SANDIA REPORT

SAND2011-3282
Unlimited Release
Printed January 2011

TChem - A Software Toolkit for the Analysis of Complex Kinetic Models

Cosmin Safta, Habib Najm, Omar Knio

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2011-3282
Unlimited Release
Printed January 2011

TChem - A Software Toolkit for the Analysis of Complex Kinetic Models

Cosmin Safta, Habib N. Najm
Sandia National Laboratories, Livermore, CA
{csafta,hnnajm}@sandia.gov
and
Omar Knio
Johns Hopkins University, Baltimore, MD
knio@jhu.edu

Abstract

The TChem toolkit is a software library that enables numerical simulations using complex chemistry and facilitates the analysis of detailed kinetic models. The toolkit provide capabilities for thermodynamic properties based on NASA polynomials and species production/consumption rates. It incorporates methods that can selectively modify reaction parameters for sensitivity analysis. The library contains several functions that provide analytically computed Jacobian matrices necessary for the efficient time advancement and analysis of detailed kinetic models.

Acknowledgment

This work was supported by the US Department of Energy (DOE), Office of Basic Energy Sciences (BES) Division of Chemical Sciences, Geosciences, and Biosciences. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

Contents

1 Introduction **9**

2 Thermodynamic properties, equation of state, and mixture formulae **11**

3 Reaction-rate expressions **13**

4 Jacobian Matrices **17**

Analytical expressions for $\mathcal{F}$ 18

Efficient evaluation of the $\mathcal{F}$ terms 22

Analytical expressions for $\overline{\mathcal{J}}$ 22

Analytical expressions for $\widetilde{\mathcal{J}}$ 23

5 Examples **25**

ign-c 25

ign-cpp 28

ign-f 30

6 Library Functions **33**

TC_chg.c 33

TC_edit.c 34

TC_init.c 34

TC_mlms.c 34

TC_rr.c 37

TC_spec.c 40

TC_src.c 41

TC_thermo.c 45

7 Nomenclature **51**

Non-dimensional values 51

References **52**

List of Figures

5.1 Left Frame: variation of the ignition delay times with the equivalence ratio (Φ) for iso-octane/air mixtures at various pressures and initial temperature of 1000K. Right Frame: Variation of ignition delay time with initial temperature for a stoichiometric iso-octane/mixtures at various pressures. The kinetic model involves 874 species and 3796 reactions [5] 27

5.2 Left Frame: Variation of the ignition delay times with the initial mixture temperature for 100 ON, 80 ON and 60 ON at 15 atm and 45 atm. Right Frame: Variation of ignition delay time for 50 ON at 15 atm, 30 atm, and 45 atm. The kinetic model involves 1034 species and 4236 reactions [4, 3] 28

5.3 Mass fraction profiles during the constant pressure ignition of a stoichimetric methane-air mixture at 1atm. Methane combustion is modeled using the GRI-Mech v3.0 kinetic model (53 species, 325 reactions) [7]. 30

This page intentionally left blank.

Chapter 1

Introduction

Consider the system of stiff ordinary differential equations (ODE's) that describe the evolution of a homogenous mixture of N_{spec} species in an open container

$$\begin{aligned} \frac{\partial T}{\partial t} &= -\frac{1}{\rho c_p} \sum_{k=1}^{N_{\text{spec}}} h_k \dot{\omega}_k \\ \frac{\partial Y_k}{\partial t} &= \frac{\dot{\omega}_k}{\rho} \quad k = 1 \ldots N_{\text{spec}}, \end{aligned} \tag{1.1}$$

where ρ is the density, T the temperature, Y_k the mass fraction of species k, c_p and c_{p_k} the specific heat of the mixture and species k, respectively and $\dot{\omega}_k$ the reaction rate of species k. In some cases, the ODE for select species (usually the most abundant in the mixture) is replaced by an equation enforcing mass conservation. For example

$$Y_L = 1 - \sum_{k=1, k \neq L}^{N_{\text{spec}}} Y_k \tag{1.2}$$

can substitute the ODE for species L in (1.1).

The system of equations (1.1) can aid in the design of kinetic models for increasingly complex fuels. It allows the computation of ignition delay times and mixture compositions that can be compared against experimental results from shock tube experiments. This system is a canonical configuration which enables the study of chemical reaction pathways and testing of algorithms for reducing detailed kinetic models for use in more complex configurations.

TChem is as an open source software library that facilitates the analysis of complex kinetic models and provides tools for incorporating these models in combustion simulations. The example codes, written in C, C++, and Fortran, that are part of the library provide model solvers for the system of ODE 1.1. While the library is standalone, the example applications require the open source libraries CVODE [1] (for the C and C++ models) and DVODE[2] (for the Fortran model).

This report is organized as follows. Chapter 2 contains expressions for the thermodynamic properties and chemical composition formulae implemented in the library. Chapter 3 provides an overview of the reaction rate expressions. Chapter 4 describes the derivation and calculation of the Jacobian matrices.

[1] `https://computation.llnl.gov/casc/sundials`
[2] `https://computation.llnl.gov/casc/software.html`

Chapter 5 describes C, C++ and Fortran example codes that use TChem. The user interface for all functions is presented in Chapter 6.

Chapter 2

Thermodynamic properties, equation of state, and mixture formulae

The heat capacity at constant pressure (C_p), molar enthalpy (H), and entropy (S) for a mixture are given by :

$$C_p = \sum_{k=1}^{N_{\text{spec}}} X_k C_{p_k}, \quad H = \sum_{k=1}^{N_{\text{spec}}} X_k H_k, \quad S = \sum_{k=1}^{N_{\text{spec}}} X_k S_k \tag{2.1}$$

where subscript k stands for species "k" and N_{spec} is the number of species in the mixture. The standard-state thermodinamic properties for a thermally perfect gas are computed based on NASA polynomials [6] :

$$\frac{C_{p_k}}{\Re} = a_{0,k} + T\left(a_{1,k} + T\left(a_{2,k} + T\left(a_{3,k} + a_{4,k}T\right)\right)\right), \tag{2.2}$$

$$\frac{H_k}{\Re} = T\left(a_{0,k} + T\left(\frac{a_{1,k}}{2} + T\left(\frac{a_{2,k}}{3} + T\left(\frac{a_{3,k}}{4} + \frac{a_{4,k}}{5}T\right)\right)\right)\right) + a_{5,k}, \tag{2.3}$$

$$\frac{S_k}{\Re} = a_{0,k}\ln T + T\left(a_{1,k} + T\left(\frac{a_{2,k}}{2} + T\left(\frac{a_{3,k}}{3} + \frac{a_{4,k}}{4}T\right)\right)\right) + a_{6,k}, \tag{2.4}$$

Here X_k the mole fraction of species k, and $\Re$ the universal gas constant. The specific thermodynamic properties in mass units are optained by dividing the above expressions by the molecular weights:

$$c_{p_k} = C_{p_k}/W_k, \quad h_k = H_k/W_k, \quad s_k = S_k/W_k \tag{2.5}$$

and the mixture properties computed as

$$c_p = \sum Y_k c_{p_k}, \quad h = \sum Y_k h_k, \quad s = \sum Y_k s_k \tag{2.6}$$

where W_k and Y_k are the molecular weight and the mass fraction of species k, respectively.

The ideal gas equation of state is used throught the library,

$$P = \rho\frac{\Re}{\overline{W}}T = \left(\sum_{k=1}^{N_{\text{spec}}} \mathfrak{X}_k\right)\Re T \tag{2.7}$$

where P is the thermodynamic pressure, $\overline{W}$ is the molecular weight of the mixture, T is the temperature, and $\mathfrak{X}_k$ is the molar concentration of species k. $\overline{W}$ is computed as

$$\overline{W} = \left(\sum_{k=1}^{N_{\text{spec}}} \frac{Y_k}{W_k}\right)^{-1} = \sum_{k=1}^{N_{\text{spec}}} X_k W_k. \tag{2.8}$$

Mass and mole fractions can be computed from each other as

$$X_k = Y_k \overline{W}/W_k, \qquad Y_k = X_k W_k/\overline{W} \tag{2.9}$$

while the molar concentration is given by $\mathfrak{X}_k = \rho Y_k/W_k$.

The units for these properties are given in Section 7. Functions that perform calculations described in this section are contained in files *TC_mlms.c* and *TC_thermo.c* and are described in Section 6.

Chapter 3

Reaction-rate expressions

The reaction rate of species k in mass units is written as

$$\dot{\omega}_k = W_k \sum_{i=1}^{N_{\text{reac}}} \nu_{ki} q_i, \qquad \nu_{ki} = \nu''_{ki} - \nu'_{ki}, \tag{3.1}$$

where N_{reac} is the number of reactions and ν'_{ki} and ν''_{ki} are the stoichiometric coefficients of species k in reaction i for the reactant and product side of the reaction, respectively. The rate-of-progress of reaction i is $q_i = \mathcal{C}_i \mathcal{R}_i$, with

$$\mathcal{C}_i = \begin{cases} 1 & \text{basic reaction} \\ \mathfrak{X}_i & 3^{\text{rd}} \text{ body enhanced, no pressure dependence} \\ \frac{P_{ri}}{1+P_{ri}} F_i & \text{unimolecular/recombination fall-off reactions} \\ \frac{1}{1+P_{ri}} F_i & \text{chemically activated bimolecular reactions} \end{cases} \tag{3.2}$$

and $\mathcal{R}_i = k_{f_i} \prod_{j=1}^{N_{\text{spec}}} \mathfrak{X}_j^{\nu'_{ji}} - k_{ri} \prod_{j=1}^{N_{\text{spec}}} \mathfrak{X}_j^{\nu''_{ji}}$. The above expressions are detailed below :

- Forward rate constant has an Arrhenius expression, $k_{f_i} = A_i T^{\beta_i} \exp\left(-\frac{E_{ai}}{\Re T}\right)$, where A_i, β_i, and E_{ai} are the pre-exponential factor, temperature exponent, and activation energy, respectively, for reaction i.

- Reverse rate constant k_{ri}. For reactions with reverse Arrhenius parameters specified, k_{ri} is computed similar to k_{f_i}. If the reverse Arrhenius parameters are not specified, k_{ri} is computed as $k_{ri} = k_{f_i}/K_{ci}$, where

$$K_{ci} = \left(\frac{p_{atm}}{\Re T}\right)^{\sum_{k=1}^{N_{\text{spec}}} \nu_{ki}} K_{p_i} = \left(\frac{p_{atm}}{\Re T}\right)^{\sum_{k=1}^{N_{\text{spec}}} \nu_{ki}} \exp\left(\sum_{k=1}^{N_{\text{spec}}} \nu_{ki} \left(\frac{s_k}{R_k} - \frac{h_k}{R_k T}\right)\right) \tag{3.3}$$

$$= \left(\frac{p_{atm}}{\Re}\right)^{\sum_{k=1}^{N_{\text{spec}}} \nu_{ki}} \exp\left(\sum_{k=1}^{N_{\text{spec}}} \nu_{ki} \left(-\ln T + \frac{s_k}{R_k} - \frac{h_k}{R_k T}\right)\right) \tag{3.4}$$

$$= \left(\frac{p_{atm}}{\Re}\right)^{\sum_{k=1}^{N_{\text{spec}}} \nu_{ki}} \exp\left(\sum_{k=1}^{N_{\text{spec}}} \nu_{ki} g_k\right). \tag{3.5}$$

Based on the polynomial expressions in (2.3) and (2.4), g_k's are computed as

$$g_k = -\ln T + \frac{s_k}{R_k} - \frac{h_k}{R_k T} \tag{3.6}$$

$$= a_{6,k} - a_{0,k} + \left(a_{0,k} - 1\right)\ln T + T\left(\frac{a_{1,k}}{2} + T\left(\frac{a_{2,k}}{6} + T\left(\frac{a_{3,k}}{12} + \frac{a_{4,k}}{20}T\right)\right)\right) - \frac{a_{5,k}}{T} \tag{3.7}$$

Note: If a reaction is irreversible, $k_r = 0$.

- Concentration of the "third-body", $\mathfrak{X}_i = \sum_{j=1}^{N_{\text{spec}}} \alpha_{ij}\mathfrak{X}_j$, where α_{ij} is the efficiency of species j in reaction i and $\mathfrak{X}_j$ is the concentration of species j. α_{ij} coefficients are set to 1 unless specified in the kinetic model description.

- Reduced pressure P_{ri}. If expression "(+M)" is used to describe a reaction, then $P_{ri} = \frac{k_{0i}}{k_{\infty i}}\mathfrak{X}_i$. For reactions that contain expressions like "(+Υ_m)" (Υ_m is the name of species m), $P_{ri} = \frac{k_{0i}}{k_{\infty i}}\mathfrak{X}_m$.

