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.

Approved for public release; further dissemination unlimited.
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 $\overline{\mathcal{F}}$ .......................... 22
   Analytical expressions for $\tilde{\mathcal{F}}$ ............................... 23

5 Examples .................................................................... 25
   ign-c ........................................................................ 25
   ign-cpp ................................................................. 28
   ign-f ................................................................. 31

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
7 Nomenclature

Non-dimensional values

References
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 stoichiometric 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 \(N_{\text{spec}}\) species in an open container

\[
\frac{dT}{dt} = -\frac{1}{\rho c_p} \sum_{k=1}^{N_{\text{spec}}} h_k \dot{\omega}_k \tag{1.1}
\]

\[
\frac{dY_k}{dt} = \frac{\dot{\omega}_k}{\rho} \quad k = 1 \ldots N_{\text{spec}},
\]

where \(\rho\) 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 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

\(^1\)https://computation.llnl.gov/casc/sundials
\(^2\)https://computation.llnl.gov/casc/software.html
overview of the reaction rate expressions. Chapter 4 describes the derivation and calculation of the Jacobian matrices. 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:

\[
C_p = \sum_{k=1}^{N_{\text{spec}}} X_k C_{pk}, \quad H = \sum_{k=1}^{N_{\text{spec}}} X_k H_k, \quad S = \sum_{k=1}^{N_{\text{spec}}} X_k S_k
\]

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

\[
\frac{C_{pk}}{\mathcal{R}} = a_{0,k} + T \left(a_{1,k} + T \left(a_{2,k} + T \left(a_{3,k} + a_{4,k}T \right)\right)\right),
\]

\[
\frac{H_k}{\mathcal{R}} = T \left(a_{0,k} + T \left(a_{1,k} + T \left(a_{2,k} + T \left(a_{3,k} + a_{4,k}T \right)\right)\right)\right) + a_{5,k},
\]

\[
\frac{S_k}{\mathcal{R}} = a_{0,k} \ln T + T \left(a_{1,k} + T \left(a_{2,k} + T \left(a_{3,k} + a_{4,k}T \right)\right)\right) + a_{6,k},
\]

Here \(X_k\) the mole fraction of species \(k\), and \(\mathcal{R}\) the universal gas constant. The specific thermodynamic properties in mass units are obtained by dividing the above expressions by the molecular weights:

\[
c_{pk} = \frac{C_{pk}}{W_k}, \quad h_k = \frac{H_k}{W_k}, \quad s_k = \frac{S_k}{W_k}
\]

and the mixture properties computed as

\[
c_p = \sum Y_k c_{pk}, \quad h = \sum Y_k h_k, \quad s = \sum Y_k s_k
\]

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 throughout the library,

\[
P = \rho \frac{\mathcal{R}}{\bar{W}} T = \left(\sum_{k=1}^{N_{\text{spec}}} \bar{X}_k\right) \mathcal{R} T
\]

where \(P\) is the thermodynamic pressure, \(\bar{W}\) is the molar weight of the mixture, \(T\) is the temperature, and \(\bar{X}_k\) is the molar concentration of species \(k\). \(\bar{W}\) is computed as

\[
\bar{W} = \left(\sum_{k=1}^{N_{\text{spec}}} \frac{Y_k}{W_k}\right)^{-1} = \sum_{k=1}^{N_{\text{spec}}} X_k W_k.
\]
Mass and mole fractions can be computed from each other as

\[ X_k = Y_k \frac{W}{W_k}, \quad Y_k = X_k \frac{W_k}{W} \] (2.9)

while the molar concentration is given by \( \chi_k = \rho Y_k / W_k \).

The units for these properties are given in Section 7. Functions that perform calculations described in this section are contained in files \texttt{TC.mlms.c} and \texttt{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

\[
\dot{\omega}_k = W_k \sum_{i=1}^{N_{\text{reac}}} v_{ki} q_i, \quad v_{ki} = v'_{ki} - v''_{ki},
\]  

(3.1)

where \(N_{\text{reac}}\) is the number of reactions and \(v'_{ki}\) and \(v''_{ki}\) 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 \(q_i = C_i \mathcal{R}_i\), with

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

(3.2)

and \(\mathcal{R}_i = k_{fi} \prod_{j=1}^{N_{\text{spec}}} \mathcal{X}_j v'_{ji} - k_{ri} \prod_{j=1}^{N_{\text{spec}}} \mathcal{X}_j v''_{ji}\). The above expressions are detailed below:

- Forward rate constant has an Arrhenius expression, \(k_{fi} = A_i T^{\beta_i} \exp \left( - \frac{E_{ai}}{R T} \right)\), where \(A_i\), \(\beta_i\), 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 \(k_{fi}\). If the reverse Arrhenius parameters are not specified, \(k_{ri}\) is computed as \(k_{ri} = k_{fi} / K_{ci}\), where

\[
K_{ci} = \left( \frac{p_{atm}}{\mathcal{R} T} \right) \sum_{k=1}^{N_{\text{spec}}} v_{ki} K_{pi} = \left( \frac{p_{atm}}{\mathcal{R} T} \right) \sum_{k=1}^{N_{\text{spec}}} v_{ki} \exp \left( \sum_{k=1}^{N_{\text{spec}}} v_{ki} \left( \frac{s_k}{R_k} - \frac{h_k}{R_k T} \right) \right) 
\]  

(3.3)

\[
= \left( \frac{p_{atm}}{\mathcal{R}} \right) \sum_{k=1}^{N_{\text{spec}}} v_{ki} \exp \left( \sum_{k=1}^{N_{\text{spec}}} v_{ki} \left( - \ln T + \frac{s_k}{R_k} - \frac{h_k}{R_k T} \right) \right) 
\]  

(3.4)

\[
= \left( \frac{p_{atm}}{\mathcal{R}} \right) \sum_{k=1}^{N_{\text{spec}}} v_{ki} \exp \left( \sum_{k=1}^{N_{\text{spec}}} v_{ki} s_k \right). 
\]  

(3.5)
Based on the polynomial expressions in (2.3) and (2.4), \( g_k \)'s are computed as
\[
g_k = -\ln T + \frac{s_k}{R_k} - \frac{h_k}{R_k T} = a_{6,k} - a_{0,k} + (a_{0,k} - 1) \ln T + T \left( \frac{a_{1,k}}{2} + T \left( \frac{a_{2,k}}{6} + T \left( \frac{a_{3,k}}{12} + \frac{a_{4,k}}{20} T \right) \right) \right) - \frac{a_{5,k}}{T} \]  

(3.6)

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

(3.7)

Note: If a reaction is irreversible, \( k_r = 0 \).

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

- Reduced pressure \( P_r \). If expression “(+M)” is used to describe a reaction, then \( P_r = \frac{k_0}{k_{\infty}} \mathcal{X}_i \). For reactions that contain expressions like “(+\( \Upsilon \_m \)” (\( \Upsilon \_m \) is the name of species \( m \)), \( P_r = \frac{k_0}{k_{\infty}} \mathcal{X}_m \).

For unimolecular/recombination fall-off reactions the Arrhenius parameters for the high-pressure limit rate constant \( k_{\infty} \) 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 \( k_{\infty} \) are given on the auxiliary reaction line that contains the keyword “HIGH”.

