the gas phase, over the solid phase, on the boundary and outside the reactor respectively. All inner nodes are evenly spaced by one unit distance. The dashed blue line

is the node coupling by the gas diffusion and the solid red line is the catalytic reaction between two phases. The left boundary is zero mass flux or von Neumann–type, so the first unit volume half distance to the boundary. The right boundary is the balanced mass flux with external mass transfer coefficient or the Danckwert's type, so the grid between 8 and 9 is the boundary. The grid 10 represents the concentration in attached mixing chamber. The distance from this grid to the reactor boundary on the plot does not matter since the molar flux is computed by the external mass transfer coefficient.

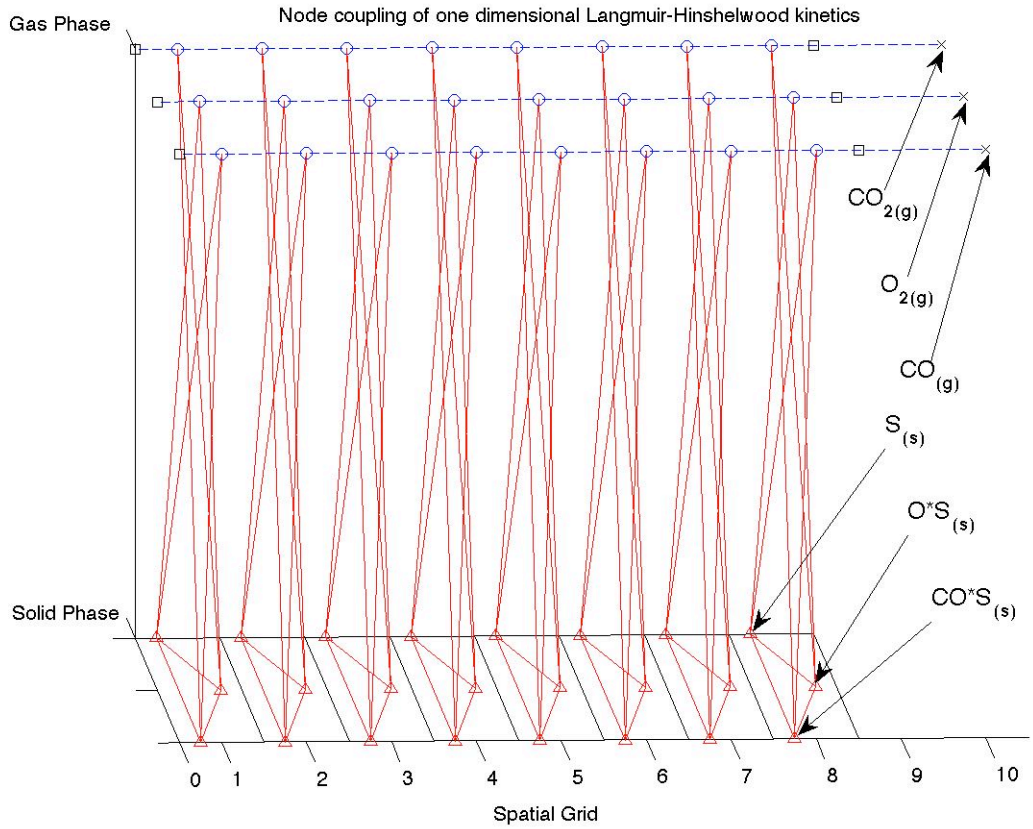

Figure 4.6 Node coupling of Langmuir-Hinshelwood CO oxidation on solid catalyst

4.4 Numerical Method for Porous Catalyst

The grid and boundary condition has been defined in the previous section. In this

section, the numerical methods for the lumped-parameter CSTR and distributed-parameter DDR models will be discussed. The mass balance equations written in the index notation are

$$\text{CSTR}: \begin{cases} \dfrac{dC_i}{dt} = \dfrac{1}{\tau}\left(C_{b,i} - C_i\right) + \alpha S_{ik} R_k \\ \dfrac{dC_i}{dt} = \qquad\qquad S_{ik} R_k \end{cases}$$

$$\text{DDR}: \begin{cases} \dfrac{dC_{i,j}}{dt} = \dfrac{1}{\tau}\left(\dfrac{C_{i,j-1} - 2C_{i,j} + C_{i,j+1}}{\Delta Z^2}\right) + \alpha S_{i,k} R_{k,j} \\ \dfrac{dC_{i,j}}{dt} = \qquad\qquad S_{i,k} R_{k,j} \end{cases}$$

where the index i is for the component, j is for the spatial grid and k is for the rate equation. The subscript for the diffusion time constant is ignored here to minimize confusion with other indices in the equation. Both the CSTR and DDR equations can be written as the combination of linear part L and nonlinear part N.

$$\text{CSTR}: \begin{cases} \dfrac{dC_i}{dt} = L_i + N_i \\ \text{Gas: } L_i = \dfrac{1}{\tau}\left(C_{b,i} - C_i\right),\ N_i = \alpha S_{ik} R_k \\ \text{Solid: } L_i = 0,\ N_i = S_{ik} R_k \end{cases}$$

$$\text{DDR}: \begin{cases} \dfrac{dC_{i,j}}{dt} = L_{i,j} + N_{i,j} \\ \text{Gas: } L_i = \dfrac{1}{\tau}\left(\dfrac{C_{i,j-1} - 2C_{i,j} + C_{i,j+1}}{\Delta Z^2}\right),\ N_i = \alpha S_{i,k} R_{k,j} \\ \text{Solid: } L_i = 0,\ N_i = S_{i,k} R_{k,j} \end{cases}$$

The equations for the algorithm of element constraint method become

$$\text{CSTR}: \begin{cases} \dfrac{dC_p}{dt} = L_p + N_p \\ \dfrac{\partial C_q}{\partial t} = L_q - M^*_{q,p}\left(\dfrac{\partial C_p}{\partial t} - L_p\right) \end{cases}$$

$$\text{DDR}: \begin{cases} \dfrac{dC_{p,j}}{dt} = L_{p,j} + N_{p,j} \\ \dfrac{\partial C_{q,j}}{\partial t} = L_{q,j} - M^*_{q,p}\left(\dfrac{\partial C_{p,j}}{\partial t} - L_{p,j}\right) \end{cases}$$

, where $M^*_{q,p}$ is the index notation for $\mathbf{M}_D^{-1}\mathbf{M}_I$, index p is for independent component and index q is for dependent component. In a two-step ECM algorithm, the linear part of the differential equation is stored while integrating the mass balance equation of the independent component and then the sparse matrix operation of $M^*_{q,p}$ to the pre-stored data and time derivative from the first step is applied to solve for the dependent component. To compute the mass balance equation numerically, there are five major categories of finite difference methods that can be used for the current transient problems. The equations of numerical method build in Transcat for CSTR and DDR model is provided in Appendix B and Appendix C respectively. The intermediate variables and equations listed in the appendix usually come in pairs, since there are equations for both gas and solid phases. The number appending to the abbreviation of the numerical method means the order of accuracy. The explicit, implicit, semi-implicit, splitting step and other high order numerical method are summarized [1-7].

The major explicit methods that can be used to integrate the mass balance equations in two phases are the Explicit Euler (EE), Runge-Kutta (RK) and Adams-Bashforth (AB) method. The common feature of these explicit methods is to treat both

the linear and nonlinear part of the differential equation explicitly, so they can be computed directly from the data of current and previous time steps. The benefits of explicit methods are that they are easy to implement and some methods like RK and AB can achieve very high order of accuracy. However, the biggest problem of this type of numerical method is the stability. When the differential equation is getting stiff, small time steps must be taken to make these methods stable. Sometimes the steps are too small to make these explicit methods feasible.

To compute a stiff problem, a preferred choice is to use the implicit methods, such as implicit Euler (IE), backward differential formulae (BDF) and Adams-Moulton (AM) methods. These implicit methods can stabilize a stiff problem, so larger time steps can be taken than in the explicit methods. Newton's method is commonly used to solve the algebraic equations derived from the finite difference equations, but it usually creates huge Jacobian matrix in a distributed problem like the DDR model and it also requires large storage space in the computer memory. The challenge for the implicit method is to solve the linear algebraic equation of the Jacobian matrix. The nonlinear reaction part of the mass balance equation makes the Jacobian matrix a function of concentration, so it needs to be updated in every single step of iteration. Therefore, a cheap algorithm like LU decomposition will not be a good candidate. Other iteration methods such as Gauss-Seidel or successive over-relaxation have sometimes been used, but the speed to convergence is still dependent largely on the structure of the Jacobian matrix. An alternative algorithm to save the storage is used the Jacobian free Newton-Krylov method, but extra work is required to approximate the Jacobian matrix and link it to the Krylov subspace [18].

For a one-dimensional diffusion-reaction problem, the structure of the Jacobian matrix resembles the plot shown on the left of figure 4.6. Each block with black border represents a node. The red squares are the first derivative of the LHHW mechanism, blue squares are the node coupling of diffusion and the green squares are the combination of diffusion and reaction. In Transcat, the sparse and banded Jacobian matrix is stored in a compact form without any effort of programming by the user and this storage format is shown on the right plot of figure 4.6 [2]. This format can eliminate unnecessary operations on the zero elements and can also reduce demand of the computer memory when solving the linear system. For the algorithm of ECM, the Jacobian matrix and its compact form are shown in figure 4.8, where the yellow squares are the node coupling of element conservation. The ECM algorithm changes the structure of the Jacobian matrix and generates extra diagonals in the Jacobian matrix, so it may not be a good choice for implicit methods.

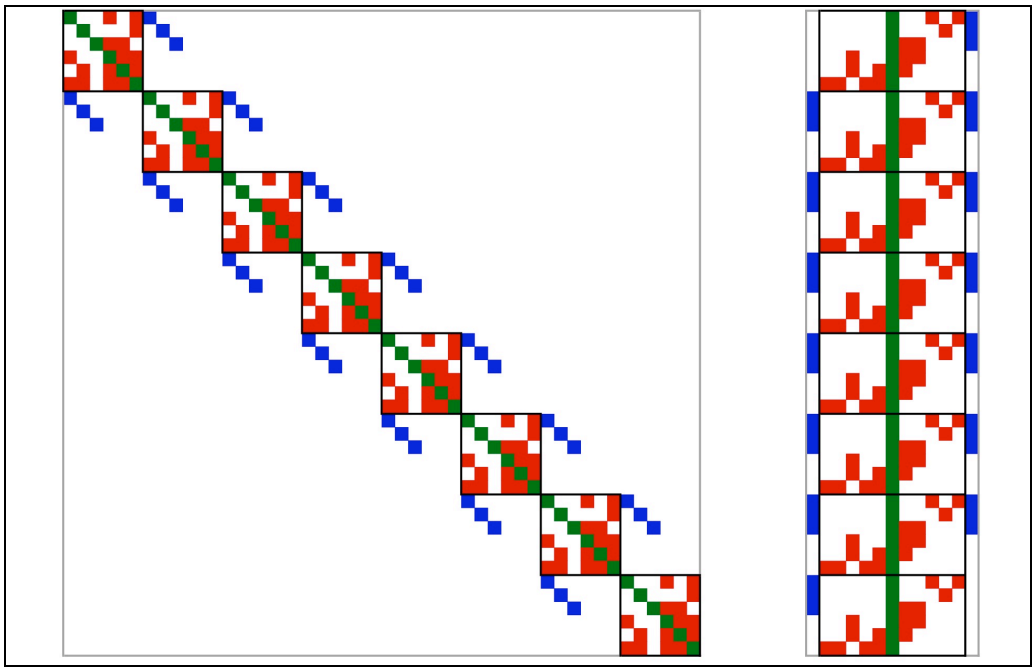

Figure 4.7 Jacobian matrix of LHHW kinetics

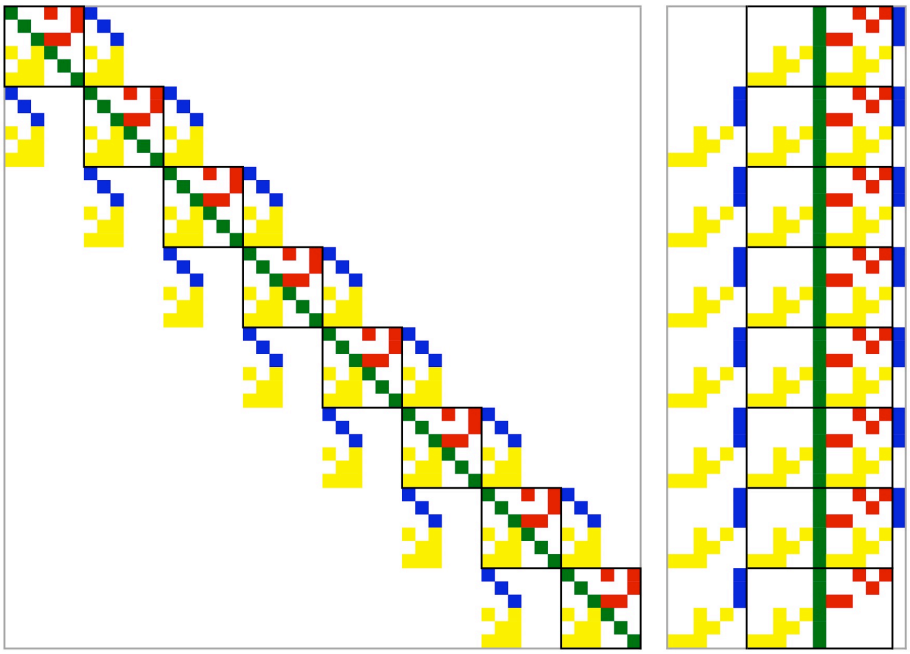

Figure 4.8 Jacobian matrix of LHHW kinetics with ECM algorithm

An intermediate method to take into account the stability, stiffness and the difficulty of implementation is a semi-implicit method. An example in this category is the combination of the Crank-Nicholson and Runge-Kutta methods (CNRK). The Crank-Nicholson method treats the linear part of the equation as half explicit and half implicit, and the nonlinear reaction part is integrated by a fully explicit Runge-Kutta method. The Crank-Nicholson method is unconditionally stable for an unsteady diffusion equation, so it is a good choice for a diffusion dominant problem. The half implicit part of the diffusion term will result in a diagonally banded matrix, which can also be stored in a compact form and solved by the cheap Thomas algorithm or LU decomposition.

The Crank-Nicholson method for a diffusion problem of two or more dimensions is usually replaced by the alternating direction implicit method (ADI) to improve the computing speed. The ADI method is a splitting method, which divide a single time step

into small sub-steps of equal duration and alternate the implicit dimension one at a time. The idea of a splitting method can also be applied to the one dimensional unsteady diffusion-reaction problem. From the node-coupling plot in figure 4.5, the diffusion coupling for gas species in the Z direction is the first dimension, and the reaction coupling between components can be treated as the second one. In the first sub-step, the diffusion term is solved implicitly, so a linear algebraic equation is derived. In the second sub-step, the component on each grid is treated as implicit. Since the reaction only takes place in a single unit volume, the Jacobian matrix is constructed only by the components on that grid. The size of the Jacobian matrix equals the number of components in that reaction system, so the storage is much smaller and the computation cost is much cheaper than an implicit method.

There are some other methods that can be used to solve the unsteady diffusion-reaction problem. In a semi-spectral method, the three steps are (1) transform and compute the diffusion term in the Fourier or Chebyshev domain, (2) inverse back to the time domain, and finally (3) integrate together with the reaction part by a high order explicit method. In the integrating factor method, the whole equation is multiplied by e^{-Lt}, where L is the linear part of the equation, to make it less stiff, but this method only works for diagonal problem such as the CSTR model. A state-of-the-art high-order method for stiff nonlinear PDEs is the exponential time-differencing, fourth-order Runge-Kutta method (ETDRK4) [19-21]. This method has been proven to be stable, fast and accurate on other unsteady diffusion reaction problems like the Allen-Cahn equation. The major challenge to implement this in Fortran is that it requires other mathematical tools to handle the complex analysis and contour integral.

The simulator for the porous catalyst is designed to handle general surface kinetics under transient conditions. Multiple numerical methods are incorporated into the simulator, so the user can choose a method according to the operating conditions. Not all of the numerical methods listed Appendix B and Appendix C are implement in Transcat, but the algorithm is well structured, so they can still be added in the future.

Reference:

[1] Beers, K. J. (2007). Numerical methods for chemical engineering : applications in Matlab®. Cambridge, UK ; New York, Cambridge University Press.

[2] Press, W. H. (1996). Numerical recipes in fortran 90 : the art of parallel scientific computing. Cambridge [England] ; New York, Cambridge University Press.

[3] Hairer, E., S. P. Nørsett, et al. (1993). Solving ordinary differential equations. Berlin ; New York, Springer-Verlag.

[4] Hairer, E. and G. Wanner Solving ordinary differential equations II. Stiff and differential-algebraic problems. Heidelberg ; New York, Springer: 1 online resource (xv, 614 p.).

[5] Lapidus, L. and G. F. Pinder (1982). Numerical solution of partial differential equations in science and engineering. New York, Wiley.

[6] Pozrikidis, C. (1998). Numerical computation in science and engineering. New York, Oxford University Press.

[7] Hoffman, J. D. (1992). Numerical methods for engineers and scientists. New York, McGraw-Hill.

[8] Herz, R. K. and S. P. Marin (1980). "Surface-Chemistry Models of Carbon-Monoxide Oxidation on Sported Platinum Catalysis." Journal of Catalysis 65(2): 281-296.

[9] Eisenstat, S. C., M. C. Gursky, et al. (1982). "Yale Sparse-Matrix Package .1. The Symmetric Codes." International Journal for Numerical Methods in Engineering 18(8): 1145-1151.

[10] Bank, R. and C. Douglas (1993). "Sparse matrix multiplication package (SMMP)." Advances in Computational Mathematics 1(1): 127-137.

[11] D'Azevedo, E. F., M. R. Fahey, et al. (2005). Vectorized sparse matrix multiply for compressed row storage format. Computational Science - Iccs 2005, Pt 1, Proceedings. V. S. Sunderam, G. D. VanAlbada, P. M. A. Sloot and J. J. Dongarra. 3514: 99-106.

[12] Cho, B. K. (1983). "Dynamic Behavior of a Single Catalyst Pellet .1. Symmetric Concentration Cycling During CO Oxidation over Pt/Al2O3." Industrial & Engineering Chemistry Fundamentals 22(4): 410-420.

[13] Koci, P., M. Kubicek, et al. (2004). "Periodic forcing of three-way catalyst with diffusion in the washcoat." Catalysis Today 98(3): 345-355.

[14] Schiesser, W. E. (1991). The numerical method of lines : integration of partial differential equations. San Diego, Academic Press.

[14] Sincovec, R. F. and N. K. Madsen (1975). "Software for Nonlinear Partial Differential Equations." ACM Trans. Math. Softw. 1(3): 232-260.

[16] Hindmarsh, A. C. (1983). ODEPACK, A Systematized Collection of ODE Solvers. Scientific computing : applications of mathematics and computing to the physical sciences. R. S. Stepleman. Amsterdam: 55-64.

[17] Ciarlet, P. G. and J. L. Lions (1990). Handbook of numerical analysis. Amsterdam ; New York, North-Holland.

[18] Knoll, D. A. and D. E. Keyes (2004). "Jacobian-free Newton-Krylov methods: a survey of approaches and applications." Journal of Computational Physics 193(2): 357-397.

[19] Cox, S. M. and P. C. Matthews (2002). "Exponential time differencing for stiff systems." Journal of Computational Physics 176(2): 430-455.

[20] Kassam, A. K. and L. N. Trefethen (2005). "Fourth-order time-stepping for stiff PDEs." Siam Journal on Scientific Computing 26(4): 1214-1233.

[21] CRASTER, R. V. and R. SASSI (2006). "Spectral algorithms for reaction-diffusion equations." Technical Report. Note del Polo.

Chapter 5 Transcat: Numerical Simulator for Modeling Transient Gas-Solid Kinetics in Porous Catalysts

The simulator built based on the theory and algorithm described from previous chapters is called Transcat, which is the syllabic abbreviation of *transient* and *catalysis*. This simulator is designed to model transient diffusion-reaction in porous catalyst with detailed gas-solid kinetics. The main structure of Transcat is written in the Fortran90 language and the simulator can run on both Macintosh and Windows platforms. Users only need to edit the variables and parameters in a simple input file, no further programming is required to run the simulation of common surface kinetics. The simulation result is exported as text and extensible markup language (XML) files, which can be easily read or handled by other applications. Users can also use the toolbox developed in Matlab to analyze, visualize and post-process the output data.

5.1 Design of TransCat

The whole idea of TransCat is to design a general solver for modeling dynamic diffusion-reaction behavior in porous catalysts under transient conditions. TransCat is capable of handling all kinds of detailed gas-solid kinetics yet it is still easy to use. This simulator is basically free of coding: the user only needs to edit a simple input file to run the simulation unless a special model for the surface kinetics is desired. Two models of reactor, three types of boundary conditions, multiple numerical methods and an optional element constraint method described in previous chapters are built into TransCat. Users

can run the simulator with concentration and temperature modulation of various types of waveforms without the effort of programming.

Multiple numerical methods can be chosen from a list of explicit, explicit-implicit or implicit methods. Adaptive time control is integrated into the simulator, so a user doesn't have to worry about the instability induced by a large step size. The estimated runtime required to finish is shown on the console window while the program is running, so the user can stop the simulation and change to another method when the modeling condition is getting too stiff or higher accuracy is desired. The element constraint method described in previous chapters is integrated into these numerical methods. It is provided as an optional feature, so a user can choose to solve the unsteady diffusion reaction equation systems directly without considering the element conservation, or use the element constraint method for either the surface species or maximum degree of freedom available.

The current version of TransCat supports temporal non-isothermal modeling with user input temperature modulation parameters. Numerical experiments such as light-off, quenching and periodic cycling of temperature of porous catalyst can be modeled by adjusting these input parameters. The diffusion coefficient, reaction rate and all other physical numbers are updated every time step according the variation of temperature. Solving the energy equation simultaneously with the concentration is currently not integrated into TransCat, but the space for computing the temperature profile is preserved in the source code for future development. The structure and design of Transcat is shown in figure 5.1, including the main program, input files, output files and post-processing toolbox developed under Matlab.

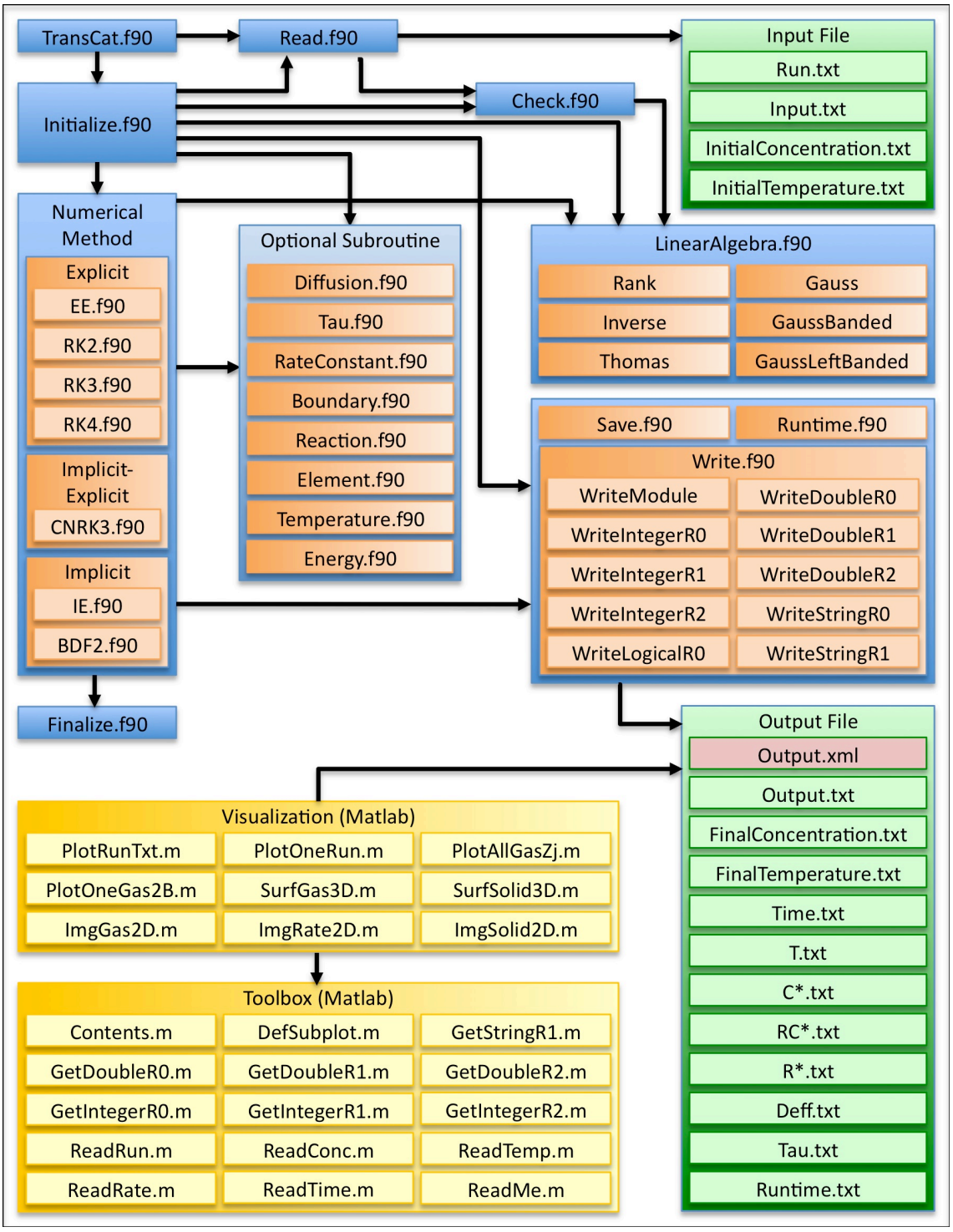

Figure 5.1 Structure and file relation of Transcat

The main flow of Transcat is shown on the upper left part of figure 5.1. In the initialization process, the simulator first reads in the input files and then checks all the input parameters. Transcat stops automatically if these parameters are incomplete, out of range or conflict with each other. The major numerical solver then calls the linear algebra packages or other optional subroutines when needed and it also writes the output files during computation to save memory usage. When the simulation is done, Transcat close all the output files and releases the memory back to the machine in the finalization step. After the simulation, the user can directly open the output files recorded in either text or XML format, or access the data by using the script file and toolbox developed in Matlab. The input files, the Fortran source code and Matlab toolbox are attached in Appendix D and Appendix E.

5.2 Running Simulation in Transcat

Running Transcat is simple. The user only needs to edit two files, run.txt and input.txt, before executing Transcat. The file run.txt is a batch file, which specifies all the numerical experiments to be carried out. Transcat executes experiments in the batch file one by one until it is finished. The file input.txt specifies the input parameters for each experiment and is saved under corresponding folders with the same name. An example batch of four numerical experiments are going to be described in this section and the input files of this batch can be found in Appendix D.

Assume we want to study the Langmuir-Hinshelwood carbon monoxide oxidation in a porous catalyst under constant feed with three different types of waveforms under periodic cycling. The first thing to do is to edit the batch file “Run.txt” and give each

experiment a name. The next is to create four folders with the same filename as in the batch file and then copy a template of file “input.txt” into these folders. Edit the input file and change the parameters to desired values before running Transcat. The user can import an initial value for the concentration or temperature distribution in the reactor or create a fresh start directly from the input file. The only difference between all the input files of this example batch is the parameter *BCW*, which specifies the modulated waveform of the boundary condition. The last thing to do is to execute the compiled application Transcat for corresponding Macintosh or Windows platform and wait for the simulation to finish. A screen shot of the console window of a running program is shown in figure 5.2. The progress and estimated time required to finish is shown on the window while the program is running.

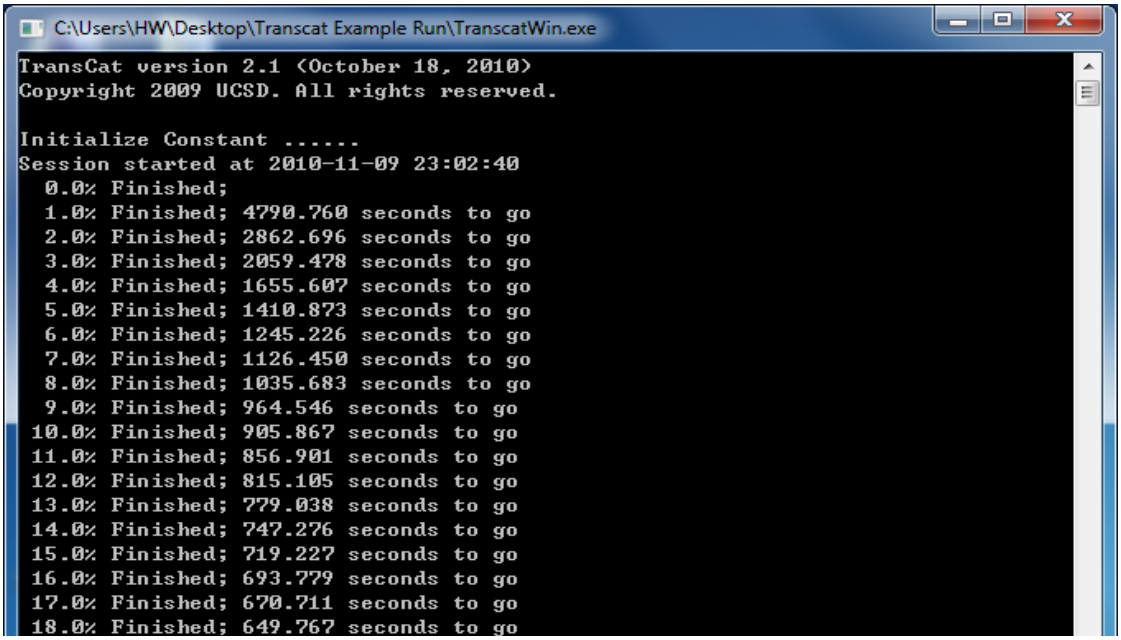

Figure 5.2 Screen shot of running Transcat in Windows

In the last step of a running Transcat, multiple output files will be stored in the folder with the corresponding name. The user can open the text and XML files directly to

see the result or use the visualization toolbox developed in Matlab. Figure 5.3 shows the generation rate of carbon dioxide under the concentration modulation of carbon monoxide of constant feed, sine wave cycling, square wave cycling and saw-tooth wave cycling. This plot is generated by the post-processing tools developed in Matlab. The detail of this tool and other visualization applications will be introduced in the next section.

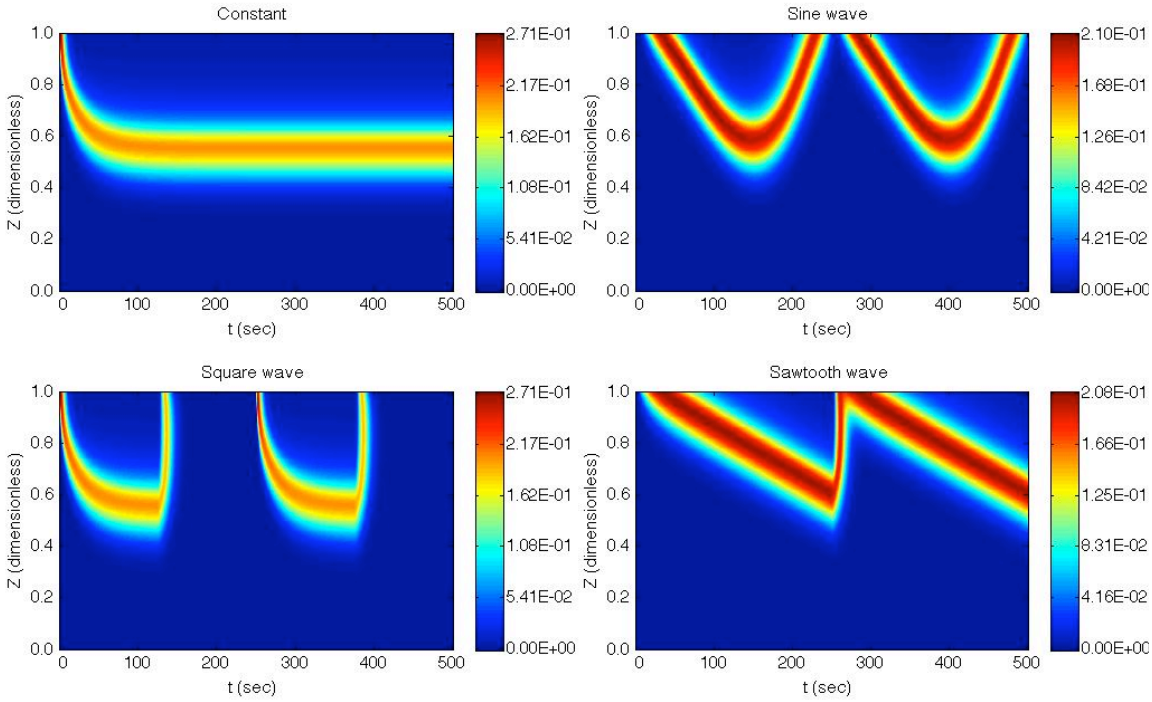

Figure 5.3 Local generation rate of CO_2 under various CO concentration inputs. Z is the dimensionless position in the porous catalyst. The zero-flux boundary is at Z=0.

5.3 Post-Processing and Visualization of Transcat

The post-processing tools for Transcat are currently built in Matlab®. The common procedure of all the post-processing applications are first to read the output file "output.xml" which stores all parameters of a simulation. The XML format of this

document cat be easily processed by many applications. The second step is to acquire the dynamic variables needed for the post-processing and visualization. The dynamic variables are the variables updated every time step, so they are stored in separate files due to massive size. The information from the XML file can redirect the program to find these files and load the dynamic variables into the program. The last step is to call the function of plotting or animation for visualization. There are several major functions to visualize the result computed from Transcat, and the saw-tooth wave modulation experiment from previous section is used as an example to be processed by these functions. Figure 5.4 is the change of gas pressure and surface coverage vs. time, and all six components involved in the LHHW kinetics are plotted by the functions "ImgGas2D.m" (Appendix E Table E.34) and "imgSolid2D.m" (Appendix E Table E.35) . Figure 5.5 is the image plot of all reaction rates of LHHW kinetics, including the adsorption and desorption of carbon monoxide, adsorption of oxygen and the generation of carbon dioxide. The corresponding mechanism is shown on the top of each subplot automatically. Figure 5.6 is the gas pressure on the boundaries of porous catalyst under saw-tooth modulation of carbon monoxide, and the corresponding gas component is shown on the top of each subplot. The tools can also animate the variation of gas concentration and surface coverage with time. Figure 5.7 shows eight instances of the animation of gas pressure and surface coverage of LHHW kinetics in porous catalysts under the saw-tooth modulation of carbon monoxide feed, and the time variable of simulation is updated on the title of the animation. There are also other tools to compare all the simulation results in single batch like "PlotRunTxt.m" (Appendix E Table E.27),

which generates the plots of gas concentration and surface coverage, reaction rates and the gas response on the boundaries.

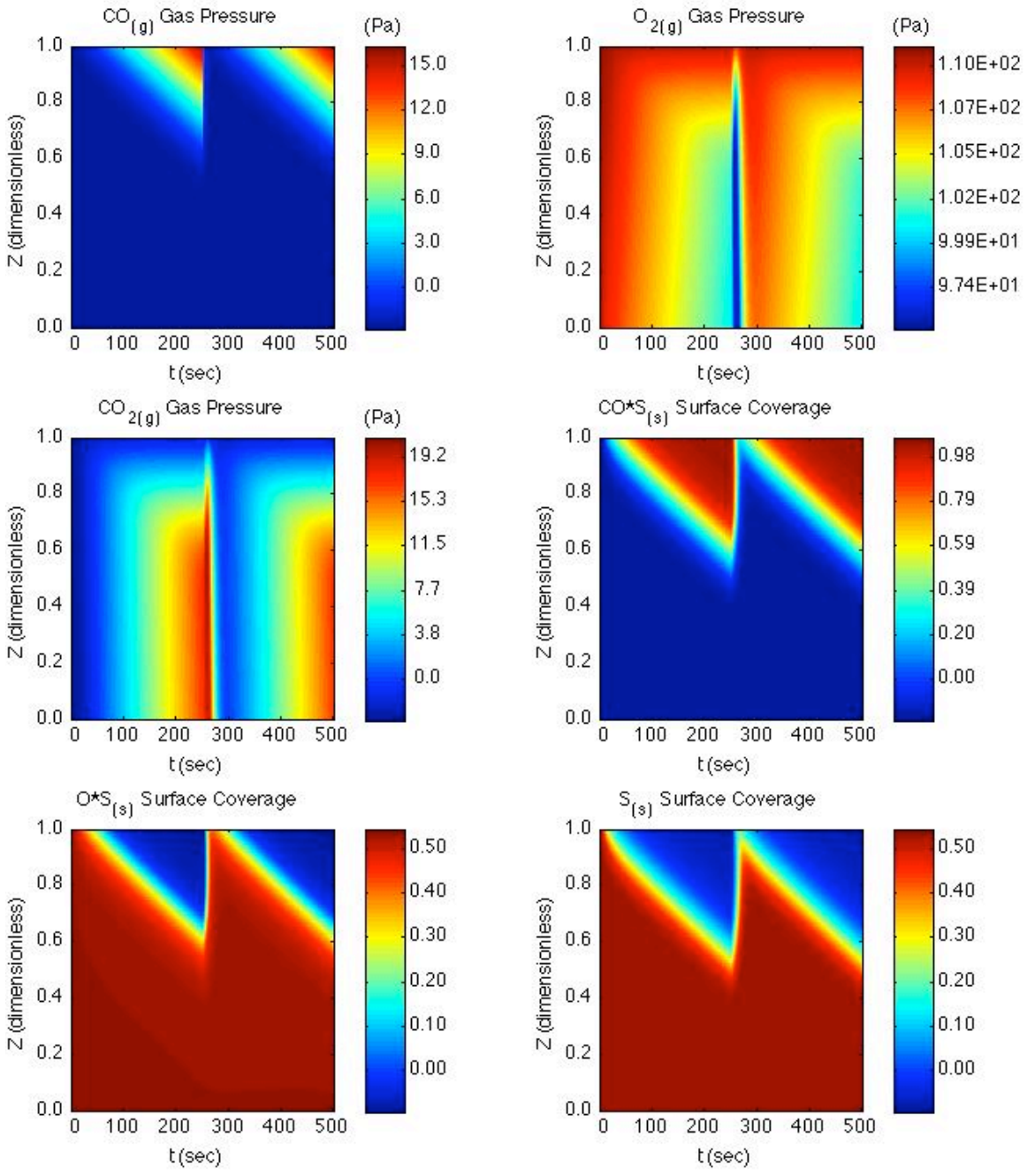

Figure 5.4 Gas pressure and surface coverage of LHHW kinetics in porous catalysts under saw-tooth modulation of carbon monoxide

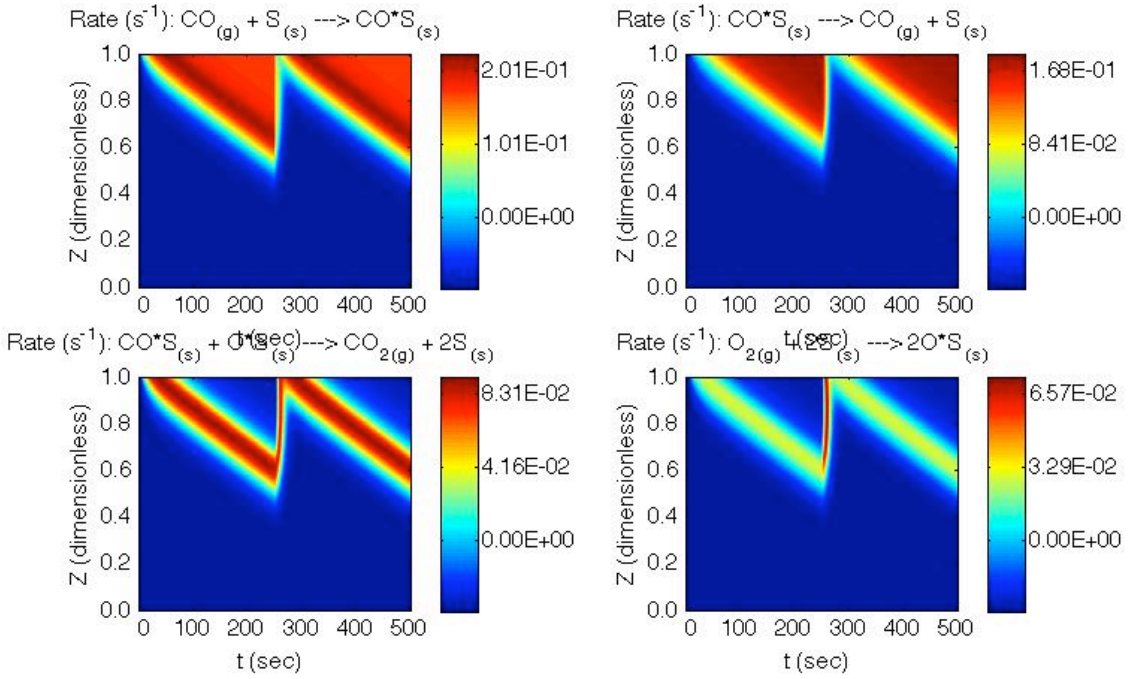

Figure 5.5 Adsorption, desorption and reaction rate of LHHW kinetics in porous catalyst under saw-tooth modulation of carbon monoxide

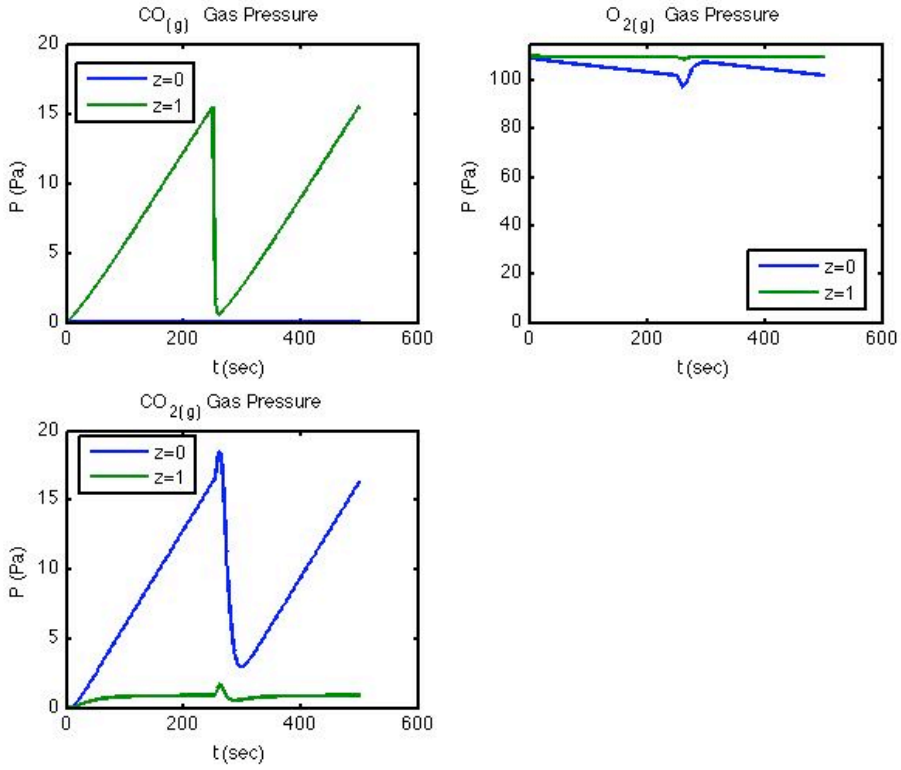

Figure 5.6 Gas pressure LHHW kinetics on the boundaries of porous catalysts under saw-tooth modulation of carbon monoxide

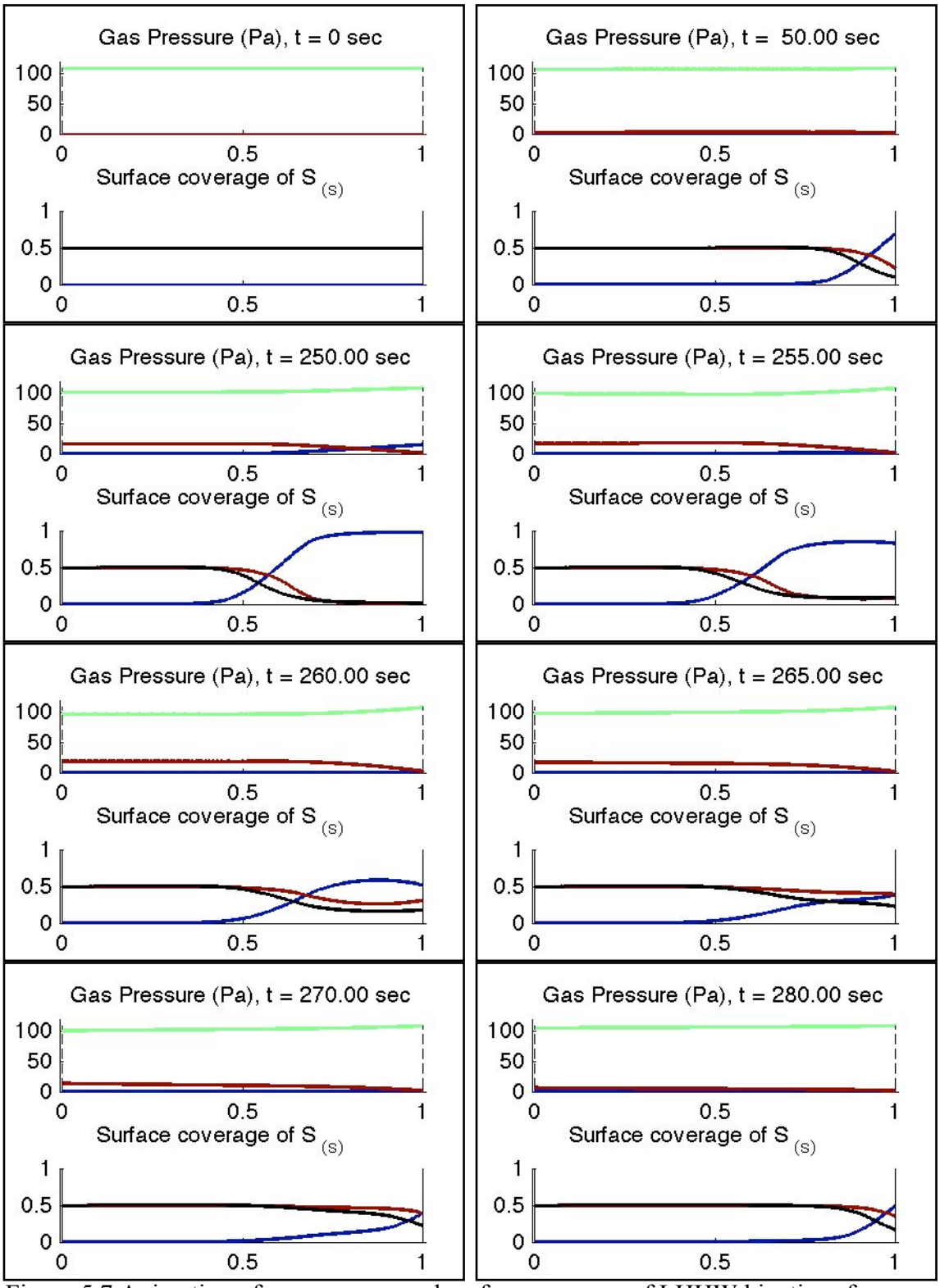

Figure 5.7 Animation of gas pressure and surface coverage of LHHW kinetics of porous catalysts under saw-tooth modulation of carbon monoxide

5.4 Future Development of Transcat

The current version of Transcat is capable of modeling the heterogeneous reaction in a porous catalyst and running simulations of numerical experiments under various transient conditions, but there are more features should be developed for this general solver. The algorithm of solving the energy equation together with the mass balance equation is not integrated. Transcat currently can only handle spatially isothermal simulations and non-isothermal cases with spatially-controlled temperature. Numerical stiffness is still a problem for direct numerical simulation of catalysis under the realistic parameter values. Even using a stable numerical method, the simulation time may take days and weeks. More state of the art numerical methods should be attempted to improve the efficiency.

The user interface for running the simulation and post-processing are spread into several pieces. The simulator itself is an executable Fortran application, the input and output file are in either text or XML format and the post-processing tools are written in Matlab. An application with better graphic user interface (GUI), which integrates all the procedures, should be developed. The Fortran source code can be compiled and build into a dynamic-link library, which can be called by the application. The future development of Transcat should make it as user friendly as possible, where the user can use the GUI to edit the input parameters, run the simulation, process the output data and visualize the result.

Chapter 6 Validation of Transcat

The simulation results of Transcat must be able to accurately model any transient diffusion reaction in a porous catalyst. In order to validate the correctness of simulation results from Transcat, several approaches are taken and are described in this chapter. When shutting off the surface reaction by either setting the temperature low or by using infinitesimal amount of catalyst filling in the reactor, the result should be equivalent to an unsteady diffusion equation. When the simulation time is long enough, the model can approach a steady state, where analytical solutions exist for certain surface kinetics. When a locally equilibrium adsorption model is simulated, the result should conform to an approximated solution under the partial equilibrium assumption. Transcat is also tested against the Fisher-Kolmogorov equation where an exact solution exists for this homogeneous transient diffusion reaction equation [1,2]. Transcat must be able to fit real experimental data as well, so it can be used to predict the dynamic response of a porous catalysts under different operation conditions. In this chapter, a variety of models are run in Transcat and compared to the solution of prior cases to validate the correctness of the simulator.

6.1 Inert Gas Diffusion in Porous Solid Catalysts

The first validation case for TransCat is inert gas diffusion in a porous catalyst. When a single gas component is released into the catalyst with low adsorption rate, high desorption rate and no surface reaction, the gas concentration profile should closely conform to the analytical solution of an unsteady diffusion equation. To model the inert

diffusion process in Transcat, the ratio of adsorption to desorption rate are set small and boundary conditions are the von Neumann-type boundary on one side and Dirichlet-type boundary on the other. The analytical solution to the unsteady diffusion equation with the same boundary condition is given in the following formula [3]. The simulation result from Transcat compared to this solution is shown in figure 6.1.

$$C_i(Z,t) = 1 + \sum_{n=1}^{\infty} \frac{2(-1)^n}{\left(n-\frac{1}{2}\right)\pi} \cos\left[\left(n-\frac{1}{2}\right)\pi Z\right] \exp\left[-\frac{1}{\tau_i}\left(n-\frac{1}{2}\right)^2 \pi^2 t\right]$$

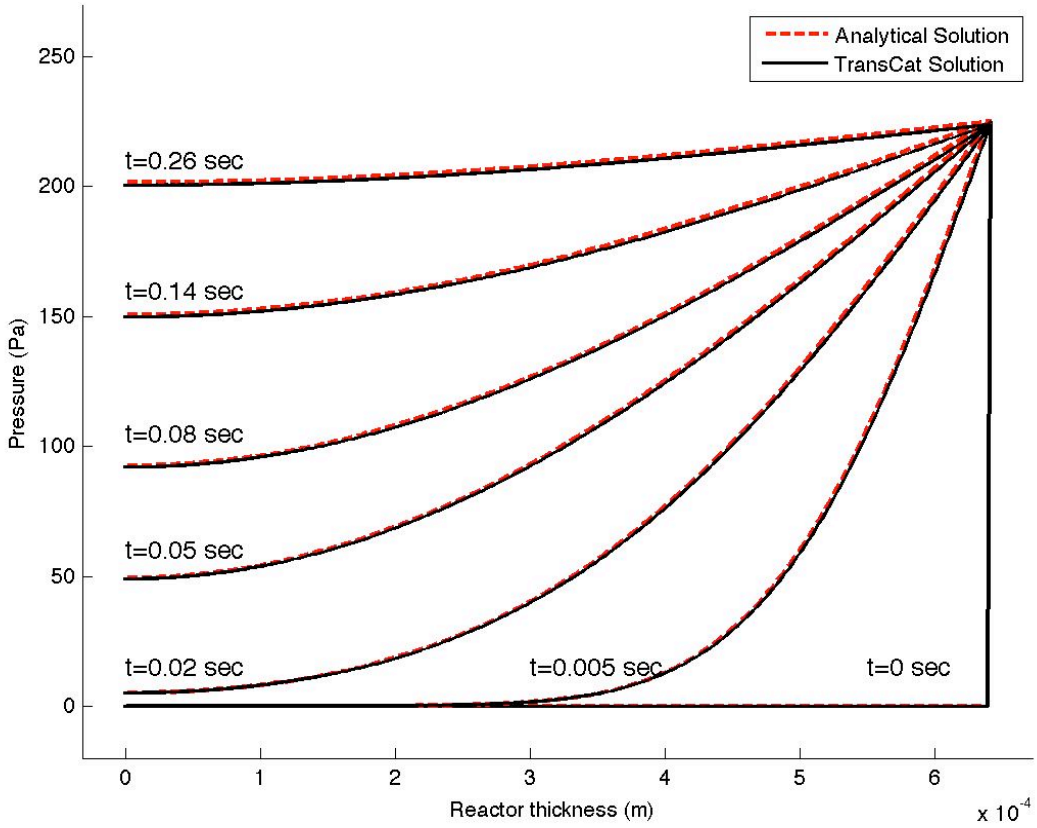

Figure 6.1 Comparing the gas concentration profile of analytical solution of unsteady diffusion equation to the simulation result from Transcat computed by Crank-Nicholson with third-order Runge-Kutta method

The response of gas concentration shows good agreement to both the analytical solution and the simulation result from Transcat. It is reasonable that the concentration

curves computed from Transcat are lower due to a small adsorption and desorption process in Transcat and the inherent error of the numerical method as well. The modeling of inert diffusion is also an important approach to determine the Knudsen diffusion coefficient of a gas in porous catalyst by fitting the simulation result to experimental data. Once the diffusivity of inert Knudsen flow is found, it can be used as a reference to calculate the diffusion coefficient for other gas components in the same porous catalysts. Figure 6.2 is the response of gas concentration of Neon at 308 K to a step change on one side of dynamic diffusion reactor [4]. The zigzag curves are the measured gas concentration in the mixing chamber attached to either side of the reactor. The green and blue lines are the simulation result from Transcat by adjusting the effective diffusion coefficient and volume flow rate on the boundary. As seen in figure 6.2, the simulation result can accurately fit the gas concentration of Neon measured on both side of the dynamic diffusion reactor. The diffusion coefficient which fits the experimental data in the figure is 4.80×10^{-7} m^2/sec.

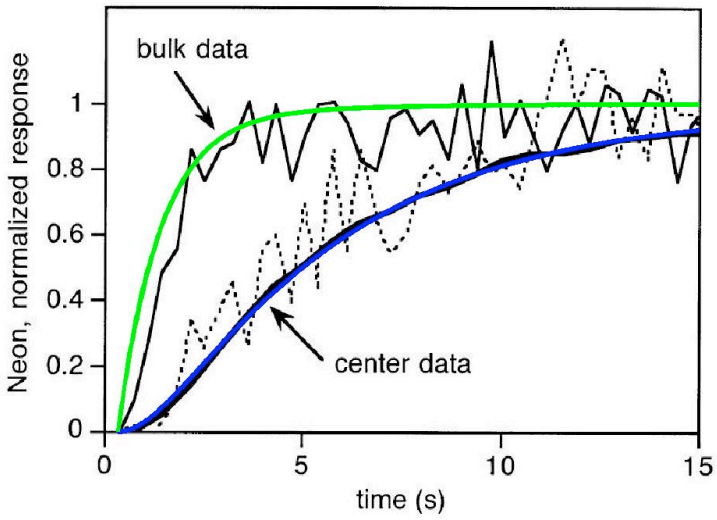

Figure 6.2 Inert diffusion response of neon in a dynamic diffusion reactor. Data are solid black and dashed lines from [4]. Colored curves are Transcat results.

6.2 Steady-State Linear Adsorption

Consider a surface mechanism of adsorption, desorption and surface reaction in a porous catalyst.

$$\begin{cases} A_{(g)} + S_{(s)} \underset{k_2}{\overset{k_1}{\rightleftharpoons}} AS_{(s)} \\ AS_{(s)} \xrightarrow{k_3} B_{(g)} + S_{(s)} \end{cases}$$

The adsorption and desorption rates of these kinetics can be designed to make the equilibrium coverage of A about 0.01. The reaction rate can also be set to a small number which is much smaller than the rate of desorption such that the surface reaction only affects the surface coverage of A slightly. The surface coverage of A and reaction rate is approximately first-order in gas concentration of A in the following equation.

$$r_1 = k_1 a \Theta_A \approx k_1 a K_A C_A$$

where a is the moles of catalytic site per unit volume in mol/m^3, Θ_A is the surface coverage of component A, K_A is the equilibrium constant of adsorption and desorption of A in m^3/mol and C_A is the concentration of A in mol/m^3. According to this set up, the results for steady-state concentration of A can be checked against results expected for steady state, first-order, essentially irreversible reaction. Figure 6.3 shows the gas concentration and the surface coverage of this mechanism approaching steady state from an initial transient. The resulting Thiele modulus in this case is

$$\phi = L\sqrt{\frac{k}{D_{eff}}} = 6.4 \times 10^{-4} \sqrt{\frac{1.5295}{4.0 \times 10^{-7}}} = 1.2515$$

and the effectiveness factor within pore is [5,6]

$$\eta = \frac{\cosh(0)}{\cosh(\phi)} = 0.529$$

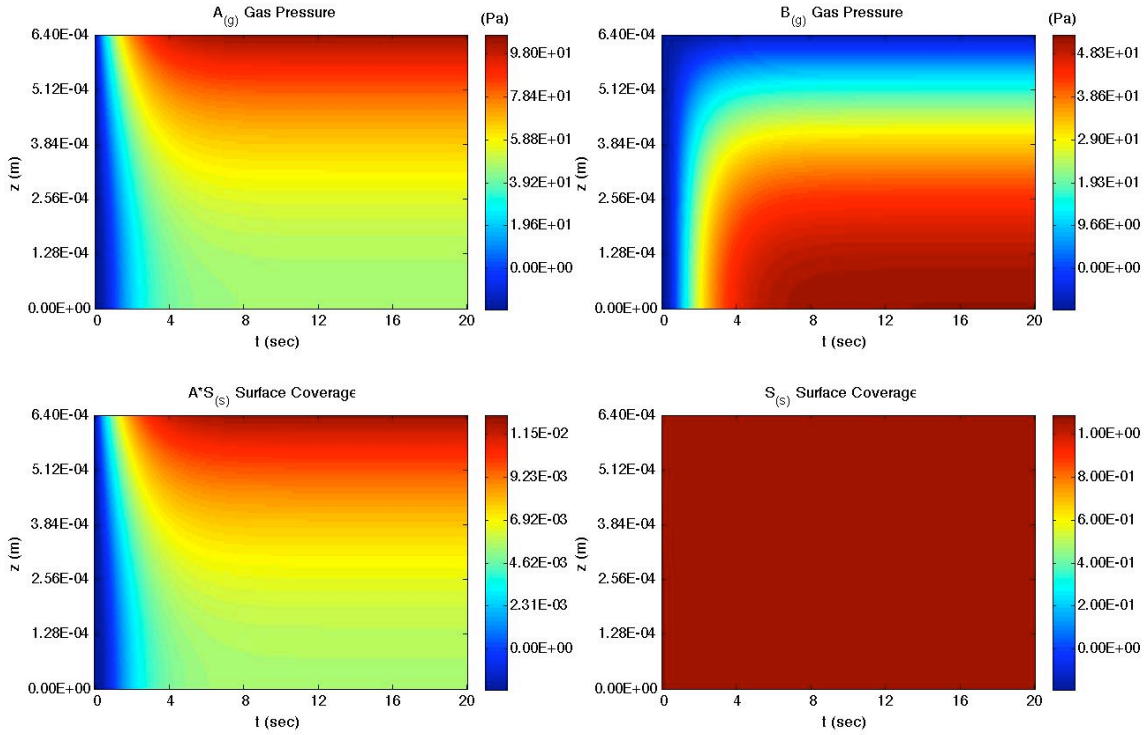

Figure 6.3 Gas concentration and surface coverage

The effectiveness factor of first-order reaction also equals to the ratio of gas concentration on two sides of the boundary, which can be calculated from the gas concentration on the boundary shown in figure 6.4.

$$\eta = \frac{C_A(z=0)}{C_A\left(z=6.4\times10^{-4}\right)} = \frac{P_A(z=0)}{P_A\left(z=6.4\times10^{-4}\right)} = \frac{51.70}{97.96} = 0.528$$

The effectiveness factor calculated from the computation result of Transcat match the shows good agreement to the steady state, first order irreversible reaction within a pore.

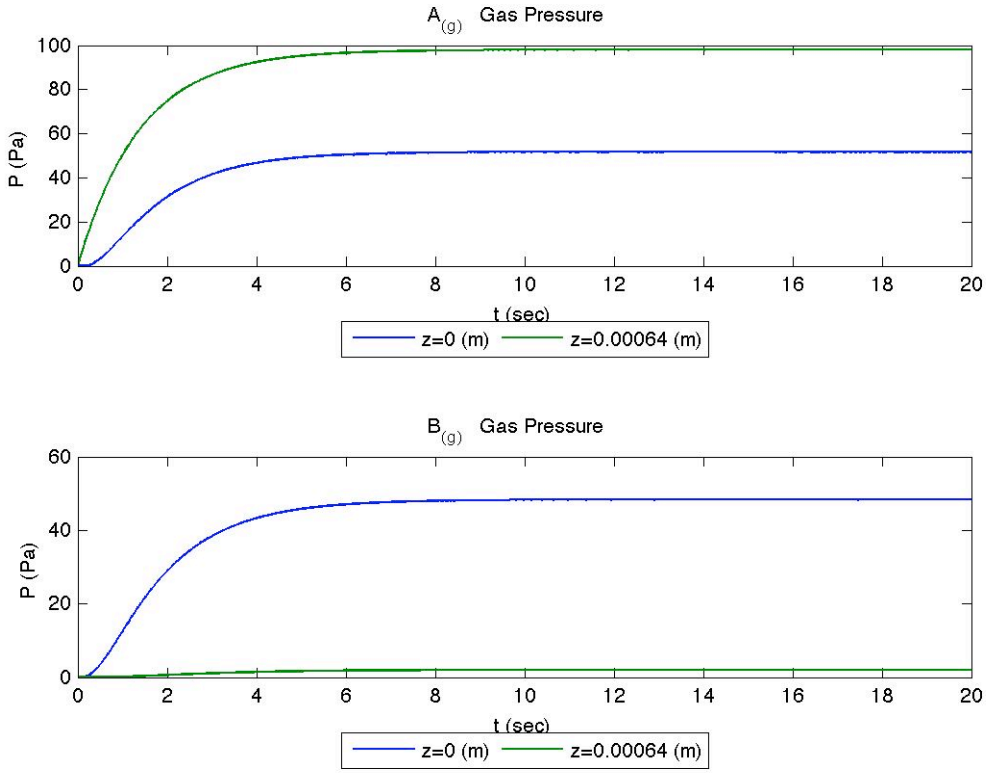

Figure 6.4 Gas concentrations approaching steady state

Other than steady state adsorption, Transcat also gives good result in modeling the transient response of carbon monoxide adsorption and desorption on a Pt/Al_2O_3 catalyst. Figure 6.5 is a concentration modulation of a square wave of carbon monoxide on one side of porous pellet. Transcat gives good fitting under realistic adsorption and desorption parameters. The only inconsistency is when the gas concentration is turn off, the dropping of gas concentration is not as fast as predicted by Transcat. This may be caused by the fluctuation of pumping rate, which cannot hold constant flow when approaching ultra high vacuum, so the response of gas concentration may be slightly different under extreme conditions.

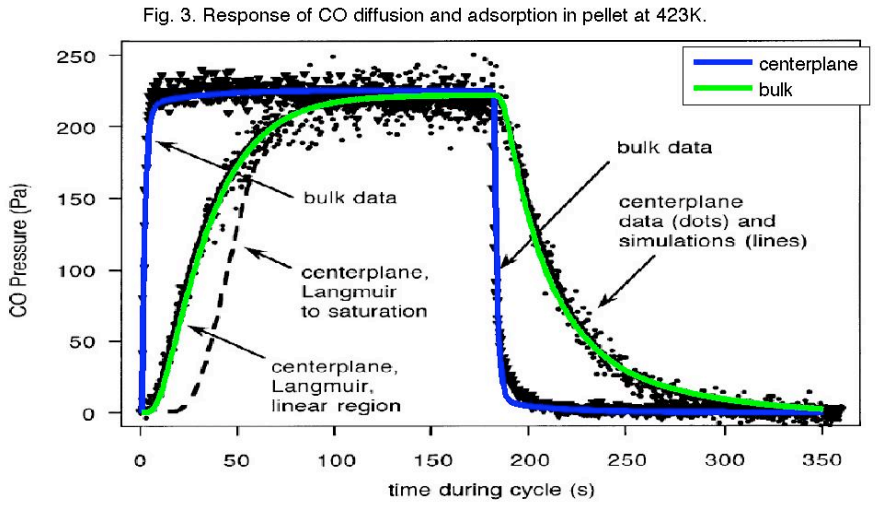

Figure 6.5 Transient response of CO in catalyst pellet. Dots and black dashed lines are experiment data and model from [4]. Colored curves are Transcat results.

6.3 Steady-State Nonlinear Adsorption

When the simulation runs long enough, the result of the transient diffusion-reaction equation with constant boundary condition should approach the solution of a steady state equation. For certain surface kinetics, the analytical solution exists at steady state such as the adsorption and desorption of single gas component in a porous catalyst. However, the analytical solution of this type of process is constant all over the reactor, which is unsuitable for validation of concentration profiles. When steady state is achieved, a more complicated surface kinetics can be used to validate the result of Transcat such as the following mechanism.

$$\begin{cases} A_{(g)} + S_{(s)} \underset{k_2}{\overset{k_1}{\rightleftharpoons}} AS_{(s)} \\ AS_{(s)} + S_{(s)} \xrightarrow{k_3} B_{(g)} + 2S_{(s)} \end{cases}$$

The gas concentration and surface coverage of this kinetics solved by Transcat is

provided in figure 6.6. The simulation time is set long enough to pass the initial transient and both the gas concentration and surface coverage almost achieve steady state.

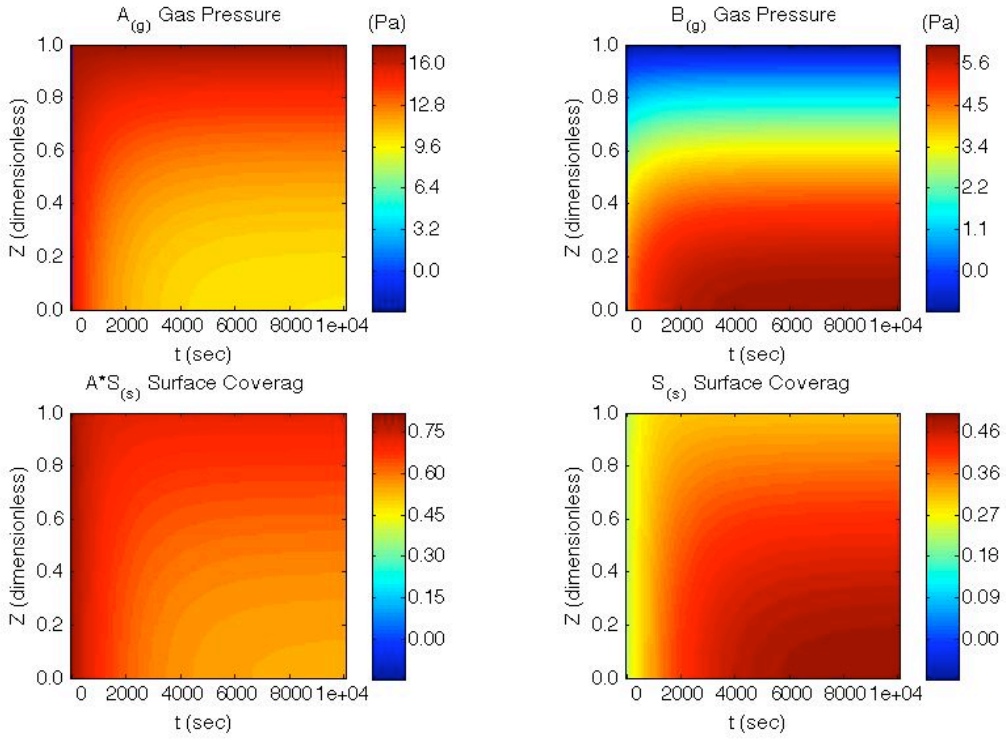

Figure 6.6 Gas concentration and surface coverage achieving steady state

An alternative method to derive the steady state solution is to solve the gas concentration iteratively by the shooting method of the nonlinear ordinary differential equations. The shooting method starts from an initial guess at the left boundary. The concentrations of the next node to the right are evaluated by the first derivative and second derivative of the node on the left. The surface coverage for solid components can be calculated by quadratic equations based on the gas concentrations [7,8]. The bisection method is used to iterate until the solution converges. The resulting equations are given as following and the solution of the shooting method is shown in figure 6.6.

$$
\begin{cases}
\Psi_{A,j+1} = \Psi_{A,j} + dZ\left(\left.\dfrac{d\Psi_A}{dZ}\right|_j\right), \left.\dfrac{d\Psi_A}{dZ}\right|_j = \left.\dfrac{d\Psi_A}{dZ}\right|_{j-1} + dZ\left[\tau_A\alpha\left(-k_1\Psi_{A,j}\Theta_{S,j} + k_2\Theta_{A,j}\right)\right] \\
\Psi_{B,j+1} = \Psi_{B,j} + dZ\left(\left.\dfrac{d\Psi_B}{dZ}\right|_j\right), \left.\dfrac{d\Psi_B}{dZ}\right|_j = \left.\dfrac{d\Psi_B}{dZ}\right|_{j-1} + dZ\left[\tau_B\alpha\left(k_3\Theta_{A,j}\Theta_{S,j}\right)\right] \\
\Theta_{A,j} = (k_1\Psi_{A,j} + k_2 + k_3 - \sqrt{\left(k_1 C^g_{A,j} + k_2 + k_3\right)^2 - 4k_1k_3\Psi_{A,j}}) / 2k_3 \\
\Theta_{S,j} = 1 - \Theta_{A,j}
\end{cases}
$$

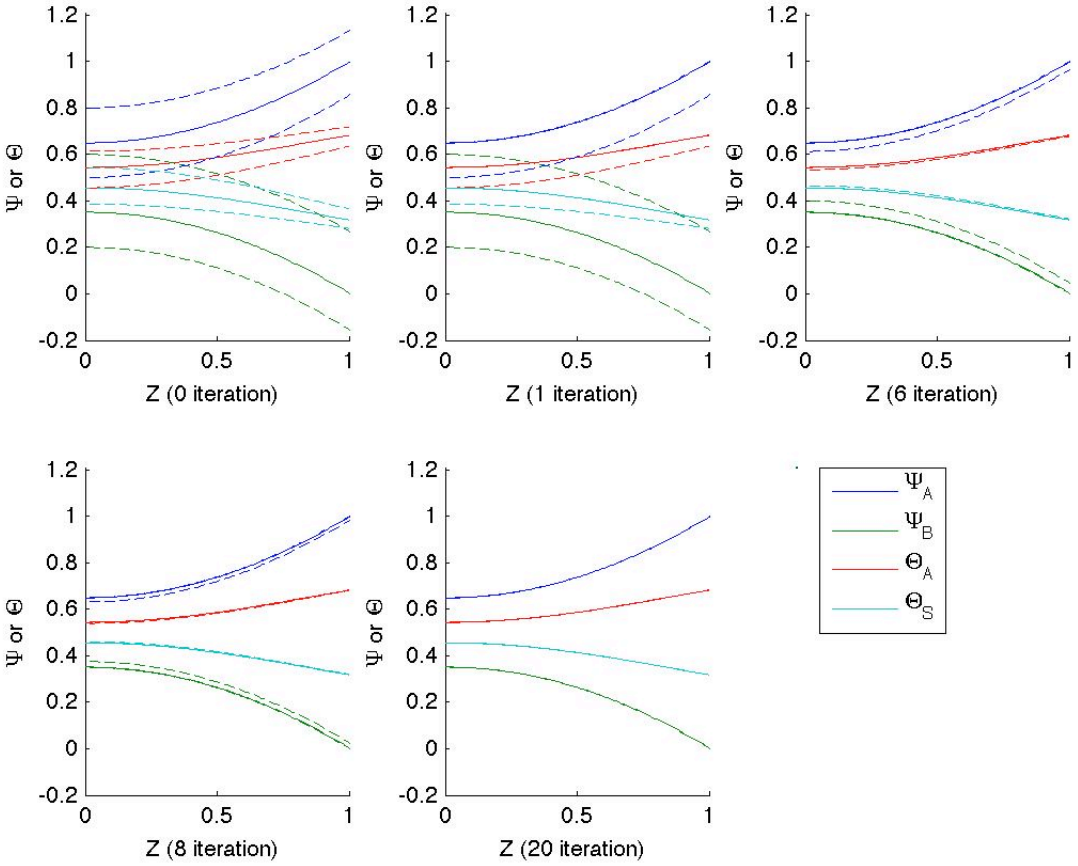

Figure 6.7 Comparing solutions between shooting method and Transcat

In figure 6.7, the solid lines are the solution computed by Transcat and the dashed lines are the upper and lower bound of gas concentration of shooting method. The solution of shooting method converges fast and the result show good agreement to the solution of Transcat after 20 iterations.

6.4 Fisher-Kolmogorov Equation

Fisher's equation also called Fisher-Kolmogorov equation is probably the simplest unsteady diffusion reaction equation where an analytical solution exists [1,2]. Fisher's equation and its solution is provided as following

$$\begin{cases} \dfrac{\partial C}{\partial t} = \dfrac{\partial^2 C}{\partial Z^2} + C(1-C) \\ C = \left[1 - A\exp\left(\dfrac{Z - ct}{\sqrt{6}}\right)\right]^{-2}, \; c = 5/\sqrt{6} \end{cases}$$

where c is the speed of traveling wave, and A is a constant determined by the initial condition. To model Fisher's equation in Transcat, both the diffusion time constant and surface-to-gas capacity ratio are set to one. The traveling wave solution computed by Transcat is shown in figure 6.9. The solution from Transcat matches the analytical solution and can maintain the same shape of traveling wave with increasing of time. Transcat is designed for modeling heterogeneous catalytic reactions, but from the computation result of Fisher's equation, it is actually capable of solving the unsteady diffusion-reaction equation in a homogeneous system under transient conditions.

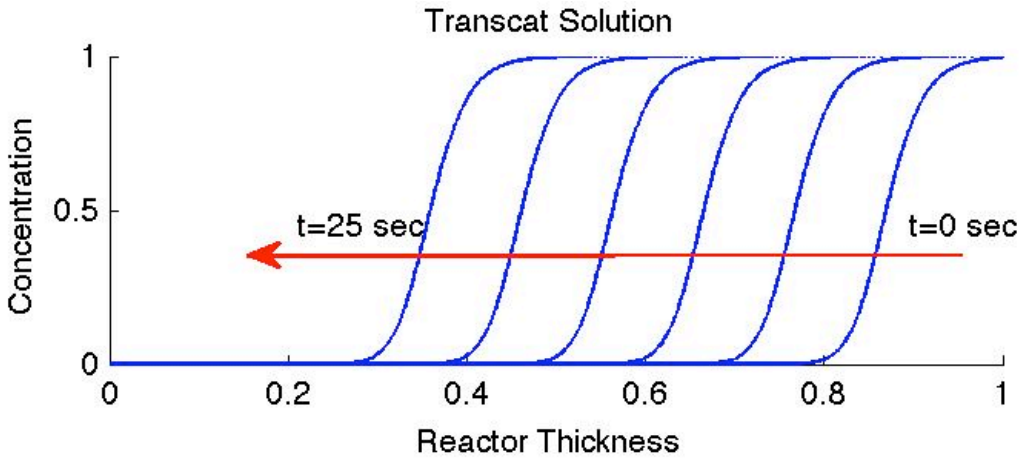

Figure 6.9 Wave propagation of Fisher-Kolmogorov equation computed by Transcat

To validate the Transcat, four cases have been used. The first case is, when setting the surface-to-gas capacity ratio small, the solution of Transcat conforms to the analytical solution of a unsteady diffusion equation. The second one is that the solution of steady-state linear adsorption computed from Transcat agrees the existing analytical solutions for a steady-state, first-order and essentially irreversible reaction by comparing the concentration profile, Thiele modulus and effectiveness factor. The third one is to compute the solution of a more complicated nonlinear adsorption and desorption mechanism by the shooting method, which exact by matches to the result from Transcat. The last one is that Transcat can model the propagation of the traveling wave of the homogeneous Fisher-Kolmogorov equation where an exact solution exists. In addition to the confirmation by theoretical modeling, Transcat also shows good fitting of experimental data of inert neon diffusion and carbon monoxide adsorption in a porous Pt/Al_2O_3 catalyst. Based on these case studies, it is concluded that Transcat can accurately model transient diffusion-reaction behavior in a porous catalyst.

Reference:

[1] Kaliappan, P. (1984). "An Exact Solution for Traveling Waves of UT=DUXX+U-UK." Physica D 11(3): 368-374.

[2] Zwillinger, D. (1998). Handbook of differential equations. San Diego, Calif., Academic Press.

[3] Kreyszig, E. (1999). Advanced engineering mathematics. New York, Wiley.

[4] Nett-Carrington, L. C. and R. K. Herz (2002). "Spatiotemporal patterns within a porous catalyst: dynamic carbon monoxide oxidation in a single-pellet reactor." Chemical Engineering Science 57(8): 1459-1474.

[5] Fogler, H. S. (2006). Elements of chemical reaction engineering. Upper Saddle River, NJ, Prentice Hall PTR.

[6] Levenspiel, O. (1999). Chemical reaction engineering. New York, Wiley: 1 online resource (xvi, 668 p.).

[7] Hairer, E., S. P. Nørsett, et al. (1993). Solving ordinary differential equations. Berlin ; New York, Springer-Verlag.

[8] Hoffman, J. D. (2001). Numerical methods for engineers and scientists. New York, Marcel Dekker.

Chapter 7 Stiffness Reduced Model of Heterogeneous Catalysis

As early in 1982, Oh and Hegedus developed the first dynamic model for the chemisorption process of carbon monoxide over a supported Pt catalyst in response to a step change of feed [1]. They discovered that even reducing ten-fold both the adsorption and desorption rate, the model still showed good agreement to the experimental data. They concluded that the adsorption and desorption rate were sufficiently fast, so the process rapidly reached equilibrium at all time. A mathematical model with this assumption of quasi-equilibrium adsorption of carbon monoxide oxidation developed in our group also shows good fitting of the experimental data and can reduce the computational cost as well [2].

An interesting question arises. Is it possible to use less stiff parameters to model an extremely stiff problem of dynamic heterogeneous catalysis but still obtain good agreement? If so, what are the criteria and how much computation time could be saved? Several runs are carried out base on parameter of fitting the dynamic response of CO adsorption experiment in previous chapter. The kinetics of CO adsorption on a bimodal distribution of active sites over Pt/Al_2O_3 is used, and the sticking probability of CO adsorption is used as a variable for a series of numerical experiments [2]. The adsorption rate of weaker activities site is a hundred times smaller than the stronger site. The maximum adsorption rate of CO over the stronger site with the sticking probability set as 1 over a Pt atom is

$$k_{CO}^{ads} = \sqrt{\frac{RT}{2\pi M_{CO}}}\sigma S_{CO}^{ads} = 7.78 \times 10^{6}\ m^{3} mol^{-1} s^{-1}$$

where R is the ideal gas constant, T is the temperature at 423.15 K, M_{CO} is the molecular weight of CO, σ is the area occupied by one mole of surface Pt atom (5.5×10^4 m^2mol^{-1}) and S is the sticking probability.