  For *unimolecular/recombination fall-off reactions* the Arrhenius parameters for the high-pressure limit rate constant k_∞ are given on the reaction line, while the parameters for the low-pressure limit rate constant k_0 are given on the auxiliary reaction line that contains the keyword "LOW". For *chemically activated bimolecular reactions* the parameters for k_0 are given on the reaction line while the parameters for k_∞ are given on the auxiliary reaction line that contains the keyword "HIGH".

- $$F_i = \begin{cases} 1 & \text{Lindemann reaction} \\ F_{cent}^{1/\left(1+(A/B)^2\right)} & \text{Troe reaction} \\ dT^e\left(a\exp\left(-\frac{b}{T}\right) + \exp\left(-\frac{T}{c}\right)\right)^X & \text{SRI reaction} \end{cases} \tag{3.8}$$

  1. For the Troe form, F_{cent}, A, and B are

  $$F_{cent} = (1-a)\exp\left(-\frac{T}{T^{***}}\right) + a\exp\left(-\frac{T}{T^{*}}\right) + \exp\left(-\frac{T^{**}}{T}\right) \tag{3.9}$$

  $$A = \log_{10}P_{ri} - 0.67\log_{10}F_{cent} - 0.4 \tag{3.10}$$

  $$B = 0.806 - 1.1762\log_{10}F_{cent} - 0.14\log_{10}P_{ri} \tag{3.11}$$

  Parameters a, T^{***}, T^{*}, and T^{**} are provided (in this order) in the kinetic model description for each Troe-type reaction. If T^{**} is ommited, only the first two terms are used to compute F_{cent}.

  2. For the SRI form exponent X is computed as $X = \left(1 + (\log_{10}P_{ri})^2\right)^{-1}$. Parameters a, b, c, d, and e are provided in the kinetic model description for each SRI-type reaction. If d and e are ommited, these parameters are set to $d = 1$ and $e = 0$.

Note on units for reaction rates

In most cases, the kinetic models input files contain parameters that are based on calories, cm, moles,

kelvin, seconds. The mixture temperature and species molar concentrations are necessary to compute the reaction rate. Molar concentrations are computed as

$$\mathfrak{X}_k = \rho \frac{Y_k}{W_k} \qquad \left[\frac{kmol}{m^3}\right] \tag{3.12}$$

The molar concentrations are then multiplied by 10^{-3} to convert them to $\left[\frac{mol}{cm^3}\right]$. The molar reaction rates computed based on these values are in $\left[\frac{mol}{cm^3 \cdot s}\right]$. These molar reaction rates are multiplied by 10^3 and by the molecular weight of each species $\left[\frac{kg}{kmol}\right]$ to obtain the mass reaction rates in $\left[\frac{kg}{m^3 \cdot s}\right]$.

This page intentionally left blank.

Chapter 4

Jacobian Matrices

Efficient integration and accurate analysis of the stiff system of ODE's (1.1) requires the Jacobian matrix of the *rhs* vector. In this chapter we will derive the components of the Jacobian matrices both for systems which includes equations for all species as well as for systems of ODE's with one eqauation being replaced by the mass conservation equation (1.2).

Let

$$\overline{\Phi} = \left\{P, T, Y_1, Y_2, \ldots, Y_{N_{\text{spec}}}\right\}^T \text{ and} \tag{4.1}$$

$$\widetilde{\Phi} = \left\{P, T, Y_1, \ldots, Y_{L-1}, Y_{L+1}, \ldots, Y_{N_{\text{spec}}}\right\}^T \tag{4.2}$$

denote the set of variables for the full and reduced systems of ODE's, respectively. Altough we included pressure as an independent variable, we restrict our attention to open containers for which the thermodynamic pressure, P, is constant. Then (1.1) can be written in compact form as

$$\frac{\partial\overline{\Phi}}{\partial t} = \overline{f} = \left\{S_P, S_T, S_{Y_1}, S_{Y_2}, \ldots, S_{Y_{N_{\text{spec}}}}\right\}^T \text{ and} \tag{4.3}$$

$$\frac{\partial\widetilde{\Phi}}{\partial t} = \widetilde{f} = \left\{S_P, S_T, S_{Y_1}, \ldots, S_{Y_{L-1}}, S_{Y_{L+1}}, \ldots, S_{Y_{N_{\text{spec}}}}\right\}^T, \tag{4.4}$$

where $S_P \equiv 0$, $S_T = -\frac{1}{\rho c_p}\sum_{k=1}^{N_{\text{spec}}} h_k\dot{\omega}_k$, and $S_{Y_k} = \dot{\omega}_k/\rho$.

Let $\overline{\jmath}$ and $\widetilde{\jmath}$ be the Jacobian matrices corresponding to $\overline{f}$ and $\widetilde{f}$, respectively. In order to facilitate the derivation of partial derivatives $\partial\overline{f}_i/\partial\overline{\Phi}_j$ and $\partial\widetilde{f}_i/\partial\widetilde{\Phi}_j$ that are the elements of these matrices, it is useful to add density to the state vector. Let Φ be the extended state vector:

$$\Phi = \left\{\rho, P, T, Y_1, Y_2, \ldots, Y_{N_{\text{spec}}}\right\}^T, \tag{4.5}$$

The *rhs* term corresponding to the time advance of Φ is

$$f(\Phi) = \left\{S_\rho, S_P, -\frac{1}{\rho c_p}\sum_{k=1}^{N_{\text{spec}}} h_k\dot{\omega}_k, \frac{\dot{\omega}_1}{\rho}, \frac{\dot{\omega}_2}{\rho}, \cdots, \frac{\dot{\omega}_{N_{\text{spec}}}}{\rho}\right\}^T, \tag{4.6}$$

with $S_\rho = -\overline{W}\sum_{k=1}^{N_{\text{spec}}}\frac{\dot{\omega}_k}{W_k} + \frac{1}{c_pT}\sum_{k=1}^{N_{\text{spec}}} h_k\dot{\omega}_k$.

Let $\mathcal{F}$ be the Jacobian matrix corresponding to f. Chain-rule differentiation leads to

$$\frac{\partial \overline{f}_u}{\partial v} = \frac{\partial f_u}{\partial v} + \frac{\partial f_u}{\partial \rho}\frac{\partial \rho}{\partial v} \tag{4.7}$$

$$\frac{\partial \widetilde{f}_u}{\partial v} = \frac{\partial f_u}{\partial v} + \frac{\partial f_u}{\partial \rho}\frac{\partial \rho}{\partial v} + \frac{\partial f_u}{\partial Y_L}\frac{\partial Y_L}{\partial v}. \tag{4.8}$$

Note that each component u of $\widetilde{\Phi}$ is also a component of $\overline{\Phi}$ and Φ, and the corresponding *rhs* terms are the same

$$\widetilde{f}_u\left(\widetilde{\Phi}\right) = \overline{f}_u\left(\overline{\Phi}\right) = f_u(\Phi) \tag{4.9}$$

In terms of actual components

$$\overline{\mathcal{J}}_{i,j} = \mathcal{F}_{i+1,j+1} + \mathcal{F}_{i+1,1}\frac{\partial \rho}{\partial \overline{\Phi}_j} \qquad i,j = 1,2,\ldots,N_{\text{spec}}+2 \tag{4.10}$$

$$\widetilde{\mathcal{J}}_{i,j} = \mathcal{F}_{i+i_s,j+j_s} + \mathcal{F}_{i+i_s,1}\frac{\partial \rho}{\partial \widetilde{\Phi}_j} + \mathcal{F}_{i+i_s,L+3}\frac{\partial Y_L}{\partial \widetilde{\Phi}_j} \qquad i,j = 1,2,\ldots,N_{\text{spec}}+1. \tag{4.11}$$

where

$$i_s, j_s = \begin{cases} 1 & i, j+1 < 3+L \\ 2 & \text{otherwise} \end{cases} \tag{4.12}$$

Analytical expressions for $\mathcal{F}$

Analytical for expressions for the matrix components are provided line by line below. Note that, in the expressions below, whenever the summation limits are omitted, the sums are over all N_{spec} species.

- Line #1: **Density equation**. Density is not currently advanced through an equation for the time being. Since $\mathcal{F}_{1j} = \frac{\partial f_1}{\partial \Phi_j}$ are not needed, all components on the first line of $\mathcal{F}$ are set to 0.

- Line #2: **Pressure equation**. The source term for the pressure equation is assumed constant: $f_2 = \overline{f}_2 = const$.

$$\frac{\partial f_2}{\partial \Phi_j} \equiv 0 \text{ for all j } \Rightarrow \mathcal{F}_{2,j} \equiv 0. \tag{4.13}$$

- Line #3: **Temperature equation**: $\mathcal{F}_{3,j} = \frac{\partial f_3}{\partial \Phi_j}$, where $f_3 = -\frac{1}{\rho c_p}\sum_{k=1}^{N_{\text{spec}}} h_k \dot{\omega}_k$. Here

$$c_p = \sum_{k=1}^{N_{\text{spec}}} Y_k c_{p_k}(T) \text{ and } \dot{\omega}_k = \dot{\omega}_k(T, \mathfrak{X}_1, \mathfrak{X}_2, \ldots, \mathfrak{X}_{N_{\text{spec}}}). \tag{4.14}$$

$\mathfrak{X}_k$ is the molar concentration of species k

$$\mathfrak{X}_k = \frac{\rho Y_k}{W_k}. \tag{4.15}$$

The individual components of $\mathcal{F}_{3,*}$ are computed as

$$\mathcal{F}_{3,1} = \frac{\partial f_3}{\partial \rho} = \frac{1}{\rho^2 c_p}\sum h_k\dot{\omega}_k - \frac{1}{\rho c_p}\sum h_k\frac{\partial \dot{\omega}_k}{\partial \rho} = \frac{1}{\rho c_p}\sum h_k\left(\frac{\dot{\omega}_k}{\rho} - \frac{\partial \dot{\omega}_k}{\partial \rho}\right) \tag{4.16}$$

$$\mathcal{F}_{3,2} = 0 \tag{4.17}$$

$$\mathcal{F}_{3,3} = \frac{\partial f_3}{\partial T} = \frac{1}{\rho c_p^2}\frac{\partial c_p}{\partial T}\sum h_k\dot{\omega}_k - \frac{1}{\rho c_p}\sum c_{p_k}\dot{\omega}_k - \frac{1}{\rho c_p}\sum h_k\frac{\partial \dot{\omega}_k}{\partial T} \tag{4.18}$$

$$\mathcal{F}_{3,3+j} = \frac{\partial f_3}{\partial Y_j} = \frac{1}{\rho c_p^2}c_{p\,j}\sum h_k\dot{\omega}_k - \frac{1}{\rho c_p}\sum h_k\frac{\partial \dot{\omega}_k}{\partial Y_j}, j = 1,2,\ldots,N_{\text{spec}} \tag{4.19}$$

- Line #(3+k), k=1…N_{spec}: **Species equations**. For subsequent lines $3+k$ of the Jacobian matrix $\mathcal{F}$:

$$\mathcal{F}_{3+k,1} = \frac{\partial f_{3+k}}{\partial \rho} = \frac{\partial\left(\dot{\omega}_k/\rho\right)}{\partial \rho} = \frac{1}{\rho}\left(\frac{\partial \dot{\omega}_k}{\partial \rho} - \frac{\dot{\omega}_k}{\rho}\right) \tag{4.20}$$

$$\mathcal{F}_{3+k,2} = \frac{\partial\left(\dot{\omega}_k/\rho\right)}{\partial P} \equiv 0 \tag{4.21}$$

$$\mathcal{F}_{3+k,3} = \frac{\partial\left(\dot{\omega}_k/\rho\right)}{\partial T} = \frac{1}{\rho}\frac{\partial \dot{\omega}_k}{\partial T} \tag{4.22}$$

$$\mathcal{F}_{3+k,3+j} = \frac{\partial\left(\dot{\omega}_k/\rho\right)}{\partial Y_j} = \frac{1}{\rho}\frac{\partial \dot{\omega}_k}{\partial Y_j}, j,k = 1,2,\ldots,N_{\text{spec}} \tag{4.23}$$

The values for heat capacities and their derivaties are computed based on the NASA polynomial fits as

$$\frac{\partial c_{p_k}}{\partial T} = R_k\Big(a_{1,k} + T\left(2a_{2,k} + T\left(3a_{3,k} + 4a_{4,k}T\right)\right)\Big), \tag{4.24}$$

$$\frac{\partial c_p}{\partial T} = \sum Y_k\frac{\partial c_{p_k}}{\partial T}, \qquad \frac{\partial c_p}{\partial Y_j} = c_{p\,j} \tag{4.25}$$