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

(3.8)

1. For the Troe form, \( F_{\text{cent}}, A, \) and \( B \) are
\[
F_{\text{cent}} = (1 - a) \exp \left( -\frac{T}{T^{***}} \right) + a \exp \left( -\frac{T}{T^*} \right) + \exp \left( -\frac{T^{**}}{T} \right) 
\]

(3.9)

\[
A = \log_{10} P_r - 0.67 \log_{10} F_{\text{cent}} - 0.4 \\
B = 0.806 - 1.1762 \log_{10} F_{\text{cent}} - 0.14 \log_{10} P_r
\]

(3.10)

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

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

\[ \bar{\chi}_k = \rho \frac{Y_k}{W_k} \left[ \frac{kmol}{m^3} \right] \] (3.12)

The molar concentrations are then multiplied by \(10^{-3}\) to convert them to \(\frac{mol}{cm^3}\). The molar reaction rates computed based on these values are in \(\frac{mol}{cm^3s}\). These molar reaction rates are multiplied by \(10^3\) and by the molecular weight of each species \(\frac{kg}{kmol}\) to obtain the mass reaction rates in \(\frac{kg}{m^3s}\).
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 equation being replaced by the mass conservation equation (1.2).

Let
\[
\Phi = \{P, T, Y_1, Y_2, \ldots, Y_{N_{\text{spec}}} \}^T \text{ and } \tilde{\Phi} = \{P, T, Y_1, \ldots, Y_{L-1}, Y_{L+1}, \ldots, Y_{N_{\text{spec}}} \}^T
\]

(4.1) (4.2)
denote the set of variables for the full and reduced systems of ODE’s, respectively. Although 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

\[
\frac{\partial \Phi}{\partial t} = \bar{f} = \{S_P, S_T, S_{Y_1}, S_{Y_2}, \ldots, S_{Y_{N_{\text{spec}}}} \}^T \text{ and } \frac{\partial \tilde{\Phi}}{\partial t} = \tilde{f} = \{S_P, S_T, S_{Y_1}, \ldots, S_{Y_{L-1}}, S_{Y_{L+1}}, \ldots, S_{Y_{N_{\text{spec}}}} \}^T,
\]

(4.3) (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 \(\bar{f}\) and \(\tilde{f}\) be the Jacobian matrices corresponding to \(\bar{f}\) and \(\tilde{f}\), respectively. In order to facilitate the derivation of partial derivatives \(\partial \bar{f}_i / \partial \Phi_j\) and \(\partial \tilde{f}_i / \partial \tilde{\Phi}_j\) that are the elements of these matrices, it is useful to add density to the state vector. Let \(\Phi\) be the extended state vector:

\[
\Phi = \{\rho, P, T, Y_1, Y_2, \ldots, Y_{N_{\text{spec}}} \}^T,
\]

(4.5)
The rhs term corresponding to the time advance of \(\Phi\) is

\[
f(\Phi) = \left\{ S_P, S_P, -\frac{1}{\rho c_p \sum_{k=1}^{N_{\text{spec}}} h_k \dot{\omega}_k}, \frac{\dot{\omega}_1}{\rho}, \frac{\dot{\omega}_2}{\rho}, \ldots, \frac{\dot{\omega}_{N_{\text{spec}}}}{\rho} \right\}^T,
\]

(4.6)
with \(S_P = -\bar{W} \sum_{k=1}^{N_{\text{spec}}} \frac{\dot{\omega}_k}{W_k} + \frac{1}{c_p T} \sum_{k=1}^{N_{\text{spec}}} h_k \dot{\omega}_k\).
Let $\mathcal{F}$ be the Jacobian matrix corresponding to $f$. Chain-rule differentiation leads to

$$\frac{\partial \mathcal{F}_u}{\partial v} = \frac{\partial f_u}{\partial v} + \frac{\partial f_u}{\partial \rho} \frac{\partial \rho}{\partial v}$$

(4.7)

$$\frac{\partial \tilde{\mathcal{F}}_u}{\partial v} = \frac{\partial f_u}{\partial v} + \frac{\partial f_u}{\partial \rho} \frac{\partial \rho}{\partial v} + \frac{\partial f_u}{\partial Y_L} \frac{\partial Y_L}{\partial v}$$

(4.8)

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

$$\tilde{f}_u (\tilde{\Phi}) = f_u (\Phi)$$

(4.9)

In terms of actual components

$$\tilde{J}_{i,j} = \mathcal{F}_{i+1,j+1} + \mathcal{F}_{i+1,1} \frac{\partial \rho}{\partial \Phi_j} \quad i, j = 1, 2, \ldots, N_{\text{spec}} + 2$$

(4.10)

$$\tilde{\mathcal{J}}_{i,j} = \mathcal{J}_{i+i_s,j+j_s} + \mathcal{J}_{i+i_s,1} \frac{\partial \rho}{\partial \Phi_j} + \mathcal{J}_{i+i_s,L+3} \frac{\partial Y_L}{\partial \Phi_j} \quad i, j = 1, 2, \ldots, N_{\text{spec}} + 1.$$ 

(4.11)

where

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

(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 $N_{\text{spec}}$ 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: $f_2 = \mathcal{F}_2 = \text{const.}$

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

(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

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

(4.14)
$\chi_k$ is the molar concentration of species $k$

\[ \chi_k = \frac{\rho Y_k}{W_k}. \]  

(4.15)

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

\[ \mathcal{J}_{3,1} = \frac{\partial f_3}{\partial \rho} = \frac{1}{\rho^2 c_p} \sum h_k \omega_k - \frac{1}{\rho c_p} \sum h_k \frac{\partial \omega_k}{\partial \rho} = \frac{1}{\rho c_p} \sum h_k \left( \frac{\omega_k}{\rho} - \frac{\partial \omega_k}{\partial \rho} \right) \]  

(4.16)

\[ \mathcal{J}_{3,2} = 0 \]  

(4.17)

\[ \mathcal{J}_{3,3} = \frac{\partial f_3}{\partial T} = \frac{1}{\rho c_p^2} \sum h_k \omega_k - \frac{1}{\rho c_p} \sum c_{pk} \omega_k - \frac{1}{\rho c_p} \sum h_k \frac{\partial \omega_k}{\partial T} \]  

(4.18)

\[ \mathcal{J}_{3,3+j} = \frac{\partial f_3}{\partial \tilde{Y}_j} = \frac{1}{\rho c_p^2} c_{pj} \sum h_k \omega_k - \frac{1}{\rho c_p} \sum h_k \frac{\partial \omega_k}{\partial \tilde{Y}_j}, j = 1, 2, \ldots, N_{\text{spec}} \]  

(4.19)

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

\[ \mathcal{J}_{3+k,1} = \frac{\partial f_{3+k}}{\partial \rho} = \frac{\partial (\omega_k/\rho)}{\partial \rho} = \frac{1}{\rho} \left( \frac{\partial \omega_k}{\partial \rho} - \frac{\omega_k}{\rho} \right) \]  

(4.20)

\[ \mathcal{J}_{3+k,2} = \frac{\partial (\omega_k/\rho)}{\partial T} = 0 \]  

(4.21)

\[ \mathcal{J}_{3+k,3} = \frac{\partial (\omega_k/\rho)}{\partial \tilde{Y}_j} = \frac{1}{\rho} \frac{\partial \omega_k}{\partial \tilde{Y}_j} \]  

(4.22)

\[ \mathcal{J}_{3+k,3+j} = \frac{\partial (\omega_k/\rho)}{\partial \tilde{Y}_j} = \frac{1}{\rho} \frac{\partial \omega_k}{\partial \tilde{Y}_j}, j, k = 1, 2, \ldots, N_{\text{spec}} \]  

(4.23)

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

\[ \frac{\partial c_{pk}}{\partial T} = R_k \left( a_{1,k} + T \left( 2a_{2,k} + T \left( 3a_{3,k} + 4a_{4,k} T \right) \right) \right), \]  