Ten runs are carried out in Transcat with the sticking probability reduced by 10 times in each run and the desorption rate coefficient is also reduced by the same order of magnitude. The diffusion time constant is 0.175 sec^{-1}, the surface-to-gas capacity ratio is 1.52×10^2 and both of them are held as constants in all these runs. The normalized maximum adsorption rate by the feed gas pressure of 225 Pascal is 4.92×10^5 sec^{-1}. The ratio of adsorption to desorption rate is kept at 0.554 for both active sites. The results of gas concentration are plotted in figure 7.1. The CO gas concentration on the boundary subject to the step change of gas feed is not shown in the plot. The gas concentration profiles on the other side of boundary are plotted and can be sorted into three categories.

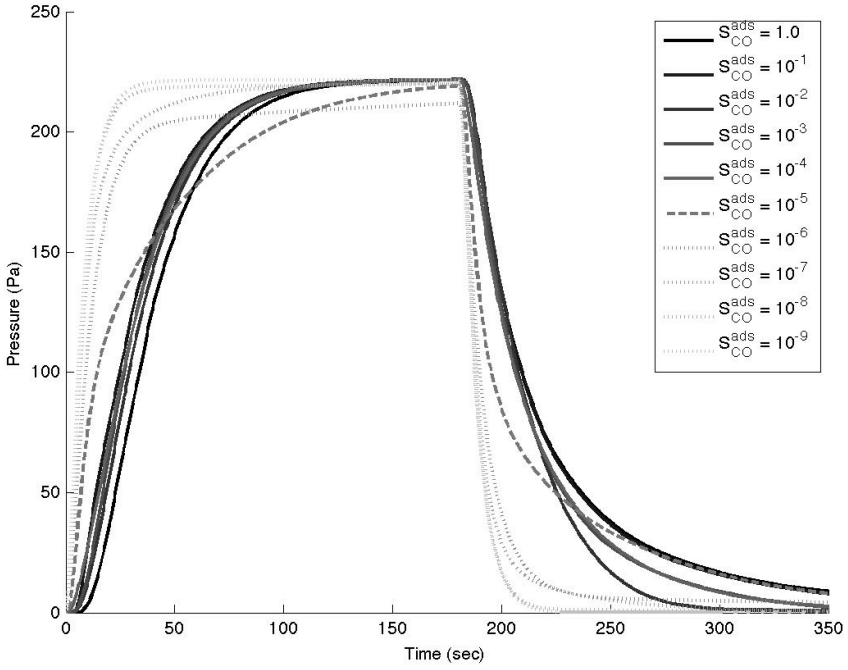

Figure 7.1 Carbon monoxide gas concentration of stiffness reduced model

The first category in figure 7.1 is for the simulations with large sticking probability, which is between 1 and 10^{-4}. The gas concentrations of these runs are the darker solid lines shown in the plot. The penetration of CO gas through the reactor is slower than the runs with smaller sticking probability. When the sticking probability is larger than 10^{-4} the gas concentration shows good agreement with the experimental data. The desorption rate in each of these cases can be further slightly adjusted to get a better fit to each other. The second category is the grey dashed line with sticking probability of 10^{-5}, which shows moderate penetration of CO gas through the reactor. The runs with fast response of both adsorption and desorption curves are the third category where the sticking probability is equal to or less than 10^{-6}. It is clear that when the sticking probability is large enough, the interaction of feed CO gas with the active sites on the surface is much stronger than gas transport between the two boundaries. The runs with large sticking probability suggest that the adsorption-desorption process is close to equilibrium as long as the equilibrium constants are kept to the same, fixed adsorption-to-desorption rate ratio in these cases. The spatiotemporal pattern of gas pressure and surface coverage of CO on both active sites are shown in figure 7.2. The spatiotemporal patterns of simulations whose sticking probability are larger than 10^{-4} are very similar, the data used in figure 7.2 is the sticking probability of 1.

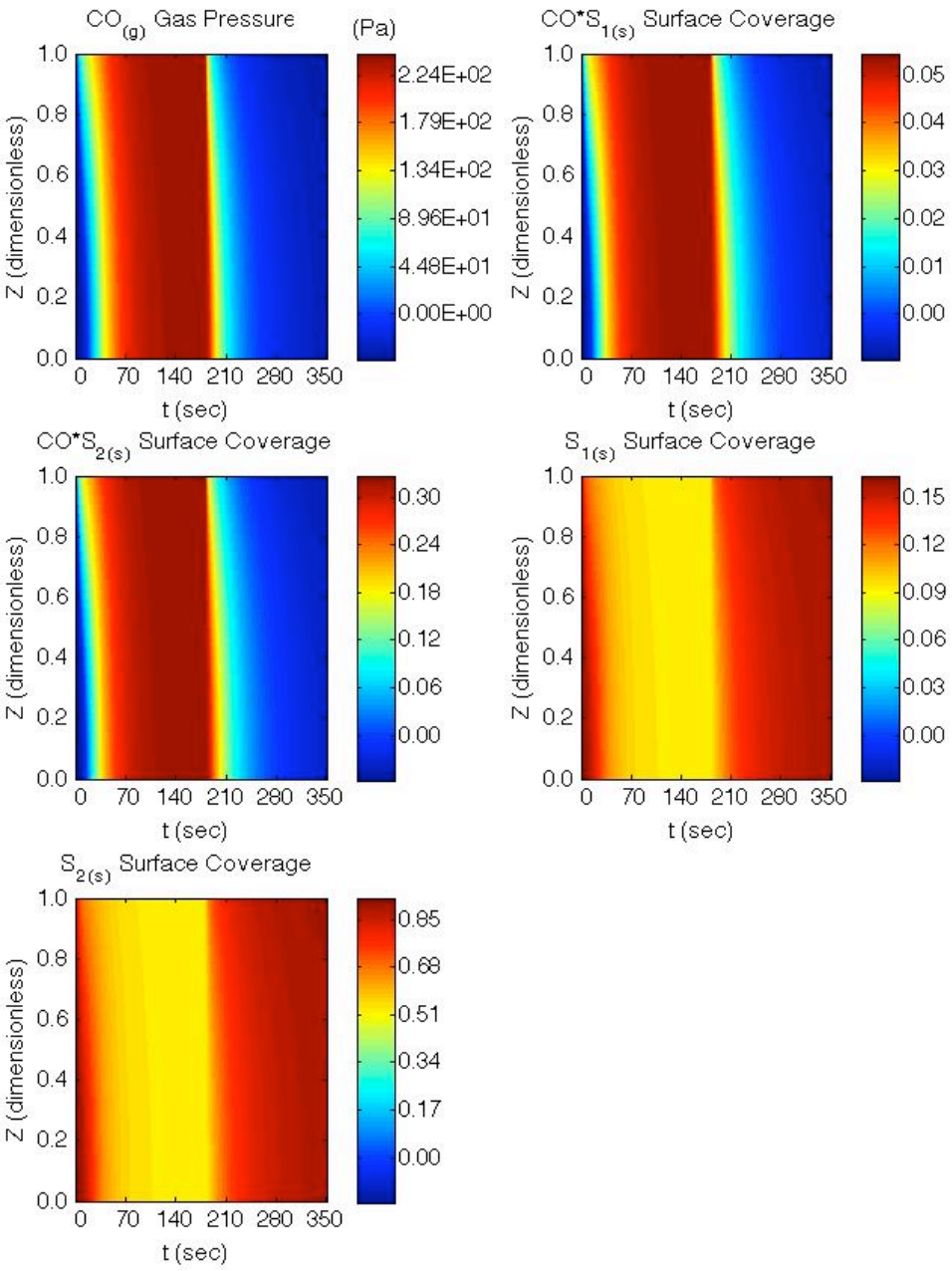

Figure 7.2 Spatiotemporal pattern of gas pressure and surface of CO adsorption-desorption process to step change in a dynamic diffusion reactor

When the sticking probability drops to 10^{-5}, the adsorption and desorption curves gradually shift to the diffusion dominated region. When it goes down further, the model can no longer represent this dynamic adsorption process, since the reaction rate constant

is around the same order of magnitude, or less, than the diffusion time constant. The reciprocal of adsorption rate becomes 0.203 sec when the sticking probability is 10^{-5}, compared to the diffusion time constant 0.175 sec. Another set of numerical simulations are carried out by using larger or smaller gas-to-surface capacity ratio, but the results didn't show any apparent change in the concentration profile. The only difference is the total time required for the surface to become saturated or depleted.

Most solid catalysts are made porous to acquire large active surface area, but the porous structure, on the other hand, slows down the gas transport process. The fast catalytic reaction and slow pore diffusion make the numerical modeling difficult since it usually result in stiff partial differential equations. Simulating the operation of a porous catalyst under dynamic conditions is even harder due to the rigorous, unsteady boundary conditions. Figure 7.3 shows a simple adsorption-desorption process of CO over Pt/Al_2O_3 catalyst with realistic rate constants. It can take up to eight hours simulation on a computer with Intel® 3.16 GHz Core 2 Duo processor. The computing time can be reduced by using smaller rate constants based on the quasi-equilibrium assumption. As long as the rate constants are still much larger than the diffusion time constant, a less stiff sets of parameter can be used in the simulation and computing cost can drop dramatically. When the sticking probability is on the borderline of stiffness or the condition is no longer stiff at all, reducing the rate constants does not improve the computing cost effectively. The adaptive time control of Transcat is also inspected in figure 7.3. While the CO feed is switching on and off at the boundary, Transcat adjusts the size of the time step adaptively according to the dynamic condition in the reactor.

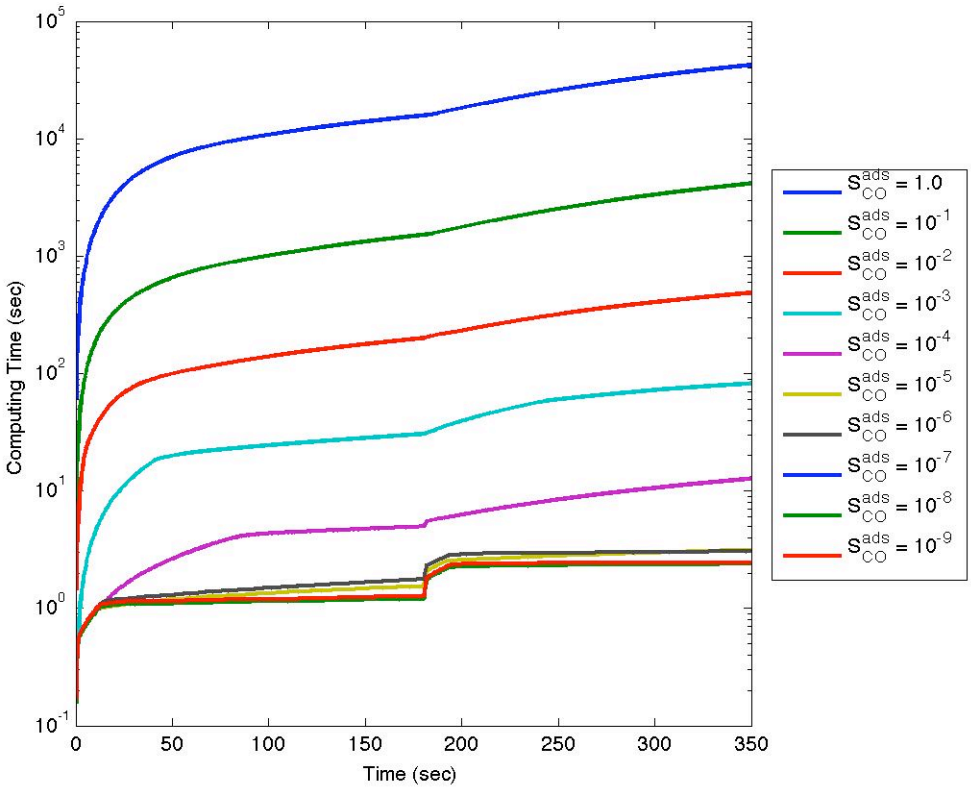

Figure 7.3 Computing time of CO adsorption-desorption process with various coefficient of sticking probability

Quasi-equilibrium may exist in a slow-mixing reactor like the a porous catalyst under dynamic conditions. Several criteria are found based on the study of CO adsorption. There must be fairly large quantity of surface sites for the gas to interact with, so the empty sites are filled, or loaded sites consumed, gradually. The rate constants of adsorption, desorption and surface reaction are much larger than the diffusion time constants. The process reaches the state of partial-equilibrium rapidly; a fluctuation or even a step change on the boundary does not affect equilibrium much. When the rate constant is decreased or increase at constant net forward and reverse rate, the spatiotemporal pattern for both gas pressure and surface coverage are similar in different

runs. The thermal effect is not studied in this case, so the possibility of equilibrium shift by temperature variance is not considered here.

Another study of more complicated surface kinetics of Langmuir-Hinshelwood-Hougen-Watson (LHHW) carbon monoxide oxidation supports the existence of quasi-equilibrium in a transient and spatial distributed catalysts [3]. Figure 7.4, 7.5 and 7.6 are the gas concentration and surface coverage of LHHW carbon monoxide oxidation at three instances, where the time is 25, 50 and 150 respectively. A similar pattern of surface coverage is found in all these three figures, but in different spatial location in the catalyst within the area of the orange block with dashed boarder. This is a hot zone where competing adsorption, desorption and surface reaction can be found. This zone moves from right to the left with increasing of time. This is a type of equilibrium where the dynamics of diffusion and reaction does not change the pattern of this hot zone and where the zone only moves to different location in the catalyst at different times. The gas concentration may be affected by the position of this hot zone, but the reaction rate is not. On the left of this hot zone, the catalyst is almost inactive since no carbon monoxide can penetrate this far and no net adsorption of oxygen occurs due to saturation. On the right side of this hot zone is equilibrium adsorption and desorption of carbon monoxide, since it is a much stronger adsorbate than oxygen. Only a small amount of oxygen adsorbs in this area and it reacts right away. The repeating pattern of the hot zone, inactive zone of weak adsorbate, and equilibrium zone of strong adsorbate imply the existence of quasi-equilibrium in a spatially distributed reactor under transient conditions.

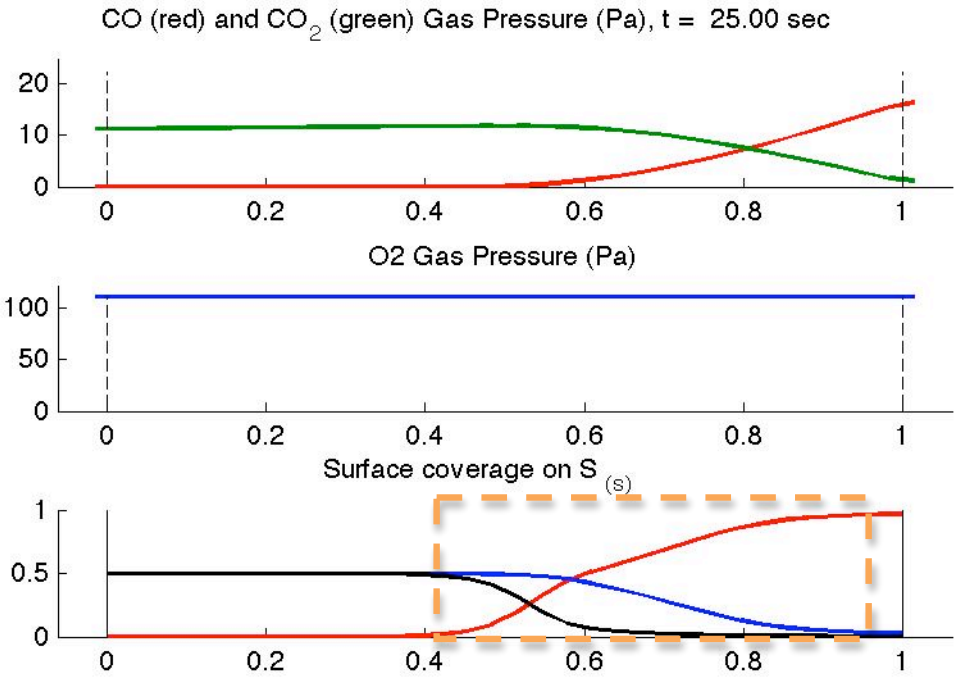

Figure 7.4 Gas concentration and surface coverage of LHHW CO oxidation (t=25 sec)

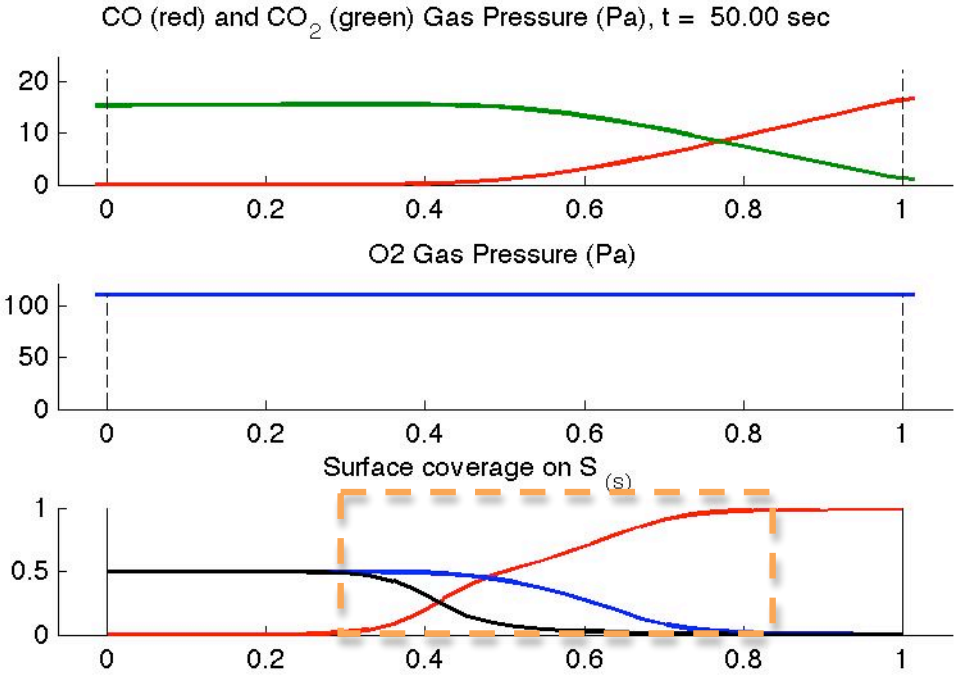

Figure 7.5 Gas concentration and surface coverage of LHHW CO oxidation (t=50 sec)

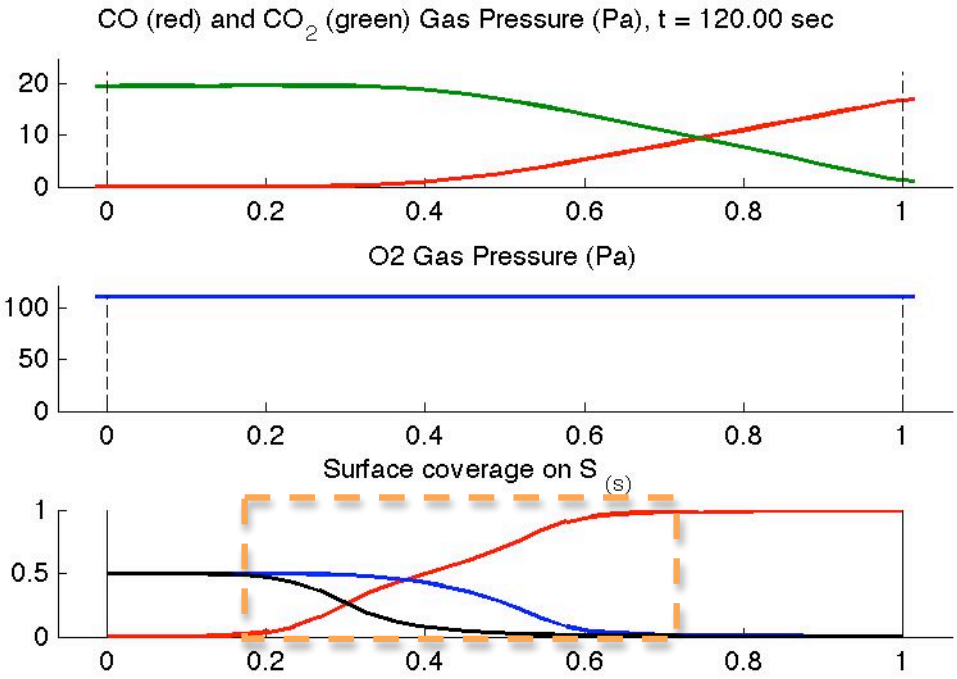

Figure 7.6 Gas concentration and surface coverage of LHHW CO oxidation (t=120 sec)

When using a general solver like Transcat to model an extremely stiff problem under transient operation conditions, a set of smaller rate constants, yet still much larger than the diffusion time constant, can be used attempt to fit the data first. If the result shows good agreement, then increase the stiffness by raising the rate constant of the fastest rate in the kinetics and adjust other rate constant to keep the forward and reverse rate the same for the intermediate. Run the simulation again and the cost of computing should increase roughly proportional to the magnitude of the increase in the rate constant. If the result is almost the same as the previous less stiff simulation, the process is in close approach to equilibrium. Although not exactly the same, the model with less stiff set of rate constants is good enough to model the problem efficiently.

Reference:

[1] Oh, S. H. and L. L. Hegedus (1982). "Dynamics of High-Temperature Carbon-Monoxide Chemisorption on Platinum Alumina by Fast-Response IR Spectroscopy." Acs Symposium Series 178: 79-103.

[2] Nett-Carrington, L. C. and R. K. Herz (2002). "Spatiotemporal patterns within a porous catalyst: dynamic carbon monoxide oxidation in a single-pellet reactor." Chemical Engineering Science 57(8): 1459-1474.

[3] Herz, R. K. and S. P. Marin (1980). "Surface-chemistry Models of Carbon-Monoxide Oxidation on Supported Platinum Catalysts." Journal of Catalysis 65(2): 281-296.

Chapter 8 Composition Modulation of Carbon Monoxide Oxidation

Composition modulation is a temporal contacting pattern, which prevents the heterogeneous catalytic process from attaining steady state in order to improve the performance of a reactor [1]. The contacting patterns include the operating strategy of the concentration, temperature, mixing, or any transient technique that can enhance the selectivity, yield or reaction rate. Carrying out experiments of all possible contacting patterns to optimize a process is expensive and time consuming. If the performance of a heterogeneous catalyst under composition modulation can be accurately modeled on a computer, it would be a great saving in both time and money. Because of the comprehensive capability of Transcat, it can predict the performance of any surface kinetics under composition modulation, which is also the most valuable part of this research work. The theory of stiffness-reduced model of heterogeneous catalysis can make simulation even more efficient. In this chapter, several cases are studied to show how to approach optimized reactor performance by engineering the composition modulation strategy for a heterogeneous catalytic process in Transcat.

8.1 Improvement of Reactor Performance by Composition Modulation

Carbon monoxide (CO) oxidation over Pt/Al_2O_3 catalyst is the most studied catalytic process in our group. Early works have been done to understand the behavior of this process under transient operation conditions [2,3]. Higher transient generation rates of carbon dioxide (CO_2) are found when the feed of CO is switched off, and a model of bimodal distribution of active sites is used to explain this phenomenon [2]. Since

Transcat can handle arbitrary kinetics; the transient response of both single and bimodal active sites can be checked again by Transcat. Figure 8.1 is the curve fitting to experimental data of CO oxidation at 423K under a period of on-and-off operation of the CO feed [2]. The plot on the left is fitting by the model of a single active site and the other on the right is by the model of bimodal active sites. The experimental data shows that the gas pressure of CO_2 is suddenly increased, which forms a peak in the profile, when one of the reactants, CO, is turned off. This is a role model in which the conversion of reactant can be improved by composition modulation. It is reasonable that the model of bimodal active sites fits better to the experimental data since CO can adsorb on multiple sites of Pt [2,4]. Although the fitting to the curve is not exact, the purpose is to explain why there is peak formation. To explain why the step change of input CO can cause the large quantity of CO_2 production, the single site model is actually more clearly in illustrating this phenomenon.

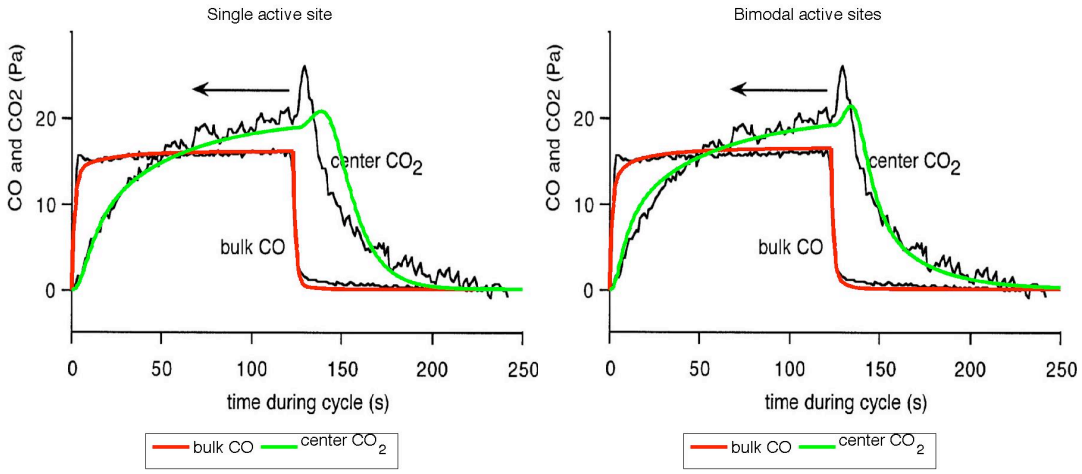

Figure 8.1 Fitting experimental data of carbon monoxide oxidation at 423K. Experiment data are from [2].

The whole operation process can be explained by the reaction rate and the dynamic gas pressure and surface coverage in the reactor of the single site model, which

are shown in Figure 8.2 and Figure 8.3 respectively. If we look closely to the dynamic behavior of the distribution of reaction rates, we can see a clear storage process when the CO is turned on and a burn off process when it is turned off. Initially the CO starts to penetrate the catalyst and react with stored oxygen, but oxygen is a weak adsorbate, which cannot compete with CO for catalytic sites. The hot zone of reaction further penetrates and moves towards the other side of the catalyst and also stores large quantity of CO over the surface wherever it passes. When the CO is turned off, there are more empty catalytic sites available for the oxygen (O_2) to adhere, which then starts to react with the stored CO and clear out more empty sites. As a result, the root cause of the peak formation in the CO_2 gas profile is this burn off process, which suddenly consumes the entire stored CO until is depleted.

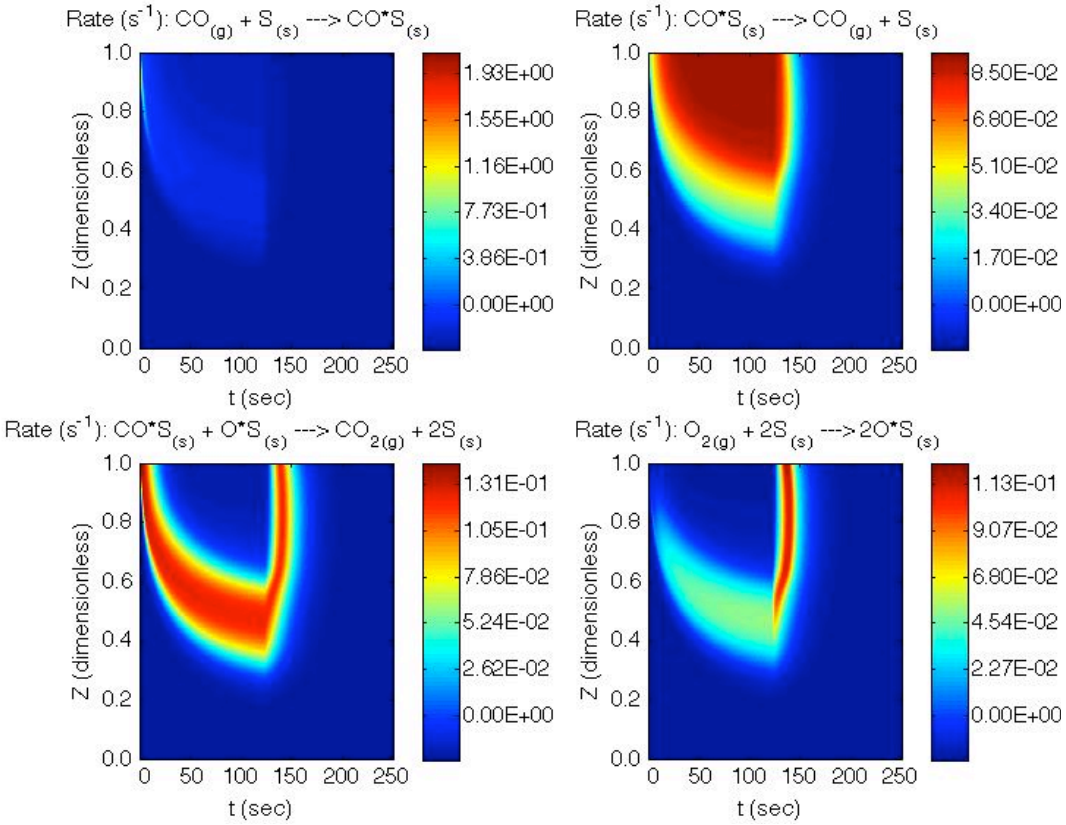

Figure 8.2 Adsorption, desorption and reaction rate of single site model of CO oxidation

The storage and burn off process is even more clearly seen on a series of plots of dynamic gas pressure and surface coverage with increasing time, which are shown in figure 8.3. When the CO is turned on, the reaction zone starts to penetrate the reactor and is followed by the storage zone. Once the CO is switched off, both the reaction zone and storage zone start to consume all stored CO. This broad burn off zone makes the sudden increase in the CO_2 gas concentration and generates the peak. Based on the simulation result from Transcat, the relative strength of the adsorbate to the active site causes the storage of surface species on the catalyst and the change of the contacting pattern causes the burn off process. The combination of storage and burn off process increase the overall conversion.

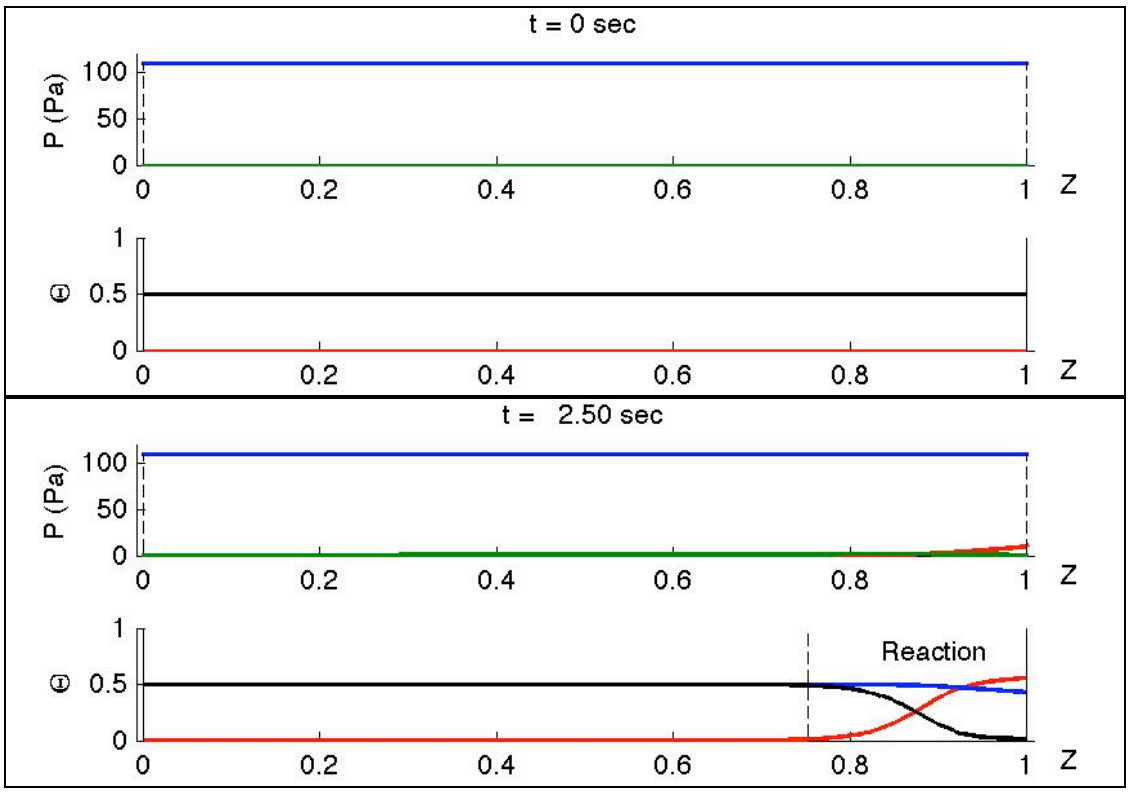

Figure 8.3 Dynamic gas pressure and surface coverage of single site model of CO oxidation

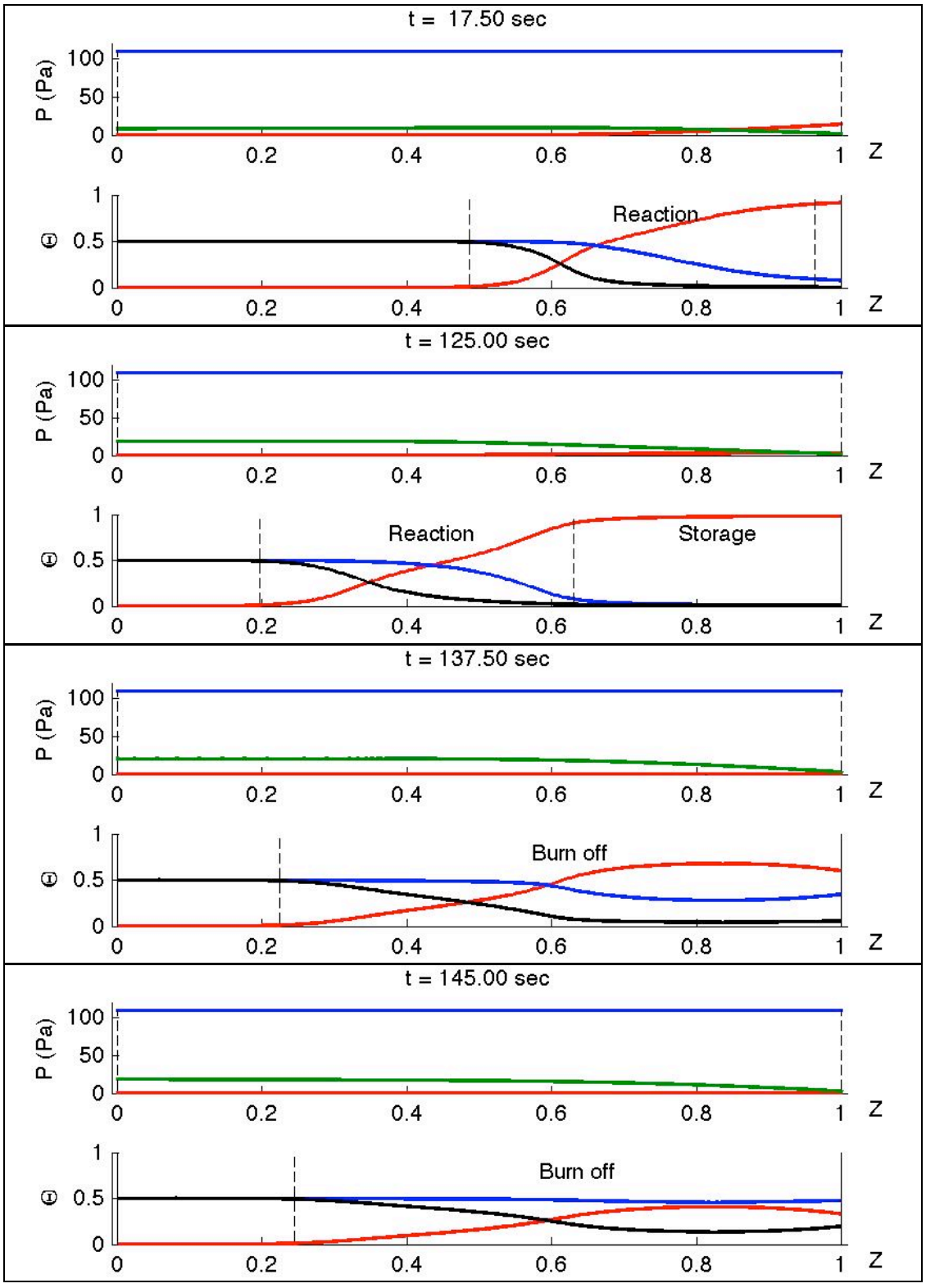

Figure 8.3 Continued

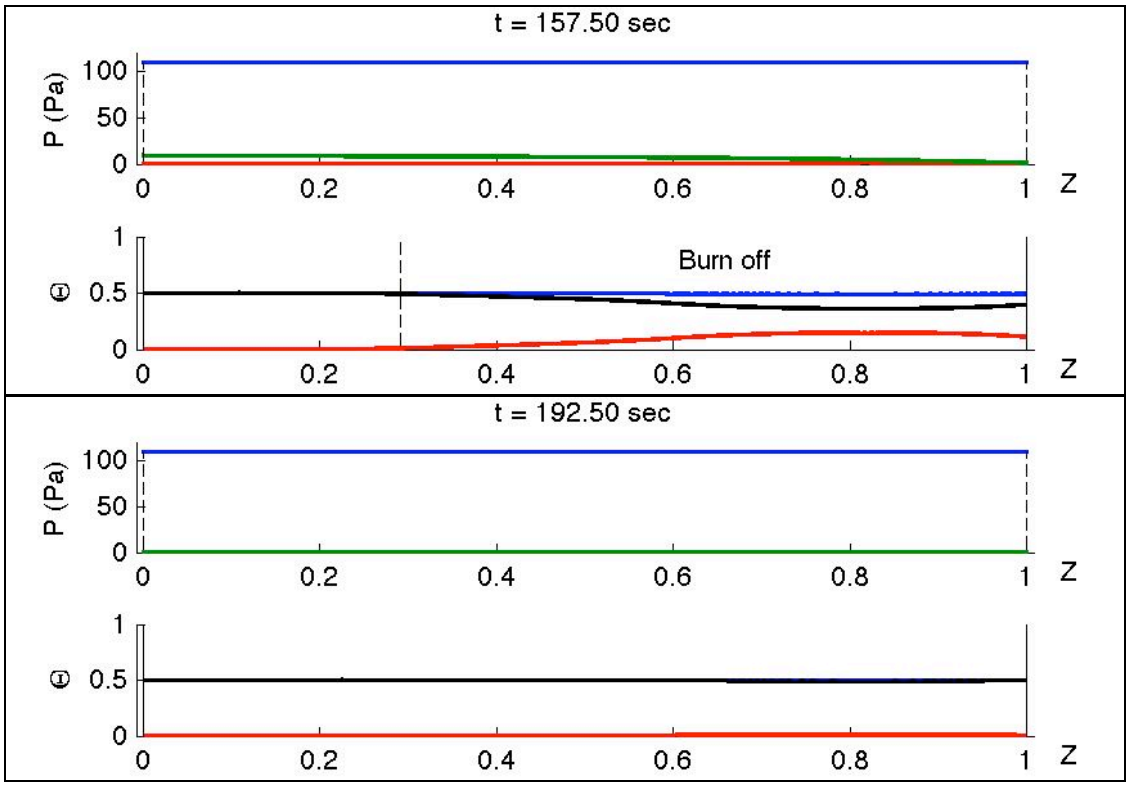

Figure 8.3 Continued

A more generalized explanation for the conversion improved by composition modulation is that the temporal contacting pattern can increase the spatial usage of the catalyst. Under steady operation, the reaction is confined in a partial region of the catalyst due to the diffusion limitation and surface kinetics. Outside this region, part of the feed is stored rather than reacted and the selection of storage is determined by the relative strength of adsorption and desorption rate of adsorbate. When the temporal contacting pattern is changed, the stored species on the surface start to react and enhance the spatial usage of catalyst, so the overall conversion can be improved. To design a temporal contacting pattern, the concept is try to convert as much feed as possible in either reaction or storage on the surface in one period, and then use the stored species in another. In the rest of this chapter, several approaches to optimize the operating condition of

heterogeneous catalytic processes are illustrated by the numerical simulations from Transcat.

8.2 Frequency Optimization of Carbon Monoxide Oxidation

Transcat has the capability of predicting catalyst performance under different operating conditions, where only limited information available. In a previous section, Transcat performed the ability to fit the experimental data of carbon monoxide oxidation at 423K, and explained the response of a catalyst under a complete cycle of on-and-off CO modulation. Assume this is the only experimental data available, and we want to design an optimal temporal contacting strategy to maximize the conversion of CO by computer simulation. A straightforward method under this circumstance is to find the optimum operation frequency of this square input signal of CO.

To carry out a series of numerical simulation in Transcat, a user only needs to edit the batch file and use the same parameters as in the curve-fitting run except for the CO input frequency. Transcat will execute these runs with varying frequency in a single batch, and the user only needs to harvest the data by means of the visualization toolbox to see results. Figure 8.4 is a plot of the turnover rate of CO_2, time averaged generation rate of CO_2 per unit site per second, to the CO modulation frequency of the square wave. Some patterns of turnover rate are also shown on this plot. The time averaged turnover rate of CO_2 goes through a maximum with varying modulation frequency of CO. The operation under constant CO feed is also shown in this plot to compare to the result of cycling simulations. The turnover rate of steady feed is 0.0331 s^{-1}, which is lower than that of the modulation frequency between 0.2 s^{-1} and 0.02 s^{-1}. The optimum frequency shown in the

plot is 0.04 s^{-1}, whose turnover rate is 0.0413 s^{-1}.

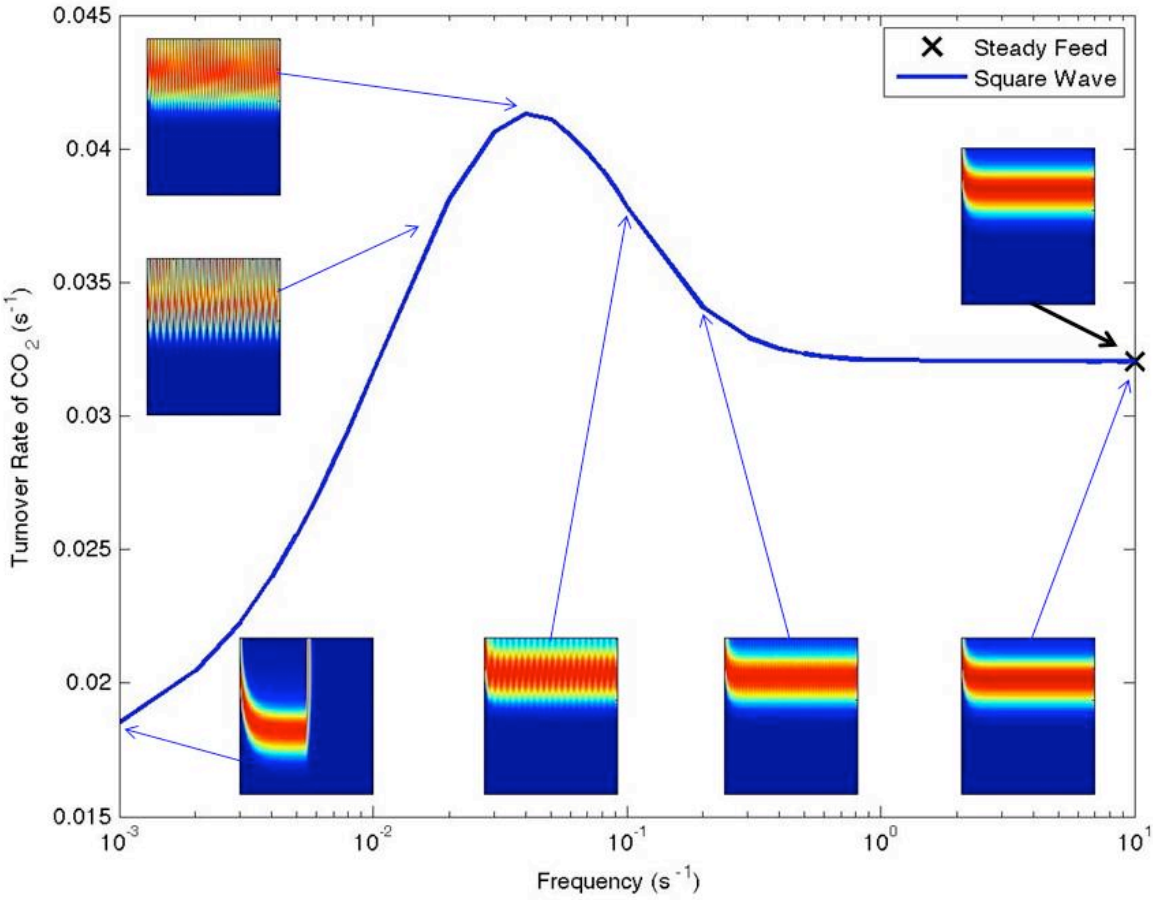

Figure 8.4 Turnover rate of CO_2 vs. the square wave CO modulation frequency

Before comparing the pattern of the turnover rate, we have to define the two legs, or periods within a cycle, shown in these images. One is the advance leg which represent the reaction penetrating into the catalyst and storing surface species when the CO is turned on. The other is the recess leg, which is the burn off process when the CO is turned off. When the CO modulation frequency is fast, only the advance leg is found. The pattern of the turnover rate is similar to the pattern of steady feed, since there is not enough time for the reactor to respond, that is, not enough time to burn off the stored CO on the surface. When the modulation is slow, both the advance and recess leg are found.

The recess leg only appears in a short period and there is a gap in reaction rate before proceeding another cycle, so the conversion is low. When the process is operated at optimum frequency (0.04 s^{-1}), a signature pattern is found compared to a higher and lower frequency and is show in the middle image of Figure 8.5.

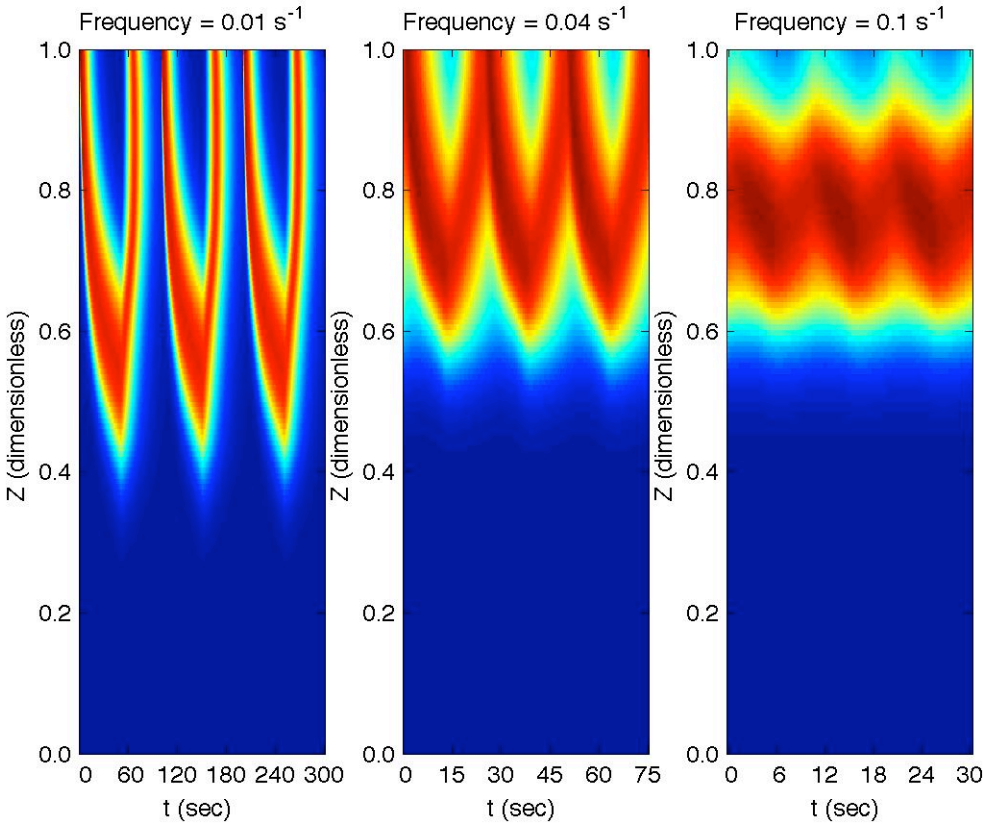

Figure 8.5 Pattern of turnover rate comparing to a higher and lower frequency

The pattern at optimum frequency shows three characteristics. The first one is that the advance leg and recess leg are connected between cycles. Comparing to a lower frequency on the left of which the legs are disconnected, the disconnection means an unnecessary wait for the next cycle. The second one is that both legs are attached to the edge of the reactor boundary. This implies no saturated storage of surface species, so the reaction zone starts from the edge of reactor, not anywhere inside the reactor. The plot on

the right shows that higher frequency detaches both legs and pushes the reaction zone further into the catalyst. The last characteristic is that the shape of the advance leg and the recess leg are close to symmetry in a single cycle. This means the reaction is operated around a stoichiometric ratio of surface species where the maximum conversion is reached. Base on the inspection of the simulation result, the spatiotemporal pattern of turnover rate at optimum frequency for the on-and-off cycling of CO is to see a continuous, symmetric and boundary-attached reaction zone.

Another point of view to the optimum frequency is to examing the storage of surface species. Figure 8.6 is the profile of gas concentration and surface coverage right before switching off the CO feed at three different frequencies. The first one is operated at lower frequency than the optimum. At this frequency, CO penetrates deeper into the reactor and forms a reaction zone and a large storage zone. At a higher frequency than the optimum, the storage zone is smaller and the reaction zone is about the same as the lower frequency case. At optimum frequency, there is no apparent storage and the reaction zone is the largest of these three. The common feature between the two non-optimum cases is that both of them have saturated storage of CO near the boundary. However, for the case of optimum frequency, there is no saturated CO surface coverage on the catalytic sites near the boundary. Thus, we can conclude that the pattern of surface coverage to achieve the highest turnover rate under composition modulation is that a frequency can maximize the unsaturated and also minimize the saturated CO coverage on the surface near the external boundary.

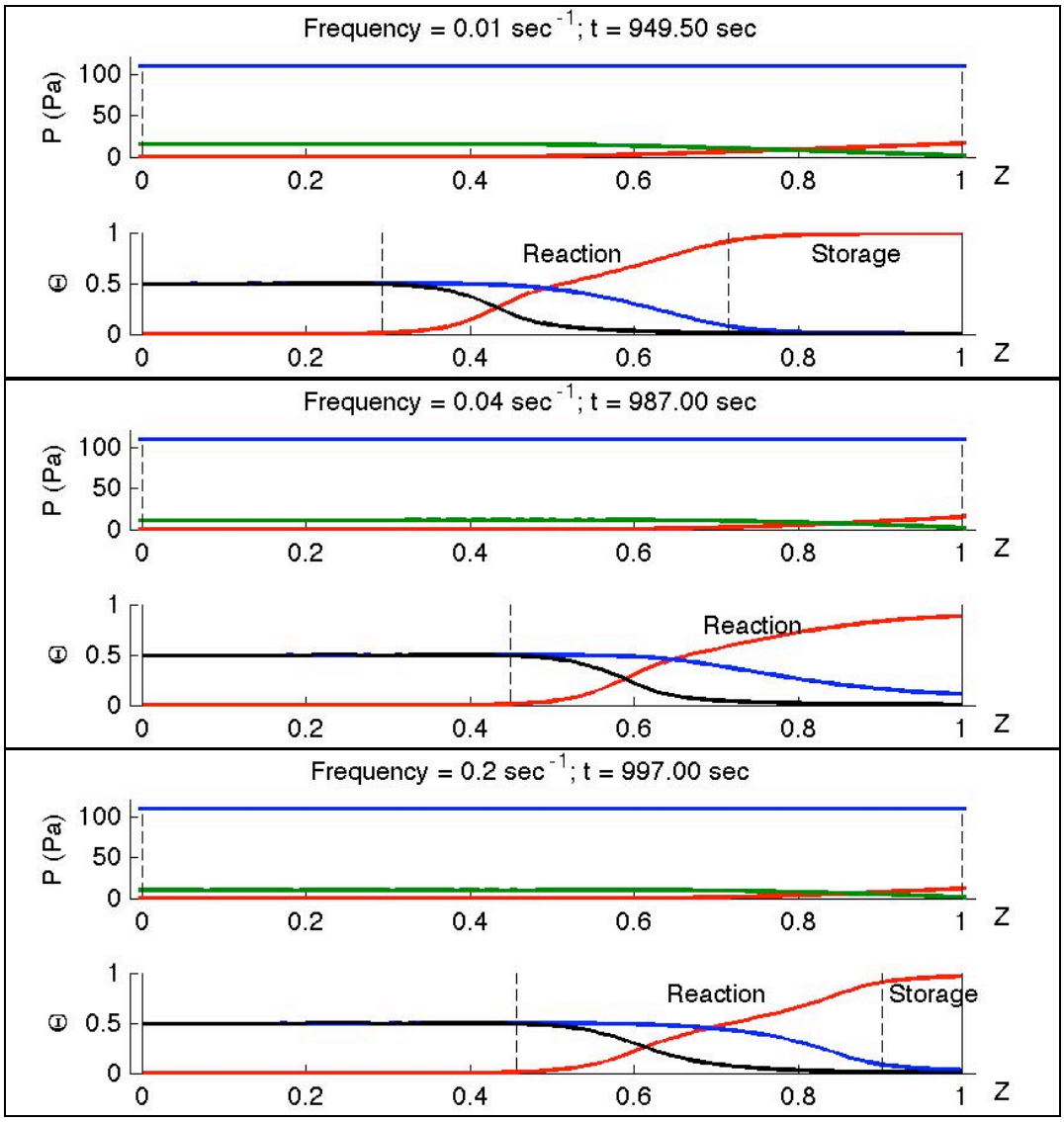

Figure 8.6 Gas pressure and surface coverage before switch off carbon monoxide. The CO gas pressure is modulated at the left boundary at Z=1.

Transcat can optimize the modulation frequency of a heterogeneous catalytic process with only limited information, which make it a great tool in improving the performance a reactor. The study of the spatiotemporal pattern of turnover rate and surface coverage is specific but not limited to LHHW CO oxidation. At optimum frequency, the spatiotemporal pattern of turnover rate is continuous, symmetric and boundary-attached, and the storage of saturated surface species out side the reaction zone

is at minimum.

8.3 Other Strategies of Composition Modulation

The strategy of composition modulation in heterogeneous catalysis is a broad research field. Any variable that can be manipulated in periodic forcing is a candidate for modulation. In previous sections, only composition modulation in a square wave has been tried to optimize the catalyst performance. There are numerous variables could be attempt in Transcat to further optimize this process. These variables include waveform, temperature, phase shift and stoichiometric ratio for modulation of single or multiple components.

Another series of numerical experiments for CO oxidation were carried out in Transcat by using various shapes of CO input signal. Two extra waveforms, sine wave and sawtooth wave are used, and the results of CO_2 turnover rates are shown in figure 8.7. The curve of sine wave and sawtooth wave show similar trends as the square wave discussed in the previous section. Each of these input signals can give higher turnover rates of CO_2 than steady operation in a range of frequency cycling. The length of penetration in the catalyst is similar for all these waveforms, and the spatiotemporal pattern at optimum frequency agrees with the conclusions in the previous section. The pattern of turnover rate of the sawtooth wave is loosely attached to the boundary, which means a storage zone on the surface, but the reaction is stronger than the others due to more area in the darkest red. The patterns of all these waveforms are continuous between cycles and close to symmetric geometry, even for the unsymmetrical sawtooth and square input signals. The optimization attempt is not as successful since the input signal that

generates the highest turnover rate is still the square wave. However, the spatiotemporal pattern at optimum frequency confirms the conclusions from the previous section.

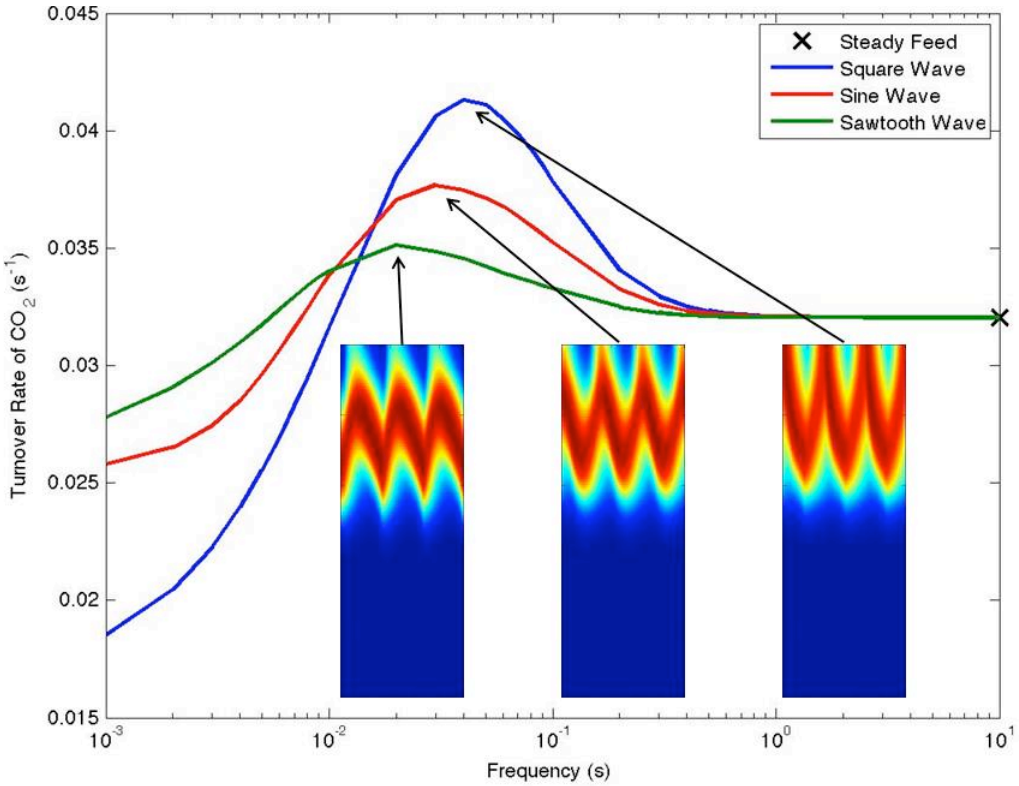

Figure 8.7 Turnover rate of CO_2 to the CO modulation frequency of square wave, sine wave and sawtooth wave

Reference:

[1] Silveston, P. L. (1998). "Composition modulation of catalytic reactors." Amsterdam, Gordon and Breach Science.

[2] Nett-Carrington, L. C. and R. K. Herz (2002). "Spatiotemporal patterns within a porous catalyst: dynamic carbon monoxide oxidation in a single-pellet reactor." Chemical Engineering Science 57(8): 1459-1474.

[3] Herz, R. K. (2004). "Spatiotemporal patterns in a porous catalyst during light-off and quenching of carbon monoxide oxidation." Chemical Engineering Science 59(19): 3983-3991.

[4] Masel, R. I. (1996). Principles of adsorption and reaction on solid surfaces. New York, Wiley.

Chapter 9 Conclusion

Transcat is general solver, which can model any heterogeneous catalytic process in a porous catalyst under transient operating conditions. The comprehensive capabilities of Transcat make it a useful tool to model, explain and predict the behavior of any surface kinetics. This numerical simulator is capable of handling unlimited components, arbitrary kinetics and various boundary conditions, so it can compute almost any unsteady diffusion reaction kinetics in porous catalyst. An optional algorithm of element constrain method is built in this simulator to guaranteed mass and element conservation during numerical computation. Transcat can complete computation tasks efficiently by built-in multiple adaptive numerical methods and the theory of stiffness-reduced modeling. Transcat is validated by conforming the simulation result to partial equilibrium, steady state and analytical solutions and is able to fit experimental data to ensure the correctness of a modeling problem. Transcat can perform studies in frequency, waveform and other contacting pattern of CO oxidation under periodic forcing to optimize the turnover rate of CO_2. When the process is operated at optimum frequency, the spatiotemporal pattern of turnover rate is continuous, symmetric and boundary-attached and the saturated storage is at minimum.

Several research groups have developed a numerical simulator for modeling the heterogeneous catalysis in porous catalyst, but the Transcat is the only solver that considers diffusion resistance and can handle arbitrary kinetics in a porous catalyst. The algorithm of computing the mocrokinetics is composed of complex sparse matrix operation and can even take derivatives automatically for the Jacobian matrix in Newton's method. The built-in capability of multiple boundary conditions in Transcat

makes it capable of modeling any porous catalysts that can be mathematical computed by one-dimensional unsteady diffusion-reaction equations. By manipulating the parameters, Transcat, although not designed to, can also compute a homogeneous unsteady diffusion-reaction problem like the Fisher-Kolmogorov equation.

The element constraint method (ECM) is an original algorithm in this research work to ensure mass and element conservation during numerical integration. The possibility of error propagation in total elements is studied in partial equilibrium, steady-state and numerical approximations under transient conditions. Although the efficiency in total computing time can only improved by 10% at most for some numerical methods, the ECM has been successfully integrated into both explicit and implicit methods. The structure of the Jacobian matrix is critical in solving the iterative Newton's method together with an element constraint algorithm. Pivoting the Jacobian matrix could reduce the computational costs, which change its structure derived from an unsteady diffusion reaction equation.

The current version of Transcat is separated into several pieces. The main program is developed in Fortran90, the input and output files are written in either text or extensible markup language (XML) files and the post-processing and visualization tools are built in Matlab. A better graphic user interface should be developed in the future to integrate all these pieces and make Transcat easier to use.

Transcat is also carefully validated by several case studies, and it is also capable of fitting experimental data, so the user should be confident in the simulation results. The expandability of Transcat makes it capable of integrating the algorithm of solving the energy equation and newly developed numerical algorithms in the future.

The stiffness-reduced modeling is a whole new idea of modeling gas-solid kinetics based on the assumption of partial equilibrium assumption and moving reaction zones in different partitions of the catalyst. Although only the CO adsorption and oxidation kinetics are studied, this idea should also work in other kinetics where the diffusion time constant is much smaller than the reaction rate constants. This modeling technique can turn a super stiff problem, where simulation is almost infeasible, into a practical problem. Unlike traditional partial equilibrium approximations or pseudo steady state approximations, the advantage is that derivation of equation is not required.

Transcat can optimize a process with only limited information, which makes it a valuable tool in predicting and optimizing a heterogeneous catalytic process under transient operating conditions. Most composition modulation optimization is based on experiments and empirical rules. Transcat can further explain the behavior of catalytic reactions under periodic forcing based on scientific premises. The study in the frequency and waveform modulation of CO oxidation gives a good example in finding spatiotemporal patterns of reaction rate and surface coverage at optimum operating conditions. More variables can be checked in the future to get a general idea of how to improve a catalytic process by composition modulation.

Appendix A Consistency and Accuracy Analysis of the Second Derivative on Non-uniformed Grid

The one-dimensional finite volume method generates non-uniformed mesh on the boundary. The finite difference formulae of derivates must be redefined, so the accuracy in azimuthally direction has the same magnitude of order comparing to uniformed grid. The accuracy and consistency analysis of differential formulae on non-uniformed grid is provided as following. Assume the coordinates of dimensionless grids are z_j, where $j=1,2,3,\ldots\ldots N_z$. The corresponding concentration is $C_j=C(z_j)$, and $\Delta z_j=,z_{j+1}-z_j$ where $j=1,2,3,\ldots\ldots N_z-1$. The distribution of variables on grid points is shown in the figure A.1.

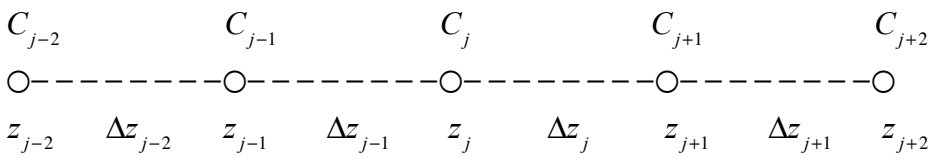

Figure A.1 Distribution of variables on non-uniformed grid

The Taylor expansion of discrete variables C_{j-3}, C_{j-2}, C_{j-1}, C_{j+1}, C_{j+2}, and C_{j+3} at z_j are

$$C_{j-3}=C_j-\left(\Delta z_{j-3}+\Delta z_{j-2}+\Delta z_{j-1}\right)\left.\frac{\partial C}{\partial z}\right|_j+\frac{\left(\Delta z_{j-3}+\Delta z_{j-2}+\Delta z_{j-1}\right)^2}{2!}\left.\frac{\partial^2 C}{\partial r^2}\right|_j$$
$$-\frac{\left(\Delta z_{j-3}+\Delta z_{j-2}+\Delta z_{j-1}\right)^3}{3!}\left.\frac{\partial^3 C}{\partial r^3}\right|_j+\frac{\left(\Delta z_{j-3}+\Delta z_{j-2}+\Delta z_{j-1}\right)^4}{4!}\left.\frac{\partial^4 C}{\partial r^4}\right|_j+HOT$$

$$C_{j-2}=C_j-\left(\Delta z_{j-2}+\Delta z_{j-1}\right)\left.\frac{\partial C}{\partial z}\right|_j+\frac{\left(\Delta z_{j-2}+\Delta z_{j-1}\right)^2}{2!}\left.\frac{\partial^2 C}{\partial r^2}\right|_j-\frac{\left(\Delta z_{j-2}+\Delta z_{j-1}\right)^3}{3!}\left.\frac{\partial^3 C}{\partial r^3}\right|_j$$
$$+\frac{\left(\Delta z_{j-2}+\Delta z_{j-1}\right)^4}{4!}\left.\frac{\partial^4 C}{\partial r^4}\right|_j+HOT$$

$$C_{j-1} = C_j - \Delta z_{j-1}\left.\frac{\partial C}{\partial z}\right|_j + \left.\frac{\Delta z_{j-}^2}{2!}\frac{\partial^2 C}{\partial z^2}\right|_j - \left.\frac{\Delta z_{j-}^3}{3!}\frac{\partial^3 C}{\partial z^3}\right|_j + \left.\frac{\Delta z_{j-}^4}{4!}\frac{\partial^4 C}{\partial z^4}\right|_j - \left.\frac{\Delta z_{j-}^5}{5!}\frac{\partial^5 C}{\partial z^5}\right|_j + HOT$$

$$C_{j+1} = C_j + \Delta z_j\left.\frac{\partial C}{\partial z}\right|_j + \left.\frac{\Delta z_j^2}{2!}\frac{\partial^2 C}{\partial z^2}\right|_j + \left.\frac{\Delta z_j^3}{3!}\frac{\partial^3 C}{\partial z^3}\right|_j + \left.\frac{\Delta z_j^4}{4!}\frac{\partial^4 C}{\partial z^4}\right|_j + \left.\frac{\Delta z_j^5}{5!}\frac{\partial^5 C}{\partial z^5}\right|_j + HOT$$

$$C_{j+2} = C_j + \left(\Delta z_j + \Delta z_{j+1}\right)\left.\frac{\partial C}{\partial z}\right|_j + \left.\frac{\left(\Delta z_j + \Delta z_{j+1}\right)^2}{2!}\frac{\partial^2 C}{\partial r^2}\right|_j + \left.\frac{\left(\Delta z_j + \Delta z_{j+1}\right)^3}{3!}\frac{\partial^3 C}{\partial r^3}\right|_j + \left.\frac{\left(\Delta z_j + \Delta z_{j+1}\right)^4}{4!}\frac{\partial^4 C}{\partial r^4}\right|_j + HOT$$

$$C_{j+3} = C_j + \left(\Delta z_j + \Delta z_{j+1} + \Delta z_{j+2}\right)\left.\frac{\partial C}{\partial z}\right|_j + \left.\frac{\left(\Delta z_j + \Delta z_{j+1} + \Delta z_{j+2}\right)^2}{2!}\frac{\partial^2 C}{\partial r^2}\right|_j + \left.\frac{\left(\Delta z_j + \Delta z_{j+1} + \Delta z_{j+2}\right)^3}{3!}\frac{\partial^3 C}{\partial r^3}\right|_j + \left.\frac{\left(\Delta z_j + \Delta z_{j+1} + \Delta z_{j+2}\right)^4}{4!}\frac{\partial^4 C}{\partial r^4}\right|_j + HOT$$

where *HOT* in the formulae means the high order terms. The accuracy and consistency of center, backward and forward differential formulae are analyzed by means of the Taylor table. The center difference formula of second derivative on non-uniformed grid is

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_j - bC_{j-1} - cC_j - dC_{j+1} = \varepsilon_c$$

The Taylor table is

	C_j	$\left.\frac{\partial C}{\partial z}\right\|_j$	$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	$\left.\frac{\partial^3 C}{\partial z^3}\right\|_j$
$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	0	0	1	0
$-bC_{j-1}$	$-b$	$b\Delta z_{j-1}$	$-b\frac{\Delta z_{j-1}^2}{2!}$	$b\frac{\Delta z_{j-1}^3}{3!}$
$-cC_j$	$-c$	0	0	0
$-dC_{j+1}$	$-d$	$-d\Delta z_j$	$-d\frac{\Delta z_j^2}{2!}$	$-d\frac{\Delta z_j^3}{3!}$
ε_c	0	0	0	ε_c

The linear algebraic equations and solution are

$$\begin{cases} b+c+d=0 \\ \left(\Delta z_{j-1}\right)b-\left(\Delta z_j\right)d=0 \\ \left(\Delta z_{j-1}^2\right)b+\left(\Delta z_j^2\right)d=2 \end{cases}, \text{ where } \begin{cases} b=\dfrac{2}{\Delta z_{j-1}\left(\Delta z_{j-1}+\Delta z_j\right)} \\ c=\dfrac{-2}{\Delta z_{j-1}\Delta z_j} \\ d=\dfrac{2}{\Delta z_j\left(\Delta z_{j-1}+\Delta z_j\right)} \end{cases}$$

The center difference formula of second derivative and leading order error on a non-uniform grid are

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_{i,j}=\frac{2}{\Delta z_{j-1}\left(\Delta z_{j-1}+\Delta z_j\right)}C_{j-1}-\frac{2}{\Delta z_{j-1}\Delta z_j}C_j+\frac{2}{\Delta z_j\left(\Delta z_{j-1}+\Delta z_j\right)}C_{j+1}$$

$$, \ \varepsilon_c=\frac{\Delta z_{j-1}-\Delta z_j}{3}\left.\frac{\partial^3 C}{\partial z^3}\right|_j$$

The backward difference formula of second derivative on non-uniformed grid is

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_j - aC_{j-2}-bC_{j-1}-cC_j=\varepsilon_b$$

The Taylor table is

	C_j	$\left.\frac{\partial C}{\partial z}\right\|_j$	$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	$\left.\frac{\partial^3 C}{\partial z^3}\right\|_j$
$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	0	0	1	0
$-aC_{j-2}$	$-a$	$a\left(\Delta z_{j-2}+\Delta z_{j-1}\right)$	$-a\frac{\left(\Delta z_{j-2}+\Delta z_{j-1}\right)^2}{2!}$	$a\frac{\left(\Delta z_{j-2}+\Delta z_{j-1}\right)^3}{3!}$
$-bC_{j-1}$	$-b$	$b\Delta z_{j-1}$	$-b\frac{\Delta z_{j-1}^2}{2!}$	$b\frac{\Delta z_{j-1}^3}{3!}$
$-cC_j$	$-c$	0	0	0
ε_b	0	0	0	ε_b

The linear algebraic equations and solution are

$$\begin{cases} a+b+c=0 \\ \left(\Delta z_{j-2}+\Delta z_{j-1}\right)a+\left(\Delta z_{j-1}\right)\text{b}=0 \\ \left(\Delta z_{j-2}+\Delta z_{j-1}\right)^2 a+\left(\Delta z_{j-1}\right)^2 b=2 \end{cases}, \text{ where } \begin{cases} a=\dfrac{2}{\Delta z_{j-2}\left(\Delta z_{j-2}+\Delta z_{j-1}\right)} \\ b=\dfrac{-2}{\Delta z_{j-2}\Delta z_{j-1}} \\ c=\dfrac{2}{\Delta z_{j-2}\left(\Delta z_{j-2}+\Delta z_{j-1}\right)} \end{cases}$$

The backward difference formula of second derivative and leading order error on a non-uniform grid are

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_{i,j}=\frac{2}{\Delta z_{j-2}\left(\Delta z_{j-2}+\Delta z_{j-1}\right)}C_{j-2}-\frac{2}{\Delta z_{j-2}\Delta z_{j-1}}C_{j-1}-\frac{2}{\Delta z_{j-2}\left(\Delta z_{j-2}+\Delta z_{j-1}\right)}C_j,$$

$$\varepsilon_b=\frac{\Delta z_{j-2}+2\Delta z_{j-1}}{3}\left.\frac{\partial^3 C}{\partial z^3}\right|_j$$

The forward difference formula of second derivative on non-uniformed grid is

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_j - cC_j - dC_{j+1} - eC_{j+2} = \varepsilon_f$$

The Taylor table is

	C_j	$\left.\frac{\partial C}{\partial z}\right\|_j$	$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	$\left.\frac{\partial^3 C}{\partial z^3}\right\|_j$
$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	0	0	1	0
$-cC_j$	$-c$	0	0	0
$-dC_{j+1}$	$-d$	$-d\Delta z_j$	$-d\frac{\Delta z_j^2}{2!}$	$-d\frac{\Delta z_j^3}{3!}$
$-eC_{j+2}$	$-e$	$-e\left(\Delta z_j+\Delta z_{j+1}\right)$	$-e\frac{\left(\Delta z_j+\Delta z_{j+1}\right)^2}{2!}$	$-e\frac{\left(\Delta z_j+\Delta z_{j+1}\right)^3}{3!}$
ε_f	0	0	0	ε_f

The linear algebraic equations and solution are

$$\begin{cases} c+d+e=0 \\ \left(\Delta z_j\right)d+\left(\Delta z_j+\Delta z_{j+1}\right)e=0 \\ \left(\Delta z_j\right)^2 d+\left(\Delta z_j+\Delta z_{j+1}\right)^2 e=2 \end{cases}, \text{ where } \begin{cases} c=\dfrac{2}{\Delta z_j\left(\Delta z_j+\Delta z_{j+1}\right)} \\ d=\dfrac{-2}{\Delta z_j \Delta z_{j+1}} \\ e=\dfrac{2}{\Delta z_{j+1}\left(\Delta z_j+\Delta z_{j+1}\right)} \end{cases}$$

The forward difference formula of second derivative and leading order error on a non-uniform grid are

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_{i,j}=\frac{2}{\Delta z_j\left(\Delta z_j+\Delta z_{j+1}\right)}C_j-\frac{2}{\Delta z_j \Delta z_{j+1}}C_{j+1}+\frac{2}{\Delta z_{j+1}\left(\Delta z_j+\Delta z_{j+1}\right)}C_{j+2}$$

$$\varepsilon_f=-\frac{2\Delta z_j+\Delta z_{j+1}}{3}\left.\frac{\partial^3 C}{\partial z^3}\right|_j$$

All the leading order errors for these three methods are only first order in space, but the leading order error is second order accurate on uniform grid. The center difference formula of second derivative on uniform grid and leading order error are

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_{i,j}=\frac{C_{i+1,j}-2C_{i,j}+C_{i-1,j}}{\Delta z^2},\ \varepsilon=-\frac{\Delta z^2}{12}\left.\frac{\partial^4 C}{\partial z^4}\right|_{i,j}$$

In order to get higher order accuracy, a four-point method is used to evaluate the second derivative on the first and last unit volume. One inner node is added to get second order accuracy. For the first unit volume, the four-point differential formulae of second derivative is assumed to be

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_j-bC_{j-1}-cC_j-dC_{j+1}-eC_{j+2}=\varepsilon_l$$

The Taylor table is

	C_j	$\left.\frac{\partial C}{\partial z}\right\|_j$	$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	$\left.\frac{\partial^3 C}{\partial z^3}\right\|_j$	$\left.\frac{\partial^4 C}{\partial z^4}\right\|_j$
$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	0	0	1	0	0
$-bC_{j-1}$	$-b$	$\mathrm{b}\Delta z_{j-1}$	$-b\frac{\Delta z_{j-1}^2}{2!}$	$\mathrm{b}\frac{\Delta z_{j-1}^3}{3!}$	$-b\frac{\Delta z_{j-1}^4}{4!}$
$-cC_j$	$-c$	0	0	0	0
$-dC_{j+1}$	$-d$	$-d\Delta z_j$	$-d\frac{\Delta z_j^2}{2!}$	$-d\frac{\Delta z_j^3}{3!}$	$-d\frac{\Delta z_j^4}{4!}$
$-eC_{j+2}$	$-e$	$-e\left(\Delta z_j+\Delta z_{j+1}\right)$	$-e\frac{\left(\Delta z_j+\Delta z_{j+1}\right)^2}{2!}$	$-e\frac{\left(\Delta z_j+\Delta z_{j+1}\right)^3}{3!}$	$-e\frac{\left(\Delta z_j+\Delta z_{j+1}\right)^4}{4!}$
ε_l	0	0	0	0	ε_l

The linear algebraic equations and solution are

$$\begin{cases} b+c+d+e=0 \\ \left(\Delta z_{j-1}\right)\mathrm{b}-\left(\Delta z_j\right)d-\left(\Delta z_j+\Delta z_{j+1}\right)e=0 \\ \left(\Delta z_{j-1}\right)^2 b+\left(\Delta z_j\right)^2 d+\left(\Delta z_j+\Delta z_{j+1}\right)^2 e=2 \\ \left(\Delta z_{j-1}\right)^3 \mathrm{b}-\left(\Delta z_j\right)^3 d-\left(\Delta z_j+\Delta z_{j+1}\right)^3 e=0 \end{cases}$$

and

$$\begin{cases} b=\dfrac{2\left[\Delta z_j+\left(\Delta z_j+\Delta z_{j+1}\right)\right]}{\Delta z_{j-1}\left(\Delta z_{j-1}+\Delta z_j\right)\left(\Delta z_{j-1}+\Delta z_j+\Delta z_{j+1}\right)} \\ c=\dfrac{2\left[\Delta z_{j-1}-\Delta z_j-\left(\Delta z_j+\Delta z_{j+1}\right)\right]}{\Delta z_{j-1}\Delta z_j\left(\Delta z_j+\Delta z_{j+1}\right)} \\ d=\dfrac{2\left[\left(\Delta z_j+\Delta z_{j+1}\right)-\Delta z_{j-1}\right]}{\left(\Delta z_{j-1}+\Delta z_j\right)\Delta z_j\Delta z_{j+1}} \\ e=\dfrac{2\left[\Delta z_{j-1}-\Delta z_j\right]}{\left(\Delta z_{j-1}+\Delta z_j+\Delta z_{j+1}\right)\left(\Delta z_j+\Delta z_{j+1}\right)\Delta z_{j+1}} \end{cases}$$

The four-point difference formula of second derivative for the first unit volume and leading order error is

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_j = \frac{2\left[\Delta z_j + \left(\Delta z_j + \Delta z_{j+1}\right)\right]}{\Delta z_{j-1}\left(\Delta z_{j-1} + \Delta z_j\right)\left(\Delta z_{j-1} + \Delta z_j + \Delta z_{j+1}\right)} C_{j-1} + \frac{2\left[\Delta z_{j-1} - \Delta z_j - \left(\Delta z_j + \Delta z_{j+1}\right)\right]}{\Delta z_{j-1}\Delta z_j\left(\Delta z_j + \Delta z_{j+1}\right)} C_j$$

$$+ \frac{2\left[\left(\Delta z_j + \Delta z_{j+1}\right) - \Delta z_{j-1}\right]}{\left(\Delta z_{j-1} + \Delta z_j\right)\Delta z_j \Delta z_{j+1}} C_{j+1} + \frac{2\left[\Delta z_{j-1} - \Delta z_j\right]}{\left(\Delta z_{j-1} + \Delta z_j + \Delta z_{j+1}\right)\left(\Delta z_j + \Delta z_{j+1}\right)\Delta z_{j+1}} C_{j+2}$$

$$, \varepsilon_l = \frac{\left(-\Delta z_{j-1}\right)\Delta z_j + \left(-\Delta z_{j-1}\right)\left(\Delta z_j + \Delta z_{j+1}\right) + \Delta z_j\left(\Delta z_j + \Delta z_{j+1}\right)}{12} \left.\frac{\partial^4 C}{\partial z^4}\right|_j$$

The four-point differential formulae of second derivative for the last unit volume is assumed to be

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_j - aC_{j-2} - bC_{j-1} - cC_j - dC_{j+1} = \varepsilon_r$$

The Taylor table is

	C_j	$\left.\frac{\partial C}{\partial z}\right\|_j$	$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	$\left.\frac{\partial^3 C}{\partial z^3}\right\|_j$	$\left.\frac{\partial^4 C}{\partial z^4}\right\|_j$
$\left.\frac{\partial^2 C}{\partial z^2}\right\|_j$	0	0	1	0	0
$-aC_{j-2}$	$-a$	$a\left(\Delta z_{j-2} + \Delta z_{j-1}\right)$	$-a\frac{\left(\Delta z_{j-2} + \Delta z_{j-1}\right)^2}{2!}$	$a\frac{\left(\Delta z_{j-2} + \Delta z_{j-1}\right)^3}{3!}$	$-a\frac{\left(\Delta z_{j-2} + \Delta z_{j-1}\right)^4}{4!}$
$-bC_{j-1}$	$-b$	$b\Delta z_{j-1}$	$-b\frac{\Delta z_{j-1}^2}{2!}$	$b\frac{\Delta z_{j-1}^3}{3!}$	$-b\frac{\Delta z_{j-1}^4}{4!}$
$-cC_j$	$-c$	0	0	0	0
$-dC_{j+1}$	$-d$	$-d\Delta z_j$	$-d\frac{\Delta z_j^2}{2!}$	$-d\frac{\Delta z_j^3}{3!}$	$-d\frac{\Delta z_j^4}{4!}$
ε_r	0	0	0	0	ε_r

The linear algebraic equations and solution are

$$\begin{cases} a+b+c+d=0 \\ \left(\Delta z_{j-2}+\Delta z_{j-1}\right)a+\left(\Delta z_{j-1}\right)\mathrm{b}-\left(\Delta z_{j}\right)d=0 \\ \left(\Delta z_{j-2}+\Delta z_{j-1}\right)^{2}a+\left(\Delta z_{j-1}\right)^{2}b+\left(\Delta z_{j}\right)^{2}d=2 \\ \left(\Delta z_{j-2}+\Delta z_{j-1}\right)^{3}a+\left(\Delta z_{j-1}\right)^{3}\mathrm{b}-\left(\Delta z_{j}\right)^{3}d=0 \end{cases}$$

and

$$\begin{cases} a=-\dfrac{2\left[\Delta z_{j-1}-\Delta z_{j}\right]}{\Delta z_{j-2}\left(\Delta z_{j-2}+\Delta z_{j-1}\right)\left(\Delta z_{j-2}+\Delta z_{j-1}+\Delta z_{j}\right)} \\ b=\dfrac{2\left[\left(\Delta z_{j-2}+\Delta z_{j-1}\right)-\Delta z_{j}\right]}{\Delta z_{j-2}\Delta z_{j-1}\left(\Delta z_{j-1}+\Delta z_{j}\right)} \\ c=-\dfrac{2\left[\left(\Delta z_{j-2}+\Delta z_{j-1}\right)\Delta z_{j-1}-\Delta z_{j}\right]}{\left(\Delta z_{j-2}+\Delta z_{j-1}\right)\Delta z_{j-1}\Delta z_{j}} \\ d=\dfrac{2\left[\left(\Delta z_{j-2}+\Delta z_{j-1}\right)+\Delta z_{j-1}\right]}{\left(\Delta z_{j-2}+\Delta z_{j-1}+\Delta z_{j}\right)\left(\Delta z_{j-1}+\Delta z_{j}\right)\Delta z_{j}} \end{cases}$$

The four-point difference formula of second derivative for the last unit volume and leading order error is

$$\left.\frac{\partial^{2}C}{\partial z^{2}}\right|_{j}=\frac{2\left[-\Delta z_{j-1}+\Delta z_{j}\right]}{\Delta z_{j-2}\left(\Delta z_{j-2}+\Delta z_{j-1}\right)\left(\Delta z_{j-2}+\Delta z_{j-1}+\Delta z_{j}\right)}C_{j-2}+\frac{2\left[\left(\Delta z_{j-2}+\Delta z_{j-1}\right)-\Delta z_{j}\right]}{\Delta z_{j-2}\Delta z_{j-1}\left(\Delta z_{j-1}+\Delta z_{j}\right)}C_{j-1}$$
$$+\frac{2\left[-\left(\Delta z_{j-2}+\Delta z_{j-1}\right)-\Delta z_{j-1}+\Delta z_{j}\right]}{\left(\Delta z_{j-2}+\Delta z_{j-1}\right)\Delta z_{j-1}\Delta z_{j}}C_{j}+\frac{2\left[\left(\Delta z_{j-2}+\Delta z_{j-1}\right)+\Delta z_{j-1}\right]}{\left(\Delta z_{j-2}+\Delta z_{j-1}+\Delta z_{j}\right)\left(\Delta z_{j-1}+\Delta z_{j}\right)\Delta z_{j}}C_{j+1}$$
$$,\ \varepsilon_{r}=\frac{\left(-\Delta z_{j-2}-\Delta z_{j-1}\right)\left(-\Delta z_{j-1}\right)+\left(-\Delta z_{j-2}-\Delta z_{j-1}\right)\Delta z_{j}+\left(-\Delta z_{j-1}\right)\Delta z_{j}}{12}\left.\frac{\partial^{4}C}{\partial z^{4}}\right|_{j}$$

If the grid is defined by the finite volume method with the first and last node half unit distance to the boundary, the second derivative with second order accurate in space can be acquired by applying the four-point differential formulae. The grid definition is provided in figure A.2. The circles in the plot represent for inner node and squares are the position of boundary. The reactor is divided into N_z equally spaced volume and is

numbered for 1 to N_z. The subscript 0 and N_z+1 represent the left and right boundary respectively.