The partial derivatives of the species reaction rates, $\dot{\omega}_k(T,\mathfrak{X}_1,\mathfrak{X}_2,\ldots)$, with respect to various indepenend variables are computed as

$$\left.\frac{\partial \dot{\omega}_k}{\partial \rho}\right|_{T,Y_s} = \sum_{l=1}^{N_{\text{spec}}}\frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_l}\frac{\partial \mathfrak{X}_l}{\partial \rho} + \frac{\partial \dot{\omega}_k}{\partial T}\cancelto{0}{\frac{\partial T}{\partial \rho}} + \cancelto{0}{\frac{\partial \dot{\omega}_k}{\partial \rho}}\cancelto{0}{\frac{\partial \rho}{\partial \rho}} = \sum_{l=1}^{N_{\text{spec}}}\frac{Y_l}{W_l}\frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_l} \tag{4.26}$$

$$\left.\frac{\partial \dot{\omega}_k}{\partial Y_j}\right|_{\rho,T,Y_{s\neq j}} = \sum_{l=1}^{N_{\text{spec}}}\frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_l}\frac{\partial \mathfrak{X}_l}{\partial Y_j} + \frac{\partial \dot{\omega}_k}{\partial T}\cancelto{0}{\frac{\partial T}{\partial Y_j}} + \frac{\partial \dot{\omega}_k}{\partial \rho}\cancelto{0}{\frac{\partial \rho}{\partial Y_j}} = \frac{\rho}{W_j}\frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_j} \tag{4.27}$$

$$\tag{4.28}$$

The steps for the calculation of $\frac{\partial \dot{\omega}_k}{\partial T}$ and $\frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_l}$ are itemized below

– Derivatives of reaction rate

$$\dot{\omega}_k = W_k \sum_{i=1}^{N_{\text{reac}}} \nu_{ki} q_i \Rightarrow \frac{\partial \dot{\omega}_k}{\partial T} = W_k \sum_{i=1}^{N_{\text{reac}}} \nu_{ki} \frac{\partial q_i}{\partial T}, \quad \frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_l} = W_k \sum_{i=1}^{N_{\text{reac}}} \nu_{ki} \frac{\partial q_i}{\partial \mathfrak{X}_l}$$

– Derivatives of rate-of-progress variables

$$q_i = \mathcal{C}_i \mathcal{R}_i \Rightarrow \frac{\partial q_i}{\partial T} = \frac{\partial \mathcal{C}_i}{\partial T} \mathcal{R}_i + \mathcal{C}_i \frac{\partial \mathcal{R}_i}{\partial T}, \quad \frac{\partial q_i}{\partial \mathfrak{X}_l} = \frac{\partial \mathcal{C}_i}{\partial \mathfrak{X}_l} \mathcal{R}_i + \mathcal{C}_i \frac{\partial \mathcal{R}_i}{\partial \mathfrak{X}_l}$$

– Derivatives of $\mathcal{C}_i$

1. Basic reactions $\mathcal{C}_i = 1$: $\frac{\partial \mathcal{C}_i}{\partial T} \equiv \frac{\partial \mathcal{C}_i}{\partial \mathfrak{X}_l} \equiv 0$
2. 3rdbody-enhanced reactions $\mathcal{C}_i = \mathfrak{X}_i$: $\frac{\partial \mathcal{C}_i}{\partial T} \equiv 0$, $\frac{\partial \mathcal{C}_i}{\partial \mathfrak{X}_l} = \alpha_{il}$
3. Unimolecular/recombination fall-off reactions $\mathcal{C}_i = \frac{P_{ri}}{1+P_{ri}} F_i$

$$\frac{\partial \mathcal{C}_i}{\partial T} = \frac{1}{(1+P_{ri})^2} \frac{\partial P_{ri}}{\partial T} F_i + \frac{P_{ri}}{1+P_{ri}} \frac{\partial F_i}{\partial T}$$
$$\frac{\partial \mathcal{C}_i}{\partial \mathfrak{X}_l} = \frac{1}{(1+P_{ri})^2} \frac{\partial P_{ri}}{\partial \mathfrak{X}_l} F_i + \frac{P_{ri}}{1+P_{ri}} \frac{\partial F_i}{\partial \mathfrak{X}_l}$$

(a) $P_r = \frac{k_{0i}}{k_{\infty i}} \mathfrak{X}_i \Rightarrow \frac{\partial P_{ri}}{\partial T} = \frac{k'_{0i} k_{\infty i} - k_{0i} k'_{\infty i}}{k^2_{\infty i}} \mathfrak{X}_i, \quad \frac{\partial P_{ri}}{\partial \mathfrak{X}_l} = \frac{k_{0i}}{k_{\infty i}} \alpha_{il}.$

(b) $P_{ri} = \frac{k_{0i}}{k_{\infty i}} \mathfrak{X}_m \Rightarrow \frac{\partial P_{ri}}{\partial T} = \frac{k'_{0i} k_{\infty i} - k_{0i} k'_{\infty i}}{k^2_{\infty i}} \mathfrak{X}_m, \quad \frac{\partial P_{ri}}{\partial \mathfrak{X}_l} = \frac{k_{0i}}{k_{\infty i}} \delta_{lm}$, where δ_{lm} is Kroenecker delta symbol.

(c) For Lindemann form $F_i = 1 \Rightarrow \frac{\partial F_i}{\partial T} \equiv \frac{\partial F_i}{\partial \mathfrak{X}_l} \equiv 0$.

(d) For Troe form

$$\frac{\partial F}{\partial T} = \frac{\partial F}{\partial F_{cent}} \frac{\partial F_{cent}}{\partial T} + \frac{\partial F}{\partial P_r} \frac{\partial P_r}{\partial T}$$
$$\frac{\partial F}{\partial \mathfrak{X}_l} = \frac{\partial F}{\partial F_{cent}} \cancel{\frac{\partial F_{cent}}{\partial \mathfrak{X}_l}}^{0} + \frac{\partial F}{\partial P_r} \frac{\partial P_r}{\partial \mathfrak{X}_l} = \frac{\partial F}{\partial P_r} \frac{\partial P_r}{\partial \mathfrak{X}_l}$$
$$\frac{\partial F}{\partial F_{cent}} = \frac{F}{F_{cent}\left(1+\left(\frac{A}{B}\right)^2\right)} - F \ln F_{cent} \left(\frac{2A}{B^3}\right) \frac{A_F B - B_F A}{\left(1+\left(\frac{A}{B}\right)^2\right)^2}$$
$$\frac{\partial F}{\partial P_r} = F \ln F_{cent} \left(\frac{2A}{B^3}\right) \frac{A_{P_r} B - B_{P_r} A}{\left(1+\left(\frac{A}{B}\right)^2\right)^2}$$

where

$$A_F = \frac{\partial A}{\partial F_{cent}} = -\frac{0.67}{F_{cent} \ln 10}, \quad B_F = \frac{\partial B}{\partial F_{cent}} = -\frac{1.1762}{F_{cent} \ln 10}$$
$$A_{P_r} = \frac{\partial A}{\partial P_r} = \frac{1}{P_r \ln 10}, \quad B_{P_r} = \frac{\partial B}{\partial P_r} = -\frac{0.14}{P_r \ln 10}$$
$$\frac{\partial F_{cent}}{\partial T} = -\frac{1-a}{T^{***}} \exp\left(-\frac{T}{T^{***}}\right) - \frac{a}{T^*} \exp\left(-\frac{T}{T^*}\right) + \frac{T^{**}}{T^2} \exp\left(-\frac{T^{**}}{T}\right)$$

(e) For SRI form

$$\begin{aligned}\frac{\partial F}{\partial T} =& F\left(\frac{e}{T}+\frac{\partial X}{\partial P_r}\frac{\partial P_r}{\partial T}\ln\left(a\exp\left(-\frac{b}{T}\right)+\exp\left(-\frac{T}{c}\right)\right)\right.\\ &\left.+X\frac{\frac{ab}{T^2}\exp\left(-\frac{b}{T}\right)-\frac{1}{c}\exp\left(-\frac{T}{c}\right)}{a\exp\left(-\frac{b}{T}\right)+\exp\left(-\frac{T}{c}\right)}\right)\\ \frac{\partial F}{\partial \mathfrak{X}_l} =& F\ln\left(a\exp\left(-\frac{b}{T}\right)+\exp\left(-\frac{T}{c}\right)\right)\frac{\partial X}{\partial P_r}\frac{\partial P_r}{\partial \mathfrak{X}_l}\\ \frac{\partial X}{\partial P_r} =& -X^2\frac{2\log_1 0 P_r}{P_r\ln 10}\end{aligned}$$

4. Chemically activated bimolecular reactions: $C_i = \frac{1}{1+P_{ri}}F_i$

$$\begin{aligned}\frac{\partial C_i}{\partial T} &= -\frac{1}{(1+P_{ri})^2}\frac{\partial P_{ri}}{\partial T}F_i+\frac{1}{1+P_{ri}}\frac{\partial F_i}{\partial T}\\ \frac{\partial C_i}{\partial \mathfrak{X}_l} &= -\frac{1}{(1+P_{ri})^2}\frac{\partial P_{ri}}{\partial \mathfrak{X}_l}F_i+\frac{1}{1+P_{ri}}\frac{\partial F_i}{\partial \mathfrak{X}_l}\end{aligned}$$

Partial derivatives of P_{ri} and F_i are computed similar to the ones above.

– Derivatives of $\mathcal{R}_i$

$$\begin{aligned}\frac{\partial \mathcal{R}_i}{\partial T} &= k'_{f_i}\prod_{j=1}^{N_{\text{spec}}}\mathfrak{X}_j^{\nu'_{ji}}-k'_{ri}\prod_{j=1}^{N_{\text{spec}}}\mathfrak{X}_j^{\nu''_{ji}}\\ \frac{\partial \mathcal{R}_i}{\partial \mathfrak{X}_l} &= \frac{k_{f_i}\nu'_{li}\prod_{j=1}^{N_{\text{spec}}}\mathfrak{X}_j^{\nu'_{ji}}}{\mathfrak{X}_l}-\frac{k_{ri}\nu''_{li}\prod_{j=1}^{N_{\text{spec}}}\mathfrak{X}_j^{\nu''_{ji}}}{\mathfrak{X}_l}\end{aligned}$$

1. $k_{f_i} = A_i T^{\beta_i}\exp\left(-\frac{E_{ai}}{\Re T}\right) = A_i\exp\left(\beta_i\ln T-\frac{T_{ai}}{T}\right)$, where $T_{ai} = E_{ai}/\Re$. The derivative with respect to temperature can be calculated as $k'_{f_i} = \frac{k_{f_i}}{T}\left(\beta_i+\frac{T_{ai}}{T}\right)$
2. If reverse Arrhenius parameters are provided, k'_{ri} is computed similar to above. If k_{ri} is computed based on k_{f_i} and the equilibrium constant K_{ci}, then its derivative is

$$\begin{aligned}k_{ri} = \frac{k_{f_i}}{K_{ci}} \Rightarrow k'_{ri} &= \frac{k'_{f_i}K_{ci}-k_{f_i}K'_{ci}}{K_{ci}^2} = \frac{\frac{k_{f_i}}{T}\left(\beta_i+\frac{T_{ai}}{T}\right)}{K_{ci}}-\frac{k_{f_i}}{K_{ci}}\frac{K'_{ci}}{K_{ci}}\\ &= k_{ri}\left(\frac{1}{T}\left(\beta_i+\frac{T_{ai}}{T}\right)-\frac{K'_{ci}}{K_{ci}}\right).\end{aligned}$$

Since $K_{ci} = \left(\frac{p_{atm}}{\Re}\right)^{\sum_{k=1}^{N_{\text{spec}}}\nu_{ki}}\exp\left(\sum_{k=1}^{N_{\text{spec}}}\nu_{ki}g_k\right) \Rightarrow \frac{K'_{ci}}{K_{ci}} = \sum_{k=1}^{N_{\text{spec}}}\nu_{ki}g'_k$. It follows that

$$k'_{ri} = k_{ri}\left(\frac{1}{T}\left(\beta_i+\frac{T_{ai}}{T}\right)-\sum_{k=1}^{N_{\text{spec}}}\nu_{ki}g'_k\right)$$

where g'_k is computed based on NASA polynomial fits as

$$g'_k = \frac{1}{T}\left(a_{0,k} - 1 + \frac{a_{5,k}}{T}\right) + \frac{a_{1,k}}{2} + T\left(\frac{a_{2,k}}{3} + T\left(\frac{a_{3,k}}{4} + \frac{a_{4,k}}{5}T\right)\right)$$

Efficient evaluation of the $\mathcal{F}$ terms

- Step 1:

$$\begin{aligned}
\mathcal{F}_{3+i,2} \equiv & 0 \\
\mathcal{F}_{3+i,3} = & \frac{1}{\rho}\frac{\partial \dot{\omega}_i}{\partial T} = \frac{W_i}{\rho}\left[\sum_{j=1}^{N_{\text{reac}}} \nu_{ij}\frac{\partial C_j}{\partial T}\left(\mathcal{R}_{fj} - \mathcal{R}_{rj}\right) + \sum_{j=1}^{N_{\text{reac}}} \nu_{ij} C_j \left(\mathcal{R}_{fj}\frac{k'_{fj}}{k_{fj}} - \mathcal{R}_{rj}\frac{k'_{rj}}{k_{rj}}\right)\right] \\
\mathcal{F}_{3+i,3+k} = & \frac{1}{\rho}\frac{\partial \dot{\omega}_i}{\partial Y_k} = \frac{1}{W_k}\frac{\partial \dot{\omega}_i}{\partial \mathfrak{X}_k} = \frac{W_i}{W_k}\left[\sum_{j=1}^{N_{\text{reac}}} \nu_{ij}\frac{\partial C_j}{\partial \mathfrak{X}_k}\left(\mathcal{R}_{fj} - \mathcal{R}_{rj}\right) + \sum_{j=1}^{N_{\text{reac}}} \nu_{ij} C_j \frac{\mathcal{R}_{fj}\nu'_{kj} - \mathcal{R}_{rj}\nu''_{kj}}{\mathfrak{X}_k}\right], \\
& k = 1, 2, \ldots, N_{\text{spec}}
\end{aligned}$$

  Here $\mathcal{R}_{fj}$ and $\mathcal{R}_{rj}$ are the forward and reverse parts, respectively of $\mathcal{R}_j$: $\mathcal{R}_{fj} = k_{fi}\prod_{j=1}^{N_{\text{spec}}} \mathfrak{X}_j^{\nu'_{ji}}$, $\mathcal{R}_{rj} = k_{ri}\prod_{j=1}^{N_{\text{spec}}} \mathfrak{X}_j^{\nu''_{ji}}$.

