SAND2020-10762

SANDIA REPORT

SAND2020-AAAA
Printed September, 2020

Sandia
National
Laboratories

TChem v2.0 - A Software Toolkit for the Analysis of Complex Kinetic Models

Kyungjoo Kim, Oscar Diaz-Ibarra, Cosmin Safta, Habib N. Najm

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

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

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@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ACKNOWLEDGMENT

This work is supported as part of the Computational Chemical Sciences Program funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Chemical Sciences, Geosciences and Biosciences Division. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

TChem is an open source software library for solving complex computational chemistry problems and analyzing detailed chemical kinetic models. The software provides support for

- complex kinetic models for gas-phase and surface chemistry,
- thermodynamic properties based on NASA polynomials,
- species production/consumption rates,
- stable time integrator for solving stiff time ordinary differential equations,
- reactor models such as homogenous gas-phase ignition (with analytical Jacobian matrices), continuously stirred tank reactor, plug-flow reactor.

This toolkit builds upon earlier versions that were written in C and featured tools for gas-phase chemistry only. The current version of the software was completely refactored in C++, uses an object-oriented programming model, and adopts Kokkos [2, 6] as its portability layer to make it ready for the next generation computing architectures i.e., multi/many core computing platforms with GPU accelerators. We have expanded the range of kinetic models to include surface chemistry and have added examples pertaining to Continuously Stirred Tank Reactors (CSTR) and Plug Flow Reactor (PFR) models to complement the homogenous ignition examples present in the earlier versions. To exploit the massive parallelism available from modern computing platforms, the current software interface is designed to evaluate samples in parallel, which enables large scale parametric studies, e.g. for sensitivity analysis and model calibration.

1.1. Nomenclature

In the table below, *ro* stands for reaction order, for the forward and reverse paths, respectively.

Notation	**Description**	**Units**
N_{spec}	number of species	-
N_{spec}^{g}	number of gas-phase species	-
N_{spec}^{s}	number of surface species	-
$N_{spec}^{s,n}$	number of surface species in phase *n*	-
N_{reac}	number of reactions	-
ρ	gas-phase density	kg/m^3
P	thermodynamic pressure	Pa
T	temperature	K
C_p	mixture heat capacity at constant pressure	J/(K.kmol)
$C_{p,k}$	for species *k*	J/(K.kmol)
c_p	specific	J/(K.kg)

$c_{p,k}$	specific, for species *k*	J/(K.kg)
H	mixture molar enthalpy	J/kmol
H_k	for species *k*	J/kmol
h_p	specific	J/kg
$h_{p,k}$	specific, for species *k*	J/kg
S	mixture molar entropy	J/(kmol.K)
S_k	for species *k*	J/(K.kmol)
s	specific	J/(K.kg)
s_k	specific, for species *k*	J/(K.kg)
G	Gibbs free energy for the mixture	J/kmol
G_k	for species *k*	J/kmol
g	specific	J/kg
g_k	specific, for species *k*	J/kg
Y_k	mass fraction of species *k*	-
X_k	mole fraction of species *k*	-
Z_k	site fraction of species *k*	-
$Z_k^{(n)}$	for species *k* in phase *n*	-
$\mathfrak{X}_k$	molar concentration of species *k*	kmol/m^3
Γ_n	surface site density of phase *n*	kmol/m^2
$\sigma_k(n)$	site occupancy by species *k* in phase *n*	-
W	mixture molecular weight	kg/kmol
W_k	for species *k*	kg/kmol
R	universal gas constant	J/(kmol.K)
k_{fi}	forward rate constant of reaction *i*	$\frac{(\text{kmol/m}^3)^{(1-ro)}}{\text{s}}$
k_{ri}	reverse rate constant of reaction *i*	$\frac{(\text{kmol/m}^3)^{(1-ro)}}{\text{s}}$
$\dot{q}_i$	rate of progress of reaction *i*	$\text{kmol/(m}^3\text{.s)}$
γ_i	sticking coefficient for reaction *i*	$\frac{(\text{kmol/m}^3)^{(1-ro)}}{\text{s}}$
$\dot{\omega}_k$	molar production rate of species *k*	$\text{kmol/(m}^3\text{.s)}$
$\dot{s}_k$	surface molar production rate of species *k*	$\text{kmol/(m}^2\text{.s)}$

2. BUILDING TCHEM

TChem is designed and implemented using Kokkos (a performance portable parallel programming model) and it requires Kokkos and KokkosKernels. For testing, we use the GTEST infrastructure. Additionally, it can use OpenBLAS or Intel MKL (more precisely we use CBLAS and LAPACK interface from those libraries).

For convenience, we explain how to build TChem using the following set of environment variables that users can modify according to their working environments.

```
/// repositories
export TCHEM_REPOSITORY_PATH=/where/you/clone/tchem/git/repo
export KOKKOS_REPOSITORY_PATH=/where/you/clone/kokkos/git/repo
export KOKKOSKERNELS_REPOSITORY_PATH=/where/you/clone/kokkoskernels/git/repo
export GTEST_REPOSITORY_PATH=/where/you/clone/gtest/git/repo

/// build directories
export TCHEM_BUILD_PATH=/where/you/build/tchem
export KOKKOS_BUILD_PATH=/where/you/build/kokkos
export KOKKOSKERNELS_BUILD_PATH=/where/you/build/kokkoskernels
export GTEST_BUILD_PATH=/where/you/build/gtest

/// install directories
export TCHEM_INSTALL_PATH=/where/you/install/tchem
export KOKKOS_INSTALL_PATH=/where/you/install/kokkos
export KOKKOSKERNELS_INSTALL_PATH=/where/you/install/kokkoskernels
export GTEST_INSTALL_PATH=/where/you/install/gtest
export OPENBLAS_INSTALL_PATH=/where/you/install/openblas
export LAPACKE_INSTALL_PATH=/where/you/install/lapacke
```

2.1. Download Libraries

Clone Kokkos, KokkosKernels and TChem repositories. Note that we use the develop branch of Kokkos and KokkosKernels.

```
git clone https://path.to.tchem/TChem ${TCHEM_REPOSITORY_PATH}
git clone https://github.com/kokkos/kokkos.git ${KOKKOS_REPOSITORY_PATH}
cd ${KOKKOS_REPOSITORY_PATH}; git checkout -- track origin/develop
git clone https://github.com/kokkos/kokkos-kernels.git ${KOKKOSKERNELS_REPOSITORY_PATH}
cd ${KOKKOSKERNELS_REPOSITORY_PATH}; git checkout --track origin/develop
git clone https://github.com/google/googletest.git ${GTEST_REPOSITORY_PATH}
```

Here, we compile and install the TPLs separately; TChem will then be built against these TPLs.

2.2. Building Libraries and Configuring TChem

2.2.1. Kokkos

The example below builds Kokkos on Intel Sandybridge architectures and installs it to `${KOKKOS_INSTALL_PATH}`. For more details, see [2].

```
cd ${KOKKOS_BUILD_PATH}
cmake \
    -D CMAKE_INSTALL_PREFIX="${KOKKOS_INSTALL_PATH}" \
    -D CMAKE_CXX_COMPILER="${CXX}"  \
    -D Kokkos_ENABLE_SERIAL=ON \
    -D Kokkos_ENABLE_OPENMP=ON \
    -D Kokkos_ENABLE_DEPRECATED_CODE=OFF \
    -D Kokkos_ARCH_SNB=ON \
    ${KOKKOS_REPOSITORY_PATH}
make -j install
```

To compile for NVIDIA GPUs, one can customize the following cmake script. Note that, for this case, we use the `nvcc_wrapper` compiler. The architecture flag indicates that the host architecture is Intel SandyBridge and the GPU architecture is a Volta 70 generation. With Kokkos 3.1, the CUDA architecture flag is optional (the script automatically detects the correct CUDA arch flag).

```
cd ${KOKKOS_BUILD_PATH}
cmake \
    -D CMAKE_INSTALL_PREFIX="${KOKKOS_INSTALL_PATH}" \
    -D CMAKE_CXX_COMPILER="${KOKKOS_REPOSITORY_PATH}/bin/nvcc_wrapper"  \
    -D Kokkos_ENABLE_SERIAL=ON \
    -D Kokkos_ENABLE_OPENMP=ON \
    -D Kokkos_ENABLE_CUDA:BOOL=ON \
    -D Kokkos_ENABLE_CUDA_UVM:BOOL=OFF \
    -D Kokkos_ENABLE_CUDA_LAMBDA:BOOL=ON \
    -D Kokkos_ENABLE_DEPRECATED_CODE=OFF \
    -D Kokkos_ARCH_VOLTA70=ON \
    -D Kokkos_ARCH_SNB=ON \
    ${KOKKOS_REPOSITORY_PATH}
make -j install
```

2.2.2. KokkosKernels

Compiling KokkosKernels follows the Kokkos configuration information now available under `${KOKKOS_INSTALL_PATH}`. Please note that, on Max OSX, “lib64” will need to be replaced by “lib”.

```
cd ${KOKKOSKERNELS_BUILD_PATH}
cmake \
    -D CMAKE_INSTALL_PREFIX="${KOKKOSKERNELS_INSTALL_PATH}" \
    -D CMAKE_CXX_COMPILER="${CXX}"  \
    -D CMAKE_CXX_FLAGS="-g"  \
    -D KokkosKernels_INST_LAYOUTRIGHT:BOOL=ON \
    -D Kokkos_DIR="${KOKKOS_INSTALL_PATH}/lib64/cmake/Kokkos" \
    -D KokkosKernels_ENABLE_TPL_LAPACKE:BOOL=ON \
    -D KokkosKernels_ENABLE_TPL_CBLAS:BOOL=ON \
    -D CBLAS_INCLUDE_DIRS="/opt/local/include" \
    ${KOKKOSKERNELS_REPOSITORY_PATH}
make -j install
```

The `CXX` environment variable needs to be set to the C++ compiler of choice. For GPUs, the compiler is changed to `nvcc_wrapper` by adding

```
-D CMAKE_CXX_COMPILER="${KOKKOS_INSTALL_PATH}/bin/nvcc_wrapper"
```

2.2.3. GTEST

We use GTEST [1] as our testing infrastructure. GTEST can be configured and installed with the following cmake script.

```
cd ${GTEST_BUILD_PATH}
cmake \
    -D CMAKE_INSTALL_PREFIX="${GTEST_INSTALL_PATH}" \
    -D CMAKE_CXX_COMPILER="${CXX}"  \
    ${GTEST_REPOSITORY_PATH}
make -j install
```

2.2.4. TChem

The following example cmake script compiles TChem on for host computations, linking with the libraries described in the above e.g., kokkos, kokkoskernels, gtest and openblas. The openblas and lapacke libraries are required on a host device, providing an optimized version of dense linear algebra library. With an Intel compiler, one can replace these libraries with Intel MKL by adding an option `TCHEM_ENABLE_MKL=ON` instead of using openblas and lapacke. On Mac OSX, we use the openblas library managed by *macports*. This version of openblas has different header names and we need to distinguish this version of the code from others that are typically used on Linux distributions. To discern the two version of the code, cmake looks for `cblas_openblas.h` to check whether the installed version is from macports. This mechanism can be broken if macports openblas is changed later. The macports openblas version includes the lapacke interface and one can remove `LAPACKE_INSTALL_PATH` from the configure script.

```
cd ${TCHEM_BUILD_PATH}
cmake \
    -D CMAKE_INSTALL_PREFIX="${TCHEM_INSTALL_PATH}" \
    -D CMAKE_CXX_COMPILER="${CXX}" \
    -D CMAKE_BUILD_TYPE=RELEASE \
    -D TCHEM_ENABLE_VERBOSE=OFF \
    -D TCHEM_ENABLE_KOKKOS=ON \
    -D TCHEM_ENABLE_KOKKOSKERNELS=ON \
    -D TCHEM_ENABLE_TEST=ON \
    -D TCHEM_ENABLE_EXAMPLE=ON \
    -D KOKKOS_INSTALL_PATH="${KOKKOS_INSTALL_PATH}" \
    -D KOKKOSKERNELS_INSTALL_PATH="${KOKKOSKERNELS_INSTALL_PATH}" \
    -D OPENBLAS_INSTALL_PATH="${OPENBLAS_INSTALL_PATH}" \
    -D LAPACKE_INSTALL_PATH="${LAPACKE_INSTALL_PATH}" \
    -D GTEST_INSTALL_PATH="${GTEST_INSTALL_PATH}" \
    ${TCHEM_SRC_PATH}
make -j install
```

For GPUs, we can use the above cmake script and replace the compiler choice by `-D CMAKE_CXX_COMPILER="${KOKKOS_INSTALL_PATH}/bin/nvcc_wrapper"`.

3. INPUT FILES

TChem requires several input files to prescribe the modeling choices. For a gas-phase system the user provides (1) the reaction mechanisms and (2) thermal properties. Alternatively, these can be provided inside the same file with appropriate keyword selection. For the homogenous 0D ignition utility an additional file specifies the input state vectors and other modeling choices. For surface chemistry calculations, the surface chemistry model and the corresponding thermal properties can be specified in separate files or, similarly to the gas-phase chemistry case, in the same file, with appropriate keywords. Three more files are needed for the model problems with both gas and surface interface. In additional to the surface chemistry and thermodynamic properties' files, the parameters that specify the model problem are provided in a separate file.

1. Reaction Mechanism Input File

   TChem uses input files that follow the Chemkin Software syntax. A complete description can be found in Kee et al. [10]

2. Thermal Property Data

   TChem currently employs the 7-coefficient NASA polynomials. The format for the data input follows specifications in Table I of McBride et al. [11]. Support for 9-coefficient NASA polynomials is expected in the next TChem release.

3. Input State Vectors

   The format of the sample.dat file is:

   ```
   T P SPECIES_NAME1 SPECIES_NAME2 ... SPECIES_NAMEN
   T#1 P#1 Y1#1 Y2#1 ... YN#1 (sample #1)
   T#2 P#2 Y1#2 Y2#2 ... YN#2 (sample #2)
   ...
   ...
   ...
   T#M P#M Y1#M Y2#M ... YN#M (sample #M)
   ```

   Here T is the temperature [K], P is the pressure [Pa] and SPECIES_NAME1 is the name of the first gas species from the reaction mechanism input file. Y1#1 is the mass fraction of SPECIES_NAME1 in sample #1. The sum of the mass fractions on each row has to be equal to one since TChem does not normalize mass fractions. New samples can be created by adding rows to the input file. The excerpt below illustrates a setup for an example with 8 samples using a mixture of CH_4, O_2, N_2, and Ar:

   ```
   T P CH4 O2 N2 AR
    800  101325 1.48e-01 1.97e-01 6.43e-01 1.14e-02
    800  101325 2.82e-02 2.25e-01 7.34e-01 1.30e-02
    800 4559625 1.48e-01 1.97e-01 6.43e-01 1.14e-02
    800 4559625 2.82e-02 2.25e-01 7.34e-01 1.30e-02
   1250  101325 1.48e-01 1.97e-01 6.43e-01 1.14e-02
   ```

```
1250  101325 2.82e-02 2.25e-01 7.34e-01 1.30e-02
1250 4559625 1.48e-01 1.97e-01 6.43e-01 1.14e-02
1250 4559625 2.82e-02 2.25e-01 7.34e-01 1.30e-02
```

The eight samples in the above example correspond to the corners of a cube in a 3D parameter space with temperatures between 800 K and 1250 K, pressures between 1 atm to 45 atm, and equivalence ratios (ϕ) for methane/air mixtures between 0.5 to 3.

4. Surface Reaction Mechanism Input File and Thermal Property Data

   TChem uses a the specifications in Coltrin et al. [5] for the input file for the surface reaction mechanism and the corresponding thermodynamic properties for the surface species.

5. Input site fractions

   The format of the inputSurf.dat file is:

   ```
   SURF_SPECIES_NAME1 SURF_SPECIES_NAME2 ... SURF_SPECIES_NAMEN
   Z1#1 Z2#1 ... ZN#1 (sample #1)
   Z1#2 Z2#2 ... ZN#2 (sample #2)
   ...
   ...
   ...
   Z1#M Z2#M ... ZN#M (sample #M)
   ```

   where SURF_SPECIES_NAME1 is the name of the first surface species in the surface mechanism file and Z1#1 is the site fraction of this species for sample #1, and so forth.

4. THERMODYNAMIC PROPERTIES

We first present conversion formulas and the gas-phase equation of state, followed by a description of molar and mass-based expression for several thermodynamic properties.