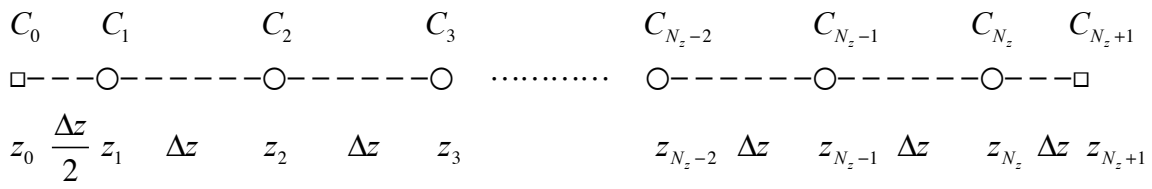

Figure A.2. Grid definition of finite volume method

The second derivatives and leading order error for the first and last reactor volume from the four-point differential formulae are

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_{j=1} = \frac{\frac{16}{5}C_0 - 5C_1 + 2C_2 - \frac{1}{5}C_3}{\Delta z^2},\ \varepsilon_1 = \frac{\Delta z^2}{24}\left.\frac{\partial^4 C}{\partial z^4}\right|_{j=1}$$

$$\left.\frac{\partial^2 C}{\partial z^2}\right|_{j=N_z} = \frac{-\frac{1}{5}C_{Nz-2} + 2C_{Nz-1} - 5C_{Nz} + \frac{16}{5}C_{Nz+1}}{\Delta z^2},\ \varepsilon_{Nz} = \frac{\Delta z^2}{24}\left.\frac{\partial^4 C}{\partial z^4}\right|_{j=N_z}$$

Appendix B Numerical Method for CSTR Model

Most of the equations in this appendix show up in pair. The upper one is the equation for the gas phase and the lower one is for the solid phase. The mass balance equation of CSTR in index notation is given as following

$$\frac{dC_i}{dt} = \begin{cases} \frac{1}{\tau}\left(C_{in,i} - C_i\right) + \alpha S_{ik} R_k \\ S_{ik} R_k \end{cases}$$

C : concentration
t : time variable, sec
τ : resisdence time, sec
C_{in} : inlet concentration
α : surface to gas capacity ratio
S : stoichiometric matrix
R : vector of rate equation
i : index of component
k : index of reaction rate

The concentration and reaction rate are dimensionless due to the normalize procedure described in chapter two, and the dimension of the equation is the inverse of time. The following sections are the equations and algorithm of corresponding numerical methods.

B.1 Explicit Method

There are three types of explicit numerical method in this section, explicit Euler (EE), Runge-Kutta (RK) and Adams-Bashforth (AB), and the number appended to the abbreviation of each method represent the order of accuracy. The slope for the differential equation is f and the time interval is Δt. The superscript for the variables is for time step and the subscript is for component.

Explicit Euler (EE)

$$C_i^{n+1} = \begin{cases} C_i^n + \dfrac{\Delta t}{\tau}\left(C_{in,i}^n - C_i^n\right) + \Delta t \alpha S_{ik} R_k^n \\ C_i^n + \Delta t S_{ik} R_k^n \end{cases}$$

Second-order Runge-Kutta (RK2)

$$f_i^1 = \begin{cases} \dfrac{1}{\tau}\left(C_{in,i}^n - C_i^n\right) + \alpha S_{ik} R_k^n, \; C_i^* = C_i^n + \Delta t\left(\dfrac{1}{2} f_i^1\right) \\ S_{ik} R_k^n \end{cases}$$

$$f_i^2 = \begin{cases} \dfrac{1}{\tau}\left(C_{in,i}^* - C_i^*\right) + \alpha S_{ik} R_k^*, \; C_i^{n+1} = C_i^n + \Delta t\left(f_i^2\right) \\ S_{ik} R_k^* \end{cases}$$

Third-order Runge-Kutta (RK3)

$$f_i^1 = \begin{cases} \dfrac{1}{\tau}\left(C_{in,i}^n - C_i^n\right) + \alpha S_{ik} R_k^n, \; C_i^* = C_i^n + \dfrac{\Delta t}{2}\left(f_i^1\right) \\ S_{ik} R_k^n \end{cases}$$

$$f_i^2 = \begin{cases} \dfrac{1}{\tau}\left(C_{in,i}^* - C_i^*\right) + \alpha S_{ik} R_k^*, \; C_i^{**} = C_i^n + \Delta t\left(-f_i^1 + 2 f_i^2\right) \\ S_{ik} R_k^* \end{cases}$$

$$f_i^3 = \begin{cases} \dfrac{1}{\tau}\left(C_{in,i}^{**} - C_i^{**}\right) + \alpha S_{ik} R_k^{**}, \; C_i^{n+1} = C_i^n + \Delta t\left(\dfrac{1}{6} f_i^1 + \dfrac{2}{3} f_i^2 + \dfrac{1}{6} f_i^3\right) \\ S_{ik} R_k^{**} \end{cases}$$

Fourth-order Runge-Kutta (RK4)

$$f_i^1 = \begin{cases} \dfrac{1}{\tau}\left(C_{in,i}^n - C_i^n\right) + \alpha S_{ik} R_k^n, \; C_i^* = C_i^{t_n} + \Delta t\left(\dfrac{1}{2} f_i^1\right) \\ S_{ik} R_k^n \end{cases}$$

$$f_i^2 = \begin{cases} \dfrac{1}{\tau}\left(C_{in,i}^* - C_i^*\right) + \alpha S_{ik} R_k^*, \; C_i^{**} = C_i^n + \Delta t\left(\dfrac{1}{2} f_i^2\right) \\ S_{ik} R_k^* \end{cases}$$

$$f_i^3 = \begin{cases} \frac{1}{\tau}\left(C_{in,i}^{**} - C_i^{**}\right) + \alpha S_{ik} R_k^{**} \\ S_{ik} R_k^{**} \end{cases}, \; C_i^{***} = C_i^n + \Delta t\left(f_i^3\right)$$

$$f_i^4 = \begin{cases} \frac{1}{\tau}\left(C_{in,i}^{***} - C_i^{***}\right) + \alpha S_{ik} R_k^{***} \\ S_{ik} R_k^{***} \end{cases}, \; C_i^{n+1} = C_i^n + \Delta t\left(\frac{1}{6} f_i^1 + \frac{1}{3} f_i^2 + \frac{1}{3} f_i^3 + \frac{1}{6} f_i^4\right)$$

Second-order Adams-Bashforth (AB2)

$$f_i^n = \begin{cases} \frac{1}{\tau}\left(C_{in,i}^n - C_i^n\right) + \alpha S_{ik} R_k^n \\ S_{ik} R_k^n \end{cases}, \; C_i^{n+1} = C_i^n + \Delta t\left(\frac{3}{2} f_i^n - \frac{1}{2} f_i^{n-1}\right)$$

Third-order Adams-Bashforth (AB3)

$$f_i^n = \begin{cases} \frac{1}{\tau}\left(C_{in,i}^n - C_i^n\right) + \alpha S_{ik} R_k^n \\ S_{ik} R_k^n \end{cases}, \; C_i^{n+1} = C_i^n + \Delta t\left(\frac{23}{12} f_i^n - \frac{16}{12} f_i^{n-1} + \frac{5}{12} f_i^{n-2}\right)$$

Fourth-order Adams-Bashforth (AB4)

$$f_i^n = \begin{cases} \frac{1}{\tau}\left(C_{in,i}^n - C_i^n\right) + \alpha S_{ik} R_k^n \\ S_{ik} R_k^n \end{cases}, \; C_i^{n+1} = C_i^n + \Delta t\left(\frac{55}{24} f_i^n - \frac{59}{24} f_i^{n-1} + \frac{37}{24} f_i^{n-2} - \frac{3}{8} f_i^{n-3}\right)$$

Fifth-order Adams-Bashforth (AB5)

$$f_i^n = \begin{cases} \frac{1}{\tau}\left(C_{in,i}^n - C_i^n\right) + \alpha S_{ik} R_k^n \\ S_{ik} R_k^n \end{cases}$$

$$C_i^{n+1} = C_i^n + \Delta t\left(\frac{1901}{720} f_i^n - \frac{2774}{720} f_i^{n-1} + \frac{2616}{720} f_i^{n-2} - \frac{1274}{720} f_i^{n-3} + \frac{251}{720} f_i^{n-4}\right)$$

B.2 Implicit Method

There are three types of implicit numerical method in this section, implicit Euler (IE), backward differential formula (BDF) and Adams-Moulton (AM), and the number appended to the abbreviation of each method represent the order of accuracy. The function for Newton's method is F and the corresponding Jacobian matrix is J. The δ in these equation is the Kronecker delta.

Implicit Euler (IE)

$$F_l = \begin{cases} C_l^{n+1} - C_l^n - \dfrac{\Delta t}{\tau}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) - \Delta t \alpha S_{lk} R_k^{n+1} \\ C_l^{n+1} - C_l^n - \Delta t S_{lk} R_k^{n+1} \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \dfrac{\Delta t}{\tau}\delta_{i,l} - \Delta t \alpha S_{ik} \dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \Delta t \alpha S_{ik} \dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Second-order backward differential formula (BDF2)

$$F_l = \begin{cases} C_l^{n+1} - \dfrac{4}{3}C_l^n + \dfrac{1}{3}C_l^{n-1} - \dfrac{2}{3}\dfrac{\Delta t}{\tau}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) - \dfrac{2}{3}\Delta t \alpha S_{lk} R_k^{n+1} \\ F_l = C_l^{n+1} - \dfrac{4}{3}C_l^n + \dfrac{1}{3}C_l^{n-1} - \dfrac{2}{3}\Delta t S_{lk} R_k^{n+1} \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \dfrac{2}{3}\dfrac{\Delta t}{\tau}\delta_{i,l} - \dfrac{2}{3}\Delta t \alpha S_{ik} \dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \dfrac{2}{3}\Delta t \alpha S_{ik} \dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Third-order backward differential formula BDF3

$$F_l = \begin{cases} C_l^{n+1} - \frac{18}{11}C_l^n + \frac{9}{11}C_l^{n-1} - \frac{2}{11}C_l^{n-2} - \frac{6}{11}\frac{\Delta t}{\tau}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) - \frac{6}{11}\Delta t\alpha S_{lk}R_k^{n+1} \\ C_l^{n+1} - \frac{18}{11}C_l^n + \frac{9}{11}C_l^{n-1} - \frac{2}{11}C_l^{n-2} - \frac{6}{11}\Delta t S_{lk}R_k^{n+1} \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \frac{6}{11}\frac{\Delta t}{\tau}\delta_{i,l} - \frac{6}{11}\Delta t\alpha S_{ik}\frac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \frac{6}{11}\Delta t\alpha S_{ik}\frac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Fourth-order backward differential formula BDF4

$$F_l = \begin{cases} C_l^{n+1} - \frac{48}{25}C_l^n + \frac{36}{25}C_l^{n-1} - \frac{16}{25}C_l^{n-2} + \frac{3}{25}C_l^{n-3} - \frac{12}{25}\frac{\Delta t}{\tau}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) \\ \quad - \frac{12}{25}\Delta t\alpha S_{lk}R_k^{n+1} \\ C_l^{n+1} - \frac{48}{25}C_l^n + \frac{36}{25}C_l^{n-1} - \frac{16}{25}C_l^{n-2} + \frac{3}{25}C_l^{n-3} - \frac{12}{25}\Delta t S_{lk}R_k^{n+1} \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \frac{12}{25}\frac{\Delta t}{\tau}\delta_{i,l} - \frac{12}{25}\Delta t\alpha S_{ik}\frac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \frac{12}{25}\Delta t\alpha S_{ik}\frac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Fifth-order backward differential formula BDF5

$$F_l = \begin{cases} C_l^{n+1} - \frac{300}{137}C_l^n + \frac{300}{137}C_l^{n-1} - \frac{200}{137}C_l^{n-2} + \frac{75}{137}C_l^{n-3} - \frac{12}{137}C_l^{n-4} \\ \quad - \frac{60}{137}\frac{\Delta t}{\tau}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) - \frac{60}{137}\Delta t\alpha S_{lk}R_k^{n+1} \\ C_l^{n+1} - \frac{300}{137}C_l^n + \frac{300}{137}C_l^{n-1} - \frac{200}{137}C_l^{n-2} + \frac{75}{137}C_l^{n-3} - \frac{12}{137}C_l^{n-4} \\ \quad - \frac{60}{137}\Delta t S_{lk}R_k^{n+1} \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \dfrac{60}{137}\dfrac{\Delta t}{\tau}\delta_{i,l} - \dfrac{60}{137}\Delta t\alpha S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \dfrac{60}{137}\Delta t\alpha S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Sixth-order backward differential formula BDF6

$$F_l = \begin{cases} C_l^{n+1} - \dfrac{120}{49}C_l^n + \dfrac{150}{49}C_l^{n-1} - \dfrac{400}{147}C_l^{n-2} + \dfrac{75}{49}C_l^{n-3} - \dfrac{24}{49}C_l^{n-4} + \dfrac{10}{147}C_l^{n-4} \\ \quad - \dfrac{20}{49}\dfrac{\Delta t}{\tau}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) - \dfrac{20}{49}\Delta t\alpha S_{lk}R_k^{n+1} \\ C_l^{n+1} - \dfrac{120}{49}C_l^n + \dfrac{150}{49}C_l^{n-1} - \dfrac{400}{147}C_l^{n-2} + \dfrac{75}{49}C_l^{n-3} - \dfrac{24}{49}C_l^{n-4} + \dfrac{10}{147}C_l^{n-4} \\ \quad - \dfrac{20}{49}\Delta t S_{lk}R_k^{n+1} \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \dfrac{20}{49}\dfrac{\Delta t}{\tau}\delta_{i,l} - \dfrac{20}{49}\Delta t\alpha S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \dfrac{20}{49}\Delta t\alpha S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Second-order Adams-Moulton (AM2)

$$F_l = \begin{cases} C_l^{n+1} - C_l^n - \dfrac{\Delta t}{\tau}\left[\dfrac{1}{2}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) + \dfrac{1}{2}\left(C_{in,l}^n - C_l^n\right)\right] - \Delta t\alpha S_{lk}\left(\dfrac{1}{2}R_k^{n+1} + \dfrac{1}{2}R_k^n\right) \\ C_l^{n+1} - C_l^n - \Delta t S_{lk}\left(\dfrac{1}{2}R_k^{n+1} + \dfrac{1}{2}R_k^n\right) \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \dfrac{1}{2}\dfrac{\Delta t}{\tau}\delta_{i,l} - \dfrac{1}{2}\Delta t\alpha S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \dfrac{1}{2}\Delta t S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Third-order Adams-Moulton (AM3)

$$F_l = \begin{cases} C_l^{n+1} - C_l^n - \dfrac{\Delta t}{\tau}\left[\dfrac{5}{12}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) + \dfrac{2}{3}\left(C_{in,l}^{n} - C_l^{n}\right) - \dfrac{1}{12}\left(C_{in,l}^{n-1} - C_l^{n-1}\right)\right] \\ \qquad - \Delta t \alpha S_{lk}\left(\dfrac{5}{12}R_k^{n+1} + \dfrac{2}{3}R_k^n - \dfrac{1}{12}R_k^{n-1}\right) \\ C_l^{n+1} - C_l^n - \Delta t S_{lk}\left(\dfrac{5}{12}R_k^{n+1} + \dfrac{2}{3}R_k^n - \dfrac{1}{12}R_k^{n-1}\right) \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \dfrac{5}{12}\dfrac{\Delta t}{\tau}\delta_{i,l} - \dfrac{5}{12}\Delta t \alpha S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \dfrac{5}{12}\Delta t S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Fourth-order Adams-Moulton (AM4)

$$F_l = \begin{cases} C_l^{n+1} - C_l^n - \dfrac{\Delta t}{\tau}\left[\dfrac{3}{8}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) + \dfrac{19}{24}\left(C_{in,l}^{n} - C_l^{n}\right) - \dfrac{5}{24}\left(C_{in,l}^{n-1} - C_l^{n-1}\right)\right] \\ \qquad - \dfrac{\Delta t}{\tau}\left[\dfrac{1}{24}\left(C_{in,l}^{n-2} - C_l^{n-2}\right)\right] - \Delta t \alpha S_{lk}\left(\dfrac{3}{8}R_k^{n+1} + \dfrac{19}{24}R_k^n - \dfrac{5}{24}R_k^{n-1} + \dfrac{1}{24}R_k^{n-2}\right) \\ C_l^{n+1} - C_l^n - \Delta t S_{lk}\left(\dfrac{3}{8}R_k^{n+1} + \dfrac{19}{24}R_k^n - \dfrac{5}{24}R_k^{n-1} + \dfrac{1}{24}R_k^{n-2}\right) \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \dfrac{3}{8}\dfrac{\Delta t}{\tau}\delta_{i,l} - \dfrac{3}{8}\Delta t \alpha S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \dfrac{3}{8}\Delta t S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

Fifth-order Adams-Moulton (AM5)

$$F_i = \begin{cases} C_l^{n+1} - C_l^n - \dfrac{\Delta t}{\tau}\left[\dfrac{251}{720}\left(C_{in,l}^{n+1} - C_l^{n+1}\right) + \dfrac{646}{720}\left(C_{in,l}^{n} - C_l^{n}\right) - \dfrac{264}{720}\left(C_{in,l}^{n-1} - C_l^{n-1}\right)\right] \\ \qquad - \dfrac{\Delta t}{\tau}\left[\dfrac{106}{720}\left(C_{in,l}^{n-2} - C_l^{n-2}\right) - \dfrac{19}{720}\left(C_{in,i}^{n-3} - C_l^{n-3}\right)\right] \\ \qquad - \Delta t \alpha S_{lk}\left(\dfrac{251}{720}R_k^{n+1} + \dfrac{646}{720}R_k^{n} - \dfrac{264}{720}R_k^{n-1} + \dfrac{106}{720}R_k^{n-2} - \dfrac{19}{720}R_k^{n-3}\right) \\ C_l^{n+1} - C_l^n - \Delta t S_{lk}\left(\dfrac{251}{720}R_k^{n+1} + \dfrac{646}{720}R_k^{n} - \dfrac{264}{720}R_k^{n-1} + \dfrac{106}{720}R_k^{n-2} - \dfrac{19}{720}R_k^{n-3}\right) \end{cases}$$

$$J_{i,l} = \begin{cases} \delta_{i,l} + \dfrac{251}{720}\dfrac{\Delta t}{\tau}\delta_{i,l} - \dfrac{251}{720}\Delta t \alpha S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \\ \delta_{i,l} - \dfrac{251}{720}\Delta t S_{ik}\dfrac{\partial R_k^{n+1}}{\partial C_l^{n+1}} \end{cases}$$

$$C_i^{n+1} \leftarrow C_i^{n+1} + J_{i,l}^{-1}\left(-F_l\right)$$

B.3 Semi-implicit Method

The semi-implicit are the combination of implicit Crank-Nicholson and explicit method. Three explicit methods are add to CN, they are explicit Euler, second and third order Runge-Kutta method.

Crank-Nicholson with explicit Euler (CNEE)

$$C_i^{n+1} = \begin{cases} \left(\dfrac{2\tau - \Delta t}{2\tau + \Delta t}\right)C_i^n + \left(\dfrac{\Delta t}{2\tau + \Delta t}\right)\left(C_{in,i}^{n+1} + C_{in,i}^{n}\right) + \left(\dfrac{2\tau \Delta t}{2\tau + \Delta t}\right)\alpha S_{ik} R_k^n \\ C_i^n + \Delta t S_{ik} R_k^n \end{cases}$$

Crank-Nicholson with second-order Runge-Kutta (CNRK2)

$$C_i^{n+1/2} = \begin{cases} \left(\dfrac{4\tau - \Delta t}{4\tau + \Delta t}\right)C_i^n + \left(\dfrac{\Delta t}{4\tau + \Delta t}\right)\left(C_{in,i}^{n+1/2} + C_{in,i}^{n}\right) + \left(\dfrac{2\tau \Delta t}{4\tau + \Delta t}\right)\alpha S_{ik} R_k^n \\ C_i^n + \left(\dfrac{\Delta t}{2}\right)S_{ik} R_k^n \end{cases}$$

$$C_i^{n+1} = \begin{cases} \left(\frac{4\tau - \Delta t}{4\tau + \Delta t}\right) C_i^{n+1/2} + \left(\frac{\Delta t}{4\tau + \Delta t}\right)\left(C_{in,i}^{n+1/2} + C_{in,i}^{n+1}\right) + \left(\frac{4\tau\Delta t}{4\tau + \Delta t}\right)\alpha S_{ik}\left(R_k^{n+1/2} + \frac{1}{4}R_k^n\right) \\ C_i^{n+1/2} + \Delta t\alpha S_{ik}\left(R_k^{n+1/2} + \frac{1}{4}R_k^n\right) \end{cases}$$

Crank-Nicholson with third-order Runge-Kutta (CNRK3)

$$C_i^{n+8/15} = \begin{cases} \left(\frac{15\tau - 4\Delta t}{15\tau + 4\Delta t}\right) C_i^n + \left(\frac{4\Delta t}{15\tau + 4\Delta t}\right)\left(C_{in,i}^{n+8/15} + C_{in,i}^n\right) + \left(\frac{8\tau\Delta t}{15\tau + 4\Delta t}\right)\alpha S_{ik} R_k^n \\ C_i^n + \left(\frac{8\Delta t}{15}\right) S_{ik} R_k^n \end{cases}$$

$$C_i^{n+2/3} = \begin{cases} \left(\frac{15\tau - \Delta t}{15\tau + \Delta t}\right) C_i^{n+8/15} + \left(\frac{\Delta t}{15\tau + \Delta t}\right)\left(C_{in,i}^{n+2/3} + C_{in,i}^{n+8/15}\right) \\ \qquad + \left(\frac{5\tau\Delta t}{15\tau + \Delta t}\right)\alpha S_{ik} R_k^{n+2/3} - \left(\frac{17\tau\Delta t/4}{15\tau + \Delta t}\right)\alpha S_{ik} R_k^n \\ C_i^{n+8/15} + \Delta t S_{ik}\left(\frac{1}{3}R_k^{n+2/3} - \frac{17}{60}R_k^n\right) \end{cases}$$

$$C_i^{n+1} = \begin{cases} \left(\frac{6\tau - \Delta t}{6\tau + \Delta t}\right) C_i^{n+2/3} + \frac{\Delta t}{6\tau + \Delta t}\left(C_{in,i}^{n+1} + C_{in,i}^{n+2/3}\right) \\ \qquad + \left(\frac{9\tau\Delta t/2}{6\tau + \Delta t}\right)\alpha S_{ik} R_k^{n+1} - \left(\frac{5\tau\Delta t/2}{6\tau + \Delta t}\right)\alpha S_{ik} R_k^{n+2/3} \\ C_i^{n+2/3} + \Delta t S_{ik}\left(\frac{3}{4}R_k^{n+1} - \frac{5}{12}R_k^{n+2/3}\right) \end{cases}$$

Appendix C Numerical Method for DDR Model

Most of the equations in this appendix show up in pair. The upper one is the equation for the gas phase and the lower one is for the solid phase. The mass balance equation of DDR in index notation is given as following

$$\frac{dC_{i,j}}{dt} = \begin{cases} \frac{1}{\tau}\left(\frac{C_{i,j-1} - 2C_{i,j} + C_{i,j+1}}{\Delta Z^2} \right) + \alpha S_{i,k} R_{k,j} \\ S_{i,k} R_{k,j} \end{cases}$$

C : concentration
t : time variable, sec
τ : diffusion time constant, sec
ΔZ : grid size
α : surface to gas capacity ratio
S : stoichiometric matrix
R : rate equation matrix
i : index of component
j : index of grid point
k : index of reaction rate

The concentration and reaction rate are dimensionless due to the normalize procedure described in chapter two, and the dimension of the equation is the inverse of time. The following sections are the equations and algorithm of corresponding numerical methods.

B.1 Explicit Method

There are three types of explicit numerical method in this section, explicit Euler (EE), Runge-Kutta (RK) and Adams-Bashforth (AB), and the number appended to the abbreviation of each method represent the order of accuracy. The slope for the

differential equation is f and the time interval is Δt. The superscript for the variables is for time step and the subscript is for component and grid point.

Explicit Euler (EE)

$$C_{i,j}^{n+1} = \begin{cases} C_{i,j}^{n} + \Delta t \dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{n} - 2C_{i,j}^{n} + C_{i,j+1}^{n}}{\Delta Z^2} \right) + \Delta t \alpha S_{i,k} R_{k,j}^{n} \\ C_{i,j}^{n} + \Delta t S_{i,k} R_{k,j}^{n} \end{cases}$$

Second-order Runge-Kutta (RK2)

$$f_{i,j}^{1} = \begin{cases} \dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{n} - 2C_{i,j}^{n} + C_{i,j+1}^{n}}{\Delta Z^2} \right) + \alpha S_{i,k} R_{k,j}^{n}, \; C_{i,j}^{*} = C_{i,j}^{n} + \Delta t\left(\dfrac{1}{2} f_{i,j}^{1} \right) \\ S_{i,k} R_{k,j}^{n} \end{cases}$$

$$f_{i,j}^{2} = \begin{cases} \dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{*} - 2C_{i,j}^{*} + C_{i,j+1}^{*}}{\Delta Z^2} \right) + \alpha S_{i,k} R_{k,j}^{*}, \; C_{i,j}^{n+1} = C_{i,j}^{n} + \Delta t\left(f_{i,j}^{2} \right) \\ S_{i,k} R_{k,j}^{*} \end{cases}$$

Third-order Runge-Kutta (RK3)

$$f_{i,j}^{1} = \begin{cases} \dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{n} - 2C_{i,j}^{n} + C_{i,j+1}^{n}}{\Delta Z^2} \right) + \alpha S_{i,k} R_{k,j}^{n}, \; C_{i,j}^{*} = C_{i,j}^{n} + \Delta t\left(\dfrac{1}{2} f_{i,j}^{1} \right) \\ S_{i,k} R_{k,j}^{n} \end{cases}$$

$$f_{i,j}^{2} = \begin{cases} \dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{*} - 2C_{i,j}^{*} + C_{i,j+1}^{*}}{\Delta Z^2} \right) + \alpha S_{i,k} R_{k,j}^{*}, \; C_{i,j}^{**} = C_{i,j}^{n} + \Delta t\left(-f_{i,j}^{1} + 2 f_{i,j}^{2} \right) \\ S_{i,k} R_{k,j}^{*} \end{cases}$$

$$f_{i,j}^{3} = \begin{cases} \dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{**} - 2C_{i,j}^{**} + C_{i,j+1}^{**}}{\Delta Z^2} \right) + \alpha S_{i,k} R_{k,j}^{**}, \; C_{i,j}^{n+1} = C_{i,j}^{n} + \Delta t\left(\dfrac{1}{6} f_{i,j}^{1} + \dfrac{2}{3} f_{i,j}^{2} + \dfrac{1}{6} f_{i,j}^{3} \right) \\ S_{i,k} R_{k,j}^{**} \end{cases}$$

Fourth-order Runge-Kutta (RK4)

$$f_{i,j}^{1}=\begin{cases}\dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{n}-2C_{i,j}^{n}+C_{i,j+1}^{n}}{\Delta Z^{2}}\right)+\alpha S_{i,k}R_{k,j}^{n}, & C_{i,j}^{*}=C_{i,j}^{n}+\Delta t\left(\dfrac{1}{2}f_{i,j}^{1}\right)\\ S_{i,k}R_{k,j}^{n}\end{cases}$$

$$f_{i,j}^{2}=\begin{cases}\dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{*}-2C_{i,j}^{*}+C_{i,j+1}^{*}}{\Delta Z^{2}}\right)+\alpha S_{i,k}R_{k,j}^{*}, & C_{i,j}^{**}=C_{i,j}^{n}+\Delta t\left(\dfrac{1}{2}f_{i,j}^{2}\right)\\ S_{i,k}R_{k,j}^{*}\end{cases}$$

$$f_{i,j}^{3}=\begin{cases}\dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{**}-2C_{i,j}^{**}+C_{i,j+1}^{**}}{\Delta Z^{2}}\right)+\alpha S_{i,k}R_{k,j}^{**}, & C_{i,j}^{***}=C_{i,j}^{n}+\Delta t\left(f_{i,j}^{3}\right)\\ S_{i,k}R_{k,j}^{**}\end{cases}$$

$$f_{i,j}^{3}=\begin{cases}\dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{***}-2C_{i,j}^{***}+C_{i,j+1}^{***}}{\Delta Z^{2}}\right)+\alpha S_{i,k}R_{k,j}^{***},\\ S_{i,k}R_{k,j}^{***}\end{cases}$$

$$C_{i,j}^{n+1}=C_{i,j}^{n}+\Delta t\left(\frac{1}{6}f_{i,j}^{1}+\frac{1}{3}f_{i,j}^{2}+\frac{1}{3}f_{i,j}^{3}+\frac{1}{6}f_{i,j}^{4}\right)$$

Second-order Adams-Bashforth (AB2)

$$f_{i,j}^{n}=\begin{cases}\dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{n}-2C_{i,j}^{n}+C_{i,j+1}^{n}}{\Delta Z^{2}}\right)+\alpha S_{i,k}R_{k,j}^{n}\\ S_{i,k}R_{k,j}^{n}\end{cases}$$

$$C_{i,j}^{n+1}=C_{i,j}^{n}+\Delta t\left(\frac{3}{2}f_{i,j}^{n}-\frac{1}{2}f_{i,j}^{n-1}\right)$$

Third-order Adams-Bashforth (AB3)

$$f_{i,j}^{n}=\begin{cases}\dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{n}-2C_{i,j}^{n}+C_{i,j+1}^{n}}{\Delta Z^{2}}\right)+\alpha S_{i,k}R_{k,j}^{n}\\ S_{i,k}R_{k,j}^{n}\end{cases}$$

$$C_{i,j}^{n+1}=C_{i,j}^{n}+\Delta t\left(\frac{23}{12}f_{i,j}^{n}-\frac{16}{12}f_{i,j}^{n-1}+\frac{5}{12}f_{i,j}^{n-2}\right)$$

Fourth-order Adams-Bashforth (AB4)

$$f_{i,j}^{n}=\begin{cases}\dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{n}-2C_{i,j}^{n}+C_{i,j+1}^{n}}{\Delta Z^{2}}\right)+\alpha S_{i,k}R_{k,j}^{n}\\ S_{i,k}R_{k,j}^{n}\end{cases}$$

$$C_{i,j}^{n+1}=C_{i,j}^{n}+\Delta t\left(\frac{55}{24}f_{i,j}^{n}-\frac{59}{24}f_{i,j}^{n-1}+\frac{37}{24}f_{i,j}^{n-2}-\frac{3}{8}f_{i,j}^{n-3}\right)$$

Fifth-order Adams-Bashforth (AB5)

$$f_{i,j}^{n}=\begin{cases}\dfrac{1}{\tau}\left(\dfrac{C_{i,j-1}^{n}-2C_{i,j}^{n}+C_{i,j+1}^{n}}{\Delta Z^{2}}\right)+\alpha S_{i,k}R_{k,j}^{n}\\ S_{i,k}R_{k,j}^{n}\end{cases}$$

$$C_{i,j}^{n+1}=C_{i,j}^{n}+\Delta t\left(\frac{1901}{720}f_{i,j}^{n}-\frac{2774}{720}f_{i,j}^{n-1}+\frac{2616}{720}f_{i,j}^{n-2}-\frac{1274}{720}f_{i,j}^{n-3}+\frac{251}{720}f_{i,j}^{n-4}\right)$$

B.2 Implicit Method

There are three types of implicit numerical method in this section, implicit Euler (IE), backward differential formula (BDF) and Adams-Moulton (AM) method. The number appended to the abbreviation of each method represents the order of accuracy. The equations listed here are the general form of applying each method to the mass balance equation of DDR. To solve the nonlinear algebraic equations, further action is required to manage the storage of the concentration, so it can be iterate efficiently by either relaxation or Newton's method.

Implicit Euler (IE)

$$\begin{cases}C_{i,j}^{n+1}-C_{i,j}^{n}-\dfrac{\Delta t}{\tau\Delta Z^{2}}\left(C_{i,j-1}^{n+1}-2C_{i,j}^{n+1}+C_{i,j+1}^{n+1}\right)-\Delta t\alpha S_{i,k}R_{k,j}^{n+1}=0\\ C_{i,j}^{n+1}-C_{i,j}^{n}-\Delta t S_{i,k}R_{k,j}^{n+1}=0\end{cases}$$

Second-order backward differential formula (BDF2)

$$\begin{cases} C_{i,j}^{n+1} - \frac{4}{3}C_{i,j}^{n} + \frac{1}{3}C_{i,j}^{n-1} - \frac{2}{3}\frac{\Delta t}{\tau \Delta Z^2}\left(C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}\right) - \frac{2}{3}\Delta t \alpha S_{i,k} R_{k,j}^{n+1} = 0 \\ C_{i,j}^{n+1} - \frac{4}{3}C_{i,j}^{n} + \frac{1}{3}C_{i,j}^{n-1} - \frac{2}{3}\Delta t S_{i,k} R_{k,j}^{n+1} = 0 \end{cases}$$

Third-order backward differential formula (BDF3)

$$\begin{cases} C_{i,j}^{n+1} - \frac{18}{11}C_{i,j}^{n} + \frac{19}{11}C_{i,j}^{n-1} - \frac{2}{11}C_{i,j}^{n-2} - \frac{6}{11}\frac{\Delta t}{\tau \Delta Z^2}\left(C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}\right) - \frac{6}{11}\Delta t \alpha S_{i,k} R_{k,j}^{n+1} = 0 \\ C_{i,j}^{n+1} - \frac{18}{11}C_{i,j}^{n} + \frac{19}{11}C_{i,j}^{n-1} - \frac{2}{11}C_{i,j}^{n-2} - \frac{6}{11}\Delta t S_{i,k} R_{k,j}^{n+1} = 0 \end{cases}$$

Fourth-order backward differential formula (BDF4)

$$\begin{cases} C_{i,j}^{n+1} - \frac{48}{25}C_{i,j}^{n} + \frac{36}{25}C_{i,j}^{n-1} - \frac{16}{25}C_{i,j}^{n-2} + \frac{3}{25}C_{i,j}^{n-3} - \frac{12}{25}\frac{\Delta t}{\tau \Delta Z^2}\left(C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}\right) \\ \quad - \frac{12}{25}\Delta t \alpha S_{i,k} R_{k,j}^{n+1} = 0 \\ C_{i,j}^{n+1} - \frac{48}{25}C_{i,j}^{n} + \frac{36}{25}C_{i,j}^{n-1} - \frac{16}{25}C_{i,j}^{n-2} + \frac{3}{25}C_{i,j}^{n-3} - \frac{12}{25}\Delta t S_{i,k} R_{k,j}^{n+1} = 0 \end{cases}$$

Fifth-order backward differential formula (BDF5)

$$\begin{cases} C_{i,j}^{n+1} - \frac{300}{137}C_{i,j}^{n} + \frac{300}{137}C_{i,j}^{n-1} - \frac{200}{137}C_{i,j}^{n-2} + \frac{75}{137}C_{i,j}^{n-3} - \frac{12}{137}C_{i,j}^{n-4} \\ \quad - \frac{60}{137}\frac{\Delta t}{\tau \Delta Z^2}\left(C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}\right) - \frac{60}{137}\Delta t \alpha S_{i,k} R_{k,j}^{n+1} = 0 \\ C_{i,j}^{n+1} - \frac{300}{137}C_{i,j}^{n} + \frac{300}{137}C_{i,j}^{n-1} - \frac{200}{137}C_{i,j}^{n-2} + \frac{75}{137}C_{i,j}^{n-3} - \frac{12}{137}C_{i,j}^{n-4} - \frac{60}{137}\Delta t S_{i,k} R_{k,j}^{n+1} = 0 \end{cases}$$

Sixth-order backward differential formula (BDF6)

$$\begin{cases} C_{i,j}^{n+1} - \frac{120}{49}C_{i,j}^{n} + \frac{150}{49}C_{i,j}^{n-1} - \frac{400}{147}C_{i,j}^{n-2} + \frac{75}{49}C_{i,j}^{n-3} - \frac{24}{49}C_{i,j}^{n-4} + \frac{10}{147}C_{i,j}^{n-5} \\ \quad - \frac{20}{49}\frac{\Delta t}{\tau \Delta Z^2}\left(C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}\right) - \frac{20}{49}\Delta t \alpha S_{i,k} R_{k,j}^{n+1} = 0 \end{cases}$$

$$\begin{cases} C_{i,j}^{n+1} - \dfrac{120}{49} C_{i,j}^{n} + \dfrac{150}{49} C_{i,j}^{n-1} - \dfrac{400}{147} C_{i,j}^{n-2} + \dfrac{75}{49} C_{i,j}^{n-3} - \dfrac{24}{49} C_{i,j}^{n-4} + \dfrac{10}{147} C_{i,j}^{n-5} \\ \quad - \dfrac{20}{49} \Delta t S_{i,k} R_{k,j}^{n+1} = 0 \end{cases}$$

Second-order Adams-Moulton (AM2)

$$\begin{cases} C_{i,j}^{n+1} - C_{i,j}^{n} - \Delta t \alpha S_{i,k} \left(\dfrac{1}{2} R_{k,j}^{n+1} + \dfrac{1}{2} R_{k,j}^{n} \right) \\ \quad - \dfrac{\Delta t}{\tau} \left[\dfrac{1}{2} \dfrac{C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}}{\Delta Z^2} + \dfrac{1}{2} \dfrac{C_{i,j-1}^{n} - 2C_{i,j}^{n} + C_{i,j+1}^{n}}{\Delta Z^2} \right] = 0 \\ C_{i,j}^{n+1} - C_{i,j}^{n} - \Delta t S_{i,k} \left(\dfrac{1}{2} R_{k,j}^{n+1} + \dfrac{1}{2} R_{k,j}^{n} \right) = 0 \end{cases}$$

Third-order Adams-Moulton (AM3)

$$\begin{cases} C_{i,j}^{n+1} - C_{i,j}^{n} - \Delta t \alpha S_{i,k} \left(\dfrac{5}{12} R_{k,j}^{n+1} + \dfrac{2}{3} R_{k,j}^{n} - \dfrac{1}{12} R_{k,j}^{n-1} \right) \\ \quad - \dfrac{\Delta t}{\tau} \left[\dfrac{5}{12} \dfrac{C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}}{\Delta Z^2} + \dfrac{2}{3} \dfrac{C_{i,j-1}^{n} - 2C_{i,j}^{n} + C_{i,j+1}^{n}}{\Delta Z^2} \right] \\ \quad - \dfrac{\Delta t}{\tau} \left[-\dfrac{1}{12} \dfrac{C_{i,j-1}^{n-2} - 2C_{i,j}^{n-2} + C_{i,j+1}^{n-2}}{\Delta Z^2} \right] = 0 \\ C_{i,j}^{n+1} - C_{i,j}^{n} - \Delta t S_{i,k} \left(\dfrac{5}{12} R_{k,j}^{n+1} + \dfrac{2}{3} R_{k,j}^{n} - \dfrac{1}{12} R_{k,j}^{n-1} \right) = 0 \end{cases}$$

Fourth-order Adams-Moulton (AM4)

$$\begin{cases} C_{i,j}^{n+1} - C_{i,j}^{n} - \Delta t \alpha S_{i,k} \left(\dfrac{3}{8} R_{k,j}^{n+1} + \dfrac{19}{24} R_{k,j}^{n} - \dfrac{5}{24} R_{k,j}^{n-1} + \dfrac{1}{24} R_{k,j}^{n-2} \right) \\ \quad - \dfrac{\Delta t}{\tau} \left[\dfrac{3}{8} \dfrac{C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}}{\Delta Z^2} + \dfrac{19}{24} \dfrac{C_{i,j-1}^{n} - 2C_{i,j}^{n} + C_{i,j+1}^{n}}{\Delta Z^2} \right] \\ \quad - \dfrac{\Delta t}{\tau} \left[-\dfrac{5}{24} \dfrac{C_{i,j-1}^{n-2} - 2C_{i,j}^{n-2} + C_{i,j+1}^{n-2}}{\Delta Z^2} + \dfrac{1}{24} \dfrac{C_{i,j-1}^{n-3} - 2C_{i,j}^{n-3} + C_{i,j+1}^{n-3}}{\Delta Z^2} \right] = 0 \end{cases}$$

$$\left\{ C_{i,j}^{n+1} - C_{i,j}^{n} - \Delta t S_{i,k} \left(\frac{3}{8} R_{k,j}^{n+1} + \frac{19}{24} R_{k,j}^{n} - \frac{5}{24} R_{k,j}^{n-1} + \frac{1}{24} R_{k,j}^{n-2} \right) = 0 \right.$$

Fifth-order Adams-Moulton (AM5)

$$\left\{ \begin{aligned} & C_{i,j}^{n+1} - C_{i,j}^{n} - \Delta t \alpha S_{i,k} \left(\frac{251}{720} R_{k,j}^{n+1} + \frac{646}{720} R_{k,j}^{n} - \frac{264}{720} R_{k,j}^{n-1} + \frac{106}{720} R_{k,j}^{n-2} - \frac{19}{720} R_{k,j}^{n-3} \right) \\ & \quad - \frac{\Delta t}{\tau} \left[\frac{251}{720} \frac{C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}}{\Delta Z^2} + \frac{646}{720} \frac{C_{i,j-1}^{n} - 2C_{i,j}^{n} + C_{i,j+1}^{n}}{\Delta Z^2} \right] \\ & \quad - \frac{\Delta t}{\tau} \left[- \frac{264}{720} \frac{C_{i,j-1}^{n-2} - 2C_{i,j}^{n-2} + C_{i,j+1}^{n-2}}{\Delta Z^2} + \frac{106}{720} \frac{C_{i,j-1}^{n-3} - 2C_{i,j}^{n-3} + C_{i,j+1}^{n-3}}{\Delta Z^2} \right] \\ & \quad - \frac{\Delta t}{\tau} \left[- \frac{19}{720} \frac{C_{i,j-1}^{n-4} - 2C_{i,j}^{n-4} + C_{i,j+1}^{n-4}}{\Delta Z^2} \right] = 0 \\ & C_{i,j}^{n+1} - C_{i,j}^{n} - \Delta t S_{i,k} \left(\frac{251}{720} R_{k,j}^{n+1} + \frac{646}{720} R_{k,j}^{n} - \frac{264}{720} R_{k,j}^{n-1} + \frac{106}{720} R_{k,j}^{n-2} - \frac{19}{720} R_{k,j}^{n-3} \right) = 0 \end{aligned} \right.$$

B.3 Semi-implicit Method

The semi-implicit are the combination of implicit Crank-Nicholson and explicit method. Three explicit methods are added to CN, which are the explicit Euler, second and third order Runge-Kutta method. These methods can be solved efficiently by tri-diagonal Thomas algorithm or LU decomposition.

Crank-Nicholson with explicit Euler (CNEE)

$$\left\{ \begin{aligned} & -\left(\frac{\Delta t}{2\tau \Delta Z^2} \right) C_{i,j-1}^{n+1} + \left(1 + \frac{\Delta t}{\tau \Delta Z^2} \right) C_{i,j}^{n+1} - \left(\frac{\Delta t}{2\tau \Delta Z^2} \right) C_{i,j+1}^{n+1} = \\ & \quad \left(\frac{\Delta t}{2\tau \Delta Z^2} \right) C_{i,j-1}^{n} + \left(1 - \frac{\Delta t}{\tau \Delta Z^2} \right) C_{i,j}^{n} + \left(\frac{\Delta t}{2\tau \Delta Z^2} \right) C_{i,j+1}^{n} + \Delta t \alpha S_{i,k} R_{k,j}^{n} \\ & C_{i,j}^{n+1} = C_{i,j}^{n} + \Delta t S_{i,k} R_{k,j}^{n} \end{aligned} \right.$$

Crank-Nicholson with second-order Runge-Kutta (CNRK2)

$$\begin{cases} -\left(\frac{\Delta t}{4\tau\Delta Z^2}\right)C_{i,j-1}^{n+1/2}+\left(1+\frac{\Delta t}{2\tau\Delta Z^2}\right)C_{i,j}^{n+1/2}-\left(\frac{\Delta t}{4\tau\Delta Z^2}\right)C_{i,j+1}^{n+1/2}= \\ \quad \left(\frac{\Delta t}{4\tau\Delta Z^2}\right)C_{i,j-1}^{n}+\left(1-\frac{\Delta t}{2\tau\Delta Z^2}\right)C_{i,j}^{n}+\left(\frac{\Delta t}{4\tau\Delta Z^2}\right)C_{i,j+1}^{n}+\frac{\Delta t}{2}\alpha S_{i,k}R_{k,j}^{n} \\ C_{i,j}^{n+1/2}=C_{i,j}^{n}+\frac{\Delta t}{2}S_{i,k}R_{k,j}^{n} \end{cases}$$

$$\begin{cases} -\frac{\Delta t}{2\tau\Delta Z^2}C_{i,j-1}^{n+1}+\left(1+\frac{\Delta t}{\tau\Delta Z^2}\right)C_{i,j}^{n+1}-\frac{\Delta t}{2\tau\Delta Z^2}C_{i,j+1}^{n+1}= \\ \quad \left(\frac{\Delta t}{2\tau\Delta Z^2}\right)C_{i,j-1}^{n+1/2}+\left(1-\frac{\Delta t}{\tau\Delta Z^2}\right)C_{i,j}^{n+1/2}+\left(\frac{\Delta t}{2\tau\Delta Z^2}\right)C_{i,j+1}^{n+1/2}+\Delta t\alpha S_{i,k}\left(R_{k,j}^{n+1/2}+\frac{1}{4}R_{k,j}^{n}\right) \\ C_{i,j}^{n+1}=C_{i,j}^{n+1/2}+\Delta t S_{i,k}\left(R_{k,j}^{n+1/2}+\frac{1}{4}R_{k,j}^{n}\right) \end{cases}$$

Crank-Nicholson with third-order Runge-Kutta (CNRK3)

$$-\left(\frac{\bar{h}_{rk}}{2\tau\Delta Z^2}\right)C_{i,j-1}^{rk+1}+\left(1+\frac{\bar{h}_{rk}}{\tau\Delta Z^2}\right)C_{i,j}^{rk+1}-\left(\frac{\bar{h}_{rk}}{2\tau\Delta Z^2}\right)C_{i,j+1}^{rk+1}=$$
$$\left(\frac{\bar{h}_{rk}}{2\tau\Delta Z^2}\right)C_{i,j-1}^{rk}+\left(1-\frac{\bar{h}_{rk}}{\tau\Delta Z^2}\right)C_{i,j}^{rk}+\left(\frac{\bar{h}_{rk}}{2\tau\Delta Z^2}\right)C_{i,j+1}^{rk}+\bar{h}_{rk}\alpha S_{i,k}\left(\bar{\beta}_{rk}R_{k,j}^{rk}+\bar{\zeta}_{rk}R_{k,j}^{rk}\right)$$

The parameters of CNRK3 are

$$\begin{cases} rk=1:\ \bar{h}_1=\alpha_1\Delta t & ,\bar{\beta}_1=\beta_1/\alpha_1 & ,\ \bar{\zeta}_1=0 \\ rk=2:\ \bar{h}_2=(\alpha_2-\alpha_1)\Delta t & ,\bar{\beta}_2=\beta_3/(\alpha_2-\alpha_1) & ,\ \bar{\zeta}_2=(\beta_2-\beta_1)/(\alpha_2-\alpha_1) \\ rk=3:\ \bar{h}_3=(1-\alpha_2)\Delta t & ,\bar{\beta}_3=\gamma_3/(1-\alpha_2) & ,\ \bar{\zeta}_3=(\gamma_2-\beta_3)/(1-\alpha_2) \end{cases}$$

$$\begin{cases} \alpha_1=\frac{8}{15}, & \alpha_2=\frac{2}{3} & \\ \beta_1=\alpha_1=\frac{8}{15}, & \beta_2=\gamma_1=\frac{1}{4}, & \beta_3=\frac{5}{12} \\ \gamma_1=\frac{1}{4}, & \gamma_2=0, & \gamma_3=\frac{3}{4} \\ \zeta_1=0, & \zeta_2=\beta_2-\beta_1=\frac{-17}{60}, & \zeta_3=-\beta_3=-\frac{5}{12} \end{cases}$$

Appendix D Input Files of Transcat

Table D.1 Input file "Run.txt"

```
Number of runs, nRun =
4
List of runs, RunList =
Constant
Sine
Square
Sawtooth
```

Table D.2 Input file "Input.txt"

```
Description of this run (500 characters maximum), runDescription =
This is an example run for Transcat of LHHW CO oxidation under constant CO feed.
Model of the reactor (1: Dynamic diffusion reactor (DDR), 2: Continuous stir tank reactor (CSTR)), reactorType =
1
Thermal property of this run (1: Isothermal, 2: Temperature ramp, 3: Non-isothermal), thermalType =
1
Numerical method for this run (EE,RK2,RK3,RK4,IE,BDF2,CNRK3,LOA), numericalMethod =
CNRK3
Use the algorithm of element constraint method (0:don't use, 1:catalytic sites only, 2:use ecm algorithm), ecm =
1
Initial time (s), tiRun =
0.000000000000000E+000
Final time (s), tfRun =
2.500000000000000E+002
Number of time step, nts =
1000
Number of gas component, ngc =
3
Number of solid component, nsc =
2
Number of catalytic site, ncs =
1
Chemical species, Component =
CO_(_g_),O_2_(_g_),CO_2_(_g_),CO*S_(_s_),O*S_(_s_),S_(_s_)
Molecular weight of gas component (g/gmol), MWGas =
2.800000000000000E-002,3.200000000000000E-002,4.400000000000000E-002
Molecular weight of reference gas (g/gmol), rMW =
2.800000000000000E-002
Effective diffusion coefficient of reference gas (m^2/s), rDeff =
4.000000000000000E-007
Temperature of reference gas (K), rTemp =
4.231500000000000E+002
Initial temperature (K), iniT =
4.231500000000000E+002
Reactor diameter (m), ld =
4.000000000000000E-003
```

Table D.2 Continued

Reactor thickness (m), lz =
6.400000000000000E-004
Number of grid in Z-direction (for DDR only), nGrid =
256
Porosity of reactor (dimensionless), catP =
3.000000000000000E-001
Weight of reactor (kg), catW =
2.283000000000000E-005
Weight fraction of catalyst (dimensionless), catWF =
2.000000000000000E-002
Molecular weight of catalyst (kg/gmol), catMW =
1.950840000000000E-001
Surface area occupied per mole catalyst on flat surface (m^2/gmol), catS =
5.500000000000000E+004
Dispersion of catalyst (fraction of catalyst exposed, gmol catalyst atom on surface/gmol catalyst atom in a particle), catD =
1.000000000000000E-003
Fraction of catalytic site, such as linear, bridgebound or triply coordinated, to catalyst (gmol catalytic site on surface/gmol catalyst on surface+, catSF =
1.000000000000000E-000
Number of chemical element, nce =
3
Chemical element, Element =
C,O,S
Formula matrix, M =
1,0,1,1,0,0
1,2,2,1,1,0
0,0,0,1,1,1
Number of reaction with elementary rate equation, nrer =
1
Detailed mechanism of reaction with elementary rate equation (Note: If nrer is 0, leave one extra line underneath.), MechE =
-1, 0, 0, 1, 0,-1
Input elementary rate constant for Arrhenius' equation (Af, Nf, Ef, Ar, Nr, Er) (Note: If nrer is 0, leave one extra line underneath.), ERC =
3.780000000000000E+002,5.000000000000000E-001,0.000000000000000E+000,1.000000000000000E+010,0.000000000000000E+000,8.400000000000000E+004
Number of algebraic rate equation, nrar =
1
Detailed mechanism of reaction with algebraic rate equation (Note: If nrar is 0, leave one extra line underneath.), MechA =
0, 0, 1,-1,-1, 2
Input algebraic rate constant for Arrhenius' equation (Af, Nf, Ef, Ar, Nr, Er) (Note: If nrar is 0, leave one extra line underneath.), ARC =
8.000000000000000E+015,0.000000000000000E+000,6.300000000000000E+004,0.000000000000000E+000,0.000000000000000E+000,0.000000000000000E+000
Order of component for algebraic rate equation (forward and reverse rate) (Note: If nrar is 0, leave one extra line underneath.), ARO =
0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0
Number of user-defined rate equation, nrur =
1

Table D.2 Continued

Detailed mechanism of reaction with user-defined rate equation (Note: If nrur is 0, leave one extra line underneath.), MechU =
0,-1, 0, 0, 2,-2
Input user-defined rate constant for Arrhenius' equation (Af, Nf, Ef, Ar, Nr, Er) (Note: If nrur is 0, leave one extra line underneath.), URC =
6.200000000000000E-000,5.000000000000000E-001,0.000000000000000E+000,0.000000000000000E+000,0.000000000000000E+000,0.000000000000000E+000
Input initial concentration from a file (0:No, 1:Yes), inputIC =
0
Input filename of initial concentration, inputFIC =
InitialConcentration.txt
Input initial temperature from a file (0:No, 1:Yes), inputIT =
0
Input filename of initial temperature, inputFIT =
FinalTemperature.txt
Initial gas pressure (Pa), IniGP =
0.000000000000000E+000,1.100000000000000E+002,0.000000000000000E+000
Initial surface converage (Dimensionless), IniSC =
0.000000000000000E-000,5.000000000000000E-001,5.000000000000000E-001
Concentration boundary condition at Z=0 and 1 (1: Modulated (Dirichlet-type), 2: Zero mass flux (von Neumann-type), 3: Balanced mass flux (Danckwerts' type)), CBC =
3,3
Maximum pressure of gas component at Z=0 and 1 (Pa), BCPmax =
0.000000000000000E+000,0.000000000000000E+000,0.000000000000000E+000
1.650000000000000E+001,1.100000000000000E+002,0.000000000000000E+000
Minimum pressure of gas component at Z=0 and 1 (Pa), BCPmin =
0.000000000000000E+000,0.000000000000000E+000,0.000000000000000E+000
0.000000000000000E+000,0.000000000000000E+000,0.000000000000000E+000
Waveform of gas component at Z=0 and 1 (1:Constant, 2:Sine, 3:Square, 4:Sawtooth), BCW =
1,1,1
1,1,1
Frequence of modulated gas component at Z=0 and 1 (Hz), BCF =
0.000000000000000E+000,0.000000000000000E+000,0.000000000000000E+000
4.000000000000000E-003,0.000000000000000E+000,0.000000000000000E+000
Phase lag of modulated gas component at Z=0 and 1 (0~1 Wavelength), BCL =
0.000000000000000E+000,0.000000000000000E+000,0.000000000000000E+000
0.000000000000000E-001,0.000000000000000E+000,0.000000000000000E+000
Volume of boundary chamber at Z=0 and 1 for Danckwerts' type boundary (m3), BCCV =
4.400000000000000E-008,5.600000000000000E-007
Volume flow rate of reference gas in boundary chamber at Z=0 and 1 for Danckwerts' type boundary (m3/s), BCRQ =
1.900000000000000E-010,7.000000000000000E-007
Molecular weight of reference gas in boundary chamber at Z=0 and 1 for Danckwerts' type boundary (kg/mol), BCRM =
2.018000000000000E-002,2.018000000000000E-002
Temperature of reference gas in boundary chamber at Z=0 and 1 for Danckwerts' type boundary (K), BCRT =
3.080000000000000E+002,3.080000000000000E+002
Initial temperature of boundary chamber at Z=0 and 1 (K), BCT =
4.231500000000000E+002,4.231500000000000E+002
External mass transfer coeffefficient of boundary chamber (m/s), BCEMTC =

Table D.2 Continued

5.000000000000000E-002,5.000000000000000E-002
Temperature boundary condition at Z=0 and 1 for non-isothermal run (0:Isothermal, 1:Modulated (Dirichlet-type), 2:Zero heat flux (von Neumann-type), 3:Balanced heat flux (Danckwerts' type)), TBC =
0,0
Maximum temperature at Z=0 and 1 (Pa), BCTmax =
4.230000000000000E+002,4.230000000000000E+002
Minimum temperature at Z=0 and 1 (Pa), BCTmin =
3.000000000000000E+002,3.000000000000000E+002
Waveform of temperature at Z=0 and 1 (1:Constant, 2:Sine, 3:Square), BCTW =
1,1
Frequence of temperature at Z=0 (Hz), BCTF =
0.000000000000000E+000,0.000000000000000E+000
Phase lag of temperature at Z=0 (0~1 Wavelength), BCTL =
0.000000000000000E+000,0.000000000000000E+000

Appendix E Source Code of Transcat

Table E.1 Transcat.f90

```
!-----------------------------------------------------------------------
! TransCat.f90
! Description: A general purpose numerical simulator for transient gas-solid kinetic in porous catalytic reactor.
! Last update by Hsu-Wen Hsiao on 8/1/10.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! MODULE mBoundary
! Description: Variables and parameter of concentration and temperature boundary.
!-----------------------------------------------------------------------
MODULE mBoundary
IMPLICIT NONE
SAVE
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: BCCmax, BCCmin
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: BCF, BCL
INTEGER,          DIMENSION(:,:), ALLOCATABLE :: BCW
DOUBLE PRECISION, DIMENSION(2) :: BCTmax, BCTmin, BCTF, BCTL, BCEMTC
INTEGER,          DIMENSION(2) :: BCTW
END MODULE mBoundary
!-----------------------------------------------------------------------
! MODULE mDynamic
! Description: Dynamic Variables and parameter
!-----------------------------------------------------------------------
MODULE mDynamic
IMPLICIT NONE
SAVE
! main control parameter
INTEGER :: reactorType, thermalType, ecm
INTEGER, DIMENSION(2) :: CBC, TBC
! component
INTEGER :: ngc, nsc, ncs
INTEGER :: ntsc, ntc
INTEGER :: gStart, gEnd, sStart, sEnd, cStart, cEnd
! element
INTEGER :: iStart, iEnd, dStart, dEnd
INTEGER, DIMENSION(:), ALLOCATABLE :: MDiMYI, MDiMYL, MDiMIYI, MDiMIYL
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: SCtot
! time
DOUBLE PRECISION :: tiRun, tfRun, tRun, dtRun
DOUBLE PRECISION :: tRuntime
INTEGER :: nts
! space
INTEGER :: ngz, rStart, rEnd, nmStart, nmEnd
DOUBLE PRECISION :: dZ
! concentration and temperature
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: C
DOUBLE PRECISION, DIMENSION(:),   ALLOCATABLE :: T
```

Table E.1 Continued

```
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: LC, NC
! numbers
DOUBLE PRECISION :: alpha
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: Tau, TauB
DOUBLE PRECISION, PARAMETER :: idealGC=8.314472D0
! adaptive control
LOGICAL :: switchMethod
INTEGER :: dtLevel, dtLevelOld
LOGICAL :: tooStiff
INTEGER, PARAMETER :: maxdtLevel=30                    ! maximum dtLevel for adaptive control
INTEGER, PARAMETER :: nst=10                           ! number of sub-timestep for adaptive control
DOUBLE PRECISION, PARAMETER :: tolMinC=0.0D0,  tolMinT=0.0D0 ! tolerance of minimum concentration and temperature
DOUBLE PRECISION, PARAMETER :: tolMaxDC=1.0D-4         ! tolerance of maximum C difference 10^-4 (adjust this if not smooth)
DOUBLE PRECISION, PARAMETER :: tolMaxDT=1.0D-2         ! tolerance of maximum T difference (adjust this if not smooth)
END MODULE mDynamic
!-----------------------------------------------------------------------------------------------------------------------
! MODULE Element
! Description: Variables and parameters for element constraint method.
!-----------------------------------------------------------------------------------------------------------------------
MODULE mElement
IMPLICIT NONE
SAVE
INTEGER :: nce, nic, ndc
INTEGER, DIMENSION(:,:), ALLOCATABLE :: M, MIMD, MI, MD
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: MDi, MDiM, MDiMI
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: MDiMY, MDiMIY
INTEGER, DIMENSION(:), ALLOCATABLE :: Msc, MscI, MscL
END MODULE mElement
!-----------------------------------------------------------------------------------------------------------------------
! MODULE mInformation
! Description: Information for this run
!-----------------------------------------------------------------------------------------------------------------------
MODULE mInformation
IMPLICIT NONE
SAVE
CHARACTER(LEN=8) :: verTransCat="2.1"
CHARACTER(LEN=32) :: verDate="October 18, 2010"
LOGICAL :: stopRun
CHARACTER(LEN=128) :: stopInfo
CHARACTER(LEN=128) :: TransCatDir
INTEGER :: nRun, iRun
CHARACTER(LEN=128), DIMENSION(:), ALLOCATABLE :: Runlist
CHARACTER(LEN=128) :: numericalMethod, run
CHARACTER(LEN=500) :: runDescription
INTEGER :: inputIC, inputIT
CHARACTER(LEN=128) :: inputFIC, inputFIT
```

Table E.1 Continued

```
END MODULE mInformation
!-----------------------------------------------------------------------
! MODULE mParameter
! Description: Parameter for each run
!-----------------------------------------------------------------------
MODULE mParameter
IMPLICIT NONE
SAVE
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: IniGP, IniSC, MWGas
CHARACTER(LEN=128), DIMENSION(:), ALLOCATABLE :: Component
CHARACTER(LEN=128), DIMENSION(:), ALLOCATABLE :: Element
DOUBLE PRECISION :: rMW,  rDeff, rTemp, iniT
DOUBLE PRECISION :: lz, ld, catP, catW
DOUBLE PRECISION, PARAMETER :: pi=3.14159265358979300
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: catWF, catMW, catS, catD, catSF
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: catSC, catCS
INTEGER :: nGrid, ncr
INTEGER, DIMENSION(:,:), ALLOCATABLE :: MechE, MechA, MechU, Mech
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: ERC, ARC, URC, ARO, EAURC
INTEGER, DIMENSION(:,:), ALLOCATABLE :: RIM
INTEGER, DIMENSION(:), ALLOCATABLE :: RIV
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: BCPmax, BCPmin
DOUBLE PRECISION, DIMENSION(2) :: BCCV, BCRQ, BCRM, BCRT
DOUBLE PRECISION, DIMENSION(2) :: BCT
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: Z
DOUBLE PRECISION :: maxGC, totSC, totCS, totSA, totRCA, totRV, totGV, totSV
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: Deff
END MODULE mParameter
!-----------------------------------------------------------------------
! MODULE mReaction
! Description: Reaction variables and parameters
!-----------------------------------------------------------------------
MODULE mReaction
IMPLICIT NONE
SAVE
INTEGER :: ntr, near, nrer, nrar, nrur
INTEGER, DIMENSION(:,:), ALLOCATABLE :: S
INTEGER, DIMENSION(:), ALLOCATABLE :: SY, SYI, SYL
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: AC, R, RC
INTEGER, DIMENSION(:), ALLOCATABLE :: RI, RL
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: RP
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: DRDC
INTEGER, DIMENSION(:), ALLOCATABLE :: JYI, JYL
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: JR
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: JRY
DOUBLE PRECISION, DIMENSION(:,:,:), ALLOCATABLE :: JUR ! reaction part of Jacobian (for user defined rate)
```

Table E.1 Continued

```
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: JRC, JRP
INTEGER, DIMENSION(:), ALLOCATABLE :: JRI, JRL
INTEGER, DIMENSION(:), ALLOCATABLE :: SJRI, SJRL, SJYI, SJYRL, SJYCL, SJY
INTEGER :: erStart, erEnd, arStart, arEnd, urStart, urEnd
END MODULE mReaction
!-----------------------------------------------------------------------------------------------------------------------
! MODULE mUserDefine
! Description: Variables and parameters for user-defined rate equation
!-----------------------------------------------------------------------------------------------------------------------
MODULE mUserDefine
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: UDR
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: UDRD
LOGICAL, DIMENSION(:,:), ALLOCATABLE :: nzUDRD
INTEGER :: nudr
END MODULE mUserDefine
!-----------------------------------------------------------------------------------------------------------------------
! MODULE mRuntime
! Description: Runtime variables
!-----------------------------------------------------------------------------------------------------------------------
MODULE mRuntime
IMPLICIT NONE
SAVE
DOUBLE PRECISION :: iCPUTime, fCPUTime, dCPUTime, timeLeft
INTEGER :: iRuntime
END MODULE mRuntime
!-----------------------------------------------------------------------------------------------------------------------
! MODULE mSave
! Description: Variables to save file
!-----------------------------------------------------------------------------------------------------------------------
MODULE mSave
IMPLICIT NONE
SAVE
CHARACTER(LEN=128) :: formatRuntime, formatTime, formatC, formatR, formatT, formatDeff, formatTau, formatRC
INTEGER, DIMENSION(:), ALLOCATABLE :: UnitC, UnitR, UnitDeff, UnitTau, UnitRC
CHARACTER(LEN=256), DIMENSION(:), ALLOCATABLE :: FileC, FileR, FileDeff, FileTau, FileRC
CHARACTER(LEN=256) :: fileRuntime, fileTime, fileT
INTEGER :: unitRuntime, unitTime, unitT
END MODULE mSave
!-----------------------------------------------------------------------------------------------------------------------
! Editor's note:
! Thomas algorithm
! Balance of charge in ECM
! Version 3.0
! Version 4.0
! Multigrid and relaxation method for higher dimension
!-----------------------------------------------------------------------------------------------------------------------
```

Table E.1 Continued

```
PROGRAM TransCat
USE mInformation
USE mDynamic
IMPLICIT NONE
CHARACTER(LEN=256), EXTERNAL :: fInsInt2 !, fInsInt
WRITE(6,fInsInt2("(A16,1X,A",len(trim(verTransCat)),",1X,'(',A",len(trim(verDate)),",')')")) &
    "TransCat version", trim(verTransCat), trim(verDate)
WRITE(6,"(A41)") "Copyright 2009 UCSD. All rights reserved."
CALL sRead("Run.txt") ! Read File Run.txt
DO iRun=1,nRun
   CALL sInitialize
   switchMethod=.False.;
   SELECT CASE (numericalMethod)
      CASE ("LOA")   ! Adaptively switching between low order methods (EE and IE)
         CALL sLOA(tfRun,nts)
      CASE ("MOA")   ! Adaptively switching between medium order methods (RKW3, CNRKW3 and BDF3)
         CALL sMOA(tfRun,nts)
      CASE ("HOA")   ! Adaptively switching between medium order methods (RK5 and BDF5)
         CALL sHOA(tfRun,nts)
      CASE ("EE")    ! Explicit Euler method
         CALL sEE(tfRun,nts)
      CASE ("RK2")   ! Second order Runge-Kutta method
         CALL sRK2(tfRun,nts)
      CASE ("RK3")   ! Third order Runge-Kutta method
         CALL sRK3(tfRun,nts)
      CASE ("RK4")   ! Fourth order Runge-Kutta method
         CALL sRK4(tfRun,nts)
      CASE ("RK5")   ! Fifth order Runge-Kutta method
         CALL sRK5(tfRun,nts)
      CASE ("CNRK3") ! Crank-Nicholson and third order Runge-Kutta method
         CALL sCNRK3(tfRun,nts)
      CASE ("IE")    ! Implicit Euler method
         CALL sIE(tfRun,nts)
      CASE ("BDF2")  ! Second order backward differential formula
         CALL sBDF2(tfRun,nts)
      CASE ("BDF3")  ! Third order backward differential formula
         CALL sBDF3(tfRun,nts)
      CASE ("BDF4")  ! Fourth order backward differential formula
         CALL sBDF4(tfRun,nts)
      CASE ("BDF5")  ! Fifth order backward differential formula
         CALL sBDF5(tfRun,nts)
      CASE DEFAULT   ! Adaptive switching between medium order method (RKW3, CNRKW3 and BDF3)
         CALL sMOA(tfRun,nts)
   END SELECT
   CALL sFinalize
END DO
```

Table E.2 sInitialize.f90

```
!-----------------------------------------------------------------------
! Subroutine sInitialize
! Description: Initialize variables and parameters for TransCat.
! Created by Hsu-Wen Hsiao on 2/7/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
SUBROUTINE sInitialize
USE mInformation
USE mDynamic
USE mParameter
USE mElement
USE mBoundary
USE mReaction
USE mRuntime
USE mSave
USE mUserDefine
IMPLICIT NONE
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
INTEGER, EXTERNAL :: fRank, fCountC
INTEGER :: i, j, k, l, a, nonzero
run=RunList(iRun); ! initialize parameters for this run ----------------------------------------------------------
WRITE(6,fInsInt("(/A10,1X,A",len_trim(run),",1X,A6)")) "Initialize", run, "......"
CALL sRead("Input.txt") ! Read file Input.txt from run folder
ntsc=nsc+ncs+ncs;  ntc=ngc+nsc+ncs; ! initialize parameter for component ----------------------------------------------
gStart=1;       gEnd=gStart+ngc-1;
sStart=gEnd+1; sEnd=sStart+nsc-1;
cStart=sEnd+1; cEnd=cStart+ncs-1;
!WRITE(6,"(5(1X,A5,1X,I3,','))") "ngc=", ngc, "nsc=", nsc, "ncs=", ncs, "ntsc=", ntsc, "ntc=", ntc
!WRITE(6,"(6(1X,A7,1X,I3,','))") "gStart=",gStart,"gEnd=",gEnd,"sStart=",sStart,"sEnd=",sEnd,"cStart=",cStart,"cEnd=",cEnd
SELECT CASE (reactorType) ! define spatial grid and time variable ------------------------------------------------
  CASE (1) ! DDR
    ngz=nGrid;
  CASE (2) ! CSTR
    nGrid=1; ! CSTR is a lumped system
    ngz=1;
END SELECT
dZ=1.0D0/ngz;
SELECT CASE (CBC(1)) ! left BC
  CASE (1) ! Dirichlet-type BC
    ngz=ngz+1;
    rStart=2; ! starting grid of reactor
  CASE (2) ! von Neumann-type BC
    ngz=ngz+1;
    rStart=2;
  CASE (3) ! Danckwerts' type BC
    ngz=ngz+2;
```

Table E.2 Continued

```
   rStart=3;
END SELECT
SELECT CASE (CBC(2)) ! right BC
  CASE (1) ! Dirichlet-type BC
    ngz=ngz+1;
  CASE (2) ! von Neumann-type BC
    ngz=ngz+1;
  CASE (3) ! Danckwerts' type BC
    ngz=ngz+2;
END SELECT
rEnd=rStart+nGrid-1; ! ending grid of reactor
nmStart=2; ! starting grid for numerical integration
nmEnd=ngz-1; ! ending grid for numerical integration
ALLOCATE(Z(ngz));
Z(rStart)=0.5D0*dZ; ! half grid to left boundary
DO j=rStart-1,1,-1; Z(j)=Z(j+1)-dZ; END DO ! grid before the starting grid of reactor
DO j=rStart+1,ngz,+1; Z(j)=Z(j-1)+dZ; END DO ! grid after the ending grid of reactor
tRun=tiRun; ! initial time
dtRun=(tfRun-tiRun)/nts; ! time step size
dtLevel=1;      ! parameter for adaptive time control
dtLevel0ld=dtLevel; ! last dtLevel of adaptive time control
!WRITE(6,"(3(1X,A8,1X,I3,',',))") "CBC(1)= ", CBC(1), "rStart=", rStart, "nmStart=", nmStart
!WRITE(6,"(3(1X,A8,1X,I3,',',))") "CBC(2)= ", CBC(2), "rEnd=", rEnd, "nmEnd=", nmEnd
!WRITE(6,"(14X,2(1X,A8,1X,I3,',',))") "nGrid=", nGrid, "ngz=", ngz
!WRITE(6,"(1X,A3,1X,1P1E10.3E2)") "dZ=", dZ
!WRITE(6,fInsInt("(1X,A6,1X,",ngz-1,"(f7.4,',',f7.4)")) "Z=    ", Z
!WRITE(6,"(4(1X,A6,1P1E9.2E2,',',),1X,A6,I4,',',)") "tiRun=", tiRun, "tfRun", tfRun, "tRun=", tRun, "dtRun=", dtRun
!WRITE(6,"(1X,A8,I4,',',1X,A8,I4)") "nts=", nts, "dtLevel=", dtLevel
ALLOCATE(C(ngz,ntc),T(ngz)) ! initialize concentration and temperature array ----------------------------------------------------------------|
DO j=1,ngz; DO i=1,ntc; C(j,i)=0.0D0; END DO; T(j)=0.0D0; END DO
DO j=1,ngz; C(j,gStart:gEnd)=IniGP(1:ngc); END DO ! use IniGP as initial condition
DO j=rStart,rEnd; C(j,sStart:cEnd)=IniSC(1:ntsc); END DO; ! use IniSC as initial condition
DO j=1,ngz; T(j)=iniT; END DO ! use iniT as initial condition
ALLOCATE(LC(ngz,ntc),NC(ngz,ntc)) ! initialize linear part and nonlinear part for storage
DO i=1,ntc; DO j=1,ngz; LC(j,i)=0.0D0; NC(j,i)=0.0D0; END DO; END DO
IF (inputIC) THEN ! overwrite initial concentration from file (gas pressure and surface coverage)
  CALL sRead("Initial Concentration")
  CALL sCheck("Initial Concentration")
END IF
IF (inputIT) THEN ! overwrite initial temperature from file
  CALL sRead("Initial Temperature");
  CALL sCheck("Initial Temperature")
END IF
!WRITE(6,fInsInt2("(1X,A3/,",ngz,"(1X,",ntc-1,"(1P1E10.2E3,',',),1P1E10.2E3/)\)")) "C =", transpose(C)
!WRITE(6,fInsInt("(1X,A3/,1X,",ngz-1,"(1P1E10.2E3,',',),1P1E10.2E3)")) "T =", T
!WRITE(6,fInsInt2("(1X,A4/,",ngz,"(1X,",ntc-1,"(1P1E10.2E3,',',),1P1E10.2E3/)\)")) "LC =", transpose(LC)
```