- Step 2: Once $\mathcal{F}_{3+i,3+k}$ are evaluated for all k, then $\mathcal{F}_{3+i,1}$ is computed as

$$\mathcal{F}_{3+i,1} = \frac{1}{\rho}\left(\frac{\partial \dot{\omega}_i}{\partial \rho} - \frac{\dot{\omega}_i}{\rho}\right) = \frac{1}{\rho}\left(-\frac{\dot{\omega}_k}{\rho} + \sum_{k=1}^{N_{\text{spec}}} \frac{Y_k}{W_k}\frac{\partial \dot{\omega}_i}{\partial \mathfrak{X}_k}\right) = \frac{1}{\rho}\left(-\frac{\dot{\omega}_k}{\rho} + \sum_{k=1}^{N_{\text{spec}}} Y_k \mathcal{F}_{3+i,3+k}\right)$$

- Step 3:

$$\begin{aligned}
\mathcal{F}_{3,1} = & \frac{1}{\rho c_p}\sum_{i=1}^{N_{\text{spec}}} h_i\left(\frac{\dot{\omega}_i}{\rho} - \frac{\partial \dot{\omega}_i}{\partial \rho}\right) = -\frac{1}{c_p}\sum h_i \mathcal{F}_{3+i,1} \\
\mathcal{F}_{3,2} \equiv & 0 \\
\mathcal{F}_{3,3} = & \frac{1}{\rho c_p}\left[\frac{1}{c_p}\frac{\partial c_p}{\partial T}\sum_{i=1}^{N_{\text{spec}}} h_i\dot{\omega}_i - \sum_{i=1}^{N_{\text{spec}}} c_{p_i}\dot{\omega}_i\right] - \frac{1}{\rho c_p}\sum_{i=1}^{N_{\text{spec}}} h_i\frac{\partial \dot{\omega}_i}{\partial T} \\
= & \frac{1}{\rho c_p}\left[\frac{1}{c_p}\frac{\partial c_p}{\partial T}\sum_{i=1}^{N_{\text{spec}}} h_i\dot{\omega}_i - \sum_{i=1}^{N_{\text{spec}}} c_{p_i}\dot{\omega}_i\right] - \frac{1}{c_p}\sum_{i=1}^{N_{\text{spec}}} h_i\mathcal{F}_{3+i,3}
\end{aligned}$$

Analytical expressions for $\overline{\mathcal{J}}$

- Line #1: **Pressure equation**. The source term for the pressure equation is assumed constant: $f_2 = \overline{f}_2 = const$.

$$\overline{\mathcal{J}}_{1,j} \equiv 0 \text{ for all } j \tag{4.29}$$

- Line #2: **Temperature equation** :

$$\overline{\mathcal{J}}_{2,1} = \mathcal{F}_{3,2} + \mathcal{F}_{3,1}\frac{\partial \rho}{\partial P} \tag{4.30}$$

$$\overline{\mathcal{J}}_{2,2} = \mathcal{F}_{3,3} + \mathcal{F}_{3,1}\frac{\partial \rho}{\partial T} \tag{4.31}$$

$$\overline{\mathcal{J}}_{2,2+k} = \mathcal{F}_{3,3+k} + \mathcal{F}_{3,1}\frac{\partial \rho}{\partial Y_k} \tag{4.32}$$

- Line #i=3,...,N_{spec}+2: **Species equations** :

$$\overline{\mathcal{J}}_{i,1} = \mathcal{F}_{i+1,2} + \mathcal{F}_{i+1,1}\frac{\partial \rho}{\partial P} \tag{4.33}$$

$$\overline{\mathcal{J}}_{i,2} = \mathcal{F}_{i+1,3} + \mathcal{F}_{i+1,1}\frac{\partial \rho}{\partial T} \tag{4.34}$$

$$\overline{\mathcal{J}}_{i,2+k} = \mathcal{F}_{i+1,3+k} + \mathcal{F}_{i+1,1}\frac{\partial \rho}{\partial Y_k}, k = 1,2,\ldots,N_{\text{spec}} \tag{4.35}$$

For this case density is a dependent variable, calculated based on the ideal gas equation of state:

$$\rho = \frac{P}{\Re T \sum_{k=1}^{N_{\text{spec}}} \frac{Y_k}{W_k}} \tag{4.36}$$

The partial derivaties of density with respect to the independent variables are computed as

$$\frac{\partial \rho}{\partial P} = \frac{\rho}{P}, \quad \frac{\partial \rho}{\partial T} = -\frac{\rho}{T}, \quad \frac{\partial \rho}{\partial Y_k} = -\frac{\rho \overline{W}}{W_k}. \tag{4.37}$$

Analytical expressions for $\widetilde{\mathcal{J}}$

- Line #1: **Pressure equation**. The source term for the pressure equation is assumed constant: $f_2 = \widetilde{f}_2 = const$.

$$\widetilde{\mathcal{J}}_{1,j} \equiv 0 \text{ for all } j \tag{4.38}$$

- Line #2: **Temperature equation**:

$$\widetilde{\mathcal{J}}_{2,1} = \mathcal{F}_{3,2} + \mathcal{F}_{3,1}\frac{\partial \rho}{\partial P} + \mathcal{F}_{3,L+3}\frac{\partial Y_L}{\partial P} \tag{4.39}$$

$$\widetilde{\mathcal{J}}_{2,2} = \mathcal{F}_{3,3} + \mathcal{F}_{3,1}\frac{\partial \rho}{\partial T} + \mathcal{F}_{3,L+3}\frac{\partial Y_L}{\partial T} \tag{4.40}$$

$$\widetilde{\mathcal{J}}_{2,2+k} = \mathcal{F}_{3,3+k_s} + \mathcal{F}_{3,1}\frac{\partial \rho}{\partial Y_k} + \mathcal{F}_{3,L+3}\frac{\partial Y_L}{\partial Y_k}, \tag{4.41}$$

$$k = 1,\ldots,L-1,L+1,\ldots,N_{\text{spec}} \tag{4.42}$$

where

$$k_s = \begin{cases} k & k < 3+L \\ k+1 & \text{otherwise} \end{cases} \tag{4.43}$$

- Line #i=3,…,N_{spec}+1: **Species equations** :

$$\widetilde{\mathcal{J}}_{i,1} = \mathcal{F}_{i+i_s,2} + \mathcal{F}_{i+i_s,1}\frac{\partial \rho}{\partial P} + \mathcal{F}_{i+i_s,L+3}\frac{\partial Y_L}{\partial P} \tag{4.44}$$

$$\widetilde{\mathcal{J}}_{i,2} = \mathcal{F}_{i+i_s,3} + \mathcal{F}_{i+i_s,1}\frac{\partial \rho}{\partial T} + \mathcal{F}_{i+i_s,L+3}\frac{\partial Y_L}{\partial T} \tag{4.45}$$

$$\widetilde{\mathcal{J}}_{i,2+k} = \mathcal{F}_{i+i_s,3+k_s} + \mathcal{F}_{i+i_s,1}\frac{\partial \rho}{\partial Y_k} + \mathcal{F}_{i+i_s,L+3}\frac{\partial Y_L}{\partial Y_k}, \tag{4.46}$$

$$k = 1,\ldots,L-1,L+1,\ldots,N_{\text{spec}} \tag{4.47}$$

where k_s is defined above and

$$i_s = \begin{cases} 1 & i+1 < 3+L \\ 2 & \text{otherwise} \end{cases} \tag{4.48}$$

For this case, density and mass fraction of species L are dependent variables, calculated as

$$\rho = \frac{P}{\Re T\left(\frac{1}{W_L} + \sum_{\substack{k=1 \\ k\neq L}}^{N_{\text{spec}}} Y_k\left(\frac{1}{W_k} - \frac{1}{W_L}\right)\right)}$$

$$Y_L = 1 - \sum_{\substack{k=1 \\ k\neq L}}^{N_{\text{spec}}} Y_k$$

The partial derivaties of density and Y_L with respect to the independent variables are computed as

$$\frac{\partial \rho}{\partial P} = \frac{\rho}{P}, \quad \frac{\partial \rho}{\partial T} = -\frac{\rho}{T}, \quad \frac{\partial \rho}{\partial Y_k} = -\rho\overline{W}\left(\frac{1}{W_k} - \frac{1}{W_L}\right)$$

$$\frac{\partial Y_L}{\partial P} = \frac{\partial Y_L}{\partial T} = 0, \quad \frac{\partial Y_L}{\partial Y_k} = -1.$$

Chapter 5

Examples

This section presents example C, C++, and Fortran 77 example codes that use some of the functionalities provided by TChem. The C and C++ examples requires the CVODE[1] library [2]. The Fortran example requires DVODE[2] library [1]. All these examples numerically integrate a 0D ignition model (1.1) for a range of parameters that control the initial fuel-oxidizer mixture, temperature, and pressure.

ign-c

The *main* function of the C code is contained in *ign.c*. The main function defines the parameters, then proceeds to set up the simulation: parameter setup (in *setup.c*), TChem initialization, CVODE setup *initcvode.c*. The time advancement of the sytem of ODE's (1.1) is implemented in *doIgn.c*. The function contained in *doIgnReinit.c* is a more compact version of doIgn.c (limited output) and takes advantage of the re-initialization option of CVODE to perform a series of time integrations while the kinetic model parameters are modified. An output function is provided in *output.c*. Functions providing the *rhs* and the Jacobian matrix corresponding to (1.1) are implemented in *rhsjac.c*.

Select sections of the the various components of the C code are described below.

ign.c

- Function *TC_initChem* is used to initalize TChem, while function *TC_setThermoPres* sends the thermodynamic pressure to the library.

```
TC_initChem( mechfile, thermofile, (int) withTab, 0.2) ;
pressure *= pfac ;
TC_setThermoPres(pressure) ;
```

  The parameters used in these functions are described below in *setup.c*.

- Function *TC_getSpos* returns the position in of a species in the list of species read by TChem from the kinetic model file.

```
for ( i = 0 ; i < specinno ; i++)
{
```

[1] https://computation.llnl.gov/casc/sundials
[2] https://computation.llnl.gov/casc/software.html

```
    int ispec = TC_getSpos( \&SpecName[i*LENGTHOFSPECNAME],
                                                strlen(\&SpecName[i*LENGTHOFSPECNAME]) ) ;
    ...
    scal[ispec+1] = SpecMsFr[i] ;
  }
```

- The ignition code expects mole fractions for the species that enter the fresh mixture. It the converts these values to mass fractions using *TC_getMl2Ms*

```
/* convert mole to mass fractions */
double *msfr = (double *) malloc(Nspec * sizeof(double)) ;
TC_getMl2Ms( \&(scal[1]), Nspec, msfr ) ;
for ( i = 1 ; i < Nspec+1 ; i++) scal[i] = msfr[i-1];
```

- Before the end of the execution the main function call *TC_reset* to clear all internal TChem arrays.

setup.c

The function contained in this file, *setup*, initializes parameters with default values, the reads from a setup file custom values for the paramters the user wants to change.