4.1. Mass-Molar Conversions

The molar mass of the mixture, W, is computed as

$$W = \sum_{k=1}^{N_{spec}} X_k W_k = 1 \Bigg/ \left(\sum_{k=1}^{N_{spec}} \frac{Y_k}{W_k} \right) \tag{4.1.1}$$

where X_k and Y_k are the mole and mass fractions, respectively, of species k, and W_k is the molecular weight of species k. It is assumed (TChem does not check this), that mole and mass fractions sum up to one

$$\sum_{k=1}^{N_{spec}} X_k = \sum_{k=1}^{N_{spec}} Y_k = 1 \tag{4.1.2}$$

Mass and mole fractions can be computed from each other as

$$X_k = Y_k W / W_k, \; Y_k = X_k W_k / W \tag{4.1.3}$$

The the molar concentration of species k is given by $\mathfrak{X}_k = \rho Y_k / W_k = \rho X_k / W$, and the molar concentration of the mixture is given by

$$\sum_{k=1}^{N_{spec}} \mathfrak{X}_k = \rho / W \tag{4.1.4}$$

For problems that include heterogenous chemistry, the site fractions Z_k describe the composition of species on the surface. The number of surface phases is denoted by N_{phase} and the site fractions are normalized with respect to each phase.

$$\sum_{k=1}^{N_{spec}^{s,n}} Z_k^{(n)} = 1, \; \text{for} n = 1, \ldots N_{phase}. \tag{4.1.5}$$

Here, $N_{spec}^{s,n}$ is the number of species on surface phase n. TChem currently handles 1 surface phase only, $N_{phase} = 1$. The surface concentration of surface species k is given by

$$\mathfrak{X}_k = Z_k^{(n)} \Gamma_n / \sigma_k(n) \tag{4.1.6}$$

where Γ_n is the surface site density of surface phase n and $\sigma_k(n)$ is the site occupancy number for species k. $\sigma_k(n)$ represents the number of sites in phase n occupied by species k.

4.2. Equation of State

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

$$P=\rho\frac{R}{\sum_{k=1}^{N_{spec}}X_kW_k}T=\rho R\left(\sum_{k=1}^{N_{spec}}\frac{Y_k}{W_k}\right)T=\rho\frac{R}{W}T=\left(\sum_{k=1}^{N_{spec}}\mathfrak{X}_k\right)RT \tag{4.2.1}$$

where P is the thermodynamic pressure, W and W_k are the molecular weights of the mixture and of species k, respectively, T is the temperature, and $\mathfrak{X}_k$ is the molar concentration of species k.

4.3. Gas-Phase Species Properties

The standard-state thermodynamic properties for a thermally perfect gas are computed based on NASA polynomials [11]. The molar heat capacity at constant pressure for species k is computed as

$$\frac{C_{p,k}}{R}=a_{0,k}+T(a_{1,k}+T(a_{2,k}+T\left(a_{3,k}+a_{4,k}T\right))) \tag{4.3.1}$$

where R is the universal gas constant. The molar enthalpy is computed as

$$\frac{H_k}{R}=\int_{T_0}^{T}C_{p,k}dT+H_{k,T_0}=T\left(a_{0,k}+T\left(\frac{a_{1,k}}{2}+T\left(\frac{a_{2,k}}{3}+T\left(\frac{a_{3,k}}{4}+\frac{a_{4,k}}{5}T\right)\right)\right)\right)+a_{5,k} \tag{4.3.2}$$

The molar entropy is given by

$$\frac{S_k^0}{R}=\int_{T_0}^{T}\frac{C_{p,k}}{T}dT+S_{k,T_0}=a_{0,k}\ln T+T\left(a_{1,k}+T\left(\frac{a_{2,k}}{2}+T\left(\frac{a_{3,k}}{3}+\frac{a_{4,k}}{4}T\right)\right)\right)+a_{6,k} \tag{4.3.3}$$

The temperature units are Kelvin in the polynomial expressions above. Other thermodynamics properties are computed based on the polynomial fits above. The molar heat capacity at constant volume $C_{v,k}$, the internal energy U_k, and the Gibbs free energy G_k are given by

$$C_{v,k}=C_{p,k}-R,\ U_k=H_k-RT,\ G_k^0=H_k-TS_k^0 \tag{4.3.4}$$

The mixture properties in molar units are given by

$$C_p=\sum_{k=1}^{N_{spec}}X_kC_{p,k},\ C_v=\sum_{k=1}^{N_{spec}}X_kC_{v,k},\ H=\sum_{k=1}^{N_{spec}}X_kH_k,\ U=\sum_{k=1}^{N_{spec}}X_kU_k \tag{4.3.5}$$

where X_k the mole fraction of species k. The entropy and Gibbs free energy for species k account for the entropy of mixing and thermodynamic pressure

$$S_k=S_k^0-R\ln\left(X_k\frac{P}{P_{atm}}\right),\ G_k=S_k-TS_k \tag{4.3.6}$$

The mixture values for these properties are computed as above

$$S=\sum_{k=1}^{N_{spec}} X_k S_k,\ G=\sum_{k=1}^{N_{spec}} X_k G_k \tag{4.3.7}$$

The specific thermodynamic properties in mass units are obtained by dividing the above expression by the species molecular weight, W_k,

$$c_{p,k}=C_{p,k}/W_k,\ c_{v,k}=C_{v,k}/W_k,\ h_k=H_k/W_k,\ u_k=U_k/W_k,\ s_k^0=S_k^0/W_k,\ g_k^0=G_k^0/W_k \tag{4.3.8}$$

and

$$s_k=S_k/W_k,\ g_k=G_k/W_k \tag{4.3.9}$$

For the thermodynamic properties in mass units the mixture properties are given by

$$c_p=\sum_{k=1}^{N_{spec}} Y_k c_{p,k},\ c_v=\sum_{k=1}^{N_{spec}} Y_k c_{v,k},\ h=\sum_{k=1}^{N_{spec}} Y_k h_k,\ u=\sum_{k=1}^{N_{spec}} Y_k u_k,\ s=\sum_{k=1}^{N_{spec}} Y_k s_k,\ g=\sum_{k=1}^{N_{spec}} Y_k g_k \tag{4.3.10}$$

where Y_k is the mass fraction of species k.

The mixture properties in mass units can also be evaluated from the equivalent molar properties as

$$c_p=C_p/W,\ c_v=C_v/W,\ h_k=H/W,\ u=U/W,\ s=S/W,\ g=G/W \tag{4.3.11}$$

where W is the molecular weight of the mixture.

4.4. Examples

A example to compute the specific heat at constant pressure c_p and the specific enthalpy h is provided in `example/TChem_ThermalProperties.cpp`. Sections 7.1.2) and 7.1.1) present more details on the top-level function calls employed in this example. This example can be used in batch mode, and several samples are computed simultaneously. The next figure was constructed based on 4×10^5 samples spanning several temperature and equivalent ratio values for methane/air mixtures.

4.5. Surface Species Properties

The thermal properties of the surface species are computed with the same type of expressions used by the gas-phase species described above.

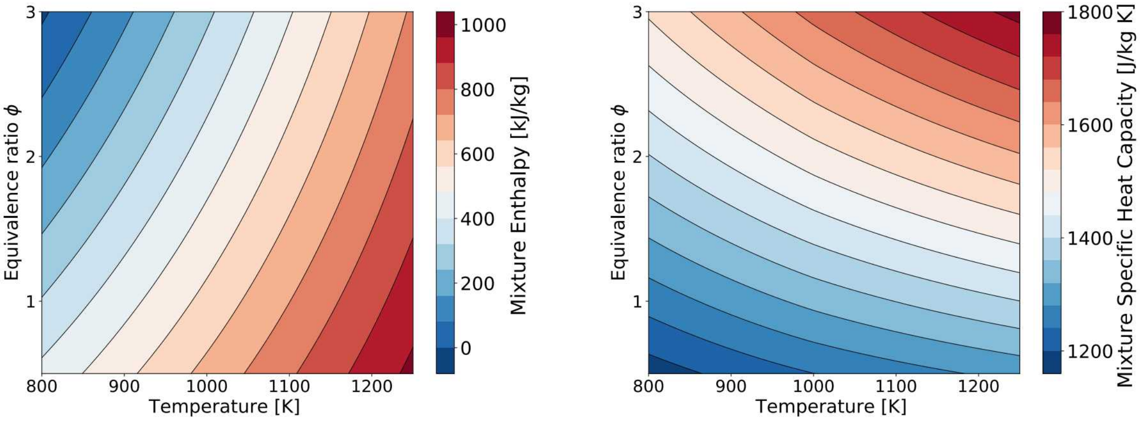

Figure 4-1. Specific enthalpy h and heat capacity c_p for a mixture of 53 species (the set of species in the GRI-Mech v3.0 model).

5. REACTION RATES

In this chapter we present reaction rate expressions for gas-phase reactions in Section 5.1 and for surface species or between surface and gas-phase species in Section 5.2.

5.1. Gas-Phase Chemistry

The production rate for species k in molar units is written as

$$\dot{\omega}_k = \sum_{i=1}^{N_{reac}} \nu_{ki} q_i, \;\; \nu_{ki} = \nu''_{ki} - \nu'_{ki}, \tag{5.1.1}$$

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 $q_i = \mathscr{C}_i \mathscr{R}_i$, with $\mathscr{C}_i$ corresponding to different reaction types shown in Table 5-1

$\mathscr{C}_i$	Reaction Type
1	basic reaction
$\mathfrak{X}_i$	3-rd body enhanced, no pressure dependence
$\frac{\mathrm{Pr}_i}{1+\mathrm{Pr}_i} F_i$	unimolecular/recombination fall-off reactions
$\frac{1}{1+\mathrm{Pr}_i} F_i$	chemically activated bimolecular reactions

Table 5-1. Gas-phase reaction types.

and $\mathscr{R}_i$ given by

$$\mathscr{R}_i = k_{f\,i} \prod_{j=1}^{N_{spec}} \mathfrak{X}_j^{\nu'_{ji}} - k_{r\,i} \prod_{j=1}^{N_{spec}} \mathfrak{X}_j^{\nu''_{ji}} \tag{5.1.2}$$

The above expressions are presented below.

5.1.1. Forward and Reverse Rate Constants

The evaluation of the forward rate constant typically follows an Arrhenius expression,

$$k_{f\,i} = A_i T^{\beta_i} \exp\left(-\frac{E_i}{RT}\right), \tag{5.1.3}$$

where A_i, β_i, and E_i are the pre-exponential factor, temperature exponent, and activation energy, respectively, for reaction i. For reactions with reverse Arrhenius parameters specified, the reverse rate constant k_{ri} is computed similar to k_{f_i}. If the reverse Arrhenius parameters are not specified, k_{ri} is computed as

$$k_{ri} = k_{f_i}/K_{ci}, \tag{5.1.4}$$

where K_{ci} is the equilibrium constant (in concentration units) for reaction i

$$K_{ci} = \left(\frac{P_{atm}}{RT}\right)^{\sum_{k=1}^{N_{spec}} \nu_{ki}} K_{p_i} \text{ and } K_{p_i} = \exp\left(\sum_{k=1}^{N_{spec}} \nu_{ki}\left(\frac{S_k}{R} - \frac{H_k}{RT}\right)\right). \tag{5.1.5}$$

When computing the equilibrium constant, the atmospheric pressure, $P_{atm} = 1\text{atm}$, and the universal gas constant R are converted to cgs units, dynes/cm^2 and erg/(mol.K), respectively.

If a reaction is irreversible, its reverse rate constant is set to zero, $k_r = 0$.

5.1.2. *Concentration of the "Third-Body"*

If the expression "+M" is present in the reaction string, some of the species might have custom efficiencies for their contribution in the mixture. For these reactions, the mixture concentration is computed as

$$\mathfrak{X}_i = \sum_{j=1}^{N_{spec}} \alpha_{ij}\mathfrak{X}_j, \tag{5.1.6}$$

where α_{ij} is the efficiency of species j in reaction i and $\mathfrak{X}_j$ is the concentration of species j. Coefficients α_{ij} are set to 1 unless specified on auxiliary lines corresponding to specific reactions.

5.1.3. *Pressure-dependent Reactions*

If expression "(+M)" is used to describe a reaction, then the reduced pressure Pr, shown in the expressions presented in Table 5-1 is computed as

$$\mathrm{Pr}_i = \frac{k_{0i}}{k_{\infty i}}\mathfrak{X}_i \tag{5.1.7}$$

where $\mathfrak{X}_i$ is the mixture concentration, possibly enhanced with specific species efficiences for reaction i.

For reactions that contain expressions like "(+Υ_m)", where Υ_m is the name of species m, the reduced pressure is computed as

$$\mathrm{Pr}_i = \frac{k_{0i}}{k_{\infty i}}\mathfrak{X}_m. \tag{5.1.8}$$

F_i	Reaction Type
1	Lindemann reaction
$F_{cent}^{1/\left(1+(A/B)^2\right)}$	Troe reaction
$dT^e\left(a\exp\left(-\frac{b}{T}\right)+\exp\left(-\frac{T}{c}\right)\right)^X$	SRI reaction

Table 5-2. Expressions for computing F_i.

- For *unimolecular/recombination fall-off reactions* the Arrhenius parameters for the high-pressure limit rate constant, k_∞, are provided 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**.

Table 5-2 shows the expressions for F_i (see also Table 5-1)

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

  $$F_{cent} = (1-a)\exp\left(-\frac{T}{T^{***}}\right) + a\exp\left(-\frac{T}{T^{*}}\right) + \exp\left(-\frac{T^{**}}{T}\right), \tag{5.1.9}$$

  $$A = \log_{10}\mathrm{Pr}_i - 0.67\log_{10}F_{cent} - 0.4,\;\; B = 0.806 - 1.1762\log_{10}F_{cent} - 0.14\log_{10}\mathrm{Pr}_i \tag{5.1.10}$$

  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_{cent}.

- For the SRI form exponent X is computed as

  $$X = \left(1 + \left(\log_{10}\mathrm{Pr}_i\right)^2\right)^{-1}. \tag{5.1.11}$$

  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 $e = 0$.

Miller [7] has developed an alternative expression for the pressure dependence for fall-off reactions that cannot be fitted with a single Arrhenius rate expression. This approach employs several Arhenius expressions

$$k_{f,l} = \sum_{j=1}^{M} A_{l,j} T^{\beta_{l,j}} \exp\left(-\frac{E_{l,j}}{RT}\right) \tag{5.1.12}$$

corresponding to a pressure p_l in a specified set $\{p_1, p_2, \dots p_N\}$. The rate constant at an arbitrary pressure p, $p_l <= p < p_{l+1}$, is computed by linear interpolation of $\log k_f$ as a function of $\log p$

$$\log k_f(T,p) = \log k_{f,l}(T) + (\log p - \log p_l)\frac{\log k_{f,l+1}(T) - \log k_{f,l}(T)}{\log p_{l+1} - \log p_l} \tag{5.1.13}$$

For $p < p_1$ the Arrhenius rate is set to $k_f = k_{f,1}$, and similar for $p > p_N$, $k_f = k_{f,N}$, where N is the number of pressures for which the Arrhenius factors are provided, for a specific reaction. This formulation can be combined with 3rd-body information, e.g. $\mathscr{C}_i = \mathfrak{X}_i$ for a specific reaction i.

5.1.4. 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 rates and the species production rates. Molar concentrations computed as illustrated in Section 4.1 are in [kmol/m^3]. For the purpose of reaction rates evaluation, the concentrations are transformed to [mol/cm^3]. The resulting reaction rates and species production rates are in [mol/(cm^3.s)]. In the last step these are converted to SI units [kg/(m^3.s)] (in molar units) and [kg/(m^3.s)] (in mass units).

5.1.5. Example

The production rate for species k in mole units $\dot{\omega}_k$ [kmol/m^3/s] is computed via the function call shown in Section 7.2.2)) and in mass units $\dot{\omega}_k W_k$ [kg/m^3/s] via the function call shown in Section 7.2.1. The example provided in `src/example/TChem_NetProductionRatesPerMass.cpp` computes the production rate in mass units for gas-phase species.

5.2. Surface Chemistry

The production rate for gas and surface species k in molar units is written as

$$\dot{s}_k = \sum_{i=1}^{N_{reac}} \nu_{ki} q_i,\ \ \nu_{ki} = \nu''_{ki} - \nu'_{ki}, \tag{5.2.1}$$

where N_{reac} is the number of reactions on the surface phase 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 q_i of the *ith* surface reaction is equal to:

$$q_i = k_{f\,i} \prod_{j=1}^{N_{spec}} \mathfrak{X}_j^{\nu'_{ji}} - k_{ri} \prod_{j=1}^{N_{spec}} \mathfrak{X}_j^{\nu''_{ji}} \tag{5.2.2}$$

Where $\mathfrak{X}_j$ is the concentration of the species j. If the species j is a gas species, this is the molar concentration, $\mathfrak{X}_j = Y_j\rho\big/W_j$. If, on the other hand, species j is a surface species, it surface molar concentration is computed by Eq. (4.1.6).

