Rowan J. Gollan and Peter A. Jacobs

The Eilmer Flow Simulation Program:

Reacting Gas Thermochemistry

with the thermally-perfect-gas model

`for Version 4.0`

March 7, 2025

Technical Report 2018/04
School of Mechanical & Mining Engineering
The University of Queensland

Abstract

Eilmer is a program for the simulation of compressible gas flows. It is based on a finite-volume formulation of the mass, momentum, energy and species conservation equations, plus a model of the gas thermochemistry. This report contains the description of the single-temperature reacting-gas formulation, which consists of a mixture of thermally-perfect gas species and a finite-rate kinetics scheme that determines the evolution of the composition of that mixture. This reacting gas module may be used as part of a flow simulation or in a scripted calculation of your own design. Wrappers for Lua, Python3 and Ruby are available for scripted calculations and example calculations are provided in this report. The final example shows the use of the reacting-gas model in a flow simulation.

Contribution Acknowlegements

We would like to thank and acknowledge contributions that have helped to improve this document.

- Andrew Parker, Fluid Gravity Engineering, for identifying a typsetting error in the enthalpy expressions for extrapolated calculations beyond the valid temperature range.

Contents

1 Introduction **1**

2 A thermally-perfect-gas model **3**
2.1 A single-species thermally-perfect gas 3
2.2 Mixing rules for a collection of thermally perfect gases 5

3 Finite-rate chemistry **7**
3.1 Rates of species change due to chemical reaction 7
3.2 Reaction rate coefficients 8
3.3 Solving the chemical kinetic ordinary differential equation 9
3.4 Coupling chemistry effects to the flow solver 11

4 The reactions-scheme file **13**
4.1 Overview of input file format 14
4.2 Details of the `Reaction` table 15
4.3 Extra control of the chemistry scheme 19
4.3.1 Selection and extra control of the ODE methods 21

5 Examples of Use **23**
5.1 Fixed-volume nitrogen reactor 23
5.1.1 Lua programming interface 24
5.1.2 Python programming interface 28
5.1.3 Ruby programming interface 29
5.1.4 CEAgas model 30
5.2 Ignition times for hydrogen 31
5.3 Dissociating nitrogen flow over a 2D cylinder 34
5.3.1 Eilmer input script 35
5.3.2 Results 36

References **39**

A Input files for hydrogen combustion **41**
A.1 Stanford, 2011 41
A.2 Evans-Schexnayder 46

A.3 Rogers-Schexnayder 51

CHAPTER 1

Introduction

When you work with the Eilmer flow solver, you should start your analysis with the simplest gas model that is appropriate. For flows with moderate temperatures of a few hundred degrees, that might very well be the ideal gas model, with fixed composition and heat capacities. But, at higher temperatures, the gas in your flow might undergo chemical reactions. For example in air, by 2 000 K oxygen molecules will begin to dissociate, and by 4 000 K nitrogen molecules will begin to dissociate.

Atmospheric entry of a spacecraft and combustion within a ducted, supersonic flow are typical examples of hypersonic flows that might be analysed with the Eilmer. In these cases, the temperatures may be in the range 1000 K to several thousand degrees and velocities may be a few kilometres per second. Thus, it is likely that changes to the chemical composition of the gas will be significant and also likely that the timescales for these changes will be comparable to the transit time for the gas through the flow domain. Now, for your gas-dynamic analysis with Eilmer, the equations of the flow solver need to be complemented by a set of thermochemical relations describing the behaviour of the gas as a mixture of chemical species that undergoes finite-rate chemical reactions.

In this report, we first consider the thermally-perfect gas model with one or more chemical components (Section 2), all of which have perfect collisional behaviour but each having all internal energy modes excited to an equilibrium described by a single temperature. This has become the "work horse" gas model for hypersonic flow analysis with Eilmer. We then describe the generic finite-rate chemical reaction scheme (Section 3) that can be used with this gas model, followed by a description of the configuration file specification (Section 4). The final section (5) describes some sample applications, in the form of scripted calculations and as part of a flow simulation with the Eilmer flow solver.

The gas module is available as part of the Eilmer source code from the public repository https://github.com/gdtk-uq/gdtk. The home page for Eilmer may be found at https://gdtk.uqcloud.net/docs/eilmer/about/.

CHAPTER 2

A thermally-perfect-gas model

The thermodynamic relations for the gas mixture of given composition are presented here. The implementation of finite-rate chemical effects, which define how the composition changes with time, is discussed in Section 3.

2.1 A single-species thermally-perfect gas

The assumed behaviour of a thermally perfect gas is that all internal energy modes are in equilibrium at a single temperature. For atoms this means that the Boltzmann distributions for translational and electronic energy are governed by one temperature value. Similarly for molecules, the Boltzmann distributions for translational, rotational, vibrational and electronic energy are described by a single temperature value.

To model a thermally perfect gas requires a knowledge of how the gas stores energy as a function of temperature. It is convenient to have available the specific heat at constant pressure as a function of temperature, $C_p(T)$. From this, specific enthalpy of the gas can be computed as

$$h = \int_{T_{ref}}^{T} C_p(T)dT + h(T_{ref}) \tag{2.1}$$

and entropy is given as

$$s = \int_{T_{ref}}^{T} \frac{C_P(T)}{T}dT + s(T_{ref}). \tag{2.2}$$

The transport properties, viscosity and thermal conductivity, can be calculated as a function of temperature for a single component of the gas mix. The transport properties for a single component can be combined by an appropriate mixing rule to give a mixture viscosity and thermal conductivity.