Below, *NiterMax* is the maximum number of time steps, *oFreq* is the frequency of solution output to disk, *Tini* is the initial mixture temperature, *Temp_id* is the temperature threshold for the ignition delay time, *deltat* is the initial time step size in seconds, *deltatMax* is the maximum time step size in seconds, *tEnd* is the end time for the time advancement, *deltaTemp* is the maximum allowed temperature change per time step. The thermodynamic pressure is given by the product $pfac \times 1.01325 \times 10^5 Pa$. *CVrelt* and *CVsmall* are tolerances for the CVODE library. The kinetic model file name is stored in *mechfile* and the name of the file with the NASA polynomials for the thermodynamic properties is stored in *thermofile*. *withTab* is a flag that specifies whether TChem should use interpolation tables to compute various properties, while *getIgnDel* is a flag allows the user to stop the integration once the ignition take place.

```
*NiterMax   = 100000     ; /* Maximum no. of iterations */
*oFreq      = 10         ; /* Output frequency          */
*Tini       = 1000.0     ; /* Initial temperature [K]   */
*Temp_id    = 1500.0     ; /* Threshold temperature for ignition delay [K]  */
*deltat     = 1.e-10     ; /* Initial time step size [s]  */
*deltatMax  = 1.e-4      ; /* Maximum time step size [s]  */
*tEnd       = 2.0e0      ; /* End integration time   [s]  */
*deltaTemp  = 1.0        ; /* Maximum temperature change per time step [K]  */
*pressure   = 1.01325e5  ; /* Pressure [Pa] */
*pfac       = 1.0        ; /* Pressure scale factor [ ]  */

*CVrelt         = 1.e-12 ;
*CVsmall        = 1.e-20 ;
*CVmaxord       = 5      ;
*CVmaxnumsteps = 10000   ;

strcpy(mechfile   ,"chem.inp" ) ;
strcpy(thermofile,"therm.dat") ;

*withTab = 0 ;  /* no tabulation          */
*getIgnDel = 0 ;/* no ignition delay stop */
```

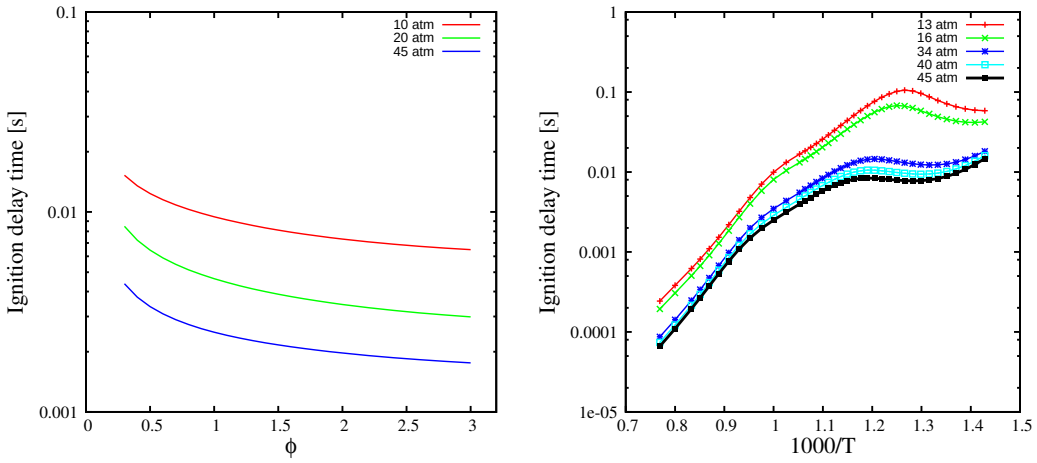

Figure 5.1. Left Frame: variation of the ignition delay times with the equivalence ratio (Φ) for iso-octane/air mixtures at various pressures and initial temperature of 1000K. Right Frame: Variation of ignition delay time with initial temperature for a stoichiometric iso-octane/mixtures at various pressures. The kinetic model involves 874 species and 3796 reactions [5]

rhsjac.c

The functions that compute *rhs* source terms and Jacobian matrices can be used for both full systems (temperature and all species) or restricted systems (temperature + all species - one species). In the later case, the mass fraction of the negelected species is set to ensure mass conservation. The user should specify at compile time "-DALLSPEC" for the former case. If this flag is neglected, the code will compiled for restricted systems.

The algorithms are *doIgn.c*, *doIgnReinit.c*, and *output.c* are straigthforward.

Sample results

Sample simulations are provided in *example/ign-c/run*. These simulations can be replicated using the bash scripts located in the corresponding directories. All script files have the extension *.x*. These scripts can also be used to generate the ignition delay time results for iso-Octane and Primary Reference Fuel (PRF) mixtures shown in Figs. 5.1 and 5.2, respectively.

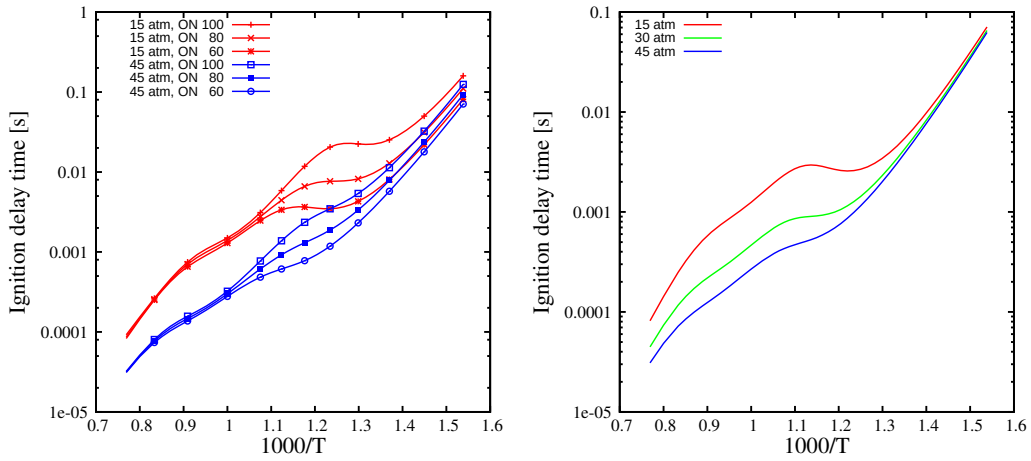

Figure 5.2. Left Frame: Variation of the ignition delay times with the initial mixture temperature for 100 ON, 80 ON and 60 ON at 15 atm and 45 atm. Right Frame: Variation of ignition delay time for 50 ON at 15 atm, 30 atm, and 45 atm. The kinetic model involves 1034 species and 4236 reactions [4, 3]

ign-cpp

This C++ example code is located in *example/ign-cpp/src*. Sample simulation data and scripts using a methane kinetic model (53 species, 325 reactions) [7] and a iso-octane kinetic model (871 species, 3792 reactions) [5] are located in *example/ign-c/run*. Subsequent directories' names are self-explanatory.

The *main* function of the C++ code is contained in *ign.cpp*. The main function handles the setup of the simulation, initialization of TChem, then transfers the control to the *StiffInteg* class. The *StiffInteg* class implements functions for the right hand side (*rhs*) and Jacobian matrix corresponding to the system of ordinarry differential equations 1.1. It handles the setup of CVODE and controls the time integration of 1.1. Select code sections are described below:

ign.cpp

- Setup parameters. Below, *NiterMax* is the maximum number of time steps, *oFreq* is the frequency of solution output to disk, *Tini* is the initial mixture temperature, *deltat* is the initial time step size in seconds, *deltatMax* is the maximum time step size in seconds, *tEnd* is the end time for the time advancement, *deltaTemp* is the maximum allowed temperature change per time step. The thermodynamic pressure is given by the product $pfac \times 1.01325 \times 10^5 Pa$. *CVrelt* and *CVsmall* are tolerances for the CVODE library. The kinetic model file name is stored in *mechfile* and the name of the file with the NASA polynomials for the thermodynamic properties is stored in *thermofile*. *withTab* is a flag that specifies whether TChem should use interpolation tables to compute various properties.

```
/* Set default values */
```

```
NiterMax = 20000 ;
oFreq    = 100   ;
Tini      = 1000.0    ; /* [K]  */
deltat    = 1.e-9     ; /* [s]  */
deltatMax = 1.e-6     ; /* [s]  */
tEnd      = 0.1e0     ; /* [s]  */
deltaTemp = 10.0      ; /* [K]  */
pfac   = 1.0       ; /* [ ]  */
CVrelt  = 1.e-12 ;
CVsmall = 1.e-20 ;
strcpy(mechfile  ,"chem.inp" ) ;
strcpy(thermofile,"therm.dat") ;
withTab = false ; /* no tabulation by default */
```

The default values of these parameters can be changed through the setup file. In addition to these parameters the setup files should contain mole fractions for the initial fuel-oxidizer mixture. Each species mole fractions should be provided on separate lines, starting with the keyword *spec* followed by the species name, and the mole fraction value. Examples setup files, "input.setup" are provided for each example.

- Function *TC_initChem* is used to initalize TChem, while function *TC_setThermoPres* sends the thermodynamic pressure to the library.

```
TC_initChem( mechfile, thermofile, (int) withTab, 0.2) ;
pressure *= pfac ;
TC_setThermoPres(pressure) ;
```

- The constructor for the *StiffInteg* class requires a pointer to the initial condition array *scal*, the number of species *Nspec*, and the maximum temperature change per time step.

```
StiffInteg intgcvode( scal, Nspec, deltaTemp ) ;
```

- The *compute* function of *StiffInteg* handles the time advancement of the system of ordinary differential equations (1.1).

```
intgcvode.compute( tEnd, &deltat, deltatMax, NiterMax, oFreq ) ;
```

StiffInteg.cpp

- Function *StiffInteg::chemrhs* handles the computation of the *rhs* source terms through a call to TChem. *StiffInteg::chemrhs* can handle chemical systems that include all species, $\left(T, Y_1, \ldots, Y_{N_{\text{spec}}}\right)$, or alternatively $\left(T, Y_1, \ldots, Y_{N_{\text{spec}}-1}\right)$. In the later case, the mass fraction of the last species is computed as $Y_{N_{\text{spec}}} = 1 - \sum_{k=1}^{N_{\text{spec}}-1} Y_k$. Below, *tempNmsfr* is the pointer to the array holding the temperature and species mass fractions, and *rhsvals* is the pointer to the array that stores the rhs values.

```
TC_getSrc ( tempNmsfr, Nspec_+1, rhsvals ) ;
```

- Function *StiffInteg::chemjac* handles the computation of the Jacobian matrix . Similar to *StiffInteg::chemrhs*, this function can provide matrices for both types of chemical systems mentioned above, by appropriate calls to the TChem library. Below, *tempNmsfr* is the pointer to the array holding the temperature and species mass fractions, *jactmp* is the pointer to the array that stores the Jacobian matrix, and *useJacAnl* is a flag which indicates whether or not to use analytical expressions for the Jacobian matrix terms (default is analytical).

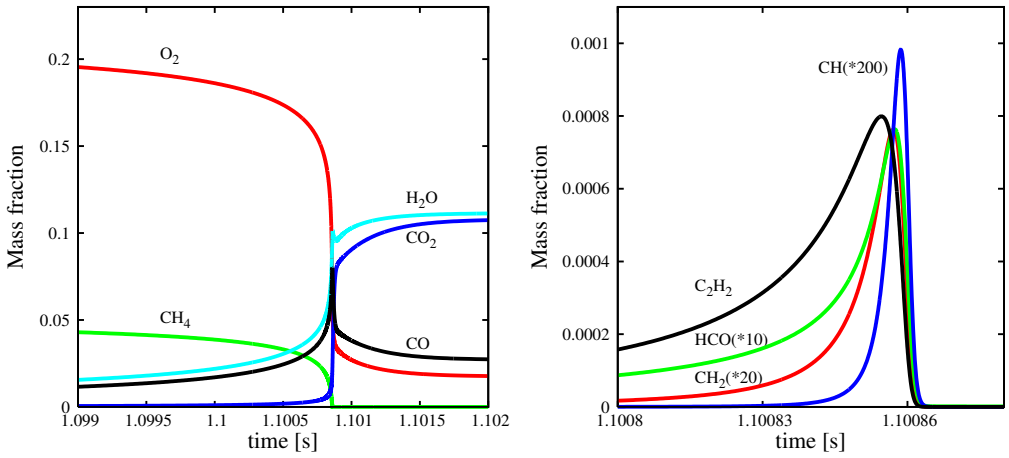

Figure 5.3. Mass fraction profiles during the constant pressure ignition of a stoichimetric methane-air mixture at 1atm. Methane combustion is modeled using the GRI-Mech v3.0 kinetic model (53 species, 325 reactions) [7].

```
TC_getJacTYN ( tempNmsfr, Nspec_, jactmp, (unsigned int) useJacAnl ) ;
TC_getJacTYNm1 ( tempNmsfr, Nspec_, jactmp, (unsigned int) useJacAnl ) ;
```

- *StiffInteg::compute* outputs data to several files during the time advancement: (1) *ignsol.dat* contains on each row the time [s], time step [s], temperature [K], and species mass fractions, (2) *ys.out* contains the time [s], temperature [K], and species mass fractions, (3) *cs.out* contains the time [s], temperature [K], and species molar concentrations [kmol/m^3], and (4) *h.out* contains the time [s] and mixture specific enthalpy [J/kg].