Table E.2 Continued

```
!WRITE(6,fInsInt2("(1X,A4/,",ngz,"(1X,",ntc-1,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "NC =", transpose(NC)
maxGC=0.0D0; ! derive maximum gas concentration, maxGC, and nondimensionlize gas concentration ----------------------------------------|
DO j=1,2
  SELECT CASE (CBC(j))
    CASE (1) ! Dirichlet-type
       DO i=1,ngc; maxGC=MAX(maxGC,BCPmax(j,i)/idealGC/BCT(j)); END DO
    CASE (3) ! Danckwerts' type
       DO i=1,ngc; maxGC=MAX(maxGC,BCPmax(j,i)/idealGC/BCT(j)); END DO
  END SELECT
END DO
IF (abs(maxGC) <= 0.0D0) THEN ! Batch reactor (von Neumann-type boundary on both boundary)
  DO i=gStart,gEnd; DO j=1,ngz; maxGC=MAX(maxGC,C(j,i)/idealGC/T(j)); END DO; END DO
END IF
CALL sCheck("maxGC")
DO i=gStart,gEnd
  DO j=1,ngz
    C(j,i)=(C(j,i)/idealGC/T(j))/maxGC; ! nondimensionlize maximum gas concentration of modulation
  END DO
END DO
IF (CBC(1)==1 .OR. CBC(1)==3 .OR. CBC(2)==1 .OR. CBC(2)==3) THEN ! intialize maximum and minimum concentration for modulation
  ALLOCATE(BCCmax(2,ngc),BCCmin(2,ngc)); DO j=1,ngc; DO k=1,2; BCCmax(k,j)=0.0D0; BCCmin(k,j)=0.0D0; END DO; END DO
  DO i=1,ngc
    DO j=1,2
      BCCmax(j,i)=(BCPmax(j,i)/idealGC/BCT(j))/maxGC; ! nondimensionlize maximum gas concentration of modulation
      BCCmin(j,i)=(BCPmin(j,i)/idealGC/BCT(j))/maxGC;  ! nondimensionlize minimum gas concentration of modulation
    END DO
  END DO
END IF
CALL sBoundary ! update boundary
!WRITE(6,"(1X,A7,1X,1P1E10.2E3)") "maxGC =", maxGC
!WRITE(6,fInsInt2("(1X,A3/,",ngz,"(1X,",ntc-1,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "C =", transpose(C)
!WRITE(6,fInsInt2("(1X,A8/,",2,"(1X,",ngc-1,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "BCCmax =", transpose(BCCmax)
!WRITE(6,fInsInt2("(1X,A8/,",2,"(1X,",ngc-1,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "BCCmin =", transpose(BCCmin)
ALLOCATE(catSC(ncs),catCS(ncs)) ! parameter of catalyst ----------------------------------------------------------------------------|
DO i=1,ncs
  catSC(i)=catSF(i)*catD(i)*(1.0D0/catS(i)); ! surface concentration of catalytic sites, (gmol/m^2)
  catCS(i)=catSF(i)*catD(i)*(catWF(i)*catW/catMW(i)); ! surface catalytic sites in reactor, (gmol)
END DO
totSC=SUM(catSC); ! total surface concentration of catalytic sites, (gmol/m^2)
totCS=SUM(catCS); ! total surface catalytic sites in reactor, (gmol)
totSA=totSC/totCS; ! total surface area of catalyst, (m^2)
totRCA=0.25*pi*(ld**2); ! total cross-sectional area of reactor, (m^2)
totRV=totRCA*lz; ! total reactor volume, (m^3)
totGV=CatP*totRV;
totSV=totRV-totGV;
alpha=totCS/(totGV*maxGC); ! surface-to-gas capacity ratio
```

Table E.2 Continued

```
!WRITE(6,fInsInt("(1X,A7/,1X,",ncs-1,"(1P1E10.2E3,',',1P1E10.2E3)")) "catSC =", catSC
!WRITE(6,fInsInt("(1X,A7/,1X,",ncs-1,"(1P1E10.2E3,',',1P1E10.2E3)")) "catCS =", catCS
!WRITE(6,"(2(1X,A7,1X,1P1E10.2E3,',',1X,A7,1X,1P1E10.2E3)") "totSC =", totSC, "totCS =", totCS, "totSA =", totSA
!WRITE(6,"(3(1X,A7,1X,1P1E10.2E3,',',1X,A7,1X,1P1E10.2E3)") "totRCA=",totRCA, "totRV =",totRV, "totGV =",totGV, "totSV =",totSV
!WRITE(6,"(1X,A7,1X,1P1E10.2E3)") "alpha =", alpha
ALLOCATE(Deff(ngz,ngc),Tau(ngz,ngc)) ! parameter of diffusion ----------------------------------------------------------------|
DO i=1,ngc; DO j=1,ngz; Deff(j,i)=0.0D0; Tau(j,i)=0.0D0; END DO; END DO
IF (CBC(1)==3 .OR. CBC(2)==3) THEN
  ALLOCATE(TauB(2,ngc))
  DO i=1,ngc; DO j=1,2; TauB(j,i)=0.0D0; END DO; END DO
END IF
CALL sDiffusion ! update Deff according to temperture
CALL sTau ! update Tau according to temperture
!WRITE(6,fInsInt2("(1X,A6/,",ngz,"(1X,",ngc-1,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "Deff =", transpose(Deff)
!WRITE(6,fInsInt2("(1X,A5/,",ngz,"(1X,",ngc-1,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "Tau =", transpose(Tau)
!WRITE(6,fInsInt2("(1X,A6/,",2,"(1X,",ngc-1,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "TauB =", transpose(TauB)
ncr=nrer+nrar+nrur; ! number of chemical reaction. ! initialize reaction parameters ----------------------------------------|
ALLOCATE(Mech(ncr,ntc),EAURC(ncr,6))
IF (nrer > 0) THEN
  Mech( 1:nrer,:)=MechE(:,:); ! combine mechanism of elementary, algebraic and user-defined rate equation
  EAURC(1:nrer,:)=  ERC(:,:); ! combine rate constant of elementary, algebraic and user-defined rate equation
END IF
IF (nrar > 0) THEN
  Mech(1+nrer:nrer+nrar,:)=MechA(:,:);
  EAURC(1+nrer:nrer+nrar,:)=ARC(:,:);
END IF
IF (nrur > 0) THEN
  Mech( 1+nrer+nrar:nrer+nrar+nrur,:)=MechU(:,:);
  EAURC(1+nrer+nrar:nrer+nrar+nrur,:)=  URC(:,:);
END IF
!WRITE(6,"(3(1X,A6,1X,I3,',',1X,A5,1X,I3)") "nrer =", nrer, "nrar =", nrar, "nrur =", nrur, "ncr =", ncr
!WRITE(6,fInsInt2("(1X,A7/,",nrer,"(1X,",ntc-1,"(I4,',',I4/)\)")) "MechE =", transpose(MechE)
!WRITE(6,fInsInt2("(1X,A7/,",nrar,"(1X,",ntc-1,"(I4,',',I4/)\)")) "MechA =", transpose(MechA)
!WRITE(6,fInsInt2("(1X,A7/,",nrur,"(1X,",ntc-1,"(I4,',',I4/)\)")) "MechU =", transpose(MechU)
!WRITE(6,fInsInt2("(1X,A6/,",ncr,"(1X,",ntc-1,"(I4,',',I4/)\)")) "Mech =", transpose(Mech)
!WRITE(6,fInsInt2("(1X,A5/,",nrer,"(1X,",5,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "ERC =", transpose(ERC)
!WRITE(6,fInsInt2("(1X,A5/,",nrar,"(1X,",5,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "ARC =", transpose(ARC)
!WRITE(6,fInsInt2("(1X,A5/,",nrur,"(1X,",5,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "URC =", transpose(URC)
!WRITE(6,fInsInt2("(1X,A7/,",ncr,"(1X,",5,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "EAURC =", transpose(EAURC)
ALLOCATE(RIM(2,ncr)) ! initialize reaction idendity matrix ------------------------------------------------------------------|
erStart=0; erEnd=0; ! initialize starting and ending position of elementary rate equations
arStart=0; arEnd=0; ! initialize starting and ending position of algebraic rate equations
urStart=0; urEnd=0; ! initialize starting and ending position of user-defined rate equations
nonzero=0; ! a dummy counter for nonzero entries in an array
ntr=0; ! initialize number of total rate
near=0; ! number of elementary and aalgebraic rate
```

Table E.2 Continued

```
DO i=1,ncr
  DO j=1,2
    IF (EAURC(i,1+3*(j-1))>0) THEN
      ntr=ntr+1; ! computer the number of total rate (it's actually the sum of elementary, algebraic and user-defined rate)
      IF (i > 0 .AND. i <= nrer+nrar) THEN
        near=near+1;
      END IF
      RIM(j,i)=ntr; ! construt reaction idendity matrix
      !DO k=1,ntc
      !  l=((-1)**j)*Mech(i,k);
      !  IF (l>0) THEN
      !    IF (i>0 .AND. i <= nrer) THEN
      !        nonzero=nonzero+l; ! calculate nonzero entries for RI and RJ
      !        ELSEIF (i>nrer .AND. i <= (nrer+nrar)) THEN
      !          nonzero=nonzero+1;
      !    END IF
      !  END IF
      !END DO
        IF (i>0 .AND. i <= nrer) THEN
          DO k=1,ntc
            l=((-1)**j)*Mech(i,k);
            IF (l>0) THEN
            nonzero=nonzero+l; ! calculate nonzero entries for RI and RJ
          END IF
          END DO
        ELSEIF (i>nrer .AND. i <= (nrer+nrar)) THEN
          DO k=1,ntc
          l=ARO(i-nrer,k+(j-1)*ntc);
          IF (l/=0) THEN
            nonzero=nonzero+1;
          END IF
        END DO
        END IF
      ELSE
        RIM(j,i)=0; ! not included if rate constant is zero
      END IF
    END DO
    IF (i > 0 .AND. i <= nrer) THEN ! define the position of elementary, algebraic and user-defined rate
      erEnd=max(erEnd,ntr);
    ELSEIF (i > nrer .AND. i<= (nrer+nrar)) THEN
      arEnd=max(arEnd,ntr);
    ELSEIF (i > (nrer+nrar) .AND. i<= (nrer+nrar+nrur)) THEN
      urEnd=max(urEnd,ntr);
    END IF
  END DO
  erStart=1;
```

Table E.2 Continued

```
IF (nrer == 0) THEN; erEnd=erStart-1; END IF
arStart=erEnd+1;
IF (nrar == 0) THEN; arEnd=arStart-1; END IF
urStart=arEnd+1;
IF (nrur == 0) THEN; urEnd=urStart-1; END IF
IF (nrur > 0) THEN;
  nudr=urEnd-urStart+1;
  ALLOCATE(UDR(nudr),UDRD(nudr,ntc),nzUDRD(nudr,ntc),JUR(nudr,ntc,ngz))
  DO k=1,nudr
       UDR(k)=0.0D0;
       DO i=1,ntc
         UDRD(k,i)=0.0D0;
         nzUDRD(k,i)=0;
         DO j=1,ngz
           JUR(k,i,j)=0.0D0;
         END DO
    END DO
  END DO
END IF
!WRITE(6,"(1X,A5,1X,I3,',',1X,A9,1X,I3)") "ntr =", ntr, "nonzero =", nonzero
!WRITE(6,fInsInt2("(1X,A5/,",2,"(1X,",ncr-1,"(I4,','),I4/)\)")) "RIM =", transpose(RIM)
!WRITE(6,"(1X,A9,1X,I3,',',1X,A9,1X,I3)") "erStart =", erStart, "erEnd=", erEnd
!WRITE(6,"(1X,A9,1X,I3,',',1X,A9,1X,I3)") "arStart =", arStart, "arEnd=", arEnd
!WRITE(6,"(1X,A9,1X,I3,',',1X,A9,1X,I3)") "urStart =", urStart, "urEnd=", urEnd
ALLOCATE(RIV(ntr)); ! construt reaction idendity vector ---------------------------------------------------------------------------|
k=0;
DO i=1,ncr
  DO j=1,2
    IF (RIM(j,i)/=0) THEN
         k=k+1;
         RIV(k)=i*((-1)**(j-1)); ! construt reaction idendity vector
       END IF
  END DO
END DO
ALLOCATE(RI(near+1),RL(nonzero),RP(nonzero))
DO i=1,urStart-erStart+1; RI(i)=0; END DO
DO i=1,nonzero; RL(i)=0; RP(i)=0.0D0; END DO
k=0;
RI(1)=k+1;
DO i=erStart,erEnd
  DO j=1,ntc
    IF (Mech(abs(RIV(i)),j)*RIV(i)<0) THEN
         DO l=1,abs(Mech(abs(RIV(i)),j))
           k=k+1;
           RL(k)=j; ! construct RL
           RP(k)=1.0D0; ! construct RP (power of elementary rate is always 1)
```

Table E.2 Continued

```
    END DO
  END IF
END DO
RI(i+1)=k+1; ! construct RI
END DO
DO i=arStart,arEnd
  IF (RIV(i) > 0) THEN ! forward reaction
    DO j=1,ntc
    !IF (Mech(abs(RIV(i)),j)*RIV(i)<0) THEN
      !  k=k+1;
      !  RL(k)=j; ! construct RL
      !  IF (RIV(i) > 0) THEN ! forward reaction
      !    RP(k)=ARO(abs(RIV(i))-nrer,j); ! construct RP (power for algebraic rate)
      !  ELSEIF (RIV(i) < 0) THEN ! reverse reaction
      !    RP(k)=ARO(abs(RIV(i))-nrer,ntc+j);
      !  END IF
      !END IF
      IF (ARO(abs(RIV(i))-nrer,j)/=0) THEN
          k=k+1;
          RL(k)=j; ! construct RL
        RP(k)=ARO(abs(RIV(i))-nrer,j); ! construct RP (power for algebraic rate)
          !IF (RIV(i) > 0) THEN ! forward reaction
          !  RP(k)=ARO(abs(RIV(i))-nrer,j); ! construct RP (power for algebraic rate)
          !ELSEIF (RIV(i) < 0) THEN ! reverse reaction
          !  RP(k)=ARO(abs(RIV(i))-nrer,ntc+j);
          !END IF
        END IF
    END DO
  ELSEIF (RIV(i) < 0) THEN ! reverse reaction
    DO j=ntc+1,ntc+ntc
      IF (ARO(abs(RIV(i))-nrer,j)/=0) THEN
          k=k+1;
          RL(k)=j-ntc; ! construct RL
        RP(k)=ARO(abs(RIV(i))-nrer,j); ! construct RP (power for algebraic rate)
        END IF
    END DO
  END IF
  RI(i+1)=k+1; ! construct RI
END DO
!WRITE(6,fInsInt("(1X,A5/,1X,",ntr-1,"(I4,',',),I4)")) "RIV =", RIV
!WRITE(6,fInsInt("(1X,A4/,1X,",ntr+1-1,"(I4,',',),I4)")) "RI =", RI
!WRITE(6,fInsInt("(1X,A4/,1X,",RI(near+1)-1-1,"(I4,',',),I4)")) "RL =", RL
!WRITE(6,fInsInt("(1X,A4/,1X,",RI(near+1)-1-1,"(1P1E10.2E3,',',),1P1E10.2E3)")) "RP =", RP
ALLOCATE(AC(3,ntr)) ! initialize Arrehnius Coefficient array, (pre-exponential,temperature order,activation energy) ----------------|
DO i=1,ntr
  IF     (RIV(i) > 0) THEN
```

Table E.2 Continued

```
    AC(1:3,i)=EAURC(abs(RIV(i)),1:3);
  ELSEIF (RIV(i) < 0) THEN
    AC(1:3,i)=EAURC(abs(RIV(i)),4:6);
  END IF
END DO
!WRITE(6,fInsInt("(1X,A5/,1X,",ntr-1,"(I4,',
'),I4)")) "RIV =", RIV
!WRITE(6,fInsInt2("(1X,A7/,",ncr,"(1X,",5,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "EAURC =", transpose(EAURC)
!WRITE(6,fInsInt2("(1X,A4/,",ntr,"(1X,",2,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "AC =", AC
DO i=1,min(ntr,near) ! normalize pre-exponential factor of Arrehnius equation to the dimension (1/s)
  AC(1,i)=AC(1,i)/totSC;
  DO j=RI(i),RI(i+1)-1
    IF     (RL(j) >= gStart .AND. RL(j) <= gEnd) THEN ! gas component
      AC(1,i)=AC(1,i)*(maxGC**RP(j));
    ELSEIF (RL(j) >= sStart .AND. RL(j) <= cEnd) THEN ! solid component
      AC(1,i)=AC(1,i)*(totSC**RP(j));
    ELSE ! The input use-defined rate should be normalized by user, since no RP is available.
    END IF
  END DO
END DO
ALLOCATE(RC(ntr,ngz),R(ntr,ngz)) ! initilize reaction rate and rate constant array
DO i=1,ngz; DO j=1,ntr; RC(j,i)=0.0D0; R(j,i)=0.0D0; END DO; END DO
CALL sRateConstant ! update rate constant RC by Arrehnius equation
!WRITE(6,fInsInt2("(1X,A4/,",ntr,"(1X,",2,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "AC =", AC
!WRITE(6,fInsInt2("(1X,A4/,",ntr,"(1X,",ngz-1,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "RC =", transpose(RC)
!WRITE(6,fInsInt2("(1X,A3/,",ntr,"(1X,",ngz-1,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "R =", transpose(R)
ALLOCATE(DRDC(ntr,ntc)) ! construct the reaction part of Jacobian matrix----------------------------------------------------------|
DO i=1,ntc; DO j=1,ntr; DRDC(j,i)=0.0D0; END DO; END DO
nonzero=0;
DO i=erStart,erEnd
  DO j=1,ntc
    DO k=RI(i),RI(i+1)-1
      IF (RL(k)==j) THEN
        DRDC(i,j)=DRDC(i,j)+1.0D0; ! construct coefficient for elementary rate
      END IF
    END DO
  END DO
  nonzero=nonzero+fCountC(RL(RI(i):RI(i+1)-1),RI(i+1)-RI(i),ntc); ! count the size for Yale's storage of Jacobian matrix
END DO
DO i=arStart,arEnd
  DO j=1,ntc
    DO k=RI(i),RI(i+1)-1
      IF (RL(k)==j) THEN
        DRDC(i,j)=RP(k); ! construct coefficient for algebraic rate
      END IF
    END DO
  END DO
```

Table E.2 Continued

```
  nonzero=nonzero+fCountC(RL(RI(i):RI(i+1)-1),RI(i+1)-RI(i),ntc); ! count the size of Yale's storage of Jacobian matrix
END DO
!WRITE(6,fInsInt2("(1X,A6/,",ntr,"(1X,",ntc-1,"(1P1E10.2E3,',',),1P1E10.2E3/)\)")) "DRDC =", transpose(DRDC)
!WRITE(6,"(1X,A9,1X,I3)") "nonzero =", nonzero
ALLOCATE(JYI(ntc+1),JYL(nonzero))  ! initialize Yale's format of Jacobian matrix----------------------------------------------------------|
ALLOCATE(JR(ntc,ngz),JRY(nonzero,ngz),JRC(nonzero),JRI(nonzero+1))
DO j=1,ngz; DO i=1,ntc; JR(i,j)=0.0D0; END DO; END DO
k=0;
JYI(1)=k+1;
JRI(1)=1;
nonzero=0;
DO i=1,ntc
  DO j=1,ntr
    IF (abs(DRDC(j,i)) > 0.0D0) THEN
      k=k+1;
      JYL(k)=j; ! construct JYL
      DO l=1,ngz
        JRY(k,l)=0.0D0; ! initialize JRY
      END DO
      JRC(k)=DRDC(j,i); ! put the coefficient into sparse form
      IF (j >= erStart .AND. j <= erEND) THEN ! for elementary rate
        JRI(k+1)=JRI(k)+(RI(j+1)-RI(j)-1); ! construct JRI for elementary rate
        nonzero=nonzero+(RI(j+1)-RI(j)-1); ! count the entries for JRL and JRP
      ELSEIF (j >= arStart .AND. j <= arEND) THEN ! for algebraic rate
        DO l=RI(j),RI(j+1)-1
          IF (RL(l) == i) THEN
            IF(abs(RP(l)-1.0D0) > 0.0D0) THEN ! if the order of component x=RL(l) in derivative dRj/dCx is not zero
              JRI(k+1)=JRI(k)+(RI(j+1)-RI(j)); ! construct JRI for algebraic rate
              nonzero=nonzero+(RI(j+1)-RI(j)); ! count the entries for JRL and JRP
            ELSE
              JRI(k+1)=JRI(k)+(RI(j+1)-RI(j)-1); ! if the order of component x=RL(l) in derivative dRj/dCx is zero
              nonzero=nonzero+(RI(j+1)-RI(j)-1); ! count the entries for JRL and JRP
            END IF
            EXIT
          END IF
        END DO
      END IF
    END IF
  END DO
  JYI(i+1)=k+1;
END DO
ALLOCATE(JRL(nonzero),JRP(nonzero)) ! initialize Yale's sparse storage of derivative of rate
nonzero=0;
DO i=1,ntc
  DO j=erStart,erEnd
    DO k=RI(j),RI(j+1)-1
```

Table E.2 Continued

```
    IF (RL(k) == i) THEN
      DO l=RI(j),k-1
        nonzero=nonzero+1;
        JRL(nonzero)=RL(l);
        JRP(nonzero)=RP(l);
      END DO
      DO l=k+1,RI(j+1)-1
        nonzero=nonzero+1;
        JRL(nonzero)=RL(l);
        JRP(nonzero)=RP(l);
      END DO
      EXIT
    END IF
  END DO
END DO
DO j=arStart,arEnd
  DO k=RI(j),RI(j+1)-1
    IF (RL(k) == i) THEN
      DO l=RI(j),k-1
        nonzero=nonzero+1;
        JRL(nonzero)=RL(l);
        JRP(nonzero)=RP(l);
      END DO
      IF (abs(RP(k)-1.0D0) > 0.0D0) THEN ! if order is not zero
        nonzero=nonzero+1;
        JRL(nonzero)=RL(l);
        JRP(nonzero)=RP(l)-1.0D0;
      END IF
      DO l=k+1,RI(j+1)-1
        nonzero=nonzero+1;
        JRL(nonzero)=RL(l);
        JRP(nonzero)=RP(l);
      END DO
      EXIT
    END IF
  END DO
END DO
!WRITE(6,fInsInt("(1X,A5/,1X,",ntc,"(I4,',',I4)")) "JYI =", JYI
!WRITE(6,fInsInt("(1X,A5/,1X,",JYI(ntc+1)-1-1,"(I4,',',I4)")) "JYL =", JYL
!WRITE(6,fInsInt2("(1X,A4/,",ntc,"(1X,",ngz-1,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "JR =", transpose(JR)
!WRITE(6,fInsInt2("(1X,A5/,1X,",JYI(ntc+1)-1,"(",ngz-1,"(1P1E10.2E3,',',1P1E10.2E3/))")) "JRY =", transpose(JRY)
!WRITE(6,fInsInt("(1X,A5/,1X,",JYI(ntc+1)-1-1,"(1P1E10.2E3,',',1P1E10.2E3)")) "JRC =", JRC
!WRITE(6,fInsInt("(1X,A5/,1X,",JYI(ntc+1)-1,"(I4,',',I4)")) "JRI =", JRI
!WRITE(6,fInsInt("(1X,A5/,1X,",JRI(JYI(ntc+1))-1,"(I4,',',I4)")) "JRL =", JRL
!WRITE(6,fInsInt("(1X,A5/,1X,",JRI(JYI(ntc+1))-1,"(1P1E10.2E3,',',1P1E10.2E3)")) "JRP =", JRP
```

Table E.2 Continued

```
ALLOCATE(S(ntc,ntr)); ! initialize stiochiometric matrix S ----------------------------------------------------------------------------
nonzero=0; ! counter for nonzero entries
DO i=1,ntc
  DO j=1,ntr
    IF (Mech(abs(RIV(j)),i) /= 0) THEN
      S(i,j)=SIGN(-1,RIV(j))*Mech(abs(RIV(j)),i); ! construct stiochiometric matrix S
      nonzero=nonzero+1; ! count nonzero entries in S
    ELSE
      S(i,j)=0;
    END IF
  END DO
END DO
ALLOCATE(SY(nonzero),SYI(ntc+1),SYL(nonzero)) ! convert S to Yale's sparse format
k=0; SYI(1)=k+1;
DO i=1,ntc
  DO j=1,ntr
    IF (S(i,j) /= 0) THEN
      k=k+1;
      SYI(i+1)=k+1;
      SYL(k)=j;
      SY(k)=S(i,j);
    END IF
  END DO
END DO
CALL sReaction ! compute nonlinear reaction part and store in array NC
ALLOCATE(SJRI(ntc+1)) ! construct integer vector of stoichiometric matrix of Jacobian
nonzero=0;
SJRI(1)=1;
DO i=1,ntc
  SJRI(i+1)=SJRI(i);
  DO j=1,ntc
    DO k=SYI(i),SYI(i+1)-1
      DO l=JYI(j),JYI(j+1)-1
        IF (SYL(k) == JYL(l)) THEN
          nonzero=1;
        END IF
      END DO
    END DO
    IF (nonzero == 1) THEN
      SJRI(i+1)=SJRI(i+1)+1;
      nonzero=0;
    END IF
  END DO
END DO
ALLOCATE(SJRL(SJRI(ntc+1)-1)) ! construct location vector of stoichiometric matrix of Jacobian
DO i=1,SJRI(ntc+1)-1; SJRL(i)=0; END DO
```

Table E.2 Continued

```
nonzero=0;
DO i=1,ntc
  DO j=1,ntc
    DO k=SYI(i),SYI(i+1)-1
      DO l=JYI(j),JYI(j+1)-1
        IF (SYL(k) == JYL(l)) THEN
          SJRL(nonzero+1)=j;
          EXIT
        END IF
      END DO
      IF(SJRL(nonzero+1) /= 0) THEN
        nonzero=nonzero+1;
        EXIT
      END IF
    END DO
  END DO
END DO
ALLOCATE(SJYI(SJRI(ntc+1)-1+1))
nonzero=0;
a=1;
SJYI(a)=nonzero+1;
DO i=1,ntc
  DO j=1,ntc
    DO k=SYI(i),SYI(i+1)-1
      DO l=JYI(j),JYI(j+1)-1
        IF (SYL(k) == JYL(l)) THEN
          nonzero=nonzero+1;
        END IF
      END DO
    END DO
    IF (nonzero /= 0) THEN
      a=a+1;
      SJYI(a)=SJYI(a-1)+nonzero;
      nonzero=0;
    END IF
  END DO
END DO
ALLOCATE(SJYRL(SJYI(SJRI(ntc+1)-1+1)-1),SJYCL(SJYI(SJRI(ntc+1)-1+1)-1),SJY(SJYI(SJRI(ntc+1)-1+1)-1))
nonzero=0;
DO i=1,ntc
  DO j=1,ntc
    DO k=SYI(i),SYI(i+1)-1
      DO l=JYI(j),JYI(j+1)-1
        IF (SYL(k) == JYL(l)) THEN
          nonzero=nonzero+1;
          SJYRL(nonzero)=JYL(l);
```

Table E.2 Continued

```
      SJYCL(nonzero)=1;
      SJY(nonzero)=SY(k);
    END IF
  END DO
 END DO
END DO
!WRITE(6,fInsInt2("(1X,A3/,",ntc,"(1X,",ntr-1,"(I4,',',),I4/)\)")) "S =", Transpose(S)
!WRITE(6,fInsInt("(1X,A5/,1X,",ntc+1-1,"(I4,',',),I4)")) "SYI =", SYI
!WRITE(6,fInsInt("(1X,A5/,1X,",SYI(ntc+1)-1-1,"(I4,',',),I4)")) "SYL =", SYL
!WRITE(6,fInsInt("(1X,A5/,1X,",SYI(ntc+1)-1-1,"(I4,',',),I4)")) "SY  =", SY
!WRITE(6,fInsInt2("(1X,A4/,",ngz,"(1X,",ntc-1,"(1P1E10.2E3,',',),1P1E10.2E3/)\)")) "NC =", transpose(NC)
!WRITE(6,fInsInt2("(1X,A7/,",ntc,"(",ntc-1,"(1P1E10.2E3,',',),1P1E10.2E3/)\)")) "S*DRDC=", transpose(MATMUL(1.0D0*S,DRDC))
!WRITE(6,fInsInt("(1X,A6/,1X,",ntc+1-1,"(I4,',',),I4)")) "SJRI =", SJRI
!WRITE(6,fInsInt("(1X,A6/,1X,",SJRI(ntc+1)-1-1,"(I4,',',),I4)")) "SJRL =", SJRL
!WRITE(6,fInsInt("(1X,A5/,1X,",SJRI(ntc+1)-1+1-1,"(I4,',',),I4)")) "SJYI =", SJYI
!WRITE(6,fInsInt("(1X,A5/,1X,",SJYI(SJRI(ntc+1)-1+1)-1-1,"(I4,',',),I4)")) "SJYRL =", SJYRL
!WRITE(6,fInsInt("(1X,A5/,1X,",SJYI(SJRI(ntc+1)-1+1)-1-1,"(I4,',',),I4)")) "SJYCL =", SJYCL
!WRITE(6,fInsInt("(1X,A5/,1X,",SJYI(SJRI(ntc+1)-1+1)-1-1,"(I4,',',),I4)")) "SJY  =", SJY
SELECT CASE (ecm) ! initialize parameters for element constraint method ----------------------------------------------------------------|
  CASE (0) ! no ECM
    ndc=0; ! no dependent component
  CASE (1) ! catalytic site
    ndc=fRank(M((nce-ncs+1):nce,1:ntc),ncs,ntc); ! empty sites are dependent
  CASE (2) ! ECM
    ndc=fRank(M,nce,ntc); ! number of dependent component equals to the rank of formula matrix
END SELECT
nic=ntc-ndc; ! number of independent component
iStart=1;      iEnd=iStart+nic-1; ! starting and ending position of independent component
dStart=iEnd+1; dEnd=dStart+ndc-1; ! starting and ending position of dependent component
IF (ecm == 1 .OR. ecm == 2) THEN ! initialize sparse matrix for subroutine sElement ----------------------------------------------------|
  ALLOCATE(MIMD(ndc,ntc));
  MIMD=M(nce-ndc+1:nce,1:ntc);
  CALL sCheck("MIMD")
  ALLOCATE(MI(ndc,nic),MD(ndc,ndc));
  MI=M(nce-ndc+1:nce,          1:ntc-ndc);
  MD=M(nce-ndc+1:nce,ntc-ndc+1:ntc    );
  CALL sCheck("MI")
  CALL sCheck("MD")
  ALLOCATE(MDi(ndc,ndc),MDiM(ndc,ntc),MDiMI(ndc,nic))
  MDi=1.0D0*MD;
  CALL sInverse(MDi,ndc)
  MDiM =MATMUL(MDi,1.0D0*MIMD);
  MDiMI=MATMUL(MDi,1.0D0*MI);
  nonzero=0; ! construct sparse matrix of (MD^-1)*M
  DO i=1,ndc
```

Table E.2 Continued

```
  DO j=1,ntc
    IF (abs(MDiM(i,j)) > 0.0D0) THEN
      nonzero=nonzero+1;
    END IF
  END DO
END DO
ALLOCATE(MDiMY(nonzero),MDiMYI(ndc+1),MDiMYL(nonzero))
k=0;
MDiMYI(1)=k+1
DO i=1,ndc
  DO j=1,ntc
    IF (abs(MDiM(i,j)) > 0.0D0) THEN
      k=k+1;
      MDiMY(k)=MDiM(i,j);
      MDiMYL(k)=j;
    END IF
  END DO
  MDiMYI(i+1)=k+1;
END DO
nonzero=0; ! construct sparse matrix of (MD^-1)*MI
DO i=1,ndc
  DO j=1,nic
    IF (abs(MDiMI(i,j)) > 0.0D0) THEN
      nonzero=nonzero+1;
    END IF
  END DO
END DO
ALLOCATE(MDiMIY(nonzero),MDiMIYI(ndc+1),MDiMIYL(nonzero))
k=0;
MDiMIYI(1)=k+1
DO i=1,ndc
  DO j=1,nic
    IF (abs(MDiMI(i,j)) > 0.0D0) THEN
      k=k+1;
      MDiMIY(k)=MDiMI(i,j);
      MDiMIYL(k)=j;
    END IF
  END DO
  MDiMIYI(i+1)=k+1;
END DO
IF (nsc >= 0) THEN
  k=0;
  DO i=(nce-ncs+1),nce
    DO j=sStart,sEnd
      IF ( M(i,j)>0 ) THEN
        k=k+1;
```

Table E.2 Continued

```
      END IF
    END DO
  END DO
  ALLOCATE(MscI(ncs+1),MscL(k),Msc(k))
k=0;
  MscI(1)=k+1;
DO i=(nce-ncs+1),nce
  DO j=sStart,sEnd
  IF ( M(i,j)>0 ) THEN
    k=k+1;
    MscL(k)=j;
    Msc(k)=M(i,j);
  END IF
  END DO
  MscI(i-(nce-ncs)+1)=k+1;
END DO
ALLOCATE(SCtot(ngz,ncs))
  DO i=1,ncs
  DO j=1,ngz
    SCtot(j,i)=C(j,cStart+i-1);
  END DO
END DO
DO i=1,ncs
DO k=MscI(i),MscI(i+1)-1
  SCtot(rStart:rEnd,i)=SCtot(rStart:rEnd,i)+Msc(k)*C(rStart:rEnd,MscL(k));
  END DO
END DO
END IF
END IF
!WRITE(6,"(3(1X,A5,1X,I3,',',1X,A5,1X,I3)") "ecm =", ecm, "nce =", nce, "ndc =", ndc, "nic =", nic
!WRITE(6,"(4(1X,A8,1X,I3,',',1X,A8,1X,I3)") "iStart =", iStart, "  iEnd =", iEnd, "dStart =", dStart, "  dEnd =", dEnd
!WRITE(6,fInsInt2("(1X,A3/,",nce,"(1X,",ntc-1,"(I4,',',I4/)\)")) "M =", Transpose(M)
!WRITE(6,fInsInt2("(1X,A6/,",ndc,"(1X,",ntc-1,"(I4,',',I4/)\)")) "MIMD =", Transpose(MIMD)
!WRITE(6,fInsInt2("(1X,A4/,",ndc,"(1X,",nic-1,"(I4,',',I4/)\)")) "MI =", Transpose(MI)
!WRITE(6,fInsInt2("(1X,A4/,",ndc,"(1X,",ndc-1,"(I4,',',I4/)\)")) "MD =", Transpose(MD)
!WRITE(6,fInsInt2("(1X,A5/,",ndc,"(1X,",ndc-1,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "MDi =", Transpose(MDi)
!WRITE(6,fInsInt2("(1X,A6/,",ndc,"(1X,",ntc-1,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "MDiM =", Transpose(MDiM)
!WRITE(6,fInsInt("(1X,A8/,1X,",ndc+1-1,"(I4,',',I4)")) "MDiMYI =", MDiMYI
!WRITE(6,fInsInt("(1X,A8/,1X,",MDiMYI(ndc+1)-1-1,"(I4,',',I4)")) "MDiMYL =", MDiMYL
!WRITE(6,fInsInt("(1X,A8/,1X,",MDiMYI(ndc+1)-1-1,"(1P1E10.2E3,',',1P1E10.2E3)")) "MDiMY  =", MDiMY
!WRITE(6,fInsInt2("(1X,A7/,",ndc,"(1X,",nic-1,"(1P1E10.2E3,',',1P1E10.2E3/)\)")) "MDiMI =", Transpose(MDiMI)
!WRITE(6,fInsInt("(1X,A9/,1X,",ndc+1-1,"(I4,',',I4)")) "MDiMIYI =", MDiMIYI
!WRITE(6,fInsInt("(1X,A9/,1X,",MDiMIYI(ndc+1)-1-1,"(I4,',',I4)")) "MDiMIYL =", MDiMIYL
!WRITE(6,fInsInt("(1X,A9/,1X,",MDiMIYI(ndc+1)-1-1,"(1P1E10.2E3,',',1P1E10.2E3)")) "MDiMIY  =", MDiMIY
!WRITE(6,fInsInt("(1X,A6/,1X,",ncs+1-1,"(I4,',',I4)")) "MscI =", MscI
!WRITE(6,fInsInt("(1X,A6/,1X,",MscI(ncs+1)-1-1,"(I4,',',I4)")) "MscL =", MscL
```

Table E.2 Continued

```
!WRITE(6,fInsInt("(1X,A6/,1X,",MscI(ncs+1)-1-1,"(I4,',')),I4)")) "Msc  =", Msc
!WRITE(6,fInsInt2("(1X,A7/,",ngz,"(1X,",ncs-1,"(1P1E10.2E3,',')),1P1E10.2E3/)\)")) "SCtot =", transpose(SCtot)
CALL cpu_time(iCPUTime); ! initialize runtime variables ----------------------------------------------------------------------|
fCPUTime=iCPUTime; ! initialize final cpu clock time
dCPUTime=fCPUTime-iCPUTime; ! time span between initial and final cpu clock
iRuntime=0; ! counter for one percent of total runtime
tRuntime=tiRun; ! current runtime
CALL sRuntime(6) ! write runtime information to screen
!WRITE(6,"(2(1X,A10,1X,1P1E10.2E3,','),1X,A10,1X,1P1E10.2E3)") "iCPUTime =",iCPUTime,"fCPUTime =",fCPUTime,"dCPUTime =",dCPUTime
!WRITE(6,"(1X,A10,1X,1P1E10.2E3,',',1X,A10,1X,I4)") "tRuntime =", tRuntime, "iRuntime =", iRuntime
ALLOCATE(UnitC(ntc),FileC(ntc),UnitR(ntr),FileR(ntr)); ! initialize arrays of units to save data ----------------------------|
k=10;  ! keep the universal unit unused
fileRuntime=trim(TransCatDir)//"//"//trim(Run)//"_Runtime.txt"; ! initialize file to save *_Runtime.txt
formatRuntime="(1P1E23.15E3,',',1P1E23.15E3)";
k=k+1;
unitRuntime=k;
OPEN(UNIT=k,FILE=fileRuntime,STATUS="REPLACE")
!CALL sWrite("unitRuntime",6,1)
!CALL sWrite("fileRuntime",6,1)
!CALL sWrite("formatRuntime",6,1)
fileTime=trim(TransCatDir)//"//"//trim(Run)//"_Time.txt"; ! initialize file to save *_Time.txt
formatTime="(1P1E23.15E3)";
k=k+1;
unitTime=k;
OPEN(UNIT=unitTime,FILE=fileTime,STATUS="REPLACE")
!CALL sWrite("unitTime",6,1)
!CALL sWrite("fileTime",6,1)
!CALL sWrite("formatTime",6,1)
DO i=1,ntc
  FileC(i)=trim(TransCatDir)//"//"//trim(Run)//"//"//trim(Run)//fInsInt("_C",i,".txt"); ! initialize file to save *_C*.txt
  formatC=fInsInt("(1P1E11.3E3,",ngz-1,"(',',1P1E11.3E3))"); ! or formatC=fInsInt("(1P1E23.15E3,",ngz-1,"(',',1P1E23.15E3))");
  k=k+1;
  UnitC(i)=k;
  OPEN(UNIT=UnitC(i),FILE=FileC(i),STATUS="REPLACE")
END DO
!CALL sWrite("UnitC",6,1)
!CALL sWrite("FileC",6,1)
!CALL sWrite("formatC",6,1)
DO i=1,ntr
  FileR(i)=trim(TransCatDir)//"//"//trim(Run)//"//"//trim(Run)//fInsInt("_R",i,".txt") ! initialize file to save *_R*.txt
  formatR=fInsInt("(1P1E11.3E3,",ngz-1,"(',',1P1E11.3E3))"); ! or formatR=fInsInt("(1P1E23.15E3,",ngz-1,"(',',1P1E23.15E3))");
  k=k+1;
  UnitR(i)=k;
  OPEN(UNIT=UnitR(i),FILE=FileR(i),STATUS="REPLACE")
END DO
!CALL sWrite("UnitR",6,1)
```

Table E.2 Continued

```
!CALL sWrite("FileR",6,1)
!CALL sWrite("formatR",6,1)
IF (thermalType /= 1) THEN
  fileT=trim(TransCatDir)//"/"//trim(Run)//"/"//trim(Run)//"_T.txt"; ! initialize file to save *_T.txt
  formatT=fInsInt("(1P1E11.3E3,",ngz-1,"(',',1P1E11.3E3))"); ! or formatT=fInsInt("(1P1E23.15E3,",ngz-1,"(',',1P1E23.15E3))");
  k=k+1;
  unitT=k;
  OPEN(UNIT=unitT,FILE=fileT,STATUS="REPLACE")
  !CALL sWrite("unitT",6,1)
  !CALL sWrite("fileT",6,1)
  !CALL sWrite("formatT",6,1)
  ALLOCATE(UnitDeff(ngc),FileDeff(ngc)) ! initialize file to save *_Deff*.txt
  DO i=1,ngc
    FileDeff(i)=trim(TransCatDir)//"/"//trim(Run)//"/"//trim(Run)//fInsInt("_Deff",i,".txt");
    formatDeff=fInsInt("(1P1E11.3E3,",ngz-1,"(',',1P1E11.3E3))");! formatDeff=fInsInt("(1P1E23.15E3,",ngz-1,"(',',1P1E23.15E3))");
    k=k+1;
      UnitDeff(i)=k;
      OPEN(UNIT=UnitDeff(i),FILE=FileDeff(i),STATUS="REPLACE")
  END DO
  !CALL sWrite("UnitDeff",6,1)
  !CALL sWrite("FileDeff",6,1)
  !CALL sWrite("formatDeff",6,1)
  ALLOCATE(UnitTau(ngc),FileTau(ngc)) ! initialize file to save *_Tau*.txt
  DO i=1,ngc
    FileTau(i)=trim(TransCatDir)//"/"//trim(Run)//"/"//trim(Run)//fInsInt("_Tau",i,".txt");
    formatTau=fInsInt("(1P1E11.3E3,",ngz-1,"(',',1P1E11.3E3))"); ! formatTau=fInsInt("(1P1E23.15E3,",ngz-1,"(',',1P1E23.15E3))");
    k=k+1;
      UnitTau(i)=k;
      OPEN(UNIT=UnitTau(i),FILE=FileTau(i),STATUS="REPLACE")
  END DO
  !CALL sWrite("UnitTau",6,1)
  !CALL sWrite("FileTau",6,1)
  !CALL sWrite("formatTau",6,1)
  ALLOCATE(UnitRC(ntr),FileRC(ntr)) ! initialize file to save *_RC*.txt
  DO i=1,ntr
    FileRC(i)=trim(TransCatDir)//"/"//trim(Run)//"/"//trim(Run)//fInsInt("_RC",i,".txt");
    formatRC=fInsInt("(1P1E11.3E3,",ngz-1,"(',',1P1E11.3E3))"); ! formatRC=fInsInt("(1P1E23.15E3,",ngz-1,"(',',1P1E23.15E3))");
    k=k+1;
    UnitRC(i)=k;
      OPEN(UNIT=UnitRC(i),FILE=FileRC(i),STATUS="REPLACE")
  END DO
  !CALL sWrite("UnitRC",6,1)
  !CALL sWrite("FileRC",6,1)
  !CALL sWrite("formatRC",6,1)
END IF
CALL sSave("Dynamic");                ! save dynamic variables at tiRun ----------------------------------------------------------------|
```

Table E.2 Continued

```
CALL sSave("Output.xml")              ! save all module to file output.txt
CALL sSave("InputNew.txt")            ! save a new input file NewInput.txt when input format is changed
!CALL sSave("Output.txt")             ! save all module to file output.xml (already replaced by Output.xml)
!CALL sSave("InitialConcentration.txt") ! save initial concentration to file (legacy file, no use)
!CALL sSave("InitialTemperature.txt")  ! save initial temperature to file (legacy file, no use)
! Code below this line is for development and debugging ----------------------------------------------------------------------|
!CALL sWrite("MODULE",6,1)            ! write all module on screen
!CALL sWrite("mInformation",6,1)      ! Write module mInformation on screen
!CALL sWrite("mDynamic",6,1)          ! Write module mDynamic on screen
!CALL sWrite("mParameter",6,1)        ! Write module mParameter on screen
!CALL sWrite("mReaction",6,1)         ! Write module mReaction on screen
!CALL sWrite("mStoichiometry",6,1)    ! Write module mStoichiometry on screen
!CALL sWrite("mSave",6,1)             ! Write module mSave on screen
!CALL sWrite("mBoundary",6,1)         ! Write module mBoundary on screen
!CALL sWrite("mRuntime",6,1)          ! Write module mRuntime on screen
IF (run=="TestInput" .OR. run=="CheckParameter") THEN
OPEN(UNIT=10, FILE=trim(run)//"//"CheckParameter.txt", STATUS="REPLACE")
DO i=6,10,4
WRITE(i,"(A64)") "Standard input parameters of TransCat from Laura's paper (2002)." !-----------------------------------------|
WRITE(i,*) "                                  Value in paper    Value in TransCat"
WRITE(i,*) "      Volume of bulk chamber (m^3) :    5.60X10^-7,  ", BCCV(2)
WRITE(i,*) "Volume of centerplane chamber (m^3) :    4.40X10^-8,  ", BCCV(1)
WRITE(i,*) "          Maximum O2 pressure (Pa) :           110,  ", MAXVAL(BCPmax(:,2))
WRITE(i,*) "          Maximum CO pressure (Pa) :            16,  ", MAXVAL(BCPmax(:,1))
WRITE(i,*) " Maximum O2 concentration (mol/m^3) :            NA,  ", MAXVAL(BCCmax(:,2))*maxGC
WRITE(i,*) " Maximum CO concentration (mol/m^3) :            NA,  ", MAXVAL(BCCmax(:,1))*maxGC
WRITE(i,*) "               Temperature (K) :        423.15,  ", T(rStart)
WRITE(i,*) "          Pellet thickness (m) :  6.40X10^-4,  ", lz
WRITE(i,*) "          Pellet diameter (m) :  4.00X10^-3,  ", ld
WRITE(i,*) "    Area of each pellet face  (m) :  1.26X10^-5,  ", totRCA
WRITE(i,*) "                       Porosity :          0.3,  ", catP
WRITE(i,*) "      Weight percentage of Pt (%) :         2.00   ,  ", 0.5*SUM(catWF(1:2))
WRITE(i,*) "          Dispersion of Pt (%) :         1.00   ,  ", 0.5*SUM(catD(1:2))
WRITE(i,*) " Area per mole surface Pt (m^2/mol) :  5.50X10^+4,  ", 0.5*SUM(catS(1:2))
WRITE(i,*) "Total moles of surface Pt atom per
WRITE(i,*) "   unit volume of pellet (mol/m^3) :         0.85,  ", totCS/totRV
WRITE(i,*) "               Ratio of site one :         0.15,  ", catCS(1)/totCS
WRITE(i,*) "               Ratio of site two :         0.85,  ", catCS(2)/totCS
WRITE(i,*) "      alpha for CO and CO2 :          590,  ", (totCS/totRV)/(maxGC*MAXVAL(BCCmax(:,1)))/catP
WRITE(i,*) "                 alpha for O2 :           NA,  ", (totCS/totRV)/(maxGC*MAXVAL(BCCmax(:,2)))/catP
WRITE(i,*) "            Tau for CO  (s) :         0.31,  ", Tau(rStart,1)
WRITE(i,*) "            Tau for O2  (s) :           NA,  ", Tau(rStart,2)
WRITE(i,*) "           Tau for CO2 (s) :           NA,  ", Tau(rStart,3)
WRITE(i,*) "       Tau_cpp for CO  (s) :         5.6,  ", Tau(rStart-1,1)
WRITE(i,*) "       Tau_cpp for O2  (s) :           NA,  ", Tau(rStart-1,2)
WRITE(i,*) "       Tau_cpp for CO2 (s) :           NA,  ", Tau(rStart-1,3)
```

Table E.2 Continued

```
WRITE(i,*) "      Tau_cpl for CO  (s) :        190,  ", Tau(rStart-2,1)
WRITE(i,*) "      Tau_cpl for O2  (s) :         NA,  ", Tau(rStart-2,2)
WRITE(i,*) "      Tau_cpl for CO2 (s) :         NA,  ", Tau(rStart-2,3)
WRITE(i,*) "   k1_ads_CO (m^3/mol/s) :  7.78X10^6,  ", RC(1,rStart)/maxGC
WRITE(i,*) "   k2_ads_CO (m^3/mol/s) :  7.78X10^6,  ", RC(4,rStart)/maxGC
WRITE(i,*) "      k1_des_CO       (1/s) :        310,  ", RC(2,rStart)
WRITE(i,*) "      k2_des_CO       (1/s) :        520,  ", RC(5,rStart)
WRITE(i,*) "   k1_ads_O2 (m^3/mol/s) :        348,  ", RC(7,rStart)/maxGC
WRITE(i,*) "   k2_ads_O2 (m^3/mol/s) :        348,  ", RC(8,rStart)/maxGC
WRITE(i,*) "      k1_rxn          (1/s) :        6.1,  ", RC(3,rStart)
WRITE(i,*) "      k2_rxn          (1/s) :     0.0015,  ", RC(6,rStart)
WRITE(i,*)
END DO
CLOSE(UNIT=10)
END IF
END SUBROUTINE sInitialize
!-----------------------------------------------------------------------
! Function fCountC
! Description: This function count the number of components involved in a reaction. Repeating component will be count as one.
! Created by Hsu-Wen Hsiao on 8/5/2010.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   u: location vector of reaction rate
!   m: size of u
!   n: number of component
! Output:
!   fCountC: count of the number of component
!-----------------------------------------------------------------------
INTEGER FUNCTION fCountC(u,m,n)
IMPLICIT NONE
INTEGER, DIMENSION(m), INTENT(IN) :: u
INTEGER, INTENT(IN) :: m, n
LOGICAL, DIMENSION(n) :: v
INTEGER :: i, j
DO i=1,n
  v(i)=0;
  DO j=1,m
    IF (u(j) == i) THEN
      v(i)=1;
      exit
    END IF
  END DO
END DO
fCountC=COUNT(v);
END FUNCTION fCountC
```

Table E.3 sRead.f90

```
!-----------------------------------------------------------------------
! Subroutine sRead
! Description: Read data from an input file.
! Created by Hsu-Wen Hsiao on 3/19/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   filename: The filename of data to be read
! Output: none
!-----------------------------------------------------------------------
SUBROUTINE sRead(filename)
USE IFPOSIX
USE mInformation
USE mDynamic
USE mReaction
USE mParameter
USE mElement
USE mBoundary
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: filename
INTEGER :: i !, j
INTEGER :: dirLen,dirError
!-----------------------------------------------------------------------
SELECT CASE (filename)
  CASE ("Run.txt") ! Read File Run.txt ---------------------------------
    stopRun=0;
    CALL PXFGETCWD (TransCatDir,dirLen,dirError)
  OPEN(UNIT=10, FILE=trim(TransCatDir)//"\"//trim(filename), STATUS="OLD")
    READ(10,*); READ(10,*) nRun
    ALLOCATE(RunList(nRun)); READ(10,*); DO i=1,nRun; READ(10,*) RunList(i); END DO
  CLOSE(UNIT=10)
    CALL sCheck("Run.txt")
CASE ("Input.txt") ! Read file Input.txt from run folder -------------------------------------------------------
    OPEN(UNIT=10, FILE=trim(TransCatDir)//"\"//trim(run)//"\"//filename, STATUS="OLD")
      READ(10,*); READ(10,"(A500)") runDescription
      READ(10,*); READ(10,*) reactorType
      READ(10,*); READ(10,*) thermalType
      READ(10,*); READ(10,*) numericalMethod
      READ(10,*); READ(10,*) ecm
      READ(10,*); READ(10,*) tiRun
      READ(10,*); READ(10,*) tfRun
      READ(10,*); READ(10,*) nts
      READ(10,*); READ(10,*) ngc
      READ(10,*); READ(10,*) nsc
      READ(10,*); READ(10,*) ncs
      ALLOCATE(Component(ngc+nsc+ncs)); READ(10,*); READ(10,*) Component
```

Table E.3 Continued

```
    ALLOCATE(MWGas(ngc)); READ(10,*); READ(10,*) MWGas
    READ(10,*); READ(10,*) rMW
    READ(10,*); READ(10,*) rDeff
    READ(10,*); READ(10,*) rTemp
    READ(10,*); READ(10,*) iniT
    READ(10,*); READ(10,*) ld
    READ(10,*); READ(10,*) lz
    READ(10,*); READ(10,*) nGrid
    READ(10,*); READ(10,*) catP
    READ(10,*); READ(10,*) catW
    ALLOCATE(catWF(ncs)); READ(10,*); READ(10,*) catWF
    ALLOCATE(catMW(ncs)); READ(10,*); READ(10,*) catMW
    ALLOCATE(catS (ncs)); READ(10,*); READ(10,*) catS
    ALLOCATE(catD (ncs)); READ(10,*); READ(10,*) catD
    ALLOCATE(catSF(ncs)); READ(10,*); READ(10,*) catSF
    READ(10,*); READ(10,*) nce
    ALLOCATE(Element(nce)); READ(10,*); READ(10,*) Element
    ALLOCATE(M(nce,ngc+nsc+ncs)); READ(10,*); DO i=1,nce; READ(10,*) M(i,1:ngc+nsc+ncs); END DO
    READ(10,*); READ(10,*) nrer
    IF (nrer > 0) THEN
      ALLOCATE(MechE(nrer,ngc+nsc+ncs)); READ(10,*); DO i=1,nrer; READ(10,*) MechE(i,1:ngc+nsc+ncs); END DO
      ALLOCATE(ERC(nrer,6)); READ(10,*); DO i=1,nrer; READ(10,*) ERC(i,1:6); END DO
    ELSE
      READ(10,*); READ(10,*); READ(10,*); READ(10,*)
    END IF
    READ(10,*); READ(10,*) nrar
    IF (nrar > 0) THEN
      ALLOCATE(MechA(nrar,ngc+nsc+ncs)); READ(10,*); DO i=1,nrar; READ(10,*) MechA(i,1:ngc+nsc+ncs); END DO
      ALLOCATE(ARC(nrar,6)); READ(10,*); DO i=1,nrar; READ(10,*) ARC(i,1:6); END DO
      ALLOCATE(ARO(nrar,2*(ngc+nsc+ncs))); READ(10,*); DO i=1,nrar; READ(10,*) ARO(i,1:2*(ngc+nsc+ncs)); END DO
    ELSE
      READ(10,*); READ(10,*); READ(10,*); READ(10,*); READ(10,*); READ(10,*)
    END IF
    READ(10,*); READ(10,*) nrur
    IF (nrur > 0) THEN
      ALLOCATE(MechU(nrur,ngc+nsc+ncs)); READ(10,*); DO i=1,nrur; READ(10,*) MechU(i,1:ngc+nsc+ncs); END DO
      ALLOCATE(URC(nrur,6)); READ(10,*); DO i=1,nrur; READ(10,*) URC(i,1:6); END DO
    ELSE
      READ(10,*); READ(10,*); READ(10,*); READ(10,*)
    END IF
    READ(10,*); READ(10,*) inputIC
    READ(10,*); READ(10,*) inputFIC
    READ(10,*); READ(10,*) inputIT
    READ(10,*); READ(10,*) inputFIT
    ALLOCATE(IniGP(ngc)); READ(10,*); READ(10,*) IniGP
    ALLOCATE(IniSC(nsc+ncs)); READ(10,*); READ(10,*) IniSC
```

Table E.3 Continued

```
    READ(10,*); READ(10,*) CBC
    ALLOCATE(BCPmax(2,ngc)); READ(10,*); DO i=1,2; READ(10,*) BCPmax(i,1:ngc); END DO
    ALLOCATE(BCPmin(2,ngc)); READ(10,*); DO i=1,2; READ(10,*) BCPmin(i,1:ngc); END DO
    ALLOCATE(BCW(2,ngc));    READ(10,*); DO i=1,2; READ(10,*) BCW(i,1:ngc);    END DO
    ALLOCATE(BCF(2,ngc));    READ(10,*); DO i=1,2; READ(10,*) BCF(i,1:ngc);    END DO
    ALLOCATE(BCL(2,ngc));    READ(10,*); DO i=1,2; READ(10,*) BCL(i,1:ngc);    END DO
    READ(10,*); READ(10,*) BCCV
    READ(10,*); READ(10,*) BCRQ
    READ(10,*); READ(10,*) BCRM
    READ(10,*); READ(10,*) BCRT
    READ(10,*); READ(10,*) BCT
    READ(10,*); READ(10,*) BCEMTC
    READ(10,*); READ(10,*) TBC
    READ(10,*); READ(10,*) BCTmax
    READ(10,*); READ(10,*) BCTmin
    READ(10,*); READ(10,*) BCTW
    READ(10,*); READ(10,*) BCTF
    READ(10,*); READ(10,*) BCTL
  CLOSE(UNIT=10)
  CALL sCheck("Input.txt")
CASE ("Initial Concentration") ! Read file InitialConcentration.txt ----------------------------------------------------------------------|
  OPEN(UNIT=10, FILE=trim(TransCatDir)//"/"//trim(Run)//"/"//trim(inputFIC), STATUS="OLD")
  DO i=1,ngz; READ(10,*) C(i,1:ntc); END DO
  CLOSE(UNIT=10)
  CALL sCheck("Initial Concentration")
CASE ("Initial Temperature") ! Read file InitialTemperature.txt ----------------------------------------------------------------------|
  OPEN(UNIT=10, FILE=trim(TransCatDir)//"/"//trim(Run)//"/"//trim(inputFIT), STATUS="OLD")
    READ(10,*) T(1:ngz)
  CLOSE(UNIT=10)
  CALL sCheck("Initial Temperature")
END SELECT
END SUBROUTINE sRead
```

Table E.4 sCheck.f90

```
!-----------------------------------------------------------------------
! Subroutine sCheck
! Description: Check variable and parameter.
! Created by Hsu-Wen Hsiao on 7/16/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   varName: Name of input variable.
! Output: none
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sCheck(varName)
USE mInformation
USE mDynamic
USE mReaction
USE mParameter
USE mElement
USE mBoundary
!USE mSave
!USE mRuntime
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: varName
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
INTEGER, EXTERNAL :: fRank
CHARACTER(LEN=32), DIMENSION(:), ALLOCATABLE :: vList
INTEGER :: vSize
CHARACTER(LEN=256) :: vString
LOGICAL :: vBack
INTEGER :: i, j
SELECT CASE (varName)
  CASE ("Run.txt") ! check input parameters from file Run.txt ----------------------------------------------------------------------
    CALL sCheck("nRun")
    CALL sCheck("Runlist")
  CASE ("nRun")
    IF(nRun <= 0) THEN; stopRun=1; stopInfo="Input variable nRun must be positive. Please revise file Run.txt."; END IF
  CASE ("RunList")
    DO i=1,nRun
      IF (len_trim(RunList(i)) <= 0) THEN
        stopRun=1;
        stopInfo="RunList cannot contain empty line. Please revise file Run.txt";
      END IF
    END DO
  CASE ("Initial Concentration") ! check input initial concentration from file ----------------------------------------------------------
    !pending
  CASE ("C")
    !pending (surface coverage must be between 0.0 and 1.0, and error in total surface coverage must be less than 1%)
  CASE ("Initial Temperature") ! check input initial termperature from file ----------------------------------------------------------
```

Table E.4 Continued

```
  !pending
CASE ("T")
  !pending (must be consistent with iniT and BCT in isothermal case)
CASE ("Input.txt") ! check input parameters from file Input.txt ----------------------------------------------------------------------------------|
  vSize=57;
  ALLOCATE(vList(vSize))
    vList=(/"runDescription ", "reactorType    ", "thermalType    ", "numericalMethod", "ecm        ", &
            "tiRun          ", "tfRun          ", "nts            ", "ngc           ", "nsc        ", &
            "ncs            ", "Component      ", "MWGas          ", "rMW           ", "rDeff      ", &
            "rTemp          ", "iniT           ", "ld             ", "lz            ", "nGrid      ", &
            "catP           ", "catW           ", "catWF          ", "catMW         ", "catS       ", &
            "catD           ",                                                                      &
            "catSF          ", "nce            ", "Element        ", "M             ", "nrer       ", &
            "MechE          ", "ERC            ", "nrar           ", "MechA         ", "ARC        ", &
            "OrderAR        ", "MechU          ", "URC            ", "inputIC       ", "inputFIC   ", &
            "inputIT        ", "inputFIT       ", "IniGP          ", "IniSC         ", "CBC        ", &
            "BCPmax         ", "BCPmin         ", "BCW            ", "BCF           ", "BCL        ", &
            "BCCV           ", "BCRQ           ", "BCRM           ", "BCRT          ", "BCT        ", &
            "TBC            "/); !
    DO i=1,vSize; IF (stopRun == 0) THEN; CALL sCheck(trim(vList(i))); END IF; END DO
    IF ( (ALLOCATED(vList))) THEN; DEALLOCATE(vList); END IF
CASE ("runDescription")
  ! no criterion
CASE ("reactorType")
  IF (reactorType < 1 .OR. reactorType > 2) THEN
    stopRun=1; stopInfo="Variable reactorType must be either 1 or 2. Please revise file Input.txt.";
  END IF
CASE ("thermalType")
  IF (thermalType < 1 .OR. thermalType > 3) THEN
    stopRun=1; stopInfo="Variable thermalType must be either 1 ,2 or 3. Please revise file Input.txt.";
  END IF
CASE ("numericalMethod")
  vString="EE,RK2,RK3,RK4,RK5,IE,BDF2,BDF3,BDF4,BDF5,CNEE,CNRK2,CNRK3,LOA,MOA,HOA";
  IF (len_trim(numericalMethod) == 0) THEN
    stopRun=1;
    stopInfo="Variable numericalMethod cannot be empty. Please revise file Input.txt."
  ELSEIF ( INDEX(vString,trim(numericalMethod),vBack) == 0) THEN
    stopRun=1;
    stopInfo="Variable numericalMethod not recognized. Please revise file Input.txt."
  END IF
CASE ("ecm")
  IF (ecm < 0 .OR. ecm > 2) THEN
    stopRun=1; stopInfo="Variable ecm must be either 0 ,1 or 2. Please revise file Input.txt.";
  END IF
CASE ("tiRun")
  IF (tiRun < 0) THEN
```

Table E.4 Continued

```
    stopRun=1; stopInfo="Variable tiRun cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("tfRun")
  IF (tfRun < 0) THEN
    stopRun=1; stopInfo="Variable tfRun cannot be negative. Please revise file Input.txt.";
  ELSEIF (tfRun < tiRun) THEN
    stopRun=1; stopInfo="Variable tfRun must be larger than tiRun. Please revise file Input.txt.";
  END IF
CASE ("nts")
  IF (nts < 1) THEN
    stopRun=1; stopInfo="Variable nts must be positive integer. Please revise file Input.txt.";
  END IF
CASE ("ngc")
  IF (ngc < 0) THEN
    stopRun=1; stopInfo="Variable ngc cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("nsc")
  IF (nsc < 0) THEN
    stopRun=1; stopInfo="Variable nsc cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("ncs")
  IF (ncs < 0) THEN
    stopRun=1; stopInfo="Variable ncs cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("Component")
  ! no criterion
CASE ("MWGas")
  IF (MINVAL(MWGas) <= 0.0) THEN
    stopRun=1; stopInfo="Variable MWGas must be positive. Please revise file Input.txt.";
  END IF
CASE ("rMW")
  IF (rMW <= 0.0) THEN
    stopRun=1; stopInfo="Variable rMW must be positive. Please revise file Input.txt.";
  END IF
CASE ("rDeff")
  IF (rDeff < 0.0) THEN
    stopRun=1; stopInfo="Variable rDeff cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("rTemp")
  IF (rTemp <= 0.0) THEN
    stopRun=1;
    stopInfo="Input variable rTemp must be positive. Please revise file Input.txt.";
  END IF
CASE ("iniT")
  IF (iniT <= 0.0) THEN
    stopRun=1;
```

Table E.4 Continued

```
    stopInfo="Input variable iniT must be positive. Please revise file Input.txt.";
  END IF
CASE ("ld")
  IF (ld <= 0.0) THEN
    stopRun=1;
    stopInfo="Input variable ld must be positive. Please revise file Input.txt.";
  END IF
CASE ("lz")
  IF (lz <= 0.0) THEN
    stopRun=1;
    stopInfo="Input variable lz must be positive. Please revise file Input.txt.";
  END IF
CASE ("nGrid")
  IF (nGrid < 0) THEN
    stopRun=1;
    stopInfo="Input variable nGrid must be positive. Please revise file Input.txt.";
  END IF
  SELECT CASE (reactorType)
    CASE (1) ! DDR
      IF (nGrid < 2) THEN
          stopRun=1;
          stopInfo="Input variable nGrid must be greater than 1 for DDR. Please revise file Input.txt.";
      END IF
    CASE (2) ! CSTR
      IF (nGrid /= 1) THEN
        WRITE(6,*) "Warning: Variable nGrid has been changed to 1 for CSTR model. "// &
                   "Please revise file Input.txt to stop this warning"
      END IF
  END SELECT
CASE ("catP")
  IF (catP < 0.0 .OR. catP > 1.0) THEN
    stopRun=1;
    stopInfo="Input variable catP must be between 0.0 and 1.0. Please revise file Input.txt.";
  END IF
CASE ("catW")
  IF (catW <= 0.0) THEN
    stopRun=1;
    stopInfo="Input variable catW cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("catWF")
  IF (MINVAL(catWF) <= 0.0) THEN
    stopRun=1;
    stopInfo="Input variable catWF cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("catMW")
  IF (MINVAL(catMW) <= 0.0) THEN
```

Table E.4 Continued

```
    stopRun=1;
    stopInfo="Input variable catMW cannot be negative. Please revise file Input.txt.";
   END IF
  CASE ("catS")
   IF (MINVAL(catS) <= 0.0) THEN
    stopRun=1;
    stopInfo="Input variable catS cannot be negative. Please revise file Input.txt.";
   END IF
  CASE ("catD")
   IF (MINVAL(catD) < 0.0 .OR. MAXVAL(catD) > 1.0) THEN
    stopRun=1;
    stopInfo="Input variable catD must be between 0.0 and 1.0. Please revise file Input.txt.";
   END IF
  CASE ("catSF")
   IF (MINVAL(catSF) < 0.0 .OR. MAXVAL(catSF) > 5.0) THEN
    stopRun=1;
    stopInfo="Input variable catSF must be between 0.0 and 5.0. Please revise file Input.txt.";
   END IF
  CASE ("nce")
   IF (nce < 0) THEN
    stopRun=1; stopInfo="Variable nce cannot be negative. Please revise file Input.txt.";
   END IF
  CASE ("Element")
   ! no criterion
  CASE ("M")
   IF (MINVAL(M) < 0) THEN
    stopRun=1;
    stopInfo="Entries of formula matrix M cannot be negative. Please revise file Input.txt.";
   END IF
  CASE ("nrer")
   IF (nrer < 0) THEN
    stopRun=1; stopInfo="Variable nrer cannot be negative. Please revise file Input.txt.";
   END IF
  CASE ("MechE")
   IF (nrer >0) THEN
   !pending
   END IF
  CASE ("ERC")
   IF (nrer > 0) THEN
    IF (MINVAL(ERC) < 0.0) THEN
     stopRun=1;
     stopInfo="Entries of elementary rate constant matrix ERC cannot be negative. Please revise file Input.txt.";
    END IF
   END IF
  CASE ("nrar")
   IF (nrar < 0) THEN
```

Table E.4 Continued

```
    stopRun=1; stopInfo="Variable nrar cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("MechA")
  IF (nrar > 0) THEN
    !pending
  END IF
CASE ("ARC")
  IF (nrar > 0) THEN
    IF (MINVAL(ARC) < 0.0) THEN
      stopRun=1;
      stopInfo="Entries of algebraic rate constant matrix ARC cannot be negative. Please revise file Input.txt.";
    END IF
  END IF
CASE ("OrderAR")
  IF (nrar > 0) THEN
    !pending
  END IF
CASE ("MechU")
  IF (nrur > 0) THEN
    !pending
  END IF
CASE ("URC")
  IF (nrur > 0) THEN
    IF (MINVAL(URC) < 0.0) THEN
      stopRun=1;
      stopInfo="Entries of user-defined rate constant matrix URC cannot be negative. Please revise file Input.txt.";
    END IF
  END IF
CASE ("inputIC")
  IF (inputIC /= 0 .AND. inputIC /= 1) THEN
    stopRun=1;
    stopInfo="Variable inputIC must be either 0 or 1. Please revise file Input.txt.";
  END IF
CASE ("inputFIC")
  IF (inputIC == 1 .AND. len_trim(inputFIC) == 0) THEN
    stopRun=1;
    stopInfo="Input filename of initial concentration, inputFIC, cannot be empty. Please revise file Input.txt."
  END IF
CASE ("inputIT")
  IF (inputIT /= 0 .AND. inputIT /= 1) THEN
    stopRun=1;
    stopInfo="Variable inputIT must be either 0 or 1. Please revise file Input.txt.";
  END IF
CASE ("inputFIT")
  IF (inputIT == 1 .AND. len_trim(inputFIT) == 0) THEN
    stopRun=1;
```

Table E.4 Continued

```
  stopInfo="Input filename of initial temperature, inputFIT, cannot be empty. Please revise file Input.txt."
  END IF
CASE ("IniGP")
  IF (MINVAL(IniGP) < 0) THEN
    stopRun=1;
    stopInfo="Entries of initial gas pressure IniGP cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("IniSC")
  IF (MINVAL(IniSC) < 0.0 .OR. MAXVAL(IniSC)>1.0) THEN
    stopRun=1;
    stopInfo="Entries of initial surface coverage IniSC must be between 0.0 and 1.0. Please revise file Input.txt.";
  ELSEIF (SUM(IniSC)-1.0 > 1.0D-2) THEN ! 1% error in total surface coverage is allowed
    stopRun=1;
    stopInfo="Error in total surface coverage IniSC is greater than 1%. Please revise file Input.txt.";
  END IF
CASE ("CBC")
  IF (MINVAL(CBC) < 0 .OR. MAXVAL(CBC) > 3) THEN
    stopRun=1;
    stopInfo="Entries of concentration boundary condition CBC must be either 1, 2 or 3. Please revise file Input.txt.";
  END IF
CASE ("BCPmax")
    IF (MINVAL(BCPmax) < 0.0) THEN
      stopRun=1;
      stopInfo="Entries of BCPmax cannot be negative. Please revise file Input.txt.";
    ELSEIF (MINVAL(BCPmax-BCPmin) < 0.0) THEN
      stopRun=1;
      stopInfo="Input variable BCPmax must be greater than BCPmin. Please revise file Input.txt.";
    END IF
CASE ("BCPmin")
    IF (MINVAL(BCPmin) < 0.0) THEN
      stopRun=1;
      stopInfo="Entries of BCPmin cannot be negative. Please revise file Input.txt.";
    END IF
CASE ("BCW")
  IF (MINVAL(BCW) < 1 .OR. MAXVAL(BCW) > 4) THEN
    stopRun=1;
    stopInfo="Entries of matrix BCW must be a integer between 1 and 4. Please revise file Input.txt.";
  END IF
CASE ("BCF")
  IF (MINVAL(BCF) < 0.0) THEN
    stopRun=1;
    stopInfo="Entries of matrix BCF cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("BCL")
  IF (MINVAL(BCL) < 0.0 .OR. MAXVAL(BCL)>1.0) THEN
    stopRun=1;
```

Table E.4 Continued

```
    stopInfo="Entries of BCL must be between 0.0 and 1.0. Please revise file Input.txt.";
  END IF
CASE ("BCCV")
  IF (MINVAL(BCCV) < 0.0) THEN
    stopRun=1;
    stopInfo="Entries of matrix BCCV cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("BCRQ")
  IF (MINVAL(BCRQ) < 0.0) THEN
    stopRun=1;
    stopInfo="Entries of matrix BCRQ cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("BCRM")
  IF (MINVAL(BCRM) < 0.0) THEN
    stopRun=1;
    stopInfo="Entries of matrix BCRM cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("BCRT")
  IF (MINVAL(BCRT) < 0.0) THEN
    stopRun=1;
    stopInfo="Entries of BCRT cannot be negative. Please revise file Input.txt.";
  END IF
CASE ("BCT")
  IF (MINVAL(BCT) < 0.0) THEN
    stopRun=1;
    stopInfo="Entries of BCT cannot be negative. Please revise file Input.txt.";
  END IF
  IF (thermalType == 1) THEN
    DO i=1,2
      IF (BCT(i) /= iniT) THEN
        stopRun=1;
        stopInfo="Temperature of reactor and boundary must be consistent under isothermal condition."
        stopInfo=trim(stopInfo)//" Please revise iniT and BCT in file Input.txt.";
      END IF
    END DO
  END IF
CASE ("TBC")
  IF (MINVAL(TBC) < 0 .OR. MAXVAL(TBC) > 4) THEN
    stopRun=1;
    stopInfo="Entries of temperature boundary condition TBC must be either 1, 2, 3 or 4. Please revise file Input.txt.";
  END IF
  IF (thermalType==1) THEN
    IF (TBC(1)/=0 .OR. TBC(2)/=0) THEN
      stopRun=1;
      stopInfo="Entries of temperature boundary condition TBC must be 0 for isothermal runs. Please revise file Input.txt.";
    END IF
```

Table E.4 Continued

```
  END IF
 CASE ("MIMD")
  IF (fRank(MIMD,ndc,ntc) /= ndc) THEN
    stopRun=1;
    stopInfo="Please change the sequence of component or element in Input.txt, so the last ";
    stopInfo=fInsInt(trim(stopInfo),ndc," rows of formula matrix M are linearly independent.");
  END IF
 CASE ("MI")
  !pending
 CASE ("MD")
  IF (fRank(MD,ndc,ndc) /= ndc) THEN
    stopRun=1;
    stopInfo="Please change the sequence of component or element in Input.txt, so the last ";
    stopInfo=fInsInt2(trim(stopInfo),ndc," rows and ",ndc," columns of formula matrix M are nonsingular.");
  END IF
 CASE ("maxGC")
  IF (maxGC <= 0.0) THEN
    stopRun=1;
    stopInfo="Maximum gas concentration is zero. Please revise initial or boundary condition in file Input.txt.";
  END IF

END SELECT
IF (stopRun) THEN ! Stop run if error occur ----------------------------------------------------------------------|
  WRITE(6,*) trim(stopInfo)
  PAUSE ! This statement can keep the console wondow open in Microsoft Visual Studio. Deactivate this line Xcode under Mac OS.
  STOP
ELSE
  stopInfo="No error detected.";
END IF
END SUBROUTINE sCheck
```

Table E.5 sSave.f90

```
!------------------------------------------------------------------------------------------------------------------------|
! Subroutine sSave                                                                                                       |
! Description: Save data to a file                                                                                       |
! Created by Hsu-Wen Hsiao on 2/16/09.                                                                                   |
! Copyright 2009 UCSD. All rights reserved.                                                                              |
!------------------------------------------------------------------------------------------------------------------------|
! Input:                                                                                                                 |
!   flag: The filename or type of data to be saved                                                                       |
!------------------------------------------------------------------------------------------------------------------------|
SUBROUTINE sSave(flag)
USE mInformation
USE mParameter
USE mDynamic
USE mReaction
USE mSave
USE mRuntime
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: flag
CHARACTER(LEN=32), DIMENSION(:), ALLOCATABLE :: mList
INTEGER :: OutputUnit, FormatType
INTEGER :: mSize
INTEGER :: i, j
SELECT CASE (flag)
  CASE ("Dynamic") ! Save Dynamic variables ---------------------------------------------------------------------------|
    WRITE(unitTime,formatTime) tRun ! time
    CALL cpu_time(fCPUTime); WRITE(unitRuntime,formatRuntime) tRun, fCPUTime ! runtime
    DO i=1,ntc; WRITE(UnitC(i),formatC) C(1:ngz,i); END DO ! concentration
    DO i=1,ntr; WRITE(UnitR(i),formatR) R(i,1:ngz); END DO ! reaction rate
    IF (thermalType /= 1) THEN ! temperature ramp or non-isothermal
      WRITE(unitT,formatT) T(1:ngz) ! temperature
      DO i=1,ntr
        WRITE(UnitRC(i),formatRC) RC(i,1:ngz);
      END DO
      DO i=1,ngc
        WRITE(UnitDeff(i),formatDeff) Deff(1:ngz,i);
        WRITE(UnitTau(i),formatTau) Tau(1:ngz,i);
      END DO
    END IF
  CASE ("InputNew.txt") ! save updated input file InputNew.txt ---------------------------------------------------------|
      mSize=63;
      ALLOCATE(mList(mSize))
      mList=(/"runDescription ","reactorType    ","thermalType    ","numericalMethod","ecm            ", &
              "tiRun          ","tfRun          ","nts            ","ngc            ","nsc            ", &
              "ncs            ","Component      ","MWGas          ","rMW            ","rDeff          ", &
              "rTemp          ","iniT           ","ld             ","lz             ","nGrid          ", &
              "catP           ","catW           ","catWF          ","catMW          ","catS           ", &
```

Table E.5 Continued

```
      "catD          ","catSF          ","nce           ","Element       ","F             ", &
      "nrer          ","MechE          ","ERC           ","nrar          ","MechA         ", &
      "ARC           ","ARO            ","nrur          ","MechU         ","URC           ", &
      "inputIC       ","inputFIC       ","inputIT       ","inputFIT      ","IniGP         ", &
      "IniSC         ","CBC            ","BCPmax        ","BCPmin        ","BCW           ", &
      "BCF           ","BCL            ","BCCV          ","BCRQ          ","BCRM          ", &
      "BCRT          ","BCT            ","TBC           ","BCTmax        ","BCTmin        ", &
      "BCTW          ","BCTF           ","BCTL          ");
  OutputUnit=10; FormatType=1;
    OPEN(UNIT=OutputUnit, FILE=trim(run)//"//flag, STATUS="REPLACE")
    DO i=1,mSize; CALL sWrite(mList(i),OutputUnit,FormatType); END DO
  CLOSE(UNIT=OutputUnit)
    DEALLOCATE(mList)
CASE ("Output.xml") ! Save file Output.xml ------------------------------------------------------------------------------------------------------------------|
    OutputUnit=10; FormatType=2; ! xml
    OPEN(UNIT=OutputUnit, FILE=trim(run)//"//flag, STATUS="REPLACE")
      WRITE(OutputUnit,"(A38)") '<?xml version="1.0" encoding="UTF-8"?>'
      WRITE(OutputUnit,"(A8)") '<Module>'
      CALL sWrite("MODULE",OutputUnit,FormatType);
      WRITE(OutputUnit,"(A9)") '</Module>'
    CLOSE(UNIT=OutputUnit)
CASE ("Output.txt") ! Save file Output.txt ------------------------------------------------------------------------------------------------------------|
    OutputUnit=10; FormatType=1;
    OPEN(UNIT=OutputUnit, FILE=trim(run)//"//flag, STATUS="REPLACE")
    CALL sWrite("MODULE",OutputUnit,FormatType)
  CLOSE(UNIT=OutputUnit)
CASE ("FinalConcentration.txt") ! Save file FinalConcentration.txt ------------------------------------------------------------------------------------|
  DO i=gStart,gEnd
    DO j=1,ngz
      C(j,i)=(C(j,i)*maxGC)*idealGC*T(j); ! convert to gas pressure (no change required for surface coverage)
    END DO
  END DO
    OutputUnit=10; FormatType=0;
    OPEN(UNIT=OutputUnit, FILE=trim(run)//"//flag, STATUS="REPLACE")
    CALL sWrite("C",OutputUnit,FormatType)
  CLOSE(UNIT=OutputUnit)
CASE ("InitialConcentration.txt") ! Save file InitialConcentration.txt --------------------------------------------------------------------------------|
    OutputUnit=10; FormatType=0;
    OPEN(UNIT=OutputUnit, FILE=trim(run)//"//flag, STATUS="REPLACE")
    CALL sWrite("C",OutputUnit,FormatType)
  CLOSE(UNIT=OutputUnit)
CASE ("InitialTemperature.txt") ! Save file InitialTemperature.txt ------------------------------------------------------------------------------------|
    OutputUnit=10; FormatType=0;
    OPEN(UNIT=OutputUnit, FILE=trim(run)//"//flag, STATUS="REPLACE")
    CALL sWrite("T",OutputUnit,FormatType)
  CLOSE(UNIT=OutputUnit)
```