5.2.1. Forward and Reverse Rate Constants

The forward rate for surface reactions are computed as described in the gas phase. If parameters are not specified, the reverse rate is computed via the equilibrium constant as:

$$k_{r,i} = k_{f,i}\big/K_{c,i} \tag{5.2.3}$$

The equilibrium constant for the surface reaction i is computed as

$$K_{c,i} = K_{p,i}\left(\frac{p^o}{RT}\right)^{\left(\sum_{k=1}^{N^g_{spec}} \nu_{ki}\right)} \prod_{n=1}^{N_{phase}} (\Gamma_n)^{\Delta\sigma_{(}n,i)}. \tag{5.2.4}$$

Here, N^g_{spec} and N^s_{spec} represent the number of gas-phase and surface species, respectively, and $p^o = 1$atm. TChem currently assumes the surface site density Γ_n for all phases to be constant. The equilibrium constant in pressure units is computed as

$$K_{p,i} = \exp\left(\frac{\Delta S^o_i}{R} - \frac{\Delta H^o_i}{RT}\right) \tag{5.2.5}$$

based on entropy and enthalpy changes from reactants to products (including gas-phase and surface species). The net change for surface of the site occupancy number for phase n for reaction i is given by

$$\Delta\sigma_{(}n,i) = \sum_{k=1}^{N^{s,n}_{spec}} \nu_{ki}\sigma_k(n) \tag{5.2.6}$$

5.2.2. Sticking Coefficients

The reaction rate for some surface reactions are described in terms of the probability that a collision results in a reaction. For these reaction, the forward rate is computed as

$$k_{f,i} = \frac{\gamma_i}{\Gamma^{m_i}_{tot}}\sqrt{\frac{RT}{2\pi W}} \tag{5.2.7}$$

where γ_i is the sticking coefficient, W is the molecular weight of the gas-phase mixture, R is the universal gas constant, Γ_{tot} is the total surface site concentration over all phases, and m is the sum of stoichiometric coefficients for all surface species in reaction i.

5.2.3. Note on Units for surface production rates

The units of the surface and gas species concentration presented above are in units of kmol/m^2 (surface species) or kmol/m^3 (gas species). To match the units of the kinetic model and compute the rate constants, we transformed the concentration units to mol/cm^3 or mol/cm^2. The resulting rate constant has units of mol/cm^2. In the last step these are converted to SI units [kg/(m^2.s)].

5.2.4. Example

The production rate for species k in in molar units $\dot{s}_k$ [kmole/m^2/s] is computed through the function call listed in Sec. 7.2.3 and in mass units $\dot{s}_k W_k$ [kg/m^2/s] via the function call listed in Sec. 7.2.4). The example provided in
`src/example/TChem_NetProductionSurfacePerMass.cpp`
computes the production rates for gas-phase and surface species in mass units.

6. REACTORS

We present the setup for canonical examples that are available through TChem. All models presented in this section are setup to be run in parallel, possibly exploiting several layers of parallelism available on the platform of choice. We start with a description of a 2-nd order backward differentiation formula (BDF2) time stepping algorithm in Section 6.1. BDF2 was implemented via Kokkos and takes advantage of parallel threads available through the Kokkos interface. We then present results for homogenous batch reactors in Section 6.2, and the plug-flow reactor, in Section 6.3.

6.1. Time Integrator

When solving a system of stiff ordinary differential equations (ODEs), the time step size is limited by a stability condition rather than a truncation error. To obtain a reliable solution, we rely on the 2nd order Trapezoidal Backward Difference Formula (TrBDF2)[3]. The TrBDF2 scheme is a composite single step method, and is 2nd order accurate and *L*-stable.

Consider a following system of ODEs.

$$\frac{du_i}{dt} = f_i(u,t),\ i = 1,\ldots,N \tag{6.1.1}$$

As its name states, the method advances the solution from t_n to an intermediate time $t_{n+\gamma} = t_n + \gamma\Delta t$ by applying the trapezoidal rule.

$$u_{n+\gamma} - \gamma\frac{\Delta t}{2} f_{n+\gamma} = u_n + \gamma\frac{\Delta t}{2} f_n \tag{6.1.2}$$

Next, it uses BDF2 to march the solution from $t_{n+\gamma}$ to $t_{n+1} = t_n + \Delta t$ as follows.

$$u_{n+1} - \frac{1-\gamma}{2-\gamma}\Delta t f_{n+1} = \frac{1}{\gamma(2-\gamma)} u_{n+\gamma} - \frac{(1-\gamma)^2}{\gamma(2-\gamma)} u_n \tag{6.1.3}$$

We solve the above non-linear equations iteratively using the Newton method. The Newton equation of the first trapezoidal step is given by:

$$\left[I - \gamma\frac{\Delta}{2}\left(\frac{\partial f}{\partial u}\right)^{(k)}\right]\delta u^{(k)} = -(u^{(k)}_{n+\gamma} - u_n) + \gamma\frac{\Delta t}{2}(f^{(k)}_{n+\gamma} + f_n) \tag{6.1.4}$$

This step is followed by Newton iterations to solve the BDF2 step

$$\begin{aligned}\left[I-\frac{1-\gamma}{2-\gamma}\Delta t\left(\frac{\partial f}{\partial u}\right)^{(k)}\right]\delta u^{(k)} = \\ -\left(u_{n+1}^{(k)}-\frac{1}{\gamma(2-\gamma)}u_{n+\gamma}+\frac{(1-\gamma)^2}{\gamma(2-\gamma)}u_n\right)+\frac{1-\gamma}{2-\gamma}\Delta t f_{n+1}^{(k)}\end{aligned} \tag{6.1.5}$$

Here, we denote a Jacobian as $J=\partial f/\partial u$. The modified Jacobian's used for solving the Newton equations of the above trapezoidal rule and the BDF2 are given as follows

$$\begin{aligned}A_{tr} &= I-\gamma\frac{\Delta t}{2}J \\ A_{bdf} &= I-\frac{1-\gamma}{2-\gamma}\Delta t J\end{aligned} \tag{6.1.6}$$

while their right hand sides are defined as

$$\begin{aligned}b_{tr} &= -(u_{n+\gamma}^{(k)}-u_n)+\gamma\frac{\Delta t}{2}\left(f_{n+\gamma}^{(k)}+f_n\right) \\ b_{bdf} &= -\left(u_{n+1}^{(k)}-\frac{1}{\gamma(2-\gamma)}u_{n+\gamma}+\frac{(1-\gamma)^2}{\gamma(2-\gamma)}u_n\right)+\frac{1-\gamma}{2-\gamma}\Delta t f_{n+1}^{(k)}\end{aligned} \tag{6.1.7}$$

In this way, a Newton solver can iteratively solves a problem $A(u)\delta u = b(u)$ followed by the solution update $u = u+\delta u$.

The timestep size Δt can be adapted within a range $(\Delta t_{min}, \Delta t_{max})$ using a local error estimator.

$$\text{error} \approx 2k_\gamma \Delta t\left(\frac{1}{\gamma}f_n-\frac{1}{\gamma(1-\gamma)}f_{n+\gamma}+\frac{1}{1-\gamma}f_{n+1}\right) \quad \text{where} \quad k_\gamma = \frac{-3\gamma^2+4\gamma-2}{12(2-\gamma)} \tag{6.1.8}$$

This error is minimized when using a $\gamma = 2-\sqrt{2}$.

6.1.1. *Timestep Adaptivity*

TChem uses weighted root-mean-square (WRMS) norms to estimate the time stepping error. The same approach is used in Sundial package[8]. A weighting factor is computed as

$$w_i = 1/\left(\text{rtol}_i|u_i|+\text{atol}_i\right) \tag{6.1.9}$$

and the normalized error norm is computed as follows

$$\text{norm} = \left(\sum_i^m (\text{error}_i * w_i)^2\right)/m \tag{6.1.10}$$

where error$_i$ is given by Eq. (6.1.8). An error norm value close to 1 is considered small and the time step size is increased. If the error norm is bigger than 10, the time step size decreases by half.

6.1.2. Interface to Time Integrator

Our time integrator advances each sample in time independently in a parallel for. A namespace "Impl" is used to define a code interface for an individual sample.

```
TChem::Impl::TimeIntegrator::team_invoke_detail(
  /// kokkos team thread communicator
  const MemberType& member,
  /// abstract problem generator computing J_{prob} and f
  const ProblemType& problem,
  /// control parameters
  const ordinal_type& max_num_newton_iterations,
  const ordinal_type& max_num_time_iterations,
  /// absolute and relative tolerence size 2 array
  const RealType1DViewType& tol_newton,
  /// a vector of absolute and relative tolerence size Nspec x 2
  const RealType2DViewType& tol_time,
  /// \Delta t input, min, max
  const real_type& dt_in,
  const real_type& dt_min,
  const real_type& dt_max,
  /// time begin and end
  const real_type& t_beg,
  const real_type& t_end,
  /// input state vector at time begin
  const RealType1DViewType& vals,
  /// output for a restarting purpose: time, delta t, state vector
  const RealType0DViewType& t_out,
  const RealType0DViewType& dt_out,
  const RealType1DViewType& vals_out,
  const WorkViewType& work) {
  /// A pseudo code is illustrated here to describe the workflow

  /// This object is used to estimate the local errors
  TrBDF2<problem_type> trbdf2(problem);
  /// A_{tr} and b_{tr} are computed using the problem provided J_{prob} and f
  TrBDF2_Part1<problem_type> trbdf2_part1(problem);
  /// A_{bdf} and b_{bdf} are computed using the problem provided J_{prob} and f
  TrBDF2_Part2<problem_type> trbdf2_part2(problem);

  for (ordinal_type iter=0;iter<max_num_time_iterations && dt != zero;++iter) {
    /// evaluate function f_n
    problem.computeFunction(member, u_n, f_n);

    /// trbdf_part1 provides A_{tr} and b_{tr} solving A_{tr} du = b_{tr}
    /// and update u_gamma += du iteratively until it converges
    TChem::Impl::NewtonSolver(member, trbdf_part1, u_gamma, du);

    /// evaluate function f_gamma
    problem.computeFunction(member, u_gamma, f_gamma);

    /// trbdf_part2 provides A_{bdf} and b_{bdf} solving A_{bdf} du = b_{bdf}
    /// and update u_np += du iteratively until it converges
    TChem::Impl::NewtonSolver(member, trbdf_part2, u_np, du);

    /// evaluate function f_np
    problem.computeFunction(member, u_np, f_np);

    /// adjust time step
    trbdf2.computeTimeStepSize(member,
      dt_min, dt_max, tol_time, f_n, f_gamma, f_np, /// input for error evaluation
      dt); /// output

    /// account for the time end
    dt = ((t + dt) > t_end) ? t_end - t : dt;
  }
```

```
  /// store current time step and state vectors for a restarting purpose
```

This “TimeIntegrator” code requires the user to define a problem object. A problem class should include the following interface in order to be used with the time integrator.

```
template<typename KineticModelConstDataType>
struct MyProblem {
  ordinal_type getNumberOfTimeODEs();
  ordinal_type getNumberOfConstraints();
  /// the number of equations should be sum of number of time ODEs and number of constraints
  ordinal_type getNumberOfEquations();

  /// temporal workspace necessary for this problem class
  ordinal_type getWorkSpaceSize();

  /// x is initialized in the first Newton iteration
  void computeInitValues(const MemberType& member,
                         const RealType1DViewType& x) const;

  /// compute f(x)
  void computeFunction(const MemberType& member,
                       const RealType1DViewType& x,
                       const RealType1DViewType& f) const;

  /// compute J_{prob} at x
  void computeJacobian(const MemberType& member,
                       const RealType1DViewType& x,
                       const RealType2DViewType& J) const;
};
```

6.2. Homogenous Batch Reactors

In this example we consider a transient zero-dimensional constant-pressure problem where temperature T and species mass fractions for N_{spec} gas-phase species are resolved in a batch reactor. In this problem an initial condition is set and a time integration solver will evolve the solution until a time provided by the user.

For an open batch reactor the system of ODEs solved by TChem are given by:

Energy equation

$$\frac{dT}{dt} = -\frac{1}{\rho c_p}\sum_{k=1}^{N_{spec}} \dot{\omega}_k W_k h_k = S_T$$

Species equation

$$\frac{dY_k}{dt} = \frac{1}{\rho}\dot{\omega}_k W_k = S_{Y_k},\ k = 1 \ldots N_{spec}$$

where ρ is the density, c_p is the specific heat at constant pressure for the mixture, $\dot{w}_k$ is the molar production rate of species k, W_k is its molecular weight, and h_k is the specific enthalpy.

6.2.1. Jacobian Formulation

Efficient integration and accurate analysis of the stiff system of ODEs shown above requires the Jacobian matrix of the *rhs* vector. In this section we will derive the Jacobian matrix components.

Let

$$\Phi = \left\{T, Y_1, Y_2, \ldots, Y_{N_{spec}}\right\}^T$$

by the denote the variables in the *lhs* of the 0D system and let

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

be the extended state vector. The 0D system can be written in compact form as

$$\frac{d\Phi}{dt} = f(\Phi) \text{ and } \frac{d\tilde{\Phi}}{dt} = \tilde{f}(\tilde{\Phi})$$

where $f = \{S_T, S_{Y_1}, \ldots S_{Y_{N_{spec}}}\}^T$ and $\tilde{f} = \{S_\rho, S_P, S_T, S_{Y_1}, \ldots S_{Y_{N_{spec}}}\}^T$. The thermodynamic pressure P was introduced for completeness. For open batch reactors P is constant and $S_P \equiv 0$. The source term S_ρ is computed considering the ideal gas equation of state

$$P = \rho R \sum \frac{Y_i}{W_i} T$$

with P=const and using the expressions above for S_T and S_{Y_k},

$$S_\rho = -W \sum_{k=1}^{N_{spec}} \dot{\omega}_k + \frac{1}{c_p T} \sum_{k=1}^{N_{spec}} \dot{\omega}_k W_k h_k$$

Let $\tilde{J}$ and J be the Jacobian matrices corresponding to $\tilde{f}(\tilde{\Phi})$ and $f(\Phi)$, respectively. Chain-rule differentiation leads to

$$\frac{\partial f_u}{\partial v} = \frac{\partial \tilde{f}_u}{\partial v} + \frac{\partial \tilde{f}_u}{\partial \rho}\frac{\partial \rho}{\partial v}$$

Note that each component u of Φ is also a component of $\tilde{\Phi}$ and the corresponding *rhs* components are also the same, $f_u(\Phi) = \tilde{f}_u(\tilde{\Phi})$.