Sample results

Sample simulations are provided in *example/ign-cpp/run*. These simulations can be replicated using the bash scripts located in the corresponding directories. All script files have the extension *.x*. These scripts can also be used to generate the graphs shown in Fig. 5.3.

ign-f

The Fortran 77 example code is located in *example/ign-f/src*. Sample simulation data and scripts using a methane kinetic model (53 species, 325 reactions) and a one step model are located in *example/ign-f/run*. Subsequent directories' names are self-explanatory.

Most of the Fortran code is contained in *ign.f*. The variable names are similar to the ones described

above for *ign-c*. The subroutines that connect to TChem for the calculation of the rhs values and the Jacobian matrices are located in *rhsjac.f*

Note that in the subroutines calls that originate in the Fortran code, all parameters are passed by reference. In the TChem C-library, most functions contain parameters passed by reference as well as passed by value. In order to facilitate the inter-language calls, an additional interface is placed between the Fortran code and the TChem. The functions in this interface have all parameters passed by reference and forward the calls to the corresponding main C-interface with select parameters passed by value.

The sample Fortran code below, extracted from *ign.f*, is used to illustrate the above methodology. Here, *tcgetarhenfor* is used to extract the value of the activation energy for reaction # 0 from TChem. The value is modified, then it is sent back using *tcchgarhenfor*.

```
reacid = 0
...
posid  = 2
ierr = tcgetarhenfor(reacid,posid,acten)
...
acten = acten*1.4d0
ierr = tcchgarhenfor(reacid,posid,acten)
```

The definition for *tcgetarhenfor*, extracted from file *TC_for.c* is self explanatory:

```
int tcgetarhenfor_( int *ireac, int *ipos, double *val )
{
  int ans = 0 ;
  ans = TC_getArhenFor( *ireac, *ipos, val ) ;
  return ( ans );
}
```

The interface functions that facilitate the calls between Fortran codes and TChem are located in *TC_for.c*.

This page intentionally left blank.

Chapter 6

Library Functions

TC_chg.c

1. *int* **TC_chgArhenFor**(int i_{reac}, int i_{pos}, double *newval*)
   Change parameters for forward rate constants.

   - i_{reac} - reaction index.
   - i_{pos} - index of parameter to be changed (0) pre-exponential factor (1) temperature exponent, (2) activation energy
   - *newval* - new parameter value

2. *int* **TC_chgArhenForBack**(int i_{reac}, int i_{pos})
   Reverse changes for forward rate constants' parameters

   - i_{reac} - reaction index.
   - i_{pos} - index of parameter to be changed (0) pre-exponential factor (1) temperature exponent, (2) activation energy

3. *int* **TC_chgArhenRev**(int i_{reac}, int i_{pos}, double *newval*)
   Change parameters for forward rate constants.

   - i_{reac} - reaction index.
   - i_{pos} - index of parameter to be changed (0) pre-exponential factor (1) temperature exponent, (2) activation energy
   - *newval* - new parameter value

4. *int* **TC_chgArhenRevBack**(int i_{reac}, int i_{pos})
   Reverse changes for forward rate constants' parameters

   - i_{reac} - reaction index.
   - i_{pos} - index of parameter to be changed (0) pre-exponential factor (1) temperature exponent, (2) activation energy

TC_edit.c

1. *int* **TC_reset**()
   Frees all memory and sets variables to 0 so that *TC_initChem* can be called again without a memory leak. Not designed for use with tables.

TC_init.c

1. *int* **TC_initChem**(char *$mechfile$,char *$thermofile$, int tab, double $delT$)
   Initializes the library:
   - $mechfile$ - file containing the kinetic model (in chemkin format)
   - $thermofile$ - file containing the coefficients for NASA polynomials.
   - tab - flag: 0-use direct evaluations to compute various properties, 1-use interpolation tables
   - $delT$ - temperature step size for the interpolation tables; not used if tab=0
2. *void* **TC_setRefVal**(double rho_{ref}, double p_{ref}, double T_{ref}, double W_{ref}, double Da_{ref}, double omg_{ref}, double cp_{ref}, double h_{ref}, double tim_{ref})
   Send reference values to the library:
   - these are the density (rho_{ref}), pressure (p_{ref}), temperature (T_{ref}), molecular weight (W_{ref}), Damkohler number (Da_{ref}), reaction rate (omg_{ref}), specific heat at constant pressure (cp_{ref}), specific enthalpy (h_{ref}), time (tim_{ref}).
3. *void* **TC_setNonDim**()
   Set's the library to use non-dimensional input and functions. Can be called only after **TC_setRefVal** was called.
4. *void* **TC_setDim**()
   Set's the library to use dimensional input and functions.
5. *void* **TC_setThermoPres**(double *pressure*)
   Sends the thermodynamic pressure to the library.
   - *pressure* - dimensiona/non-dimensional thermodynamic pressure. For dimensional cases, SI units are used $[\mathrm{N/m^2}]$

TC_mlms.c

1. *int* **TCDND_getMs2Cc**(double *$scal$,int N_{vars},double *$concX$)
   Computes molar concentrations $\mathfrak{X}$'s based on temperature T and species mass fractions Y's.

$$\mathfrak{X}_k = Y_k \cdot \frac{\rho}{W_k}$$

If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

- *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], mass fractions Y [].
- N_{vars} - no. of variables $= N_{\text{spec}}+1$
- *concX* - array of doubles containing species molar concentrations $\mathfrak{X}$'s [kmol/m^3]

2. *int* **TC_getMs2Cc**(double **scal*,int N_{vars},double **concX*)
Computes molar concentrations $\mathfrak{X}$'s based on temperature T and species mass fractions Y's.

$$\mathfrak{X}_k = Y_k \cdot \frac{\rho}{W_k}$$

- *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], mass fractions Y [].
- N_{vars} - no. of variables $= N_{\text{spec}}+1$
- *concX* - array of doubles containing species molar concentrations [kmol/m^3]

3. *int* **TCDND_getMl2Ms**(double **Xspec*,int N_{spec},double **Yspec*)
Transforms mole fractions X's to mass fractions Y's (same as TC_getMl2Ms()).

$$Y_k = X_k \cdot W_k / \overline{W}$$

- *Xspec* - array of N_{spec} mole fractions X [].
- N_{spec} - no. of species
- *Yspec* - array of N_{spec} mole fractions Y [].

4. *int* **TC_getMl2Ms**(double **Xspec*,int N_{spec},double **Yspec*)
Transforms mole fractions X's to mass fractions Y's.

$$Y_k = X_k \cdot W_k / \overline{W}$$

- *Xspec* - array of N_{spec} mole fractions X [].
- N_{spec} - no. of species
- *Yspec* - array of N_{spec} mole fractions Y [].

5. *int* **TCDND_getMs2Ml**(double **Yspec*,int N_{spec},double **Xspec*)
Transforms mass fractions Y's to mole fractions X's (same as TC_getMs2Ml()).

$$X_k = Y_k \cdot \overline{W} / W_k$$

- *Yspec* - array of N_{spec} mole fractions Y [].
- N_{spec} - no. of species
- *Xspec* - array of N_{spec} mole fractions X [].

6. *int* **TC_getMs2Ml**(double *$Yspec$,int N_{spec},double *$Xspec$)
   Transforms mass fractions Y's to mole fractions X's.

$$X_k = Y_k \cdot \overline{W}/W_k$$

   - $Yspec$ - array of N_{spec} mole fractions Y [].
   - N_{spec} - no. of species
   - $Xspec$ - array of N_{spec} mole fractions X [].

7. *int* **TCDND_getMs2Wmix**(double *$Yspec$,int N_{spec},double *$Wmix$)
   Computes mixture molecular weight $\overline{W}$ based on species mass fractions Y's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

$$\overline{W} = \left(\sum_{k=1}^{N_{spec}} Y_k/W_k \right)^{-1}$$

   - $Yspec$ - array of N_{spec} mole fractions Y [].
   - N_{spec} - no. of species
   - $Wmix$ - pointer to mixture molecular weight [kg/kmol]=[g/mol].

8. *int* **TC_getMs2Wmix**(double *$Yspec$,int N_{spec},double *$Wmix$)
   Computes mixture molecular weight $\overline{W}$ based on species mass fractions Y's.

$$\overline{W} = \left(\sum_{k=1}^{N_{spec}} Y_k/W_k \right)^{-1}$$

   - $Yspec$ - array of N_{spec} mole fractions Y [].
   - N_{spec} - no. of species
   - $Wmix$ - pointer to mixture molecular weight [kg/kmol]=[g/mol].

9. *int* **TCDND_getMl2Wmix**(double *$Xspec$,int N_{spec},double *$Wmix$)
   Computes mixture molecular weight $\overline{W}$ based on species mole fractions X's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

$$\overline{W} = \frac{1}{W_{ref}} \sum_{k=1}^{N_{spec}} X_k W_k$$

   - $Xspec$ - array of N_{spec} mole fractions X [].
   - N_{spec} - no. of species
   - $Wmix$ - pointer to mixture molecular weight [kg/kmol]=[g/mol].

10. *int* **TC_getMl2Wmix**(double *$Xspec$,int N_{spec},double *$Wmix$)
Computes mixture molecular weight $\overline{W}$ based on species mole fractions X's.

$$\overline{W} = \sum_{k=1}^{N_{spec}} X_k W_k$$

- $Xspec$ - array of N_{spec} mole fractions X [].
- N_{spec} - no. of species
- $Wmix$ - pointer to mixture molecular weight [kg/kmol]=[g/mol].

TC_rr.c

1. *int* **TC_getNreac**()
Returns number of reactions N_{reac}.

2. *int* **TC_getStoiCoef**(int N_{spec}, int N_{reac}, double *stoicoef)
Returns stoichiometric coefficients' matrix. The stoichiometric coefficient for species j in reaction i is stored at position $i \cdot N_{\text{spec}} + j$. It assumes that *stoicoef* was dimentioned to at least $N_{\text{reac}} \times N_{\text{spec}}$

   - N_{spec} - no. of species
   - N_{reac} - no. of reactions
   - stoicoef - pointer to an array of doubles with the stoichiometric coefficients.

3. *int* **TC_getStoiCoefReac**(int N_{spec}, int N_{reac}, int i_{reac}, int idx, double *stoicoef)
Returns stoichiometric coefficients' array for reaction i_{reac} for either reactants ($idx = 0$) or products ($idx = 1$). The stoichiometric coefficient for species j in reaction i_{reac} is stored at position j. It assumes that *stoicoef* was dimentioned to at least N_{spec}

   - N_{spec} - no. of species
   - N_{reac} - no. of reactions
   - i_{reac} - reaction index
   - idx - 0-reactants, 1-products
   - stoicoef - pointer to an array of doubles with the stoichiometric coefficients.

4. *int* **TC_getArhenFor**(int i_{reac}, int i_{pos}, double *val)
Return current value of the Arrhenius parameters for forward rate constants. Return -1 if no data available, otherwise return 0 and store value in *val*.

   - i_{reac} - reaction index.
   - i_{pos} - index of Arrhenius parameter (0) pre-exponential factor (1) temperature exponent, (2) activation energy
   - val - pointer to the value of Arrhenius parameter.

5. *int* **TC_getArhenRev**(int i_{reac}, int i_{pos}, double **val*)
   Return current value of Arrhenius parameters for reverse rate constants. Return -1 if no data available, otherwise return 0 and store value in *val*.

   - i_{reac} - reaction index.
   - i_{pos} - index of Arrhenius parameter (0) pre-exponential factor (1) temperature exponent, (2) activation energy
   - *val* - pointer to the value of Arrhenius parameter

6. *int* **TCDND_getTY2RRml**(double **scal*, int N_{vars}, double **omega*)
   Returns molar reaction rates, $\dot{\omega}_i$, based on temperature T and mass fractions Y's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - *omega* - array of N_{spec} molar reaction rates [kmol/(m^3·s)]

7. *int* **TC_getTY2RRml**(double **scal*, int N_{vars}, double **omega*)
   Returns molar reaction rates, $\dot{\omega}_i$, based on T and Y's.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - *omega* - array of N_{spec} molar reaction rates $\dot{\omega}_i$ [kmol/(m^3·s)].