(4.24)

\[ \frac{\partial c_p}{\partial T} = \sum Y_k \frac{\partial c_{pk}}{\partial T}, \quad \frac{\partial c_p}{\partial \tilde{Y}_j} = c_{pj} \]  

(4.25)

The partial derivatives of the species reaction rates, $\omega_k(T, \chi_1, \chi_2, \ldots)$, with respect to various independent variables are computed as

\[ \frac{\partial \omega_k}{\partial \rho} \bigg|_{T, \chi_i} = \sum_{l=1}^{N_{\text{spec}}} \frac{\partial \omega_k}{\partial \chi_l} \frac{\partial \chi_l}{\partial \rho} + \frac{\partial \omega_k}{\partial T} \frac{\partial T}{\partial \rho} + \frac{\partial \omega_k}{\partial \rho} \frac{\partial \rho}{\partial \rho} = \sum_{l=1}^{N_{\text{spec}}} Y_i \frac{\partial \omega_k}{\partial \chi_l} \]  

(4.26)

\[ \frac{\partial \omega_k}{\partial \tilde{Y}_j} \bigg|_{\rho, T, \chi_i \neq j} = \sum_{l=1}^{N_{\text{spec}}} \frac{\partial \omega_k}{\partial \chi_l} \frac{\partial \chi_l}{\partial \tilde{Y}_j} + \frac{\partial \omega_k}{\partial T} \frac{\partial T}{\partial \tilde{Y}_j} + \frac{\partial \omega_k}{\partial \rho} \frac{\partial \rho}{\partial \tilde{Y}_j} = \frac{\rho}{W_j} \frac{\partial \omega_k}{\partial \chi_j} \]  

(4.27)

(4.28)

The steps for the calculation of $\frac{\partial \omega_k}{\partial \rho}$ and $\frac{\partial \omega_k}{\partial \tilde{Y}_j}$ are itemized below
Derivatives of reaction rate

\[ \dot{\omega}_k = W_k \sum_{i=1}^{N_{\text{reac}}} v_{ki} q_i = \frac{\partial \omega_k}{\partial T} W_k \sum_{i=1}^{N_{\text{reac}}} v_{ki} \frac{\partial q_i}{\partial T}, \quad \dot{\omega}_k = W_k \sum_{i=1}^{N_{\text{reac}}} v_{ki} \frac{\partial q_i}{\partial X_i} \]

Derivatives of rate-of-progress variables

\[ q_i = C_i \mathcal{R}_i \Rightarrow \frac{\partial q_i}{\partial T} = \frac{\partial C_i}{\partial T} \mathcal{R}_i + C_i \frac{\partial \mathcal{R}_i}{\partial T}, \quad \frac{\partial q_i}{\partial T} = \frac{\partial C_i}{\partial X_i} \mathcal{R}_i + C_i \frac{\partial \mathcal{R}_i}{\partial X_i} \]

Derivatives of \( C_i \)

1. Basic reactions \( C_i = 1 \): \( \frac{\partial C_i}{\partial T} = 0, \frac{\partial C_i}{\partial X_i} = 0 \)

2. 3\textsuperscript{rd} body-enhanced reactions \( C_i = X_i \): \( \frac{\partial C_i}{\partial T} = 0, \frac{\partial C_i}{\partial X_i} = \alpha_{il} \)

3. Unimolecular/recombination fall-off reactions \( C_i = \frac{P_{ri}}{1 + P_{ri} F_i} \)

(a) \( P_r = \frac{k_{0i}}{k_{ai}} X_m \Rightarrow \frac{\partial P_{ri}}{\partial T} = \frac{k_{0i} k_{ai}}{k_{2i}^2} X_i, \quad \frac{\partial P_{ri}}{\partial X_i} = \frac{k_{0i}}{k_{ai}} \alpha_{il} \)

(b) \( P_{ri} = \frac{k_{0i}}{k_{2i}} X_m \Rightarrow \frac{\partial P_{ri}}{\partial T} = \frac{k_{0i}'}{k_{2i}} X_m, \quad \frac{\partial P_{ri}}{\partial X_i} = \frac{k_{0i}'}{k_{2i}} \delta_{im} \), where \( \delta_{im} \) is Kroe-neckler delta symbol.

(c) For Lindemann form \( F_i = 1 \Rightarrow \frac{\partial F_i}{\partial T} = 0 \)

(d) For Troe form

\[ \frac{\partial F}{\partial T} = \frac{\partial F}{\partial F_{\text{cent}}} \frac{\partial F_{\text{cent}}}{\partial T} + \frac{\partial F}{\partial P_r} \frac{\partial P_r}{\partial T} \]

\[ \frac{\partial F}{\partial X_i} = \frac{\partial F}{\partial F_{\text{cent}}} \frac{\partial F_{\text{cent}}}{\partial X_i} + \frac{\partial F}{\partial P_r} \frac{\partial P_r}{\partial X_i} = \frac{\partial F_{\text{cent}}}{\partial P_r} \frac{\partial P_r}{\partial X_i} \]

\[ \frac{\partial F}{\partial F_{\text{cent}}} = \frac{F}{F_{\text{cent}} \left( 1 + (\frac{A}{B})^2 \right)} - F \ln F_{\text{cent}} \left( \frac{2A}{B^3} \right) \frac{A_F B - B_F A}{1 + \left( \frac{A}{B} \right)^2} \]

\[ \frac{\partial F}{\partial P_r} = F \ln F_{\text{cent}} \left( \frac{2A}{B^3} \right) \frac{A_F B - B_F A}{1 + \left( \frac{A}{B} \right)^2} \]

where

\[ A_F = \frac{\partial A}{\partial F_{\text{cent}}} = -0.67 \frac{F_{\text{cent}}}{\ln 10}, \quad B_F = \frac{\partial B}{\partial F_{\text{cent}}} = -1.1762 \frac{F_{\text{cent}}}{\ln 10} \]

\[ A_{Pr} = \frac{\partial A}{\partial P_r} = \frac{1}{P_r \ln 10}, \quad B_{Pr} = \frac{\partial B}{\partial P_r} = -0.14 \frac{P_r}{P_r \ln 10} \]

\[ \frac{\partial F_{\text{cent}}}{\partial T} = -0.02 \frac{T}{T^3} \exp \left( -\frac{T}{T^*} \right) - \frac{a}{T^*} \exp \left( -\frac{T}{T^*} \right) + \frac{T^*}{T^2} \exp \left( -\frac{T^*}{T} \right) \]
(e) For SRI form

\[
\frac{\partial F}{\partial T} = F \left( \frac{e}{T} + \frac{\partial X}{\partial P_r} \ln \left( a \exp \left( -\frac{b}{T} \right) + \exp \left( -\frac{T}{c} \right) \right) \right) \\
+ X \frac{ab}{T^2} \left( \exp \left( -\frac{b}{T} \right) - \frac{1}{e} \exp \left( -\frac{T}{c} \right) \right) \\
\frac{\partial F}{\partial X} = F \ln \left( a \exp \left( -\frac{b}{T} \right) + \exp \left( -\frac{T}{c} \right) \right) \frac{\partial X}{\partial P_r} \\
\frac{\partial X}{\partial P_r} = -X^2 \frac{2 \log_{10} P_r}{P_r \ln 10}
\]

4. Chemically activated bimolecular reactions: \( C_i = \frac{1}{1 + P_i} F_i \)

\[
\frac{\partial C_i}{\partial T} = -\frac{1}{(1 + P_i)^2} \frac{\partial P_i}{\partial T} F_i + \frac{1}{1 + P_i} \frac{\partial F_i}{\partial T} \\
\frac{\partial C_i}{\partial X_j} = -\frac{1}{(1 + P_i)^2} \frac{\partial P_i}{\partial X_j} F_i + \frac{1}{1 + P_i} \frac{\partial F_i}{\partial X_j}
\]

Partial derivatives of \( P_i \) and \( F_i \) are computed similar to the ones above.