Table E.5 Continued

```
  CASE ("FinalTemperature.txt") ! Save file FinalTemperature.txt ------------------------------------------------------------------------
       OutputUnit=10; FormatType=0;
       OPEN(UNIT=OutputUnit, FILE=trim(run)//"//flag, STATUS="REPLACE")
          CALL sWrite("T",OutputUnit,FormatType)
       CLOSE(UNIT=OutputUnit)
  END SELECT
  END SUBROUTINE sSave
```

Table E.6 sWrite.f90

```
!-----------------------------------------------------------------------
! Subroutine sWrite
! Description: Write formated data to an output unit.
! Created by Hsu-Wen Hsiao on 3/19/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   varName: variable name
!   OutputUnit: unit to write variables
!   FormatType: format of variable, 0:variable only, 1:variable with description, 2:XML
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sWrite(varName,OutputUnit,FormatType)
USE mBoundary
USE mDynamic
USE mElement
USE mInformation
USE mParameter
USE mReaction
USE mRuntime
USE mSave
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: varName
INTEGER, INTENT(IN) :: OutputUnit, FormatType
CHARACTER(LEN=256) :: varInfo
CHARACTER(LEN=32), DIMENSION(:), ALLOCATABLE :: mList
INTEGER :: mSize
INTEGER :: i
SELECT CASE (varName)
  CASE ("MODULE") ! all module ------------------------------------------------------------------------------------------------------------------------
    mSize=8;
    ALLOCATE(mList(mSize))
      mList=(/"mBoundary","mDynamic","mElement","mInformation","mParameter","mReaction","mRuntime","mSave"/);
      DO i=1,mSize; CALL sWrite(mList(i),OutputUnit,FormatType); END DO
    DEALLOCATE(mList)
  CASE("mBoundary") ! module mStoichiometry ------------------------------------------------------------------------------------------------------------
    varInfo="Variables and parameters of boundary condition"
    mSize=10;
    ALLOCATE(mList(mSize))
    mList=(/"BCCmax","BCCmin","BCW","BCF","BCL", &
            "BCTmax","BCTmin","BCTW","BCTF","BCTL"/);
    CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
  DEALLOCATE(mList)
    CASE ("BCCmax")
    varInfo="Maximum modulated gas concentration at boundary (Pa)";
    CALL sWriteDoubleR2(varName,varInfo,2,ngc,BCCmax,OutputUnit,FormatType)
    CASE ("BCCmin")
```

Table E.6 Continued

```
    varInfo="Minimum modulated gas concentration at boundary (Pa)";
    CALL sWriteDoubleR2(varName,varInfo,2,ngc,BCCmin,OutputUnit,FormatType)
     CASE ("BCW")
    varInfo="Waveform of modulated gas component at boundary (1:Constant, 2:Sine, 3:Square)";
    CALL sWriteIntegerR2(varName,varInfo,2,ngc,BCW,OutputUnit,FormatType)
     CASE ("BCF")
    varInfo="Frequence of modulated gas component at boundary (Hz)";
    CALL sWriteDoubleR2(varName,varInfo,2,ngc,BCF,OutputUnit,FormatType)
     CASE ("BCL")
    varInfo="Phase lag of modulated gas component at boundary (0~1 Wavelength)";
    CALL sWriteDoubleR2(varName,varInfo,2,ngc,BCL,OutputUnit,FormatType)
     CASE ("BCTmax")
    varInfo="Maximum modulated temperature at boundary (Pa)";
    CALL sWriteDoubleR1(varName,varInfo,2,BCTmax,OutputUnit,FormatType)
     CASE ("BCTmin")
    varInfo="Minimum modulated temperature at boundary (Pa)";
    CALL sWriteDoubleR1(varName,varInfo,2,BCTmin,OutputUnit,FormatType)
     CASE ("BCTW")
    varInfo="Waveform of modulated temperature at boundary (1:Constant, 2:Sine, 3:Square)";
    CALL sWriteIntegerR1(varName,varInfo,2,BCTW,OutputUnit,FormatType)
     CASE ("BCTF")
    varInfo="Frequence of modulated temperature at boundary (Hz)";
    CALL sWriteDoubleR1(varName,varInfo,2,BCTF,OutputUnit,FormatType)
     CASE ("BCTL")
    varInfo="Phase lag of modulated temperature at boundary (0~1 Wavelength)";
    CALL sWriteDoubleR1(varName,varInfo,2,BCTL,OutputUnit,FormatType)
CASE("mDynamic") ! module mDynamic ------------------------------------------------------------------------------------------------------------------------|
    varInfo="Dynamic variables and parameters"
    mSize=46;
    ALLOCATE(mList(mSize))
    mList=(/"reactorType","thermalType","ecm","CBC","TBC", &
            "ngc","nsc","ncs","ntsc","ntc","gStart","gEnd","sStart","sEnd","cStart","cEnd", &
             "iStart","iEnd","dStart","dEnd", &
             "tiRun","tfRun","tRun","dtRun","nts","tRuntime", &
            "ngz","rStart","rEnd","nmStart","nmEnd","dZ", &
            "C","T","LC","NC","Cin","Tin","Cb","Tb", &
             "alpha","Tau","Taub","Tauin", &
             "dtLevel", &
             "idealGC"/);
    CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
    DEALLOCATE(mList)
     CASE ("reactorType")
    varInfo="Model of reactor (1: Dynamic diffusion reactor (DDR), 2: Continuous stir tank reactor (CSTR))";
    CALL sWriteIntegerR0(varName,varInfo,reactorType,OutputUnit,FormatType)
     CASE ("thermalType")
    varInfo="Thermal property of this run (1: Isothermal, 2: Temperature ramp, 3: Non-isothermal)";
```

Table E.6 Continued

```
    CALL sWriteIntegerR0(varName,varInfo,thermalType,OutputUnit,FormatType)
    CASE ("ecm")
      varInfo="Use the algorithm of element constraint method (0: Don't use, 1: Catalytic sites only, 2: Use ecm algorithm)";
    CALL sWriteIntegerR0(varName,varInfo,ecm,OutputUnit,FormatType)
    CASE ("CBC")
    varInfo="Type of concentration boundary condition (1: Modulated (Dirichlet-type), "// &
        "2: Zero mass flux (von Neumann-type), 3: Balanced mass flux (Danckwerts' type))";
    CALL sWriteIntegerR1(varName,varInfo,2,CBC,OutputUnit,FormatType)
    CASE ("TBC")
    varInfo="Temperature boundary condition (0:Isothermal, 1:Modulated (Dirichlet-type), "// &
        "2:Zero heat flux (von Neumann-type), 3:Balanced heat flux (Danckwerts' type))";
    CALL sWriteIntegerR1(varName,varInfo,2,TBC,OutputUnit,FormatType)
    CASE ("ngc")
    varInfo="Number of gas component";
    CALL sWriteIntegerR0(varName,varInfo,ngc,OutputUnit,FormatType)
    CASE ("nsc")
    varInfo="Number of solid component";
    CALL sWriteIntegerR0(varName,varInfo,nsc,OutputUnit,FormatType)
    CASE ("ncs")
    varInfo="Number of catalytic site";
    CALL sWriteIntegerR0(varName,varInfo,ncs,OutputUnit,FormatType)
    CASE ("ntsc")
    varInfo="Number of total solid component";
    CALL sWriteIntegerR0(varName,varInfo,ntsc,OutputUnit,FormatType)
    CASE ("ntc")
    varInfo="Number of total component";
    CALL sWriteIntegerR0(varName,varInfo,ntc,OutputUnit,FormatType)
    CASE ("gStart")
    varInfo="Starting position of gas component";
    CALL sWriteIntegerR0(varName,varInfo,gStart,OutputUnit,FormatType)
    CASE ("gEnd")
    varInfo="Ending position of gas component";
    CALL sWriteIntegerR0(varName,varInfo,gEnd,OutputUnit,FormatType)
    CASE ("sStart")
    varInfo="Starting position of solid component";
    CALL sWriteIntegerR0(varName,varInfo,sStart,OutputUnit,FormatType)
    CASE ("sEnd")
    varInfo="Ending position of solid component";
    CALL sWriteIntegerR0(varName,varInfo,sEnd,OutputUnit,FormatType)
    CASE ("cStart")
    varInfo="Starting position of catalytic site";
    CALL sWriteIntegerR0(varName,varInfo,cStart,OutputUnit,FormatType)
    CASE ("cEnd")
    varInfo="Ending position of catalytic site";
    CALL sWriteIntegerR0(varName,varInfo,cEnd,OutputUnit,FormatType)
    CASE ("iStart")
```

Table E.6 Continued

```
    varInfo="Starting position of independent component";
    CALL sWriteIntegerR0(varName,varInfo,iStart,OutputUnit,FormatType)
     CASE ("iEnd")
    varInfo="Ending position of independent component";
    CALL sWriteIntegerR0(varName,varInfo,iEnd,OutputUnit,FormatType)
     CASE ("dStart")
    varInfo="Starting position of dependent component";
    CALL sWriteIntegerR0(varName,varInfo,dStart,OutputUnit,FormatType)
     CASE ("dEnd")
    varInfo="Ending position of dependent component";
    CALL sWriteIntegerR0(varName,varInfo,dEnd,OutputUnit,FormatType)
     CASE ("tiRun")
    varInfo="Initial time (s)";
       CALL sWriteDoubleR0(varName,varInfo,tiRun,OutputUnit,FormatType)
     CASE ("tfRun")
    varInfo="Final time (s)";
       CALL sWriteDoubleR0(varName,varInfo,tfRun,OutputUnit,FormatType)
     CASE ("tRun")
    varInfo="Runtime (s)";
       CALL sWriteDoubleR0(varName,varInfo,tRun,OutputUnit,FormatType)
     CASE ("dtRun")
    varInfo="Size of timestep (s)";
       CALL sWriteDoubleR0(varName,varInfo,dtRun,OutputUnit,FormatType)
     CASE ("nts")
       varInfo="Number of time step";
    CALL sWriteIntegerR0(varName,varInfo,nts,OutputUnit,FormatType)
     CASE ("tRuntime")
    varInfo="Current runtime (s)";
       CALL sWriteDoubleR0(varName,varInfo,tRuntime,OutputUnit,FormatType)
     CASE ("ngz")
    varInfo="Number of grid in Z-direction";
    CALL sWriteIntegerR0(varName,varInfo,ngz,OutputUnit,FormatType)
     CASE ("rStart")
    varInfo="Starting grid of reactor in Z-direction";
    CALL sWriteIntegerR0(varName,varInfo,rStart,OutputUnit,FormatType)
     CASE ("rEnd")
    varInfo="Ending grid of reactor in Z-direction";
    CALL sWriteIntegerR0(varName,varInfo,rEnd,OutputUnit,FormatType)
     CASE ("nmStart")
    varInfo="Starting grid of reactor need to be solved numerically";
    CALL sWriteIntegerR0(varName,varInfo,nmStart,OutputUnit,FormatType)
     CASE ("nmEnd")
    varInfo="Ending grid of reactor need to be solved numerically";
    CALL sWriteIntegerR0(varName,varInfo,nmEnd,OutputUnit,FormatType)
     CASE ("dZ")
    varInfo="Delta Z (dimensionless)";
```

Table E.6 Continued

```
       CALL sWriteDoubleR0(varName,varInfo,dZ,OutputUnit,FormatType)
     CASE ("C")
    varInfo="Dimensioless concentration";
    CALL sWriteDoubleR2(varName,varInfo,ngz,ntc,C,OutputUnit,FormatType)
     CASE ("T")
    varInfo="Temperature (K)";
    CALL sWriteDoubleR1(varName,varInfo,ngz,T,OutputUnit,FormatType)
     CASE ("LC")
    varInfo="Linear part of equation";
    CALL sWriteDoubleR2(varName,varInfo,ngz,ntc,LC,OutputUnit,FormatType)
     CASE ("NC")
    varInfo="Nonlinear part of equation";
    CALL sWriteDoubleR2(varName,varInfo,ngz,ntc,NC,OutputUnit,FormatType)
     CASE ("alpha")
    varInfo="Surface to gas capacity ratio (Dimensionless)";
       CALL sWriteDoubleR0(varName,varInfo,alpha,OutputUnit,FormatType)
     CASE ("Tau")
    varInfo="Diffusion time constant (Dimensionless)";
    CALL sWriteDoubleR2(varName,varInfo,ngz,ngc,Tau,OutputUnit,FormatType)
     CASE ("dtLevel")
    varInfo="Level of sub-timestep";
    CALL sWriteIntegerR0(varName,varInfo,dtlevel,OutputUnit,FormatType)
     CASE ("idealGC")
    varInfo="Ideal gas constant (J/K/gmol)";
       CALL sWriteDoubleR0(varName,varInfo,idealGC,OutputUnit,FormatType)
 CASE("mElement") ! module mElement -----------------------------------------------------------------------------------------|
     varInfo="Variables and parameters for element constraint method"
     mSize=4;
     ALLOCATE(mList(mSize))
     mList=(/"nce","nic","ndc","M"/);
     CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
   DEALLOCATE(mList)
   IF (ecm /= 0) THEN
       mSize=15;
       ALLOCATE(mList(mSize))
       mList=(/"MIMD","MI","MD","MDi","MDiM","MDiMYI","MDiMYL","MDiMY","MDiMI","MDiMIYI","MDiMIYL","MDiMIY","MscI","MscL","Msc"/);
       CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
     DEALLOCATE(mList)
   END IF
     CASE ("nce")
    varInfo="Number of chemical element";
    CALL sWriteIntegerR0(varName,varInfo,nce,OutputUnit,FormatType)
     CASE ("nic")
    varInfo="Number of independent component";
    CALL sWriteIntegerR0(varName,varInfo,nic,OutputUnit,FormatType)
     CASE ("ndc")
```

Table E.6 Continued

```
    varInfo="Number of dependent component";
    CALL sWriteIntegerR0(varName,varInfo,ndc,OutputUnit,FormatType)
     CASE ("M")
    varInfo="Formula matrix";
    CALL sWriteIntegerR2(varName,varInfo,nce,ntc,M,OutputUnit,FormatType)
     CASE ("MIMD")
    varInfo="Linearly independent formula matrix";
    CALL sWriteIntegerR2(varName,varInfo,ndc,ntc,MIMD,OutputUnit,FormatType)
     CASE ("MI")
    varInfo="Formula matrix of independent component";
    CALL sWriteIntegerR2(varName,varInfo,ndc,nic,MI,OutputUnit,FormatType)
     CASE ("MD")
    varInfo="Formula matrix of dependent component";
    CALL sWriteIntegerR2(varName,varInfo,ndc,ndc,MD,OutputUnit,FormatType)
     CASE ("MDi")
    varInfo="Inverse of the formula matrix of dependent component, MD^-1";
    CALL sWriteIntegerR2(varName,varInfo,ndc,ndc,MD,OutputUnit,FormatType)
     CASE ("MDiM")
    varInfo="Product of the inverse of the formula matrix of dependent component and the formula matrix, (MD^-1)*M";
    CALL sWriteDoubleR2(varName,varInfo,ndc,ntc,MDiM,OutputUnit,FormatType)
     CASE ("MDiMYI")
    varInfo="Integer vector MDiM";
    CALL sWriteIntegerR1(varName,varInfo,ndc+1,MDiMYI,OutputUnit,FormatType)
     CASE ("MDiMYL")
    varInfo="Location vector MDiM";
    CALL sWriteIntegerR1(varName,varInfo,MDiMYI(ndc+1)-1,MDiMYL,OutputUnit,FormatType)
     CASE ("MDiMY")
    varInfo="Yale's sparse storage of MDiM (row-wise)";
    CALL sWriteDoubleR1(varName,varInfo,MDiMYI(ndc+1)-1,MDiMY,OutputUnit,FormatType)
     CASE ("MDiMI")
    varInfo="Product of the inverse of the formula matrix of dependent component and the formula matrix of independent "// &
          "component, (MD^-1)*MI";
    CALL sWriteDoubleR2(varName,varInfo,ndc,nic,MDiMI,OutputUnit,FormatType)
     CASE ("MDiMIYI")
    varInfo="Integer vector MDiMI";
    CALL sWriteIntegerR1(varName,varInfo,ndc+1,MDiMIYI,OutputUnit,FormatType)
     CASE ("MDiMIYL")
    varInfo="Location vector MDiMI";
    CALL sWriteIntegerR1(varName,varInfo,MDiMIYI(ndc+1)-1,MDiMIYL,OutputUnit,FormatType)
     CASE ("MDiMIY")
    varInfo="Yale's sparse storage of MDiMI (row-wise)";
    CALL sWriteDoubleR1(varName,varInfo,MDiMIYI(ndc+1)-1,MDiMIY,OutputUnit,FormatType)
   CASE ("MscI")
       varInfo="Integer vector Msc";
       CALL sWriteIntegerR1(varName,varInfo,ncs+1,MscI,OutputUnit,FormatType)
   CASE ("MscL")
```

Table E.6 Continued

```
    varInfo="Location vector Msc";
    CALL sWriteIntegerR1(varName,varInfo,MscI(ncs+1)-1,MscL,OutputUnit,FormatType)
  CASE ("Msc")
    varInfo="Formula matrix of catalytic sites in Yale's sparse format.";
    CALL sWriteIntegerR1(varName,varInfo,MscI(ncs+1)-1,Msc,OutputUnit,FormatType)
CASE ("mInformation") ! module mInformation ----------------------------------------------------------------------------------------------------|
  varInfo="Information of runs"
    mSize=15;
    ALLOCATE(mList(mSize))
    mList=(/"verTransCat     ","verDate       ","stopRun       ","stopInfo      ","TransCatDir   ", &
            "nRun            ","iRun          ","RunList       ","numericalMethod","run           ", &
            "runDescription  ","inputIC       ","inputIT       ","inputFIC      ","inputFIT      "/);
  CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
  DEALLOCATE(mList)
CASE ("verTransCat")
  varInfo="Version of TransCat";
  CALL sWriteStringR0(varName,varInfo,verTransCat,OutputUnit,FormatType)
CASE ("verDate")
  varInfo="Last update of this version";
  CALL sWriteStringR0(varName,varInfo,verDate,OutputUnit,FormatType)
CASE ("stopRun")
  varInfo="Flag of error message to stop run, (0: no error detected; 1: error detected)";
  CALL sWriteLogicalR0(varName,varInfo,stopRun,OutputUnit,FormatType)
CASE ("stopInfo")
  varInfo="Error message to stop run";
  CALL sWriteStringR0(varName,varInfo,stopInfo,OutputUnit,FormatType)
CASE ("TransCatDir")
  varInfo="Directory of current running TransCat executable";
  CALL sWriteStringR0(varName,varInfo,TransCatDir,OutputUnit,FormatType)
CASE ("nRun")
  varInfo="Number of runs";
  CALL sWriteIntegerR0(varName,varInfo,nRun,OutputUnit,FormatType)
CASE ("iRun")
  varInfo="The i'th run";
  CALL sWriteIntegerR0(varName,varInfo,iRun,OutputUnit,FormatType)
CASE ("RunList")
  varInfo="List of runs";
    IF (nRun==1) THEN
      CALL sWriteStringR0(varName,varInfo,RunList(1),OutputUnit,FormatType)
    ELSE
      CALL sWriteStringR1(varName,varInfo,nRun,RunList,OutputUnit,FormatType)
    END IF
CASE ("numericalMethod")
  varInfo="Numerical method for this run (EE, RK2, RK3, RK4, CNEE, CNRK2, CNRK3, IE, BDF2)";
  CALL sWriteStringR0(varName,varInfo,numericalMethod,OutputUnit,FormatType)
CASE ("run")
```

Table E.6 Continued

```
  varInfo="ID of this run";
     CALL sWriteStringR0(varName,varInfo,run,OutputUnit,FormatType)
CASE ("runDescription")
  varInfo="Description of this run (500 characters maximum)";
  CALL sWriteStringR0(varName,varInfo,runDescription,OutputUnit,FormatType)
CASE ("inputIC")
     varInfo="Input initial concentration from file (0:No, 1:Yes)";
     CALL sWriteIntegerR0(varName,varInfo,inputIC,OutputUnit,FormatType)
CASE ("inputIT")
     varInfo="Input initial temperature from file (0:No, 1:Yes)";
     CALL sWriteIntegerR0(varName,varInfo,inputIC,OutputUnit,FormatType)
CASE ("inputFIC")
  varInfo="Filename of input initial concentration";
     CALL sWriteStringR0(varName,varInfo,inputFIC,OutputUnit,FormatType)
CASE ("inputFIT")
  varInfo="Filename of input initial temperature";
     CALL sWriteStringR0(varName,varInfo,inputFIT,OutputUnit,FormatType)
CASE("mParameter") ! module mParameter -------------------------------------------------------------------------------------------|
     varInfo="Parameters of this run"
     mSize=51;
     ALLOCATE(mList(mSize))
     mList=(/"Component","Element  ", &
             "IniGP    ","maxGC    ","IniSC    ","iniT     ", &
               "Deff     ","MWGas    ","rMW      ","rDeff    ","rTemp    ", &
               "lz       ","ld       ","nGrid    ","Z        ", &
               "catP     ","catW     ","catWF    ","catMW    ","catS     ","catD     ","catSF    ","catSC    ","catCS    ", &
               "totSC    ","totCS    ","totSA    ","totRCA   ","pi       ","totRV    ","totGV    ","totSV    ", &
             "MechE    ","ERC      ", &
             "MechA    ","ARC      ","ARO      ", &
             "MechU    ","URC      ", &
             "ncr      ","Mech     ","EAURC    ","RIM      ","RIV      ", &
             "BCPmax   ","BCPmin   ","BCCV     ","BCRQ     ","BCRM     ","BCRT     ","BCT      "/);
     CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
  DEALLOCATE(mList)
     CASE ("Component")
       varInfo="Chemical species";
       CALL sWriteStringR1(varName,varInfo,ntc,Component,OutputUnit,FormatType)
     CASE ("Element")
       varInfo="Chemical element";
       CALL sWriteStringR1(varName,varInfo,nce,Element,OutputUnit,FormatType)
     CASE ("IniGP")
    varInfo="Initial gas pressure (Pa)";
       CALL sWriteDoubleR1(varName,varInfo,ngc,IniGP,OutputUnit,FormatType);
     CASE ("maxGC")
    varInfo="Maximum gas concentration (gmol/m^3)";
       CALL sWriteDoubleR0(varName,varInfo,maxGC,OutputUnit,FormatType)
```

Table E.6 Continued

```
    CASE ("IniSC")
    varInfo="Initial surface converage (Dimensionless)";
      CALL sWriteDoubleR1(varName,varInfo,ntsc,IniSC,OutputUnit,FormatType);
    CASE ("iniT")
    varInfo="Initial temperature (K)";
      CALL sWriteDoubleR0(varName,varInfo,iniT,OutputUnit,FormatType)
    CASE ("Deff")
    varInfo="Effective diffusion coefficient (m^2/s)";
    CALL sWriteDoubleR2(varName,varInfo,ngz,ngc,Deff,OutputUnit,FormatType)
    CASE ("MWGas")
    varInfo="Molecular weight of gas component (g/gmol)";
      CALL sWriteDoubleR1(varName,varInfo,ngc,MWGas,OutputUnit,FormatType)
    CASE ("rMW")
    varInfo="Molecular weight of reference gas (g/gmol)";
      CALL sWriteDoubleR0(varName,varInfo,rMW,OutputUnit,FormatType)
    CASE ("rDeff")
    varInfo="Effective diffusion coefficient of reference gas (m^2/s)";
      CALL sWriteDoubleR0(varName,varInfo,rDeff,OutputUnit,FormatType)
    CASE ("rTemp")
    varInfo="Temperature of reference gas (K)";
      CALL sWriteDoubleR0(varName,varInfo,rTemp,OutputUnit,FormatType)
    CASE ("lz")
    varInfo="Reactor thickness (m)";
      CALL sWriteDoubleR0(varName,varInfo,lz,OutputUnit,FormatType)
    CASE ("ld")
    varInfo="Reactor diameter (m)";
      CALL sWriteDoubleR0(varName,varInfo,ld,OutputUnit,FormatType)
    CASE ("nGrid")
    varInfo="Number of grid points in reactor";
    CALL sWriteIntegerR0(varName,varInfo,nGrid,OutputUnit,FormatType)
    CASE ("Z")
    varInfo="Grid points on Z (dimensionless)";
      CALL sWriteDoubleR1(varName,varInfo,ngz,Z,OutputUnit,FormatType)
    CASE ("catP")
    varInfo="Porosity (Dimensionless)";
      CALL sWriteDoubleR0(varName,varInfo,catP,OutputUnit,FormatType)
    CASE ("catW")
    varInfo="Weight of catalyst (kg)";
      CALL sWriteDoubleR0(varName,varInfo,catW,OutputUnit,FormatType)
    CASE ("catWF")
    varInfo="Weight fraction of catalyst (dimensionless)";
      CALL sWriteDoubleR1(varName,varInfo,ncs,catWF,OutputUnit,FormatType)
    CASE ("catMW")
    varInfo="Molecular weight of catalyst (kg/gmol)";
      CALL sWriteDoubleR1(varName,varInfo,ncs,catMW,OutputUnit,FormatType)
    CASE ("catS")
```

Table E.6 Continued

```
    varInfo="Surface area occupied per mole catalyst on surface (m^2/gmol)";
      CALL sWriteDoubleR1(varName,varInfo,ncs,catS,OutputUnit,FormatType)
    CASE ("catD")
    varInfo="Dispersion of catalyst (fraction of catalyst exposed, "// &
        "mole of catalyst atom on surface/mole of catalyst atom in a particle)";
      CALL sWriteDoubleR1(varName,varInfo,ncs,catD,OutputUnit,FormatType)
    CASE ("catSF")
    varInfo="Fraction of catalytic site to catalyst, molar ratio of linear, bridgebound or triply coordinated site"// &
        " to catalyst atom on surface)";
      CALL sWriteDoubleR1(varName,varInfo,ncs,catSF,OutputUnit,FormatType)
    CASE ("catSC")
    varInfo="Surface concentration of catalytic sites (gmol/m^2)";
      CALL sWriteDoubleR1(varName,varInfo,ncs,catSC,OutputUnit,FormatType)
    CASE ("catCS")
    varInfo="Moles of each catalytic site in reactor (gmol)";
      CALL sWriteDoubleR1(varName,varInfo,ncs,catCS,OutputUnit,FormatType)
    CASE ("totSC")
    varInfo="Total surface concentration of catalytic sites (gmol/m^2)";
      CALL sWriteDoubleR0(varName,varInfo,totSC,OutputUnit,FormatType)
    CASE ("totCS")
    varInfo="Total surface catalytic sites in reactor (gmol)";
      CALL sWriteDoubleR0(varName,varInfo,totCS,OutputUnit,FormatType)
    CASE ("totSA")
    varInfo="Total active surface area of catalyst (m^2)";
      CALL sWriteDoubleR0(varName,varInfo,totSA,OutputUnit,FormatType)
    CASE ("totRCA")
    varInfo="Cross-sectional area between reactor and boundary chamber (m^2)";
      CALL sWriteDoubleR0(varName,varInfo,totRCA,OutputUnit,FormatType)
    CASE ("pi")
    varInfo="Ratio of any circle's circumference to its diameter (radian)";
      CALL sWriteDoubleR0(varName,varInfo,pi,OutputUnit,FormatType)
    CASE ("totRV")
    varInfo="Total volume of reactor (m^3)";
      CALL sWriteDoubleR0(varName,varInfo,totRV,OutputUnit,FormatType)
    CASE ("totGV")
    varInfo="Gas volume of reactor (m^3)";
      CALL sWriteDoubleR0(varName,varInfo,totGV,OutputUnit,FormatType)
    CASE ("totSV")
    varInfo="Solid volume of reactor (m^3)";
      CALL sWriteDoubleR0(varName,varInfo,totSV,OutputUnit,FormatType)
    CASE ("nrer")
      IF (nrer > 0) THEN
      varInfo="Number of elementary rate equation";
      CALL sWriteIntegerR0(varName,varInfo,nrer,OutputUnit,FormatType)
    END IF
    CASE ("MechE")
```

Table E.6 Continued

```
    IF (nrar > 0) THEN
    varInfo="Detailed mechanism for elementary rate equation";
    CALL sWriteIntegerR2(varName,varInfo,nrer,ntc,MechE,OutputUnit,FormatType)
  END IF
  CASE ("ERC")
    IF (nrer > 0) THEN
    varInfo="Arrhenius' rate constant of elementary rate equation, in the order of Af, Nf, Ef, Ar, Nr and Er";
    CALL sWriteDoubleR2(varName,varInfo,nrer,6,ERC,OutputUnit,FormatType)
  END IF
  CASE ("nrar")
    IF (nrar > 0) THEN
    varInfo="Number of algebraic rate equation";
    CALL sWriteIntegerR0(varName,varInfo,nrar,OutputUnit,FormatType)
  END IF
  CASE ("MechA")
    IF (nrar > 0) THEN
    varInfo="Detailed mechanism for algebraic rate equation";
    CALL sWriteIntegerR2(varName,varInfo,nrar,ntc,MechA,OutputUnit,FormatType)
  END IF
  CASE ("ARC")
    IF (nrar > 0) THEN
    varInfo="Arrhenius' rate constant of algebraic rate equation, in the order of Af, Nf, Ef, Ar, Nr and Er";
    CALL sWriteDoubleR2(varName,varInfo,nrar,6,ARC,OutputUnit,FormatType)
  END IF
  CASE ("ARO")
    IF (nrar > 0) THEN
    varInfo="Order of component for algebraic rate equation (forward and reverse rate)";
    CALL sWriteDoubleR2(varName,varInfo,nrar,2*ntc,ARO,OutputUnit,FormatType)
  END IF
  CASE ("nrur")
    IF (nrur > 0) THEN
    varInfo="Number of user-defined rate equation";
    CALL sWriteIntegerR0(varName,varInfo,nrur,OutputUnit,FormatType)
  END IF
  CASE ("MechU")
    IF (nrur > 0) THEN
    varInfo="Detailed mechanism for user-defined rate equation";
    CALL sWriteIntegerR2(varName,varInfo,nrur,ntc,MechU,OutputUnit,FormatType)
  END IF
  CASE ("URC")
    IF (nrur > 0) THEN
  varInfo="Arrhenius' rate constant of user-defined rate equation, in the order of Af, Nf, Ef, Ar, Nr and Er";
    CALL sWriteDoubleR2(varName,varInfo,nrur,6,URC,OutputUnit,FormatType)
  END IF
  CASE ("ncr")
  varInfo="Number of chemical reaction";
```

Table E.6 Continued

```
    CALL sWriteIntegerR0(varName,varInfo,ncr,OutputUnit,FormatType)
    CASE ("Mech")
  varInfo="Combined mechanism of elementary, algebraic and user-defined rate equation";
  CALL sWriteIntegerR2(varName,varInfo,ncr,ntc,Mech,OutputUnit,FormatType)
    CASE ("EAURC")
  varInfo="Arrhenius' rate constant of combined elementary, algebraic and user-defined rate equation, "// &
        "in the order of Af, Nf, Ef, Ar, Nr and Er";
    CALL sWriteDoubleR2(varName,varInfo,ncr,6,EAURC,OutputUnit,FormatType)
    CASE ("RIM")
  varInfo="Reaction identity matrix";
  CALL sWriteIntegerR2(varName,varInfo,2,ncr,RIM,OutputUnit,FormatType)
    CASE ("RIV")
  varInfo="Reaction identity vector";
  CALL sWriteIntegerR1(varName,varInfo,ntr,RIV,OutputUnit,FormatType)
    CASE ("BCPmax")
  varInfo="Maximum gas pressure at boundary (Pa)";
  CALL sWriteDoubleR2(varName,varInfo,2,ngc,BCPmax,OutputUnit,FormatType)
    CASE ("BCPmin")
  varInfo="Minimum gas pressure at boundary (Pa)";
  CALL sWriteDoubleR2(varName,varInfo,2,ngc,BCPmin,OutputUnit,FormatType)
    CASE ("BCCV")
  varInfo="Volume of boundary chamber (m3)";
     CALL sWriteDoubleR1(varName,varInfo,2,BCCV,OutputUnit,FormatType);
    CASE ("BCRQ")
  varInfo="Volume flow rate of reference gas in boundary chamber (m3/s)";
     CALL sWriteDoubleR1(varName,varInfo,2,BCRQ,OutputUnit,FormatType);
    CASE ("BCRM")
  varInfo="Molecular weight of reference gas in boundary chamber (kg/mol)";
     CALL sWriteDoubleR1(varName,varInfo,2,BCRM,OutputUnit,FormatType);
    CASE ("BCRT")
  varInfo="Temperature of reference gas in boundary chamber (K)";
     CALL sWriteDoubleR1(varName,varInfo,2,BCRT,OutputUnit,FormatType);
    CASE ("BCT")
  varInfo="Temperature of boundary chamber (K)";
     CALL sWriteDoubleR1(varName,varInfo,2,BCT,OutputUnit,FormatType);
CASE ("mReaction")  ! module mReaction ------------------------------------------------------------------------------------------------------------------------------------------------------|
    varInfo="Variables and parameters of reaction"
    mSize=34;
    ALLOCATE(mList(mSize))
    mList=(/"ntr","near","nrer","nrar","nrur","erStart","erEnd","arStart","arEnd","urStart","urEnd", &
           "S","SYI","SYL","SY","SJRI","SJRL","SJYI","SJYRL","SJY", &
           "AC","RC","RI","RL","RP","R", &
           "DRDC","JYI","JYL","JR","JRY", &
           "JRC","JRI","JRL","JRP"/);
    CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
  DEALLOCATE(mList)
```

Table E.6 Continued

```
    CASE ("ntr")
    varInfo="Number of total rate equations with nonzero rate constant";
    CALL sWriteIntegerR0(varName,varInfo,ntr,OutputUnit,FormatType)
    CASE ("near")
    varInfo="Number of elementary and algebraic rate equations with nonzero rate constant";
    CALL sWriteIntegerR0(varName,varInfo,near,OutputUnit,FormatType)
    CASE ("erStart")
    varInfo="Starting position of elementary rate";
    CALL sWriteIntegerR0(varName,varInfo,erStart,OutputUnit,FormatType)
    CASE ("erEnd")
    varInfo="Ending position of elementary rate";
    CALL sWriteIntegerR0(varName,varInfo,erEnd,OutputUnit,FormatType)
    CASE ("arStart")
    varInfo="Starting position of algebraic rate";
    CALL sWriteIntegerR0(varName,varInfo,arStart,OutputUnit,FormatType)
    CASE ("arEnd")
    varInfo="Ending position of algebraic rate";
    CALL sWriteIntegerR0(varName,varInfo,arEnd,OutputUnit,FormatType)
    CASE ("urStart")
    varInfo="Starting position of user-defined rate";
    CALL sWriteIntegerR0(varName,varInfo,urStart,OutputUnit,FormatType)
    CASE ("urEnd")
    varInfo="Ending position of user-defined rate";
    CALL sWriteIntegerR0(varName,varInfo,urEnd,OutputUnit,FormatType)
    CASE ("S")
    varInfo="Stoichiometric matrix";
    CALL sWriteIntegerR2(varName,varInfo,ntc,ntr,S,OutputUnit,FormatType)
    CASE ("SYI")
    varInfo="Integer vector of stoichiometric matrix";
    CALL sWriteIntegerR1(varName,varInfo,ntc+1,SYI,OutputUnit,FormatType)
    CASE ("SYL")
    varInfo="Locator vector of stoichiometric matrix";
    CALL sWriteIntegerR1(varName,varInfo,SYI(ntc+1)-1,SYL,OutputUnit,FormatType)
    CASE ("SY")
    varInfo="Yale sparse format of stoichiometric matrix (row-wise)";
    CALL sWriteIntegerR1(varName,varInfo,SYI(ntc+1)-1,SY,OutputUnit,FormatType)
    CASE ("SJRI")
    varInfo="Integer vector of stoichiometric matrix of Jacobian matrix";
    CALL sWriteIntegerR1(varName,varInfo,ntc+1,SJRI,OutputUnit,FormatType)
    CASE ("SJRL")
    varInfo="Location vector of stoichiometric matrix of Jacobian matrix";
    CALL sWriteIntegerR1(varName,varInfo,SJRI(ntc+1)-1,SJRL,OutputUnit,FormatType)
    CASE ("SJYI")
    varInfo="Integer vector of stoichiometric matrix for the first derivative of reaction rate in Jacobian matrix";
    CALL sWriteIntegerR1(varName,varInfo,SJRI(ntc+1)-1+1,SJYI,OutputUnit,FormatType)
    CASE ("SJYRL")
```

Table E.6 Continued

```
varInfo="Row location vector of stoichiometric matrix for the first derivative of reaction rate in Jacobian matrix";
CALL sWriteIntegerR1(varName,varInfo,SJYI(SJRI(ntc+1)-1+1)-1,SJYRL,OutputUnit,FormatType)
 CASE ("SJYCL")
varInfo="Column location vector of stoichiometric matrix for the first derivative of reaction rate in Jacobian matrix";
CALL sWriteIntegerR1(varName,varInfo,SJYI(SJRI(ntc+1)-1+1)-1,SJYCL,OutputUnit,FormatType)
 CASE ("SJY")
varInfo="Yale's format of stoichiometric matrix for the first derivative of reaction rate in Jacobian matrix";
CALL sWriteIntegerR1(varName,varInfo,SJYI(SJRI(ntc+1)-1+1)-1,SJY,OutputUnit,FormatType)
 CASE ("AC")
varInfo="Normalized Arrehnius' coefficient, in the order of pre-exponential factor (1/s), "// &
     "order of temperature and activation energy";
CALL sWriteDoubleR2(varName,varInfo,ntr,3,AC,OutputUnit,FormatType)
 CASE ("RC")
varInfo="Rate constant (1/s)";
CALL sWriteDoubleR2(varName,varInfo,ntr,ngz,RC,OutputUnit,FormatType)
 CASE ("RI")
varInfo="Integer vector of reaction rate";
CALL sWriteIntegerR1(varName,varInfo,urStart-erStart+1,RI,OutputUnit,FormatType)
 CASE ("RL")
varInfo="Locator vector of reaction rate";
CALL sWriteIntegerR1(varName,varInfo,RI(near+1)-1,RL,OutputUnit,FormatType)
 CASE ("RP")
varInfo="Power vector of reaction rate";
CALL sWriteDoubleR1(varName,varInfo,RI(near+1)-1,RP,OutputUnit,FormatType)
 CASE ("R")
varInfo="Reaction Rate (1/s)";
CALL sWriteDoubleR2(varName,varInfo,ntr,ngz,R,OutputUnit,FormatType)
 CASE ("DRDC")
varInfo="Coefficient of Jacobian matrix, (dRi/dCj)";
CALL sWriteDoubleR2(varName,varInfo,ntr,ntc,DRDC,OutputUnit,FormatType)
 CASE ("JYI")
varInfo="Integer vector of Jacobian matrix";
CALL sWriteIntegerR1(varName,varInfo,ntc+1,JYI,OutputUnit,FormatType)
 CASE ("JYL")
varInfo="Locator vector of jacobian matrix";
CALL sWriteIntegerR1(varName,varInfo,JYI(ntc+1)-1,JYL,OutputUnit,FormatType)
 CASE ("JR")
varInfo="Nonlinear reaction part of Jacobian matrix";
CALL sWriteDoubleR2(varName,varInfo,ntc,ngz,JR,OutputUnit,FormatType)
 CASE ("JRY")
varInfo="Yale sparse format of the reaction part of Jacobian matrix (column-wise)";
CALL sWriteDoubleR2(varName,varInfo,JYI(ntc+1)-1,ngz,JRY,OutputUnit,FormatType)
 CASE ("JRC")
varInfo="Coefficient vector of the derivative of reaction rate in Jacobian matrix";
CALL sWriteDoubleR1(varName,varInfo,JYI(ntc+1),JRC,OutputUnit,FormatType)
 CASE ("JRI")
```

Table E.6 Continued

```
    varInfo="Integer vector of the derivative of reaction rate in Jacobian matrix";
    CALL sWriteIntegerR1(varName,varInfo,JYI(ntc+1),JRI,OutputUnit,FormatType)
     CASE ("JRL")
    varInfo="Locator vector of the derivative of reaction rate in Jacobian matrix";
    CALL sWriteIntegerR1(varName,varInfo,JRI(JYI(ntc+1))-1,JRL,OutputUnit,FormatType)
     CASE ("JRP")
    varInfo="Power vector of the derivative of reaction rate in Jacobian matrix";
    CALL sWriteDoubleR1(varName,varInfo,JRI(JYI(ntc+1))-1,JRP,OutputUnit,FormatType)
CASE ("mRuntime")   ! module mRuntime ------------------------------------------------------------------------------------------------------------------------|
     varInfo="Runtime variables and parameters"
        mSize=5;
        ALLOCATE(mList(mSize))
        mList=(/"iCPUTime","fCPUTime","dCPUTime","timeLeft","iRuntime"/);
        CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
     DEALLOCATE(mList)
     CASE ("iCPUTime")
    varInfo="Initial CPU time (s)";
    CALL sWriteDoubleR0(varName,varInfo,iCPUTime,OutputUnit,FormatType)
     CASE ("fCPUTime")
    varInfo="Final CPU time (s)";
    CALL sWriteDoubleR0(varName,varInfo,fCPUTime,OutputUnit,FormatType)
     CASE ("dCPUTime")
    varInfo="CPU time interval (s)";
    CALL sWriteDoubleR0(varName,varInfo,dCPUTime,OutputUnit,FormatType)
     CASE ("timeLeft")
    varInfo="Estimated time left for a run (s)";
    CALL sWriteDoubleR0(varName,varInfo,timeLeft,OutputUnit,FormatType)
     CASE ("iRuntime")
    varInfo="Counter for one percent of total runtime";
    CALL sWriteIntegerR0(varName,varInfo,iRuntime,OutputUnit,FormatType)
CASE("mSave") ! module mSave ---------------------------------------------------------------------------------------------------------------------------|
     varInfo="Variables and parameters to save data"
        mSize=36;
        ALLOCATE(mList(mSize))
    mList=(/"fileTime","unitTime","formatTime", &
            "fileRuntime","unitRuntime","formatRuntime", &
            "fileC","UnitC","formatC", &
            "fileR","UnitR","formatR", &
            "fileT","unitT","formatT", &
            "fileDeff","UnitDeff","formatDeff", &
            "fileTau","UnitTau","formatTau", &
            "fileRC","UnitRC","formatRC", &
            "fileCb","UnitCb","formatCb", &
            "fileCin","UnitCin","formatCin", &
            "fileTaub","UnitTaub","formatTaub", &
            "fileTauin","UnitTauin","formatTauin"/);
```

Table E.6 Continued

```
   CALL sWriteModule(varName,varInfo,mSize,mList,OutputUnit,FormatType)
  DEALLOCATE(mList)
    CASE ("fileTime")
      varInfo="File name to store time";
      CALL sWriteStringR0(varName,varInfo,fileTime,OutputUnit,FormatType)
    CASE ("unitTime")
      varInfo="Unit to save time";
      CALL sWriteIntegerR0(varName,varInfo,unitTime,OutputUnit,FormatType)
    CASE ("formatTime")
      varInfo="Data format of time";
      CALL sWriteStringR0(varName,varInfo,formatTime,OutputUnit,FormatType)
    CASE ("fileRuntime")
      varInfo="File name to store runtime";
      CALL sWriteStringR0(varName,varInfo,fileRuntime,OutputUnit,FormatType)
    CASE ("unitRuntime")
      varInfo="Unit to save runtime";
      CALL sWriteIntegerR0(varName,varInfo,unitRuntime,OutputUnit,FormatType)
    CASE ("formatRuntime")
      varInfo="Data format of runtime";
      CALL sWriteStringR0(varName,varInfo,formatRuntime,OutputUnit,FormatType)
    CASE ("FileC")
      varInfo="File name to store concentration";
      CALL sWriteStringR1(varName,varInfo,ntc,FileC,OutputUnit,FormatType)
    CASE ("UnitC")
      varInfo="Unit to save concentration";
      CALL sWriteIntegerR1(varName,varInfo,ntc,UnitC,OutputUnit,FormatType)
    CASE ("formatC")
      varInfo="Data format of concentration";
      CALL sWriteStringR0(varName,varInfo,formatC,OutputUnit,FormatType)
    CASE ("FileR")
      varInfo="File name to store reaction rate";
      CALL sWriteStringR1(varName,varInfo,ntr,FileR,OutputUnit,FormatType)
    CASE ("UnitR")
      varInfo="Unit to save reaction rate";
      CALL sWriteIntegerR1(varName,varInfo,ntr,UnitR,OutputUnit,FormatType)
    CASE ("formatR")
      varInfo="Data format of reaction rate";
      CALL sWriteStringR0(varName,varInfo,formatR,OutputUnit,FormatType)
    CASE ("fileT")
      IF (thermalType /= 1) THEN
        varInfo="File name to store temperature";
        CALL sWriteStringR0(varName,varInfo,fileT,OutputUnit,FormatType)
      END IF
    CASE ("unitT")
      IF (thermalType /= 1) THEN
        varInfo="Unit to save temperature";
```

Table E.6 Continued

```
    CALL sWriteIntegerR0(varName,varInfo,unitT,OutputUnit,FormatType)
   END IF
  CASE ("formatT")
   IF (thermalType /= 1) THEN
    varInfo="Data format of temperature";
    CALL sWriteStringR0(varName,varInfo,formatT,OutputUnit,FormatType)
   END IF
  CASE ("FileDeff")
   IF (thermalType /= 1) THEN
  varInfo="File name to store diffusion coefficient";
    CALL sWriteStringR1(varName,varInfo,ngc,FileDeff,OutputUnit,FormatType)
   END IF
  CASE ("UnitDeff")
   IF (thermalType /= 1) THEN
   varInfo="Unit to save diffusion coefficient";
   CALL sWriteIntegerR1(varName,varInfo,ngc,UnitDeff,OutputUnit,FormatType)
   END IF
  CASE ("formatDeff")
   IF (thermalType /= 1) THEN
   varInfo="Data format of diffusion coefficient";
    CALL sWriteStringR0(varName,varInfo,formatDeff,OutputUnit,FormatType)
   END IF
  CASE ("FileTau")
   IF (thermalType /= 1) THEN
   varInfo="File name to store characteristic time of diffusion";
    CALL sWriteStringR1(varName,varInfo,ngc,FileTau,OutputUnit,FormatType)
   END IF
  CASE ("UnitTau")
   IF (thermalType /= 1) THEN
   varInfo="Unit to save characteristic time of diffusion";
   CALL sWriteIntegerR1(varName,varInfo,ngc,UnitTau,OutputUnit,FormatType)
   END IF
  CASE ("formatTau")
   IF (thermalType /= 1) THEN
   varInfo="Data format of characteristic time of diffusion";
    CALL sWriteStringR0(varName,varInfo,formatTau,OutputUnit,FormatType)
   END IF
  CASE ("FileRC")
   IF (thermalType /= 1) THEN
   varInfo="File name to store rate constant";
    CALL sWriteStringR1(varName,varInfo,ntr,FileRC,OutputUnit,FormatType)
   END IF
  CASE ("UnitRC")
   IF (thermalType /= 1) THEN
   varInfo="Unit to save rate constant";
   CALL sWriteIntegerR1(varName,varInfo,ntr,UnitRC,OutputUnit,FormatType)
```

Table E.6 Continued

```
    END IF
  CASE ("formatRC")
    IF (thermalType /= 1) THEN
    varInfo="Data format of rate constant";
      CALL sWriteStringR0(varName,varInfo,formatRC,OutputUnit,FormatType)
    END IF
END SELECT
END SUBROUTINE sWrite
!-----------------------------------------------------------------------
! Subroutine sWriteDoubleR0
! Description: Write formatted double precision variable to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!  vName: variable name
!  vDescription: description of variable
!  vValue: value of variable
!  vUnit: unit to write variable
!  vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-----------------------------------------------------------------------
SUBROUTINE sWriteDoubleR0(vName,vDescription,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
DOUBLE PRECISION, INTENT(IN) :: vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
CHARACTER(LEN=256), EXTERNAL :: fInsInt
INTEGER :: infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
CHARACTER(LEN=16) :: doubleFormat
IF (vValue<0.0D0) THEN; doubleFormat="1P1E23.15E3"; ELSE; doubleFormat="1PE22.15E3"; END IF
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
    outputFormat="("//trim(doubleFormat)//")"
    WRITE(vUnit,outputFormat) vValue
  CASE (1)
    vInfo=trim(vInfo)//", "//trim(vName)//" =";
    infoLen=len_trim(vInfo);
    outputFormat=fInsInt("(A",infoLen,",'"//trim(doubleFormat)//"')");
    WRITE(vUnit,outputFormat) vInfo, vValue
  CASE (2)
    xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="double" Rank="0" Dimension="1"'//'>';
```

Table E.6 Continued

```
    xmlTag(2)="</"//trim(vName)//">";
    tagLen(1)=len_trim(xmlTag(1)); tagLen(2)=len_trim(xmlTag(2));
    outputFormat=fInsInt("(8X,A",tagLen(1),","//doubleFormat//"\)");
    WRITE(vUnit,outputFormat) xmlTag(1), vValue
    outputFormat=fInsInt("(A",tagLen(2),")");
    WRITE(vUnit,outputFormat) xmlTag(2)
END SELECT
END SUBROUTINE sWriteDoubleR0
!-----------------------------------------------------------------------
! Subroutine sWriteDoubleR1
! Description: Write formatted one dimensional double precision vector to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   vName: variable name
!   vDescription: description of variable
!   vSize: size of vector
!   vValue: value of variable
!   vUnit: unit to write variable
!   vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sWriteDoubleR1(vName,vDescription,vSize,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
INTEGER, INTENT(IN) :: vSize
DOUBLE PRECISION, DIMENSION(vSize), INTENT(IN) :: vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
CHARACTER(LEN=256), EXTERNAL :: fInsInt
INTEGER :: infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
CHARACTER(LEN=16) :: doubleFormat
INTEGER :: i
IF (minval(vValue)<0.0D0) THEN; doubleFormat="1P1E23.15E3"; ELSE; doubleFormat="1P1E22.15E3"; END IF
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
    DO i=1,vSize
      IF (vValue(i)<0.0D0) THEN; doubleFormat="1P1E23.15E3"; ELSE; doubleFormat="1P1E22.15E3"; END IF
      IF (i==1) THEN; outputFormat="("//doubleFormat//"\)";
      ELSEIF (i==vSize) THEN; outputFormat="(',',"//doubleFormat//")";
      ELSE; outputFormat="(',',"//doubleFormat//"\)"; END IF
      WRITE(vUnit,outputFormat) vValue(i)
```

Table E.6 Continued

```
    END DO
  CASE (1)
    vInfo=trim(vInfo)//", "//trim(vName)//" =";
      infoLen=len_trim(vInfo);
      outputFormat=fInsInt("(A",infoLen,")");
    WRITE(vUnit,outputFormat) vInfo
      CALL sWriteDoubleR1(vName,vDescription,vSize,vValue,vUnit,0)
  CASE (2)
      xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="double" Rank="1" Dimension="';
      xmlTag(1)=fInsInt(trim(xmlTag(1)),vSize,'">');
      xmlTag(2)="</"//trim(vName)//">";
      tagLen(1)=len_trim(xmlTag(1));
      tagLen(2)=len_trim(xmlTag(2));
      outputFormat=fInsInt("(8X,A",tagLen(1),"\)");
      WRITE(vUnit,outputFormat) xmlTag(1)
      DO i=1,vSize
        IF (vValue(i)<0.0D0) THEN; doubleFormat="1P1E23.15E3"; ELSE; doubleFormat="1P1E22.15E3"; END IF
        IF (i>1) THEN; outputFormat="(',',"//doubleFormat//"\)"; ELSE; outputFormat="("//doubleFormat//"\)"; END IF
        WRITE(vUnit,outputFormat) vValue(i)
      END DO
      outputFormat=fInsInt("(A",tagLen(2),")");
      WRITE(vUnit,outputFormat) xmlTag(2)
END SELECT
END SUBROUTINE sWriteDoubleR1
!-----------------------------------------------------------------------------------------------------------------------
! Subroutine sWriteDoubleR2
! Description: Write formatted two dimensional double precision array to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------------------------------------------------------
! Input:
!   vName: variable name
!   vDescription: description of variable
!   nRow: rows of array
!   nCol: columns of array
!   vValue: value of variable
!   vUnit: unit to write variable
!   vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-----------------------------------------------------------------------------------------------------------------------
SUBROUTINE sWriteDoubleR2(vName,vDescription,nRow,nCol,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
INTEGER, INTENT(IN) :: nRow, nCol
DOUBLE PRECISION, DIMENSION(nRow,nCol), INTENT(IN) :: vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
```

Table E.6 Continued

```
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2, fInsInt3
INTEGER :: infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
CHARACTER(LEN=16) :: doubleFormat
INTEGER :: i, j
IF (minval(vValue)<0.0D0) THEN; doubleFormat="1P1E23.15E3"; ELSE; doubleFormat="1P1E22.15E3"; END IF
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
    outputFormat=fInsInt2("(\",nRow,"(/",(nCol-1),"(''//trim(doubleFormat)//'', ', '),''//trim(doubleFormat)//''))");
    WRITE(vUnit,outputFormat) transpose(vValue)
  CASE (1)
    vInfo=trim(vInfo)//", ''//trim(vName)//'' =";
    infoLen=len_trim(vInfo);
    outputFormat=fInsInt3("(A",infoLen,", ",nRow,"(/",(nCol-1),"(''//trim(doubleFormat)//'', ', '),''//trim(doubleFormat)//''))");
    WRITE(vUnit,outputFormat) vInfo, transpose(vValue)
  CASE (2)
    xmlTag(1)='<''//trim(vName)//'' Description="''//trim(vInfo)//'" Datatype="double" Rank="2" Dimension="';
    xmlTag(1)=fInsInt2(trim(xmlTag(1)),nRow,",",nCol,"">');
    xmlTag(2)="</''//trim(vName)//">";
    tagLen(1)=len_trim(xmlTag(1)); tagLen(2)=len_trim(xmlTag(2));
    outputFormat=fInsInt("(8X,A",tagLen(1),",''//doubleFormat//''\)");
    WRITE(vUnit,outputFormat) xmlTag(1), vValue(1,1)
    outputFormat="(A1,''//doubleFormat//'\)";
    DO i=1,nRow
      DO j=1,nCol
        IF(i+j>2) THEN
          WRITE(vUnit,outputFormat) ",", vValue(i,j)
        END IF
      END DO
    END DO
    outputFormat=fInsInt("(A",tagLen(2),")");
    WRITE(vUnit,outputFormat) xmlTag(2)
END SELECT
END SUBROUTINE sWriteDoubleR2
!-----------------------------------------------------------------------
! Subroutine sWriteIntegerR0
! Description: Write formatted integer scalar to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!  vName: variable name
!  vDescription: description of variable
```

Table E.6 Continued

```
!!   vValue: value of variable
!    vUnit: unit to write variable
!    vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-------------------------------------------------------------------------------
SUBROUTINE sWriteIntegerR0(vName,vDescription,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
INTEGER, INTENT(IN) :: vValue, vUnit, vFormat
INTEGER, EXTERNAL :: fIntDig
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2, fInsInt3
INTEGER :: vLen, infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
vLen=fIntDig(abs(vValue));
IF (vValue<0) THEN; vLen=vLen+1; END IF
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
     outputFormat=fInsInt("I",vLen,")");
     WRITE(vUnit,outputFormat) vValue
  CASE (1)
     vInfo=trim(vInfo)//", "//trim(vName)//" =";
     infoLen=len_trim(vInfo);
     outputFormat=fInsInt2("A",infoLen,",/I",vLen,")");
     WRITE(vUnit,outputFormat) vInfo, vValue
  CASE (2)
     xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="integer" Rank="0" Dimension="1">';
     xmlTag(2)="</"//trim(vName)//">";
     tagLen(1)=len_trim(xmlTag(1));
     tagLen(2)=len_trim(xmlTag(2));
     outputFormat=fInsInt3("(8X,A",tagLen(1),",I",vLen,",A",tagLen(2),")");
     WRITE(vUnit,outputFormat) xmlTag(1), vValue, xmlTag(2)
END SELECT
END SUBROUTINE sWriteIntegerR0
!-------------------------------------------------------------------------------
! Subroutine sWriteIntegerR1
! Description: Write formatted one dimensional integer vector to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-------------------------------------------------------------------------------
! Input:
!  vName: variable name
!  vDescription: description of variable
!  vSize: size of vector
```

Table E.6 Continued

```
!  vValue: value of variable
!  vUnit: unit to write variable
!  vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-------------------------------------------------------------------------------
RECURSIVE SUBROUTINE sWriteIntegerR1(vName,vDescription,vSize,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
INTEGER, INTENT(IN) :: vSize
INTEGER, DIMENSION(vSize), INTENT(IN) :: vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
INTEGER, EXTERNAL :: fIntDig
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt3, fInsInt4
INTEGER :: infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
INTEGER :: nDigit
INTEGER :: i
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
      DO i=1,vSize
        nDigit=fIntDig(abs(vValue(i)));
        IF (vValue(i)<0) THEN; nDigit=nDigit+1; END IF
        IF (i==1) THEN
          outputFormat=fInsInt("(I",nDigit,"\)");
        ELSEIF (i==vSize) THEN
          outputFormat=fInsInt("(',',I",nDigit,")");
        ELSE
          outputFormat=fInsInt("(',',I",nDigit,"\)");
        END IF
        WRITE(vUnit,outputFormat) vValue(i)
      END DO
  CASE (1)
    vInfo=trim(vInfo)//", "//trim(vName)//" =";
    infoLen=len_trim(vInfo);
      outputFormat=fInsInt("(A",infoLen,")");
    WRITE(vUnit,outputFormat) vInfo
      CALL sWriteIntegerR1(vName,vDescription,vSize,vValue,vUnit,0)
  CASE (2)
      xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="integer" Rank="1" Dimension="';
      xmlTag(1)=fInsInt(trim(xmlTag(1)),vSize,'">');
      xmlTag(2)="</"//trim(vName)//">";
      tagLen(1)=len_trim(xmlTag(1));
      tagLen(2)=len_trim(xmlTag(2));
```

Table E.6 Continued

```
    nDigit=fIntDig(maxval(abs(vValue)));
    outputFormat=fInsInt("(8X,A",tagLen(1),"\)");
    WRITE(vUnit,outputFormat) xmlTag(1)
    DO i=1,vSize
      nDigit=fIntDig(abs(vValue(i)));
      IF (vValue(i)<0) THEN; nDigit=nDigit+1; END IF
      IF (i==1) THEN
        outputFormat=fInsInt("I",nDigit,"\)");
      ELSE
        outputFormat=fInsInt(",I",nDigit,"\)");
      END IF
      WRITE(vUnit,outputFormat) vValue(i)
    END DO
    outputFormat=fInsInt("(A",tagLen(2),")");
      WRITE(vUnit,outputFormat) xmlTag(2)
END SELECT
END SUBROUTINE sWriteIntegerR1
!-----------------------------------------------------------------------
! Subroutine sWriteIntegerR2
! Description: Write formatted two dimensional integer array to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   vName: variable name
!   vDescription: description of variable
!   nRow: rows of array
!   nCol: columns of array
!   vValue: value of variable
!   vUnit: unit to write variable
!   vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-----------------------------------------------------------------------
SUBROUTINE sWriteIntegerR2(vName,vDescription,nRow,nCol,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
INTEGER, INTENT(IN) :: nRow, nCol
INTEGER, DIMENSION(nRow,nCol), INTENT(IN) :: vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
INTEGER, EXTERNAL :: fIntDig
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2, fInsInt4, fInsInt5
INTEGER :: infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
INTEGER :: maxDigit
```

Table E.6 Continued

```
INTEGER :: i, j
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
      maxDigit=fIntDig(maxval(abs(vValue)));
      IF (minval(vValue)<0) THEN; maxDigit=maxDigit+1; END IF
      outputFormat=fInsInt4("(\",nRow,"(/(",(nCol-1),"(I",maxDigit,",',',"),I",maxDigit,")))");
      WRITE(vUnit,outputFormat) transpose(vValue)
  CASE (1)
      vInfo=trim(vInfo)//", "//trim(vName)//" =";
      infoLen=len_trim(vInfo);
      maxDigit=fIntDig(maxval(abs(vValue)));
      IF (minval(vValue)<0) THEN; maxDigit=maxDigit+1; END IF
      outputFormat=fInsInt5("(A",infoLen,",' ',nRow,"(/(",(nCol-1),"(I",maxDigit,",' ',' '),I",maxDigit,")))");
      WRITE(vUnit,outputFormat) vInfo, transpose(vValue)
  CASE (2)
    xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="integer" Rank="2" Dimension="';
    xmlTag(1)=fInsInt2(trim(xmlTag(1)),nRow,',',nCol,'">');
      xmlTag(2)="</"//trim(vName)//">";
      tagLen(1)=len_trim(xmlTag(1));
      tagLen(2)=len_trim(xmlTag(2));
      maxDigit=fIntDig(maxval(abs(vValue)));
      outputFormat=fInsInt("(8X,A",tagLen(1),"\)");
      WRITE(vUnit,outputFormat) xmlTag(1)
      DO i=1,nRow
        DO j=1,nCol
          IF(i+j>2) THEN
            WRITE(vUnit,"(A1\)") ","
          END IF
          IF (vValue(i,j)<0) THEN
            outputFormat=fInsInt("(I",maxDigit+1,"\)");
          ELSE
            outputFormat=fInsInt("(I",maxDigit,"\)");
          END IF
          WRITE(vUnit,outputFormat) vValue(i,j)
        END DO
      END DO
      outputFormat=fInsInt("(A",tagLen(2),")");
      WRITE(vUnit,outputFormat) xmlTag(2)
END SELECT
END SUBROUTINE sWriteIntegerR2
!-------------------------------------------------------------------------------
! Subroutine sWriteLogicalR0                                                   -
! Description: Write formatted scalar logical variable to a unit               -
! Created by Hsu-Wen Hsiao on 4/4/09.                                          -
! Copyright 2009 UCSD. All rights reserved.                                    -
```

Table E.6 Continued

```
!-----------------------------------------------------------------------------
! Input:
!   vName: variable name
!   vDescription: description of variable
!   vValue: value of variable
!   vUnit: unit to write variable
!   vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-----------------------------------------------------------------------------
SUBROUTINE sWriteLogicalR0(vName,vDescription,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
Logical, INTENT(IN) :: vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
INTEGER, EXTERNAL :: fIntDig
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
INTEGER :: infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
      outputFormat="(I1)";
      WRITE(vUnit,outputFormat) vValue
  CASE (1)
     vInfo=trim(vInfo)//", "//trim(vName)//" =";
     infoLen=len_trim(vInfo);
     outputFormat=fInsInt("(A",infoLen,",/I1)");
     WRITE(vUnit,outputFormat) vInfo, vValue
  CASE (2)
     xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="'//'logical" Rank="0" Dimension="1"//'">';
     xmlTag(2)="</"//trim(vName)//">";
     tagLen(1)=len_trim(xmlTag(1));
     tagLen(2)=len_trim(xmlTag(2));
     outputFormat=fInsInt2("(8X,A",tagLen(1),",I1,A",tagLen(2),")");
     WRITE(vUnit,outputFormat) xmlTag(1), vValue, xmlTag(2)
END SELECT
END SUBROUTINE sWriteLogicalR0
!-----------------------------------------------------------------------------
! Subroutine sWriteStringR0
! Description: Write a formatted string to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------------
! Input:
```

Table E.6 Continued

```
!  vName: variable name
!  vDescription: description of variable
!  vValue: value of variable
!  vUnit: unit to write variable
!  vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-----------------------------------------------------------------------
SUBROUTINE sWriteStringR0(vName,vDescription,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription, vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
INTEGER, EXTERNAL :: fIntDig
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2, fInsInt3
INTEGER :: vLen, infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
vLen=len_trim(vValue);
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
     outputFormat=fInsInt("A",vLen,")");
     WRITE(vUnit,outputFormat) vValue
  CASE (1)
     vInfo=trim(vInfo)//", "//trim(vName)//" =";
     infoLen=len_trim(vInfo);
     outputFormat=fInsInt2("(A",infoLen,",/A",vLen,")");
     WRITE(vUnit,outputFormat) vInfo, vValue
  CASE (2)
     xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="string" Rank="0" Dimension="';
     xmlTag(1)=fInsInt(trim(xmlTag(1)),vLen,'">')
     xmlTag(2)="</"//trim(vName)//">";
     tagLen(1)=len_trim(xmlTag(1));
     tagLen(2)=len_trim(xmlTag(2));
     outputFormat=fInsInt3("(8X,A",tagLen(1),",A",vLen,",A",tagLen(2),")");
     WRITE(vUnit,outputFormat) xmlTag(1), vValue, xmlTag(2)
END SELECT
END SUBROUTINE sWriteStringR0
!-----------------------------------------------------------------------
! Subroutine sWriteStringR1
! Description: Write formatted one dimensional string vector to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!  vName: variable name
```

Table E.6 Continued

```
!  vDescription: description of variable
!  vSize: dimension of vector
!  vValue: value of variable
!  vUnit: unit to write variable
!  vFormat: format of variable, 0: value only, 1: value with description, 2: xml
! Output: none
!-----------------------------------------------------------------------
SUBROUTINE sWriteStringR1(vName,vDescription,vSize,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
INTEGER, INTENT(IN) :: vSize
CHARACTER(LEN=*), DIMENSION(vSize), INTENT(IN) :: vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
INTEGER :: vLen, infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
INTEGER :: i
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
      DO i=1,vSize
        vLen=len_trim(vValue(i));
        IF (i==1) THEN
          outputFormat=fInsInt("A",vLen,"\)");
          WRITE(vUnit,outputFormat) vValue(i)
        ELSEIF (i==vSize) THEN
          WRITE(vUnit,"(A1\)") ","
          outputFormat=fInsInt("A",vLen,")");
          WRITE(vUnit,outputFormat) vValue(i)
        ELSE
          WRITE(vUnit,"(A1\)") ","
          outputFormat=fInsInt("A",vLen,"\)");
          WRITE(vUnit,outputFormat) vValue(i)
        END IF
      END DO
  CASE (1)
      vInfo=trim(vInfo)//", "//trim(vName)//" =";
      infoLen=len_trim(vInfo);
      outputFormat=fInsInt("A",infoLen,")");
      WRITE(vUnit,outputFormat) vInfo
      DO i=1,vSize
        vLen=len_trim(vValue(i));
        IF (i==1) THEN
          outputFormat=fInsInt("A",vLen,"\)");
```

Table E.6 Continued

```
    WRITE(vUnit,outputFormat) vValue(i)
  ELSEIF (i==vSize) THEN
    WRITE(vUnit,"(A1\)") "","
    outputFormat=fInsInt("(A",vLen,",A1)");
    WRITE(vUnit,outputFormat) vValue(i)
  ELSE
    WRITE(vUnit,"(A1\)") ","
    outputFormat=fInsInt("(A",vLen,"\)");
    WRITE(vUnit,outputFormat) vValue(i)
  END IF
 END DO
CASE (2)
 xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="string" Rank="1" Dimension="';
 DO i=1,vSize-1
  xmlTag(1)=fInsInt(xmlTag(1),len_trim(vValue(i)),',')
 END DO
 xmlTag(1)=fInsInt(xmlTag(1),len_trim(vValue(vSize)),'">');
 xmlTag(2)="</"//trim(vName)//">";
 tagLen(1)=len_trim(xmlTag(1)); tagLen(2)=len_trim(xmlTag(2));
 outputFormat=fInsInt("(8X,A",tagLen(1),"\)");
 WRITE(vUnit,outputFormat) xmlTag(1)
 DO i=1,vSize
  vLen=len_trim(vValue(i));
  IF (i>1) THEN
   WRITE(vUnit,"(A1\)") ","
  END IF
  outputFormat=fInsInt("(A",vLen,"\)");
  WRITE(vUnit,outputFormat) vValue(i)
 END DO
 outputFormat=fInsInt("(A",tagLen(2),")");
 WRITE(vUnit,outputFormat) xmlTag(2)
END SELECT
END SUBROUTINE sWriteStringR1
!-----------------------------------------------------------------------
! Subroutine sWriteModule
! Description: Write formatted module and its variables to a unit
! Created by Hsu-Wen Hsiao on 4/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!  vName: name of the module
!  vDescription: description of module
!  vSize: number of variable of this module
!  vValue: list of variable of this module
!  vUnit: unit to write module
!  vFormat: format of variable, 0: module only, 1: module with description, 2: xml
```

Table E.6 Continued

```
! Output: none
!-------------------------------------------------------------------------------
SUBROUTINE sWriteModule(vName,vDescription,vSize,vValue,vUnit,vFormat)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(IN) :: vName, vDescription
INTEGER, INTENT(IN) :: vSize
CHARACTER(LEN=*), DIMENSION(vSize), INTENT(IN) :: vValue
INTEGER, INTENT(IN) :: vUnit, vFormat
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
INTEGER :: vLen, infoLen
CHARACTER(LEN=256) :: outputFormat, vInfo
CHARACTER(LEN=256), DIMENSION(2) :: xmlTag
INTEGER, DIMENSION(2) :: tagLen
INTEGER :: i
vLen=len_trim(vName);
vInfo=vDescription;
SELECT CASE (vFormat)
  CASE (0)
     outputFormat=fInsInt("(A",vLen,")");
     WRITE(vUnit,outputFormat) vName
     DO i=1,vSize; CALL sWrite(trim(vValue(i)),vUnit,vFormat); END DO
  CASE (1)
     vInfo="MODULE "//trim(vName)//" : "//trim(vInfo);
     infoLen=len_trim(vInfo);
     outputFormat=fInsInt2("(A",infoLen,",/A",vLen,")");
     WRITE(vUnit,outputFormat) vInfo, vName
     DO i=1,vSize; CALL sWrite(trim(vValue(i)),vUnit,vFormat); END DO
  CASE (2)
     xmlTag(1)='<'//trim(vName)//' Description="'//trim(vInfo)//'" Datatype="'//'module'//'">';
     xmlTag(2)="</"//trim(vName)//">";
     tagLen(1)=len_trim(xmlTag(1));
     tagLen(2)=len_trim(xmlTag(2));
     outputFormat=fInsInt("(4X,A",tagLen(1),")")
     WRITE(vUnit,outputFormat) xmlTag(1)
     DO i=1,vSize; CALL sWrite(trim(vValue(i)),vUnit,vFormat); END DO
     outputFormat=fInsInt("(4X,A",tagLen(2),")")
     WRITE(vUnit,outputFormat) xmlTag(2)
END SELECT
END SUBROUTINE sWriteModule
!-------------------------------------------------------------------------------
! fIntDig.f90
! Description: Count the digits of integer x
! Created by Hsu-Wen Hsiao on 2/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-------------------------------------------------------------------------------
! Input:
```

Table E.6 Continued

```
!   x: integer (x>=0)
! Output:
!   fIntDig: Digits of integer x
!-----------------------------------------------------------------------
! Editor's Notes: none
!-----------------------------------------------------------------------
INTEGER FUNCTION fIntDig(x)
IMPLICIT NONE
INTEGER, INTENT(IN) :: x
INTEGER :: i, j
IF (x<0) THEN; j=1; ELSE; j=0; END IF
i=abs(x);
IF (i>0) THEN
  fIntDig=0;
  DO WHILE (i>0)
      fIntDig=fIntDig+1;
      i=(i-mod(i,10))/10;
  END DO
ELSEIF(i==0) THEN
  fIntDig=1;
END IF
fIntDig=fIntDig+j; ! one more digit for negative x
END FUNCTION fIntDig
!-----------------------------------------------------------------------
! Function fInsInt
! Description: Insert an integer between two strings.
! Created by Hsu-Wen Hsiao on 2/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   ls: string on the left
!   mi: integer in the middle
!   rs: string on the right
! Output:
!   fInsInt: New string after insertion operation.
!-----------------------------------------------------------------------
CHARACTER(LEN=256) FUNCTION fInsInt(ls,mi,rs)
IMPLICIT NONE
INTEGER, INTENT(IN) :: mi
CHARACTER(LEN=*), INTENT(IN) :: ls, rs
INTEGER :: i,j
fInsInt=trim(rs);
i=mi;
j=0;
DO WHILE (i>0)
  j=mod(i,10);
```

Table E.6 Continued

```
  fInsInt=char(j+48)//trim(fInsInt);
  i=(i-j)/10;
END DO
fInsInt=trim(ls)//trim(fInsInt);
END FUNCTION fInsInt
!-----------------------------------------------------------------------
! Function fInsInt2
! Description: Insert two integers between three strings.
! Created by Hsu-Wen Hsiao on 2/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input: (from left to right)
!   s1: string 1
!   i12: integer between string 1 and string 2
!   s2: string 2
!   i23: integer between string 2 and string 3
!   s3: string 3
! Output:
!   fInsInt: New string after insertion operation.
!-----------------------------------------------------------------------
CHARACTER(LEN=256) FUNCTION fInsInt2(s1,i12,s2,i23,s3)
IMPLICIT NONE
INTEGER, INTENT(IN) :: i12, i23
CHARACTER(LEN=*), INTENT(IN) :: s1, s2, s3
CHARACTER(LEN=256), EXTERNAL :: fInsInt
fInsInt2=fInsInt(s2,i23,s3);
fInsInt2=fInsInt(s1,i12,fInsInt2);
END FUNCTION fInsInt2
!-----------------------------------------------------------------------
! Function fInsInt3
! Description: Insert three integers between four strings.
! Created by Hsu-Wen Hsiao on 2/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input: (from left to right)
!   s1: string 1
!   i12: integer between string 1 and string 2
!   s2: string 2
!   i23: integer between string 2 and string 3
!   s3: string 3
!   i34: integer between string 3 and string 4
!   s4: string 4
! Output:
!   fInsInt: New string after insertion operation.
!-----------------------------------------------------------------------
CHARACTER(LEN=256) FUNCTION fInsInt3(s1,i12,s2,i23,s3,i34,s4)
```

Table E.6 Continued

```
IMPLICIT NONE
INTEGER, INTENT(IN) :: i12, i23, i34
CHARACTER(LEN=*), INTENT(IN) :: s1, s2, s3, s4
CHARACTER(LEN=256), EXTERNAL :: fInsInt
fInsInt3=fInsInt(s3,i34,s4);
fInsInt3=fInsInt(s2,i23,fInsInt3);
fInsInt3=fInsInt(s1,i12,fInsInt3);
END FUNCTION fInsInt3
!-----------------------------------------------------------------------
! Function fInsInt4
! Description: Insert four integers between five strings.
! Created by Hsu-Wen Hsiao on 2/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input: (from left to right)
!   s1: string 1
!   i12: integer between string 1 and string 2
!   s2: string 2
!   i23: integer between string 2 and string 3
!   s3: string 3
!   i34: integer between string 3 and string 4
!   s4: string 4
!   i45: integer between string 4 and string 5
!   s5: string 5
! Output:
!   fInsInt: New string after insertion operation.
!-----------------------------------------------------------------------
CHARACTER(LEN=256) FUNCTION fInsInt4(s1,i12,s2,i23,s3,i34,s4,i45,s5)
IMPLICIT NONE
INTEGER, INTENT(IN) :: i12, i23, i34, i45
CHARACTER(LEN=*), INTENT(IN) :: s1, s2, s3, s4, s5
CHARACTER(LEN=256), EXTERNAL :: fInsInt
fInsInt4=fInsInt(s4,i45,s5);
fInsInt4=fInsInt(s3,i34,fInsInt4);
fInsInt4=fInsInt(s2,i23,fInsInt4);
fInsInt4=fInsInt(s1,i12,fInsInt4);
END FUNCTION fInsInt4
!-----------------------------------------------------------------------
! Function fInsInt5
! Description: Insert five integers between four strings.
! Created by Hsu-Wen Hsiao on 2/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input: (from left to right)
!   s1: string 1
!   i12: integer between string 1 and string 2
```

Table E.6 Continued

```
!   s2: string 2
!   i23: integer between string 2 and string 3
!   s3: string 3
!   i34: integer between string 3 and string 4
!   s4: string 4
!   i45: integer between string 4 and string 5
!   s5: string 5
!   i56: integer between string 5 and string 6
!   s6: string 6
! Output:
!   fInsInt: New string after insertion operation.
!-------------------------------------------------------------------------------
CHARACTER(LEN=256) FUNCTION fInsInt5(s1,i12,s2,i23,s3,i34,s4,i45,s5,i56,s6)
IMPLICIT NONE
INTEGER, INTENT(IN) :: i12, i23, i34, i45, i56
CHARACTER(LEN=*), INTENT(IN) :: s1, s2, s3, s4, s5, s6
CHARACTER(LEN=256), EXTERNAL :: fInsInt
fInsInt5=fInsInt(s5,i56,s6);
fInsInt5=fInsInt(s4,i45,fInsInt5);
fInsInt5=fInsInt(s3,i34,fInsInt5);
fInsInt5=fInsInt(s2,i23,fInsInt5);
fInsInt5=fInsInt(s1,i12,fInsInt5);
END FUNCTION fInsInt5
```

Table E.7 sRuntime.f90

```
!-----------------------------------------------------------------------
! Subroutine sRuntime
! Description: Compute runtime and write to a unit
! Created by Hsu-Wen Hsiao on 3/15/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   OutputUnit: unit to write runtime information
!-----------------------------------------------------------------------
SUBROUTINE sRuntime(outputUnit)
USE mRuntime
USE mDynamic
IMPLICIT NONE
INTEGER, INTENT(IN) :: outputUnit
CHARACTER(LEN=256), EXTERNAL :: fInsInt
CHARACTER(LEN=256) :: outputFormat
CHARACTER(LEN=256) :: outputInfo
DOUBLE PRECISION :: percentFinished
INTEGER :: fYr, fMon, fDay, fHr, fMin, fSec, f100th
percentFinished=1.0D2*((tRun-tiRun)/(tfRun-tiRun)); ! calculate finished percentage
SELECT CASE (outputUnit)
  CASE (6) ! write runtime information to screen ----------------------------------------------------------
    IF (iRuntime == 0) THEN ! initial runtime ---------------------------------------------------------
        CALL getdat(fYr,fMon,fDay)
        CALL gettim(fHr,fMin,fSec,f100th)
        outputInfo="Session started at";
        outputFormat=fInsInt("(A",len_trim(outputInfo),"\)");
        WRITE(outputUnit,outputFormat) outputInfo
        outputFormat="(1X,I4,'-\)";
      WRITE(outputUnit,outputFormat) fYr
        IF (fMon<10) THEN; outputFormat="('0',I1,'-\)"; ELSE; outputFormat="(I2,'-\)"; END IF
        WRITE(outputUnit,outputFormat) fMon
        IF (fDay<10) THEN; outputFormat="('0',I1,\)"; ELSE; outputFormat="(I2\)"; END IF
        WRITE(outputUnit,outputFormat) fDay
        IF (fHr<10) THEN; outputFormat="(1X,'0',I1,':\)"; ELSE; outputFormat="(1X,I2,':\)"; END IF
        WRITE(outputUnit,outputFormat) fHr
        IF (fMin<10) THEN; outputFormat="('0',I1,':\)"; ELSE; outputFormat="(I2,':\)"; END IF
        WRITE(outputUnit,outputFormat) fMin
        IF (fSec<10) THEN; outputFormat="('0',I1)"; ELSE; outputFormat="(I2)"; END IF
        WRITE(outputUnit,outputFormat) fSec
        WRITE(OutputUnit,"(f5.1,A11,1X)") percentFinished, "% Finished;"
    ELSEIF ((iRuntime == 100 .OR. tRun == tfRun) .AND. dtLevel == 1) THEN ! final runtime ---------------------------------------------------
        CALL cpu_time(fCPUTime)
        dCPUTime=fCPUTime-iCPUTime;
        WRITE(OutputUnit,"(f5.1,A11\)") percentFinished, "% Finished;"
        IF (dCPUTime>1) THEN
```