8. *int* **TCDND_getTY2RRms**(double **scal*, int N_{vars}, double **omega*)
   Returns mass reaction rates based on temperature T and species mass fractions Y's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - *omega* - array of N_{spec} mass reaction rates [kg/(m^3·s)]

9. *int* **TC_getTY2RRms**(double **scal*, int N_{vars}, double **omega*)
   Returns mass reaction rates based on temperature T and species mass fractions Y's.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - *omega* - array of N_{spec} mass reaction rates [kg/(m^3·s)].

10. *int* **TCDND_getTXC2RRml**(double **scal*, int N_{vars}, double **omega*)
Returns molar reaction rates based on temperature T and molar concentrations $\mathfrak{X}$'s. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,\mathfrak{X}_1,\mathfrak{X}_2,...,\mathfrak{X}_N)$, temperature T [K], molar concentrations $\mathfrak{X}$ [kmol/m^3].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - *omega* - array of N_{spec} molar reaction rates [kmol/(m^3·s)].

11. *int* **TC_getTXC2RRml**(double **scal*, int N_{vars}, double **omega*)
Returns molar reaction rates based on temperature T and molar concentrations $\mathfrak{X}$'s.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,\mathfrak{X}_1,\mathfrak{X}_2,\ldots,\mathfrak{X}_N)$, temperature T [K], molar concentrations $\mathfrak{X}$ [kmol/m^3].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - *omega* - array of N_{spec} molar reaction rates [kmol/(m^3·s)].

12. *int* **TCDND_getTXC2RRms**(double **scal*, int N_{vars}, double **omega*)
Returns non-dimensional mass reaction rates based on temperature T and molar concentrations $\mathfrak{X}$'s. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,\mathfrak{X}_1,\mathfrak{X}_2,...,\mathfrak{X}_N)$, temperature T [K], molar concentrations $\mathfrak{X}$ [kmol/m^3].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - *omega* - array of N_{spec} mass reaction rates [kg/(m^3·s)].

13. *int* **TC_getTXC2RRms**(double **scal*, int N_{vars}, double **omega*)
Returns mass reaction rates based on temperature T and molar concentrations $\mathfrak{X}$'s.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,\mathfrak{X}_1,\mathfrak{X}_2,...,\mathfrak{X}_N)$, temperature T [K], molar concentrations $\mathfrak{X}$ [kmol/m^3].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - *omega* - array of N_{spec} mass reaction rates [kg/(m^3·s)].

14. *int* **TC_getRops**(double **scal*, int N_{vars}, double **datarop*)
Returns rate-of-progress variables based on temperature T and species mass fractions Y's.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,Y_1,Y_2,...,Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - datarop - array of N_{reac} rate-of-progress variables [kmol/(m^3·s)].

15. *int* **TC_getRfRb**(double **scal*, int N_{vars}, double *dataRfrb)
    Returns rate-of-progress variables based on temperature T and species mass fractions Y's.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - dataRfrb - array of N_{reac} forward rate-of-progress variables and N_{reac} reverse rate-of-progress variables [kmol/(m^3·s)].

TC_spec.c

User functions:

1. *int* **TC_getNspec**()
   Returns number of species.

2. *int* **TC_getNelem**()
   RReturns number of elements.

3. *int* **TC_getNvars**()
   Returns number of variables, currently no. of species plus one.

4. *int* **TC_getSnames**(int N_{spec},char ***snames*)
   Returns species names.

    - N_{spec} - no. of species, needs to match the library's internal value.
    - *snames* - array of charecters, the allocation needs to be at least N_{spec}*LENGTHOFSPECNAME. Currently LENGTHOFSPECNAME is set to 32. Name of species i starts at position $i * LENGTHOFSPECNAME$ in the array.

5. *int* **TC_getSnameLen**()
   Returns length of species names.

6. *int* **TC_getSpos**(const char ***sname*, const int *slen*)
   Returns index of species *sname*.

    - *sname* - name of species.
    - *slen* - length of species name

7. *int* **TC_getSmass**(int N_{spec},double ***Wi*)
   Returns molar masses for all species.

    - N_{spec} - no. of species, needs to match the library's internal value.
    - *Wi* - pointer to an array with molecular masses [kg/kmol] for all species. The allocation size should be at least N_{spec}.

TC_src.c

1. *int* **TCDND_getSrc**(double **scal*,int N_{vars},double **omega*)
   Returns the source term for
   $$\frac{\partial T}{\partial t} = \omega_0, \frac{\partial Y_i}{\partial t} = \omega_i,$$
   based on temperature T and species mass fractions Y's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - *omega* : pointer to an array of doubles with the source terms for temperature and species mass fractions equations: *omega*[0] : [K/s], *omega*[1... N_{spec}] : [1/s]

2. *int* **TC_getSrc**(double **scal*,int N_{vars},double **omega*)
   Returns the source term for
   $$\frac{\partial T}{\partial t} = \omega_0, \frac{\partial Y_i}{\partial t} = \omega_i,$$
   based on temperature T and species mass fractions Y's.

   - *scal* - pointer to an pointer to an of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - *omega* : pointer to an array of doubles with the source terms for temperature and species mass fractions equations: *omega*[0] : [K/s], *omega*[1... N_{spec}] : [1/s]

3. *int* **TCDND_getSrcCons**(double **scal*,int N_{vars},double **omega*)
   Returns the source term for
   $$\frac{\partial \rho}{\partial t} = \omega_0, \rho\frac{\partial Y_i}{\partial t} = \omega_i,$$
   based on density ρ and species mass fractions Y's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(\rho, Y_1, Y_2, ..., Y_N)$, density ρ [kg/m^3], species mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - *omega* : pointer to an array of doubles with the source terms for density and species mass fractions equations: *omega*[0] : [kg/(m^3·s)], *omega*[1... N_{spec}] : [kg/(m^3·s)]

4. *int* **TC_getSrcCons**(double **scal*,int N_{vars},double **omega*)
   Returns source term for
   $$\frac{\partial \rho}{\partial t} = \omega_0, \rho \frac{\partial Y_i}{\partial t} = \omega_i,$$
   based on density ρ and species mass fractions Y's.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(\rho, Y_1, Y_2, ..., Y_N)$, density ρ [kg/m^3], species mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - *omega* : pointer to an array with the source terms for density and species mass fractions equations: *omega*[0] : [kg/(m^3·s)], *omega*[1... N_{spec}] : [kg/(m^3·s)]

5. *int* **TCDND_getJacTYNm1**(double **scal*, int N_{spec}, double *jac, unsigned int useJacAnl)
   Computes Jacobian matrix for the system $(T, Y_1, Y_2, \ldots, Y_{N-1})$ based on temperature T and species mass fractions Y's using either analytical expressions of numerical derivatives. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{spec} - no. of species N_{spec}
   - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j * N_{\text{spec}} + i]$.
   - useJacAnl - flag for Jacobian type (1-analytical,other values-numerical)

6. *int* **TC_getJacTYNm1**(double **scal*, int N_{spec}, double *jac, unsigned int useJacAnl)
   Computes Jacobian matrix for the system $(T, Y_1, Y_2, \ldots, Y_{N-1})$ based on temperature T and species mass fractions Y's using either analytical expressions of numerical derivatives.

   - *scal* - pointer to an pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{spec} - no. of species N_{spec}
   - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j * N_{\text{spec}} + i]$.
   - useJacAnl - flag for Jacobian type (1-analytical,other values-numerical)

7. *int* **TCDND_getJacTYNm1anl**(double **scal*, int N_{spec}, double *jac)
   Computes Jacobian matrix for the system $(T, Y_1, Y_2, \ldots, Y_{N-1})$ based on temperature T and species mass fractions Y's using either analytical expressions. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].

- N_{spec} - no. of species N_{spec}
- jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j * N_{\text{spec}} + i]$.

8. *int* **TC_getJacTYNm1anl**(double **scal*, int N_{spec}, double *jac)
Computes Jacobian matrix for the system $(T, Y_1, Y_2, \ldots, Y_{N-1})$ based on temperature T and species mass fractions Y's using either analytical expressions.

   - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{spec} - no. of species N_{spec}
   - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j * N_{\text{spec}} + i]$.

9. *int* **TCDND_getJacTYN**(double **scal*, int N_{spec}, double *jac, unsigned int useJacAnl)
Computes Jacobian matrix for the system $(T, Y_1, Y_2, \ldots, Y_N)$ based on temperature T and species mass fractions Y's using either analytical expressions of numerical derivatives. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{spec} - no. of species N_{spec}
   - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j * (N_{\text{spec}} + 1) + i]$.
   - useJacAnl - flag for Jacobian type (1-analytical,other values-numerical)

10. *int* **TC_getJacTYN**(double **scal*, int N_{spec}, double *jac, unsigned int useJacAnl)
Computes Jacobian matrix for the system $(T, Y_1, Y_2, \ldots, Y_N)$ based on temperature T and species mass fractions Y's using either analytical expressions of numerical derivatives.

    - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{spec} - no. of species N_{spec}
    - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j * (N_{\text{spec}} + 1) + i]$.
    - useJacAnl - flag for Jacobian type (1-analytical,other values-numerical)

11. *int* **TCDND_getJacTYNanl**(double **scal*, int N_{spec}, double *jac)
Computes Jacobian matrix for the system $(T, Y_1, Y_2, \ldots, Y_N)$ based on temperature T and species mass fractions Y's using either analytical expressions. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

- *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,Y_1,Y_2,...,Y_N)$, temperature T [K], species mass fractions Y [].
- N_{spec} - no. of species N_{spec}
- jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j*(N_{\text{spec}}+1)+i]$.

12. *int* **TC_getJacTYNanl**(double **scal*, int N_{spec}, double *jac)
Computes Jacobian matrix for the system $(T,Y_1,Y_2,\ldots,Y_N)$ based on temperature T and species mass fractions Y's using either analytical expressions.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,Y_1,Y_2,...,Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{spec} - no. of species N_{spec}
    - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j*(N_{\text{spec}}+1)+i]$.

13. *int* **TC_getJacRPTYN**(double **scal*, int N_{spec}, double *jac, unsigned int useJacAnl)
Computes Jacobian matrix for the system $(\rho,P,T,Y_1,Y_2,\ldots,Y_N)$ based on temperature T and species mass fractions Y's using either analytical expressions of numerical derivatives.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,Y_1,Y_2,...,Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{spec} - no. of species N_{spec}
    - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j*(N_{\text{spec}}+3)+i]$.
    - useJacAnl : flag for Jacobian type (1-analytical,other values-numerical).

14. *int* **TC_getJacRPTYNanl**(double **scal*, int N_{spec}, double *jac)
Computes the Jacobian matrix for the system $(\rho,P,T,Y_1,Y_2,\ldots,Y_N)$ based on temperature T and species mass fractions Y's using analytical expressions.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,Y_1,Y_2,...,Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{spec} - no. of species N_{spec}
    - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j*(N_{\text{spec}}+3)+i]$.

15. *int* **TC_getJacRPTYNnum**(double **scal*, int N_{spec}, double *jac)
Computes the Jacobian matrix for the system $(\rho,P,T,Y_1,Y_2,\ldots,Y_N)$ based on temperature T and species mass fractions Y's using numerical derivatives.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T,Y_1,Y_2,...,Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{spec} - no. of species N_{spec}
    - jac - pointer to array of doubles with the Jacobian matrix J. Element J_{ij} is stored at $jac[j*(N_{\text{spec}}+3)+i]$.

TC_thermo.c

1. *int* **TCDND_getRhoMixMs**(double **scal*,int N_{vars},double *rhomix)
   Computes density based on temperature T and species mass fractions Y's using the equation of state. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - rhomix - pointer to mixture density [kg/m^3]

2. *int* **TC_getRhoMixMs**(double **scal*,int N_{vars},double *rhomix)
   Computes density based on temperature T and species mass fractions Y's using the equation of state.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - rhomix - pointer to mixture density [kg/m^3]

3. *int* **TCDND_getRhoMixMl**(double **scal*,int N_{vars},double *rhomix)
   Computes density based on temperature T and species mole fractions X's using the equation of state. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, X_1, X_2, ..., X_N)$, temperature T [K], species mole fractions X [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - rhomix - pointer to mixture density [kg/m^3]

4. *int* **TC_getRhoMixMl**(double **scal*,int N_{vars},double *rhomix)
   Computes density based on temperature T and species mole fractions X's using the equation of state.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, X_1, X_2, ..., X_N)$, temperature T [K], species mole fractions X [].
   - N_{vars} - no. of variables $= N_{\text{spec}}+1$
   - rhomix - pointer to mixture density [kg/m^3]

5. *int* **TCDND_getTMixMs**(double **scal*,int N_{vars},double *Tmix)
   Computes temperature based on density ρ and species mass fractions Y's using the equation of state. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

   - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(\rho, Y_1, Y_2, ..., Y_N)$, density ρ [kg/m^3], species mass fractions Y [].