- Derivatives of \( \mathcal{R}_i \)

\[
\frac{\partial \mathcal{R}_i}{\partial T} = k_i' \prod_{j=1}^{N_{\text{spec}}} \mathcal{X}_j^{v_i'} - k_i \prod_{j=1}^{N_{\text{spec}}} \mathcal{X}_j^{v_i} \\
\frac{\partial \mathcal{R}_i}{\partial X_j} = k_i \mathcal{X}_i^{v_i} \prod_{j=1}^{N_{\text{spec}}} \mathcal{X}_j^{v_i} \frac{1}{\mathcal{X}_i} - k_i \mathcal{X}_i^{v_i} \prod_{j=1}^{N_{\text{spec}}} \mathcal{X}_j^{v_i} \frac{1}{\mathcal{X}_i}
\]

1. \( k_i = A_i T^{\beta_i} \exp \left( -\frac{E_{ai}}{R T} \right) = A_i \exp \left( \beta_i \ln T - \frac{T_a}{T} \right) \), where \( T_a = E_{ai}/R \). The derivative with respect to temperature can be calculated as \( k_i' = \frac{k_i}{k_i} \left( \beta_i + \frac{T_a}{T} \right) \)

2. If reverse Arrhenius parameters are provided, \( k_i' \) is computed similar to above. If \( k_i \) is computed based on \( k_f \) and the equilibrium constant \( K_{ci} \), then its derivative is

\[
k_i = \frac{k_{fi}}{K_{ci}} \Rightarrow k_i' = \frac{k_{fi}' K_{ci} - k_{fi} K_{ci}'}{K_{ci}^2} = \frac{k_{fi}'}{K_{ci}^2} \left( \beta_i + \frac{T_a}{T} \right) - \frac{k_{fi} K_{ci}'}{K_{ci}^2 K_{ci}} \\
= k_i \left( \frac{1}{T} \left( \beta_i + \frac{T_a}{T} \right) - \frac{K_{ci}'}{K_{ci}^2} \right)
\]

Since \( K_{ci} = \left( \frac{P_{\text{atm}}}{R} \right) \sum_{k=1}^{N_{\text{spec}}} v_{ki} \exp \left( \sum_{k=1}^{N_{\text{spec}}} v_{ki} g_k \right) = \frac{K_{ci}'}{K_{ci}} = \sum_{k=1}^{N_{\text{spec}}} v_{ki} g_k' \). It follows that

\[
k_i' = k_i \left( \frac{1}{T} \left( \beta_i + \frac{T_a}{T} \right) - \sum_{k=1}^{N_{\text{spec}}} v_{ki} g_k' \right)
\]
Analytical expressions for \( \bar{f} \)

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

Efficient evaluation of the \( \mathcal{F} \) terms

- Step 1:
  \[
  \mathcal{F}_{3+i,2} \equiv 0
  \]
  \[
  \mathcal{F}_{3+i,3} = \frac{1}{\rho} \frac{\partial \mathcal{F}_{3+i,2}}{\partial T} = \frac{W_i}{\rho} \left[ \sum_{j=1}^{N_{\text{spec}}} Y_j \frac{\partial C_j}{\partial T} \left( \mathcal{R}_{f_j} - \mathcal{R}_{r_j} \right) + \sum_{j=1}^{N_{\text{spec}}} v_{ij} C_j \left( \frac{k'_j}{k_{f_j}} - \frac{k'_j}{k_{r_j}} \right) \right]
  \]
  \[
  \mathcal{F}_{3+i,3+k} = \frac{1}{\rho} \frac{\partial \mathcal{F}_{3+i,3}}{\partial Y_k} = \frac{W_i}{W_k} \left[ \sum_{j=1}^{N_{\text{spec}}} v_{ij} C_j \left( \mathcal{R}_{f_j} - \mathcal{R}_{r_j} \right) + \sum_{j=1}^{N_{\text{spec}}} v_{ij} C_j \left( \frac{\mathcal{R}_{f_j} \nu_{k_j}^{i,j} - \mathcal{R}_{r_j} \nu_{k_j}^{i,j}}{\mathcal{F}_{j}} \right) \right],
  \]
  \( k = 1, 2, \ldots, N_{\text{spec}} \)

Here \( \mathcal{R}_{f,j} \) and \( \mathcal{R}_{r,j} \) are the forward and reverse parts, respectively of \( \mathcal{R}_j \):

\[
\mathcal{R}_{f_j} = k_{f_j} \prod_{j=1}^{N_{\text{spec}}} \mathcal{X}_{j}^{i,j},
\]

\[
\mathcal{R}_{r_j} = k_{r_j} \prod_{j=1}^{N_{\text{spec}}} \mathcal{X}_{j}^{i,j}.
\]

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

\[
\mathcal{F}_{3+i,1} = \frac{1}{\rho} \left( \frac{\partial \mathcal{F}_{3+i,2}}{\partial T} - \frac{\partial \mathcal{F}_{3+i,3}}{\partial Y_k} \right) = \frac{1}{\rho} \left( -\frac{\mathcal{F}_{3+i,2}}{\partial T} + \sum_{k=1}^{N_{\text{spec}}} Y_k \frac{\partial \mathcal{F}_{3+i,3}}{\partial Y_k} \right) = \frac{1}{\rho} \left( -\frac{\mathcal{F}_{3+i,2}}{\partial T} + \sum_{k=1}^{N_{\text{spec}}} Y_k \mathcal{F}_{3+i,3+k} \right)
\]

- Step 3:

\[
\mathcal{F}_{3,1} = \frac{1}{\rho c_p} \sum_{i=1}^{N_{\text{spec}}} h_i \left( \frac{\partial \mathcal{F}_{3+i,1}}{\partial Y_k} \right) = -\frac{1}{c_p} \sum_{i=1}^{N_{\text{spec}}} h_i \mathcal{F}_{3+i,1}
\]

\[
\mathcal{F}_{3,2} \equiv 0
\]

\[
\mathcal{F}_{3,3} = \frac{1}{\rho c_p} \left[ \frac{1}{c_p} \frac{\partial c_p}{\partial T} \sum_{i=1}^{N_{\text{spec}}} h_i \mathcal{F}_{3+i,2} - \sum_{i=1}^{N_{\text{spec}}} c_{p,i} \mathcal{F}_{3+i,3} \right] = \frac{1}{\rho c_p} \left[ \frac{1}{c_p} \frac{\partial c_p}{\partial T} \sum_{i=1}^{N_{\text{spec}}} h_i \mathcal{F}_{3+i,3} - \sum_{i=1}^{N_{\text{spec}}} c_{p,i} \mathcal{F}_{3+i,3} \right]
\]

Analytical expressions for \( \bar{f} \)

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

\[
\bar{f}_{1,j} = 0 \text{ for all } j
\]
• Line #2: **Temperature equation**: 
\[
\begin{align*}
\tilde{J}_{2,1} &= f_{3,2} + f_{3,1} \frac{\partial \rho}{\partial P}
\quad \text{(4.30)} \\
\tilde{J}_{2,2} &= f_{3,3} + f_{3,1} \frac{\partial \rho}{\partial T}
\quad \text{(4.31)} \\
\tilde{J}_{2,2+k} &= f_{3,3+k} + f_{3,1} \frac{\partial \rho}{\partial Y_k}
\quad \text{(4.32)}
\end{align*}
\]