Table E.7 Continued

```
          outputFormat=fInsInt("(1X,A22,1X,f",ceiling(log10(dCPUTime))+4,".3,1X,A7)");
        ELSE
          outputFormat="(1X,A22,1X,f5.3,1X,A7)";
        END IF
        WRITE(OutputUnit,outputFormat) "Total computing time =", dCPUTime, "seconds"
        CALL getdat(fYr,fMon,fDay)
        CALL gettim(fHr,fMin,fSec,f100th)
        outputInfo="Session ended at";
        outputFormat=fInsInt("(A",len_trim(outputInfo),"\)");
        WRITE(outputUnit,outputFormat) outputInfo
        outputFormat="(1X,I4,'-'\)";
      WRITE(outputUnit,outputFormat) fYr
        IF (fMon<10) THEN; outputFormat="('0',I1,'-'\)"; ELSE; outputFormat="(I2,'-'\)"; END IF
        WRITE(outputUnit,outputFormat) fMon
        IF (fDay<10) THEN; outputFormat="('0',I1,\)"; ELSE; outputFormat="(I2\)"; END IF
        WRITE(outputUnit,outputFormat) fDay
        IF (fHr<10) THEN; outputFormat="(1X,'0',I1,':'\)"; ELSE; outputFormat="(1X,I2,':'\)"; END IF
        WRITE(outputUnit,outputFormat) fHr
        IF (fMin<10) THEN; outputFormat="('0',I1,':'\)"; ELSE; outputFormat="(I2,':'\)"; END IF
        WRITE(outputUnit,outputFormat) fMin
        IF (fSec<10) THEN; outputFormat="('0',I1)"; ELSE; outputFormat="(I2)"; END IF
        WRITE(outputUnit,outputFormat) fSec
    ELSE ! every 1% finished -----------------------------------------------------------------------------------------|
        CALL cpu_time(fCPUTime)
        dCPUTime=fCPUTime-iCPUTime;
        timeLeft=dCPUTime*(1.0D2/percentFinished-1.0D0)
        WRITE(OutputUnit,"(f5.1,A11\)") percentFinished, "% Finished;"
        IF (timeLeft>1.0D0) THEN
          outputFormat=fInsInt("(1X,f",ceiling(log10(timeLeft))+4,".3,1X,A13)");
        ELSE
          outputFormat="(1X,f5.3,1X,A13)";
        END IF
        WRITE(OutputUnit,outputFormat) timeLeft, "seconds to go"
    END IF
    IF (iRuntime < 100) THEN; iRuntime=iRuntime+1; END IF
    tRuntime=tfRun-1.0D-2*(100-iRuntime)*(tfRun-tiRun);
  CASE DEFAULT ! default output of simulation time and runtime -------------------------------------------------------|
    CALL cpu_time(fCPUTime)
    outputFormat="(1P1E22.15E3,',',1P1E22.15E3)";
    WRITE(outputUnit,outputFormat) tRun, fCPUTime
END SELECT
END SUBROUTINE sRuntime
```

Table E.8 sFinalize.f90

```
!-----------------------------------------------------------------------------|
! Subroutine sFinalize                                                        |
! Description: Finalize arrays for TransCat                                  |
! Created by Hsu-Wen Hsiao on 2/7/09.                                        |
! Copyright 2009 UCSD. All rights reserved.                                   |
!-----------------------------------------------------------------------------|
SUBROUTINE sFinalize
USE mBoundary
USE mDynamic
USE mElement
USE mInformation
USE mParameter
USE mReaction
USE mRuntime
USE mSave
USE mUserDefine
IMPLICIT NONE
INTEGER :: i
CALL sSave("FinalConcentration.txt") ! Save the final concentration and temperature so they can be used as the initial ----------|
CALL sSave("FinalTemperature.txt")   ! condition for other runs. Just set inputIC, inputIT, inputFIC and inputFIT in the input.txt.
CLOSE(UNIT=unitRuntime) ! close files -------------------------------------------------------------------------------------------|
CLOSE(UNIT=unitTime)
DO i=1,ntc; CLOSE(UNIT=UnitC(i)); END DO
DO i=1,ntr; CLOSE(UNIT=UnitR(i)); END DO
IF (thermalType /= 1) THEN
  CLOSE(UNIT=unitT)
  DO i=1,ngc; CLOSE(UNIT=UnitDeff(i)); END DO
  DO i=1,ngc; CLOSE(UNIT=UnitTau(i)); END DO
  DO i=1,ntr; CLOSE(UNIT=UnitRC(i)); END DO
END IF
IF (ALLOCATED(BCW   )) THEN; DEALLOCATE(BCW   ); END IF ! deallocate mBoundary module ------------------------------------------|
IF (ALLOCATED(BCF   )) THEN; DEALLOCATE(BCF   ); END IF
IF (ALLOCATED(BCL   )) THEN; DEALLOCATE(BCL   ); END IF
IF (ALLOCATED(BCCmax)) THEN; DEALLOCATE(BCCmax); END IF
IF (ALLOCATED(BCCmin)) THEN; DEALLOCATE(BCCmin); END IF
IF (ALLOCATED(CC    )) THEN; DEALLOCATE(CC    ); END IF ! deallocate mDynamic module ------------------------------------------|
IF (ALLOCATED(T     )) THEN; DEALLOCATE(T     ); END IF
IF (ALLOCATED(LC    )) THEN; DEALLOCATE(LC    ); END IF
IF (ALLOCATED(NC    )) THEN; DEALLOCATE(NC    ); END IF
IF (ALLOCATED(Tau   )) THEN; DEALLOCATE(Tau   ); END IF
IF (ALLOCATED(TauB  )) THEN; DEALLOCATE(TauB  ); END IF
IF (ALLOCATED(M     )) THEN; DEALLOCATE(M     ); END IF ! deallocate mElement module ------------------------------------------|
IF (ALLOCATED(MIMD  )) THEN; DEALLOCATE(MIMD  ); END IF
IF (ALLOCATED(MI    )) THEN; DEALLOCATE(MI    ); END IF
IF (ALLOCATED(MD    )) THEN; DEALLOCATE(MD    ); END IF
IF (ALLOCATED(MDi   )) THEN; DEALLOCATE(MDi   ); END IF
```

Table E.8 Continued

```
IF (ALLOCATED(MDiM   )) THEN; DEALLOCATE(MDiM   ); END IF
IF (ALLOCATED(MDiMI  )) THEN; DEALLOCATE(MDiMI  ); END IF
IF (ALLOCATED(MDiMY  )) THEN; DEALLOCATE(MDiMY  ); END IF
IF (ALLOCATED(MDiMYI )) THEN; DEALLOCATE(MDiMYI ); END IF
IF (ALLOCATED(MDiMYL )) THEN; DEALLOCATE(MDiMYL ); END IF
IF (ALLOCATED(MDiMIY )) THEN; DEALLOCATE(MDiMIY ); END IF
IF (ALLOCATED(MDiMIYI))) THEN; DEALLOCATE(MDiMIYI); END IF
IF (ALLOCATED(MDiMIYL))) THEN; DEALLOCATE(MDiMIYL); END IF
IF (ALLOCATED(Msc    )) THEN; DEALLOCATE(Msc    ); END IF
IF (ALLOCATED(MscI   )) THEN; DEALLOCATE(MscI   ); END IF
IF (ALLOCATED(MscL   )) THEN; DEALLOCATE(MscL   ); END IF
IF (ALLOCATED(SCtot  )) THEN; DEALLOCATE(SCtot  ); END IF
IF (iRun==nRun) .AND. (ALLOCATED(Runlist))) THEN; DEALLOCATE(RunList); END IF ! deallocate mInformation module ----------|
IF (ALLOCATED(Component))) THEN; DEALLOCATE(Component); END IF ! deallocate mParameter module ----------|
IF (ALLOCATED(MWGas  )) THEN; DEALLOCATE(MWGas  ); END IF
IF (ALLOCATED(catWF  )) THEN; DEALLOCATE(catWF  ); END IF
IF (ALLOCATED(catMW  )) THEN; DEALLOCATE(catMW  ); END IF
IF (ALLOCATED(catS   )) THEN; DEALLOCATE(catS   ); END IF
IF (ALLOCATED(catD   )) THEN; DEALLOCATE(catD   ); END IF
IF (ALLOCATED(catSF  )) THEN; DEALLOCATE(catSF  ); END IF
IF (ALLOCATED(Element)) THEN; DEALLOCATE(Element); END IF
IF (ALLOCATED(MechE  )) THEN; DEALLOCATE(MechE  ); END IF
IF (ALLOCATED(ERC    )) THEN; DEALLOCATE(ERC    ); END IF
IF (ALLOCATED(MechA  )) THEN; DEALLOCATE(MechA  ); END IF
IF (ALLOCATED(ARC    )) THEN; DEALLOCATE(ARC    ); END IF
IF (ALLOCATED(ARO    )) THEN; DEALLOCATE(ARO    ); END IF
IF (ALLOCATED(MechU  )) THEN; DEALLOCATE(MechU  ); END IF
IF (ALLOCATED(URC    )) THEN; DEALLOCATE(URC    ); END IF
IF (ALLOCATED(IniGP  )) THEN; DEALLOCATE(IniGP  ); END IF
IF (ALLOCATED(IniSC  )) THEN; DEALLOCATE(IniSC  ); END IF
IF (ALLOCATED(BCPmax )) THEN; DEALLOCATE(BCPmax ); END IF
IF (ALLOCATED(BCPmin )) THEN; DEALLOCATE(BCPmin ); END IF
IF (ALLOCATED(Z      )) THEN; DEALLOCATE(Z      ); END IF
IF (ALLOCATED(catSC  )) THEN; DEALLOCATE(catSC  ); END IF
IF (ALLOCATED(catCS  )) THEN; DEALLOCATE(catCS  ); END IF
IF (ALLOCATED(Deff   )) THEN; DEALLOCATE(Deff   ); END IF
IF (ALLOCATED(Mech   )) THEN; DEALLOCATE(Mech   ); END IF
IF (ALLOCATED(EAURC  )) THEN; DEALLOCATE(EAURC  ); END IF
IF (ALLOCATED(RIM    )) THEN; DEALLOCATE(RIM    ); END IF
IF (ALLOCATED(RIV    )) THEN; DEALLOCATE(RIV    ); END IF
IF (ALLOCATED(AC     )) THEN; DEALLOCATE(AC     ); END IF ! deallocate mReaction module ----------|
IF (ALLOCATED(R      )) THEN; DEALLOCATE(R      ); END IF
IF (ALLOCATED(RC     )) THEN; DEALLOCATE(RC     ); END IF
IF (ALLOCATED(RI     )) THEN; DEALLOCATE(RI     ); END IF
IF (ALLOCATED(RL     )) THEN; DEALLOCATE(RL     ); END IF
IF (ALLOCATED(RP     )) THEN; DEALLOCATE(RP     ); END IF
```

Table E.8 Continued

```
IF (ALLOCATED(DRDC))) THEN; DEALLOCATE(DRDC); END IF
IF (ALLOCATED(JYI ))) THEN; DEALLOCATE(JYI ); END IF
IF (ALLOCATED(JYL ))) THEN; DEALLOCATE(JYL ); END IF
IF (ALLOCATED(JR  ))) THEN; DEALLOCATE(JR  ); END IF
IF (ALLOCATED(JRY ))) THEN; DEALLOCATE(JRY ); END IF
IF (ALLOCATED(JUR ))) THEN; DEALLOCATE(JUR ); END IF
IF (ALLOCATED(JRC ))) THEN; DEALLOCATE(JRC ); END IF
IF (ALLOCATED(JRI ))) THEN; DEALLOCATE(JRI ); END IF
IF (ALLOCATED(JRL ))) THEN; DEALLOCATE(JRL ); END IF
IF (ALLOCATED(JRP ))) THEN; DEALLOCATE(JRP ); END IF
IF (ALLOCATED(S   ))) THEN; DEALLOCATE(S   ); END IF
IF (ALLOCATED(SY  ))) THEN; DEALLOCATE(SY  ); END IF
IF (ALLOCATED(SYI ))) THEN; DEALLOCATE(SYI ); END IF
IF (ALLOCATED(SYL ))) THEN; DEALLOCATE(SYL ); END IF
IF (ALLOCATED(SJRI))) THEN; DEALLOCATE(SJRI); END IF
IF (ALLOCATED(SJRL))) THEN; DEALLOCATE(SJRL); END IF
IF (ALLOCATED(SJYI ))) THEN; DEALLOCATE(SJYI ); END IF
IF (ALLOCATED(SJYRL))) THEN; DEALLOCATE(SJYRL); END IF
IF (ALLOCATED(SJYCL))) THEN; DEALLOCATE(SJYCL); END IF
IF (ALLOCATED(SJY  ))) THEN; DEALLOCATE(SJY  ); END IF
IF (ALLOCATED(UDR  ))) THEN; DEALLOCATE(UDR  ); END IF ! deallocate mUserDefine module ----------|
IF (ALLOCATED(UDRD ))) THEN; DEALLOCATE(UDRD ); END IF
IF (ALLOCATED(nzUDRD))) THEN; DEALLOCATE(nzUDRD); END IF
IF (ALLOCATED(UnitC  ))) THEN; DEALLOCATE(UnitC  ); END IF ! deallocate mSave module ----------|
IF (ALLOCATED(FileC  ))) THEN; DEALLOCATE(FileC  ); END IF
IF (ALLOCATED(UnitR  ))) THEN; DEALLOCATE(UnitR  ); END IF
IF (ALLOCATED(FileR  ))) THEN; DEALLOCATE(FileR  ); END IF
IF (ALLOCATED(UnitDeff))) THEN; DEALLOCATE(UnitDeff); END IF
IF (ALLOCATED(UnitTau ))) THEN; DEALLOCATE(UnitTau ); END IF
IF (ALLOCATED(UnitRC  ))) THEN; DEALLOCATE(UnitRC  ); END IF
IF (ALLOCATED(FileDeff))) THEN; DEALLOCATE(FileDeff); END IF
IF (ALLOCATED(FileTau ))) THEN; DEALLOCATE(FileTau ); END IF
IF (ALLOCATED(FileRC  ))) THEN; DEALLOCATE(FileRC  ); END IF
END SUBROUTINE sFinalize
```

Table E.9 sEE.f90

```
!-----------------------------------------------------------------------------------------------------------------------------
! Subroutine sEE
! Description: Explicit Euler method.
! Created by Hsu-Wen Hsiao on 4/6/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------------------------------------------------------------
! Input:
!   tf: final time
!   nt: number of time step
!-----------------------------------------------------------------------------------------------------------------------------
RECURSIVE SUBROUTINE sEE(tf,nt)
USE mDynamic
USE mReaction
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: tf                  ! final time
INTEGER, INTENT(IN) :: nt                           ! number of time step
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: C0 ! pre-stored concentration of previous time step
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: T0   ! pre-stored temperature of previous time step
DOUBLE PRECISION :: tRun0                           ! pre-stored time of previous time step
DOUBLE PRECISION :: dt                              ! timestep size
DOUBLE PRECISION :: maxDC, minC, maxDT, minT        ! parameter for adaptive time control
INTEGER :: it , i, j, flagLastStep                  ! dummy indices
IF (dtLevel > maxdtLevel) THEN ! initialize array for adaptive time control ----------------------------------------------------
  WRITE(6,*) "Warning ( dtLevel > 200 ): Unstable numerical method!! Please use smaller tfRun or change numerical method."
END IF
maxDC=0.0D0;     ! maximum concentration change
maxDT=0.0D0;     ! maximum temperature change
minC=MINVAL(C);  ! minimum concentration
minT=MINVAL(T);  ! minimum temperature
flagLastStep=nt; ! flag to syncronize the last step
tooStiff=.FALSE.; ! signal of stiffness for the sub-loop of adaptive time control
ALLOCATE(C0(ngz,ntc),T0(ngz)) ! initialize array ------------------------------------------------------------------------------
DO i=1,ntc; DO j=1,ngz; C0(j,i)=0.0D0; END DO; END DO
DO j=1,ngz; T0(j)=0.0D0; END DO
tRun0=0.0D0;
dt=(tf-tRun)/(1.0D0*nt);  ! delta t at this dtLevel
IF (thermalType /= 1) THEN ! non-isothermal
  IF (thermalType == 2) THEN; CALL sTemperature; END IF
  CALL sDiffusion
  CALL sTau
  CALL sRateConstant
END IF
CALL sBoundary
CALL sReaction
DO it=1,nt ! time ingegration -------------------------------------------------------------------------------------------------
  IF (tRun+dt >= tf .OR. it==nt) THEN ! syncronize the last delta t in case of round off error ----------------------------------
```

Table E.9 Continued

```
  flagLastStep=0;
  dt=tf-tRun;
ELSE
  flagLastStep=flagLastStep-1;
END IF
C0=C; T0=T; tRun0=tRun; ! store data of previous time step
DO i=gStart,gEnd ! compute the linear diffusion part and store in LC
  DO j=rStart,rEnd
    LC(j,i)=((j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2); ! For CSTR, dZ is 1. Gas diffuse on both sides, so the equation match
  END DO
  IF (CBC(1) == 3) THEN ! Danckwerts' BC on the left
      LC(rStart-1,i)=(C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart-1,i))/Tau(rStart-1,i);
      LC(rStart,i)=LC(rStart,i)+(C(rStart-1,i)-C(rStart,i))/TauB(1,i);
  END IF
  IF (CBC(2) == 3) THEN ! Danckwerts' BC on the right
      LC(rEnd+1,i)=(C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd,i)-C(rEnd+1,i))/Tau(rEnd+1,i);
      LC(rEnd,i)=LC(rEnd,i)+(C(rEnd+1,i)-C(rEnd,i))/TauB(2,i);
  END IF
END DO
DO i=gStart,min(gEnd,iEnd)
  DO j=nmStart,nmEnd ! including gas on the boundary
      C(j,i)=C(j,i)+dt*(LC(j,i)+NC(j,i));
    END DO
END DO
DO i=sStart,min(cEnd,iEnd)
  DO j=rStart,rEnd ! solid only
    C(j,i)=C(j,i)+dt*(LC(j,i)+NC(j,i));
    END DO
END DO
IF (ecm /= 0) THEN ! execute ecm algorithm
  CALL sElement(C0,dt) ! update C(nmStart:nmEnd,max(gStart,dStart):gEnd) for gas and C(rStart:rEnd,max(sStart,dStart):cEnd)
END IF
maxDC=max(MAXVAL(abs(C(nmStart:nmEnd,gStart:gEnd)-C0(nmStart:nmEnd,gStart:gEnd))), &
          MAXVAL(abs(C(rStart:rEnd,sStart:cEnd)-C0(rStart:rEnd,sStart:cEnd)))));     ! maximum concentration change
minC=min(MINVAL(C(nmStart:nmEnd,gStart:gEnd)),MINVAL(C(rStart:rEnd,sStart:cEnd))); ! minimum concentration
IF (thermalType==3) THEN ! solve energy equation
  CALL sEnergy
  maxDT=MAXVAL(abs(T(nmStart:nmEnd)-T0(nmStart:nmEnd))); ! maximum change in temperature
  minT=MINVAL(T);                                        ! minimum temperature
END IF
IF (maxDC > tolMaxDC .OR. minC < tolMinC .OR. maxDT > tolMaxDT .OR. minT < tolMinT) THEN ! adaptive time control ----------------|
    tooStiff=.TRUE.;
  C=C0; T=T0; tRun=tRun0;
    IF (switchMethod) THEN; RETURN; END IF
  IF (dtLevel == 1) THEN ! main loop
    tRun0=min(tRun+dt,tf); ! store in tRun0 temporary
```

Table E.9 Continued

```
      dtLevel=max(dtLevelOld-1,1); ! set the dtLevel to dtLevelOld-1, so the next adaptive loop starts the same as last dtLevel
    DO WHILE (tRun < tRun0)
      IF (tooStiff) THEN
        dtLevel=dtLevel+1; ! increase the number of sub-loops if the previous sub-loop returns stiff signal
        tooStiff=.FALSE.;
      ELSE
        dtLevel=max(dtLevel-1,1); ! decrease the number of sub-loops if the previous sub-loop didn't return stiff signal
        tooStiff=.FALSE.;
      END IF
      CALL sEE(min(tRun+((1.0D0/nst)**max((dtLevel-2),0))*dt,tRun0),nst) ! adaptive time integration
    END DO
      dtLevelOld=dtLevel; ! save the last dtLevel, so the next adaptive loop will start at the same level
    dtLevel=1; ! important!! pull back to dtLevel 1 if the DO WHILE condition satisfied
  ELSEIF (dtLevel > 1) THEN ! sub loop
    RETURN
  ELSE
    WRITE(6,*) "Error: dtLevel <= 0."
    PAUSE
    STOP
  END IF
 ELSE ! passed
  tooStiff=.FALSE.;
  IF (flagLastStep == 0) THEN ! syncronize the last delta t in case of round off error
    tRun=tf;
  ELSE
    tRun=tRun+dt;
  END IF
 END IF
 IF (thermalType /= 1) THEN ! update variables and parameters ---------------------------------------------------------------------|
     IF (thermalType == 2) THEN; CALL sTemperature; END IF ! temperature ramp
     CALL sDiffusion
     CALL sTau
     CALL sRateConstant
 END IF
 CALL sBoundary
 CALL sReaction ! compute the nonlinear reaction part and store in NC (alpha*S*R for gas and S*R for solid)
 IF (dtLevel==1 .AND. .NOT.switchMethod) THEN; CALL sSave("Dynamic"); END IF ! save date
 IF (tRun >= tRuntime) THEN; CALL sRuntime(6); END IF ! write runtime information on screen
 IF (tRun>tf) THEN
   WRITE(6,*) "Warning: Current time is greater than final time. tRun > tf at dtLevel ", dtLevel, " and tRun-tf=", tRun-tf, ".";
 END IF
 IF (flagLastStep == 0) THEN; EXIT; END IF ! exit this loop
END DO
IF (ALLOCATED(C0)) THEN; DEALLOCATE(C0); END IF ! deallocate array ---------------------------------------------------------------|
IF (ALLOCATED(T0)) THEN; DEALLOCATE(T0); END IF
END SUBROUTINE sEE
```

Table E.10 sRK2.f90

```
!-----------------------------------------------------------------------
! Subroutine sRK2
! Description: Second order Runge-Kutta method (midpoint method).
! Created by Hsu-Wen Hsiao on 8/14/10.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   tf: final time
!   nt: number of time step
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sRK2(tf,nt)
USE mDynamic
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: tf ! final time
INTEGER, INTENT(IN) :: nt          ! number of time step
!-----------------------------------------------------------------------
DOUBLE PRECISION, PARAMETER :: zeroC=EPSILON(zeroC) ! concentration less than thiswill be set to zero
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: C0 ! pre-stored concentration of previous time step
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: T0   ! pre-stored temperature of previous time step
DOUBLE PRECISION :: tRun0                           ! pre-stored time of previous time step
DOUBLE PRECISION, DIMENSION(2), PARAMETER :: h=(/5.0D-1, 1.0D0/) ! rk time step size
DOUBLE PRECISION, DIMENSION(2) :: hdt
DOUBLE PRECISION :: dt                              ! step size
DOUBLE PRECISION :: maxDC, minC, maxDT, minT        ! parameter for adaptive time control
INTEGER :: it , i, j, rk, flagLastStep              ! dummy indices
IF (dtLevel > maxdtLevel) THEN ! initialize array for adaptive time control ----------------------------------------------------------------
   WRITE(6,*) "Warning ( dtLevel > 200 ): Unstable numerical method!! Please use smaller tfRun or change numerical method."
END IF
maxDC=0.0D0;     ! maximum concentration change
maxDT=0.0D0;     ! maximum temperature change
minC=MINVAL(C);  ! minimum concentration
minT=MINVAL(T);  ! minimum temperature
flagLastStep=nt;  ! flag to syncronize the last step
tooStiff=.FALSE.;! signal of stiffness for the sub-loop of adaptive time control
ALLOCATE(C0(ngz,ntc),T0(ngz)) ! initialize array ----------------------------------------------------------------
DO i=1,ntc; DO j=1,ngz; C0(j,i)=0.0D0; END DO; END DO
DO j=1,ngz; T0(j)=0.0D0; END DO
tRun0=0.0D0;
dt=(tf-tRun)/(1.0D0*nt);    ! delta t at this dtLevel
DO rk=1,2; hdt(rk)=h(rk)*dt; END DO
IF (thermalType /= 1) THEN ! non-isothermal
  IF (thermalType == 2) THEN; CALL sTemperature; END IF
  CALL sDiffusion; CALL sTau; CALL sRateConstant
END IF
CALL sBoundary; CALL sReaction
DO it=1,nt ! time ingegration ----------------------------------------------------------------
```

Table E.10 Continued

```
IF (tRun+dt >= tf .OR. it==nt) THEN ! syncronize the last delta t in case of round off error ----------------------------------------|
  flagLastStep=0;
  dt=tf-tRun;
    DO rk=1,2; hdt(rk)=h(rk)*dt; END DO
ELSE
  flagLastStep=flagLastStep-1;
END IF
C0=C; T0=T; tRun0=tRun; ! store data of previous time step
DO rk=1,2
  DO i=gStart,gEnd ! compute the linear diffusion part and store in LC
    IF (CBC(1) == 3) THEN ! Danckwerts' BC on the left
      LC(rStart-1,i)=(C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart,i)-C(rStart-1,i))/dZ/Tau(rStart-1,i);
    END IF
    DO j=rStart,rEnd
      LC(j,i)=(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2); ! For CSTR, dZ is 1, so the equation match.
    END DO
    IF (CBC(2) == 3) THEN ! Danckwerts' BC on the right
      LC(rEnd+1,i)=(C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd,i)-C(rEnd+1,i))/dZ/Tau(rEnd+1,i);
    END IF
  END DO
  DO i=gStart,min(gEnd,iEnd)
    DO j=nmStart,nmEnd ! including gas on the boundary
      C(j,i)=C(j,i)+hdt(rk)*(LC(j,i)+NC(j,i));
      END DO
  END DO
  DO i=sStart,min(cEnd,iEnd)
    DO j=rStart,rEnd ! solid only
      C(j,i)=C(j,i)+hdt(rk)*(LC(j,i)+NC(j,i));
      END DO
  END DO
  IF (ecm /= 0) THEN ! execute ecm algorithm
    CALL sElement(C0,dt) ! update C(nmStart:nmEnd,max(gStart,dStart):gEnd) for gas and C(rStart:rEnd,max(sStart,dStart):cEnd)
  END IF
  IF (thermalType==3) THEN ! solve energy equation
    CALL sEnergy
    maxDT=MAXVAL(abs(T(nmStart:nmEnd)-T0(nmStart:nmEnd))); ! maximum change in temperature
    minT=MINVAL(T);                                        ! minimum temperature
  END IF
    tRun=tRun0+hdt(rk);
  IF (thermalType /= 1) THEN ! update variables and parameters ----------------------------------------------------------------------|
    IF (thermalType == 2) THEN; CALL sTemperature; END IF ! temperature ramp
      CALL sDiffusion
      CALL sTau
      CALL sRateConstant
  END IF
  CALL sBoundary
```

Table E.10 Continued

```
  CALL sReaction ! compute the nonlinear reaction part and store in NC (alpha*S*R for gas and S*R for solid)
 END DO
 maxDC=max(MAXVAL(abs((C(nmStart:nmEnd,gStart:gEnd)-C0(nmStart:nmEnd,gStart:gEnd))), &
          MAXVAL(abs((C(rStart:rEnd,sStart:cEnd)-C0(rStart:rEnd,sStart:cEnd))));     ! maximum concentration change
 minC=MINVAL(C); ! minimum concentration
 IF (maxDC > tolMaxDC .OR. minC < tolMinC .OR. maxDT > tolMaxDT .OR. minT < tolMinT) THEN ! adaptive time control ----------|
    tooStiff=.TRUE.;
   C=C0; T=T0; tRun=tRun0;
    IF (switchMethod) THEN; RETURN; END IF
   IF (dtLevel == 1) THEN ! main loop
    tRun0=min(tRun+dt,tf); ! store in tRun0 temporary
     dtLevel=max(dtLevelOld-1,1); ! set the dtLevel to dtLevelOld-1, so the next adaptive loop starts the same as last dtLevel
    DO WHILE (tRun < tRun0)
     IF (tooStiff) THEN
      dtLevel=dtLevel+1; ! increase the number of sub-loops if the previous sub-loop returns stiff signal
      tooStiff=.FALSE.;
     ELSE
      dtLevel=max(dtLevel-1,1); ! decrease the number of sub-loops if the previous sub-loop didn't return stiff signal
      tooStiff=.FALSE.;
     END IF
     CALL sRK2(min(tRun+((1.0D0/nst)**max((dtLevel-2),0))*dt,tRun0),nst) ! adaptive time integration
    END DO
     dtLevelOld=dtLevel; ! save the last dtLevel, so the next adaptive loop will start at the same level
    dtLevel=1; ! important!! pull back to dtLevel 1 if the DO WHILE condition satisfied
   ELSEIF (dtLevel > 1) THEN ! sub loop
     RETURN
   ELSE
     WRITE(6,*) "Error: dtLevel <= 0."; PAUSE; STOP
    END IF
   ELSE ! passed
    tooStiff=.FALSE.;
    IF (flagLastStep == 0) THEN ! syncronize the last delta t in case of round off error
      tRun=tf;
    END IF
   END IF
   IF (dtLevel==1 .AND. .NOT.switchMethod) THEN; CALL sSave("Dynamic"); END IF ! save date
   IF (tRun >= tRuntime) THEN; CALL sRuntime(6); END IF ! write runtime information on screen
   IF (tRun>tf) THEN
    WRITE(6,*) "Warning: Current time is greater than final time. tRun > tf at dtLevel ", dtLevel, " and tRun-tf=", tRun-tf, ".";
   END IF
   IF (flagLastStep == 0) THEN; EXIT; END IF ! exit this loop
 END DO
 IF (ALLOCATED(C0)) THEN; DEALLOCATE(C0); END IF ! deallocate array ----------------------------------------------------------|
 IF (ALLOCATED(T0)) THEN; DEALLOCATE(T0); END IF
 END SUBROUTINE sRK2
```

Table E.11 sRK3.f90

```
!-----------------------------------------------------------------------
! Subroutine sRK3
! Description: Standard third order Runge-Kutta method.
! Created by Hsu-Wen Hsiao on 8/14/10.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   tf: final time
!   nt: number of time step
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sRK3(tf,nt)
USE mDynamic
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: tf ! final time
INTEGER, INTENT(IN) :: nt       ! number of time step
!-----------------------------------------------------------------------
!DOUBLE PRECISION, PARAMETER :: zeroC=EPSILON(zeroC)  ! concentration less than thiswill be set to zero
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: C0   ! pre-stored concentration of previous time step
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: T0     ! pre-stored temperature of previous time step
DOUBLE PRECISION, DIMENSION(:,:,:), ALLOCATABLE :: FC ! store LC+NC from previous time step
DOUBLE PRECISION :: tRun0, tRUn1                      ! pre-stored time of previous time step
DOUBLE PRECISION :: dt                                ! step size
DOUBLE PRECISION :: maxDC, minC, maxDT, minT          ! parameter for adaptive time control
INTEGER :: it , i, j, rk, flagLastStep                ! dummy indices
IF (dtLevel > maxdtLevel) THEN ! initialize array for adaptive time control ----------------------------------------------------
   WRITE(6,*) "Warning ( dtLevel > 200 ): Unstable numerical method!! Please use smaller tfRun or change numerical method."
END IF
maxDC=0.0D0;     ! maximum concentration change
maxDT=0.0D0;     ! maximum temperature change
minC=MINVAL(C);  ! minimum concentration
minT=MINVAL(T);  ! minimum temperature
flagLastStep=nt; ! flag to syncronize the last step
tooStiff=.FALSE.;! signal of stiffness for the sub-loop of adaptive time control
ALLOCATE(C0(ngz,ntc),T0(ngz)) ! initialize array ------------------------------------------------------------------------------
DO i=1,ntc; DO j=1,ngz; C0(j,i)=0.0D0; END DO; END DO
DO j=1,ngz; T0(j)=0.0D0; END DO
ALLOCATE(FC(ngz,ntc,2))
DO rk=1,2; DO i=1,ntc; DO j=1,ngz; FC(j,i,rk)=0.0D0; END DO; END DO;END DO
tRun0=0.0D0;
dt=(tf-tRun)/(1.0D0*nt);   ! delta t at this dtLevel
IF (thermalType /= 1) THEN ! non-isothermal
   IF (thermalType == 2) THEN; CALL sTemperature; END IF
   CALL sDiffusion
   CALL sTau
   CALL sRateConstant
END IF
```

Table E.11 Continued

```
CALL sBoundary
CALL sReaction
DO it=1,nt ! time ingegration ------------------------------------------------------------------------------------|
  IF (tRun+dt >= tf .OR. it==nt) THEN ! syncronize the last delta t in case of round off error -----------------------------------|
    flagLastStep=0;
    dt=tf-tRun;
       tRun=tf;
  ELSE
    flagLastStep=flagLastStep-1;
       tRun1=tRun+dt;
  END IF
  C0=C; T0=T; tRun0=tRun; ! store data of previous time step
  DO i=gStart,gEnd ! 1st RK step ----------------------------------------------------------------------------------|
    IF (CBC(1) == 3) THEN ! Danckwerts' BC on the left
      LC(rStart-1,i)=(C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart,i)-C(rStart-1,i))/dZ/Tau(rStart-1,i);
    END IF
    DO j=rStart,rEnd
      LC(j,i)=(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2); ! For CSTR, dZ is 1. Gas diffuse on both sides, so the equation match
    END DO
    IF (CBC(2) == 3) THEN ! Danckwerts' BC on the right
      LC(rEnd+1,i)=(C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd,i)-C(rEnd+1,i))/dZ/Tau(rEnd+1,i);
    END IF
  END DO
  DO i=gStart,min(gEnd,iEnd)
    DO j=nmStart,nmEnd
         FC(j,i,1)=LC(j,i)+NC(j,i); ! store data
         C(j,i)=C(j,i)+0.5*dt*FC(j,i,1); ! integrate to next rk step
       END DO
  END DO
  DO i=sStart,min(cEnd,iEnd)
    DO j=rStart,rEnd ! solid only
         FC(j,i,1)=NC(j,i); ! store data
         C(j,i)=C(j,i)+0.5*dt*FC(j,i,1);
       END DO
  END DO
  IF (ecm /= 0) THEN ! execute ecm algorithm
    CALL sElement(C0,dt) ! update C(nmStart:nmEnd,max(gStart,dStart):gEnd) for gas and C(rStart:rEnd,max(sStart,dStart):cEnd)
  END IF
  IF (thermalType==3) THEN ! solve energy equation
    CALL sEnergy
  END IF
  tRun=tRun0+0.5*dt;
  IF (thermalType /= 1) THEN ! update variables and parameters
       IF (thermalType == 2) THEN; CALL sTemperature; END IF ! temperature ramp
       CALL sDiffusion
       CALL sTau
```

Table E.11 Continued

```
     CALL sRateConstant
  END IF
  CALL sBoundary
  CALL sReaction ! compute the nonlinear reaction part and store in NC (alpha*S*R for gas and S*R for solid)
  DO i=gStart,gEnd ! 2nd RK step ----------------------------------------------------------------------------------------|
    IF (CBC(1) == 3) THEN ! Danckwerts' BC on the left
      LC(rStart-1,i)=(C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart,i)-C(rStart-1,i))/dZ/Tau(rStart-1,i);
    END IF
    DO j=rStart,rEnd
      LC(j,i)=(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2); ! For CSTR, dZ is 1. Gas diffuse on both sides, so the equation match
    END DO
    IF (CBC(2) == 3) THEN ! Danckwerts' BC on the right
      LC(rEnd+1,i)=(C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd,i)-C(rEnd+1,i))/dZ/Tau(rEnd+1,i);
    END IF
  END DO
  DO i=gStart,min(gEnd,iEnd)
    DO j=nmStart,nmEnd
        FC(j,i,2)=LC(j,i)+NC(j,i); ! store data
        C(j,i)=C(j,i)+dt*(2*FC(j,i,2)-FC(j,i,1)); ! integrate to next rk step
     END DO
  END DO
  DO i=sStart,min(cEnd,iEnd)
    DO j=rStart,rEnd ! solid only
        FC(j,i,2)=NC(j,i); ! store data
        C(j,i)=C(j,i)+dt*(2*FC(j,i,2)-FC(j,i,1));
     END DO
  END DO
  IF (ecm /= 0) THEN ! execute ecm algorithm
    CALL sElement(C0,dt) ! update C(nmStart:nmEnd,max(gStart,dStart):gEnd) for gas and C(rStart:rEnd,max(sStart,dStart):cEnd)
  END IF
  IF (thermalType==3) THEN ! solve energy equation
    CALL sEnergy
  END IF
  tRun=tRun1;
  IF (thermalType /= 1) THEN ! update variables and parameters
     IF (thermalType == 2) THEN; CALL sTemperature; END IF ! temperature ramp
     CALL sDiffusion
     CALL sTau
     CALL sRateConstant
  END IF
  CALL sBoundary
  CALL sReaction ! compute the nonlinear reaction part and store in NC (alpha*S*R for gas and S*R for solid)
  DO i=gStart,gEnd ! 3rd RK step ----------------------------------------------------------------------------------------|
    IF (CBC(1) == 3) THEN ! Danckwerts' BC on the left
      LC(rStart-1,i)=(C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart,i)-C(rStart-1,i))/dZ/Tau(rStart-1,i);
    END IF
```

Table E.11 Continued

```
  DO j=rStart,rEnd
    LC(j,i)=(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2); ! For CSTR, dZ is 1. Gas diffuse on both sides, so the equation match
  END DO
  IF (CBC(2) == 3) THEN ! Danckwerts' BC on the right
    LC(rEnd+1,i)=(C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd+1,i)-C(rEnd+1,i))/dZ/Tau(rEnd+1,i);
  END IF
END DO
DO i=gStart,min(gEnd,iEnd)
  DO j=nmStart,nmEnd
     C(j,i)=C(j,i)+(dt/6.0D0)*(LC(j,i)+NC(j,i)+4*FC(j,i,2)+FC(j,i,1)); ! integrate to next rk step
    END DO
END DO
DO i=sStart,min(cEnd,iEnd)
  DO j=rStart,rEnd ! solid only

     C(j,i)=C(j,i)+(dt/6.0D0)*(NC(j,i)+4*FC(j,i,2)+FC(j,i,1));
    END DO
END DO
IF (ecm /= 0) THEN ! execute ecm algorithm
  CALL sElement(C0,dt) ! update C(nmStart:nmEnd,max(gStart,dStart):gEnd) for gas and C(rStart:rEnd,max(sStart,dStart):cEnd)
END IF
IF (thermalType==3) THEN ! solve energy equation
  CALL sEnergy
END IF
tRun=tRun1;
IF (thermalType /= 1) THEN ! update variables and parameters
   IF (thermalType == 2) THEN; CALL sTemperature; END IF ! temperature ramp
    CALL sDiffusion
    CALL sTau
    CALL sRateConstant
END IF
CALL sBoundary
CALL sReaction ! compute the nonlinear reaction part and store in NC (alpha*S*R for gas and S*R for solid)
IF (thermalType==3) THEN ! solve energy equation
  maxDT=MAXVAL(abs(T(nmStart:nmEnd)-T0(nmStart:nmEnd))); ! maximum change in temperature
  minT=MINVAL(T);                                        ! minimum temperature
END IF
maxDC=max(MAXVAL(abs(C(nmStart:nmEnd,gStart:gEnd)-C0(nmStart:nmEnd,gStart:gEnd))), &
          MAXVAL(abs(C(rStart:rEnd,sStart:cEnd)-C0(rStart:rEnd,sStart:cEnd))));     ! maximum concentration change
minC=MINVAL(C); ! minimum concentration
IF (maxDC > tolMaxDC .OR. minC < tolMinC .OR. maxDT > tolMaxDT .OR. minT < tolMinT) THEN ! adaptive time control ----------------|
    tooStiff=.TRUE.;
  C=C0; T=T0; tRun=tRun0;
   IF (switchMethod) THEN; RETURN; END IF
IF (dtLevel == 1) THEN ! main loop
  tRun0=min(tRun+dt,tf); ! store in tRun0 temporary
```

Table E.11 Continued

```
    dtLevel=max(dtLevelOld-1,1); ! set the dtLevel to dtLevelOld-1, so the next adaptive loop starts the same as last dtLevel
  DO WHILE (tRun < tRun0)
    IF (tooStiff) THEN
      dtLevel=dtLevel+1; ! increase the number of sub-loops if the previous sub-loop returns stiff signal
      tooStiff=.FALSE.;
    ELSE
      dtLevel=max(dtLevel-1,1); ! decrease the number of sub-loops if the previous sub-loop didn't return stiff signal
      tooStiff=.FALSE.;
    END IF
    CALL sRK3(min(tRun+((1.0D0/nst)**max((dtLevel-2),0))*dt,tRun0),nst) ! adaptive time integration
  END DO
    dtLevelOld=dtLevel; ! save the last dtLevel, so the next adaptive loop will start at the same level
    dtLevel=1; ! important!! pull back to dtLevel 1 if the DO WHILE condition satisfied
  ELSEIF (dtLevel > 1) THEN ! sub loop
    RETURN
  ELSE
    WRITE(6,*) "Error: dtLevel <= 0."
    PAUSE
    STOP
  END IF
ELSE ! passed
  tooStiff=.FALSE.;
  IF (flagLastStep == 0) THEN ! syncronize the last delta t in case of round off error
    tRun=tf;
  END IF
END IF
IF (dtLevel==1 .AND. .NOT.switchMethod) THEN; CALL sSave("Dynamic"); END IF ! save date
IF (tRun >= tRuntime) THEN; CALL sRuntime(6); END IF ! write runtime information on screen
IF (tRun>tf) THEN
  WRITE(6,*) "Warning: Current time is greater than final time. tRun > tf at dtLevel ", dtLevel, " and tRun-tf=", tRun-tf, ".";
END IF
IF (flagLastStep == 0) THEN; EXIT; END IF ! exit this loop
END DO
IF (ALLOCATED(C0)) THEN; DEALLOCATE(C0); END IF ! deallocate array ----------------------------------------------------------------|
IF (ALLOCATED(T0)) THEN; DEALLOCATE(T0); END IF
IF (ALLOCATED(FC)) THEN; DEALLOCATE(FC); END IF
END SUBROUTINE sRK3
```

Table E.12 sRK4.f90

```
!-----------------------------------------------------------------------
! Subroutine sRK4
! Description: Standard fourth order Runge-Kutta method.
! Created by Hsu-Wen Hsiao on 8/14/10.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   tf: final time
!   nt: number of time step
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sRK4(tf,nt)
USE mDynamic
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: tf ! final time
INTEGER, INTENT(IN) :: nt       ! number of time step
!-----------------------------------------------------------------------
!DOUBLE PRECISION, PARAMETER :: zeroC=EPSILON(zeroC)  ! concentration less than thiswill be set to zero
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: C0, C1   ! pre-stored concentration of previous time step
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: T0, T1     ! pre-stored temperature of previous time step
DOUBLE PRECISION :: tRun0, tRun1                          ! pre-stored time of previous time step
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: FC       ! storage for the 4th rk step
DOUBLE PRECISION, DIMENSION(4), PARAMETER ::    h=(/5.0D-1, 5.0D-1, 1.0D0, 1.0D0/) ! rk step size
DOUBLE PRECISION, DIMENSION(4), PARAMETER :: Beta=(/1.0D0/6.0D0,  2.0D0/6.0D0, 2.0D0/6.0D0, 1.0D0/6.0D0/) ! FC step size
DOUBLE PRECISION, DIMENSION(4) :: hdt
DOUBLE PRECISION :: dt                                    ! step size
DOUBLE PRECISION :: maxDC, minC, maxDT, minT              ! parameter for adaptive time control
INTEGER :: it , i, j, rk, flagLastStep                    ! dummy indices
IF (dtLevel > maxdtLevel) THEN ! initialize array for adaptive time control ----------------------------------------------------------------
  WRITE(6,*) "Warning ( dtLevel > 200 ): Unstable numerical method!! Please use smaller tfRun or change numerical method."
END IF
maxDC=0.0D0;     ! maximum concentration change
maxDT=0.0D0;     ! maximum temperature change
minC=MINVAL(C);  ! minimum concentration
minT=MINVAL(T);  ! minimum temperature
flagLastStep=nt;  ! flag to syncronize the last step
tooStiff=.FALSE.;! signal of stiffness for the sub-loop of adaptive time control
ALLOCATE(C0(ngz,ntc),T0(ngz),C1(ngz,ntc),T1(ngz)) ! initialize array ----------------------------------------------------------------
DO i=1,ntc; DO j=1,ngz; C0(j,i)=0.0D0; C1(j,i)=0.0D0; END DO; END DO
DO j=1,ngz; T0(j)=0.0D0; T1(j)=0.0D0; END DO
ALLOCATE(FC(ngz,ntc))
DO i=1,ntc; DO j=1,ngz; FC(j,i)=0.0D0; END DO; END DO
tRun0=0.0D0;
dt=(tf-tRun)/(1.0D0*nt);   ! delta t at this dtLevel
hdt=h*dt;
IF (thermalType /= 1) THEN ! non-isothermal
  IF (thermalType == 2) THEN; CALL sTemperature; END IF
```

Table E.12 Continued

```
  CALL sDiffusion
  CALL sTau
  CALL sRateConstant
END IF
CALL sBoundary
CALL sReaction
DO it=1,nt ! time ingegration ------------------------------------------------------------------------------------------|
  IF (tRun+dt >= tf .OR. it==nt) THEN ! syncronize the last delta t in case of round off error ----------------------------------|
    flagLastStep=0;
    dt=tf-tRun;
       hdt=h*dt;
       tRun=tf;
  ELSE
    flagLastStep=flagLastStep-1;
       tRun1=tRun+dt;
  END IF
  C0=C; T0=T; tRun0=tRun; ! store data of previous time step
  DO i=1,ntc; DO j=1,ngz; FC(j,i)=0.0D0; END DO; END DO
  DO rk=1,4 ! loop of RK step ------------------------------------------------------------------------------------------|
    C1=C; T1=T;
       DO i=gStart,gEnd
         IF (CBC(1) == 3) THEN ! Danckwerts' BC on the left
        LC(rStart-1,i)=(C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart,i)-C(rStart-1,i))/dZ/Tau(rStart-1,i);
      END IF
      DO j=rStart,rEnd
        LC(j,i)=(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2);
         END DO
      IF (CBC(2) == 3) THEN ! Danckwerts' BC on the right
        LC(rEnd+1,i)=(C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd,i)-C(rEnd+1,i))/dZ/Tau(rEnd+1,i);
      END IF
    END DO
       IF (rk > 0 .AND. rk < 4) THEN
      DO i=gStart,min(gEnd,iEnd)
        DO j=nmStart,nmEnd
           FC(j,i)=FC(j,i)+Beta(rk)*(LC(j,i)+NC(j,i));
             C(j,i)=C(j,i)+hdt(rk)*(LC(j,i)+NC(j,i)); ! integrate to next rk step
           END DO
      END DO
      DO i=sStart,min(cEnd,iEnd)
        DO j=rStart,rEnd ! solid only
           FC(j,i)=FC(j,i)+Beta(rk)*NC(j,i);
           C(j,i)=C(j,i)+hdt(rk)*NC(j,i);
           END DO
      END DO
       ELSEIF (rk == 4) THEN
      DO i=gStart,min(gEnd,iEnd)
```

Table E.12 Continued

```
    DO j=nmStart,nmEnd
      C(j,i)=C(j,i)+hdt(rk)*(FC(j,i)+LC(j,i)+NC(j,i)); ! integrate to next rk step
    END DO
  END DO
  DO i=sStart,min(cEnd,iEnd)
    DO j=rStart,rEnd ! solid only
      C(j,i)=C(j,i)+hdt(rk)*(FC(j,i)+NC(j,i));
    END DO
  END DO
  END IF
IF (ecm /= 0) THEN ! execute ecm algorithm
  CALL sElement(C1,dt) ! update C(nmStart:nmEnd,max(gStart,dStart):gEnd) for gas and C(rStart:rEnd,max(sStart,dStart):cEnd)
END IF
IF (thermalType==3) THEN ! solve energy equation
  CALL sEnergy
END IF
tRun=tRun0+hdt(rk);
IF (thermalType /= 1) THEN ! update variables and parameters
  IF (thermalType == 2) THEN; CALL sTemperature; END IF ! temperature ramp
  CALL sDiffusion
  CALL sTau
  CALL sRateConstant
END IF
CALL sBoundary
CALL sReaction ! compute the nonlinear reaction part and store in NC (alpha*S*R for gas and S*R for solid)
END DO
IF (thermalType==3) THEN ! solve energy equation
  maxDT=MAXVAL(abs(T(nmStart:nmEnd)-T0(nmStart:nmEnd))); ! maximum change in temperature
  minT=MINVAL(T);                                        ! minimum temperature
END IF
maxDC=max(MAXVAL(abs(C(nmStart:nmEnd,gStart:gEnd)-C0(nmStart:nmEnd,gStart:gEnd))), &
          MAXVAL(abs(C(rStart:rEnd,sStart:cEnd)-C0(rStart:rEnd,sStart:cEnd))));     ! maximum concentration change
minC=MINVAL(C); ! minimum concentration
!IF (minC < 0.0D0 .AND. abs(minC) < zeroC) THEN
!  DO i=1,ntc
!    DO j=1,ngz
!      IF (C(j,i)<0.0D0 .AND. abs(C(j,i))<zeroC) THEN
!        C(j,i)=0.0D0;
!      END IF
!    END DO
!  END DO
!  minC=MINVAL(C); ! recalculate minimum concentration
!END IF
IF (maxDC > tolMaxDC .OR. minC < tolMinC .OR. maxDT > tolMaxDT .OR. minT < tolMinT) THEN ! adaptive time control ----------------|
   tooStiff=.TRUE.;
  C=C0; T=T0; tRun=tRun0;
```

Table E.12 Continued

```
    IF (switchMethod) THEN; RETURN; END IF
  IF (dtLevel == 1) THEN ! main loop
    tRun0=min(tRun+dt,tf); ! store in tRun0 temporary
      dtLevel=max(dtLevelOld-1,1); ! set the dtLevel to dtLevelOld-1, so the next adaptive loop starts the same as last dtLevel
    DO WHILE (tRun < tRun0)
      IF (tooStiff) THEN
        dtLevel=dtLevel+1; ! increase the number of sub-loops if the previous sub-loop returns stiff signal
        tooStiff=.FALSE.;
      ELSE
        dtLevel=max(dtLevel-1,1); ! decrease the number of sub-loops if the previous sub-loop didn't return stiff signal
        tooStiff=.FALSE.;
      END IF
      CALL sRK4(min(tRun+((1.0D0/nst)**max((dtLevel-2),0))*dt,tRun0),nst) ! adaptive time integration
    END DO
      dtLevelOld=dtLevel; ! save the last dtLevel, so the next adaptive loop will start at the same level
    dtLevel=1; ! important!! pull back to dtLevel 1 if the DO WHILE condition satisfied
  ELSEIF (dtLevel > 1) THEN ! sub loop
    RETURN
  ELSE
    WRITE(6,*) "Error: dtLevel <= 0."
    PAUSE
    STOP
  END IF
ELSE ! passed
  tooStiff=.FALSE.;
  IF (flagLastStep == 0) THEN ! syncronize the last delta t in case of round off error
    tRun=tf;
  END IF
END IF
IF (dtLevel==1 .AND. .NOT.switchMethod) THEN; CALL sSave("Dynamic"); END IF ! save date
IF (tRun >= tRuntime) THEN; CALL sRuntime(6); END IF ! write runtime information on screen
IF (tRun>tf) THEN
  WRITE(6,*) "Warning: Current time is greater than final time. tRun > tf at dtLevel ", dtLevel, " and tRun-tf=", tRun-tf, ".";
END IF
IF (flagLastStep == 0) THEN; EXIT; END IF ! exit this loop
END DO
IF (ALLOCATED(C0)) THEN; DEALLOCATE(C0); END IF ! deallocate array ----------------------------------------------------------------------|
IF (ALLOCATED(T0)) THEN; DEALLOCATE(T0); END IF
IF (ALLOCATED(C1)) THEN; DEALLOCATE(C1); END IF
IF (ALLOCATED(T1)) THEN; DEALLOCATE(T1); END IF
IF (ALLOCATED(FC)) THEN; DEALLOCATE(FC); END IF
END SUBROUTINE sRK4
```

Table E.13 sCNRK3.f90

```
!-----------------------------------------------------------------------
! Subroutine sCNRK3
! Description: Crank-Nicholson and RKW3 method for time integration. (still need algorithm for ecm)
! Created by Hsu-Wen Hsiao on 2/18/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   tf: final time
!   nt: number of time step
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sCNRK3(tf,nt)
USE mDynamic
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: tf                ! final time
INTEGER, INTENT(IN) :: nt                         ! number of time step
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
DOUBLE PRECISION :: dt                            ! step size
DOUBLE PRECISION, PARAMETER :: zeroC=EPSILON(zeroC); ! zero concentration (for eliminate round off error)
DOUBLE PRECISION :: maxDC, minC, maxDT, minT      ! parameter for adaptive time control
DOUBLE PRECISION, DIMENSION(:,:,:), ALLOCATABLE :: C0 ! pre-stored concentration of previous time step and RK substep
DOUBLE PRECISION, DIMENSION(:,:),   ALLOCATABLE :: T0 ! pre-stored temperature of previous time step and RK substep
DOUBLE PRECISION :: tRun0                         ! pre-stored time of previous time step
DOUBLE PRECISION, DIMENSION(3), PARAMETER ::    h=(/8.0D0/1.5D1, 2.0D0/1.5D1, 1.0D0/3.0D0/) ! rk time step size
DOUBLE PRECISION, DIMENSION(3), PARAMETER :: Beta=(/8.0D0/1.5D1, 2.5D1/8.0D0, 9.0D0/4.0D0/) ! rk current NC step size
DOUBLE PRECISION, DIMENSION(3), PARAMETER :: Zeta=(/0.0D0     ,-1.7D1/8.0D0,-5.0D0/4.0D0/) ! rk previous NC step size
DOUBLE PRECISION, DIMENSION(:,:,:,:), ALLOCATABLE :: A
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: B
INTEGER :: it ,i, j, k, rk, flagLastStep          ! dummy indices
IF (dtLevel > maxdtLevel) THEN ! initialize array for adaptive time control ----------------------------------------------------------------------|
   WRITE(6,*) "Warning ( dtLevel > 200 ): Unstable numerical method!! Please use smaller tfRun or change numerical method."
END IF
maxDC=0.0D0;     ! maximum concentration change
maxDT=0.0D0;     ! maximum temperature change
minC=MINVAL(C);  ! minimum concentration
minT=MINVAL(T);  ! minimum temperature
flagLastStep=nt; ! flag to syncronize the last step
tooStiff=.FALSE.;! signal of stiffness for the sub-loop of adaptive time control
ALLOCATE(C0(ngz,ntc,3),T0(ngz,3)) ! initialize array ----------------------------------------------------------------------------------------|
DO rk=1,3; DO i=1,ntc; DO j=1,ngz; C0(j,i,rk)=0.0D0; END DO; END DO; END DO
DO rk=1,3; DO j=1,ngz; T0(j,rk)=0.0D0; END DO; END DO
tRun0=0.0D0;
dt=(tf-tRun)/(1.0D0*nt); ! delta t at this dtLevel
IF (thermalType /= 1) THEN ! non-isothermal
  IF (thermalType == 2) THEN; CALL sTemperature; END IF
  CALL sDiffusion
  CALL sTau
```

Table E.13 Continued

```
  CALL sRateConstant
END IF
CALL sBoundary
CALL sReaction
ALLOCATE(A(ngz,3,ngc,3),B(ngz,ntc)) ! initialize CNRK3 array ----------------------------------------------------------------------|
DO i=1,ntc
  DO j=1,ngz
    B(j,i)=0.0D0;
  END DO
END DO
DO rk=1,3
  DO i=1,ngc
    DO k=1,3
      DO j=1,ngz
        A(j,k,i,rk)=0.0D0;
      END DO
    END DO
  END DO
END DO
IF (thermalType == 1) THEN ! use LU decomposition, construct A (compact form of tridiagonal system)
  DO rk=1,3
    DO i=1,ngc
      DO j=rStart,rEnd
        A(j,1,i,rk)=     -0.5*h(rk)*dt/Tau(j,i)/(dZ**2); ! left diagonal
        A(j,2,i,rk)=1.0D0+1.0*h(rk)*dt/Tau(j,i)/(dZ**2); ! main diagonal
        A(j,3,i,rk)=     -0.5*h(rk)*dt/Tau(j,i)/(dZ**2); ! right diagonal
      END DO
      SELECT CASE (CBC(1))
        CASE (1) ! Dirichlet BC
          !A(rStart  ,1,i,rk)=0.0D0;
        CASE (2) ! Neumann BC
          A(rStart  ,3,i,rk)=A(rStart,3,i,rk)+A(rStart,1,i,rk);
          !A(rStart  ,1,i,rk)=0.0D0;
        CASE (3) ! Danckwert's BC
          !A(rStart-1,2,i,rk)=1.0D0+0.5*h(rk)*dt/Tau(rStart-2,i)+0.5*h(rk)*dt/Tau(rStart-1,i)/dZ;
          !A(rStart-1,3,i,rk)=                                 -0.5*h(rk)*dt/Tau(rStart-1,i)/dZ;
          A(rStart-1,1,i,rk)=     -0.5*h(rk)*dt/Tau(rStart-2,i);
          A(rStart-1,2,i,rk)=1.0D0+0.5*h(rk)*dt/Tau(rStart-2,i)+0.5*h(rk)*dt/Tau(rStart-1,i);
          A(rStart-1,3,i,rk)=                                  -0.5*h(rk)*dt/Tau(rStart-1,i);
          A(rStart,1,i,rk)=A(rStart,1,i,rk)-0.5*h(rk)*dt/TauB(1,i);
          A(rStart,2,i,rk)=A(rStart,2,i,rk)+0.5*h(rk)*dt/TauB(1,i);
      END SELECT
      SELECT CASE (CBC(2))
        CASE (1) ! Dirichlet BC
          !A(rEnd  ,3,i,rk)=0.0D0;
        CASE (2) ! Neumann BC
```

Table E.13 Continued

```
          A(rEnd  ,1,i,rk)=A(rStart,1,i,rk)+A(rStart,3,i,rk);
          !A(rEnd  ,3,i,rk)=0.0D0;
        CASE (3) ! Danckwert's BC
          !A(rEnd+1,1,i,rk)=                                -0.5*h(rk)*dt/Tau(rEnd+1,i)/dZ;
          !A(rEnd+1,2,i,rk)=1.0D0+0.5*h(rk)*dt/Tau(rEnd+2,i)+0.5*h(rk)*dt/Tau(rEnd+1,i)/dZ;
          A(rEnd,2,i,rk)=A(rEnd,2,i,rk)+0.5*h(rk)*dt/TauB(2,i);
          A(rEnd,3,i,rk)=A(rEnd,3,i,rk)-0.5*h(rk)*dt/TauB(2,i);
          A(rEnd+1,1,i,rk)=                                -0.5*h(rk)*dt/Tau(rEnd+1,i);
          A(rEnd+1,2,i,rk)=1.0D0+0.5*h(rk)*dt/Tau(rEnd+2,i)+0.5*h(rk)*dt/Tau(rEnd+1,i);
          A(rEnd+1,3,i,rk)=     -0.5*h(rk)*dt/Tau(rEnd+2,i);
        END SELECT
    END DO
  END DO
  DO rk=1,3 ! decompose LU and store in  original array
    DO i=gStart,gEnd
        DO j=nmStart,nmEnd
          A(j+1,1,i,rk)=A(j+1,1,i,rk)/A(j,2,i,rk);
          A(j+1,2,i,rk)=A(j+1,2,i,rk)-A(j,3,i,rk)*A(j+1,1,i,rk);
        END DO
    END DO
  END DO
END IF
DO it=1,nt ! time ingegration -----------------------------------------------------------------------------------------------|
  IF (tRun+dt >= tf .OR. it==nt) THEN ! syncronize the last delta t in case of round off error -------------------------------------|
    flagLastStep=0;
    dt=tf-tRun;
  ELSE
    flagLastStep=flagLastStep-1;
  END IF
  tRun0=tRun; ! store data of previous time step
  !C0()=C; T0=T; tRun0=tRun; ! store data of previous time step
  IF (thermalType == 1 .AND. it/=nt) THEN ! use LU decomposition
    DO rk=1,3
        C0(:,:,rk)=C; T0(:,rk)=T;
        DO i=gStart,min(gEnd,iEnd) ! store data from previous step for gas
          DO j=rStart,rEnd
          LC(j,i)=0.5*(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2);
          B(j,i)=C(j,i)+h(rk)*dt*(LC(j,i) + Zeta(rk)*NC(j,i));
          END DO
        SELECT CASE (CBC(1))
          CASE (1) ! Dirichlet
            B(rStart,i)=B(rStart,i)-A(rStart,1,i,rk)*C(rStart-1,i);
          CASE (2) ! Neumann (Do nothing)
          CASE (3) ! Danckwerts
            !LC(rStart-1,i)=0.5*((C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart  ,i)-C(rStart-1,i))/Tau(rStart-1,i)/dZ);
                LC(rStart-1,i)=0.5*((C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart  ,i)-C(rStart-1,i))/Tau(rStart-1,i));
```

Table E.13 Continued

```
      B(rStart-1,i)=C(rStart-1,i)+h(rk)*dt*LC(rStart-1,i);
      B(rStart-1,i)=B(rStart-1,i)-A(rStart-1,1,i,rk)*C(rStart-1,i);
      LC(rStart,i)=LC(rStart,i)+0.5*(C(rStart-1,i)-C(rStart,i))/TauB(1,i);
      B(rStart,i)=B(rStart,i)+0.5*h(rk)*dt*(C(rStart-1,i)-C(rStart,i))/TauB(1,i);
  END SELECT
  SELECT CASE (CBC(2))
    CASE (1) ! Dirichlet
      B(rEnd,i)=B(rEnd,i)-A(rEnd,3,i,rk)*C(rEnd+1,i);
    CASE (2) ! Neumann (Do nothing)
    CASE (3) ! Danckwerts
        !LC(rEnd+1,i)=0.5*(C(C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd   ,i)-C(rEnd+1,i))/Tau(rEnd+1,i)/dZ);
        LC(rEnd+1,i)=0.5*(C(C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd   ,i)-C(rEnd+1,i))/Tau(rEnd+1,i));
        B(rEnd+1,i)=C(rEnd+1,i)+h(rk)*dt*LC(rEnd+1,i);
        B(rEnd+1,i)=B(rEnd+1,i)-A(rEnd+1,3,i,rk)*C(rEnd+1,i);
        LC(rEnd,i)=LC(rEnd,i)+0.5*(C(rEnd+1,i)-C(rEnd,i))/TauB(2,i);
        B(rEnd,i)=B(rEnd,i)+0.5*h(rk)*dt*(C(rEnd+1,i)-C(rEnd,i))/TauB(2,i);
  END SELECT
  END DO
  DO i=sStart,min(cEnd,iEnd) ! store data from previous step for solid
    DO j=rStart,rEnd
      B(j,i)=C(j,i)+h(rk)*dt*Zeta(rk)*NC(j,i);
  END DO
  END DO
  CALL sReaction ! now we can compute the reaction part and store in NC
  DO i=gStart,min(gEnd,iEnd) ! add the reaction part for gas at current step
    DO j=rStart,rEnd
    B(j,i)=B(j,i)+dt*h(rk)*Beta(rk)*NC(j,i);
  END DO
  END DO
  DO i=gStart,min(gEnd,iEnd) ! solve for the gas component by LU decomposition
    DO j=nmStart+1,nmEnd
    B(j,i)=B(j,i)-A(j,1,i,rk)*B(j-1,i);
    END DO
    C(nmEnd,i)=B(nmEnd,i)/A(nmEnd,2,i,rk);
    DO j=(nmEnd-1),nmStart,-1
    C(j,i)=(B(j,i)-A(j,3,i,rk)*C(j+1,i))/A(j,2,i,rk);
    END DO
  END DO
  DO i=sStart,min(cEnd,iEnd) ! solve the solid component
    DO j=rStart,rEnd
      C(j,i)=B(j,i)+h(rk)*dt*NC(j,i);
    END DO
  END DO
  SELECT CASE (ecm) ! execute ecm algorithm
  CASE (0) ! non-ecm
  CASE (1)
```

Table E.13 Continued

```
    CALL sElement(C0(:,:,rk),h(rk)*dt)
  CASE (2)
    DO i=gStart,min(gEnd,iEnd) ! construct linear part for current time step
      IF (CBC(1)==3) THEN
          LC(rStart-1,i)=LC(rStart-1,i)+0.5*(((C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i) &
                                            +(C(rStart,i)-C(rStart-1,i))/Tau(rStart-1,i));
        LC(rStart,i)=LC(rStart,i)+0.5*((C(rStart-1,i)-C(rStart,i))/TauB(1,i);
        END IF
      DO j=rStart,rEnd
        LC(j,i)=LC(j,i)+0.5*(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2);
      END DO
        IF (CBC(2)==3) THEN
          LC(rEnd+1,i)=LC(rEnd+1,i)+0.5*(((C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i) &
                                        +(C(rEnd,i)-C(rEnd+1,i))/Tau(rEnd+1,i)/dZ);
          LC(rEnd,i)=LC(rEnd,i)+0.5*(C(rEnd+1,i)-C(rEnd,i))/TauB(2,i);
        END IF
    END DO
    CALL sElement(C0(:,:,rk),h(rk)*dt)
  END SELECT
 END DO
ELSE  ! use Thomas algorithm for nonisothermal and the last step of isothermal
  DO rk=1,3
    C0(:,:,rk)=C; T0(:,rk)=T;
   DO i=1,ngc
    DO j=rStart,rEnd ! construct A first
      A(j,1,i,rk)=     -0.5*h(rk)*dt/Tau(j,i)/(dZ**2); ! left diagonal
      A(j,2,i,rk)=1.0D0+1.0*h(rk)*dt/Tau(j,i)/(dZ**2); ! main diagonal
      A(j,3,i,rk)=     -0.5*h(rk)*dt/Tau(j,i)/(dZ**2); ! right diagonal
    END DO
    SELECT CASE (CBC(1))
     CASE (1) ! Dirichlet BC
     A(rStart  ,1,i,rk)=0.0D0;
    CASE (2) ! Neumann BC
     A(rStart  ,3,i,rk)=A(rStart,3,i,rk)+A(rStart,1,i,rk);
     !A(rStart  ,1,i,rk)=0.0D0;
    CASE (3) ! Danckwert's BC
     A(rStart-1,1,i,rk)=     -0.5*h(rk)*dt/Tau(rStart-2,i);
     !A(rStart-1,2,i,rk)=1.0D0+0.5*h(rk)*dt/Tau(rStart-2,i)+0.5*h(rk)*dt/Tau(rStart-1,i)/dZ;
     !A(rStart-1,3,i,rk)=                                  -0.5*h(rk)*dt/Tau(rStart-1,i)/dZ;
     A(rStart-1,2,i,rk)=1.0D0+0.5*h(rk)*dt/Tau(rStart-2,i)+0.5*h(rk)*dt/Tau(rStart-1,i);
     A(rStart-1,3,i,rk)=                                   -0.5*h(rk)*dt/Tau(rStart-1,i);
     A(rStart,1,i,rk)=A(rStart,1,i,rk)-0.5*h(rk)*dt/TauB(1,i);
     A(rStart,2,i,rk)=A(rStart,2,i,rk)+0.5*h(rk)*dt/TauB(1,i);
    END SELECT
    SELECT CASE (CBC(2))
     CASE (1) ! Dirichlet BC
```

Table E.13 Continued

```
    A(rEnd  ,3,i,rk)=0.0D0;
  CASE (2) ! Neumann BC
    A(rEnd  ,1,i,rk)=A(rStart,1,i,rk)+A(rStart,3,i,rk);
    !A(rEnd  ,3,i,rk)=0.0D0;
  CASE (3) ! Danckwert's BC
    A(rEnd,2,i,rk)=A(rEnd,2,i,rk)+0.5*h(rk)*dt/TauB(2,i);
    A(rEnd,3,i,rk)=A(rEnd,3,i,rk)-0.5*h(rk)*dt/TauB(2,i);
    A(rEnd+1,1,i,rk)=                                -0.5*h(rk)*dt/Tau(rEnd+1,i);
    A(rEnd+1,2,i,rk)=1.0D0+0.5*h(rk)*dt/Tau(rEnd+2,i)+0.5*h(rk)*dt/Tau(rEnd+1,i);
    !A(rEnd+1,1,i,rk)=                               -0.5*h(rk)*dt/Tau(rEnd+1,i)/dZ;
    !A(rEnd+1,2,i,rk)=1.0D0+0.5*h(rk)*dt/Tau(rEnd+2,i)+0.5*h(rk)*dt/Tau(rEnd+1,i)/dZ;
    A(rEnd+1,3,i,rk)=    -0.5*h(rk)*dt/Tau(rEnd+2,i);
  END SELECT
END DO
  DO i=gStart,min(gEnd,iEnd) ! construct right hand side (store data from previous step for gas)
    DO j=rStart,rEnd
    LC(j,i)=0.5*(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2);
    B(j,i)=C(j,i)+h(rk)*dt*(LC(j,i) + Zeta(rk)*NC(j,i));
    END DO
  SELECT CASE (CBC(1))
    CASE (1) ! Dirichlet
      B(rStart,i)=B(rStart,i)-A(rStart,1,i,rk)*C(rStart-1,i);
    CASE (2) ! Neumann (Do nothing)
    CASE (3) ! Danckserts
    !LC(rStart-1,i)=0.5*(((C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart  ,i)-C(rStart-1,i))/Tau(rStart-1,i))/dZ);
        !B(rStart-1,i)=C(rStart-1,i)+h(rk)*dt*LC(rStart-1,i);
        LC(rStart-1,i)=0.5*(((C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i)+(C(rStart  ,i)-C(rStart-1,i))/Tau(rStart-1,i));
        B(rStart-1,i)=C(rStart-1,i)+h(rk)*dt*LC(rStart-1,i);
        B(rStart-1,i)=B(rStart-1,i)-A(rStart-1,1,i,rk)*C(rStart-1,i);
        LC(rStart,i)=LC(rStart,i)+0.5*(C(rStart-1,i)-C(rStart,i))/TauB(1,i);
        B(rStart,i)=B(rStart,i)+0.5*h(rk)*dt*(C(rStart-1,i)-C(rStart,i))/TauB(1,i);
  END SELECT
  SELECT CASE (CBC(2))
    CASE (1) ! Dirichlet
      B(rEnd,i)=B(rEnd,i)-A(rEnd,3,i,rk)*C(rEnd+1,i);
    CASE (2) ! Neumann (Do nothing)
    CASE (3) ! Danckserts
        !LC(rEnd+1,i)=0.5*(((C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd  ,i)-C(rEnd+1,i))/Tau(rEnd+1,i)/dZ);
        !B(rEnd+1,i)=C(rEnd+1,i)+h(rk)*dt*LC(rEnd+1,i);
        LC(rEnd+1,i)=0.5*(((C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i)+(C(rEnd  ,i)-C(rEnd+1,i))/Tau(rEnd+1,i));
        B(rEnd+1,i)=C(rEnd+1,i)+h(rk)*dt*LC(rEnd+1,i);
        B(rEnd+1,i)=B(rEnd+1,i)-A(rEnd+1,3,i,rk)*C(rEnd+1,i);
        LC(rEnd,i)=LC(rEnd,i)+0.5*(C(rEnd+1,i)-C(rEnd,i))/TauB(2,i);
        B(rEnd,i)=B(rEnd,i)+0.5*h(rk)*dt*(C(rEnd+1,i)-C(rEnd,i))/TauB(2,i);
  END SELECT
  END DO
```

Table E.13 Continued

```
    DO i=sStart,min(cEnd,iEnd) ! store data from previous step for solid
      DO j=rStart,rEnd
        B(j,i)=C(j,i)+h(rk)*dt*Zeta(rk)*NC(j,i);
    END DO
    END DO
    CALL sReaction ! now we can compute the reaction part and store in NC
    DO i=gStart,min(gEnd,iEnd) ! add the reaction part for gas at current step
      DO j=rStart,rEnd
      B(j,i)=B(j,i)+dt*h(rk)*Beta(rk)*NC(j,i);
    END DO
    END DO
    DO i=gStart,min(gEnd,iEnd) ! solve by Thomas algorithm
      DO j=nmStart,nmEnd
        A(j+1,1,i,rk)=A(j+1,1,i,rk)/A(j,2,i,rk);
        A(j+1,2,i,rk)=A(j+1,2,i,rk)-A(j,3,i,rk)*A(j+1,1,i,rk);
      B(j,i)=B(j,i)-A(j,1,i,rk)*B(j-1,i);
      END DO
    C(nmEnd,i)=B(nmEnd,i)/A(nmEnd,2,i,rk);
    DO j=(nmEnd-1),nmStart,-1
      C(j,i)=(B(j,i)-A(j,3,i,rk)*C(j+1,i))/A(j,2,i,rk);
    END DO
  END DO
    DO i=sStart,min(cEnd,iEnd) ! solve the solid component
      DO j=rStart,rEnd
        C(j,i)=B(j,i)+h(rk)*dt*NC(j,i);
      END DO
    END DO
    SELECT CASE (ecm) ! execute ecm algorithm
    CASE (0) ! non-ecm
    CASE (1)
      CALL sElement(C0(:,:,rk),h(rk)*dt)
    CASE (2)
      DO i=gStart,min(gEnd,iEnd) ! construct linear part for current time step
        IF (CBC(1)==3) THEN
            LC(rStart-1,i)=LC(rStart-1,i)+0.5*(((C(rStart-2,i)-C(rStart-1,i))/Tau(rStart-2,i) &
                                          +(C(rStart,i)-C(rStart-1,i))/Tau(rStart-1,i));
          LC(rStart,i)=LC(rStart,i)+0.5*(C(rStart-1,i)-C(rStart,i))/TauB(1,i);
        END IF
      DO j=rStart,rEnd
        LC(j,i)=LC(j,i)+0.5*(C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2);
      END DO
          IF (CBC(2)==3) THEN
            LC(rEnd+1,i)=LC(rEnd+1,i)+0.5*(((C(rEnd+2,i)-C(rEnd+1,i))/Tau(rEnd+2,i) &
                                          +(C(rEnd,i)-C(rEnd+1,i))/Tau(rEnd+1,i)/dZ);
          LC(rEnd,i)=LC(rEnd,i)+0.5*(C(rEnd+1,i)-C(rEnd,i))/TauB(2,i);
          END IF
```

Table E.13 Continued

```
      END DO
      CALL sElement(C0(:,:,rK),h(rk)*dt)
    END SELECT
  END DO
END IF
maxDC=max(MAXVAL(abs(C(nmStart:nmEnd,gStart:gEnd)-C0(nmStart:nmEnd,gStart:gEnd,1))), &
          MAXVAL(abs(C(rStart:rEnd  ,sStart:cEnd)-C0(rStart:rEnd  ,sStart:cEnd,1))));  ! maximum concentration change
minC=MINVAL(C); ! minimum concentration
IF (thermalType==3) THEN ! solve energy equation
  CALL sEnergy
  maxDT=MAXVAL(abs(T(nmStart:nmEnd)-T0(nmStart:nmEnd,1))); ! maximum change in temperature
  minT=MINVAL(T);                                          ! minimum temperature
END IF
IF (maxDC > tolMaxDC .OR. minC < tolMinC .OR. maxDT > tolMaxDT .OR. minT < tolMinT) THEN ! adaptive time control ---------------|
  tooStiff=.TRUE.;
  C=C0(:,:,1); T=T0(:,1); tRun=tRun0;
  IF (dtLevel == 1) THEN ! main loop
    tRun0=min(tRun+dt,tf); ! store in tRun0 temporary
    dtLevel=max(dtLevelOld-1,1); ! set the dtLevel to dtLevelOld-1, so the next adaptive loop starts the same as last dtLevel
  DO WHILE (tRun < tRun0)
    IF (tooStiff) THEN
      dtLevel=dtLevel+1; ! increase the number of sub-loops if the previous sub-loop returns stiff signal
      tooStiff=.FALSE.;
    ELSE
      dtLevel=max(dtLevel-1,1); ! decrease the number of sub-loops if the previous sub-loop didn't return stiff signal
      tooStiff=.FALSE.;
    END IF
    CALL sCNRK3(min(tRun+((1.0D0/nst)**max((dtLevel-2),0))*dt,tRun0),nst) ! adaptive time integration
  END DO
    dtLevelOld=dtLevel; ! save the last dtLevel, so the next adaptive loop will start at the same level
  dtLevel=1; ! important!! pull back to dtLevel 1 if the DO WHILE condition satisfied
ELSEIF (dtLevel > 1) THEN ! sub loop
  RETURN
ELSE
  WRITE(6,*) "Error: dtLevel < 0."
  PAUSE
  STOP
END IF
ELSE ! passed
  tooStiff=.FALSE.;
  IF (flagLastStep == 0) THEN ! syncronize the last delta t in case of round off error
    tRun=tf;
  ELSE
    tRun=tRun+dt;
  END IF
END IF
```

Table E.13 Continued

```
IF (thermalType /= 1) THEN ! update variables and parameters ------------------------------------------------------------------------------|
    IF (thermalType == 2) THEN; CALL sTemperature; END IF ! temperature ramp
    CALL sDiffusion
    CALL sTau
    CALL sRateConstant
END IF
CALL sBoundary
CALL sReaction ! compute the nonlinear reaction part and store in NC (alpha*S*R for gas and S*R for solid)
IF (dtLevel==1 .AND. .NOT.switchMethod) THEN; CALL sSave("Dynamic"); END IF ! save date
IF (tRun >= tRuntime) THEN; CALL sRuntime(6); END IF ! write runtime information on screen
IF (tRun>tf) THEN
  WRITE(6,*) "Warning: Current time is greater than final time. tRun > tf at dtLevel ", dtLevel, " and tRun-tf=", tRun-tf, ".";
END IF
IF (flagLastStep == 0) THEN; EXIT; END IF ! exit this loop
END DO
IF (ALLOCATED(C0  )) THEN; DEALLOCATE(C0  ); END IF ! deallocate array ------------------------------------------------------------------|
IF (ALLOCATED(T0  )) THEN; DEALLOCATE(T0  ); END IF
IF (ALLOCATED(A   )) THEN; DEALLOCATE(A   ); END IF
IF (ALLOCATED(B   )) THEN; DEALLOCATE(B   ); END IF
END SUBROUTINE sCNRK3
```