Evaluation of $\tilde{J}$ components

We first identify the dependencies on the elements of $\tilde{\Phi}$ for each of the components of $\tilde{f}$

- $\tilde{f}_1 = S_\rho$. We postpone the discussion for this component.
- $\tilde{f}_2 = S_P = 0$

- $\tilde{f}_3 = S_T$. S_T is defined above. Here we highlight its dependencies on the elements of $\tilde{\Phi}$

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

  where $\mathfrak{X}_k$ is the molar concentration of species k, $\mathfrak{X}_k = \rho Y_k / W_k$.

  $$\tilde{J}_{3,1} = \frac{\partial \tilde{f}_3}{\partial \rho} = \frac{1}{\rho c_p} \sum h_k \left(\frac{\dot{\omega}_k}{\rho} - \frac{\partial \dot{\omega}_k}{\partial \rho} \right),\ \tilde{J}_{3,2} = 0, \tag{6.2.1}$$

  $$\tilde{J}_{3,3} = \frac{\partial \tilde{f}_3}{\partial T} = \frac{1}{\rho c_p^2} \frac{d c_p}{dT} \sum h_k \dot{\omega}_k - \frac{1}{\rho c_p} \sum c_{p_k} \dot{\omega}_k - \frac{1}{\rho c_p} \sum h_k \frac{\partial \dot{\omega}_k}{\partial T}, \tag{6.2.2}$$

  $$\tilde{J}_{3,3+j} = \frac{\partial \tilde{f}_3}{\partial Y_j} = \frac{1}{\rho c_p^2} c_{p_j} \sum h_k \dot{\omega}_k - \frac{1}{\rho c_p} \sum h_k \frac{\partial \dot{\omega}_k}{\partial Y_j},\ j = 1, 2, \ldots, N_{spec} \tag{6.2.3}$$

- $\tilde{f}_{3+k} = S_{Y_k}$

  $$\tilde{J}_{3+k,1} = \frac{\partial \tilde{f}_{3+k}}{\partial \rho} = \frac{W_k}{\rho} \left(\frac{\partial \dot{\omega}_k}{\partial \rho} - \frac{\dot{\omega}_k}{\rho} \right),\ \tilde{J}_{3+k,2} = \frac{\partial \tilde{f}_{3+k}}{\partial P} \equiv 0, \tag{6.2.4}$$

  $$\tilde{J}_{3+k,3} = \frac{\partial \tilde{f}_{3+k}}{\partial T} = \frac{W_k}{\rho} \frac{\partial \dot{\omega}_k}{\partial T},\ \tilde{J}_{3+k,3+j} = \frac{\partial \tilde{f}_{3+k}}{\partial Y_j} = \frac{W_k}{\rho} \frac{\partial \dot{\omega}_k}{\partial Y_j},\ j, k = 1, 2, \ldots, N_{spec} \tag{6.2.5}$$

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

  $$\frac{\partial c_p}{\partial Y_k} = c_{p_k},\ \frac{\partial c_p}{\partial T} = \sum Y_k \frac{d c_{p_k}}{dT},\ \frac{d c_{p_k}}{dT} = R_k \Big(a_{1,k} + T \big(2a_{2,k} + T \left(3a_{3,k} + 4a_{4,k} T \right) \big) \Big)$$

  The partial derivatives of the species production rates, $\dot{\omega}_k(T, \mathfrak{X}_1, \mathfrak{X}_2, \ldots)$, are computed as as

  $$\left. \frac{\partial \dot{\omega}_k}{\partial \rho} \right|_{T,Y} = \sum_{l=1}^{N_{spec}} \frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_l} \frac{\partial \mathfrak{X}_l}{\partial \rho} + \frac{\partial \dot{\omega}_k}{\partial T} \underbrace{\frac{\partial T}{\partial \rho}}_{0} + \underbrace{\frac{\partial \dot{\omega}_k}{\partial \rho}}_{0} \frac{\partial \rho}{\partial \rho} = \sum_{l=1}^{N_{spec}} \frac{Y_l}{W_l} \frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_l}, \tag{6.2.6}$$

  $$\left. \frac{\partial \dot{\omega}_k}{\partial Y_j} \right|_{\rho,T,Y_{\neq j}} = \sum_{l=1}^{N_{spec}} \frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_l} \frac{\partial \mathfrak{X}_l}{\partial Y_j} + \frac{\partial \dot{\omega}_k}{\partial T} \underbrace{\frac{\partial T}{\partial Y_j}}_{0} + \frac{\partial \dot{\omega}_k}{\partial \rho} \underbrace{\frac{\partial \rho}{\partial Y_j}}_{0} = \frac{\rho}{W_j} \frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_j} \tag{6.2.7}$$

  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 production rate $\dot{\omega}_k$ of species k

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

- Derivatives of rate-of-progress variable q_i of reaction i

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

- Derivatives of $\mathscr{C}_i$
  - Basic reactions $\mathscr{C}_i = 1$: $\frac{\partial \mathscr{C}_i}{\partial T} \equiv \frac{\partial \mathscr{C}_i}{\partial \mathfrak{X}_l} \equiv 0$
  - 3-rd body-enhanced reactions $\mathscr{C}_i = \mathfrak{X}_i$: $\frac{\partial \mathscr{C}_i}{\partial T} \equiv 0$, $\frac{\partial \mathscr{C}_i}{\partial \mathfrak{X}_l} = \alpha_{il}$
  - Unimolecular/recombination fall-off reactions $\mathscr{C}_i = \frac{\mathrm{Pr}_i}{1+\mathrm{Pr}_i} F_i$

$$\frac{\partial \mathscr{C}_i}{\partial T} = \frac{1}{(1+\mathrm{Pr}_i)^2}\frac{\partial\,\mathrm{Pr}_i}{\partial T}F_i + \frac{\mathrm{Pr}_i}{1+\mathrm{Pr}_i}\frac{\partial F_i}{\partial T} \tag{6.2.8}$$

$$\frac{\partial \mathscr{C}_i}{\partial \mathfrak{X}_l} = \frac{1}{(1+\mathrm{Pr}_i)^2}\frac{\partial\,\mathrm{Pr}_i}{\partial \mathfrak{X}_l}F_i + \frac{\mathrm{Pr}_i}{1+\mathrm{Pr}_i}\frac{\partial F_i}{\partial \mathfrak{X}_l} \tag{6.2.9}$$

    - $\mathrm{Pr}_i = \frac{k_{0i}}{k_{\infty i}}\mathfrak{X}_i \Rightarrow \frac{\partial\,\mathrm{Pr}_i}{\partial T} = \frac{k'_{0i}k_{\infty i} - k_{0i}k'_{\infty i}}{k^2_{\infty i}}\mathfrak{X}_i,\ \frac{\partial\,\mathrm{Pr}_i}{\partial \mathfrak{X}_l} = \frac{k_{0i}}{k_{\infty i}}\alpha_{il}$.
    - $\mathrm{Pr}_i = \frac{k_{0i}}{k_{\infty i}}\mathfrak{X}_m \Rightarrow \frac{\partial\,\mathrm{Pr}_i}{\partial T} = \frac{k'_{0i}k_{\infty i} - k_{0i}k'_{\infty i}}{k^2_{\infty i}}\mathfrak{X}_m,\ \frac{\partial\,\mathrm{Pr}_i}{\partial \mathfrak{X}_l} = \frac{k_{0i}}{k_{\infty i}}\delta_{lm}$, where δ_{lm} is Kroenecker delta symbol.
    - For Lindemann form $F_i = 1 \Rightarrow \frac{\partial F_i}{\partial T} \equiv \frac{\partial F_i}{\partial \mathfrak{X}_l} \equiv 0$.
    - For Troe form

$$\frac{\partial F_i}{\partial T} = \frac{\partial F_i}{\partial F_{cent}}\frac{\partial F_{cent}}{\partial T} + \frac{\partial F_i}{\partial\,\mathrm{Pr}_i}\frac{\partial\,\mathrm{Pr}_i}{\partial T}, \tag{6.2.10}$$

$$\frac{\partial F_i}{\partial \mathfrak{X}_l} = \frac{\partial F_i}{\partial F_{cent}}\underbrace{\frac{\partial F_{cent}}{\partial \mathfrak{X}_l}}_{0} + \frac{\partial Fi}{\partial\,\mathrm{Pr}_i}\frac{\partial\,\mathrm{Pr}_i}{\partial \mathfrak{X}_l} = \frac{\partial F_i}{\partial\,\mathrm{Pr}_i}\frac{\partial\,\mathrm{Pr}_i}{\partial \mathfrak{X}_l} \tag{6.2.11}$$

$$\frac{\partial F_i}{\partial F_{cent}} = \frac{F}{F_{cent}\left(1+\left(\frac{A}{B}\right)^2\right)} - F\ln F_{cent}\left(\frac{2A}{B^3}\right)\frac{A_F B - B_F A}{\left(1+\left(\frac{A}{B}\right)^2\right)^2} \tag{6.2.12}$$

$$\frac{\partial F_i}{\partial\,\mathrm{Pr}_i} = F\ln F_{cent}\left(\frac{2A}{B^3}\right)\frac{A_{\mathrm{Pr}} B - B_{\mathrm{Pr}} A}{\left(1+\left(\frac{A}{B}\right)^2\right)^2} \tag{6.2.13}$$

where

$$A_F = \frac{\partial A}{\partial F_{cent}} = -\frac{0.67}{F_{cent}\ln 10},\ B_F = \frac{\partial B}{\partial F_{cent}} = -\frac{1.1762}{F_{cent}\ln 10} \tag{6.2.14}$$

$$A_{\mathrm{Pr}} = \frac{\partial A}{\partial\,\mathrm{Pr}_i} = \frac{1}{\mathrm{Pr}_i \ln 10},\ B_{\mathrm{Pr}} = \frac{\partial B}{\partial\,\mathrm{Pr}_i} = -\frac{0.14}{\mathrm{Pr}_i \ln 10} \tag{6.2.15}$$

$$\frac{\partial F_{cent}}{\partial T} = -\frac{1-a}{T^{***}}\exp\left(-\frac{T}{T^{***}}\right) - \frac{a}{T^*}\exp\left(-\frac{T}{T^*}\right) + \frac{T^{**}}{T^2}\exp\left(-\frac{T^{**}}{T}\right) \tag{6.2.16}$$

- For SRI form

$$\begin{aligned}\frac{\partial F_i}{\partial T} =& F\left(\frac{e}{T}+\frac{\partial X}{\partial \Pr_i}\frac{\partial \Pr_i}{\partial T}\ln\left(a\exp\left(-\frac{b}{T}\right)+\exp\left(-\frac{T}{c}\right)\right)\right. \\ &\left. + X\frac{\frac{ab}{T^2}\exp\left(-\frac{b}{T}\right)-\frac{1}{c}\exp\left(-\frac{T}{c}\right)}{a\exp\left(-\frac{b}{T}\right)+\exp\left(-\frac{T}{c}\right)}\right) \qquad (6.2.17)\\ \frac{\partial F_i}{\partial \mathfrak{X}_l} =& F\ln\left(a\exp\left(-\frac{b}{T}\right)+\exp\left(-\frac{T}{c}\right)\right)\frac{\partial X}{\partial \Pr_i}\frac{\partial \Pr_i}{\partial \mathfrak{X}_l} \qquad (6.2.18)\\ \frac{\partial X}{\partial \Pr_i} =& -X^2\frac{2\log_1 0\Pr_i}{\Pr_i \ln 10} \qquad (6.2.19)\end{aligned}$$

- Chemically activated bimolecular reactions: $\mathscr{C}_i = \frac{1}{1+\Pr_i}F_i$

$$\frac{\partial \mathscr{C}_i}{\partial T} = -\frac{1}{(1+\Pr_i)^2}\frac{\partial \Pr_i}{\partial T}F_i + \frac{1}{1+\Pr_i}\frac{\partial F_i}{\partial T} \qquad (6.2.20)$$

$$\frac{\partial \mathscr{C}_i}{\partial \mathfrak{X}_l} = -\frac{1}{(1+\Pr_i)^2}\frac{\partial \Pr_i}{\partial \mathfrak{X}_l}F_i + \frac{1}{1+\Pr_i}\frac{\partial F_i}{\partial \mathfrak{X}_l} \qquad (6.2.21)$$

Partial derivatives of $\Pr_i$ and F_i are computed similar to the ones above.

- Derivatives of $\mathscr{R}_i$

$$\frac{\partial \mathscr{R}_i}{\partial T} = k'_{f\,i}\prod_{j=1}^{N_{spec}}\mathfrak{X}_j^{\nu'_{ji}} - k'_{ri}\prod_{j=1}^{N_{spec}}\mathfrak{X}_j^{\nu''_{ji}} \qquad (6.2.22)$$

$$\frac{\partial \mathscr{R}_i}{\partial \mathfrak{X}_l} = \frac{k_{f\,i}\nu'_{li}\prod_{j=1}^{N_{spec}}\mathfrak{X}_j^{\nu'_{ji}}}{\mathfrak{X}_l} - \frac{k_{ri}\nu''_{li}\prod_{j=1}^{N_{spec}}\mathfrak{X}_j^{\nu''_{ji}}}{\mathfrak{X}_l} \qquad (6.2.23)$$

  - $k_{f\,i} = A_iT^{\beta_i}\exp\left(-\frac{E_i}{RT}\right) = A_i\exp\left(\beta_i\ln T - \frac{T_{ai}}{T}\right)$, where $T_{ai} = E_i/R$. The derivative with respect to temperature can be calculated as $k'_{f\,i} = \frac{k_{f\,i}}{T}\left(\beta_i + \frac{T_{ai}}{T}\right)$
  - 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 computed as

$$\begin{aligned}k_{ri} = \frac{k_{f\,i}}{K_{ci}} \Rightarrow k'_{ri} = \frac{k'_{f\,i}K_{ci} - k_{f\,i}K'_{ci}}{K_{ci}^2} &= \frac{\frac{k_{f\,i}}{T}\left(\beta_i+\frac{T_{ai}}{T}\right)}{K_{ci}} - \frac{k_{f\,i}}{K_{ci}}\frac{K'_{ci}}{K_{ci}} \\ &= k_{ri}\left(\frac{1}{T}\left(\beta_i+\frac{T_{ai}}{T}\right)-\frac{K'_{ci}}{K_{ci}}\right). \qquad (6.2.24)\end{aligned}$$

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

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

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 $\tilde{J}$ terms

- Step 1:

$$\tilde{J}_{3+k,2} \equiv 0, \tag{6.2.25}$$

$$\tilde{J}_{3+k,3} = \frac{W_k}{\rho}\frac{\partial \dot{\omega}_k}{\partial T} = \frac{W_k}{\rho}\left[\sum_{j=1}^{N_{reac}} \nu_{kj}\frac{\partial \mathscr{C}_j}{\partial T}\left(\mathscr{R}_{fj} - \mathscr{R}_{rj}\right) + \sum_{j=1}^{N_{reac}} \nu_{kj}\mathscr{C}_j\left(\mathscr{R}_{fj}\frac{k'_{fj}}{k_{fj}} - \mathscr{R}_{rj}\frac{k'_{rj}}{k_{rj}}\right)\right], \tag{6.2.26}$$

$$\tilde{J}_{3+k,3+i} = \frac{W_k}{\rho}\frac{\partial \dot{\omega}_k}{\partial Y_i} = \frac{\partial \dot{\omega}_k}{\partial \mathfrak{X}_i} = \frac{W_k}{W_i}\left[\sum_{j=1}^{N_{reac}} \nu_{kj}\frac{\partial \mathscr{C}_k}{\partial \mathfrak{X}_i}\left(\mathscr{R}_{fj} - \mathscr{R}_{rj}\right) + \sum_{j=1}^{N_{reac}} \nu_{kj}\mathscr{C}_j\frac{\mathscr{R}_{fj}\nu'_{kj} - \mathscr{R}_{rj}\nu''_{kj}}{\mathfrak{X}_i}\right],$$
$$i = 1, 2, \ldots, N_{spec} \tag{6.2.27}$$

Here $\mathscr{R}_j$ and $\mathscr{R}_{rj}$ are the forward and reverse parts, respectively of $\mathscr{R}_j$:

$$\mathscr{R}_{fj} = k_{fj}\prod_{i=1}^{N_{spec}} \mathfrak{X}_i^{\nu'_{ij}},\ \mathscr{R}_{rj} = k_{rj}\prod_{i=1}^{N_{spec}} \mathfrak{X}_i^{\nu''_{ij}}$$

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

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

- Step 3:

$$\tilde{J}_{3,1} = \frac{1}{\rho c_p}\sum_{i=1}^{N_{spec}} W_i h_i\left(\frac{\dot{\omega}_i}{\rho} - \frac{\partial \dot{\omega}_i}{\partial \rho}\right) = -\frac{1}{c_p}\sum_{i=1}^{N_{spec}} h_i\tilde{J}_{3+i,1},\ \tilde{J}_{3,2} \equiv 0 \tag{6.2.28}$$

$$\begin{aligned}\tilde{J}_{3,3} &= \frac{1}{\rho c_p}\left[\frac{1}{c_p}\frac{\partial c_p}{\partial T}\sum_{i=1}^{N_{spec}} W_i h_i\dot{\omega}_i - \sum_{i=1}^{N_{spec}} W_i c_{p_i}\dot{\omega}_i\right] - \frac{1}{\rho c_p}\sum_{i=1}^{N_{spec}} W_i h_i\frac{\partial \dot{\omega}_i}{\partial T} \\ &= \frac{1}{\rho c_p}\left[\frac{1}{c_p}\frac{c_p}{T}\sum_{i=1}^{N_{spec}} W_i h_i\dot{\omega}_i - \sum_{i=1}^{N_{spec}} W_i c_{p_i}\dot{\omega}_i\right] - \frac{1}{c_p}\sum_{i=1}^{N_{spec}} h_i\tilde{J}_{3+i,3}\end{aligned} \tag{6.2.29}$$

Evaluation of J components

- *Temperature equation*

$$J_{1,1} = \tilde{J}_{3,3} + \tilde{J}_{3,1}\frac{\partial\rho}{\partial T},\ J_{1,1+k} = \tilde{J}_{3,3+k} + \tilde{J}_{3,1}\frac{\partial\rho}{\partial Y_k}$$