• Line \(i=3, \ldots, N_{\text{spec}}+2\): **Species equations**: 
\[
\begin{align*}
\tilde{J}_{i,1} &= f_{i+1,2} + f_{i+1,1} \frac{\partial \rho}{\partial P} 
\quad \text{(4.33)} \\
\tilde{J}_{i,2} &= f_{i+1,3} + f_{i+1,1} \frac{\partial \rho}{\partial T} 
\quad \text{(4.34)} \\
\tilde{J}_{i,2+k} &= f_{i+1,3+k} + f_{i+1,1} \frac{\partial \rho}{\partial Y_k} \quad k = 1, 2, \ldots, N_{\text{spec}}
\quad \text{(4.35)}
\end{align*}
\]

For this case density is a dependent variable, calculated based on the ideal gas equation of state:
\[
\rho = \frac{P}{\mathcal{R}T \sum_{k=1}^{N_{\text{spec}}} \frac{Y_k}{W_k}}
\quad \text{(4.36)}
\]

The partial derivatives of density with respect to the independent variables are computed as
\[
\frac{\partial \rho}{\partial P} = \frac{\rho}{P}, \quad \frac{\partial \rho}{\partial T} = -\frac{\rho}{T}, \quad \frac{\partial \rho}{\partial Y_k} = -\frac{\rho W}{W_k}
\quad \text{(4.37)}
\]

**Analytical expressions for \( \tilde{J} \)**

• Line #1: **Pressure equation**. The source term for the pressure equation is assumed constant: \( f_2 = \tilde{f}_2 = \text{const} \).
\[
\tilde{J}_{1,j} \equiv 0 \quad \text{for all } j
\quad \text{(4.38)}
\]

• Line #2: **Temperature equation**:
\[
\begin{align*}
\tilde{J}_{2,1} &= f_{3,2} + f_{3,1} \frac{\partial \rho}{\partial P} + f_{3,L+3} \frac{\partial Y_L}{\partial P} 
\quad \text{(4.39)} \\
\tilde{J}_{2,2} &= f_{3,3} + f_{3,1} \frac{\partial \rho}{\partial T} + f_{3,L+3} \frac{\partial Y_L}{\partial T} 
\quad \text{(4.40)} \\
\tilde{J}_{2,2+k} &= f_{3,3+k} + f_{3,1} \frac{\partial \rho}{\partial Y_k} + f_{3,L+3} \frac{\partial Y_L}{\partial Y_k},
\quad k = 1, \ldots, L-1, L+1, \ldots, N_{\text{spec}}
\quad \text{(4.41)}
\end{align*}
\]

where
\[
k_s = \begin{cases} 
  k & \text{if } k < 3 + L \\
  k + 1 & \text{otherwise}
\end{cases}
\quad \text{(4.43)}
\]
• Line \#i=3,\ldots,N_{\text{spec}}+1: **Species equations**:

\[ \tilde{J}_{i,1} = F_{i+i_{s},1} + F_{i+i_{s},L+3} \frac{\partial \rho}{\partial P} \]
\[ \tilde{J}_{i,2} = F_{i+i_{s},3} + F_{i+i_{s},L+3} \frac{\partial \rho}{\partial T} \]
\[ \tilde{J}_{i,2+k} = F_{i+i_{s},3+k_{s}} + F_{i+i_{s},1} \frac{\partial \rho}{\partial Y_{k}} + F_{i+i_{s},L+3} \frac{\partial Y_{L}}{\partial Y_{k}}, \]
\[ k = 1,\ldots,L-1,L+1,\ldots,N_{\text{spec}} \]

where \( k_{s} \) is defined above and

\[ i_{s} = \begin{cases} 1 & i + 1 < 3 + L \\ 2 & \text{otherwise} \end{cases} \]

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

\[ \rho = \frac{P}{\mathcal{R}T \left( \frac{1}{W_{L}} + \sum_{k=1}^{N_{\text{spec}}} Y_{k} \left( \frac{1}{W_{k}} - \frac{1}{W_{L}} \right) \right)} \]
\[ Y_{L} = 1 - \sum_{k=1}^{N_{\text{spec}}} Y_{k} \]

The partial derivatives of density and \( Y_{L} \) with respect to the independent variables are computed as

\[ \frac{\partial \rho}{\partial P} = \rho \frac{P}{P}, \quad \frac{\partial \rho}{\partial T} = -\rho \frac{T}{T}, \quad \frac{\partial \rho}{\partial Y_{k}} = -\rho W \left( \frac{1}{W_{k}} - \frac{1}{W_{L}} \right) \]
\[ \frac{\partial Y_{L}}{\partial P} = \frac{\partial Y_{L}}{\partial T} = 0, \quad \frac{\partial Y_{L}}{\partial Y_{k}} = -1. \]
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 require 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 \textit{main} function of the C code is contained in \textit{ign.c}. The main function defines the parameters, then proceeds to set up the simulation: parameter setup (in \textit{setup.c}), TChem initialization, CVODE setup \textit{inittcvode.c}. The time advancement of the system of ODE’s (1.1) is implemented in \textit{doIgn.c}. The function contained in \textit{doIgnReinit.c} is a more compact version of \textit{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 \textit{output.c}. Functions providing the \textit{rhs} and the Jacobian matrix corresponding to (1.1) are implemented in \textit{rhsjac.c}.

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

\textit{ign.c}

- Function \textit{TC_initChem} is used to initialize TChem, while function \textit{TC_setThermoPres} sends the thermodynamic pressure to the library.

\begin{verbatim}
TC_initChem( mechfile, thermofile, (int) withTab, 0.2) ;
pressure *= pfac ;
TC_setThermoPres( pressure ) ;
\end{verbatim}

The parameters used in these functions are described below in \textit{setup.c}.

- Function \textit{TC_getSpos} returns the position in of a species in the list of species read by TChem from the kinetic model file.

\(^1\)https://computation.llnl.gov/casc/sundials
\(^2\)https://computation.llnl.gov/casc/software.html
for ( i = 0 ; i < specinno ; i++)
{
    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 parameters 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 × 1.01325 × 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 ;
Figure 5.1. Left Frame: variation of the ignition delay times with the equivalence ratio ($\Phi$) 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]

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

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

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 neglected 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 dolgn.c, dolgnReinit.c, and output.c are straighthforward.

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 ordinary 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 × 1.01325 × 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 initialize 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 intgevode( scal, Nspec, deltaTemp );

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

  intgevode.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,
\( T, Y_1, \ldots, Y_{N_{\text{spec}}} \), or alternatively \( T, Y_1, \ldots, Y_{N_{\text{spec}}-1} \). 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 );

29
Figure 5.3. Mass fraction profiles during the constant pressure ignition of a stoichiometric methane-air mixture at 1 atm. Methane combustion is modeled using the GRI-Mech v3.0 kinetic model (53 species, 325 reactions) [7].

- 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).

```c
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 `.in`. 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`.

```fortran
reactid = 0
...
posid = 2
ier = tcgetarhenfor(reactid,posid,acten)
...
acten = acten*1.4d0
ier = tcchgarhenfor(reactid,posid,acten)
```

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

```c
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`. 