Table E.14 sIE.f90

```
!-----------------------------------------------------------------------
! Subroutine sIE
! Description: Implicit Euler method.
! Created by Hsu-Wen Hsiao on 4/6/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   tf: final time
!   nt: number of time step
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sIE(tf,nt)
USE mDynamic
USE mReaction
USE mElement
USE mUserDefine
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: tf
INTEGER, INTENT(IN) :: nt
DOUBLE PRECISION, PARAMETER :: zeroC=EPSILON(zeroC) ! concentration less than thiswill be set to zero (eliminate round off error)
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: C0 ! pre-stored concentration of previous time step
DOUBLE PRECISION, DIMENSION(:  ), ALLOCATABLE :: T0 ! pre-stored temperature of previous time step
DOUBLE PRECISION :: tRun0                           ! pre-stored time of previous time step
DOUBLE PRECISION :: dt                              ! time step size
DOUBLE PRECISION :: maxDC, minC, maxDT, minT        ! parameter for adaptive time control
INTEGER, DIMENSION(:,:), ALLOCATABLE :: CIM, CIV    ! component index matrix and vector
INTEGER :: nciv, ndiag, mdiag                       ! dimension of CIV, number of diagonal, position of main diagonal
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: A, A0 ! compact form of Jacobian matrix
DOUBLE PRECISION, DIMENSION(:  ), ALLOCATABLE :: B  ! right hand side of Jacobian matrix
DOUBLE PRECISION, PARAMETER :: tolNewton=1.0E-10    ! tolerance of Newton's method          (1.0E-10 is good)
INTEGER, PARAMETER :: maxIteration=13; ! max number of iteration for Newton's method (13 is good for 1.0E-10 accuracy, important!!)
INTEGER :: it , i, j, k, l, flagLastStep, nIteration  ! dummy indices
!CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2 !
IF (dtLevel > maxdtLevel) THEN ! initialize array for adaptive time control ----------------------------------------------------------
  WRITE(6,*) "Warning ( dtLevel > 200 ): Unstable numerical method!! Please use smaller tfRun or change numerical method."
END IF
maxDC=0.0D0; ! maximum concentration change
maxDT=0.0D0; ! maximum temperature change
minC=MINVAL(C);  ! minimum concentration
minT=MINVAL(T);  ! minimum temperature
flagLastStep=nt; ! flag to syncronize the last step
tooStiff=.FALSE.;! signal of stiffness for the sub-loop of adaptive time control
ALLOCATE(C0(ngz,ntc),T0(ngz)) ! initialize array to save previous step ----------------------------------------------------------
DO i=1,ntc; DO j=1,ngz; C0(j,i)=0.0D0; END DO; END DO
DO j=1,ngz; T0(j)=0.0D0; END DO
tRun0=0.0D0;
dt=(tf-tRun)/(1.0D0*nt); ! delta t at this dtLevel ! initialize parameters for IE method ----------------------------------------
```

Table E.14 Continued

```
nciv=(nmEnd-nmStart+1)*ngc+(rEnd-rStart+1)*ntsc ! dimension of CIV, total number of C need to be solved numerically
ALLOCATE(CIM(ngz,ntc),CIV(2,nciv)) ! initialize component index matrix and vector
k=0;
DO j=1,ngz
  DO i=1,ntc
    IF (i>=gStart .AND. i<= gEnd) THEN ! gas component
      IF (j >= nmStart .AND. j <= nmEnd) THEN ! grid need to be solved numerically
        k=k+1; ! index of CIV
        CIM(j,i)=k; ! store column index of CIV
        CIV(1,k)=j; ! store row index of C
        CIV(2,k)=i; ! store column index of C
      ELSE
        CIM(j,i)=0;
      END IF
    ELSEIF (i>=sStart .AND. i<=cEnd) THEN ! solid component
      IF (j>= rStart .AND. j<=rEnd) THEN ! grid of reactor
        k=k+1; ! index of CIV
        CIM(j,i)=k; ! store column index of CIV
        CIV(1,k)=j; ! store row index of C
        CIV(2,k)=i; ! store column index of C
      ELSE
        CIM(j,i)=0; ! set to zero
      END IF
    END IF
  END DO
END DO
SELECT CASE (ecm)
  CASE (0) ! no ecm
    ndiag=2*(ntc+1)-1;   ! number of banded diagonal of Jacobian matrix
    mdiag=ntc+1;         ! column loacation of main diaggonal of Jacobian matrix in the compact form
  CASE (1) ! catalytic site only (element balance equation only relates to solid components)
    ndiag=2*(ntc+1)-1;   ! number of banded diagonal of Jacobian matrix
    mdiag=ntc+1;         ! column loacation of main diaggonal of Jacobian matrix in the compact form
  CASE (2) ! ecm
    ndiag=(2*ntc-1)+ntc+1; ! number of banded diagonal of Jacobian matrix
    mdiag=(2*ntc-1)+1;     ! column loacation of main diaggonal of Jacobian matrix in the compact form
END SELECT
ALLOCATE(A(nciv,ndiag),A0(nciv,ndiag),B(nciv)) ! initialize compact form of Jacobian matrix and vector on the right hand side
DO i=1,ndiag; DO j=1,nciv; A(j,i)=0.0D0; A0(j,i)=0.0D0; END DO; END DO
DO j=1,nciv; B(j)=0.0D0; END DO
DO it=1,nt ! time ingegration ---------------------------------------------------------------------------------------------|
  IF (it<=1) THEN ! construct the template A0 of Jacobian matrix
    DO j=rStart,rEnd ! construct the compact form of banded Jacobian matrix for gas component -------------------------------------|
      DO i=gStart,min(gEnd,iEnd) ! non-ecm algorithm, add the diffusion term to prestored reacction term
        A0(CIM(j,i),mdiag-ntc)=           -dt/Tau(j,i)/(dZ**2); ! diffusion to the left grid
        A0(CIM(j,i),mdiag    )=1.0D0+2.0D0*dt/Tau(j,i)/(dZ**2); ! diffusion and reaction on same place
```

Table E.14 Continued

```
    A0(CIM(j,i),mdiag+ntc)=         -dt/Tau(j,i)/(dZ**2); ! diffusion to the right grid
   END DO
  DO i=max(gStart,dStart),gEnd ! ecm algorithm, construct Jacobian by ecm equation of gas component
    A0(CIM(j,i),mdiag-ntc)=          -dt/Tau(j,i)/(dZ**2); ! diffusion to the left grid
    A0(CIM(j,i),mdiag    )=1.0D0+2.0D0*dt/Tau(j,i)/(dZ**2); ! diffusion only on this gird
    A0(CIM(j,i),mdiag+ntc)=         -dt/Tau(j,i)/(dZ**2); ! diffusion to the right grid
    DO l=MDiMIYI(i-nic),MDiMiYI(i-nic+1)-1
      A0(CIM(j,i),mdiag-ntc-(i-MDiMIYL(l)))=MDiMIY(l)*(          -dt/Tau(j,MDiMIYL(l))/(dZ**2));
      A0(CIM(j,i),mdiag    -(i-MDiMIYL(l)))=MDiMIY(l)*(1.0D0+2.2D0*dt/Tau(j,MDiMIYL(l))/(dZ**2));
      A0(CIM(j,i),mdiag+ntc-(i-MDiMIYL(l)))=MDiMIY(l)*(          -dt/Tau(j,MDiMIYL(l))/(dZ**2));
    END DO
  END DO
 END DO
 DO j=rStart,rEnd ! construct the compact form of banded Jacobian matrix for solid component ----------------------------------------|
   DO i=sStart,min(cEnd,iEnd)  ! non-ecm algorithm, process the reaction term on solid surface
    A0(CIM(j,i),mdiag)=1.0D0;
   END DO
   DO i=max(sStart,dStart),cEnd ! ecm algorithm,construct Jacobian by ecm equation of solid component
    A0(CIM(j,i),mdiag)=+1.0D0;
    DO l=MDiMIYI(i-nic),MDiMiYI(i-nic+1)-1
     IF     (MDiMIYL(l) >= gStart .AND. MDiMIYL(l) <=gEnd) THEN
       A0(CIM(j,i),mdiag-ntc-(i-MDiMIYL(l)))=MDiMIY(l)/alpha*(          -dt/Tau(j,MDiMIYL(l))/(dZ**2));
       A0(CIM(j,i),mdiag    -(i-MDiMIYL(l)))=MDiMIY(l)/alpha*(1.0D0+2.2D0*dt/Tau(j,MDiMIYL(l))/(dZ**2));
       A0(CIM(j,i),mdiag-ntc-(i-MDiMIYL(l)))=MDiMIY(l)/alpha*(          -dt/Tau(j,MDiMIYL(l))/(dZ**2));
     ELSEIF (MDiMIYL(l) >= sStart .AND. MDiMIYL(l) <= cEnd) THEN
       A0(CIM(j,i),mdiag    -(i-MDiMIYL(l)))=MDiMIY(l);
     END IF
    END DO
   END DO
  END DO
  SELECT CASE (CBC(1)) ! left boundary
   CASE (1) ! Dirichlet on left
    DO i=gStart,gEnd !The gas concentration on the boundary is constant, so the derivative in Jacobian matrix is zero.
     A0(CIM(rStart,i),mdiag-ntc)=0.0D0;
    END DO
   CASE (2) ! Neumann on left
    DO i=gStart,min(gEnd,iEnd)
     A0(CIM(rStart,i),mdiag+ntc)=A0(CIM(rStart,i),mdiag+ntc)+A0(CIM(rStart,i),mdiag-ntc); ! add left to right
     A0(CIM(rStart,i),mdiag-ntc)=0.0D0; ! clear left
    END DO
    DO i=dStart,cEnd ! if use ecm algorithm
     A0(CIM(rStart,i),mdiag+ntc)=A0(CIM(rStart,i),mdiag+ntc)+A0(CIM(rStart,i),mdiag-ntc);
     A0(CIM(rStart,i),mdiag-ntc)=0.0D0;
    END DO
   CASE (3) ! Danckwerts on left (No ECM for boundary chamber)
    j=rStart-1;
```

Table E.14 Continued

```
    DO i=gStart,min(gEnd,iEnd)
      A0(CIM(j,i),mdiag-ntc)=     -dt/Tau(j-1,i);                ! advection mixing
      A0(CIM(j,i),mdiag    )=1.0D0+dt/Tau(j-1,i)+dt/Tau(j,i)/dZ; ! balanced mixing
      A0(CIM(j,i),mdiag+ntc)=     -dt/Tau(j,i)/dZ; ! diffusion mixing
    END DO
  END SELECT
  SELECT CASE (CBC(2)) ! right boundary
    CASE (1) ! Dirichlet on right
      DO i=gStart,gEnd !The gas concentration on the boundary is constant, so the derivative in Jacobian matrix is zero.
        A0(CIM(rEnd,i),mdiag+ntc)=0.0D0;
      END DO
    CASE (2) ! Neumann on right
      DO i=gStart,min(gEnd,iEnd) ! this loop takes care of the boundary condition
        A0(CIM(rEnd  ,i),mdiag-ntc)=A0(CIM(rEnd  ,i),mdiag-ntc)+A0(CIM(rEnd  ,i),mdiag+ntc); ! add right to left
        A0(CIM(rEnd  ,i),mdiag+ntc)=0.0D0; ! clear right
      END DO
      DO i=dStart,cEnd ! if use ecm algorithm
        A0(CIM(rEnd  ,i),mdiag-ntc)=A0(CIM(rEnd  ,i),mdiag-ntc)+A0(CIM(rEnd  ,i),mdiag+ntc);
        A0(CIM(rEnd  ,i),mdiag+ntc)=0.0D0;
      END DO
    CASE (3) ! Danckwerts on right (No ECM for boundary chamber)
      j=rEnd+1;
      DO i=gStart,min(gEnd,iEnd) ! this loop takes care of the boundary condition
        A0(CIM(j,i),mdiag-ntc)=     -dt/Tau(j,i)/dZ; ! diffusion mixing
        A0(CIM(j,i),mdiag    )=1.0D0+dt/Tau(j+1,i)+dt/Tau(j,i)/dZ; ! balanced mixing
        A0(CIM(j,i),mdiag+ntc)=     -dt/Tau(j+1,i);                ! advection mixing
      END DO
  END SELECT
END IF
C0=C; T0=T; tRun0=tRun ! store data of previous time step
IF (tRun+dt >= tf .OR. it==nt) THEN ! syncronize the last delta t in case of round off error ----------------------------------------|
  flagLastStep=0;
  dt=tf-tRun;
  tRun=tf;
ELSE
  flagLastStep=flagLastStep-1;
  tRun=tRun+dt;
END IF
IF (thermalType /= 1) THEN ! non-isothermal ! calculate the parameter at current time step ----------------------------------------|
  IF (thermalType == 2) THEN; CALL sTemperature; END IF
  CALL sDiffusion
  CALL sTau
  CALL sRateConstant
END IF
CALL sBoundary
CALL sReaction
```

Table E.14 Continued

```
CALL sDRxnDc
nIteration=0; ! initialize Newton's method ------------------------------------------------------------------------------------------------|
DO WHILE (nIteration >= 0 .AND. nIteration <= maxIteration)
  nIteration=nIteration+1;
  A=A0; ! use template
  DO k=1,nciv; B(k)=0.0D0; END DO ! clear B
  DO j=rStart,rEnd ! construct reaction part of Jacobian matrix ----------------------------------------------------------------------------|
    DO i=gStart,min(gEnd,iEnd) ! first derivate of reaction rate to gas component, dR/dC
      DO k=SJRI(i),SJRI(i+1)-1
        DO l=SJYI(i),SJYI(i+1)-1
          A(CIM(j,i),mdiag-1+SJRL(k)-i+1)=A(CIM(j,i),mdiag-1+SJRL(k)-i+1)-dt*alpha*SJY(l)*JRY(SJYCL(l),j); ! store S*(dR/dC) in A
        END DO
      END DO
      DO k=urStart,urEnd ! the first derivative of user-defined rate to gas component
        IF (S(i,k) /= 0 .AND. nzUDRD(k-near,i)) THEN
           A(CIM(j,i),mdiag-1+SJRL(k)-i+1)=A(CIM(j,i),mdiag-1+SJRL(k)-i+1)-dt*alpha*S(i,k)*JUR(k-near,i,j);
        END IF
      END DO
    END DO
    DO i=sStart,min(cEnd,iEnd) ! first derivative of rate to solid component, dR/dC
      DO k=SJRI(i),SJRI(i+1)-1
        DO l=SJYI(i),SJYI(i+1)-1
          A(CIM(j,i),mdiag-1+SJRL(k)-i+1)=A(CIM(j,i),mdiag-1+SJRL(k)-i+1)-dt*SJY(l)*JRY(SJYCL(l),j); ! store S*(dR/dC) in A
        END DO
      END DO
      DO k=urStart,urEnd ! the first derivative of user-defined rate to gas component
        IF (S(i,k) /= 0 .AND. nzUDRD(k-near,i)) THEN
           A(CIM(j,i),mdiag-1+SJRL(k)-i+1)=A(CIM(j,i),mdiag-1+SJRL(k)-i+1)-dt*S(i,k)*JUR(k-near,i,j);
        END IF
      END DO
      END DO
  END DO
  DO j=rStart,rEnd ! construct right hand side for Jacobian matrix, (B = -F) ----------------------------------------------------|
    DO i=gStart,min(gEnd,iEnd) ! non-ecm, function of diffusion reaction
      B(CIM(j,i))=C0(j,i)-C(j,i)+dt*((C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2)+NC(j,i));
    END DO
    DO i=max(gStart,dStart),gEnd ! ecm algorithm, for dependent gas component
      B(CIM(j,i))=       -C(j,i)+dt*( C(j-1,i)-2*C(j,i)+C(j+1,i) )/Tau(j,i)/dZ**2
      DO k=MDiMIYI(i-nic),MDiMIYI(i-nic+1)-1 ! gas only, the equation for solids are ignored
        B(CIM(j,i))=B(CIM(j,i))-MDiMIY(k)* &
                  (C(j,MDiMIYL(k))-dt*(C(j-1,MDiMIYL(k))-2*C(j,MDiMIYL(k))+C(j+1,MDiMIYL(k)))/Tau(j,MDiMIYL(k))/dZ**2);
      END DO
      DO k=MDiMYI( i-nic),MDiMYI( i-nic+1)-1
        IF     (MDiMYL(k) >= gStart .AND. MDiMYL(k)<= gEnd) THEN
          B(CIM(j,i))=B(CIM(j,i))+MDiMY(k)*C(j,MDiMYL(k));
        ELSEIF (MDiMYL(k) >= sStart .AND. MDiMYL(k)<= cEnd) THEN
```

Table E.14 Continued

```
    B(CIM(j,i))=B(CIM(j,i))+MDiMY(k)*alpha*C(j,MDiMYL(k));
   END IF
  END DO
 END DO
 DO i=sStart,min(cEnd,iEnd) ! non-ecm, right hand side for solid component B = -Fj,i (reaction only)
  B(CIM(j,i))=C0(j,i)-C(j,i)+dt*(+NC(j,i));
 END DO
 DO i=max(sStart,dStart),cEnd ! ecm, right hand side of ecm equation for soid component
  B(CIM(j,i))=-C(j,i);
  DO k=MDiMIYI(i-nic),MDiMIYI(i-nic+1)-1
   IF     (MDiMIYL(k) >= gStart .AND. MDiMIYL(k) <= gEnd) THEN
    B(CIM(j,i))=B(CIM(j,i))-MDiMIY(k)/alpha* &
          (C(j,MDiMIYL(k))-dt*(C(j-1,MDiMIYL(k))-2*C(j,MDiMIYL(k))+C(j+1,MDiMIYL(k)))/Tau(j,MDiMIYL(k))/dZ**2);
   ELSEIF (MDiMIYL(k) >= gStart .AND. MDiMIYL(k) <= gEnd) THEN
    B(CIM(j,i))=B(CIM(j,i))-MDiMY(k)*C(j,MDiMIYL(k));
   END IF
  END DO
  DO k=MDiMYI(i-nic),MDiMYI(i-nic+1)-1
   IF     (MDiMYL(k) >= gStart .AND. MDiMYL(k) <= gEnd) THEN
    B(CIM(j,i))=B(CIM(j,i))+MDiMY(k)/alpha*C(j,MDiMYL(k));
   ELSEIF (MDiMYL(k) >= sStart .AND. MDiMYL(k) <= cEnd) THEN
    B(CIM(j,i))=B(CIM(j,i))+MDiMY(k)*C(j,MDiMYL(k));
   END IF
  END DO
 END DO
 SELECT CASE (CBC(1)) ! left boundary
  CASE (1) ! Dirichlet on left
  CASE (2) ! Neumann on left
  CASE (3) ! Danckwerts on left
   j=rStart-1;
   DO i=gStart,min(gEnd,iEnd)
    B(CIM(j,i))=C0(j,i)-C(j,i)+dt*(-C(j,i)+C(j+1,i))/Tau(j,i)/dZ+(C(j-1,i)-C(j,i))/Tau(j-1,i)); ! compute RHS of gas B=-F
   END DO
 END SELECT
 SELECT CASE (CBC(2)) ! right boundary
  CASE (1) ! Dirichlet on right
  CASE (2) ! Neumann on right
  CASE (3) ! Danckwerts on right
   j=rEnd+1;
   DO i=gStart,min(gEnd,iEnd) ! this loop takes care of the boundary condition
    B(CIM(j,i))=C0(j,i)-C(j,i)+dt*((C(j-1,i)-C(j,i))/Tau(j,i)/dZ+(-C(j,i)+C(j+1,i))/Tau(j+1,i)); ! compute RHS of gas B=-F
   END DO
 END SELECT
 SELECT CASE (ecm)
 CASE (0)
```

Table E.14 Continued

```
   CALL sGaussBand(A,B,nciv,2*ntc+1) ! solve (J^-1)*(-F) by Gauss elemination for banded matrix, B <-- (A^-1)*B
  CASE (1)
   CALL sGaussBand(A,B,nciv,2*ntc+1) ! solve (J^-1)*(-F) by Gauss elemination for banded matrix, B <-- (A^-1)*B
  CASE (2)
   CALL sGaussLBand(A,b,nciv,2*ntc-1,ntc,mdiag,ndiag) ! solve (J^-1)*(-F) by Gauss elemination for left banded matrix
  END SELECT
  DO k=1,nciv
   C(CIV(1,k),CIV(2,k))=C(CIV(1,k),CIV(2,k))+B(k); ! update concentration
  END DO
  IF (thermalType==3) THEN ! solve energy equation
   CALL sEnergy
   maxDT=MAXVAL(abs(T(nmStart:nmEnd)-T0(nmStart:nmEnd))); ! maximum temperature change
   minT=MINVAL(T);                                        ! minimum temperature
  END IF
  IF (thermalType /= 1) THEN ! non-isothermal ! calculate the parameter at current time step ----------------------------------------|
   IF (thermalType == 2) THEN; CALL sTemperature; END IF
   CALL sDiffusion
   CALL sTau
   CALL sRateConstant
  END IF
  CALL sBoundary
  CALL sReaction
  CALL sDRxnDc
  IF (MAXVAL(abs(B)) >= 0.0D0 .AND. MAXVAL(abs(B)) < tolNewton) THEN ! exit loop if the Newton's method converges
   EXIT
  ELSEIF (nIteration >= maxIteration) THEN ! exit loop with stiff signal if didn't converge
   tooStiff=.TRUE.;
   EXIT
  END IF
 END DO
 maxDC=max(MAXVAL(abs(C(nmStart:nmEnd,gStart:gEnd)-C0(nmStart:nmEnd,gStart:gEnd))), & ! maximum concentration change from
           MAXVAL(abs(C(rStart:rEnd,sStart:cEnd)-C0(rStart:rEnd,sStart:cEnd)))));     ! numerical computation
 minC=MINVAL(C); ! minimum concentration
 IF (minC < 0.0D0 .AND. abs(minC) < zeroC) THEN ! set the negative round off error to zero
  DO i=1,ntc
   DO j=1,ngz
    IF (C(j,i) < 0.0D0 .AND. abs(C(j,i)) < zeroC) THEN
     C(j,i)=0.0D0;
    END IF
   END DO
  END DO
  minC=MINVAL(C); ! recalculate minimum concentration
 END IF
 IF (maxDC > tolMaxDC .OR. minC < tolMinC .OR. maxDT > tolMaxDT .OR. minT < tolMinT .OR. tooStiff) THEN ! adaptive time control --|
   tooStiff=.TRUE.;
  C=C0; T=T0; tRun=tRun0;
```

Table E.14 Continued

```
    IF (switchMethod) THEN; RETURN; END IF
  IF (dtLevel == 1) THEN ! main loop
    tRun0=min(tRun+dt,tf); ! store in tRun0 temporary
      dtLevel=max(dtLevelOld-1,1); ! set the dtLevel to dtLevelOld-1, so the next adaptive loop starts the same as last dtLevel
      DO WHILE (tRun < tRun0)
      IF (tooStiff) THEN
        dtLevel=dtLevel+1; ! increase the number of sub-loops if the previous sub-loop returns stiff signal
        tooStiff=.FALSE.;
      ELSE
        dtLevel=max(dtLevel-1,1); ! decrease the number of sub-loops if the previous sub-loop didn't return stiff signal
        tooStiff=.FALSE.;
      END IF
      CALL sIE(min(tRun+((1.0D0/nst)**max((dtLevel-2),0))*dt,tRun0),nst) ! adaptive time integration
    END DO
      dtLevelOld=dtLevel; ! save the last dtLevel, so the next adaptive loop will start at the same level
    dtLevel=1; ! important!! pull back to dtLevel 1 if the DO WHILE condition satisfied
  ELSEIF (dtLevel > 1) THEN ! sub loop
    RETURN
  ELSE
    WRITE(6,*) "Error: dtLevel < 0."
    PAUSE
    STOP
  END IF
ELSE ! passed
  tooStiff=.FALSE.;
END IF
IF (dtLevel==1 .AND. .NOT.switchMethod) THEN; CALL sSave("Dynamic"); END IF ! save date
IF (tRun >= tRuntime) THEN; CALL sRuntime(6); END IF ! write runtime information on screen
IF (tRun>tf) THEN
  WRITE(6,*) "Warning: Current time is greater than final time. tRun > tf at dtLevel ", dtLevel, " and tRun-tf=", tRun-tf, ".";
END IF
 IF (flagLastStep == 0) THEN; EXIT; END IF ! exit this loop
END DO
IF (ALLOCATED(C0 )) THEN; DEALLOCATE(C0 ); END IF ! deallocate array ----------------------------------------------------------------------|
IF (ALLOCATED(T0 )) THEN; DEALLOCATE(T0 ); END IF
IF (ALLOCATED(CIM)) THEN; DEALLOCATE(CIM); END IF
IF (ALLOCATED(CIV)) THEN; DEALLOCATE(CIV); END IF
IF (ALLOCATED(A  )) THEN; DEALLOCATE(A  ); END IF
IF (ALLOCATED(A0 )) THEN; DEALLOCATE(A0 ); END IF
IF (ALLOCATED(B  )) THEN; DEALLOCATE(B  ); END IF
END SUBROUTINE sIE
```

Table E.15 sBDF2.f90

```
!-----------------------------------------------------------------------
! Subroutine sBDF2
! Description: Second order backward differential formula.
! Created by Hsu-Wen Hsiao on 8/15/10.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   tf: final time
!   nt: number of time step
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sBDF2(tf,nt)
USE mDynamic
USE mReaction
USE mElement
USE mUserDefine
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: tf
INTEGER, INTENT(IN) :: nt
DOUBLE PRECISION, PARAMETER :: zeroC=EPSILON(zeroC) ! concentration less than thiswill be set to zero (eliminate round off error)
DOUBLE PRECISION, DIMENSION(:,:,:), ALLOCATABLE :: C0 ! pre-stored concentration of previous time step
DOUBLE PRECISION, DIMENSION(:,:  ), ALLOCATABLE :: T0 ! pre-stored temperature of previous time step
DOUBLE PRECISION :: tRun0                              ! pre-stored time of previous time step
DOUBLE PRECISION :: dt                                 ! time step size
DOUBLE PRECISION, PARAMETER :: h=2.0D0/3.0D0           ! BDF time ratio
DOUBLE PRECISION, DIMENSION(2), PARAMETER :: Beta=(/-4.0D0/3.0D0, 1.0D0/3.0D0/) ! BDF concentration coefficient
DOUBLE PRECISION :: hdt                                ! BDF step size
DOUBLE PRECISION :: maxDC, minC, maxDT, minT           ! parameter for adaptive time control
INTEGER, DIMENSION(:,:), ALLOCATABLE :: CIM, CIV       ! component index matrix and vector
INTEGER :: nciv, ndiag, mdiag                          ! dimension of CIV, number of diagonal, position of main diagonal
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: A, A0 ! compact form of Jacobian matrix
DOUBLE PRECISION, DIMENSION(:  ), ALLOCATABLE :: B     ! right hand side of Jacobian matrix
DOUBLE PRECISION, PARAMETER :: tolNewton=1.0E-10       ! tolerance of Newton's method          (1.0E-10 is good)
INTEGER, PARAMETER :: maxIteration=15; ! max number of iteration for Newton's method (13 is good for 1.0E-10 accuracy, important!!)
INTEGER :: it , i, j, k, l, flagLastStep, nIteration   ! dummy indices
!CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2 !
IF (dtLevel > maxdtLevel) THEN ! initialize array for adaptive time control ----------------------------------------------------------------
   WRITE(6,*) "Warning ( dtLevel > 200 ): Unstable numerical method!! Please use smaller tfRun or change numerical method."
END IF
maxDC=0.0D0; ! maximum concentration change
maxDT=0.0D0; ! maximum temperature change
minC=MINVAL(C);  ! minimum concentration
minT=MINVAL(T);  ! minimum temperature
flagLastStep=nt; ! flag to syncronize the last step
tooStiff=.FALSE.;! signal of stiffness for the sub-loop of adaptive time control
ALLOCATE(C0(ngz,ntc,2),T0(ngz,2)) ! initialize array to save previous step ----------------------------------------------------------------
DO k=1,2; C0(:,:,k)=C(:,:); END DO
```

Table E.15 Continued

```
DO k=1,2; T0(:,k)=T; END DO
tRun0=0.0D0;
dt=(tf-tRun)/(1.0D0*nt); ! delta t at this dtLevel ! initialize parameters for IE method ----------------------------------------------------------------|
nciv=(nmEnd-nmStart+1)*ngc+(rEnd-rStart+1)*ntsc ! dimension of CIV, total number of C need to be solved numerically
ALLOCATE(CIM(ngz,ntc),CIV(2,nciv)) ! initialize component index matrix and vector
k=0;
DO j=1,ngz
  DO i=1,ntc
    IF (i>=gStart .AND. i<= gEnd) THEN ! gas component
      IF (j >= nmStart .AND. j <= nmEnd) THEN ! grid need to be solved numerically
        k=k+1; ! index of CIV
        CIM(j,i)=k; ! store column index of CIV
        CIV(1,k)=j; ! store row index of C
        CIV(2,k)=i; ! store column index of C
      ELSE
        CIM(j,i)=0;
      END IF
    ELSEIF (i>=sStart .AND. i<=sEnd) THEN ! solid component
      IF (j>= rStart .AND. j<=rEnd) THEN ! grid of reactor
        k=k+1; ! index of CIV
        CIM(j,i)=k; ! store column index of CIV
        CIV(1,k)=j; ! store row index of C
        CIV(2,k)=i; ! store column index of C
      ELSE
        CIM(j,i)=0; ! set to zero
      END IF
    END IF
  END DO
END DO
SELECT CASE (ecm)
  CASE (0) ! no ecm
    ndiag=2*(ntc+1)-1;   ! number of banded diagonal of Jacobian matrix
    mdiag=ntc+1;         ! column loacation of main diaggonal of Jacobian matrix in the compact form
  CASE (1) ! catalytic site only (element balance equation only relates to solid components)
    ndiag=2*(ntc+1)-1;   ! number of banded diagonal of Jacobian matrix
    mdiag=ntc+1;         ! column loacation of main diaggonal of Jacobian matrix in the compact form
  CASE (2) ! ecm
    ndiag=(2*ntc-1)+ntc+1; ! number of banded diagonal of Jacobian matrix
    mdiag=(2*ntc-1)+1;     ! column loacation of main diaggonal of Jacobian matrix in the compact form
END SELECT
ALLOCATE(A(nciv,ndiag),A0(nciv,ndiag),B(nciv)) ! initialize compact form of Jacobian matrix and vector on the right hand side
DO i=1,ndiag; DO j=1,nciv; A(j,i)=0.0D0; A0(j,i)=0.0D0; END DO; END DO
DO j=1,nciv; B(j)=0.0D0; END DO
DO it=1,nt ! time ingegration ----------------------------------------------------------------------------------------------------------------|
  C0(:,:,mod(it,2)+1)=C; T0(:,mod(it,2)+1)=T; tRun0=tRun ! store data of previous time step
  IF (tRun+dt >= tf .OR. it==nt) THEN ! syncronize the last delta t in case of round off error ----------------------------------------------|
```

Table E.15 Continued

```
  flagLastStep=0; dt=tf-tRun; hdt=h*dt; tRun=tf;
ELSEIF (it==1) THEN !IE
  hdt=dt; flagLastStep=flagLastStep-1; tRun=tRun+dt;
ELSE
   hdt=h*dt; flagLastStep=flagLastStep-1; tRun=tRun+dt;
END IF
IF (it<=2 .OR. it==nt) THEN ! construct the template A0 of Jacobian matrix
  DO j=rStart,rEnd ! construct the compact form of banded Jacobian matrix for gas component ----------------------------------------------------|
    DO i=gStart,min(gEnd,iEnd) ! non-ecm algorithm, add the diffusion term to prestored reacction term
      A0(CIM(j,i),mdiag-ntc)=          -hdt/Tau(j,i)/(dZ**2); ! diffusion to the left grid
      A0(CIM(j,i),mdiag    )=1.0D0+2.0D0*hdt/Tau(j,i)/(dZ**2); ! diffusion and reaction on same place
      A0(CIM(j,i),mdiag+ntc)=          -hdt/Tau(j,i)/(dZ**2); ! diffusion to the right grid
    END DO
    DO i=max(gStart,dStart),gEnd ! ecm algorithm, construct Jacobian by ecm equation of gas component
      A0(CIM(j,i),mdiag-ntc)=          -hdt/Tau(j,i)/(dZ**2); ! diffusion to the left grid
      A0(CIM(j,i),mdiag    )=1.0D0+2.0D0*hdt/Tau(j,i)/(dZ**2); ! diffusion only on this gird
      A0(CIM(j,i),mdiag+ntc)=          -hdt/Tau(j,i)/(dZ**2); ! diffusion to the right grid
      DO l=MDiMIYI(i-nic),MDiMIYI(i-nic+1)-1
        A0(CIM(j,i),mdiag-ntc-(i-MDiMIYL(l)))=MDiMIY(l)*(          -hdt/Tau(j,MDiMIYL(l))/(dZ**2));
        A0(CIM(j,i),mdiag    -(i-MDiMIYL(l)))=MDiMIY(l)*(1.0D0+2.2D0*hdt/Tau(j,MDiMIYL(l))/(dZ**2));
        A0(CIM(j,i),mdiag+ntc-(i-MDiMIYL(l)))=MDiMIY(l)*(          -hdt/Tau(j,MDiMIYL(l))/(dZ**2));
      END DO
    END DO
  END DO
  DO j=rStart,rEnd ! construct the compact form of banded Jacobian matrix for solid component ----------------------------------------------------|
    DO i=sStart,min(cEnd,iEnd)  ! non-ecm algorithm, process the reaction term on solid surface
      A0(CIM(j,i),mdiag)=1.0D0;
    END DO
    DO i=max(sStart,dStart),cEnd ! ecm algorithm,construct Jacobian by ecm equation of solid component
      A0(CIM(j,i),mdiag)=+1.0D0;
      DO l=MDiMIYI(i-nic),MDiMIYI(i-nic+1)-1
        IF     (MDiMIYL(l) >= gStart .AND. MDiMIYL(l) <=gEnd) THEN
          A0(CIM(j,i),mdiag-ntc-(i-MDiMIYL(l)))=MDiMIY(l)/alpha*(          -hdt/Tau(j,MDiMIYL(l))/(dZ**2));
          A0(CIM(j,i),mdiag    -(i-MDiMIYL(l)))=MDiMIY(l)/alpha*(1.0D0+2.2D0*hdt/Tau(j,MDiMIYL(l))/(dZ**2));
          A0(CIM(j,i),mdiag+ntc-(i-MDiMIYL(l)))=MDiMIY(l)/alpha*(          -hdt/Tau(j,MDiMIYL(l))/(dZ**2));
        ELSEIF (MDiMIYL(l) >= sStart .AND. MDiMIYL(l) <= cEnd) THEN
          A0(CIM(j,i),mdiag    -(i-MDiMIYL(l)))=MDiMIY(l);
        END IF
      END DO
    END DO
  END DO
  SELECT CASE (CBC(1)) ! left boundary
    CASE (1) ! Dirichlet on left
      DO i=gStart,gEnd !The gas concentration on the boundary is constant, so the derivative in Jacobian matrix is zero.
        A0(CIM(rStart,i),mdiag-ntc)=0.0D0;
      END DO
```

Table E.15 Continued

```
    CASE (2) ! Neumann on left
      DO i=gStart,min(gEnd,iEnd)
        A0(CIM(rStart,i),mdiag+ntc)=A0(CIM(rStart,i),mdiag+ntc)+A0(CIM(rStart,i),mdiag-ntc); ! add left to right
        A0(CIM(rStart,i),mdiag-ntc)=0.0D0; ! clear left
      END DO
      DO i=dStart,cEnd ! if use ecm algorithm
        A0(CIM(rStart,i),mdiag+ntc)=A0(CIM(rStart,i),mdiag+ntc)+A0(CIM(rStart,i),mdiag-ntc);
        A0(CIM(rStart,i),mdiag-ntc)=0.0D0;
      END DO
    CASE (3) ! Danckwerts on left (No ECM for boundary chamber)
      j=rStart-1;
      DO i=gStart,min(gEnd,iEnd)
        A0(CIM(j,i),mdiag-ntc)=     -hdt/Tau(j-1,i);                ! advection mixing
        A0(CIM(j,i),mdiag    )=1.0D0+hdt/Tau(j-1,i)+hdt/Tau(j,i)/dZ; ! balanced mixing
        A0(CIM(j,i),mdiag+ntc)=                     -hdt/Tau(j,i)/dZ; ! diffusion mixing
      END DO
    END SELECT
    SELECT CASE (CBC(2)) ! right boundary
    CASE (1) ! Dirichlet on right
      DO i=gStart,gEnd !The gas concentration on the boundary is constant, so the derivative in Jacobian matrix is zero.
        A0(CIM(rEnd,i),mdiag+ntc)=0.0D0;
      END DO
    CASE (2) ! Neumann on right
      DO i=gStart,min(gEnd,iEnd) ! this loop takes care of the boundary condition
        A0(CIM(rEnd  ,i),mdiag-ntc)=A0(CIM(rEnd  ,i),mdiag-ntc)+A0(CIM(rEnd  ,i),mdiag+ntc); ! add right to left
        A0(CIM(rEnd  ,i),mdiag+ntc)=0.0D0; ! clear right
      END DO
      DO i=dStart,cEnd ! if use ecm algorithm
        A0(CIM(rEnd  ,i),mdiag-ntc)=A0(CIM(rEnd  ,i),mdiag-ntc)+A0(CIM(rEnd  ,i),mdiag+ntc);
        A0(CIM(rEnd  ,i),mdiag+ntc)=0.0D0;
      END DO
    CASE (3) ! Danckwerts on right (No ECM for boundary chamber)
      j=rEnd+1;
      DO i=gStart,min(gEnd,iEnd) ! this loop takes care of the boundary condition
        A0(CIM(j,i),mdiag-ntc)=                     -hdt/Tau(j,i)/dZ; ! diffusion mixing
        A0(CIM(j,i),mdiag    )=1.0D0+hdt/Tau(j+1,i)+hdt/Tau(j,i)/dZ; ! balanced mixing
        A0(CIM(j,i),mdiag+ntc)=     -hdt/Tau(j+1,i);                ! advection mixing
      END DO
    END SELECT
  END IF
  IF (thermalType /= 1) THEN ! non-isothermal ! calculate the parameter at current time step ----------------------------------------------|
    IF (thermalType == 2) THEN; CALL sTemperature; END IF
    CALL sDiffusion
    CALL sTau
    CALL sRateConstant
  END IF
```

Table E.15 Continued

```
CALL sBoundary
CALL sReaction
CALL sDRxnDc
nIteration=0; ! initialize Newton's method ---------------------------------------------------------------------------------|
DO WHILE (nIteration >= 0 .AND. nIteration <= maxIteration)
  nIteration=nIteration+1;
  A=A0; ! use template
  DO k=1,nciv; B(k)=0.0D0; END DO ! clear B
  DO j=rStart,rEnd ! construct reaction part of Jacobian matrix -----------------------------------------------------------------|
    DO i=gStart,min(gEnd,iEnd) ! first derivate of reaction rate to gas component, dR/dC
      DO k=SJRI(i),SJRI(i+1)-1
        DO l=SJYI(i),SJYI(i+1)-1
          A(CIM(j,i),mdiag-1+SJRL(k)-i+1)=A(CIM(j,i),mdiag-1+SJRL(k)-i+1)-hdt*alpha*SJY(l)*JRY(SJYCL(l),j); ! store S*(dR/dC) in A
        END DO
      END DO
      DO k=urStart,urEnd ! the first derivative of user-defined rate to gas component
        IF (S(i,k) /= 0 .AND. nzUDRD(k-near,i)) THEN
          A(CIM(j,i),mdiag-1+SJRL(k)-i+1)=A(CIM(j,i),mdiag-1+SJRL(k)-i+1)-hdt*alpha*S(i,k)*JUR(k-near,i,j);
        END IF
      END DO
    END DO
    DO i=sStart,min(cEnd,iEnd) ! first derivative of rate to solid component, dR/dC
      DO k=SJRI(i),SJRI(i+1)-1
        DO l=SJYI(i),SJYI(i+1)-1
          A(CIM(j,i),mdiag-1+SJRL(k)-i+1)=A(CIM(j,i),mdiag-1+SJRL(k)-i+1)-hdt*SJY(l)*JRY(SJYCL(l),j); ! store S*(dR/dC) in A
        END DO
      END DO
      DO k=urStart,urEnd ! the first derivative of user-defined rate to gas component
        IF (S(i,k) /= 0 .AND. nzUDRD(k-near,i)) THEN
          A(CIM(j,i),mdiag-1+SJRL(k)-i+1)=A(CIM(j,i),mdiag-1+SJRL(k)-i+1)-hdt*S(i,k)*JUR(k-near,i,j);
        END IF
      END DO
      END DO
  END DO
  DO j=rStart,rEnd ! construct right hand side for Jacobian matrix, (B = -F) ---------------------------------------------------|
    DO i=gStart,min(gEnd,iEnd) ! non-ecm, function of diffusion reaction
      B(CIM(j,i))=-Beta(1)*C0(j,i,mod(it,2)+1)-Beta(2)*C0(j,i,mod(it,2))-C(j,i) &
                          +hdt*((C(j-1,i)-2*C(j,i)+C(j+1,i))/Tau(j,i)/(dZ**2)+NC(j,i));
    END DO
    DO i=max(gStart,dStart),gEnd ! ecm algorithm, for dependent gas component
      B(CIM(j,i))=       -C(j,i)+hdt*( C(j-1,i)-2*C(j,i)+C(j+1,i) )/Tau(j,i)/dZ**2
      DO k=MDiMIYI(i-nic),MDiMIYI(i-nic+1)-1 ! gas only, the equation for solids are ignored
        B(CIM(j,i))=B(CIM(j,i))-MDiMIY(k)* &
                 (C(j,MDiMIYL(k))-hdt*(C(j-1,MDiMIYL(k))-2*C(j,MDiMIYL(k))+C(j+1,MDiMIYL(k)))/Tau(j,MDiMIYL(k))/dZ**2);
      END DO
      DO k=MDiMYI( i-nic),MDiMYI( i-nic+1)-1
```

Table E.15 Continued

```
    IF     (MDiMYL(k) >= gStart .AND. MDiMYL(k)<= gEnd) THEN
      B(CIM(j,i))=B(CIM(j,i))+MDiMY(k)*C(j,MDiMYL(k));
    ELSEIF (MDiMYL(k) >= sStart .AND. MDiMYL(k)<= cEnd) THEN
      B(CIM(j,i))=B(CIM(j,i))+MDiMY(k)*alpha*C(j,MDiMYL(k));
    END IF
  END DO
END DO
DO i=sStart,min(cEnd,iEnd) ! non-ecm, right hand side for solid component B = -Fj,i (reaction only)
  B(CIM(j,i))=-Beta(1)*C0(j,i,mod(it,2)+1)-Beta(2)*C0(j,i,mod(it,2))-C(j,i)+hdt*(+NC(j,i));
END DO
DO i=max(sStart,dStart),cEnd ! ecm, right hand side of ecm equation for soid component
  B(CIM(j,i))=-C(j,i);
  DO k=MDiMIYI(i-nic),MDiMIYI(i-nic+1)-1
    IF     (MDiMIYL(k) >= gStart .AND. MDiMIYL(k) <= gEnd) THEN
      B(CIM(j,i))=B(CIM(j,i))-MDiMIY(k)/alpha* &
            (C(j,MDiMIYL(k))-hdt*(C(j-1,MDiMIYL(k))-2*C(j,MDiMIYL(k))+C(j+1,MDiMIYL(k)))/Tau(j,MDiMIYL(k))/dZ**2);
    ELSEIF (MDiMIYL(k) >= gStart .AND. MDiMIYL(k) <= gEnd) THEN
      B(CIM(j,i))=B(CIM(j,i))-MDiMY(k)*C(j,MDiMIYL(k));
    END IF
  END DO
  DO k=MDiMYI(i-nic),MDiMYI(i-nic+1)-1
    IF     (MDiMYL(k) >= gStart .AND. MDiMYL(k) <= gEnd) THEN
      B(CIM(j,i))=B(CIM(j,i))+MDiMY(k)/alpha*C(j,MDiMYL(k));
    ELSEIF (MDiMYL(k) >= sStart .AND. MDiMYL(k) <= cEnd) THEN
      B(CIM(j,i))=B(CIM(j,i))+MDiMY(k)*C(j,MDiMYL(k));
    END IF
  END DO
END DO
SELECT CASE (CBC(1)) ! left boundary
  CASE (1) ! Dirichlet on left
  CASE (2) ! Neumann on left
  CASE (3) ! Danckwerts on left
    j=rStart-1;
    DO i=gStart,min(gEnd,iEnd)
      B(CIM(j,i))=-Beta(1)*C0(j,i,mod(it,2)+1)-Beta(2)*C0(j,i,mod(it,2))-C(j,i) &
            +hdt*((-C(j,i)+C(j+1,i))/Tau(j,i)/dZ+(C(j-1,i)-C(j,i))/Tau(j-1,i)); ! compute RHS of gas B=-F
    END DO
END SELECT
SELECT CASE (CBC(2)) ! right boundary
  CASE (1) ! Dirichlet on right
  CASE (2) ! Neumann on right
  CASE (3) ! Danckwerts on right
    j=rEnd+1;
    DO i=gStart,min(gEnd,iEnd) ! this loop takes care of the boundary condition
      B(CIM(j,i))=-Beta(1)*C0(j,i,mod(it,2)+1)-Beta(2)*C0(j,i,mod(it,2))-C(j,i) &
```

Table E.15 Continued

```
        +hdt*((C(j-1,i)-C(j,i))/Tau(j,i)/dZ+(-C(j,i)+C(j+1,i))/Tau(j+1,i)); ! compute RHS of gas B=-F
      END DO
    END SELECT
    SELECT CASE (ecm)
    CASE (0)
      CALL sGaussBand(A,B,nciv,2*ntc+1) ! solve (J^-1)*(-F) by Gauss elemination for banded matrix, B <-- (A^-1)*B
    CASE (1)
      CALL sGaussBand(A,B,nciv,2*ntc+1) ! solve (J^-1)*(-F) by Gauss elemination for banded matrix, B <-- (A^-1)*B
    CASE (2)
      CALL sGausslBand(A,b,nciv,2*ntc-1,ntc,mdiag,ndiag) ! solve (J^-1)*(-F) by Gauss elemination for left banded matrix
    END SELECT
    DO k=1,nciv
      C(CIV(1,k),CIV(2,k))=C(CIV(1,k),CIV(2,k))+B(k); ! update concentration
    END DO
    IF (thermalType==3) THEN ! solve energy equation
      CALL sEnergy
      maxDT=MAXVAL(abs(T(nmStart:nmEnd)-T0(nmStart:nmEnd,mod(it,2)+1))); ! maximum temperature change
      minT=MINVAL(T);                                                    ! minimum temperature
    END IF
    IF (thermalType /= 1) THEN ! non-isothermal ! calculate the parameter at current time step ----------------------------------------|
      IF (thermalType == 2) THEN; CALL sTemperature; END IF
      CALL sDiffusion
      CALL sTau
      CALL sRateConstant
    END IF
    CALL sBoundary
    CALL sReaction
    CALL sDRxnDc
    IF (MAXVAL(abs(B)) >= 0.0D0 .AND. MAXVAL(abs(B)) < tolNewton) THEN ! exit loop if the Newton's method converges
      EXIT
    ELSEIF (nIteration >= maxIteration) THEN ! exit loop with stiff signal if didn't converge
      tooStiff=.TRUE.;
      EXIT
    END IF
  END DO
  maxDC=max(MAXVAL(abs(C(nmStart:nmEnd,gStart:gEnd)-C0(nmStart:nmEnd,gStart:gEnd,mod(it,2)+1))), & ! maximum concentration change
        MAXVAL(abs(C(rStart:rEnd,sStart:cEnd)-C0(rStart:rEnd,sStart:cEnd,mod(it,2)+1))));     ! from numerical computation
  minC=MINVAL(C(:,:)); ! minimum concentration
  WRITE(6,*) tRun
  IF (minC < 0.0D0 .AND. abs(minC) < zeroC) THEN ! set the negative round off error to zero
    DO i=1,ntc
      DO j=1,ngz
        IF (C(j,i) < 0.0D0 .AND. abs(C(j,i)) < zeroC) THEN
          C(j,i)=0.0D0;
        END IF
      END DO
```

Table E.15 Continued

```
    END DO
    minC=MINVAL(C); ! recalculate minimum concentration
  END IF
  IF (maxDC > tolMaxDC .OR. minC < tolMinC .OR. maxDT > tolMaxDT .OR. minT < tolMinT .OR. tooStiff) THEN ! adaptive time control --|
      tooStiff=.TRUE.;
    C=C0(:,:,mod(it,2)+1); T=T0(:,mod(it,2)+1); tRun=tRun0;
      IF (switchMethod) THEN; RETURN; END IF
    IF (dtLevel == 1) THEN ! main loop
      tRun0=min(tRun+dt,tf); ! store in tRun0 temporary
        dtLevel=max(dtLevelOld-1,1); ! set the dtLevel to dtLevelOld-1, so the next adaptive loop starts the same as last dtLevel
        DO WHILE (tRun < tRun0)
        IF (tooStiff) THEN
          dtLevel=dtLevel+1; ! increase the number of sub-loops if the previous sub-loop returns stiff signal
          tooStiff=.FALSE.;
        END IF
        ELSE
          dtLevel=max(dtLevel-1,1); ! decrease the number of sub-loops if the previous sub-loop didn't return stiff signal
          tooStiff=.FALSE.;
        END IF
        CALL sBDF2(min(tRun+((1.0D0/nst)**max((dtLevel-2),0))*dt,tRun0),nst) ! adaptive time integration
      END DO
        dtLevelOld=dtLevel; ! save the last dtLevel, so the next adaptive loop will start at the same level
      dtLevel=1; ! important!! pull back to dtLevel 1 if the DO WHILE condition satisfied
    ELSEIF (dtLevel > 1) THEN ! sub loop
      RETURN
    ELSE
      WRITE(6,*) "Error: dtLevel < 0."
      PAUSE; STOP
    END IF
  ELSE ! passed
    tooStiff=.FALSE.;
  END IF
  IF (dtLevel==1 .AND. .NOT.switchMethod) THEN; CALL sSave("Dynamic"); END IF ! save date
  IF (tRun >= tRuntime) THEN; CALL sRuntime(6); END IF ! write runtime information on screen
  IF (tRun>tf) THEN
    WRITE(6,*) "Warning: Current time is greater than final time. tRun > tf at dtLevel ", dtLevel, " and tRun-tf=", tRun-tf, ".";
  END IF
  IF (fLagLastStep == 0) THEN; EXIT; END IF ! exit this loop
END DO
IF (ALLOCATED(C0 )) THEN; DEALLOCATE(C0 ); END IF ! deallocate array ----------------------------------------------------------------|
IF (ALLOCATED(T0 )) THEN; DEALLOCATE(T0 ); END IF
IF (ALLOCATED(CIM)) THEN; DEALLOCATE(CIM); END IF
IF (ALLOCATED(CIV)) THEN; DEALLOCATE(CIV); END IF
IF (ALLOCATED(A  )) THEN; DEALLOCATE(A  ); END IF
IF (ALLOCATED(A0 )) THEN; DEALLOCATE(A0 ); END IF
IF (ALLOCATED(B  )) THEN; DEALLOCATE(B  ); END IF
END SUBROUTINE sBDF2
```

Table E.16 sLOA.f90

```
!-----------------------------------------------------------------------
! Subroutine sLOA
! Description: Lower order adaptive method for transcat. This subroutine is working, but the stragedy of switching between method
!              is still underdeveloping
! Created by Hsu-Wen Hsiao on 8/14/10.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   tf: final time
!   nt: number of time step
!-----------------------------------------------------------------------
RECURSIVE SUBROUTINE sLOA(tf,nt)
USE mDynamic
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: tf ! final time
INTEGER, INTENT(IN) :: nt          ! number of time step
INTEGER, PARAMETER :: nMethod=2    ! number of numerical method
LOGICAL, DIMENSION(maxdtLevel,nMethod) :: IsStiff ! check if the method is stiff at a dtLevel
DOUBLE PRECISION, DIMENSION(maxdtLevel,nMethod) :: tMethod ! runtime of method to finish one second of a task
DOUBLE PRECISION :: tRun0, tRun1 ! initial and final time of each interval
INTEGER, PARAMETER :: loaStragedy=1
DOUBLE PRECISION :: bestTime, swapTime
INTEGER :: bestMethod
DOUBLE PRECISION :: startTime, endTime ! time of testing mehtod
DOUBLE PRECISION :: dt, subdt          ! step size
INTEGER :: it, flagLastStep, i, j, k   ! dummy indices
!CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
dt=(tf-tRun)/(1.0D0*nt); ! delta t at this dtLevel ! initialize parameters for IE method ----------------------------------------
flagLastStep=nt; ! flag to syncronize the last step
IF (dtLevel > maxdtLevel) THEN ! initialize array for adaptive time control ----------------------------------------------------------
   WRITE(6,*) "Warning ( dtLevel > 200 ): Unstable numerical method!! Please use smaller tfRun or change numerical method."
END IF
switchMethod=.TRUE.; ! set to true so can switch between methods
tooStiff=.FALSE.;    ! signal of stiffness for the sub-loop of adaptive time control
dtLevel=1;
dtLevelOld=dtLevel;
tRun0=tRun;
tRun1=tRun+dt;
DO i=1,nMethod
  DO j=1,maxdtLevel
     IsStiff(j,i)=1;
     tMethod(j,i)=1.0/EPSILON(tMethod(j,i));
  END DO
EnD DO
bestMethod=1;
DO it=1,nt ! main loop ----------------------------------------------------------------------------------------------------------
```

Table E.16 Continued

```
IF (tRun+dt >= tf .OR. it==nt) THEN ! syncronize the last delta t in case of round off error ----------------------------------------|
   flagLastStep=0;
   dt=tf-tRun;
      tRun1=tf;
ELSE
   flagLastStep=flagLastStep-1;
      tRun1=tRun+dt;
END IF
tRun0=tRun;
dtLevel=dtLevelOld;

DO WHILE (tRun<tRun1 .OR. tooStiff) ! choosing between methods until tRun=tRun1 ----------------------------------------------------------|
  tooStiff=0; ! tiptoe a small time step to get an idea of the stiffness and computing of each method at different dtLevel
  DO i=1,nMethod
    DO j=MIN(dtLevel+2,maxdtLevel),MAX(dtLevel-2,1),-1 ! test two before and after dtLevel
       subdt=((1.0D0/nst)**max((j-2+1),0))*dt;
       IF (tRun+subdt > tRun1) THEN
         subdt=tRun1-tRun;
       END IF
    CALL CPU_TIME(startTime)
    SELECT CASE (i)
       CASE (1)
       CALL sEE(tRun+subdt,1) ! test integrate one step size
       CASE (2)
          CALL sIE(tRun+subdt,1) ! test integrate one step size
    END SELECT
    CALL CPU_TIME(endTime)
    tMethod(j,i)=nst**j*(endTime-startTime)/(j-1)/dt;
  IF(tMethod(j,i) < 0.0D0) THEN
     tMethod(j,i)=1.0/EPSILON(tMethod(j,i));
    END IF
    IF (tooStiff) THEN
      IsStiff(j,i)=1;
      tooStiff=0;
    ELSE
      IsStiff(j,i)=0;
    END IF
    !WRITE(6,*) j, subdt, IsStiff(j,i), tMethod(j,i)
    IF (tRun >= tRun1) THEN; EXIT; END IF
  END DO
  IF (tRun >= tRun1) THEN; EXIT; END IF
END DO
DO j=maxdtLevel,1,-1 ! find the time of current best method
  IF(IsStiff(j,bestMethod)) THEN ! if not too stiff
    k=min(j+1,maxdtLevel);
      EXIT
```

Table E.16 Continued

```
   END IF
  END DO
  !WRITE(6,*) k
  !WRITE(6,*) (k-1)+MINLOC(tMethod(k:maxdtLevel,bestMethod))
  !j=(k-1)+MINLOC(tMethod(k:maxdtLevel,bestMethod),1);
  bestTime=tMethod((k-1)+MINLOC(tMethod(k:maxdtLevel,bestMethod),1),bestMethod);
  !WRITE(6,*) "old record", bestMethod, bestTime
  SELECT CASE (loaStragedy)
    CASE (1) ! lowest runtime
      DO i=1,nMethod
        DO j=maxdtLevel,1,-1
          IF(IsStiff(j,i)) THEN ! find stiff point
              k=min(j+1,maxdtLevel);
              swapTime=tMethod((k-1)+MINLOC(tMethod(k:maxdtLevel,i),1),i);
              IF(swapTime > 0.0D0 .AND. swapTime < bestTime) THEN
                bestMethod=i;
              bestTime=swapTime;
              dtLevel=k;
              END IF
              EXIT
          END IF
        END DO
      END DO
    CASE (2) ! highest dtLevel
      bestMethod=1;
      dtLevel=maxdtLevel;
      DO i=1,nMethod
        DO j=maxdtLevel,1,-1
          IF(.NOT.IsStiff(j,i)) THEN
              dtLevel=j;
              bestMethod=i;
              bestTime=tMethod(j,i);
            ELSE
              EXIT
            END IF
        END DO
      END DO
  END SELECT
  !WRITE(6,*) "new record", bestMethod, dtLevel, bestTime
  !WRITE(6,*) "tooStiff=",tooStiff, ", switchMethod=", switchMethod, ", dtLevel=", dtLevel, "bMethod =", bestMethod
  !WRITE(6,"(4(1X,A11))") "subdt", "tRun", "tRun0", "tRun1"
  !WRITE(6,"(4(1X,1P1E11.3E3))") subdt, tRun, tRun0, tRun1
  !WRITE(6,fInsInt2("(1X,A9/,",maxdtLevel,"(",nMethod,"(1X,I1)/))")) "IsStiff =", transpose(IsStiff)
  !WRITE(6,fInsInt2("(1X,A9/,",maxdtLevel,"(",nMethod,"(1X,1P1E11.3E3)/))")) "tMethod =", transpose(tMethod)
  !WRITE(6,"(1X,A9/, 2 ( 10 (1X,I1)/))") "IsStiff =", transpose(IsStiff)
  !WRITE(6,fInsInt2("(",nMethod,"(",maxdtLevel,"(1X,I1)/))")) transpose(IsStiff)
```

Table E.16 Continued

```
  IF (tRun < tRun1) THEN
    SELECT CASE (bestMethod)
      CASE (1)
        CALL sEE(tRun1,CEILING((tRun1-tRun)/(dt*(1.0D0/nts)**(min(dtLevel-1,1)))));
      CASE (2)
        CALL sIE(tRun1,CEILING((tRun1-tRun)/(dt*(1.0D0/nts)**(min(dtLevel-1,1)))));
    END SELECT
  ELSE
    EXIT
  END IF
  !WRITE(6,*) "**************************************************"
  !WRITE(6,*) tRun
  !WRITE(6,*) tRun0, dt
  !WRITE(6,*) tooStiff
END DO ! while
dtLevel0ld=dtLevel;
dtLevel=1;
IF (flagLastStep == 0) THEN ! syncronize the last delta t in case of round off error
    tRun=tf;
END IF
CALL sSave("Dynamic");
IF (tRun >= tRuntime) THEN; CALL sRuntime(6); END IF ! write runtime information on screen
IF (tRun>tf) THEN
  WRITE(6,*) "Warning: Current time is greater than final time. tRun > tf at dtLevel ", dtLevel, " and tRun-tf=", tRun-tf, ".";
END IF
END DO ! it
END SUBROUTINE sLOA
```

Table E.17 sElement.f90

```
!-----------------------------------------------------------------------
! Subroutine sElement
! Description: Subroutine of element constraint method to compute the concentration of dependent component. The linear part of
!              equation must be computed and stored in array LC before calling this subroutine.
! Created by Hsu-Wen Hsiao on 2/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   C0: concentration of previous time step
!   dt: step size
!-----------------------------------------------------------------------
SUBROUTINE sElement(C0,dt)
USE mElement
USE mDynamic
IMPLICIT NONE
CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
DOUBLE PRECISION, DIMENSION(ngz,ntc), INTENT(IN) :: C0
DOUBLE PRECISION, INTENT(IN) :: dt
DOUBLE PRECISION, PARAMETER :: zeroC=EPSILON(zeroC); ! zero concentration (for eliminate round off error)
INTEGER :: i, j, k, l
SELECT CASE (ecm)
CASE (1) ! Catalytic Site only ------------------------------------------------------------------------------------------
C(rStart:rEnd,cStart:cEnd)=SCtot(rStart:rEnd,1:ncs);
DO i=1,ncs
  DO k=MscI(i),MscI(i+1)-1
    C(rStart:rEnd,nic+i)=C(rStart:rEnd,nic+i)-Msc(k)*C(rStart:rEnd,MscL(k));
  END DO
END DO
CASE (2) ! Full ECM -----------------------------------------------------------------------------------------------------
DO k=1,max(0,ngc-nic) ! enter this loop if ngc > nic
  i=nic+k;
  DO j=nmStart,nmEnd ! grid solved numerically ! dependent gas component
    C(j,i)=0.0D0; ! initialize
    DO l=MDiMYI(k),MDiMYI(k+1)-1 ! apply concentration and linear part of previous time step
      IF      (MDiMYL(l) >= gStart .AND. MDiMYL(l) <= gEnd) THEN ! multiplication without alpha
        C(j,i)=C(j,i)+MDiMY(l)*(C0(j,MDiMYL(l))+dt*LC(j,MDiMYL(l)));
      ELSEIF (MDiMYL(l) >= sStart .AND. MDiMYL(l) <= cEnd) THEN ! multiply by alpha
        C(j,i)=C(j,i)+alpha*MDiMY(l)*(C0(j,MDiMYL(l))); ! (linear part of solid component is zero)
        !C(j,i)=C(j,i)+alpha*MDiMY(l)*(C0(j,MDiMYL(l))+dt*LC(j,MDiMYL(l)));(Keep this! Don't use this equation since no linear part)
      END IF
    END DO
  END DO
  DO l=MDiMIYI(k),MDiMIYI(k+1)-1 ! apply concentration of independent component at current time step
    IF      (MDiMIYL(l) >= gStart .AND. MDiMIYL(l) <= gEnd) THEN ! multiplication without alpha
      C(j,i)=C(j,i)-MDiMIY(l)*C(j,MDiMIYL(l)); ! correct by independent gas component
    !ELSEIF (MDiMIYL(l) >= sStart .AND. MDiMIYL(l) <= cEnd) THEN ! multiply by alpha
      !C(j,i)=C(j,i)-alpha*MDiMIY(l)*C(j,MDiMIYL(l)); ! correct by independent solid component and alpha(redundant since ngc>nic)
```

Table E.17 Continued

```
      END IF
    END DO
  END DO
END DO
DO k=1+max(0,ngc-nic),min(nsc,ndc-ncs)+max(0,ngc-nic) ! enter this loop for dependent solid component only (non-catalytic site)
  i=nic+k;
  DO j=rStart,rEnd ! grid in reactor
      C(j,i)=C0(j,i);
      DO l=MDiMIYI(k),MDiMIYI(k+1)-1 ! apply concentration of independent component at current time step
        IF     (MDiMIYL(l) >= gStart .AND. MDiMIYL(l) <= gEnd) THEN ! divide by alpha
        C(j,i)=C(j,i)-MDiMIY(l)*( (C(j,MDiMIYL(l))-C0(j,MDiMIYL(l))) - dt*LC(j,MDiMIYL(l)) )/alpha;! correct by indep gas and alpha
        ELSEIF (MDiMIYL(l) >= sStart .AND. MDiMIYL(l) <= cEnd) THEN ! divide without alpha
        C(j,i)=C(j,i)-MDiMIY(l)*(  C(j,MDiMIYL(l))-C0(j,MDiMIYL(l))  ); ! correct by independent solid component
        END IF
      END DO
  END DO
END DO
C(rStart:rEnd,cStart:cEnd)=SCtot(rStart:rEnd,1:ncs);
DO i=1,ncs
  DO k=MscI(i),MscI(i+1)-1
    C(rStart:rEnd,cStart+i-1)=C(rStart:rEnd,cStart+i-1)-Msc(k)*C(rStart:rEnd,MscL(k));
  END DO
END DO
CASE DEFAULT ! no ecm -------------------------------------------------------------------------------------------------------|
WRITE(6,*) "The subroutine sElement should not be called for non-ecm algorithm. Please check your input file and source code."
STOP
END SELECT
!IF (tRun<= 0.0D0) THEN
!WRITE(6,"(3(1X,A5,1X,I3,',',1X,A5,1X,I3)") "ecm =", ecm, "nce =", nce, "ndc =", ndc, "nic =", nic
!WRITE(6,"(4(1X,A8,1X,I3,',',1X,A8,1X,I3)") "iStart =", iStart, "  iEnd =", iEnd, "dStart =", dStart, "  dEnd =", dEnd
!WRITE(6,fInsInt("(1X,A8/,1X,",ndc+1-1,"(I4,','),I4)")) "MDiMYI =", MDiMYI
!WRITE(6,fInsInt("(1X,A8/,1X,",MDiMYI(ndc+1)-1-1,"(I4,','),I4)")) "MDiMYL =", MDiMYL
!WRITE(6,fInsInt("(1X,A8/,1X,",MDiMYI(ndc+1)-1-1,"(1P1E10.2E3,','),1P1E10.2E3)")) "MDiMY  =", MDiMY
!WRITE(6,fInsInt("(1X,A9/,1X,",ndc+1-1,"(I4,','),I4)")) "MDiMIYI =", MDiMIYI
!WRITE(6,fInsInt("(1X,A9/,1X,",MDiMIYI(ndc+1)-1-1,"(I4,','),I4)")) "MDiMIYL =", MDiMIYL
!WRITE(6,fInsInt("(1X,A9/,1X,",MDiMIYI(ndc+1)-1-1,"(1P1E10.2E3,','),1P1E10.2E3)")) "MDiMIY  =", MDiMIY
!WRITE(6,fInsInt("(1X,A6/,1X,",ncs+1-1,"(I4,','),I4)")) "MscI =", MscI
!WRITE(6,fInsInt("(1X,A6/,1X,",MscI(ncs+1)-1-1,"(I4,','),I4)")) "MscL =", MscL
!WRITE(6,fInsInt("(1X,A6/,1X,",MscI(ncs+1)-1-1,"(I4,','),I4)")) "Msc  =", Msc
!WRITE(6,fInsInt2("(1X,A7/,",ngz,"(1X,",ncs-1,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "SCtot =", transpose(SCtot)
!WRITE(6,fInsInt2("(1X,A3/,",ngz,"(1X,",ntc-1,"(1P1E10.2E3,','),1P1E10.2E3/)\)")) "C =", transpose(C)
!END IF
END SUBROUTINE sElement
```

Table E.18 LinearAlgebra.f90

```
!-----------------------------------------------------------------------
! Function fRank
! Description: Compute the rank of a matrix by Thomas algorithm.
! Created by Hsu-Wen Hsiao on 7/22/2010.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   B: input matrix (integer matrix)
!   nRow: number of rows
!   nCol: number of columns
! Output:
!   fRank: rank of input matrix
!-----------------------------------------------------------------------
INTEGER FUNCTION fRank(B,nRow,nCol)
IMPLICIT NONE
INTEGER, DIMENSION(nRow,nCol), INTENT(IN) :: B ! input matrix
INTEGER, INTENT(IN) :: nRow, nCol ! number of rows and columns in matrix B
DOUBLE PRECISION, DIMENSION(nRow,nCol) :: A
DOUBLE PRECISION, DIMENSION(nCol) :: swap
LOGICAL :: zero
INTEGER :: i, j, k
DO j=1,nCol ! Convert the integer matrix AI to double precision matrix A ----------
  DO i=1,nRow
    A(i,j)=1.0D0*B(i,j);
  END DO
END DO
DO i=1,nRow-1 ! Forward sweep of Thomas algorithm ----------
  IF (abs(A(i,i)) > 0.0D0) THEN ! sweep
    DO j=i+1,nRow
      A(j,i:nCol)=A(j,i:nCol)+(-A(j,i)/A(i,i))*A(i,i:nCol);
    END DO
  ELSE ! swap first, then sweep
    DO j=i+1,nRow
      IF (abs(A(j,i)) >0.0D0) THEN
        swap=A(i,1:nCol);
        A(i,1:nCol)=A(j,1:nCol);
        A(j,1:nCol)=swap;
        EXIT
      END IF
    END DO
    IF (j<nRow) THEN
      DO k=j+1,nRow
        A(k,i:nCol)=A(k,i:nCol)+(-A(k,i)/A(i,i))*A(i,i:nCol);
      END DO
    END IF
  END IF
```

Table E.18 Continued

```
END DO
fRank=nRow; ! calculate the rank of A ------------------------------------------------------------------------------
DO i=1,nRow
  zero=.TRUE.
  DO j=1,nCol
    IF (A(i,j)>0.0D0 .OR. A(i,j)<0.0D0) THEN
        zero=.FALSE. ! check if nonzero
      END IF
  END DO
  IF (zero) THEN
    fRank=fRank-1;
  END IF
END DO
END FUNCTION fRank
!---------------------------------------------------------------------------------------------------
! Subroutine sGauss.f90                                                                            |
! Description: Solve a linear system Ax=b by Gauss elimination. The size of matrix A is n-by-n, and the output x is stored in b.
! Created by Hsu-Wen Hsiao on 8/1/10.                                                              |
! Copyright 2010 UCSD. All rights reserved.                                                        |
!---------------------------------------------------------------------------------------------------
SUBROUTINE sGauss(A,b,n)
IMPLICIT NONE
INTEGER,INTENT(IN) :: n
DOUBLE PRECISION, DIMENSION(n,n) :: A
DOUBLE PRECISION, DIMENSION(n) :: b
INTEGER :: i, j
DO j=1,n-1 ! forward sweep
  DO i=j+1,n
    A(i,j)=-A(i,j)/A(j,j);
  A(i,j+1:n)=A(i,j+1:n)+A(i,j)*A(j,j+1:n);
    b(i)=b(i)+A(i,j)*b(j);
  END DO
END DO
b(n)=b(n)/A(n,n); ! backward substitution
DO i=n-1,1,-1
  DO j=i+1,n
    b(i)=b(i)-A(i,j)*b(j);
  END DO
  b(i)=b(i)/A(i,i);
END DO
END Subroutine sGauss
!---------------------------------------------------------------------------------------------------
! Subroutine sGaussBand.f90                                                                        |
! Description: Solve a banded linear system Ax=b by Gauss elimination, in which A is stored in compact form. The size of matrix A |
!   is n-by-d, where n is the size of origina matrix A and d is the number of banded diagonal. The output x is stored in b.      |
! Created by Hsu-Wen Hsiao on 8/1/10.                                                              |
```

Table E.18 Continued

```
! Copyright 2010 UCSD. All rights reserved.
!-------------------------------------------------------------------------------
SUBROUTINE sGaussBand(A,b,n,d)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n, d
DOUBLE PRECISION, DIMENSION(n,d) :: A
DOUBLE PRECISION, DIMENSION(n), INTENT(INOUT) :: b
INTEGER :: m
INTEGER :: i, j
m=(d+1)/2; ! position of main diagonal
DO i=1,n-m+1 ! forward sweep
  DO j=1,m-1
    A(i+j,m-j)=-A(i+j,m-j)/A(i,m);
    A(i+j,m-j+1:d-j)=A(i+j,m-j+1:d-j)+A(i+j,m-j)*A(i,m+1:d);
    b(i+j)=b(i+j)+A(i+j,m-j)*b(i);
  END DO
END DO
DO i=n-m+2,n-1
  DO j=1,n-i
    A(i+j,m-j)=-A(i+j,m-j)/A(i,m);
    A(i+j,m-j+1:m-j+n-i)=A(i+j,m-j+1:m-j+n-i)+A(i+j,m-j)*A(i,m+1:m+n-i);
    b(i+j)=b(i+j)+A(i+j,m-j)*b(i);
  END DO
END DO
DO i=n,n-m+2,-1 ! backward substitution
  DO j=1,n-i
     b(i)=b(i)-A(i,m+j)*b(i+j);
  END DO
  b(i)=b(i)/A(i,m);
END DO
DO i=n-m+1,1,-1
  DO j=1,m-1
    b(i)=b(i)-A(i,m+j)*b(i+j);
  END DO
  b(i)=b(i)/A(i,m);
END DO
END Subroutine sGaussBand
!-------------------------------------------------------------------------------
! Subroutine sGaussLBand.f90
! Description: Solve a left diagonally dominant banded linear system Ax=b with A stored in compact form by Gauss elimination. The
!              size of matrix A is n, the number of left diagonal is ldiag and the number of right diagonal is rdiag. The position
!              of main diagonal is mdiag. The total number of diagonal is ndiag. The output solution of x is stored in b.
! Created by Hsu-Wen Hsiao on 8/10/10.
! Copyright 2010 UCSD. All rights reserved.
!-------------------------------------------------------------------------------
SUBROUTINE sGaussLBand(A,b,n,ldiag,rdiag,mdiag,ndiag)
```

Table E.18 Continued

```
IMPLICIT NONE
INTEGER, INTENT(IN) :: n, ldiag, rdiag, mdiag, ndiag
DOUBLE PRECISION, DIMENSION(n,ndiag) :: A
DOUBLE PRECISION, DIMENSION(n), INTENT(INOUT) :: b
INTEGER :: i, j
DO i=1,n-ldiag ! forward sweep
  DO j=1,ldiag
    b(i+j)=b(i+j)-A(i+j,mdiag-j)/A(i,mdiag)*b(i);
    A(i+j,mdiag-j:ndiag-j)=-A(i+j,mdiag-j)*(A(i,mdiag:ndiag)/A(i,mdiag))+A(i+j,mdiag-j:ndiag-j);
  END DO
END DO
DO i=n-ldiag+1,n-rdiag ! forward sweep
  DO j=1,n-i
    b(i+j)=b(i+j)-A(i+j,mdiag-j)/A(i,mdiag)*b(i);
    A(i+j,mdiag-j:ndiag-j)=-A(i+j,mdiag-j)*(A(i,mdiag:ndiag)/A(i,mdiag))+A(i+j,mdiag-j:ndiag-j);
  END DO
END DO
DO i=n-rdiag,n-1 ! forward sweep
  DO j=1,n-i
    b(i+j)=b(i+j)-A(i+j,mdiag-j)/A(i,mdiag)*b(i);
    A(i+j,mdiag-j:mdiag-j+n-i)=-A(i+j,mdiag-j)*(A(i,mdiag:mdiag+n-i)/A(i,mdiag))+A(i+j,mdiag-j:mdiag-j+n-i);
  END DO
END DO
DO i=n,n-rdiag,-1 ! backward substitution
  DO j=1,n-i
    b(i)=b(i)-A(i,mdiag+j)*b(i+j);
  END DO
  b(i)=b(i)/A(i,mdiag);
END DO
DO i=n-rdiag-1,1,-1 ! backward substitution
  DO j=1,rdiag
    b(i)=b(i)-A(i,mdiag+j)*b(i+j);
  END DO
  b(i)=b(i)/A(i,mdiag);
END DO
END Subroutine sGaussLBand
!-----------------------------------------------------------------------
! Subroutine sInverse
! Description: Calculate the inverse of a square matrix by Gauss-Jordan method.
! Created by Hsu-Wen Hsiao on 8/3/2010.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   A: input matrix (double precision)
!   n: size of matrix A
! Output:
```

Table E.18 Continued

```
!  A: inverse of matrix A
!-----------------------------------------------------------------------
SUBROUTINE sInverse(A,n)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
DOUBLE PRECISION, DIMENSION(n,n), INTENT(INOUT) :: A
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: B
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: swap
INTEGER :: i, j
ALLOCATE(B(n,2*n),swap(2*n))
DO i=1,n
  DO j=1,n
    B(i,j)=A(i,j);
      IF (i == j) THEN
        B(i,n+j)=1.0D0;
      ELSE
        B(i,n+j)=0.0D0;
      END IF
  END DO
END DO
DO j=1,n-1
  IF (abs(B(j,j)) <= 0.0D0) THEN ! swap if zero on main diagonal
      DO i=j+1,n
          IF (abs(B(i,j)) > 0.0D0) THEN
              swap=B(j,1:2*n);
              B(j,1:2*n)=B(i,1:2*n);
              B(i,1:2*n)=swap;
          END IF
      END DO
  END IF
  B(j,1:2*n)=B(j,1:2*n)/B(j,j);
  DO i=j+1,n
      IF (abs(B(i,j)) > 0.0D0) THEN
          B(i,1:2*n)=B(i,1:2*n)/B(i,j); B(i,1:2*n)=B(i,1:2*n)-B(j,1:2*n);
      END IF
  END DO
END DO
B(n,1:2*n)=B(n,1:2*n)/B(n,n);
DO j=n,2,-1
    DO i=j-1,1,-1
        B(i,1:2*n)=B(i,1:2*n)-B(i,j)*B(j,1:2*n);
    END DO
END DO
A=B(1:n,n+1:2*n)
DEALLOCATE(B,swap)
END SUBROUTINE sInverse
```

Table E.19 sBoundary.f90

```
!-----------------------------------------------------------------------------------------------------------------------------|
! Subroutine sBoundary                                                                                                        |
! Description: Update the concentration and/or temperature on the boundary.                                                  |
! Created by Hsu-Wen Hsiao on 8/4/2010.                                                                                      |
! Copyright 2010 UCSD. All rights reserved.                                                                                  |
!-----------------------------------------------------------------------------------------------------------------------------|
SUBROUTINE sBoundary
USE mDynamic
USE mBoundary
IMPLICIT NONE
DOUBLE PRECISION, EXTERNAL :: fModulation
INTEGER :: i
SELECT CASE (CBC(1)) ! concentration on left boundary ---------------------------------------------------------------------|
  CASE (1) ! Dirichlet-type
    DO i=1,ngc
      C(rStart-1,i)=fModulation(tRun,BCW(1,i),BCCmax(1,i),BCCmin(1,i),BCF(1,i),BCL(1,i));
    END DO
  CASE (2) ! von Neumann-type
    C(rStart-1,gStart:gEnd)=C(rStart,gStart:gEnd); ! redundant process for CSTR
  CASE (3) ! Danckwerts' type
    DO i=1,ngc
      C(rStart-2,i)=fModulation(tRun,BCW(1,i),BCCmax(1,i),BCCmin(1,i),BCF(1,i),BCL(1,i));
    END DO
END SELECT
SELECT CASE (CBC(2)) ! concentration on right boundary --------------------------------------------------------------------|
  CASE (1) ! Dirichlet-type
    DO i=1,ngc
      C(rEnd+1,i)=fModulation(tRun,BCW(2,i),BCCmax(2,i),BCCmin(2,i),BCF(2,i),BCL(2,i));
    END DO
  CASE (2) ! von Neumann-type
    C(rEnd+1,gStart:gEnd)=C(rEnd,gStart:gEnd); ! redundant process for CSTR
  CASE (3) ! Danckwerts' type
    DO i=1,ngc
      C(rEnd+2,i)=fModulation(tRun,BCW(2,i),BCCmax(2,i),BCCmin(2,i),BCF(2,i),BCL(2,i));
    END DO
END SELECT
SELECT CASE (TBC(1)) ! temperature on left boundary -----------------------------------------------------------------------|
  CASE (0) ! Isothermal
    ! Do nothing (or check something)
  CASE (1) ! Modulated (Dirichlet-type)
    T(rStart-1)=fModulation(tRun,BCTW(1),BCTmax(1),BCTmin(1),BCTF(1),BCTL(1));
  CASE (2) ! adiabatic
    T(rStart-1)=T(rStart);
  CASE (3) ! Danckwerts' type?? or called something else  (or add heat source)
    T(rStart-2)=fModulation(tRun,BCTW(1),BCTmax(1),BCTmin(1),BCTF(1),BCTL(1));
END SELECT
```