- *Species equations*

$$J_{i,1} = \tilde{J}_{1+i,3} + \tilde{J}_{i+1,1}\frac{\partial\rho}{\partial T}, \tag{6.2.30}$$

$$J_{i,1+k} = \tilde{J}_{i+1,3+k} + \tilde{J}_{i+1,1}\frac{\partial\rho}{\partial Y_k},\ k = 1,2,\dots,N_{spec} \tag{6.2.31}$$

For $P = const$ density is a dependent variable, calculated based on the ideal gas equation of state:

$$\rho = \frac{P}{RT\sum_{k=1}^{N_{spec}}\frac{Y_k}{W_k}}$$

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

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

6.2.2. Running the 0D Ignition Utility

The executable to run this example is installed at `TCHEM_INSTALL_PATH/example/`, and the inputs parameters are (`./TChem_IgnitionZeroDSA.x --help`):

```
options:
 --OnlyComputeIgnDelayTime     bool      If true, simulation will end when Temperature is equal
     to T_threshold
                                         (default: --OnlyComputeIgnDelayTime=false)
 --T_threshold                 double    Temp threshold in ignition delay time
                                         (default: --T_threshold=1.5e+03)
 --atol-newton                 double    Absolute tolerence used in newton solver
                                         (default: --atol-newton=1.0e-10)
 --chemfile                    string    Chem file name e.g., chem.inp
                                         (default: --chemfile=chem.inp)
 --dtmax                       double    Maximum time step size
                                         (default: --dtmax=1.00e-01)
 --dtmin                       double    Minimum time step size
                                         (default: --dtmin=1.00e-08)
 --echo-command-line           bool      Echo the command-line but continue as normal
 --help                        bool      Print this help message
 --inputsPath                  string    path to input files e.g., data/inputs
                                         (default: --inputsPath=data/ignition-zero-d/CO/)
 --max-newton-iterations       int       Maximum number of newton iterations
                                         (default: --max-newton-iterations=100)
 --max-time-iterations         int       Maximum number of time iterations
                                         (default: --max-time-iterations=1000)
 --output_frequency            int       save data at this iterations
                                         (default: --output_frequency=-1)
 --rtol-newton                 double    Relative tolerance used in newton solver
                                         (default: --rtol-newton=1.0e-06)
 --samplefile                  string    Input state file name e.g.,input.dat
```

```
                                        (default: --samplefile=sample.dat)
 --tbeg                         double    Time begin
                                        (default: --tbeg=0.0)
 --team-size                    int       User defined team size
                                        (default: --team-size=-1)
 --tend                         double    Time end
                                        (default: --tend=1.0)
 --thermfile                    string    Therm file namee.g., therm.dat
                                        (default: --thermfile=therm.dat)
 --time-iterations-per-interval int       Number of time iterations per interval to store qoi
                                        (default: --time-iterations-per-interval=10)
 --tol-time                     double    Tolerance used for adaptive time stepping
                                        (default: --tol-time=1.0e-04)
 --use_prefixPath               bool      If true, input file are at the prefix path
                                        (default: --use_prefixPath=true)
 --vector-size                  int       User defined vector size
                                        (default: --vector-size=-1)
 --verbose                      bool      If true, printout the first Jacobian values
                                        (default: --verbose=true)
Description:
 This example computes the solution of an ignition problem
```

GRIMech 3.0 model

We can create a bash scripts to provide inputs to TChem. For example the following script runs an ignition problem with the GRIMech 3.0 model [14]:

```
exec=$TCHEM_INSTALL_PATH/TChem_IgnitionZeroDSA.x
inputs=$TCHEM_INSTALL_PATH/data/ignition-zero-d/gri3.0/
save=1
dtmin=1e-8
dtmax=1e-3
tend=2
max_time_iterations=260
max_newton_iterations=20
atol_newton=1e-12
rtol_newton=1e-6
tol_time=1e-6

$exec --inputsPath=$inputs --tol-time=$tol_time --atol-newton=$atol_newton --rtol-newton=
    $rtol_newton --dtmin=$dtmin --max-newton-iterations=$max_newton_iterations --output_frequency
    =$save --dtmax=$dtmax --tend=$tend --max-time-iterations=$max_time_iterations
```

In the above bash script the “inputs” variables is the path to where the inputs files are located in this case (`TCHEM_INSTALL_PATH/example/data/ignition-zero-d/gri3.0`). In this directory, the gas reaction mechanism is defined in “chem.inp” and the thermal properties in “therm.dat”. Additionally, “sample.dat” contains the initial conditions for the simulation.

The parameters “dtmin” and “dtmax” control the size of the time steps in the solver. The decision on increase or decrease time step depends on the parameter “tol_time”. This parameter controls the error in each time iteration, thus, a bigger value will allow the solver to increase the time step while a smaller value will result in smaller time steps. The time-stepping will end when the time reaches “tend”. The simulation will also end when the number of time steps reache “max_time_iterations”. The absolute and relative tolerances in the Newton solver in each iteration are set with “atol_newton” and “rtol_newton”, respectively, and the maximum number of Newton solver iterations is set with “max_newton_iterations”.

The user can specify how often a solution is saved with the parameter "save". Thus, a solution will be saved at every iteration for this case. The default value of this input is -1, which means no output will be saved. The simulation results are saved in "IgnSolution.dat", with the following format:

```
iter   t   dt   Density[kg/m3]   Pressure[Pascal] Temperature[K] MF_SPECIES1 ... MF_SPECIESN
```

where MF_SPECIES1 respresents the mass fraction of species #1, and so forth. Finally, we provide two methods to compute the ignition delay time. In the first approach, we save the time where the gas temperature reaches a threshold temperature. This temperature is set by default to 1500K. In the second approach, save the location of the inflection point for the temperature profile as a function of time, also equivalent to the time when the second derivative of temperature with respect to time is zero. The result of these two methods are saved in files "IgnitionDelayTimeTthreshold.dat" and "IgnitionDelayTime.dat", respectively.

GRIMech 3.0 results

The results presented below are obtained by running `TCHEM_INSTALL_PATH/example/TChem_IgnitionZeroDSA.x` with an initial temperature of 1000K, pressure of 1atm and a stoichiometric equivalence ratio (ϕ) for methane/air mixtures. The input files are located at `TCHEM_INSTALL_PATH/example/data/ignition-zero-d/gri3.0/` and selected parameters were presented above. The outputs of the simulation were saved every iteration in "IgnSolution.dat". Time profiles for temperature and mass fractions for selected species are presented in Figs. 6-1.

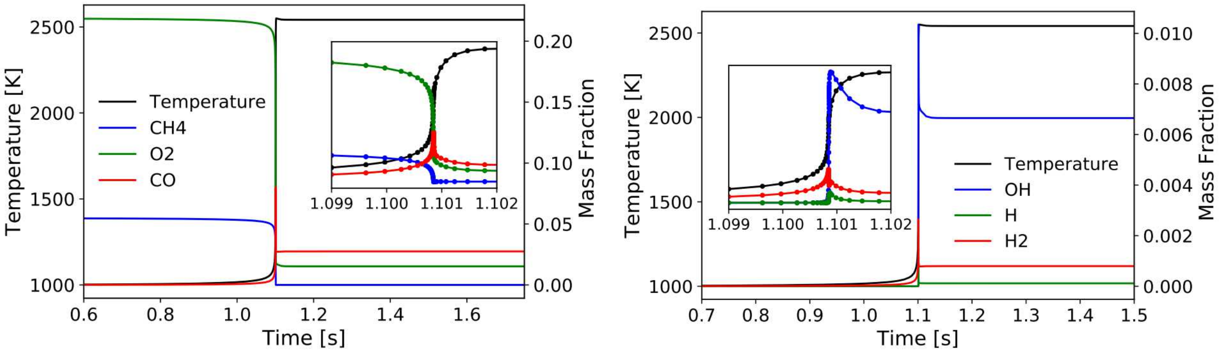

Figure 6-1. Left frame: Temperature and CH_4, O_2, CO Mass Fractions; Right frame: Temperature and OH, H, $H2$ Mass Fractions

The ignition delay time values based on the two alternative computations discussed above are 1.100791s and 1.100854s, respectively. The scripts to setup and run this example and the jupyter-notebook used to create these figures can be found under `TCHEM_INSTALL_PATH/example/runs/gri3.0_IgnitionZeroD`.

GRIMech 3.0 results parametric study

Fig. 6-2 shows the ignition delay time as a function of the initial temperature and equivalence ratio values. These results are based on settings provided in `TCHEM_INSTALL_PATH/example/runs/gri3.0_IgnDelay` and correspond to 100 samples. "TChem_IgnitionZeroDSA.x" runs these samples in parallel. The wall-time is between $200-300s$ on a 3.1GHz Intel Core i7 cpu. We also provide a jupyter-notebook to produce the sample file "sample.dat" and to generate the figure presented above.

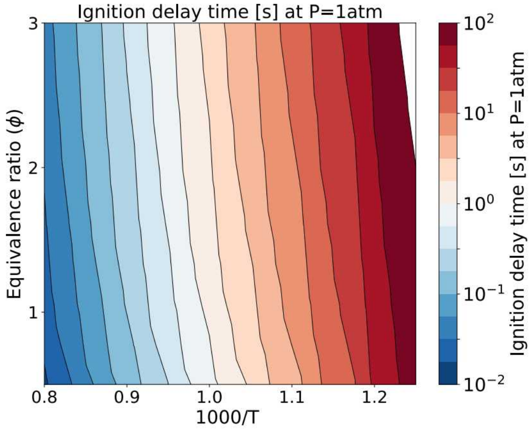

Figure 6-2. Ignition delay times [s] at P=1 atm for several CH_4/air equivalence ratio ϕ and initial temperature values. Results are based on the GRI-Mech v3.0 kinetic model.

6.2.3. *Ignition Delay Time Parameter Study for IsoOctane*

We present a parameter study for several equivalence ratio, pressure, and initial temperature values for iso-Octane/air mixtures. The iso-Octane reaction mechanism used in this study consists of 874 species and 3796 elementary reactions [12]. We selected four pressure values, $\{10, 16, 34, 45\}$ [atm]. For each case we ran a number of simulations that span a grid of 30 initial conditions each for the equivalence ratio and temperature resulting in 900 samples for each pressure value. Each sample was run on a test bed with a Dual-Socket Intel Xeon Platinum architecture.

The data produced by this example is located at `TCHEM_INSTALL_PATH/example/runs/isoOctane_IgnDelay`. Because of the time to produce a result we save the data in a hdf5 format in `isoOctaneIgnDelayBlake.hdf5`. Figs. 6-3 and 6-4 show ignition delay times results for the conditions specified above. These figures were generated with the jupyter notebook shared in the results directory.

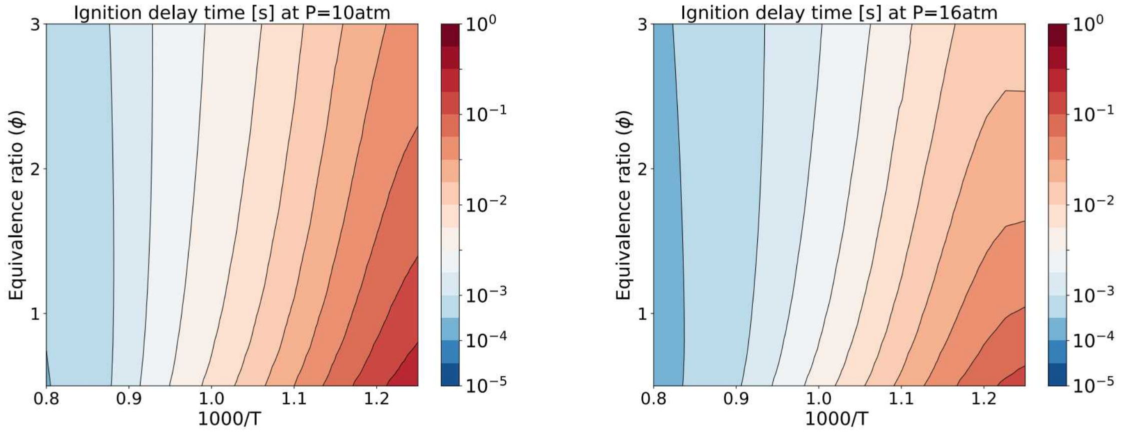

Figure 6-3. Ignition delay times [s] at 10atm (left frame) and 16atm (right frame) for several equivalence ratio (vertical axes) and temperature (horizontal axes) values for iso-Octane/air mixtures.

6.3. Plug Flow Reactor (PFR) Problem with Gas and Surfaces Reactions

6.3.1. Problem Definition

The plug flow reactor (PFR) example employs both gas-phase and surface species. The PFR is assumed to be in steady state, therefore a system of differential-algebraic equations (DAE) must be resolved. The ODE part of the problem correspond to the solution of energy, momentum, total mass and species mass balance. The algebraic constraint arises from the assumption that the PFR problem is a steady-state problem. Thus, the surface composition on the wall must be stationary.

The equations for the species mass fractions Y_k, temperature T, axial velocity u, and continuity (represented by density ρ) resolved by TChem were derived from Ref. [9].

$$\frac{dY_k}{dz} = \frac{1}{\rho u}\dot{\omega}_k W_k + \frac{P_r^{'}}{\rho u A_c}\dot{s}_k W_k - \frac{P_r^{'}}{\rho u A_c} Y_k \sum_{j=1}^{N_{spec}^g} \dot{s}_j W_j \tag{6.3.1}$$

$$\frac{dT}{dz} = -\frac{1}{\rho u c_p}\sum_{k=1}^{N_{spec}^g} \dot{\omega}_k W_k h_k - \frac{P_r^{'}}{\rho u A_c c_p}\sum_{k=1}^{N_{spec}^g} \dot{s}_k W_k h_k \tag{6.3.2}$$

$$\frac{du}{dz} = -\gamma \frac{P_r^{'}}{\rho A_c}\sum_{k=1}^{N_{spec}^g} \dot{s}_k W_k - \frac{R}{um}\left(\frac{1}{\bar{W}}\frac{dT}{dz} + T\sum_{k=1}^{N_{spec}^g}\frac{dY_k}{dz}\frac{1}{W_k}\right) \tag{6.3.3}$$

$$\frac{d\rho}{dz} = \frac{P_r^{'}}{u A_c}\sum_{k=1}^{N_{spec}^g} \dot{s}_k W_k - \frac{\rho}{u}\frac{du}{dz} \tag{6.3.4}$$

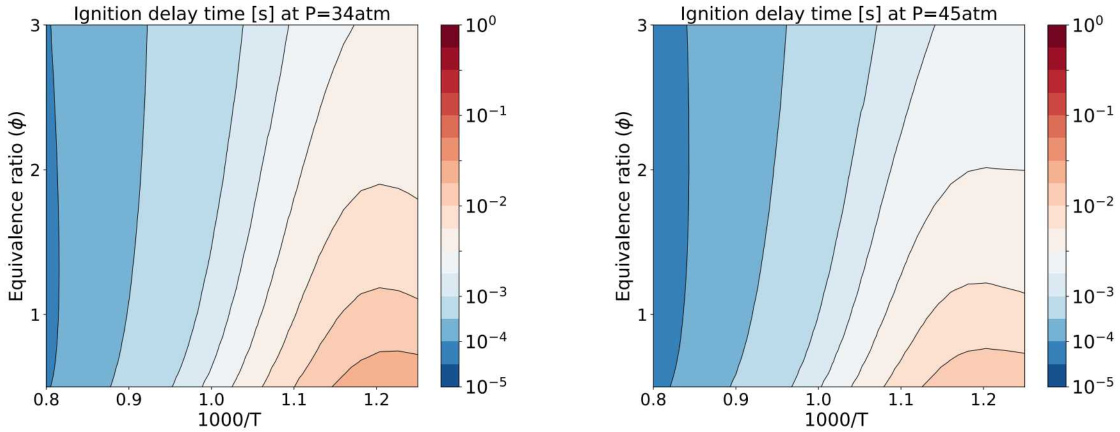

Figure 6-4. Ignition delay times [s] at 34atm (left frame) and 45atm (right frame) for several equivalence ratio (vertical axes) and temperature (horizontal axes) values for iso-Octane/air mixtures.

where $\gamma = \frac{1+\frac{p}{\rho u^2}}{1-\frac{p}{\rho u^2}}$, $m = 1 - \frac{p}{\rho u^2}$, A_c is the surface area, P_r' is the surface chemistry parameter. In the equations above $\dot{s}_k$ represents the surface chemistry production rate for a gas-phase species k.