31
Chapter 6

Library Functions

TC_chg.c

1. `int TC_chgArhenFor(int ireac, int ipos, double newval)`
   Change parameters for forward rate constants.
   - `ireac` - reaction index.
   - `ipos` - 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 ireac, int ipos)`
   Reverse changes for forward rate constants’ parameters
   - `ireac` - reaction index.
   - `ipos` - index of parameter to be changed (0) pre-exponential factor (1) temperature exponent, (2) activation energy

3. `int TC_chgArhenRev(int ireac, int ipos, double newval)`
   Change parameters for forward rate constants.
   - `ireac` - reaction index.
   - `ipos` - 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 ireac, int ipos)`
   Reverse changes for forward rate constants’ parameters
   - `ireac` - reaction index.
   - `ipos` - 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 [N/m²]

TC_mlms.c

1. `int TCDND_getMs2Cc(double *scal, int Nvars, double *concX)`
   Computes molar concentrations $X$’s based on temperature $T$ and species mass fractions $Y$’s.

$$X_k = Y_k \cdot \frac{\rho}{W_k}$$
If the non-dimensional flag is ON (using \texttt{TC\_setNonDim}) this function expects non-dimensional input and will provide non-dimensional output.

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

2. \texttt{int TC\_getMs2Cc(double *scal,int Nvars,double *concX)}
Computes molar concentrations $\chi$’s based on temperature $T$ and species mass fractions $Y$’s.

\[ \chi_k = Y_k \cdot \frac{\rho}{W_k} \]

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

3. \texttt{int TCDND\_getMl2Ms(double *Xspec,int Nspec,double *Yspec)}
Transforms mole fractions $X$’s to mass fractions $Y$’s (same as TC\_getMl2Ms()).

\[ Y_k = X_k \cdot \frac{W}{W_k} \]

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

4. \texttt{int TC\_getMl2Ms(double *Xspec,int Nspec,double *Yspec)}
Transforms mole fractions $X$’s to mass fractions $Y$’s.

\[ Y_k = X_k \cdot \frac{W}{W_k} \]

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

5. \texttt{int TCDND\_getMs2Ml(double *Yspec,int Nspec,double *Xspec)}
Transforms mass fractions $Y$’s to mole fractions $X$’s (same as TC\_getMs2Ml()).

\[ X_k = Y_k \cdot \frac{W}{W_k} \]

- \textit{Yspec} - array of $N_{\text{spec}}$ mole fractions $Y$ [].
- $N_{\text{spec}}$ - no. of species
- \textit{Xspec} - array of $N_{\text{spec}}$ mole fractions $X$ [].
6. int TC_getMs2MI(double *Yspec, int Nspec, double *Xspec)
Transforms mass fractions $Y$’s to mole fractions $X$’s.

$$X_k = Y_k \cdot \frac{W_{ref}}{W_k}$$

- $Yspec$ - array of $Nspec$ mole fractions $Y$ []
- $Nspec$ - no. of species
- $Xspec$ - array of $Nspec$ mole fractions $X$ []

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

$$W = \left( \sum_{k=1}^{Nspec} \frac{Y_k}{W_k} \right)^{-1}$$

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

8. int TC_getMs2Wmix(double *Yspec, int Nspec, double *Wmix)
Computes mixture molecular weight $W$ based on species mass fractions $Y$’s.

$$W = \left( \sum_{k=1}^{Nspec} \frac{Y_k}{W_k} \right)^{-1}$$

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

9. int TCDND_getMi2Wmix(double *Xspec, int Nspec, double *Wmix)
Computes mixture molecular weight $W$ based on species mole fractions $X$’s. If the non-dimensionnal flag is ON (using TC_setNonDim) this function expects non-dimensional input and will provide non-dimensional output.

$$W = \frac{1}{W_{ref}} \sum_{k=1}^{Nspec} X_k W_k$$

- $Xspec$ - array of $Nspec$ mole fractions $X$ []
- $Nspec$ - no. of species
- $Wmix$ - pointer to mixture molecular weight [kg/kmol]=[g/mol].
10. `int TC_getMl2Wmix(double *Xspec, int Nspec, double *Wmix)`

Computes mixture molecular weight $\bar{W}$ based on species mole fractions $X$'s.

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

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

**TC_rr.c**

1. `int TC_getNreac()`

   Returns number of reactions $N_{\text{reac}}$.

2. `int TC_getStoiCoef(int Nspec, int Nreac, 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 dimensioned to at least $N_{\text{reac}} \times N_{\text{spec}}$

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

3. `int TC_getStoiCoefReac(int Nspec, int Nreac, int ireac, int idx, double *stoicoef)`

   Returns stoichiometric coefficients’ array for reaction $i_{\text{reac}}$ for either reactants ($idx = 0$) or products ($idx = 1$). The stoichiometric coefficient for species $j$ in reaction $i_{\text{reac}}$ is stored at position $j$. It assumes that $stoicoef$ was dimensioned to at least $N_{\text{spec}}$

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

4. `int TC_getArhenFor(int ireac, int ipos, 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 $val$.

   - $i_{\text{reac}}$ - reaction index.
   - $i_{\text{pos}}$ - index of Arrhenius parameter (0) pre-exponential factor (1) temperature exponent, (2) activation energy
5. `int TC_getArhenRev(int ireac, int ipos, 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.

   - `ireac` - reaction index.
   - `ipos` - 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 Nvars, 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 $Y$.
   - `Nvars` - no. of variables = $N_{\text{spec}} + 1$
   - `omega` - array of $N_{\text{spec}}$ molar reaction rates [kmol/(m$^3$·s)]

7. `int TC_getTY2RRml(double *scal, int Nvars, 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 $Y$.
   - `Nvars` - no. of variables = $N_{\text{spec}} + 1$
   - `omega` - array of $N_{\text{spec}}$ molar reaction rates $\dot{\omega}_i$ [kmol/(m$^3$·s)].

8. `int TCDND_getTY2RRRms(double *scal, int Nvars, 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 $Y$.
   - `Nvars` - no. of variables = $N_{\text{spec}} + 1$
   - `omega` - array of $N_{\text{spec}}$ mass reaction rates [kg/(m$^3$·s)]

9. `int TC_getTY2RRRms(double *scal, int Nvars, 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 $Y$.
   - `Nvars` - no. of variables = $N_{\text{spec}} + 1`
• **omega** - array of \( N_{\text{spec}} \) mass reaction rates \([\text{kg}/(\text{m}^3 \cdot \text{s})]\).

10. `int TCDND_getTXC2RRml(double *scal, int Nvars, double *omega)`
Returns molar reaction rates based on temperature \( T \) and molar concentrations \( \mathcal{X} \)'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 \((T, \mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_N)\), temperature \( T \) [K], molar concentrations \( \mathcal{X} \) [kmol/m\(^3\)].
- `Nvars` - no. of variables = \( N_{\text{spec}} + 1 \)
- `omega` - array of \( N_{\text{spec}} \) molar reaction rates \([\text{kmol}/(\text{m}^3 \cdot \text{s})]\).

11. `int TC.getTXC2RRml(double *scal, int Nvars, double *omega)`
Returns molar reaction rates based on temperature \( T \) and molar concentrations \( \mathcal{X} \)'s.

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

12. `int TCDND_getTXC2RRms(double *scal, int Nvars, double *omega)`
Returns non-dimensional mass reaction rates based on temperature \( T \) and molar concentrations \( \mathcal{X} \)'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 \((T, \mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_N)\), temperature \( T \) [K], molar concentrations \( \mathcal{X} \) [kmol/m\(^3\)].
- `Nvars` - no. of variables = \( N_{\text{spec}} + 1 \)
- `omega` - array of \( N_{\text{spec}} \) mass reaction rates \([\text{kg}/(\text{m}^3 \cdot \text{s})]\).