- N_{vars} - no. of variables $= N_{\text{spec}} + 1$
- Tmix - pointer to temperature [K]

6. *int* **TC_getTMixMs**(double **scal*,int N_{vars},double *Tmix)
   Computes temperature based on density ρ and species mass fractions Y's using the equation of state.
   - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(\rho, Y_1, Y_2, ..., Y_N)$, density ρ [kg/m^3], species mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}} + 1$
   - Tmix - pointer to temperature [K]

7. *int* **TCDND_getTMixMl**(double **scal*,int N_{vars},double *Tmix)
   Computes temperature based on density ρ and species mole fractions X's using the equation of state. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.
   - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(\rho, X_1, X_2, ..., X_N)$, density ρ [kg/m^3], species mole fractions X [].
   - N_{vars} - no. of variables $= N_{\text{spec}} + 1$
   - Tmix - pointer to temperature [K]

8. *int* **TC_getTMixMl**(double **scal*,int N_{vars},double *Tmix)
   Computes temperature based on density ρ and species mole fractions X's using the equation of state.
   - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(\rho, X_1, X_2, ..., X_N)$, density ρ [kg/m^3], species mole fractions X [].
   - N_{vars} - no. of variables $= N_{\text{spec}} + 1$
   - Tmix - pointer to temperature [K]

9. *int* **TCDND_getMs2CpMixMs**(double **scal*,int N_{vars},double *cpmix)
   Computes mixture specific heat at constant pressure based on temperature T and species mass fractions Y's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.
   - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
   - N_{vars} - no. of variables $= N_{\text{spec}} + 1$
   - cpmix - pointer to mixture specific heat at constant pressure [J/(kg·K)]

10. *int* **TC_getMs2CpMixMs**(double **scal*,int N_{vars},double *cpmix)
    Computes mixture specific heat at constant pressure based on temperature T and species mass fractions Y's.
    - *scal* -pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].

- N_{vars} - no. of variables $= N_{\text{spec}} + 1$
- cpmix - pointer to mixture specific heat at constant pressure [J/(kg·K)]

11. *int* **TCDND_getMs2CvMixMs**(double **scal*,int N_{vars},double *cvmix)
Computes mixture specific heat at constant volume based on temperature T and species mass fractions Y's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{vars} - no. of variables $= N_{\text{spec}} + 1$
    - cvmix - pointer to mixture specific heat at constant volume [J/(kg·K)]

12. *int* **TC_getMs2CvMixMs**(double **scal*,int N_{vars},double *cvmix)
Computes mixture specific heat at constant volume based on temperature T and species mass fractions Y's.

    - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{vars} - no. of variables $= N_{\text{spec}} + 1$
    - cvmix - pointer to mixture specific heat at constant volume [J/(kg·K)]

13. *int* **TCDND_getMl2CpMixMl**(double **scal*,int N_{vars},double *cvmix)
Computes mixture heat capacity at constant pressure based on temperature T and species mole fractions X's. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, X_1, X_2, ..., X_N)$, temperature T [K], species mole fractions X [].
    - N_{vars} - no. of variables $= N_{\text{spec}} + 1$
    - cvmix - pointer to mixture heat capacity at constant volume [J/(kmol·K)]

14. *int* **TC_getMl2CpMixMl**(double **scal*,int N_{vars},double *cvmix)
Computes mixture heat capacity at constant pressure based on temperature T and species mole fractions X's.

    - *scal* - pointer to an array of $N_{\text{spec}} + 1$ doubles $(T, X_1, X_2, ..., X_N)$, temperature T [K], species mole fractions X [].
    - N_{vars} - no. of variables $= N_{\text{spec}} + 1$
    - cvmix - pointer to mixture heat capacity at constant volume [J/(kmol·K)]

15. *int* **TCDND_getCpSpecMs**(double t,int N_{spec},double *cpi)
Computes species specific heat at constant pressure based on temperature T. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

- t - temperature T [K]
- N_{spec} - no. of species
- cpi - array with species specific heats at constant pressure [J/(kg·K)]

16. *int* **TC_getCpSpecMs**(double t,int N_{spec},double *cpi)
Computes species specific heat at constant pressure based on temperature T.

    - t - temperature T [K]
    - N_{spec} - no. of species
    - cpi - array with species specific heats at constant pressure [J/(kg·K)]

17. *int* **TCDND_getCpSpecMl**(double t,int N_{spec},double *cpi)
Computes species heat capacities at constant pressure based on temperature T. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - t - temperature T [K]
    - N_{spec} - no. of species
    - cpi - array with species heat capacities at constant pressure [J/(kmol·K)]

18. *int* **TC_getCpSpecMl**(double t,int N_{spec},double *cpi)
Computes species heat capacities at constant pressure based on temperature.

    - t - temperature T [K] Y [].
    - N_{spec} - no. of species
    - cpi - array with species heat capacities at constant pressure [J/(kmol·K)]

19. *int* **TCDND_getMs2HmixMs**(double **scal*,int N_{vars},double **hmix*)
Computes mixture specific enthalpy based on temperature and species mass fractions. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - *hmix* - pointer to mixture specific enthalpy [J/kg].

20. *int* **TC_getMs2HmixMs**(double **scal*,int N_{vars},double **hmix*)
Computes mixture specific enthalpy based on temperature and species mass fractions.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, Y_1, Y_2, ..., Y_N)$, temperature T [K], species mass fractions Y [].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - *hmix* - pointer to mixture specific enthalpy [J/kg].

21. *int* **TCDND_getMl2HmixMl**(double **scal*,int N_{vars},double **hmix*)
Computes mixture molar enthalpy based on temperature and species mole fractions. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, X_1, X_2, ..., X_N)$, temperature T [K], species mole fractions X [].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - *hmix* - pointer to mixture molar enthalpy [J/kmol].

22. *int* **TC_getMl2HmixMl**(double **scal*,int N_{vars},double **hmix*)
Computes mixture molar enthalpy based on temperature T and species mole fractions X's.

    - *scal* - pointer to an array of $N_{\text{spec}}+1$ doubles $(T, X_1, X_2, ..., X_N)$, temperature T [K], species mole fractions X [].
    - N_{vars} - no. of variables $= N_{\text{spec}}+1$
    - *hmix* - pointer to mixture molar enthalpy [J/kmol].

23. *int* **TCDND_getHspecMs**(double *t*,int N_{spec},double **hi*)
Computes species specific enthalpies based on temperature T. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - *t* - temperature T [K]
    - N_{spec} - no. of species
    - *hi* - array of N_{spec} doubles with species specific enthalpies [J/kg]

24. *int* **TC_getHspecMs**(double *t*,int N_{spec},double **hi*)
Computes species specific enthalpies based on temperature T.

    - *t* - temperature T [K]
    - N_{spec} - no. of species
    - *hi* - array of N_{spec} doubles with species specific enthalpies [J/kg]

25. *int* **TCDND_getHspecMl**(double *t*,int N_{spec},double **hi*)
Computes species molar enthalpies based on temperature T. If the non-dimensional flag is ON (using *TC_setNonDim*) this function expects non-dimensional input and will provide non-dimensional output.

    - *t* - temperature T [K]
    - N_{spec} - no. of species
    - *hi* - array of N_{spec} doubles with species molar enthalpies [J/kmol]

26. *int* **TC_getHspecMl**(double *t*,int N_{spec},double **hi*)
Computes species molar enthalpies based on temperature T.

- *t* - temperature *T* [K]
- N_{spec} - no. of species
- *hi* - array of N_{spec} doubles with species molar enthalpies [J/kmol]

Chapter 7

Nomenclature

- Temperature : T - $[K]$.
- Density : ρ - $\left[\frac{kg}{m^3}\right]$.
- Thermodynamic pressure : P - $\left[\frac{kg}{m \cdot s}\right]$.
- Mass and mole fraction of species k: Y_k and X_k, respectively - [].
- Molar concentration of species k: $\mathfrak{X}_k$, respectively - $[\frac{kmol}{m^3}]$.

- Molecular weight of species k and of the mixture: W_k and $\overline{W}$, respectively - $\left[\frac{g}{mol}\right] = \left[\frac{kg}{kmol}\right]$.
- Universal gas constant, $\Re = 8.314472 \times 10^3 \left[\frac{J}{kmol \cdot K}\right] = 1.98721 \left[\frac{cal}{mol \cdot K}\right]$.
- Molar heat capacity at constant pressure (species k and mixture): $C_{p,k}$ and C_p, respectively - $\left[\frac{J}{kmol \cdot K}\right]$.
- Specific heat capacity at constant pressure (species k and mixture): $c_{p,k}$ and c_p, respectively - $\left[\frac{J}{kg \cdot K}\right]$.
- Molar enthalpy (species k and mixture): H_k and H, respectively - $\left[\frac{J}{kmol}\right]$.
- Specific enthalpy (species k and mixture): h_k and h, respectively - $\left[\frac{J}{kg}\right]$.
- Mass reaction rate of species k: $\dot{\omega}_k$ - $\left[\frac{kg}{m^3 \cdot s}\right]$.
- Molar reaction rate of species k: $\dot{\omega}_k$ - $\left[\frac{kmol}{m^3 \cdot s}\right]$.

Non-dimensional values

- Temperature : $T^* = T/T_{\text{ref}}$.
- Density : $\rho^* = \rho/\rho_{\text{ref}}$.

- Molar concentration of species k: $\mathfrak{X}_k^* = \mathfrak{X}_k \cdot W_{\text{ref}}/\rho_{\text{ref}}$.
- Molecular weight of species k and of the mixture: W_k and $\overline{W}$, respectively - $\left[\frac{g}{mol}\right] = \left[\frac{kg}{kmol}\right]$.
- Molar heat capacity at constant pressure (species k and mixture): $C_{p,k}^* = C_{p,k} \cdot W_{\text{ref}}/c_{p_{\text{ref}}}$ and $C_p^* = C_p \cdot W_{\text{ref}}/c_{p_{\text{ref}}}$, respectively.
- Specific heat capacity at constant pressure (species k and mixture): $c_{p,k}^* = c_{p,k}/c_{p_{\text{ref}}}$ and $c_p^* = c_p/c_{p_{\text{ref}}}$, respectively.
- Molar enthalpy (species k and mixture): $H_k^* = H_k \cdot W_{\text{ref}}/h_{\text{ref}}$ and $H^* = H \cdot W_{\text{ref}}/h_{\text{ref}}$, respectively.
- Specific enthalpy (species k and mixture): $h_k^* = h_k/h_{\text{ref}}$ and $h^* = h/h_{\text{ref}}$, respectively.
- Mass reaction rate of species k: $\dot{\omega}_k^* = \dot{\omega}_k \cdot tim_{\text{ref}}/\rho_{\text{ref}}$
- Molar reaction rate of species k: $\dot{\omega}_k^* = \dot{\omega}_k \cdot W_{\text{ref}}\, tim_{\text{ref}}/\rho_{\text{ref}}$.

References

[1] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh. VODE: A Variable Coefficient ODE Solver. *SIAM J. Sci. Stat. Comput.*, 10:1038–1051, 1989.

[2] S. D. Cohen and A. C. Hindmarsh. CVODE, a Stiff/Nonstiff ODE Solver in C. *Comput. Phys.*, 10(2):138–143, 1996.

[3] H.J. Curran, P. Gaffuri, W.J. Pitz, and C.K. Westbrook. A comprehensive modeling study of iso-octane oxidation. *Combust. Flame*, 129:253–280, 2002.

[4] H.J. Curran, W.J. Pitz, C.K. Westbrook, C.V. Callahan, and F.L. Dryer. Oxidation of automotive primary reference fuels at elevated pressures. *Proc. Comb. Inst.*, 27:379–387, 1998.

[5] W. J. Pitz M. Mehl, H. J. Curran and C. K. Westbrook. Chemical kinetic modeling of component mixtures relevant to gasoline. European Combustion Meeting, 2009.

[6] B. J. McBride, S. Gordon, and M. A. Reno. Coefficients for Calculating Thermodynamic and Transport Properties of Individual Species. Technical Report NASA TM-4513, NASA, 1993.

[7] G.P. Smith, D.M. Golden, M. Frenklach, N.W. Moriarty, B. Eiteneer, M. Goldenberg, C.T. Bowman, R.K. Hanson, S. Song, W.C. Gardiner Jr., V.V. Lissianski, and Q. Zhiwei. *www.me.berkeley.edu/gri_mech/*.

DISTRIBUTION:

1	MS	9051	Habib Najm, 8351 (electronic copy)
1	MS	9051	Bert Debusschere, 8351 (electronic copy)
1	MS	9155	Jerry McNeish, 8954 (electronic copy)
1	MS	9051	Damian Rouson, 8351 (electronic copy)
1	MS	9051	Cosmin Safta, 8954 (electronic copy)
1	MS	9151	Jim Costa, 8950 (electronic copy)
1	MS	9054	Andrew McIlroy, 8350 (electronic copy)
1	MS	0899	Technical Library, 9536 (electronic copy)

Sandia National Laboratories