Algebraic constraint

$$\dot{s}_k = 0 \quad k = 1, \ldots, N_{spec}^{s}.$$

Here N_{spec}^{s} represent all surface species.

The number of ODEs is equal to the number of gas-phases species with three additional equations for thermodynamic temperature, continuity and momentum. The number of constraints is equal to the number of surfaces species. This PFR formulation assumes that surface reactions are taking place on the channel wall and gas-phase reactions inside the channel. Wall friction and heat transfer at the wall are neglected in this example.

6.3.2. Jacobian Formulation

The current implementation uses a numerical jacobian based on forward finite differences [13].

6.3.3. Running the Plug Flow Reactor with Surface Reactions Utility

The executable for this example is installed under `TCHEM_INSTALL_PATH/example/`. The inputs for this example are obtained through

```
./TChem_PlugFlowReactor.x --help
```

```
Usage: ./TChem_PlugFlowReactor.x [options]
  options:
  --Area                        double    Cross-sectional Area
                                          (default: --Area=5.3e-04)
  --Pcat                        double    Chemically active perimeter,
                                          (default: --Pcat=2.6e-02)
  --atol-newton                 double    Absolute tolerance used in newton solver
                                          (default: --atol-newton=1.e-12)
  --batchsize                   int       Batchsize the same state vector described in state file
       is cloned
                                          (default: --batchsize=1)
  --chemSurffile                string    Chem file name e.g., chemSurf.inp
                                          (default: --chemSurffile=chemSurf.inp)
  --chemfile                    string    Chem file name e.g., chem.inp
                                          (default: --chemfile=chem.inp)
  --dzmax                       double    Maximum dz step size
                                          (default: --dzmax=1.0e-06)
  --dzmin                       double    Minimum dz step size
                                          (default: --dzmin=1.0e-10)
  --echo-command-line           bool      Echo the command-line but continue as normal
  --help                        bool      Print this help message
  --initial_condition           bool      If true, use a newton solver to obtain initial
      condition of the constraint
                                          (default: --initial_condition=True)
  --inputSurffile               string    Input state file name e.g., inputSurfGas.dat
                                          (default: --inputSurffile=inputSurf.dat)
  --inputVelocityfile           string    Input state file name e.g., inputVelocity.dat
                                          (default: --inputVelocityfile=inputVelocity.dat)
  --max-newton-iterations       int       Maximum number of newton iterations
                                          (default: --max-newton-iterations=100)
  --max-z-iterations         int       Maximum number of z iterations
                                          (default: --max-z-iterations=4000)
  --output_frequency            int       save data at this iterations
                                          (default: --output_frequency=-1)
  --prefixPath                  string    prefixPath e.g.,inputs/
                                          (default: --prefixPath=data/plug-flow-reactor/X/)
  --rtol-newton                 double    Relative tolerance used in newton solver
                                          (default: --rtol-newton=1.0e-06)
  --samplefile                  string    Input state file name e.g., input.dat
                                          (default: --samplefile=sample.dat)
  --zbeg                        double    Position begin
                                          (default: --zbeg=0)
  --team-size                   int       User defined team size
                                          (default: --team-size=-1)
  --zend                        double    Position  end
                                          (default: --zend=2.5e-02)
  --thermSurffile               string    Therm file name e.g.,thermSurf.dat
                                          (default: --thermSurffile=thermSurf.dat)
  --thermfile                   string    Therm file name e.g., therm.dat
                                          (default: --thermfile=therm.dat)
  --time-iterations-per-intervalint       Number of time iterations per interval to store qoi
                                          (default: --time-iterations-per-interval=10)
  --tol-z                    double    Tolerance used for adaptive z stepping
                                          (default: --tol-z=1.0e-04)
  --transient_initial_condition bool      If true, use a transient solver to obtain initial
      condition of the constraint
                                          (default: --transient_initial_condition=false)
  --use_prefixPath              bool      If true, input file are at the prefix path
                                          (default: --use_prefixPath=true)
  --vector-size                 int       User defined vector size
                                          (default: --vector-size=-1)
  --verbose                     bool      If true, printout the first Jacobian values
                                          (default: --verbose=true)
Description:
  This example computes Temperature, density, mass fraction and site fraction for a plug flow
      reactor
```

The following shell script sets the input parameters and runs the PFR example

```
exec=$TCHEM_INSTALL_PATH/example/TChem_PlugFlowReactor.x
inputs=$TCHEM_INSTALL_PATH/example/data/plug-flow-reactor/CH4-PTnogas/
Area=0.00053
Pcat=0.025977239243415308
dzmin=1e-12
dzmax=1e-5
zend=0.025
tol_z=1e-8
max_z_iterations=310
max_newton_iterations=20
atol_newton=1e-12
rtol_newton=1e-8
save=1
transient_initial_condition=false
initial_condition=true

$exec --prefixPath=$inputs --initial_condition=$initial_condition --transient_initial_condition=
    $transient_initial_condition --Area=$Area  --Pcat=$Pcat --tol-z=$tol_z --atol-newton=
    $atol_newton --rtol-newton=$rtol_newton --dzmin=$dzmin --max-newton-iterations=
    $max_newton_iterations --output_frequency=$save --dzmax=$dzmax --zend=$zend --max-time-
    iterations=$max_z_iterations
```

We ran the example in the install directory

`TCHEM_INSTALL_PATH/example/runs/PlugFlowReactor/CH4-PTnogas`

Thus, all the paths are relative to this directory. This script will run the executable

`TCHEM_INSTALL_PATH/example/TChem_PlugFlowReactor.x`

with the input files located at

`TCHEM_INSTALL_PATH/example/data/plug-flow-reactor/CH4-PTnogas/.`

These files correspond to the gas-phase and surface reaction mechanisms ("chem.inp" and "chemSurf.inp") and their corresponding thermo files ("therm.dat" and "thermSurf.dat"). The operating condition at the inlet of the reactor, i.e. the gas composition, in "sample.dat", and the initial guess for the site fractions, in "inputSurf.dat", are also required. The format and description of these files are presented in Section 3. The gas velocity at the inlet is provided in "inputVelocity.dat".

The "Area" [m^2] is the cross area of the channel and "Pcat" [m] is the chemical active perimeter of the PFR. The step size is controled by "dzmin", "dzmax", and "tol_z", the simulation will end with the "z"(position) is equal to "zend" or when it reaches the "max_z_iterations'. The relative and absolute tolerance in the Newton solver are set through "atol_newton" and "rtol_newton". The description of the integration method can be found in Section 6.1. The "save" parameter sets the output frequency, in this case equal to 1, which means the information will be saved every stepin "PFRSolution.dat". The following header is saved in the output file

```
iter t dt Density[kg/m3] Pressure[Pascal] Temperature[K] SPECIES1 (Mass Fraction) ... SPECIESN (
    Mass Fraction) SURFACE_SPECIES1 (Site Fraction) ... SURFACE_SPECIESN (Site Fraction) Velocity
    [m/s]
```

The inputs "transient_initial_condition" and "initial_condition" allow us to pick a method to compute an initial condition that satisfies the system of DAE equation as described in Section 6.3.4. In this case, the simulation will use a Newton solver to find an initial surface site fraction to meet the constraint presented above.

Results

The gas-phase and surface mechanisms used in this example represents the catalytic combustion of methane on platinum and was developed by Blondal and co-workers [4]. These mechanisms have 15 gas species, 20 surface species, 47 surface reactions and no gas-phase reactions. The total number of ODEs is 18 and there are 20 constrains. One simulation took about 12s to complete on a MacBook Pro with a 3.1GHz Intel Core i7 processor. Time profiles for temperature, density, velocity, mass fractions and site fractions for selected species are presented in Figs 6-5, 6-6. Scripts and jupyter notebooks for this example are located under

`TCHEM_INSTALL_PATH/example/runs/PlugFlowReactor/CH4-PTnogas`

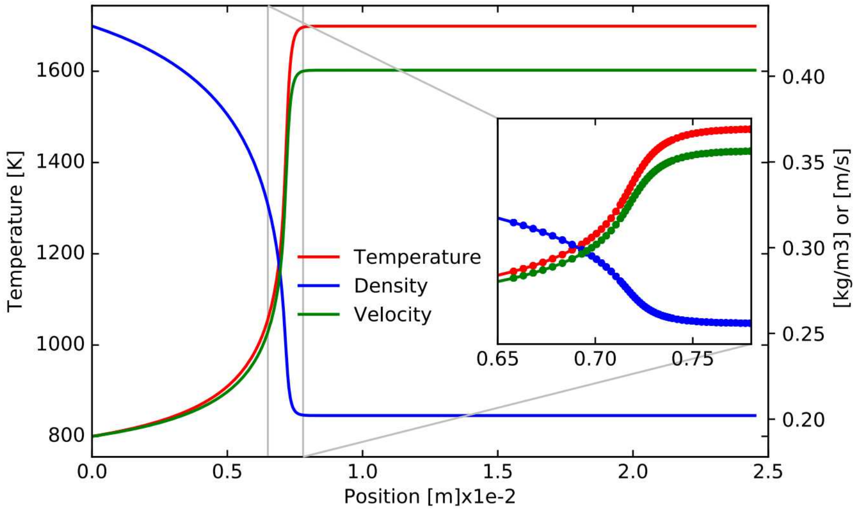

Figure 6-5. Gas Temperature (left axis), velocity and density (both on right axis) along the PFR.

Parametric Study

The executable

`TCHEM_INSTALL_PATH/example/TChem_PlugFlowReactor.x`

can be also used with more than one sample. In this example, we ran it with eight samples. The inputs for this run are located at

`TCHEM_INSTALL_PATH/example/data/plug-flow-reactor/CH4-PTnogas_SA`.

A script and a jupyter-notebook to reproduce this example are placed under

`TCHEM_INSTALL_PATH/example/runs/PlugFlowReactor/CH4-PTnogas_SA`.

These samples correspond to combination of values for the molar fraction of CH_4, $\{0.04, 0.08\}$, inlet gas temperature, $\{800, 9000\}$ [K], and velocity, $\{0.0019, 0.0038\}$ [m/s]. The bash script to run this problem is listed below

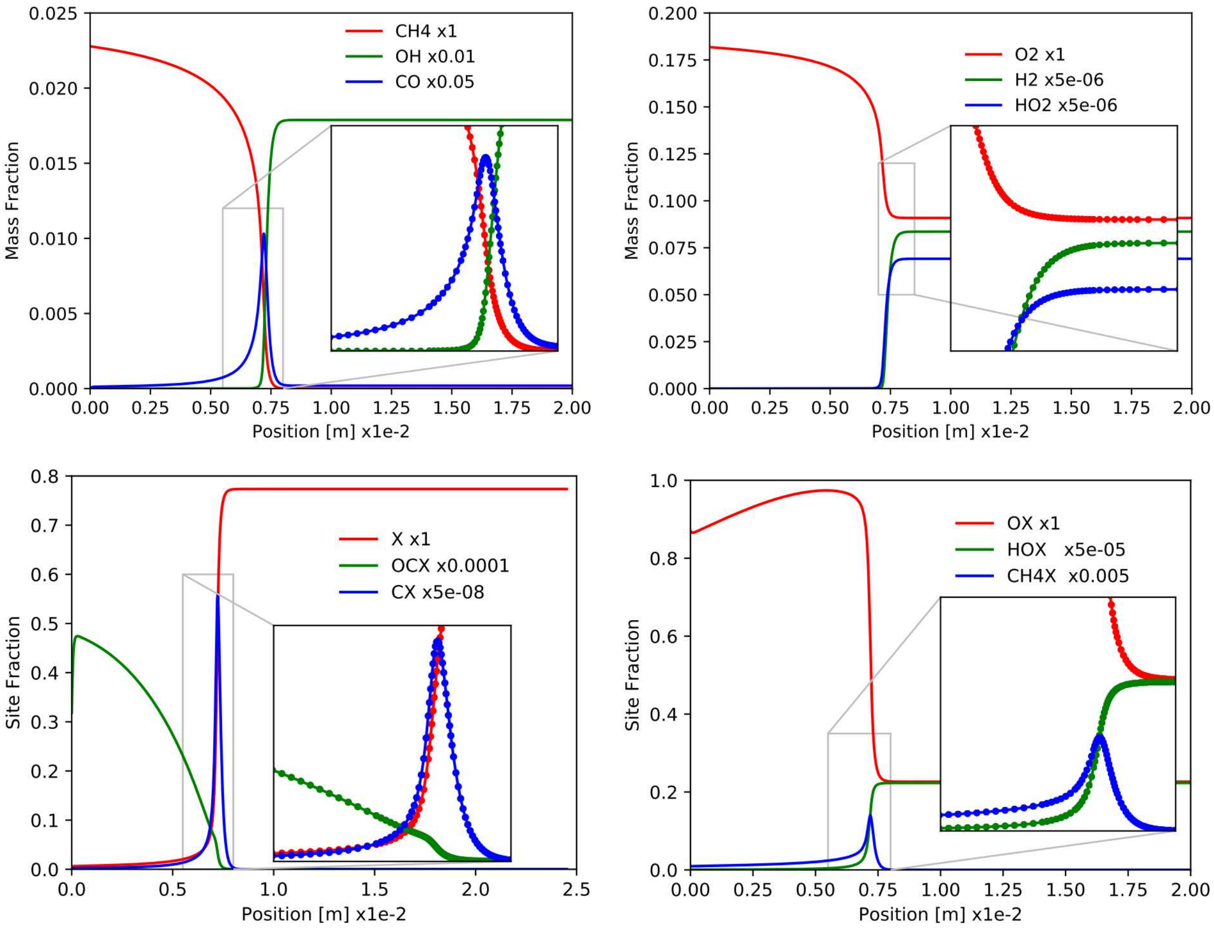

Figure 6-6. Gas-phase species mass fractions and surface species site fractions.

```
exec=$TCHEM_INSTALL_PATH/example/TChem_PlugFlowReactor.x
use_prefixPath=false
inputs=$TCHEM_INSTALL_PATH/example/data/plug-flow-reactor/CH4-PTnogas/
inputs_conditions=inputs/

chemfile=$inputs"chem.inp"
thermfile=$inputs"therm.dat"
chemSurffile=$inputs"chemSurf.inp"
thermSurffile=$inputs"thermSurf.dat"
samplefile=$inputs_conditions"sample.dat"
inputSurffile=$inputs_conditions"inputSurf.dat"
inputVelocityfile=$inputs_conditions"inputVelocity.dat"

save=1
dzmin=1e-12
dzmax=1e-5
zend=0.025
max_newton_iterations=100
max_z_iterations=2000
atol_newton=1e-12
rtol_newton=1e-8
tol_z=1e-8
Area=0.00053
Pcat=0.025977239243415308
transient_initial_condition=true
```

```
initial_condition=false

$exec --use_prefixPath=$use_prefixPath --chemfile=$chemfile --thermfile=$thermfile --chemSurffile
    =$chemSurffile --thermSurffile=$thermSurffile --samplefile=$samplefile --inputSurffile=
    $inputSurffile --inputVelocityfile=$inputVelocityfile --initial_condition=$initial_condition
    --transient_initial_condition=$transient_initial_condition --Area=$Area  --Pcat=$Pcat --tol-z
    =$tol_z --atol-newton=$atol_newton --rtol-newton=$rtol_newton --dzmin=$dzmin --max-newton-
    iterations=$max_newton_iterations --output_frequency=$save --dzmax=$dzmax --zend=$zend --max-
    time-iterations=$max_z_iterations
```

In the above script we did not use a prefix path ("use_prefixPath=false") instead we provided the name of the inputs files: "chemfile", "thermfile", "chemSurffile", "thermSurffile", "samplefile", "inputSurffile", "inputVelocityfile". The files for the reaction mechanism ("chem.inp" and "chemSurf.inp") and the thermo files ("therm.dat" and "thermSurf.dat") are located under `TCHEM_INSTALL_PATH/example/data/plug-flow-reactor/CH4-PTnogas/`

The files with the inlet conditions ("sample.dat", "inputSurf.dat" and"inputVelocity.dat") are located in the "input" directory, located under the run directory. One can set a different path for the input files with the command-line option "use_prefixPath". Additionally, one can also use the option "transient_initial_condition=true", to activate the transient solver to find initial condition for the PFR 6.3.4.