13. `int TC.getTXC2RRms(double *scal, int Nvars, double *omega)`
Returns mass reaction rates based on temperature \( T \) and molar concentrations \( \mathcal{X} \)'s.

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

14. `int TC.getRops(double *scal, int Nvars, 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 \((T, Y_1, Y_2, \ldots, Y_N)\), temperature \( T \) [K], species mass fractions \( Y \) [].
- `Nvars` - no. of variables = \( N_{\text{spec}} + 1 \)
- `datarop` - array of \( N_{\text{reac}} \) rate-of-progress variables \([\text{kmol}/(\text{m}^3 \cdot \text{s})]\).
15. `int TC_getRfRb(double *scal, int Nvars, 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 $Y$[].
- `Nvars` - no. of variables = $N_{\text{spec}} + 1$
- `dataRfrb` - array of $N_{\text{reac}}$ forward rate-of-progress variables and $N_{\text{reac}}$ reverse rate-of-progress variables [kmol/(m$^3 \cdot $s)].

**TC_spec.c**

**User functions:**

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

2. `int TC_getNelem()`
   Returns number of elements.

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

4. `int TC_getSnames(int Nspec, char *snames)`
   Returns species names.
   - `Nspec` - no. of species, needs to match the library’s internal value.
   - `snames` - array of characters, the allocation needs to be at least $N_{\text{spec}} \cdot \text{LENGTHOFSPECNAME}$. Currently LENGTHOFSPECNAME is set to 32. Name of species $i$ starts at position $i \cdot \text{LENGTHOFSPECNAME}$ 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 Nspec, double *Wi)`
   Returns molar masses for all species.
   - `Nspec` - 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 $N_{\text{spec}}$. 
**TC_src.c**

1. `int TCDND_getSrc(double *scal, int Nvars, double *omega)`
   Returns the source term for
   \[
   \frac{\partial T}{\partial t} = \omega_0, \quad \frac{\partial Y_i}{\partial t} = \omega_i,
   \]
   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 array of $N_{\text{spec}} + 1$ doubles $(T, Y_1, Y_2, \ldots, Y_N)$, temperature $T$ [K], species mass fractions $Y$ [].
   - `$N_{\text{vars}}$` - 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_{\text{spec}}]` : [1/s]

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

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

3. `int TCDND_getSrcCons(double *scal, int Nvars, double *omega)`
   Returns the source term for
   \[
   \frac{\partial \rho}{\partial t} = \omega_0, \quad \rho \frac{\partial Y_i}{\partial t} = \omega_i,
   \]
   based on density $\rho$ 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 $(\rho, Y_1, Y_2, \ldots, Y_N)$, density $\rho$ [kg/m$^3$], species mass fractions $Y$ [].
   - `$N_{\text{vars}}$` - 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_{\text{spec}}]` : [kg/(m$^3$·s)]
4. int TC::getSrcCons(double *scal, int Nvars, 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 \( \rho \) and species mass fractions \( Y_i \).
   - **scal** - pointer to an array of \( N_{\text{spec}} + 1 \) doubles \( (\rho, Y_1, Y_2, ..., Y_N) \), density \( \rho \) [kg/m\(^3\)], species mass fractions \( Y_i \) [].
   - **Nvars** - 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)], \( \omega_i \) [kg/(m\(^3\)·s)]

5. int TCDND::getJacTYNm1(double *scal, int Nspec, double *jac, unsigned int useJacAnl)
   Computes Jacobian matrix for the system \( (T, Y_1, Y_2, ..., Y_{N-1}) \) based on temperature \( T \) and species mass fractions \( Y_i \)’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 \( Y_i \) [].
   - **Nspec** - no. of species \( N_{\text{spec}} \)
   - **jac** - pointer to array of doubles with the Jacobian matrix \( J \). Element \( J_{ij} \) is stored at \( jac[j \times N_{\text{spec}} + i] \).
   - **useJacAnl** - flag for Jacobian type (1-analytical, other values-numerical)

6. int TC::getJacTYNm1(double *scal, int Nspec, double *jac, unsigned int useJacAnl)
   Computes Jacobian matrix for the system \( (T, Y_1, Y_2, ..., Y_{N-1}) \) based on temperature \( T \) and species mass fractions \( Y_i \)’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 \( Y_i \) [].
   - **Nspec** - no. of species \( N_{\text{spec}} \)
   - **jac** - pointer to array of doubles with the Jacobian matrix \( J \). Element \( J_{ij} \) is stored at \( jac[j \times N_{\text{spec}} + i] \).
   - **useJacAnl** - flag for Jacobian type (1-analytical, other values-numerical)

7. int TCDND::getJacTYNm1anl(double *scal, int Nspec, double *jac)
   Computes Jacobian matrix for the system \( (T, Y_1, Y_2, ..., Y_{N-1}) \) based on temperature \( T \) and species mass fractions \( Y_i \)’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 \( Y_i \) [].
8. **`int TC.getJacTYNm1anl`**(double *`scal`, int `Nspec`, 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 $(T,Y_1,Y_2,\ldots,Y_N)$, temperature $T$ [K], species mass fractions $Y$ [].
- **`Nspec`** - no. of species $N_{\text{spec}}$
- **`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 `Nspec`, 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 $(T,Y_1,Y_2,\ldots,Y_N)$, temperature $T$ [K], species mass fractions $Y$ [].
- **`Nspec`** - no. of species $N_{\text{spec}}$
- **`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 `Nspec`, 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 $(T,Y_1,Y_2,\ldots,Y_N)$, temperature $T$ [K], species mass fractions $Y$ [].
- **`Nspec`** - no. of species $N_{\text{spec}}$
- **`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 `Nspec`, 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.
12. `int TC_getJacTYNanl(double *scal, int Nspec, 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 \((T, Y_1, Y_2, \ldots, Y_N)\), temperature \(T\) [K], species mass fractions \(Y\) [].
- \(N_{\text{spec}}\) - no. of species \(N_{\text{spec}}\)
- `jac` - pointer to array of doubles with the Jacobian matrix \(J\). Element \(J_{ij}\) is stored at \(jac[j \times (N_{\text{spec}} + 1) + i]\).

13. `int TC_getJacRPTYN(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 or numerical derivatives.

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

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

15. `int TC_getJacRPTYNnum(double *scal, int Nspec, 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 \((T, Y_1, Y_2, \ldots, Y_N)\), temperature \(T\) [K], species mass fractions \(Y\) [].
- \(N_{\text{spec}}\) - no. of species \(N_{\text{spec}}\)
- `jac` - pointer to array of doubles with the Jacobian matrix \(J\). Element \(J_{ij}\) is stored at \(jac[j \times (N_{\text{spec}} + 3) + i]\).
TC_thermo.c

1. `int TCDND_getRhoMixMs(double *scal, int Nvars, 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 $Y$ [].
   - `Nvars` - no. of variables $= N_{\text{spec}} + 1$
   - `rhomix` - pointer to mixture density [kg/m$^3$]

2. `int TC_getRhoMixMs(double *scal, int Nvars, 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 $Y$ [].
   - `Nvars` - no. of variables $= N_{\text{spec}} + 1$
   - `rhomix` - pointer to mixture density [kg/m$^3$]

3. `int TCDND_getRhoMixMl(double *scal, int Nvars, 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 $X$ [].
   - `Nvars` - no. of variables $= N_{\text{spec}} + 1$
   - `rhomix` - pointer to mixture density [kg/m$^3$]

4. `int TC_getRhoMixMl(double *scal, int Nvars, 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 $X$ [].
   - `Nvars` - no. of variables $= N_{\text{spec}} + 1$
   - `rhomix` - pointer to mixture density [kg/m$^3$]

5. `int TCDND_getTMixMs(double *scal, int Nvars, double *Tmix)`
   Computes temperature based on density $\rho$ 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 ($\rho, Y_1, Y_2, \ldots, Y_N$), density $\rho$ [kg/m$^3$], species mass fractions $Y$ [].
• $N_{\text{vars}}$ - no. of variables = $N_{\text{spec}} + 1$
• Tmix - pointer to temperature [K]

6. int TC\_getTMixMs(double *scal, int $N_{\text{vars}}$, double *Tmix)
Computes temperature based on density $\rho$ and species mass fractions $Y$’s using the equation of state.