Table E.19 Continued

```
SELECT CASE (TBC(2)) ! right boundary ------------------------------------------------------------------
  CASE (0) ! Isothermal
    ! Do nothing (or check something)
  CASE (1) ! Dirichlet-type
    T(rEnd+1)=fModulation(tRun,BCTW(2),BCTmax(2),BCTmin(2),BCTF(2),BCTL(2));
  CASE (2) ! adiabatic (or add heat source)
    T(rEnd+1)=T(rEnd);
  CASE (3) ! Danckwerts' type?? or called something else  (or add heat source)
    T(rEnd+2)=fModulation(tRun,BCTW(2),BCTmax(2),BCTmin(2),BCTF(2),BCTL(2));
END SELECT
END SUBROUTINE sBoundary
!-----------------------------------------------------------------------------
! Function fModulation
! Description: Compute concentration or temperature modulation.
! Created by Hsu-Wen Hsiao on 7/21/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------------
! Input:
!   t: runtime
!   w: waveform
!   Amax: maximum amplitude
!   Amin: minmum amplitude
!   f: frequency
!   pl: phase lag
! Output:
!   fConcMod
!-----------------------------------------------------------------------------
DOUBLE PRECISION FUNCTION fModulation(t,w,Amax,Amin,f,pl)
IMPLICIT NONE
INTEGER, INTENT(IN):: w
DOUBLE PRECISION, INTENT(IN) :: t, Amax, Amin, f, pl
DOUBLE PRECISION, PARAMETER :: twoPI=6.28318530717958 6
!-----------------------------------------------------------------------------
SELECT CASE (w)
  CASE (1) ! Constant
    fModulation=Amax;
  CASE (2) ! Sine
    fModulation=Amin+(Amax-Amin)*0.5*(1.0+sin(twoPI*(f*t+pl)));
  CASE (3) ! Square
      IF (mod((f*t+pl),1.0) < 0.5 ) THEN
          fModulation=Amax; ELSE; fModulation=Amin;
      END IF
  CASE (4) ! Sawtooth1
     fModulation=Amin+(Amax-Amin)*(f*t+(1-pl)-floor(f*t+(1-pl)));
END SELECT
END FUNCTION fModulation
```

Table E.20 sDiffusion.f90

```
!-----------------------------------------------------------------------
! Subroutine sDiffusion                                                |
! Description: Compute the effective diffusion coefficient base on the temperature. This subroutine only doesn't apply to |
!              isothermal runs, since the dimension of Tau doesn't match.
! Created by Hsu-Wen Hsiao on 8/4/2010.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
SUBROUTINE sDiffusion
USE mDynamic
USE mParameter
USE mBoundary
IMPLICIT NONE
INTEGER :: i, j
DO i=1,ngc
  IF (CBC(1) == 3) THEN
    Deff(rStart-1,i)=rDeff*sqrt((T(rStart-1)/rTemp)*(rMW/MWGas(i)));  ! Danckwerts' BC on left
  END IF
  DO j=rStart,rEnd
    Deff(     j,i)=rDeff*sqrt((T(     j)/rTemp)*(rMW/MWGas(i))); ! reactor
  END DO
  IF (CBC(2) == 3) THEN
    Deff(  rEnd+1,i)=rDeff*sqrt((T(  rEnd+1)/rTemp)*(rMW/MWGas(i))); ! Danckwerts' BC on right
  END IF
END DO
END SUBROUTINE sDiffusion
```

Table E.21 sTau.f90

```
!-----------------------------------------------------------------------
! Subroutine sTau                                                      |
! Description: Compute the diffusion time constant for DDR. This subroutine only doesn't apply to isothermal runs, since the
!              dimension of Tau doesn't match. Make sure the diffusion coeffcient is already updated before calling this
!              subroutine.                                             |
! Created by Hsu-Wen Hsiao on 8/4/2010.                                |
! Copyright 2010 UCSD. All rights reserved.                            |
!-----------------------------------------------------------------------
SUBROUTINE sTau
USE mDynamic
USE mParameter
USE mBoundary
IMPLICIT NONE
INTEGER :: i, j
SELECT CASE (reactorType)
  CASE (1) ! DDR
    DO i=1,ngc
       DO j=rStart,rEnd
          Tau(j,i)=catP*(lz**2)/Deff(j,i);
       END DO
    IF (CBC(1) == 3) THEN ! Danckwerts' BC on left
      !Tau(rStart-2,i)=BCCV(1)/(BCRQ(1)*sqrt(CT(rStart-1)/BCRT(1))*(BCRM(1)/MWGas(i)))); ! time constant for inlet flow
      !Tau(rStart-1,i)=BCCV(1)*lz/Deff(rStart-1,i)/totRCA; ! time constant for external mass transfer
         Tau(rStart-2,i)=BCCV(1)/(BCRQ(1)*sqrt((CT(rStart-1)/BCRT(1))*(BCRM(1)/MWGas(i)))); ! time constant for inlet flow
         Tau(rStart-1,i)=BCCV(1)/totRCA/BCEMTC(1); ! time constant for external mass transfer
         TauB(1,i)=(BCCV(1)/totGV/Tau(rStart-1,i)/dZ-1.0D0/Tau(rStart,i)/(dZ**2))**-1;
    END IF
    IF (CBC(2) == 3) THEN ! Danckwerts' BC on right
      !Tau(rEnd+1,i)=BCCV(2)*lz/Deff(rEnd+1,i)/totRCA; ! time constant for external mass transfer
      !Tau(rEnd+2,i)=BCCV(2)/(BCRQ(2)*sqrt((CT(rEnd+1)/BCRT(2))*(BCRM(2)/MWGas(i)))); ! time constant for inlet flow
         Tau(rEnd+1,i)=BCCV(2)/totRCA/BCEMTC(2); ! time constant for external mass transfer
         Tau(rEnd+2,i)=BCCV(2)/(BCRQ(2)*sqrt((CT(rEnd+1)/BCRT(2))*(BCRM(2)/MWGas(i)))); ! time constant for inlet flow
         TauB(2,i)=(BCCV(2)/totGV/Tau(rEnd+1,i)/dZ-1.0D0/Tau(rEnd,i)/(dZ**2))**-1;
    END IF
    END DO
  CASE (2) ! CSTR
    DO i=1,ngc
      DO j=rStart,rEnd
        Tau(j,i)=totGV*lz/totRCA/Deff(j,i); !<---------------------- here!!
        !! This might be wrong
      END DO
    END DO
END SELECT
END SUBROUTINE sTau
```

Table E.22 sRateConstant.f90

```
!-----------------------------------------------------------------------
! Subroutine sRateConstant
! Description: Compute the rate constant according to the Arrehnuis equation.
! Created by Hsu-Wen Hsiao on 8/4/2010.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
SUBROUTINE sRateConstant
USE mReaction
USE mDynamic
IMPLICIT NONE
INTEGER :: k, j
DO j=rStart,rEnd ! reactions occur in reactor only
  DO k=1,ntr
    RC(k,j)=AC(1,k)*(T(j)**AC(2,k))*EXP(-AC(3,k)/idealGC/T(j));
  END DO
END DO
END SUBROUTINE sRateConstant
```

Table E.23 UserDefineRate.f90

```
!-----------------------------------------------------------------------
! Subroutine sUDRate
! Description: Subroutine to input user-defined rate equation for TransCat.
! Created by Hsu-Wen Hsiao on 8/13/10.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   k: rate constant (vector)
!   C: concentration (vector)
!   T: temperature
!   n: number of component
!   m: number of user-defined rate equation
! Output:
!   UDR: user-defined rate
!-----------------------------------------------------------------------
SUBROUTINE sUDRate(k,nrur,URC,maxGC,totSC,perSite,C,T,n,m)
USE mUserDefine
IMPLICIT NONE
DOUBLE PRECISION, DIMENSION(m), INTENT(IN) :: k
DOUBLE PRECISION, DIMENSION(nrur,6), INTENT(IN) :: URC
DOUBLE PRECISION :: maxGC, totSC
DOUBLE PRECISION, DIMENSION(m) :: perSite
DOUBLE PRECISION, DIMENSION(n), INTENT(IN) :: C
DOUBLE PRECISION :: T
INTEGER, INTENT(IN) :: nrur, n, m
!--- Inpute user define rate equation below this line ------------------------------------------------------------------
DOUBLE PRECISION, DIMENSION(m) :: kappa
DOUBLE PRECISION, DIMENSION(m) :: kO2ads
DOUBLE PRECISION :: maxO2SurCov=5.0D-1 ! maximum O2 surface coverage
!DOUBLE PRECISION, DIMENSION(2) :: perSite=(/1.5D-1, 8.5D-1/) ! percentage of site
IF (m==2) THEN ! two site model (catSF(1)=15%; casSF(2)=85%)
kO2ads(1)=URC(1,1)*((T/4.2315D2)**URC(1,2));
kappa(1)=(perSite(1)-C(5)/maxO2SurCov)/(perSite(1)-C(5));
UDR(1)  =kO2ads(1)*maxGC*C(2)*(perSite(1)-C(5)/maxO2SurCov-kappa(1)*C(4))**2;
kO2ads(2)=URC(2,1)**((T/4.2315D2)**URC(2,2));
kappa(2)=(perSite(2)-C(7)/maxO2SurCov)/(perSite(2)-C(7));
UDR(2)  =kO2ads(2)*maxGC*C(2)*(perSite(2)-C(7)/maxO2SurCov-kappa(2)*C(6))**2;
ELSEIF (m==1) THEN ! single site
kO2ads(1)=URC(1,1)*((T/4.2315D2)**URC(1,2));
kappa(1)=(perSite(1)-C(5)/maxO2SurCov)/(perSite(1)-C(5));
UDR(1)  =kO2ads(1)*maxGC*C(2)*(perSite(1)-C(5)/maxO2SurCov-kappa(1)*C(4))**2;
END IF

!--- Inpute user define rate equation above this line ------------------------------------------------------------------

!-------------------------------- Instruction ----------------------------------
```

Table E.23 Continued

```
!
! Example of user defined oxygen adsorption rate equation in two site LHHW mecahnism for CO oxidation
! Assume there are two user-defined rate equation. (nrur is 2 in the input file)
! The O2 adsorption mechanism are:
!                    k1
! Site 1: O2(g) + 2 S1(s) ---> 2 O*S1(s) (irreversible)
!      r1 = k1 * C_O2 * ( 1 - 2 * C_O*S1 - kappa * C_CO*S1 )^2, where kappa=( 1 - 2 * C_O*S1 )/( 1 - C_O*S1 )
!
!                    k2
! Site 2: O2(g) + 2 S2(s) ---> 2 O-S2(s) (irreversible)
!      r2 = k2 * C_O2 * ( 1 - 2 * C_O*S2 - kappa * C_CO*S2 )^2, where kappa=( 1 - 2 * C_O*S2 )/( 1 - C_O*S2 )
!
! IF the corresponding sequence of component defined by user in file Input.txt is as following
!   Index of component          1          2          3          4          5          6          7          8          9
!   Component                   CO(g),     O2(g),     CO2(g),    CO*S1(s),  O*S1(s),   CO*S2(s),  O*S2(s),   S1(s),    S2(s)
!   Notation in rate equation   C1         C2         C3         C4         C5         C6         C7         C8         C9
!   Notation in Fortran         C(1)       C(2)       C(3)       C(4)       C(5)       C(6)       C(7)       C(8)       C(9)
!
! The rate equation then become,
!   r1 = k1*C2*(1-2*C5-kappa*C4)^2, where kappa=(1-2*C5)/(1-C5)
!   r2 = k2*C2*(1-2*C7-kappa*C6)^2, where kappa=(1-2*C7)/(1-C7)
!
! According to the sequence of component, the mechanism in the input file should be, MechU=
!  0,-1, 0, 2, 0, 0,-2, 0
!  0,-1, 0, 0, 0, 2, 0,-2
!
! The Arrehnius' rate coefficient in the input file should be, URC=     (The dimension of Af and Ar must be 1/sec.)
!     Af         Nf         Ef         Ar         Nr         Er
! 6.26E+000,5.00E-001,0.00E+000,0.00E+000,0.00E+000,0.00E+000  <----- for site 1 (subscript 1 in the following equation)
! 6.26E+000,5.00E-001,0.00E+000,0.00E+000,0.00E+000,0.00E+000  <----- for site 2 (subscript 2 in the following equation)
!
! Because the adsorption process is assumed to be irriversbel, the pre-exponetial factor Ar in the input file must be set as zero.
! The rate constant in this expresstion is: k1=Af_1 * ( T )^ Nf_1 * exp( -Ef_1 /R /T )
!                                           k2=Af_2 * ( T )^ Nf_2 * exp( -Ef_2 /R /T )
!
! The TransCat will translate the mechanism and Arrehnius's rate constants in the input file and put the rate equation in the
! following order: forward rate 1, reverse rate 1, forward rate 2, reverse rate 2, ..........................
!
! TransCat will skip the rate equation with zero rate constant, so only the two forward rate and corresponding rate constant,
! if required, need to be defined. The rate constant k has been computed by the Arrehius equation at temperature T before calling
! this function, so it doesn't need to be recalculated unless you want to use another model. To compute the reaction rate at
! temperature T, the Fortran code looks like this (UDR is the user-defined rate)
!
! DOUBLE PRECISION :: kappa
! kappa       =(1-2*C(5))/(1-C(5))
! UDR(1)=k(1)*C(2)*(1-2*C(5)-kappa*C(4))**2;
```

Table E.23 Continued

```
! kappa      =(1-2*C(7))/(1-C(7))
! UDR(2)=k(2)*C(2)*(1-2*C(7)-kappa*C(6))**2;
!
!-----------------------------------------------------------------------------------------------------------------------
END SUBROUTINE sUDRate
!-----------------------------------------------------------------------------------------------------------------------
!-----------------------------------------------------------------------------------------------------------------------
! Subroutine sUDRateDer                                                                                                 |
! Description: Subroutine to input the first derivative of user-defined rate equation for implicit method of TransCat. |
! Created by Hsu-Wen Hsiao on 8/13/10.                                                                                  |
! Copyright 2010 UCSD. All rights reserved.                                                                             |
!-----------------------------------------------------------------------------------------------------------------------
! Input:
!   k: rate constant (vector)
!   C: concentration (vector)
!   T: temperature
!   n: number of component
!   m: number of user-defined rate equation
! Output:
!   UDRD: user-defined rate derivative
!-----------------------------------------------------------------------------------------------------------------------
SUBROUTINE sUDRateDer(k,C,T,n,m)
USE mUserDefine
IMPLICIT NONE
DOUBLE PRECISION, DIMENSION(m), INTENT(IN) :: k
DOUBLE PRECISION, DIMENSION(n), INTENT(IN) :: C
DOUBLE PRECISION :: T
INTEGER, INTENT(IN) :: n, m
!--- Inpute user define rate derivative below this line -----------------------------------------------------------------
!
DOUBLE PRECISION, DIMENSION(m) :: kappa
nzUDRD(1,1:n)=(/ 0, 1, 0, 1, 1, 0, 0, 0, 0/); ! nonzero derivative of rate one
nzUDRD(2,1:n)=(/ 0, 1, 0, 0, 0, 1, 1, 0, 0/); ! nonzero derivative of rate two

kappa(1) =(1-2*C(5))/(1-C(5));
UDRD(1,2)=   k(1)      *(1-2*C(5)-kappa(1)*C(4))**2;
UDRD(1,4)=-2*k(1)*C(2)*(1-2*C(5)-kappa(1)*C(4))*kappa(1);
UDRD(1,5)= 2*k(1)*C(2)*(1-2*C(5)-kappa(1)*C(4))*(-2+C(4)/((1-C(5))**2));

kappa(2) =(1-2*C(7))/(1-C(7))
UDRD(2,2)=   k(2)      *(1-2*C(7)-kappa(2)*C(6))**2;
UDRD(2,6)=-2*k(2)*C(2)*(1-2*C(7)-kappa(2)*C(6))*kappa(2);
UDRD(2,7)= 2*k(2)*C(2)*(1-2*C(7)-kappa(2)*C(6))*(-2+C(6)/((1-C(7))**2));
!-- Inpute user define rate equation above this line ------------------ Instruction ------------------------------------|
!-----------------------------------------------------------------------------------------------------------------------|
!
! The derivative of reaction rate to each component should be defined in this subroutine. If two user-defined rate equation are
```

Table E.23 Continued

```
!
!   r1 = k1*C2*(1-2*C5-kappa*C4)^2, where kappa=(1-2*C5)/(1-C5)
!   r2 = k2*C2*(1-2*C7-kappa*C6)^2, where kappa=(1-2*C7)/(1-C7)
!
! The first derivative of rate to each of nine components in algebraic notation
!
!   dr1/dC1= 0
! dr1/dC2= k1*(1-2*C5-kappa*C4)^2
!   dr1/dC3= 0
! dr1/dC4=-2*k1*C2*(1-2*C5-kappa*C4)*kappa
! dr1/dC5= 2*k1*C2*(1-2*C5-kappa*C4)*(-2+C4*(1-C5)^-2)
!   dr1/dC6= 0
!   dr1/dC7= 0
!   dr1/dC8= 0
!   dr1/dC9= 0
!
!   dr2/dC1= 0
! dr2/dC2= k2*(1-2*C7-kappa*C6)^2
!   dr2/dC3= 0
!   dr2/dC4= 0
!   dr2/dC5= 0
! dr2/dC6=-2*k2*C2*(1-2*C7-kappa*C6)*kappa
! dr2/dC7= 2*k2*C2*(1-2*C7-kappa*C6)*(-2+C6*(1-C7)^-2)
!   dr2/dC8=
!   dr2/dC9=
!
! To speed up the computation, we only input the nonzero derivative. In Fortran language a logical array is used and the final
! for the first derivative of rate is provided as following. (Note:The logical array, nzUDRD, and double precision array, UDRD,
! have already been defined in the module mUserDefine.)
!
! DOUBLE PRECISION, DIMENSION(m) :: kappa
! nzUDRD=(/ 0, 1, 0, 1, 1, 0, 0, 0, 0  &
!           0, 1, 0, 0, 0, 1, 1, 0, 0 /);
!
! kappa(1) =(1-2*C(5))/(1-C(5));
! UDRD(1,2)=   k(1)     *(1-2*C(5)-kappa*C(4))**2;
! UDRD(1,4)=-2*k(1)*C(2)*(1-2*C(5)-kappa*C(4))*kappa;
! UDRD(1,5)= 2*k(1)*C(2)*(1-2*C(5)-kappa*C(4))*(-2+C(4)/((1-C(5))**2));
!
! kappa(2)=(1-2*C(7))/(1-C(7))
! UDRD(2,2)=   k(2)     *(1-2*C(7)-kappa*C(6))**2;
! UDRD(2,6)=-2*k(2)*C(2)*(1-2*C(7)-kappa*C(6))*kappa;
! UDRD(2,7)= 2*k(2)*C(2)*(1-2*C(7)-kappa*C(6))*(-2+C(6)/((1-C(7))**2));
!
!-----------------------------------------------------------------------------------------------------------------------|
END SUBROUTINE sUDRateDer
```

Table E.23 Continued

```
!-----------------------------------------------------------------------
! Function fUDRate
! Description: Function to input user-defined rate equation for TransCat.
! Created by Hsu-Wen Hsiao on 2/4/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
! Input:
!   k: rate constant (vector)
!   C: concentration (vector)
!   T: temperature
!   n: number of component
!   m: number of user-defined rate equation
! Output:
!   fUserRate: user-defined rate
!-----------------------------------------------------------------------
FUNCTION fUDRate(k,C,T,n,m)
IMPLICIT NONE
DOUBLE PRECISION, DIMENSION(m) :: fUDRate
DOUBLE PRECISION, DIMENSION(m), INTENT(IN) :: k
DOUBLE PRECISION, DIMENSION(n), INTENT(IN) :: C
DOUBLE PRECISION :: T
INTEGER, INTENT(IN) :: n, m
!--- Inpute user define rate equation below this line ------------------
DOUBLE PRECISION :: kappa
kappa      =(1-2*C(5))/(1-C(5))
fUDRate(1)=k(1)*C(2)*(1-2*C(5)-kappa*C(4))**2;
kappa      =(1-2*C(7))/(1-C(7))
fUDRate(2)=k(2)*C(2)*(1-2*C(7)-kappa*C(6))**2;
!--- Inpute user define rate equation above this line ------------------
!-------------------------------------------- Instruction --------------
! Example of user defined oxygen adsorption rate equation in two site LHHW mecahnism for CO oxidation
! Assume there are two user-defined rate equation. (nrur is 2 in the input file)
! The O2 adsorption mechanism are:
!                    k1
! Site 1: O2(g) + 2 S1(s) ---> 2 O*S1(s) (irreversible)
!       r1 = k1 * C_O2 * ( 1 - 2 * C_O*S1 - kappa * C_CO*S1 )^2, where kappa=( 1 - 2 * C_O*S1 )/( 1 - C_O*S1 )
!
!                    k2
! Site 2: O2(g) + 2 S2(s) ---> 2 O-S2(s) (irreversible)
!       r2 = k2 * C_O2 * ( 1 - 2 * C_O*S2 - kappa * C_CO*S2 )^2, where kappa=( 1 - 2 * C_O*S2 )/( 1 - C_O*S2 )
!
! IF the corresponding sequence of component defined by user in file Input.txt is as following
!  Index of component          1          2          3          4          5          6          7          8          9
!  Component                   CO(g),     O2(g),     CO2(g),    CO*S1(s),  O*S1(s),   CO*S2(s),  O*S2(s),   S1(s),    S2(s)
!  Notation in rate equation   C1         C2         C3         C4         C5         C6         C7         C8         C9
!  Notation in Fortran         C(1)       C(2)       C(3)       C(4)       C(5)       C(6)       C(7)       C(8)       C(9)
```

Table E.23 Continued

```
!
! The rate equation then become,
!   r1 = k1*C2*(1-2*C5-kappa*C4)^2, where kappa=(1-2*C5)/(1-C5)
!   r2 = k2*C2*(1-2*C7-kappa*C6)^2, where kappa=(1-2*C7)/(1-C7)
!
! According to the sequence of component, the mechanism in the input file should be, MechU=
!  0,-1, 0, 2, 0, 0,-2, 0
!  0,-1, 0, 0, 0, 2, 0,-2
!
! The Arrehnius' rate coefficient in the input file should be, URC=     (The dimension of Af and Ar must be 1/sec.)
!     Af      Nf      Ef      Ar      Nr      Er
!  6.26E+000,5.00E-001,0.00E+000,0.00E+000,0.00E+000,0.00E+000  <----- for site 1 (subscript 1 in the following equation)
!  6.26E+000,5.00E-001,0.00E+000,0.00E+000,0.00E+000,0.00E+000  <----- for site 2 (subscript 2 in the following equation)
!
! Because the adsorption process is assumed to be irriversbel, the pre-exponetial factor Ar in the input file must be set as zero.
! The rate constant in this expresstion is: k1=Af_1 * ( T )^ Nf_1 * exp( -Ef_1 /R /T )
!                                           k2=Af_2 * ( T )^ Nf_2 * exp( -Ef_2 /R /T )
!
! The TransCat will translate the mechanism and Arrehnius's rate constants in the input file and put the rate equation in the
! following order: forward rate 1, reverse rate 1, forward rate 2, reverse rate 2, .....................
!
! TransCat will skip the rate equation with zero rate constant, so only the two forward rate and corresponding rate constant,
! if required, need to be defined. The rate constant k has been computed by the Arrehius equation at temperature T before calling
! this function, so it doesn't need to be recalculated unless you want to use another model. To compute the reaction rate at
! temperature T, the Fortran code looks like this
!
! DOUBLE PRECISION :: kappa
! kappa      =(1-2*C(5))/(1-C(5))
! fUserRate(1)=k(1)*C(2)*(1-2*C(5)-kappa*C(4))**2;
! kappa      =(1-2*C(7))/(1-C(7))
! fUserRate(2)=k(2)*C(2)*(1-2*C(7)-kappa*C(6))**2;
!
!-----------------------------------------------------------------------------------------------------------------------|
END FUNCTION fUDRate
```

Table E.24 sReaction.f90

```
!-----------------------------------------------------------------------------|
! Subroutine sReaction                                                        |
! Description: Subroutine to compute reaction rate. Please make sure the rate constant is updated before calling this subroutine. |
! Created by Hsu-Wen Hsiao on 2/4/09.                                         |
! Copyright 2009 UCSD. All rights reserved.                                   |
!-----------------------------------------------------------------------------|
SUBROUTINE sReaction
USE mReaction
USE mDynamic
USE mUserDefine
USE mParameter
IMPLICIT NONE
!INTERFACE
!  FUNCTION fUserRate(k,C,n,m)
!  IMPLICIT NONE
!  DOUBLE PRECISION, DIMENSION(m) :: fUserRate
!  DOUBLE PRECISION, DIMENSION(m), INTENT(IN) :: k
!  DOUBLE PRECISION, DIMENSION(n), INTENT(IN) :: C
!  DOUBLE PRECISION :: T
!  INTEGER, INTENT(IN) :: n, m
!  END FUNCTION fUserRate
!END INTERFACE
INTEGER :: i, j, k, l
!-----------------------------------------------------------------------------|
DO j=rStart,rEnd ! compute rate
  DO k=erStart,erEnd ! compute elementary rate
    R(k,j)=RC(k,j);
    DO l=RI(k),RI(k+1)-1
      R(k,j)=R(k,j)*C(j,RL(l));
    END DO
  END DO
  DO k=arStart,arEnd ! compute algebraic rate
    R(k,j)=RC(k,j);
    DO l=RI(k),RI(k+1)-1
      R(k,j)=R(k,j)*(C(j,RL(l))**RP(l));
    END DO
  END DO
  IF (nrur > 0) THEN
    CALL sUDRate(RC(urStart:urEnd,j),nrur,URC,maxGC,totSC,catSF,C(j,1:ntc),T(j),ntc,nudr)
      R(urStart:urEnd,j)=UDR;
  END IF
END DO
DO i=gStart,min(gEnd,iEnd) ! This loop computes alpha*S*R for gas equation. The integer iEnd is embedded ecm algorithm
  DO j=rStart,rEnd         ! for independent component. Only the reaction part of indenpendent components is required
    NC(j,i)=0.0D0;         ! to be computed in ecm algorithm.
    DO k=SYI(i),SYI(i+1)-1
      NC(j,i)=NC(j,i)+SY(k)*R(SYL(k),j);
```

Table E.24 Continued

```
    END DO
    NC(j,i)=alpha*NC(j,i);
  END DO
END DO
DO i=sStart,min(cEnd,iEnd) ! compute S*R for solid equation
  DO j=rStart,rEnd
    NC(j,i)=0.0D0;
    DO k=SYI(i),SYI(i+1)-1
      NC(j,i)=NC(j,i)+SY(k)*R(SYL(k),j);
    END DO
  END DO
END DO
END SUBROUTINE sReaction
!-----------------------------------------------------------------------
! Subroutine sDRxnDC
! Description: Subroutine to compute the first derivative of reaction for the Jacabian matrix JY. Please make sure the rate
!              constants have been updated before calling this subroutine.
!
! Created by Hsu-Wen Hsiao on 8/9/2010.
! Copyright 2010 UCSD. All rights reserved.
!-----------------------------------------------------------------------
SUBROUTINE sDRxnDC
USE mReaction
USE mDynamic
USE mUserDefine
IMPLICIT NONE
!CHARACTER(LEN=256), EXTERNAL :: fInsInt, fInsInt2
INTEGER :: i, j, k
DO j=1,ngz ! compute the first derivative of rate ------------------------------------------------------------------
  DO i=1,JYI(ntc+1)-1
    JRY(i,j)=JRC(i)*RC(JYL(i),j);
    DO k=JRI(i),JRI(i+1)-1
      IF     (JYL(i) >= erStart .AND. JYL(i) <= erEnd) THEN ! derivative of elementary rate
        JRY(i,j)=JRY(i,j)*C(j,JRL(k));
      ELSEIF (JYL(i) >= arStart .AND. JYL(i) <= arEnd) THEN ! derivative of algebraic rate
        JRY(i,j)=JRY(i,j)*(C(j,JRL(k))**JRP(k));
      END IF
    END DO
  END DO
  IF (nrur > 0) THEN ! user-define first derivative of rate equation ----------------------------------------------
    CALL sUDRateDer(RC(urStart:urEnd,j),C(j,1:ntc),T(j),ntc,nudr)
      JUR(1:nudr,1:ntc,j)=UDRD;
  END IF
END DO
END SUBROUTINE sDRxnDC
```

Table E.25 sEnergy.f90

```
!-----------------------------------------------------------------------
! Subroutine sEnergy
! Description: Solve the energy equation for the temperature distribution in the reactor.
! Created by Hsu-Wen Hsiao on 4/19/09.
! Copyright 2009 UCSD. All rights reserved.
!-----------------------------------------------------------------------
SUBROUTINE sEnergy
USE mDynamic
IMPLICIT NONE
INTEGER :: j
DOUBLE PRECISION :: a
! The purpose of this subroutine is update T(1:ngz) by solving the energy equation.
! The following is a testing subroutine of changing the tenth decimal of T in a periodic manner.
CALL RANDOM_NUMBER(a)
a=FLOOR(a*10);
DO j=1,ngz
  T(j)=1.0D-2*FLOOR(1.0D2*T(j))+1.0D-10*(a+j);
END DO
!IF (tRun == tiRun) THEN; CALL sWrite("T",6,1); END IF
END SUBROUTINE sEnergy
```

Table E.26 sTemperature.f90

```
!-----------------------------------------------------------------------
! Subroutine sTemperature                                              |
! Description: User defined temperature function for reactor. A example temperature ramp subroutine is geven as following.  |
! Created by Hsu-Wen Hsiao on 4/19/09.                                 |
! Copyright 2009 UCSD. All rights reserved.                            |
!-----------------------------------------------------------------------
SUBROUTINE sTemperature
USE mDynamic
IMPLICIT NONE
DOUBLE PRECISION, PARAMETER :: TwoPi=6.28318530717958 6
DOUBLE PRECISION :: T_Max=400, T_Min=300, T_Frequency, T_PhaseLag
DOUBLE PRECISION :: T_Ramp
INTEGER :: i
! User Input ----------------------------------------------------------|
T_Max=400;       ! maximum temperature
T_Min=300;       ! minimum temperature
T_Frequency=1.0;  ! frequency
T_PhaseLag=0.0;   ! phase lag
T_Ramp=T_Min+(T_Max-T_Min)*0.5*(1.0+sin(TwoPi*(T_Frequency*tRun+T_PhaseLag))); ! sine wave modulation
! End User Input ------------------------------------------------------|
SELECT CASE (CBC(1)) ! left boundary ----------------------------------|
  CASE (1) ! boundary chamber on the left
    T(rStart-1)=T_Ramp
  CASE (2) ! boundary chamber on the left
    T(rStart-1)=T_Ramp
  CASE (3) ! inlet flow and boundary chamber on the left
    T(rStart-2)=T_Ramp
    T(rStart-1)=T_Ramp
END SELECT
  DO i=rStart,rEnd; ! reactor -------------------------------------------|
    T(i)=T_Ramp
  END DO
SELECT CASE (CBC(2)) ! right boundary ---------------------------------|
  CASE (1) ! boundary chamber on the right
    T(rEnd+1)=T_Ramp
  CASE (2) ! boundary chamber on the right
    T(rEnd+1)=T_Ramp
  CASE (3) ! boundary chamber and inlet flow on the right
    T(rEnd+1)=T_Ramp
    T(rEnd+2)=T_Ramp
END SELECT
END SUBROUTINE sTemperature
```

Table E.27 PlotRunTxt.m

```
%--------------------------------------------------------------------------%
% PlotRunTxt.m                                                  01/27/2010 %
% Description: Plot the simulation result of all runs in the file Run.txt.%
% Created by Hsu-Wen Hsiao on 7/29/09.                                     %
% Copyright 2009 UCSD. All rights reserved.                                %
%--------------------------------------------------------------------------%
clear;
% identify working path and directory ------------------------------------%
GraphicDir=cd;                          % path of graphic functions
cd('..');                               % moving up one directory
TransCatDir=cd;                         % this should be the path of TransCat
cd(GraphicDir);                         % go back to the graphic directory
path([cd filesep 'Toolbox'],path);    % add functions in Toolbox folder
RunTxtPath=[TransCatDir filesep 'Run.txt']; % location of file Run.txt
[nRun,RunList]=readRun(RunTxtPath);   % read file Run.txt
nFigure=0;                              % number of figures
for iRun=1:nRun                         % plot all runs in run.txt
  run=strtrim(RunList(iRun,:));       % name of this run
  RunDir=[TransCatDir filesep run];   % directory of each run
  % read and process output data ---------------------------------------%
  outputXML=xmlread([TransCatDir filesep run filesep 'Output.xml']);%Output
  reactorType=getIntegerR0(outputXML,'reactorType'); % reactorType
  thermalType=getIntegerR0(outputXML,'thermalType'); % thermalType
  nts=getIntegerR0(outputXML,'nts')+1;% number of time step + initial step
  ngz=getIntegerR0(outputXML,'ngz');  % number of grid in z-direction
  ntc=getIntegerR0(outputXML,'ntc');  % number of total component
  ngc=getIntegerR0(outputXML,'ngc');  % number of gas component
  ntr=getIntegerR0(outputXML,'ntr');  % number of elementary rate
  ncr=getIntegerR0(outputXML,'ncr');  % number of chemical reaction
  Z=getDoubleR1(outputXML,'Z');   % dimensionless coordinate Z
  dZ=getDoubleR0(outputXML,'dZ');     % maximum gas concentration
  lz=getDoubleR0(outputXML,'lz');     % pellet thickness
  RIV=getIntegerR1(outputXML,'RIV');  % Reaction identity vector
  RIM=getIntegerR2(outputXML,'RIM');  % Reaction identity matrix
  Mech=getIntegerR2(outputXML,'Mech');% Reaction Mechanism
  maxGC=getDoubleR0(outputXML,'maxGC');% maximum gas concentration
  totCS=getDoubleR0(outputXML,'totCS');% total catalytic sites
  idealGC=getDoubleR0(outputXML,'idealGC');% ideal gas constant
  iniT=getDoubleR0(outputXML,'iniT'); % initial temperature
  Component=getStringR1(outputXML,'Component'); % components
  CBC=getIntegerR1(outputXML,'CBC');  % Reaction identity vector
  rStart=getIntegerR0(outputXML,'rStart'); % starting grid of reactor
  rEnd=getIntegerR0(outputXML,'rEnd');% ending grid of reactor
  % read other text files ----------------------------------------------%
  C=readConc(RunDir,run,ngz,nts,ntc); % read concentration
  R=readRate(RunDir,run,ngz,nts,ntr); % read reaction rate
  t=readTime(RunDir,run);             % read time
  if thermalType ~= 1
    T=readTemp(RunDir,run);           % read T except for isothermal
  end
  % process data for ploting ------------------------------------------%
  BCZ=[Z(rStart)-0.5*dZ Z(rEnd)+0.5*dZ];
  switch CBC(1)
      case 1 % Dirichlet
          BCG(1)=rStart-1;
      case 2 % Neumann
          BCG(1)=rStart;
      case 3 % Danckwerts
          BCG(1)=rStart-1;
  end
    switch CBC(2)
      case 1 % Dirichlet
          BCG(2)=rEnd+1;
      case 2 % Neumann
          BCG(2)=rEnd;
      case 3 % Danckwerts
          BCG(2)=rEnd+1;
  End
```

Table E.27 Continued

```
%Z=lz*Z; % dimensional coordinate z
  if thermalType == 1 % isothermal run
    P=maxGC*C(1:ngz,1:nts,1:ngc)*idealGC*iniT; % C convert to pressure
  else % non-isothermal case
    P=zeros(ngz,nts,ngc);
    for i=1:ngc
        P(1:ngz,1:nts,i)=maxGC*C(1:ngz,1:nts,i)*idealGC.*T(1:ngz,1:nts);
    end
  end
  % 2D image plot of gas pressure and surface coverage -------------------%
  nFigure=nFigure+1;
  figure(nFigure);
  clf(nFigure);
  set(nFigure,'Name','2D Image Plot of Gas Pressure and Surface Coverage');
  [m,n]=defSubplot(ntc); % define figure grid
  for i=1:ngc
    subplot(m,n,i);
    imgGas2D(t,Z(rStart:rEnd),P(rStart:rEnd,1:nts,i),Component(i,:));
  end
  for i=ngc+1:ntc
    subplot(m,n,i);
    imgSolid2D(t,Z(rStart:rEnd),C(rStart:rEnd,1:nts,i),Component(i,:));
  end
  saveas(nFigure,[RunDir filesep run '_AllComp2D.fig']);
  % 2D image plot of reaction rate ---------------------------------------%
  nFigure=nFigure+1;
  figure(nFigure);
  clf(nFigure);
  set(nFigure,'Name','2D Image Plot of Reaction Rate');
  [m,n]=defSubplot(ntr); % define figure grid
  for i=1:ntr
    subplot(m,n,i);
    imgRate2D(t,Z(rStart:rEnd),R(rStart:rEnd,1:nts,i),...
        sign(RIV(i))*Mech((abs(RIV(i))),:),Component);
  end
  saveas(nFigure,[RunDir filesep run '_AllRate2D.fig']);
  % 2D plot of all gas pressure on the boundary --------------------------%
  nFigure=nFigure+1;
  figure(nFigure);
  clf(nFigure);
  set(nFigure,'Name','All Gas Pressure on Boundary vs Time');
  [m,n]=defSubplot(2); % define figure grid
  for i=1:2
    subplot(m,n,i);
    plotAllGasZj(t,Z,BCG(i),ngc,P,Component(1:ngc,:),BCZ(i));
  end
  saveas(nFigure,[RunDir filesep run '_BoundaryAllGas.fig']);
  % 2D plot of each gas pressure on both boundaries ----------------------%
  nFigure=nFigure+1;
  figure(nFigure);
  clf(nFigure);
  set(nFigure,'Name','One Gas Pressure on Both Boundaries vs Time');
  [m,n]=defSubplot(ngc); % define figure grid
  for i=1:ngc
    subplot(m,n,i);
    plotOneGas2B(t,Z(BCG),BCG,P(BCG,:,i),Component(i,:),BCZ);
  end
  saveas(nFigure,[RunDir filesep run '_BoundaryOneGas.fig']);
end
clear i m n iRun run outputXML
```

Table E.28 PlotOneRun.m

```
%-------------------------------------------------------------------------%
% PlotOneRun.m                                                  01/26/2010 %
% Description: Plot one simulation result of TransCat.                     %
% Created by Hsu-Wen Hsiao on 7/29/09.                                     %
% Copyright 2009 UCSD. All rights reserved.                                %
%-------------------------------------------------------------------------%
% Editor's Notes:
%   1. line 48 and 57 for non-isothermal case
%   2.
%-------------------------------------------------------------------------%
clear;%clf;
%-------------------- Input Parameter below This Line --------------------%
%run='Ne_Diffusion';
%run='CO_Adsorption';
%run='Fisher';
run='Fig2_Ne_Diffusion_308K';
%-------------------- Input Parameter above This Line --------------------%
% identify working path and directory -------------------------------------%
GraphicDir=cd;                          % path of graphic functions
cd('..');                               % moving up one directory
TransCatDir=cd;                         % this should be the path of TransCat
cd(GraphicDir);                         % go back to the graphic directory
path([cd filesep 'Toolbox'],path);    % add functions in Toolbox folder
RunDir=[TransCatDir '/' run];         % data folder of this run
% read and process output data --------------------------------------------%
outputXML=xmlread([TransCatDir filesep run filesep 'Output.xml']);%read xml
reactorType=getIntegerR0(outputXML,'reactorType'); % reactorType
thermalType=getIntegerR0(outputXML,'thermalType'); % thermalType
nts=getIntegerR0(outputXML,'nts')+1;   % number of time step + initial step
ngz=getIntegerR0(outputXML,'ngz');     % number of grid in z-direction
ntc=getIntegerR0(outputXML,'ntc');     % number of total component
ngc=getIntegerR0(outputXML,'ngc');     % number of gas component
ntr=getIntegerR0(outputXML,'ntr');     % number of elementary rate
ncr=getIntegerR0(outputXML,'ncr');     % number of chemical reaction
Z=getDoubleR1(outputXML,'Z');      % dimensionless coordinate Z
dZ=getDoubleR0(outputXML,'dZ');        % maximum gas concentration
lz=getDoubleR0(outputXML,'lz');        % pellet thickness
RIV=getIntegerR1(outputXML,'RIV');     % Reaction identity vector
RIM=getIntegerR2(outputXML,'RIM');     % Reaction identity matrix
Mech=getIntegerR2(outputXML,'Mech');   % Reaction Mechanism
maxGC=getDoubleR0(outputXML,'maxGC');  % maximum gas concentration
totCS=getDoubleR0(outputXML,'totCS');  % total catalytic sites
idealGC=getDoubleR0(outputXML,'idealGC');% ideal gas constant
iniT=getDoubleR0(outputXML,'iniT');    % initial temperature
Component=getStringR1(outputXML,'Component'); % components
CBC=getIntegerR1(outputXML,'CBC');     % Reaction identity vector
rStart=getIntegerR0(outputXML,'rStart');% starting grid of reactor
rEnd=getIntegerR0(outputXML,'rEnd');   % ending grid of reactor
% read other text files ---------------------------------------------------%
C=readConc(RunDir,run,ngz,nts,ntc);    % read concentration
R=readRate(RunDir,run,ngz,nts,ntr);    % read elementary rate
t=readTime(RunDir,run);                % read time
if thermalType == 1                    % isothermal run
  P=maxGC*C(1:ngz,1:nts,1:ngc)*idealGC*iniT; % convert C to pressure
else                                   % non-isothermal case
  P=zeros(ngz,nts,ngc);
  for i=1:ngc
    P(1:ngz,1:nts,i)=maxGC*C(1:ngz,1:nts,i)*idealGC.*T(1:ngz,1:nts);
  end
end
% process data for ploting ------------------------------------------------%
BCZ=[Z(rStart)-0.5*dZ Z(rEnd)+0.5*dZ];
switch CBC(1)
  case 1 % Dirichlet
    BCG(1)=rStart-1;
  case 2 % Neumann
    BCG(1)=rStart;
  case 3 % Danckwerts
```

Table E.28 Continued

```
    BCG(1)=rStart-1;
end
switch CBC(2)
  case 1 % Dirichlet
    BCG(2)=rEnd+1;
  case 2 % Neumann
    BCG(2)=rEnd;
  case 3 % Danckwerts
    BCG(2)=rEnd+1;
end
if thermalType == 1 % isothermal run
  P=maxGC*C(1:ngz,1:nts,1:ngc)*idealGC*iniT; % C convert to pressure
else % non-isothermal case
  P=zeros(ngz,nts,ngc);
  for i=1:ngc
    P(1:ngz,1:nts,i)=maxGC*C(1:ngz,1:nts,i)*idealGC.*T(1:ngz,1:nts);
  end
end
% generate plots ------------------------------------------------------%
nFigure=0; % number of figures
% 2D image plot of gas pressure and surface coverage ------------------%
nFigure=nFigure+1;
figure(nFigure);
clf(nFigure);
set(nFigure,'Name','2D Image Plot of Gas Pressure and Surface Coverage');
[m,n]=defSubplot(ntc); % define figure grid
for i=1:ngc
  subplot(m,n,i);
  imgGas2D(t,Z(rStart:rEnd),P(rStart:rEnd,1:nts,i),Component(i,:));
end
for i=ngc+1:ntc
  subplot(m,n,i);
  imgSolid2D(t,Z(rStart:rEnd),C(rStart:rEnd,1:nts,i),Component(i,:));
end
saveas(nFigure,[RunDir filesep run '_AllComp2D.fig']);
% 2D image plot of reaction rate --------------------------------------%
nFigure=nFigure+1;
figure(nFigure);
clf(nFigure);
set(nFigure,'Name','2D Image Plot of Reaction Rate');
[m,n]=defSubplot(ntr); % define figure grid
for i=1:ntr
  subplot(m,n,i);
  imgRate2D(t,Z(rStart:rEnd),R(rStart:rEnd,1:nts,i),...
      sign(RIV(i))*Mech((abs(RIV(i))),:),Component);
end
saveas(nFigure,[RunDir filesep run '_AllRate2D.fig']);
% 2D plot of all gas pressure on the boundary -------------------------%
nFigure=nFigure+1;
figure(nFigure);
clf(nFigure);
set(nFigure,'Name','All Gas Pressure on Each Boundary vs Time');
[m,n]=defSubplot(2); % define figure grid
for i=1:2
  subplot(m,n,i);
  plotAllGasZj(t,Z,BCG(i),ngc,P,Component(1:ngc,:),BCZ(i));
end
saveas(nFigure,[RunDir filesep run '_BoundaryAllGas.fig']);
% 2D plot of each gas pressure on both boundaries ---------------------%
nFigure=nFigure+1;
figure(nFigure);
clf(nFigure);
set(nFigure,'Name','One Gas Pressure on Both Boundaries vs Time');
[m,n]=defSubplot(ngc); % define figure grid
for i=1:ngc
  subplot(m,n,i);
  plotOneGas2B(t,Z(BCG),BCG,P(BCG,:,i),Component(i,:),BCZ);
end
```

Table E.28 Conti

```
saveas(nFigure,[RunDir filesep run '_BoundaryOneGas.fig']);
clear i m n nFigure GraphicDir TransCatDir RunDir outputXML
% 3D surface plot of gas pressure and surface coverage -------------------%
nFigure=nFigure+1;
figure(nFigure);
clf(nFigure);
set(nFigure,'Name','3D Surface Plot of Gas Pressure and Surface Coverage');
[m,n]=defSubplot(ntc); % define figure grid
for i=1:ngc
  subplot(m,n,i);
  surfGas3D(t,Z,P(:,:,i),Component(i,:));
end
for i=ngc+1:ntc
  subplot(m,n,i);
  surfSolid3D(t,Z,C(:,:,i),Component(i,:));
end
saveas(nFigure,[RunDir filesep run '_AllComp3D.fig']);
```

Table E.29 AnimateConcentration.m

```
%--------------------------------------------------------------------------%
% AnimateConcentration.m                                                   %
% Description: Animate the gas pressure and surface coverage with time.   %
% Created by Hsu-Wen Hsiao on 10/28/10.                                    %
% Copyright 2010 UCSD. All rights reserved.                                %
%--------------------------------------------------------------------------%
clear;
%-------------------- Input Parameter below This Line --------------------%
%run='Fig4_CO_Oxidation_423K';
%run='Fig3_CO_Adsorption_423K';
run='Fig2_Ne_Diffusion_308K';
lw=1.5;
showLegend=0; % use 1 to show legend, may slow down animation
%-------------------- Input Parameter above This Line --------------------%
% identify working path and directory -------------------------------------%
GraphicDir=cd;                          % path of graphic functions
cd('..');                               % moving up one directory
TransCatDir=cd;                         % the path of TransCat
cd(GraphicDir);                         % go back to the graphic directory
path([cd filesep 'Toolbox'],path);      % add functions in Toolbox folder
RunDir=[TransCatDir '/' run];
% read and process output data --------------------------------------------%
outputXML=xmlread([TransCatDir filesep run filesep 'Output.xml']);%read xml
reactorType=getIntegerR0(outputXML,'reactorType'); % reactorType
thermalType=getIntegerR0(outputXML,'thermalType'); % thermalType
nts=getIntegerR0(outputXML,'nts')+1;   % number of time step + initial step
ngz=getIntegerR0(outputXML,'ngz');     % number of grid in z-direction
ntc=getIntegerR0(outputXML,'ntc');     % number of total component
ngc=getIntegerR0(outputXML,'ngc');     % number of gas component
nsc=getIntegerR0(outputXML,'nsc');     % number of
ncs=getIntegerR0(outputXML,'ncs');     % number of
nce=getIntegerR0(outputXML,'nce');     % number of
ntr=getIntegerR0(outputXML,'ntr');     % number of elementary rate
ncr=getIntegerR0(outputXML,'ncr');     % number of chemical reaction
gStart=getIntegerR0(outputXML,'gStart');
gEnd=getIntegerR0(outputXML,'gEnd');
sStart=getIntegerR0(outputXML,'sStart');
sEnd=getIntegerR0(outputXML,'sEnd');
cStart=getIntegerR0(outputXML,'cStart');
cEnd=getIntegerR0(outputXML,'cEnd');
catSF=getDoubleR1(outputXML,'catSF');
Z=getDoubleR1(outputXML,'Z');      % dimensionless coordinate Z
dZ=getDoubleR0(outputXML,'dZ');        % maximum gas concentration
lz=getDoubleR0(outputXML,'lz');        % pellet thickness
```

Table E.29 Continued

```
RIV=getIntegerR1(outputXML,'RIV');     % Reaction identity vector
RIM=getIntegerR2(outputXML,'RIM');     % Reaction identity matrix
Mech=getIntegerR2(outputXML,'Mech');   % Reaction Mechanism
M=getIntegerR2(outputXML,'M');   % Reaction Mechanism
maxGC=getDoubleR0(outputXML,'maxGC');  % maximum gas concentration
totCS=getDoubleR0(outputXML,'totCS');  % total catalytic sites
idealGC=getDoubleR0(outputXML,'idealGC');% ideal gas constant
iniT=getDoubleR0(outputXML,'iniT');    % initial temperature
Component=getStringR1(outputXML,'Component'); % components
CBC=getIntegerR1(outputXML,'CBC');     % Reaction identity vector
rStart=getIntegerR0(outputXML,'rStart');% starting grid of reactor
rEnd=getIntegerR0(outputXML,'rEnd');   % ending grid of reactor
% read other text files ------------------------------------------------%
C=readConc(RunDir,run,ngz,nts,ntc);    % read concentration
R=readRate(RunDir,run,ngz,nts,ntr);    % read elementary rate
t=readTime(RunDir,run);                % read time
if thermalType == 1                    % isothermal run
  P=maxGC*C(1:ngz,1:nts,1:ngc)*idealGC*iniT; % C convert to pressure
else
% non-isothermal case
  P=zeros(ngz,nts,ngc);
  for i=1:ngc
    P(1:ngz,1:nts,i)=maxGC*C(1:ngz,1:nts,i)*idealGC.*T(1:ngz,1:nts);
  end
end
% process data for ploting ---------------------------------------------%
BCZ=[Z(rStart)-0.5*dZ Z(rEnd)+0.5*dZ];
switch CBC(1)
  case 1 % Dirichlet
    BCG(1)=rStart-1;
  case 2 % Neumann
    BCG(1)=rStart;
  case 3 % Danckwerts
    BCG(1)=rStart-1;
end
switch CBC(2)
  case 1 % Dirichlet
    BCG(2)=rEnd+1;
  case 2 % Neumann
    BCG(2)=rEnd;
  case 3 % Danckwerts
    BCG(2)=rEnd+1;
end
if thermalType == 1 % isothermal run
  P=maxGC*C(1:ngz,1:nts,1:ngc)*idealGC*iniT; % C convert to pressure
else % non-isothermal case
  P=zeros(ngz,nts,ngc);
  for i=1:ngc
    P(1:ngz,1:nts,i)=maxGC*C(1:ngz,1:nts,i)*idealGC.*T(1:ngz,1:nts);
  end
end
% generate plots -------------------------------------------------------%
nFigure=0; % number of figures
nSubplot=ncs+1; % number of subplot
hSubplot=zeros(1,nSubplot); % handle of subplot
hLine=zeros(1,ntc);
currentColormap=jet;
nColorColormap=size(currentColormap,1);
hColor=zeros(ntc,3);
for i=gStart:gEnd
  if ngc>1
    k=floor((i-1)/(ngc-1)*(nColorColormap-1))+1;
  elseif ngc==1
    k=1;
  else
    break
  end
  hColor(i,1:3)=currentColormap(k,1:3);
```

Table E.29 Continued

```
end
for i=sStart:sEnd
  if nsc>1
    k=floor((i-1-ngc)/(nsc-1)*(nColorColormap-1))+1;
  elseif nsc==1
    k=1;
  else
    break
  end
  hColor(i,1:3)=currentColormap(k,1:3);
end
for i=cStart:cEnd
  hColor(i,1:3)=[0 0 0];
end
lLimit=0-2*dZ;
rLimit=1+2*dZ;
% Animation of concentration ------------------------------------------------%
nFigure=nFigure+1;
figure(nFigure);
clf(nFigure);
set(nFigure,'Name','Animation of Gas Pressure and Surface Coverage');
hSubplot(1)=subplot(nSubplot,1,1);
hTitle(1)=title(sprintf('Gas Pressure (Pa), t = %g sec',t(1)));
maxP=max(max(max(P)));
ylim([0 maxP*1.1]);
xLim([lLimit rLimit]);
for i=gStart:gEnd
  hLine(i)=line(Z(BCG(1):BCG(2)),P(BCG(1):BCG(2),1,i));
  set(hLine(i),'Color',hColor(i,1:3),'LineWidth',lw);
end
if showLegend==1
  legend(hSubplot(1),Component(1:ngc,:),'Location','EastOutside');
end
hGasLineLB=line([0 0],[0 maxP]);
hGasLineRB=line([1 1],[0 maxP]);
switch CBC(1)
  case 1 % Dirichlet
    set(hGasLineLB,'LineStyle','--','Color','k')
  case 2 % Neumann
    set(hGasLineLB,'LineStyle','-', 'Color','k')
  case 3 % Danckwerts
    set(hGasLineLB,'LineStyle','--','Color','k')
end
switch CBC(2)
  case 1 % Dirichlet
    set(hGasLineRB,'LineStyle','--','Color','k')
  case 2 % Neumann
    set(hGasLineRB,'LineStyle','-', 'Color','k')
  case 3 % Danckwerts
    set(hGasLineRB,'LineStyle','--','Color','k')
end
for iSurfacePlot=1:ncs
  hSubplot(iSurfacePlot+1)=subplot(nSubplot,1,iSurfacePlot+1);
  hTitle(iSurfacePlot+1)=title(['Surface coverage on ' ...
                                Component(sEnd+iSurfacePlot,:)]);
  maxC=catSF(iSurfacePlot);
  ylim([0 maxC]);
  xLim([lLimit rLimit]);
  nComp=0;
  for i=sStart:cEnd
    for k=nce:-1:(nce-ncs+1)
      if M(k,i) >0
        iSite=k-(nce-ncs);
        break
      end
    end
    if iSite == iSurfacePlot
      nComp=nComp+1;
```

Table E.29 Continued

```
      legendL(1,nComp)=i;
      hLine(i)=line([0;Z(rStart:rEnd);1],...
                    [spline(Z(rStart:rStart+2),C(rStart:rStart+2,1,i),0);...
                     C(rStart:rEnd,1,i);...
                     spline(Z(rEnd-2:rEnd),C(rEnd-2:rEnd,1,i),1)]);
      set(hLine(i),'Color',hColor(i,1:3),'LineWidth',lw);
    end
  end
  hSolidLineLB=line([0 0],[0 maxC],'Color','k');
  hSolidLineRB=line([1 1],[0 maxC],'Color','k');
  if showLegend==1
    legend(hSubplot(iSurfacePlot+1),Component(legendL(1,1:nComp),:),...
           'Location','EastOutside');
  end
end
for it=2:nts
  set(get(hSubplot(1),'Title'),'String',...
  sprintf('Gas Pressure (Pa), t = %6.2f sec',t(it)));
  for i=gStart:gEnd
    set(hLine(i),'YData',P(BCG(1):BCG(2),it,i));
  end
  for i=sStart:cEnd
    set(hLine(i),'YData',...
        [spline(Z(rStart:rStart+2),C(rStart:rStart+2,it,i),0);...
         C(rStart:rEnd,it,i);...
         spline(Z(rEnd-2:rEnd),C(rEnd-2:rEnd,it,i),1)]);
  end
  pause(0.01)
end
```

Table E.30 PlotAllGasZj.m

```
%-------------------------------------------------------------------------%
% plotAllGasZj.m                                                          %
% Description: This function plot of all gas pressure at Z(j).           %
% Input:                                                                  %
%   t: vector of time                                                     %
%   z: vector of space                                                    %
%   j: j'th component of Z                                                %
%   ngc: number of gas component                                          %
%   P3D: 3D array of gas pressure (ngz x nts x ngc)                       %
%   comp: string vector of component (ngc)                                %
% Output:                                                                 %
%   none                                                                  %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function plotAllGasZj(t,z,zj,ngc,P3D,comp,BCZ)
% set figure properties -------------------------------------------------%
fs=10;         % FontSize
fw='Normal';   % FontWeight: Bold, Normal, etc...
lw=1.2;        % linewidth
P2D(1:length(t),1:ngc)=P3D(zj,:,1:ngc); % convert P from 3D to 2D array
% plot ------------------------------------------------------------------%
H=plot(t,P2D);
xlabel('t (sec)','FontSize',fs,'FontWeight',fw);
ylabel('P (Pa) ','FontSize',fs,'FontWeight',fw);
title(['Gas Pressure at z=' num2str(round(BCZ)) ' (dimensionless)']...
       ,'FontSize',fs,'FontWeight',fw);
legend(comp(1:ngc,:),'Location','NorthEastOutside','Orientation',...
    'vertical');
for i=1:ngc
  set(H(i),'LineWidth',lw);
  set(get(H(i),'Parent'),'FontSize',fs,'FontWeight',fw);
end
%-------------------------------------------------------------------------%
```

Table E.31 PlotOneGas2B.m

```
%-------------------------------------------------------------------------%
% plotOneGas2B.m                                                          %
% Description: This function plot of one gas pressure on both boundaries. %
% Input:                                                                  %
%   t: vector of time                                                     %
%   z: vector of space                                                    %
%   zj: j'th component of z                                               %
%   ngc: number of gas component                                          %
%   P3D: 3D array of gas pressure (ngz x nts x ngc)                       %
%   comp: string vector of component (ngc)                                %
% Output:                                                                 %
%   none                                                                  %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function plotOneGas2B(t,z,BCG,P,comp,BCZ)
% set figure properties -------------------------------------------------%
fs=10;         % FontSize
fw='Normal';   % FontWeight: Bold, Normal, etc...
lw=1.2;        % linewidth
% plot ------------------------------------------------------------------%
H=plot(t,P);
xlabel('t (sec)','FontSize',fs,'FontWeight',fw);
ylabel('P (Pa) ','FontSize',fs,'FontWeight',fw);
title([comp ' Gas Pressure'],'FontSize',fs,'FontWeight',fw);
legend(['z=' num2str(round(BCZ(1))) ''],['z=' num2str(round(BCZ(2))) '']...
       ,'Location','Best','Orientation','vertical');
for i=1:2
  swap=get(get(H(i),'Parent'),'YLim');
  swap(1)=0;
  set(get(H(i),'Parent'),'YLim',swap);
  set(H(i),'LineWidth',lw);
  set(get(H(i),'Parent'),'FontSize',fs,'FontWeight',fw);
end
```

Table E.32 SurfGas3D.m

```
%-------------------------------------------------------------------------%
% surfGas3D.m                                                             %
% Description: This function makes 3D surface plot of gas pressure.       %
% Input:                                                                  %
%   t: vector of time                                                     %
%   z: vector of space                                                    %
%   P: 2D array of pressure                                               %
%   comp: component                                                       %
% Output:                                                                 %
%   none                                                                  %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function surfGas3D(t,z,P,comp)
% set figure properties
fs=10;         % FontSize
fw='Normal';   % FontWeight
hvp=-5;        % horizontal view point
vvp=30;        % vertical view point
colormap(jet); % set color map
H=surf(t,z,P,'LineStyle','none'); % surface plot
grid off;
axis([min(t) max(t) min(z) max(z)]);
view(hvp,vvp);
xlabel('t (sec) ','FontSize',fs,'FontWeight',fw);
ylabel('z (m) ','FontSize',fs,'FontWeight',fw);
zlabel('P (Pa) ','FontSize',fs,'FontWeight',fw);
set(get(H,'Parent'),'FontSize',fs,'FontWeight',fw);
title([strtrim(comp) ' Gas Pressure'],'FontSize',fs,'FontWeight',fw);
```

Table E.33 SurfSolid3D.m

```
%-------------------------------------------------------------------------%
% surfSolid3D.m                                                           %
% Description: This function makes 3D surface plot of surface coverage.  %
% Input:                                                                  %
%   t: vector of time                                                     %
%   z: vector of space                                                    %
%   C: 2D array of surface coverage                                       %
%   comp: component                                                       %
% Output:                                                                 %
%   none                                                                  %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function surfSolid3D(t,z,C,comp)
% set figure properties
fs=10;       % FontSize
fw='Normal'; % FontWeight
hvp=-5;      % horizontal view point
vvp=30;      % vertical view point
colormap(jet);  % set color map
% surface plot
H=surf(t,z,C,'LineStyle','none');
grid off;
axis([min(t) max(t) min(z) max(z)]);
view(hvp,vvp);
xlabel('t (sec) ','FontSize',fs,'FontWeight',fw);
ylabel('z (m) ','FontSize',fs,'FontWeight',fw);
set(get(H,'Parent'),'FontSize',fs,'FontWeight',fw);
title([strtrim(comp) ' Surface Coverage'],'FontSize',fs,'FontWeight',fw);
```

Table E.34 ImgGas2D.m

```
%-------------------------------------------------------------------------%
% imgGas2D.m                                                              %
% Description: This function makes 2D image plot of gas pressure.         %
% Input:                                                                  %
%   t: vector of time                                                     %
%   z: vector of space                                                    %
%   P: 2D array of pressure                                               %
%   comp: component                                                       %
% Output:                                                                 %
%   none                                                                  %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function imgGas2D(t,z,P,comp)
% set figure properties -----------------------------------------------%
fs=12;         % FontSize
fw='Normal';     % FontWeight
colormap(jet); % set color map (1~64 for jet)
% rescale P to the choosen colarmap
maxP=max(max(P(:,:)));% maximum pressure
minP=min(min(P(:,:)));% minimum pressure
maxM=size(colormap,1);% upper limit of colormap
minM=1;               % lower limit of colormap
delta=(maxM-minM)/(maxP-minP); % d(colormap)/dP
P=delta*(P(:,:)-minP)+minM;                    % rescale P
yTick=(1-0.5):(length(z))/5:(length(z)+0.5);      % rescale y tick of image
yTickLabel=0:(1.0-0)/5:1; % rescale y tick label
yTickLabel=num2str(yTickLabel','%1.1f');          % convert to string
xTick=1:(length(t)-1)/5:length(t);                % rescale x tick of image
xTickLabel=t(1):(t(length(t))-t(1))/5:t(length(t));% rescale y tick label
xTickLabel=num2str(xTickLabel','%0.4g');          % convert to string
cBarTick=minM:((maxM-minM)/5):maxM;               % rescale colorbar tick
cBarTickLabel=(cBarTick-minM)/delta+minP;     % rescale colorbar tick label
```

Table E.34 Continued

```
if maxP>=10 && maxP <100 % convert to string
  cBarTickLabel=num2str(cBarTickLabel','%2.1f');
elseif maxP>=1 && maxP <10
  cBarTickLabel=num2str(cBarTickLabel','%1.2f');
else
  cBarTickLabel=num2str(cBarTickLabel','%0.2E');
end
% image plot ------------------------------------------------------------%
H=image(P);
xlabel('t (sec) ','FontSize',fs,'FontWeight',fw);
ylabel('Z (dimensionless) ','FontSize',fs,'FontWeight',fw);
title([strtrim(comp) ' Gas Pressure'],'FontSize',fs,'FontWeight',fw);
set(get(H,'Parent'),'YDir','Normal'...
                   ,'YTick',yTick...
                   ,'YTickLabel',yTickLabel...
                   ,'XTick',xTick...
                   ,'XTickLabel',xTickLabel...
                   ,'FontSize',fs,'FontWeight',fw);
% colorbar of image -----------------------------------------------------%
cBar=colorbar; % show colorbar
set(cBar,'YTick',cBarTick...
        ,'YTickLabel',cBarTickLabel...
        ,'FontSize',fs...
        ,'FontWeight',fw); % set colorbar ticks
set(get(cBar,'Title'),'String','(Pa)'...
                     ,'FontSize',fs...
                     ,'FontWeight',fw); % set colorbar title
```

Table E.35 ImgSolid2D.m

```
%-------------------------------------------------------------------------%
% imgSolid2D.m                                                            %
% Description: This function makes 2D image plot of surface coverage.     %
% Input:                                                                  %
%   t: vector of time                                                     %
%   z: vector of space                                                    %
%   P: 2D array of pressure                                               %
%   comp: component                                                       %
% Output:                                                                 %
%   none                                                                  %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function imgSolid2D(t,z,C,comp)
% set figure properties -------------------------------------------------%
fs=12;         % FontSize
fw='Normal';     % FontWeight
colormap(jet); % set color map (1~64 for jet)
% rescale C to the choosen colarmap
maxC=max(max(C(:,:)));% maximum coverage; C should be positive
%maxC=1;
%minC=min(min(C(:,:)));% minimum coverage
minC=0;
maxM=size(colormap,1);% upper limit of colormap
minM=1;               % lower limit of colormap
delta=(maxM-minM)/(maxC-minC); % d(colormap)/dC
C=delta*(C(:,:)-minC)+minM;                      % rescale C
yTick=(1-0.5):(length(z))/5:(length(z)+0.5);     % rescale y tick of image
%yTickLabel=z(1):(z(length(z))-z(1))/5:z(length(z));% rescale y tick label
yTickLabel=0:(1.0-0)/5:1;                      % rescale y tick label
%yTickLabel=num2str(yTickLabel','%0.2E');          % convert to string
yTickLabel=num2str(yTickLabel','%1.1f');          % convert to string
xTick=1:(length(t)-1)/5:length(t);                % rescale y tick of image
xTickLabel=t(1):(t(length(t))-t(1))/5:t(length(t));% rescale y tick label
xTickLabel=num2str(xTickLabel','%0.4g');          % convert to string
cBarTick=minM:((maxM-minM)/5):maxM;           % rescale colorbar tick
```

Table E.35 Continued

```
cBarTickLabel=(cBarTick-minM)/delta+minC;     % rescale colorbar tick label
if maxC > 0.01
  cBarTickLabel=num2str(cBarTickLabel','%1.2f');% convert to string
else
  cBarTickLabel=num2str(cBarTickLabel','%0.2E');
end
% image plot ------------------------------------------------------------%
%H=image(C(2:length(z)-1,:));
H=image(C);
xlabel('t (sec) ','FontSize',fs,'FontWeight',fw);
ylabel('Z (dimensionless) ','FontSize',fs,'FontWeight',fw);
title([strtrim(comp) ' Surface Coverage'],'FontSize',fs,'FontWeight',fw);
%get(get(H,'Parent'),'YTick')
set(get(H,'Parent'),'YDir','Normal');
set(get(H,'Parent'),'YTick',yTick);
set(get(H,'Parent'),'YTickLabel',yTickLabel);

set(get(H,'Parent'),'YDir','Normal'...
                   ,'YTick',yTick...
                   ,'YTickLabel',yTickLabel...
                   ,'XTick',xTick...
                   ,'XTickLabel',xTickLabel...
                   ,'FontSize',fs,'FontWeight',fw);
% colorbar of image -----------------------------------------------------%
cBar=colorbar; % show colorbar
set(cBar,'YTick',cBarTick...
        ,'YTickLabel',cBarTickLabel...
        ,'FontSize',fs...
        ,'FontWeight',fw); % set colorbar ticks
```

Table E.36 Matlab Toolbox

```
% TransCat Toolbox
% Version 2.0 16-Aug-2010
%-------------------------------------------------------------------------%
% defSubplot.m                                                            %
% Description: This function defines the subplots in a figure.            %
% Input:                                                                  %
%   n: number of subplots, must be a positive integer                     %
% Output:                                                                 %
%   row: rows of subplot in a figure                                      %
%   col: columns of subplot in a figure                                   %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function [row,col]=defSubplot(n)
col=floor(sqrt(n));
if n == col^2
  row=col;
elseif n <= col*(col+1)
  row=col+1;
else
  col=col+1;
  row=col;
end
%-------------------------------------------------------------------------%
% getStringR1.m                                                           %
% Description: Get a rank 1 sting vector from an XML document.            %
% Input:                                                                  %
%   xmlDoc: xml document from TransCat                                    %
%   vName: variable name in the xml document                              %
%   n: size of vector                                                     %
% Output:                                                                 %
%   v: string vector output                                               %
% Created by Hsu-Wen Hsiao on 8/07/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
```

Table E.36 Continued

```
%--------------------------------------------------------------------------%
function v=getStringR1(xmlDoc,vName)
y=xmlDoc.getElementsByTagName(vName); % get the element of vName
y=y.item(0);                  % only one element with this name
ySize=y.getAttribute('Dimension'); % get the dimension attribute
ySize=str2num(ySize);         % convert the dimension string to integer
ySize=size(ySize,2);
y=y.getFirstChild.getData;    % get the data of the first element
y=char(y);                    % convert y from java array to Matlab string
yLength=length(y);            % length of string
lLimit=1; rLimit=1;           % left limit and right limit of each element
for i=1:ySize
  if lLimit <= yLength
    while y(1,rLimit+1) ~= ',' % loop to bracket element
      rLimit=rLimit+1;
      if rLimit>=yLength
        break
      end
    end
    nChar=rLimit-lLimit+1;
    if i==1
      maxChar=nChar;
      v(i,:)=y(1,lLimit:rLimit);
    else
      if nChar>maxChar
        maxChar=nChar;
        v_swap=v;
        clear v
        for j=1:(i-1)
          nChar=size(v_swap(j,:),2);
          v_raw=v_swap(j,:);
          for k=1:(maxChar-nChar)
            v_raw=[v_raw ' '];
          end
          v(j,:)=v_raw;
        end
        v(i,:)=y(1,lLimit:rLimit);
      elseif nChar<maxChar
        v_raw=y(1,lLimit:rLimit);
        for k=1:(maxChar-nChar)
          v_raw=[v_raw ' '];
        end
        v(i,:)=v_raw;
      else
        v(i,:)=y(1,lLimit:rLimit);
      end
    end
    lLimit=rLimit+2;
    rLimit=lLimit;
  else
    break
  end
end
%--------------------------------------------------------------------------%
% getDoubleR0.m                                                            %
% Description: Get a rank 0 double precision scalar from an XML document. %
% Input:                                                                   %
%   xmlDoc: xml document from TransCat                                     %
%   vName: variable name in the xml document                               %
% Output:                                                                  %
%   v: double precision scalar output                                      %
% Created by Hsu-Wen Hsiao on 8/07/09.                                     %
% Copyright 2009 UCSD. All rights reserved.                                %
%--------------------------------------------------------------------------%
function v=getDoubleR0(xmlDoc,vName)
v=xmlDoc.getElementsByTagName(vName);   % get the element of vName
v=v.item(0);                            % only one element with this name
v=v.getFirstChild.getData;              % get the data of the first element
```

Table E.36 Continued

```
v=str2double(v);                        % convert string to double
%-------------------------------------------------------------------------%
% getDoubleR1.m                                                           %
% Description: Get a rank 1 double precision vector from an XML document. %
% Input:                                                                  %
%   xmlDoc: xml document from TransCat                                    %
%   vName: variable name in the xml document                              %
%   n: size of vector                                                     %
% Output:                                                                 %
%   v: double precision vector output                                     %
% Created by Hsu-Wen Hsiao on 8/07/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function v=getDoubleR1(xmlDoc,vName)
y=xmlDoc.getElementsByTagName(vName); % get the element of vName
y=y.item(0);                  % only one element with this name
ySize=y.getAttribute('Dimension'); % get the dimension attribute
ySize=str2num(ySize);         % convert the dimension string to integer
y=y.getFirstChild.getData;    % get the data of the first element
y=char(y);                    % convert y from java array to Matlab string
yLength=length(y);            % length of string
v=zeros(ySize,1);             % initialize vector
lLimit=1; rLimit=1;           % left limit and right limit of each element
for i=1:ySize
    if lLimit <= yLength
        while y(1,rLimit+1) ~= ',' % loop to bracket element
            rLimit=rLimit+1;
            if rLimit>=yLength
                break
            end
        end
        v(i,1)=str2double(y(1,lLimit:rLimit)); % convert string to double
        lLimit=rLimit+2;
        rLimit=lLimit;
    else
        break
    end
end
%-------------------------------------------------------------------------%
% getDoubleR2.m                                                           %
% Description: Get a rank 2 double precision matrix from an XML document. %
% Input:                                                                  %
%   xmlDoc: xml document from TransCat                                    %
%   vName: variable name in the xml document                              %
% Output:                                                                 %
%   v: double precision matrix output                                     %
% Created by Hsu-Wen Hsiao on 9/7/10.                                     %
% Copyright 2010 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
 function v=getDoubleR2(xmlDoc,vName)
y=xmlDoc.getElementsByTagName(vName); % get the element of vName
y=y.item(0);                  % only one element with this name
ySize=y.getAttribute('Dimension'); % get the dimension attribute
ySize=str2num(ySize);             % convert the dimension string to integer
y=y.getFirstChild.getData;    % get the data of the first element
y=char(y);                    % convert y from java array to Matlab string
yLength=length(y);            % length of string
v=zeros(ySize);                 % initialize vector
lLimit=1; rLimit=1;           % left limit and right limit of each element
for i=1:ySize(1)
    for j=1:ySize(2)
        if lLimit <= yLength
            while y(1,rLimit+1) ~= ',' % loop to bracket element
                rLimit=rLimit+1;
                if rLimit>=yLength
                    break
                end
            end
```

Table E.36 Continued

```
            v(i,j)=str2double(y(1,lLimit:rLimit)); % convert string to dbl
            lLimit=rLimit+2;
            rLimit=lLimit;
        else
            break
        end
    end
end
%--------------------------------------------------------------------------%
% getIntegerR0.m                                                           %
% Description: Get a rank 0 integer scalar from an XML document.           %
% Created by Hsu-Wen Hsiao on 8/07/09.                                     %
% Copyright 2009 UCSD. All rights reserved.                                %
%--------------------------------------------------------------------------%
function var=getIntegerR0(outputXML,varName)
x=outputXML.getElementsByTagName(varName);   % get the element of varName
y=x.item(0);                % only one element has this name
z=y.getFirstChild.getData;   % get the data of first element with this name
var=str2num(z);             % convert string to integer
%--------------------------------------------------------------------------%
% getIntegerR1.m                                                           %
% Description: Get a rank 0 integer vector from an XML document.           %
% Created by Hsu-Wen Hsiao on 9/22/09.                                     %
% Copyright 2009 UCSD. All rights reserved.                                %
%--------------------------------------------------------------------------%
function y=getIntegerR1(xmlDoc,vName)
y=xmlDoc.getElementsByTagName(vName); % get the element of varName
y=y.item(0);              % only one element has this name
ySize=y.getAttribute('Dimension'); % get the dimension attribute
ySize=str2num(ySize);             % convert the dimension string to integer
y=y.getFirstChild.getData; % get the data of first element with this name
y=str2num(y);            % convert string to integer
z=zeros(ySize);
for i=1:ySize(1)
    z(i)=y(i);
end
%--------------------------------------------------------------------------%
% getIntegerR2.m                                                           %
% Description: Get a Rank 2 integer array from an XML document.            %
% Created by Hsu-Wen Hsiao on 9/22/09.                                     %
% Copyright 2009 UCSD. All rights reserved.                                %
%--------------------------------------------------------------------------%
function z=getIntegerR2(xmlDoc,vName)
y=xmlDoc.getElementsByTagName(vName); % get the element of vName
y=y.item(0);               % only one element has this name
ySize=y.getAttribute('Dimension'); % get the dimension attribute
ySize=str2num(ySize);             % convert the dimension string to integer
y=y.getFirstChild.getData; % get the data of first element with this name
y=str2num(y);            % convert string to integer
z=zeros(ySize);
for i=1:ySize(1)
    for j=1:ySize(2)
        z(i,j)=y((i-1)*ySize(2)+j);
    end
end
%--------------------------------------------------------------------------%
% readRun.m                                                                %
% Description: This function reads the contents of file Run.txt.           %
% Input:                                                                   %
%   none                                                                   %
% Output:                                                                  %
%   nRun: number of runs                                                   %
%   RunList: list of runs                                                  %
% Created by Hsu-Wen Hsiao on 7/29/09.                                     %
% Copyright 2009 UCSD. All rights reserved.                                %
%--------------------------------------------------------------------------%
function [nRun,RunList]=readRun(RunTxtPath)
fid=fopen(RunTxtPath,'r'); % open file run.txt
```

Table E.36 Continued

```
nRun_msg=fgets(fid);      % read the 1st line, message of number of runs
nRun=fscanf(fid,'%d');    % read the 2nd line, nRun
RunList_msg=fgetl(fid);   % read the 3rd line, message of run list
for i=1:nRun              % read RunList starting from the 4th line
    iRunList=fgetl(fid);
    nChar=size(iRunList,2);
    if i==1
        maxChar=nChar;
        RunList(i,:)=iRunList;
    else
        if nChar>maxChar
            maxChar=nChar;
            tRunList=RunList;
            clear RunList
            for j=1:(i-1)
                nChar=size(tRunList(j,:),2);
                jRunList=tRunList(j,:);
                for k=1:(maxChar-nChar)
                    jRunList=[jRunList ' '];
                end
                RunList(j,:)=jRunList;
            end
            RunList(i,:)=iRunList;
        elseif nChar<maxChar
            for k=1:(maxChar-nChar)
                iRunList=[iRunList ' '];
            end
            RunList(i,:)=iRunList;
        else
            RunList(i,:)=iRunList;
        end
    end
end
status=fclose(fid);       % close file run.txt
%-------------------------------------------------------------------------%
% readConc.m                                                              %
% Description: This function reads the contents of files *_C*.txt.        %
% Input:                                                                  %
%   fDir: file directory                                                  %
%   run: name of run                                                      %
%   ngz: number of grid in z-direction                                    %
%   nts: number of time step                                              %
%   ntc: number of total components                                       %
% Output:                                                                 %
%   C: concentration                                                      %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function C=readConc(fDir,run,ngz,nts,ntc)
for i=1:ntc
  fName=['.txt']; % construct file name
  j=i;
  while j>0
    fName=[char(48+mod(j,10)) fName];
    j=(j-mod(j,10))/10;
  end
  fName=[fDir filesep run '_C' fName];
  C_raw=load(fName);
  C(1:ngz,1:nts,i)=C_raw(1:nts,1:ngz)'; % read file *_C*.txt
end
%-------------------------------------------------------------------------%
% readTemp.m                                                              %
% Description: This function reads the contents of file *_T.txt.          %
% Input:                                                                  %
%   fDir: file directory                                                  %
%   run: name of run                                                      %
% Output:                                                                 %
%   T: temperature                                                        %
```

Table E.36 Continued

```
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function T=readTemp(fDir,run)
fid=fopen([fDir filesep run '_T.txt'],'r'); % open file *_Time.txt
T=fscanf(fid,'%e');                         % read time
st=fclose(fid);                             % close file
%-------------------------------------------------------------------------%
% readRate.m                                                              %
% Description: This function reads the contents of files *_R*.txt.        %
% Input:                                                                  %
%   fDir: file directory                                                  %
%   run: name of run                                                      %
%   ngz: number of grid in z-direction                                    %
%   nts: number of time step                                              %
%   ntr: number of elementary rate                                        %
% Output:                                                                 %
%   R: reaction rate                                                      %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function R=readRate(fDir,run,ngz,nts,ntr)
for i=1:ntr
  fName=['.txt']; % construct file name
  j=i;
  while j>0
    fName=[char(48+mod(j,10)) fName];
    j=(j-mod(j,10))/10;
  end
  fName=[fDir filesep run '_R' fName];
  R_raw=load(fName);
  R(1:ngz,1:nts,i)=R_raw(1:nts,1:ngz)'; % read file *_ER*.txt
end
%-------------------------------------------------------------------------%
% readTime.m                                                              %
% Description: This function reads the contents of file *_Time.txt.       %
% Input:                                                                  %
%   fDir: file directory                                                  %
%   run: name of run                                                      %
% Output:                                                                 %
%   t: time                                                               %
% Created by Hsu-Wen Hsiao on 7/29/09.                                    %
% Copyright 2009 UCSD. All rights reserved.                               %
%-------------------------------------------------------------------------%
function t=readTime(fDir,run)
fid=fopen([fDir filesep run '_Time.txt'],'r'); % open file *_Time.txt
t=fscanf(fid,'%e');                           % read time
st=fclose(fid);                               % close file
%-------------------------------------------------------------------------%
% Readme.m for TransCat Toolbox
% Created by Hsu-Wen Hsiao on August 12, 2009.
% Copyright 2009 UCSD. All rights reserved.
%
% Description:
%   This folder contains functions and routines used for post processing
% simulation result from TranCat. To work properly, this folder should be
% placed in the same directory as other post processing M-files and folders
% containing simulation result.
%
%   These M-files are User-Contributed Routines that are being distributed
% by The MathWorks, upon request, on an "as is" basis.  A User Contributed
% Routine is not a product of The MathWorks, Inc, and The MathWorks assumes
% no responsibility for any errors that may exist in these routines.
%-------------------------------------------------------------------------%
```