Fig. 6-7 shows temperature, gas-phase species mass fractions and surface species site fractions corresponding to the example presented above

6.3.4. Initial Condition for PFR Problem

The initial condition for the PFR problem must satisfy the algebraic constraint in the DAE system. Thus, an appropriate initial condition must be provided. To solve this problem, TChem first solves a system that accounts for the constraint only. The gas-phase species mass fractions and temperature are kept constant. The constraint component can be solved either by evolving an equivalent time-dependent formulation to steady-state or by directly solving the non-linear problem directly. a steady state or a time dependent formulation. In one method, the following equation is resolved in time until the system reaches stable state. In the second method, a newton solver is used to directly resolver the constraint part($\dot{s}_k = 0$).

$$\frac{dZ_k}{dt} = \frac{\dot{s}_k}{\Gamma} \quad k \text{ correspond to surfaces species} \tag{6.3.5}$$

In the first method, the ODE system is solved until reaches steady state. This is presented at `TCHEM_REPOSITORY_PATH/src/example/TChem_SimpleSurface.cpp`. The Fig. 6-8 shows three surface species, the other species have values lower than 1e-4. This result shows the time to reach stable state is only of 1e-4 s. In the PFR example presented above, this option can be used setting "transient_initial_condition=true" and "initial_condition=false".

The example produces an output file ("InitialConditionPFR.dat") with the last iteration. This file can be used in the PFR problem as the "inputSurf.dat" file. The inputs for this example are located at `TCHEM_INSTALL_PATH/example/runs/InitialConditionPFR`.

In the second method, we used a newton solver to find a solution to the constraint. One code example of this alternative is presented at `TCHEM_REPOSITORY_PATH/src/example/TChem_InitialCondSurface.cpp`. In the PFR example presented above, the default option is “initial_condition=true”, if both option are set true, the code will execute the transient initial condition first and then the newton initial condition.

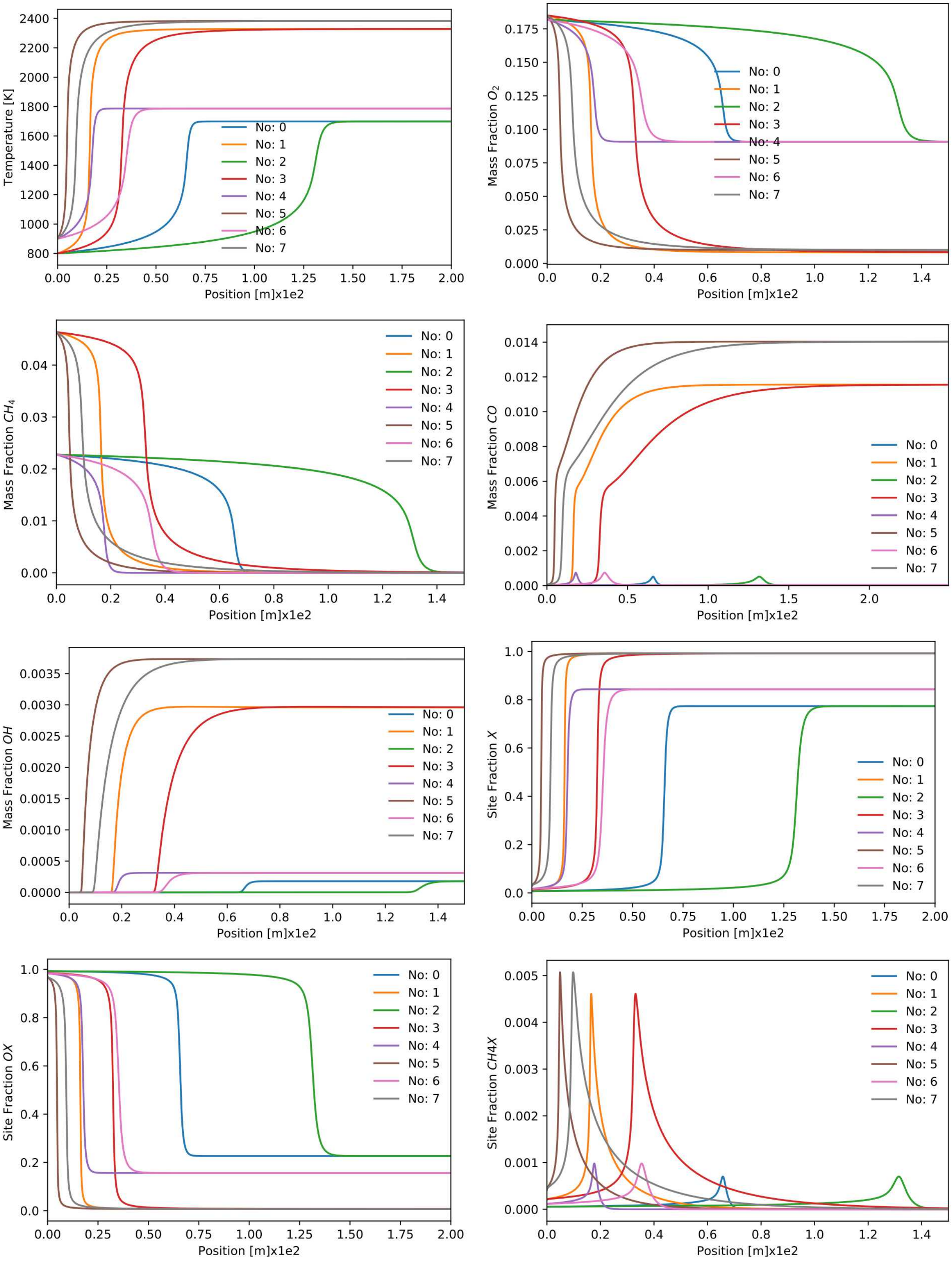

Figure 6-7. Temperature, gas-phase species mass fractions, and surface species site fractions for the example parametric study.

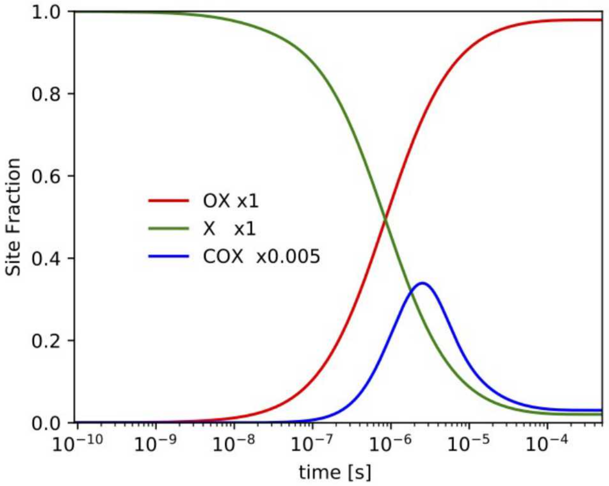

Figure 6-8. Site fractions for X (empty space), OX and COX. We start this simulation with an empty surface ($X = 1$)

7. APPLICATION PROGRAMMING INTERFACE

TChem provides two types of interfaces called `runHostBatch` and `runDeviceBatch`, respectively, to run multiple samples on either the host or the device. The `runHostBatch` uses the “Kokkos::DefaultHostExecutionSpace” with data residing in the host memory. On the other hand, “runDeviceBatch” dispatches the work to “Kokkos::DefaultExecutionSpace” which is configured via Kokkos. In general, the default execution space is configured as OpenMP or Cuda, when GPUs are available. When the device is configured to Cuda, data should be transferred to the device memory using `Kokkos::deep_copy`. The code snippet below illustrates the computation of species production rates for a number of samples in parallel. It reads the kinetic model data and a collection of input state vectors. The input data is then copied to the device memory. After requisite computations are done, the results are copied from the device memory back to the host memory.

```
#include "TChem_Util.hpp"
#include "TChem_ReactionRates.hpp"
#include "TChem_KineticModelData.hpp"

using ordinal_type = TChem::ordinal_type;
using real_type = TChem::real_type;
using real_type_1d_view = TChem::real_type_1d_view;
using real_type_2d_view = TChem::real_type_2d_view;

int main() {
  std::string chemFile("chem.inp");
  std::string thermFile("therm.dat");
  std::string periodictableFile("periodictable.dat");
  std::string inputFile("input.dat");
  std::string outputFile("omega.dat");

  Kokkos::initialize(argc, argv);
  {
    /// kinetic model is constructed and an object is constructed on host
    TChem::KineticModelData kmd(chemFile, thermFile, periodictableFile);

    /// kinetic model data is transferred to the device memory
    const auto kmcd = kmd.createConstData<TChem::exec_space>();

    /// input file includes the number of samples and the size of the state vector
    ordinal_type nBatch, stateVectorSize;
    TChem::readNumberOfSamplesAndStateVectorSize(inputFile, nBatch, stateVectorSize);

    /// create a 2d array storing the state vectors
    real_type_2d_view state("StateVector", nBatch, stateVectorSize);
    auto state_host = Kokkos::create_mirror_view(state);

    /// read the input file and store them into the host array
    TChem::readStateVectors(inputFile, state_host);
    /// if execution space is host execution space, this deep copy is a soft copy
    Kokkos::deep_copy(state, state_host);

    /// output: reaction rates (omega)
    real_type_2d_view omega("ReactionRates", nBatch, kmcd.nSpec);
```

```
    /// create a parallel policy with workspace
    /// for better performance, team size must be tuned instead of using AUTO
    Kokkos::TeamPolicy<TChem::exec_space>
      policy(TChem::exec_space(), nBatch, Kokkos::AUTO());
    const ordinal_type level = 1;
    const ordinal_type per_team_extent = TChem::ReactionRates::getWorkSpaceSize(kmcd);
    const ordinal_type per_team_scratch  =
      TChem::Scratch<real_type_1d_view>::shmem_size(per_team_extent);
    policy.set_scratch_size(level, Kokkos::PerTeam(per_team_scratch));

    /// computes reaction rates
    TChem::ReactionRates::runDeviceBatch(policy, state, omega, kmcd);
    TChem::exec_space().fence();

    /// optionally, one can move the production rate to host memory
    auto omega_host = Kokkos::create_mirror_view(omega);
    Kokkos::deep_copy(omega_host, omega);

    /// print omega_host to std::out
    for (ordinal_type s=0;s<nBatch;++s) {
      std::cout << "Sample ID = " << s << std::endl;
      for (ordinal_type k=0;k<kmcd.nSpec;++k)
         std::cout << omega_host(s, k) << std::endl;
    }
  }
  Kokkos::finalize();

  return 0;
}
```

This workflow pattern can be applied for the other similar functions, presented in Section 7.1, 7.2, and 7.4.

The homogenous batch reactor and the other examples described in Section 6 require a different workflow from the above example. For this example category the user needs to specify a time advance object including the range of time integration, time step sizes, Newton solver tolerances, etc. The following example corresponds to the homogenous batch reactor example.

```
#include "TChem_Util.hpp"
#include "TChem_KineticModelData.hpp"
#include "TChem_IgnitionZeroD.hpp"

using ordinal_type = TChem::ordinal_type;
using real_type = TChem::real_type;
using time_advance_type = TChem::time_advance_type;

using real_type_0d_view = TChem::real_type_0d_view;
using real_type_1d_view = TChem::real_type_1d_view;
using real_type_2d_view = TChem::real_type_2d_view;

using time_advance_type_0d_view = TChem::time_advance_type_0d_view;
using time_advance_type_1d_view = TChem::time_advance_type_1d_view;

using real_type_0d_view_host = TChem::real_type_0d_view_host;
using real_type_1d_view_host = TChem::real_type_1d_view_host;
using real_type_2d_view_host = TChem::real_type_2d_view_host;

using time_advance_type_0d_view_host = TChem::time_advance_type_0d_view_host;
using time_advance_type_1d_view_host = TChem::time_advance_type_1d_view_host;

int main(int argc, char *argv[]) {
  /// input files
  std::string chemFile("chem.inp");
  std::string thermFile("therm.dat");
  std::string periodictableFile("periodictable.dat");
```

```
std::string inputFile("input.dat");

/// time stepping parameters
/// the range of time begin and end
real_type tbeg(0), tend(1);
/// min and max time step size
real_type dtmin(1e-11), dtmax(1e-6);
/// maximum number of time iterations computed in a single kernels launch
ordinal_type num_time_iterations_per_interval(1);
/// adaptive time stepping tolerance which is compared with the error estimator
real_type tol_time(1e-8);
/// new ton solver absolute and relative tolerence
real_type atol_newton(1e-8), rtol_newton(1e-5);
/// max number of newton iterations
ordinal_Type max_num_newton_iterations(100);
/// max number of time ODE kernel launch
ordinal_type max_num_time_iterations(1e3);

Kokkos::initialize(argc, argv);
{
  /// kinetic model is constructed and an object is constructed on host
  TChem::KineticModelData kmd(chemFile, thermFile, periodictableFile);

  /// kinetic model data is transferred to the device memory
  const auto kmcd = kmd.createConstData<TChem::exec_space>();

  /// input file includes the number of samples and the size of the state vector
  ordinal_type nBatch, stateVectorSize;
  TChem::readNumberOfSamplesAndStateVectorSize(inputFile, nBatch, stateVectorSize);

  /// create a 2d array storing the state vectors
  real_type_2d_view state("StateVector", nBatch, stateVectorSize);
  auto state_host = Kokkos::create_mirror_view(state);

  /// read the input file and store them into the host array
  TChem::readStateVectors(inputFile, state_host);
  /// if execution space is host execution space, this deep copy is a soft copy
  Kokkos::deep_copy(state, state_host);

  /// create time advance objects
  time_advance_type tadv_default;
  tadv_default._tbeg = tbeg;
  tadv_default._tend = tend;
  tadv_default._dt = dtmin;
  tadv_default._dtmin = dtmin;
  tadv_default._dtmax = dtmax;
  tadv_default._tol_time = tol_time;
  tadv_default._atol_newton = atol_newton;
  tadv_default._rtol_newton = rtol_newton;
  tadv_default._max_num_newton_iterations = max_num_newton_iterations;
  tadv_default._num_time_iterations_per_interval = num_time_iterations_per_interval;

  /// each sample is time-integrated independently
  time_advance_type_1d_view tadv("tadv", nBatch);
  Kokkos::deep_copy(tadv, tadv_default);

  /// for print the time evolution of species, we need a host mirror view
  auto tadv_host = Kokkos::create_mirror_view(tadv);
  auto state_host = Kokkos::create_mirror_view(state);

  /// create a parallel execution policy with workspace
  Kokkos::TeamPolicy<TChem::exec_space>
    policy(TChem::exec_space(), nBatch, Kokkos::AUTO());
  const ordinal_type level = 1;
  const ordinal_type per_team_extent = TChem::IgnitionZeroD::getWorkSpaceSize(kmcd);
  const ordinal_type per_team_scratch  =
    TChem::Scratch<real_type_1d_view>::shmem_size(per_team_extent);
  policy.set_scratch_size(level, Kokkos::PerTeam(per_team_scratch));
```

```
    for (; iter < max_num_time_iterations && tsum <= tend; ++iter) {
      /// in each kernel launch, it computes the number of time iterations per
      /// interval
      TChem::IgnitionZeroD::runDeviceBatch
        (policy,
         tadv, state, /// input
         t, dt, state, /// output
         kmcd);
      Kokkos::fence();

      /// terminate this loop when all samples reach the time end
      tsum = zero;
      Kokkos::parallel_reduce(
        Kokkos::RangePolicy<TChem::exec_space>(0, nBatch),
        KOKKOS_LAMBDA(const ordinal_type &i, real_type &update) {
          tadv(i)._tbeg = t(i);
          tadv(i)._dt = dt(i);
          update += t(i);
        },
        tsum);
      Kokkos::fence();
      tsum /= nBatch;

      /// to store or print the state vectors, the data must be transferred to
      /// host memory
      Kokkos::deep_copy(tadv_host, tadv);
      Kokkos::deep_copy(state_host, state);
      UserDefinedPrintStateVector(tadv_host, state_host);
    }
  }
  Kokkos::finalize();
}
```

Funcions pertaining to several reactor models are provided in Section 7.3.

7.1. Thermodynamic Properties

This section lists all top-level function interface for thermodynamic properties. These functions are launching a parallel kernel with a given parallel execution policy.