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

7. int TCDND\_getTMixMl(double *scal, int $N_{\text{vars}}$, double *Tmix)
Computes temperature based on density $\rho$ 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 ($\rho, X_1, X_2, \ldots, X_N$), density $\rho$ [kg/m$^3$], species mole fractions $X$ [].
• $N_{\text{vars}}$ - no. of variables = $N_{\text{spec}} + 1$
• Tmix - pointer to temperature [K]

8. int TC\_getTMixMl(double *scal, int $N_{\text{vars}}$, double *Tmix)
Computes temperature based on density $\rho$ and species mole fractions $X$’s using the equation of state.

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

9. int TCDND\_getMs2CpMixMs(double *scal, int $N_{\text{vars}}$, 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 ($T, Y_1, Y_2, \ldots, Y_N$), temperature $T$ [K], species mass fractions $Y$ [].
• $N_{\text{vars}}$ - no. of variables = $N_{\text{spec}} + 1$
• cpmix - pointer to mixture specific heat at constant pressure [J/(kg·K)]
10. \( \text{int TC\_getMs2CpMixMs(double *scal, int Nvars, double *cpmix)} \)

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

11. \( \text{int TCDND\_getMs2CvMixMs(double *scal, int Nvars, double *cvmix)} \)

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

12. \( \text{int TC\_getMs2CvMixMs(double *scal, int Nvars, double *cvmix)} \)

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

13. \( \text{int TCDND\_getMl2CpMixMl(double *scal, int Nvars, double *cvmix)} \)

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 \( X \) [].
- \( N_{\text{vars}} \) - no. of variables = \( N_{\text{spec}} + 1 \)
- \( cvmix \) - pointer to mixture heat capacity at constant pressure [J/(kmol·K)]

14. \( \text{int TC\_getMl2CpMixMl(double *scal, int Nvars, double *cvmix)} \)

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 \( X \) [].
- \( N_{\text{vars}} \) - 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 Nspec,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]
• $N_{\text{spec}}$ - no. of species
• $cpi$ - array with species specific heats at constant pressure [J/(kg·K)]

16. `int TC_getCpSpecMs(double t,int Nspec,double *cpi)`
Computes species specific heat at constant pressure based on temperature $T$.

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

17. `int TCDND_getCpSpecMsI(double t,int Nspec,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]
• $N_{\text{spec}}$ - no. of species
• $cpi$ - array with species heat capacities at constant pressure [J/(kmol·K)]

18. `int TC_getCpSpecMsI(double t,int Nspec,double *cpi)`
Computes species heat capacities at constant pressure based on temperature.

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

19. `int TCDND_getMs2HmixMs(double *scal,int Nvars,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 ($T, Y_1, Y_2, \ldots, Y_N$), temperature $T$ [K], species mass fractions $Y$ []
• $N_{\text{vars}}$ - no. of variables = $N_{\text{spec}} + 1$
• $hmix$ - pointer to mixture specific enthalpy [J/kg].
20. `int TC_getMs2HmixMs(double *scal, int Nvars, 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 $Y$ [].
- $N_{\text{vars}}$ - no. of variables $= N_{\text{spec}} + 1$
- `hmix` - pointer to mixture specific enthalpy [J/kg].

21. `int TCDND_getMl2HmixMl(double *scal, int Nvars, 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 $X$ [].
- $N_{\text{vars}}$ - no. of variables $= N_{\text{spec}} + 1$
- `hmix` - pointer to mixture molar enthalpy [J/kmol].

22. `int TC_getMl2HmixMl(double *scal, int Nvars, 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 $X$ [].
- $N_{\text{vars}}$ - no. of variables $= N_{\text{spec}} + 1$
- `hmix` - pointer to mixture molar enthalpy [J/kmol].

23. `int TCDND_getHspecMs(double t, int Nspec, 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]
- $N_{\text{spec}}$ - no. of species
- `hi` - array of $N_{\text{spec}}$ doubles with species specific enthalpies [J/kg]

24. `int TC_getHspecMs(double t, int Nspec, double *hi)`
Computes species specific enthalpies based on temperature $T$.

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

25. `int TCDND_getHspecMl(double t, int Nspec, 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]
- $N_{\text{spec}}$ - no. of species
- $hi$ - array of $N_{\text{spec}}$ doubles with species molar enthalpies [J/kmol]

26. `int TC_getHspecMl(double t, int N_{\text{spec}}, double *hi)`

Computes species molar enthalpies based on temperature $T$.

- $t$ - temperature $T$ [K]
- $N_{\text{spec}}$ - no. of species
- $hi$ - array of $N_{\text{spec}}$ doubles with species molar enthalpies [J/kmol]
Chapter 7

Nomenclature

- Temperature: \( T - [K] \).
- Density: \( \rho - \left[ \frac{kg}{m^3} \right] \).
- 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 \( k \): \( \bar{X}_k \), respectively - \( \left[ \frac{kmol}{m^3} \right] \).
- Molecular weight of species \( k \) and of the mixture: \( W_k \) and \( W \), respectively - \( \left[ \frac{g}{mol} \right] = \left[ \frac{kg}{kmol} \right] \).
- Universal gas constant, \( \mathcal{R} = 8.314472 \times 10^3 \left[ \frac{J}{kmol \cdot K} \right] = 1.98721 \left[ \frac{cal}{mol \cdot K} \right] \).
- 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 \( k \): \( \omega_k - \left[ \frac{kmol}{m^3 \cdot s} \right] \).
**Non-dimensional values**

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


**DISTRIBUTION:**

<table>
<thead>
<tr>
<th>MS</th>
<th>Name</th>
<th>Phone</th>
<th>Copy Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>9051</td>
<td>Habib Najm</td>
<td>8351</td>
<td>electronic copy</td>
</tr>
<tr>
<td>9051</td>
<td>Bert Debuschere</td>
<td>8351</td>
<td>electronic copy</td>
</tr>
<tr>
<td>9155</td>
<td>Jerry McNeish</td>
<td>8954</td>
<td>electronic copy</td>
</tr>
<tr>
<td>9051</td>
<td>Damian Rouson</td>
<td>8351</td>
<td>electronic copy</td>
</tr>
<tr>
<td>9051</td>
<td>Cosmin Safta</td>
<td>8954</td>
<td>electronic copy</td>
</tr>
<tr>
<td>9151</td>
<td>Jim Costa</td>
<td>8950</td>
<td>electronic copy</td>
</tr>
<tr>
<td>9054</td>
<td>Andrew McIlroy</td>
<td>8350</td>
<td>electronic copy</td>
</tr>
<tr>
<td>0899</td>
<td>Technical Library</td>
<td>9536</td>
<td>electronic copy</td>
</tr>
</tbody>
</table>