7.1.1. SpecificHeatCapacityPerMass

```
/// Specific heat capacity per mass
/// ================================
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] CpMass - rank 2d array sized by nBatch x nSpec storing Cp per species
///   [out] CpMixMass - rank 1d array sized by nBatch
///   [in] kmcd -  a const object of kinetic model storing in device memory
#inclue "TChem_SpecificHeatCapacityPerMass.hpp"
TChem::SpecificHeatCapacityPerMass::runDeviceBatch
  (const team_policy_type &policy,
   const real_type_2d_view &state,
   const real_type_2d_view &CpMass,
   const real_type_1d_view &CpMixMass,
   cosnt KineticModelConstDataDevice &kmcd);
```

7.1.2. EnthalpyMass

```
/// Enthalpy per mass
/// =================
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] EnthalpyMass - rank 2d array sized by nBatch x nSpec storing enthalpy per species
///   [out] EnthalpyMixMass - rank 1d array sized by nBatch
///   [in] kmcd -  a const object of kinetic model storing in device memory
#inclue "TChem_EnthalpyMass.hpp"
TChem::EnthalpyMass::runDeviceBatch
  (const team_policy_type &policy,
   const real_type_2d_view &state,
   const real_type_2d_view &EnthalpyMass,
   const real_type_1d_view &EnthalpyMixMass,
   cosnt KineticModelConstDataDevice &kmcd);
```

7.1.3. InternalEnergyMass

```
/// Internal Energy per mass
/// =================
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] InternalEnergyMass - rank 2d array sized by nBatch x nSpec storing enthalpy per
    species
///   [out] InternalEnergyMixMass - rank 1d array sized by nBatch
///   [in] kmcd -  a const object of kinetic model storing in device memory
#inclue "TChem_InternalEnergyMass.hpp"
TChem::InternalEnergyMass::runDeviceBatch
  (const team_policy_type &policy,
   const real_type_2d_view &state,
   const real_type_2d_view &InternalEnergyMass,
   const real_type_1d_view &InternalEnergyMixMass,
   cosnt KineticModelConstDataDevice &kmcd);
```

7.1.4. EntropyMass

```
/// Entropy per mass
/// =================
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] EntropyMass - rank 2d array sized by nBatch x nSpec storing enthalpy per species
///   [out] EntropyMixMass - rank 1d array sized by nBatch
///   [in] kmcd -  a const object of kinetic model storing in device memory
#inclue "TChem_EntropyMass.hpp"
TChem::EntropyMass::runDeviceBatch
  (const team_policy_type &policy,
   const real_type_2d_view &state,
   const real_type_2d_view &EntropyMass,
   const real_type_1d_view &EntropyMixMass,
   cosnt KineticModelConstDataDevice &kmcd);
```

7.2. Chemical Source Terms

This section lists all top-level function interface for the computation of species production rates. These functions are launching a parallel kernel with a given parallel execution policy.

7.2.1. NetProductionRatesPerMass

```
/// Net Production Rates per mass
/// ==============
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] omega - rank 2d array sized by nBatch x nSpec storing reaction rates
///   [in] kmcd -  a const object of kinetic model storing in device memory
#inclue "TChem_NetProductionRatePerMass.hpp"
TChem::NetProductionRatePerMass::runDeviceBatch
  (const team_policy_type &policy,
   const real_type_2d_view &state,
   const real_type_2d_view &omega,
   const KineticModelConstDataDevice &kmcd);
```

7.2.2. NetProductionRatesPerMole

```
/// Net Production Rates per mole
/// ==============
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] omega - rank 2d array sized by nBatch x nSpec storing reaction rates
///   [in] kmcd -  a const object of kinetic model storing in device memory
#inclue "TChem_NetProductionRatePerMole.hpp"
TChem::NetProductionRatePerMole::runDeviceBatch
  (const team_policy_type &policy,
   const real_type_2d_view &state,
   const real_type_2d_view &omega,
   cosnt KineticModelConstDataDevice &kmcd);
```

7.2.3. NetProductionRateSurfacePerMole

```
/// Net Production Rates Surface per mole
/// ==============
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [in] zSurf - rank 2d array sized by nBatch x nSpec(Surface)
///   [out] omega - rank 2d array sized by nBatch x nSpec(Gas) storing reaction rates gas species
///   [out] omegaSurf - rank 2d array sized by nBatch x nSpec(Surface) storing reaction rates
    surface species
///   [in] kmcd -  a const object of kinetic model storing in device memory(gas phase)
///   [in] kmcdSurf -  a const object of kinetic model storing in device memory (Surface phase)
TChem::NetProductionRateSurfacePerMole::runDeviceBatch
  (const real_type_2d_view &state,
   const real_type_2d_view &zSurf,
   const real_type_2d_view &omega,
   const real_type_2d_view &omegaSurf,
   const KineticModelConstDataDevice &kmcd,
   const KineticSurfModelConstDataDevice &kmcdSurf);
```

7.2.4. NetProductionRateSurfacePerMass

```
/// Net Production Rates Surface per mass
/// ==============
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [in] zSurf - rank 2d array sized by nBatch x nSpec(Surface)
///   [out] omega - rank 2d array sized by nBatch x nSpec(Gas) storing reaction rates gas species
///   [out] omegaSurf - rank 2d array sized by nBatch x nSpec(Surface) storing reaction rates
    surface species
///   [in] kmcd -  a const object of kinetic model storing in device memory(gas phase)
///   [in] kmcdSurf -  a const object of kinetic model storing in device memory (Surface phase)
TChem::NetProductionRateSurfacePerMass::runDeviceBatch
```

```
  (const real_type_2d_view &state,
   const real_type_2d_view &zSurf,
   const real_type_2d_view &omega,
   const real_type_2d_view &omegaSurf,
   const KineticModelConstDataDevice &kmcd,
   const KineticSurfModelConstDataDevice &kmcdSurf);
```

7.3. Reactor Models

This section lists all top-level function interfaces for the reactor examples.

7.3.1. *Ignition 0D*

```
/// Ignition 0D
/// ===========
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] tadv - rank 1d array sized by nBatch storing time stepping data structure
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] t_out - rank 1d array sized by nBatch storing time when exiting the function
///   [out] dt_out - rank 1d array sized by nBatch storing time step size when exiting the
    function
///   [out] state_out - rank 2d array sized by nBatch x stateVectorSize storing updated state
    vectors
///   [in] kmcd -  a const object of kinetic model storing in device memory
#inclue "TChem_IgnitionZeroD.hpp"
TChem::IgnitionZeroD::runDeviceBatch
  (const team_policy_type &policy,
   const time_advance_type_1d_view &tadv,
   const real_type_2d_view &state,
   const real_type_1d_view &t_out,
   const real_type_1d_view &dt_out,
   const real_type_2d_view &state_out,
   cosnt KineticModelConstDataDevice &kmcd);
```

7.3.2. *SourceTerm*

```
/// SourceTerm
/// =================
///   [in] nBatch - number of samples
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] SourceTerm - rank2d array by nBatch x number of species + 1 (temperature)
///   [in] kmcd -  a const object of kinetic model storing in device memory
#include "TChem_SourceTerm.hpp"
  TChem::SourceTerm::runDeviceBatch
  (const ordinal_type nBatch,
  const real_type_2d_view& state,
  const real_type_2d_view& SourceTerm,
  const KineticModelConstDataDevice& kmcd);
```

7.3.3. *PlugFlowReactor*

```
/// Plug Flow Reactor
/// =================
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] tadv - rank 1d array sized by nBatch storing time stepping data structure
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [in] zSurf - rank2d array by nBatch x number of surface specues
///   [in] velociy -rank1d array by nBatch
///   [out] t_out - rank 1d array sized by nBatch storing time when exiting the function
```

```
///   [out] dt_out - rank 1d array sized by nBatch storing time step size when exiting the
    function
///   [out] state_out - rank 2d array sized by nBatch x stateVectorSize storing updated state
    vectors
///   [out] z_out - rank2d array by nBatch x number of surface specues
///   [out] velocity_out -rank1d array by nBatch
///   [in] kmcd -  a const object of kinetic model storing in device memory
///   [in] kmcdSurf -  a const object of surface kinetic model storing in device memory
///   [in] area - cross-sectional area
///   [in] pcat - chemically active perimeter
#inclue "TChem_PlugFlowReactor.hpp"
TChem::PlugFlowReactor::runDeviceBatch
  (const team_policy_type &policy,
   const time_advance_type_1d_view &tadv,
   const real_type_2d_view &state,
   const real_type_2d_view &z_surf,
   const real_type_1d_view &velocity,
   const real_type_1d_view &t_out,
   const real_type_1d_view &dt_out,
   const real_type_2d_view &state_out,
   const real_type_2d_view &z_out,
   const real_type_1d_view &velocity_out,
   cosnt KineticModelConstDataDevice &kmcd,
   const KineticSurfModelConstDataDevice &kmcdSurf,
   const real_type area,
   const real_type pcat);
```

7.3.4. PlugFlowReactorRHS

```
/// Plug Flow Reactor RHS
/// =================
///   [in] nBatch - number of samples
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [in] zSurf - rank 2d array by nBatch x number of surface species
///   [in] velocity - rank 2d array sized by nBatch x stateVectorSize
///   [in] kmcd -  a const object of kinetic model storing in device memory
///   [in] kmcdSurf -  a const object of surface kinetic model storing in device memory

#include "TChem_PlugFlowReactorRHS.hpp"
TChem::PlugFlowReactorRHS::runDeviceBatch
(const ordinal_type nBatch,
 const real_type_2d_view& state,
 const real_type_2d_view& zSurf,
 const real_type_2d_view& velocity,
 const real_type_2d_view& rhs,
 const KineticModelConstDataDevice& kmcd,
 const KineticSurfModelConstDataDevice& kmcdSurf);
```

7.3.5. JacobianReduced

```
/// JacobianReduced
/// =================
///   [in] nBatch - number of samples
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] Jacobian - rank 3d array by nBatch  x number of species + 1 x number of species + 1
///   [in] kmcd -  a const object of kinetic model storing in device memory
#include "TChem_JacobianReduced.hpp"
TChem::JacobianReduced::runDeviceBatch
(const ordinal_type nBatch,
const real_type_2d_view& state,
const real_type_3d_view& Jacobian,
const KineticModelConstDataDevice& kmcd);
```

7.3.6. IgnitionZeroDNumJacobian

```
/// IgnitionZeroDNumJacobian
/// =================
///   [in] nBatch - number of samples
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] jac - rank 3d array by nBatch  x number of species + 1 x number of species + 1
///   [out] fac - rank 2d array by nBatch  x number of species + 1
///   [in] kmcd -  a const object of kinetic model storing in device memory
#include "TChem_IgnitionZeroDNumJacobian.hpp"
TChem::IgnitionZeroDNumJacobian::runDeviceBatch
  (const ordinal_type nBatch,
  const real_type_2d_view& state,
  const real_type_3d_view& jac,
  const real_type_2d_view& fac,
  const KineticModelConstDataDevice& kmcd);
```

7.3.7. *InitialConditionSurface*

```
/// InitialConditionSurface
/// =================
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [in] siteFraction - rank2d array by nBatch x number of surface species
///   [out] siteFraction_out - rank2d array by nBatch x number of surface species
///   [in] kmcd -  a const object of kinetic model storing in device memory
///   [in] kmcdSurf -  a const object of surface kinetic model storing in device memory
#include "TChem_InitialCondSurface.hpp"
TChem::InitialCondSurface::runDeviceBatch
(const team_policy_type &policy,
 const real_type_2d_view &state,
 const real_type_2d_view &siteFraction,
 const real_type_2d_view  &siteFraction_out,
 const KineticModelConstDataDevice &kmcd,
const KineticSurfModelConstDataDevice &kmcdSurf);
```

7.3.8. *SimpleSurface*

```
/// Simple surface
/// =================
///   [in] policy - Kokkos parallel execution policy; league size must be nBatch
///   [in] tadv - rank 1d array sized by nBatch storing time stepping data structure
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [in] siteFraction - rank2d array by nBatch x number of surface species
///   [out] t - rank 1d array sized by nBatch storing time when exiting the function
///   [out] dt - rank 1d array sized by nBatch storing time step size when exiting the function
///   [out] siteFraction_out - rank2d array by nBatch x number of surface species
///   [in] kmcd -  a const object of kinetic model storing in device memory
///   [in] kmcdSurf -  a const object of surface kinetic model storing in device memory
#include "TChem_SimpleSurface.hpp"
TChem::SimpleSurface::runDeviceBatch
(const team_policy_type &policy,
 const time_advance_type_1d_view &tadv,
 const real_type_2d_view &state,
 const real_type_2d_view &siteFraction,
 const real_type_1d_view &t,
 const real_type_1d_view &dt,
 const real_type_2d_view  &siteFraction_out,
 const KineticModelConstDataDevice &kmcd,
 const KineticSurfModelConstDataDevice &kmcdSurf);
```

7.4. Kinetic Model - Other Interfaces

This section lists top-level function interfaces for properties derived from the kinetic model specifications.

7.4.1. Smatrix

```
/// S Matrix
/// =================
///   [in] nBatch - number of samples
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] Smatrix - rank3d array by nBatch  x number of species + 1 x twice the number of
    reaction in gas phase
///   [in] kmcd -  a const object of kinetic model storing in device memory
#include "TChem_Smatrix.hpp"
TChem::Smatrix::runDeviceBatch
  (const ordinal_type nBatch,
   const real_type_2d_view& state,
   const real_type_3d_view& Smatrix,
   const KineticModelConstDataDevice& kmcd);
```

7.4.2. RateOfProgress

```
/// RateOfProgress
/// =================
///   [in] nBatch - number of samples
///   [in] state - rank 2d array sized by nBatch x stateVectorSize
///   [out] RoPFor - rank2d array by nBatch x number of reaction in gas phase
///   [out] RoPFor - rank2d array by nBatch x number of reaction in gas phase
///   [in] kmcd -  a const object of kinetic model storing in device memory
#include "TChem_RateOfProgress.hpp"
TChem::RateOfProgress::runDeviceBatch
( const ordinal_type nBatch,
  const real_type_2d_view& state,
  const real_type_2d_view& RoPFor,
  const real_type_2d_view& RoPRev,
  const KineticModelConstDataDevice& kmcd);
```

8. SUMMARY

TChem is the subject of continual development and improvement. If you have questions about or suggestions for features to be adopted in future versions, feel free to e-mail Cosmin Safta at `csafta@sandia.gov`, or share your questions directly on the github page. In the upcoming versions we plan to expand the I/O to support YAML and HDF5 interfaces, improve the performance on GPUs and explore advanced time-stepping algorithms for stiff systems.

REFERENCES

[1] GTEST. `https://github.com/google/googletest`, 2020. Accessed: 2020-08-26.

[2] Kokkos. `https://github.com/kokkos/kokkos`, 2020. Accessed: 2020-08-26.

[3] R. E. Bank, W. M. Coughran, W. Fichtner, E. H. Grosse, D. J. Rose, and R. K. Smith. Transient simulation of silicon devices and circuits. *IEEE Transactions on Electron Devices*, 32(10):1992–2007, 1985.

[4] Katrin Blondal, Jelena Jelic, Emily Mazeau, Felix Studt, Richard H. West, and C. Franklin Goldsmith. Computer-generated kinetics for coupled heterogeneous/homogeneous systems: A case study in catalytic combustion of methane on platinum. *Industrial & Engineering Chemistry Research*, 58(38):17682–17691, 2019.

[5] Michael E. Coltrin, Robert J. Kee, Fran M. Rupley, and Ellen Meeks. Surface Chemkin-III: A FORTRAN Package for Analyzing Heterogenous Chemical Kinetics at a Solid-surface–Gas-phase Interface. Technical Report SAND96-8217, Sandia National Laboratories, Livermore, CA, 1996.

[6] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202 – 3216, 2014. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[7] X. Gou, J. A. Miller, W. Sun, and Y. Ju. PLOG Formulation. `http://engine.princeton.edu`, 2011. Accessed: 2020-08-26.

[8] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[9] R. J. Kee and H. Zhu M. E. Coltrin, P. Glarborg. *Chemically Reacting Flow: Theory, Modeling, and Simulation*. Wiley, 2 edition, 20177.

[10] R.J. Kee, F.M. Rupley, and J.A. Miller. Chemkin-II: A Fortran Chemical Kinetics Package for the Analysis of Gas Phase Chemical Kinetics. Technical Report SAND89-8009B, Sandia National Laboratories, Livermore, CA, August 1993.

[11] 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.

[12] M. Mehl, H. J. Curran, W. J. Pitz, and C. K. Westbrook. Chemical kinetic modeling of component mixtures relevant to gasoline. Technical Report LLNL-CONF-410968, Lawrence Livermore National Laboratory, 2009.

[13] D E Salane. Adaptive routines for forming jacobians numerically. *SAND86*, 8 1986.

[14] 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 Z. Qin. GRI-Mech v3.0. `http://www.me.berkeley.edu/gri_mech`, 2011. Accessed: 2020-08-26.

DISTRIBUTION

Hardcopy—External

Number of Copies	Name(s)	Company Name and Company Mailing Address

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop

Email—Internal (encrypt for OUO)

Name	Org.	Sandia Email Address
Technical Library	01177	libref@sandia.gov

Sandia
National
Laboratories