In the implementation, a thermally perfect gas is characterised by five curve fits all of which are functions of temperature:

1. specific heat at constant pressure, $C_p(T)$,
2. enthalpy, $h(T)$,

3. entropy, $s(T)$,
4. viscosity, $\mu(T)$, and
5. thermal conductivity, $k(T)$.

The form of these curve fits follows that used by McBride and Gordon [1]. The curve fits for thermodynamic properties in non-dimensional form are as follows:

$$\frac{C_p(T)}{R} = a_0T^{-2} + a_1T^{-1} + a_2 + a_3T + a_4T^2 + a_5T^3 + a_6T^4 \tag{2.3}$$

$$\frac{H(T)}{RT} = -a_0T^{-2} + a_1T^{-1}\log T + a_2 + a_3\frac{T}{2} + a_4\frac{T^2}{3} + a_5\frac{T^3}{4} + a_6\frac{T^4}{5} + \frac{a_7}{T} \tag{2.4}$$

$$\frac{S(T)}{R} = -a_0\frac{T^{-2}}{2} - a_1T^{-1} + a_2\log T + a_3T + a_4\frac{T^2}{2} + a_5\frac{T^3}{3} + a_6\frac{T^4}{4} + a_8 \tag{2.5}$$

The coefficients for these curve fits are available for a large number of gaseous species in the CEA program [1] (and associated database files). Each of these curve fits are only valid over a limited temperature range. For example, the thermodynamic curve fits for molecular nitrogen (N_2) are comprised of three segments: 200.0–1000.0 K, 1000.0–6000.0 K and 6000.0–20000.0 K. At the boundaries of these polynomials, the values do not precisely match. This mismatch causes grief for the iterative method used to determine temperature given the enthalpy because these methods rely on the underlying function varying smoothly. To overcome this, we use the idea to blend the polynomial coefficients near the boundaries presented in Gupta et al. [2]. At the 1000 K break point, we do a linear blending of the coefficients over a range of 400 K. So, the linear blend starts at 800 K and finishes at 1200 K. At the 6000 K break point, the blending range is 1000 K. It is important to note that the coefficients are blended, then used in the polynomial to evaluate the required properties. The blending is not performed on the values themselves. Beyond the range of the sets of polynomials, the values are extrapolated. The extrapolations are based on a crude assumption of constant C_p outside of the range. Thus the extrapolations are as follows:

$$\begin{aligned}
\frac{C_p(T < T_{low})}{R} &= \frac{C_p(T_{low})}{R} \\
\frac{C_p(T > T_{high})}{R} &= \frac{C_p(T_{high})}{R} \\
\frac{H(T < T_{low})}{RT} &= \frac{1}{T}\left\{H(T_{low}) - C_p(T_{low})(T_{low} - T)\right\} \\
\frac{H(T > T_{high})}{RT} &= \frac{1}{T}\left\{H(T_{high}) + C_p(T_{high})(T - T_{high})\right\} \\
\frac{S(T < T_{low})}{R} &= S(T_{low}) - C_p(T_{low})\log\left(\frac{T_{low}}{T}\right) \\
\frac{S(T > T_{high})}{R} &= S(T_{high}) + C_p(T_{high})\log\left(\frac{T}{T_{high}}\right)
\end{aligned}$$

The curve fits for viscosity and thermal conductivity are also in the same form as that used by the CEA program [1]. The curves are as follows.

$$\log\mu(T) = a_0\log T + \frac{a_1}{T} + \frac{a_2}{T^2} + a_3$$

$$\log k(T) = b_0 \log T + \frac{b_1}{T} + \frac{b_2}{T^2} + b_3$$

2.2 Mixing rules for a collection of thermally perfect gases

The thermodynamic state for a mixture of thermally perfect gases is uniquely defined by two state variables and the mixture composition. The internal energy[1] is computed as a mass fraction weighted sum of individual internal energies,

$$e = \sum_{i=1}^{N} f_i e_i = \sum_{i=1}^{N} f_i \left(h_i - R_i T\right). \tag{2.6}$$

Pressure is computed from Dalton's law of partial pressures,

$$p = \sum_{i=1}^{N} \rho_i R_i T. \tag{2.7}$$

The specific gas constant for the mixture is defined as

$$R = \sum_{i=1}^{N} f_i R_i. \tag{2.8}$$

The calculation of C_p is based on a mass fraction weighted sum of component specific heats,

$$C_p = \sum_{i=1}^{N} f_i C_{p_i}. \tag{2.9}$$

The specific heat at constant volume is then computed as

$$C_v = C_p - R. \tag{2.10}$$

The ratio of specific heats, γ, is given by its definition,

$$\gamma = \frac{C_p}{C_v}. \tag{2.11}$$

The frozen sound speed for the mixture, a, is calculated as

$$a = \sqrt{\gamma R T}. \tag{2.12}$$

During a compressible flow simulation, the values of ρ and e are most readily available from the conserved quantities that are solved for during each time increment. This leads to the specific problem of solving for the thermodynamic state of the gas mixture given ρ, e, and the mixture composition, $\overrightarrow{f}$. However, the formulae previously presented are all explicit in temperature. We solve for temperature using the Newton iteration technique for zero solving,

$$T_{n+1} = T_n - \frac{f_0(T_n)}{f_0'(T_n)}, \tag{2.13}$$

[1] Note that the GasState class uses the symbol u for specific internal energy.

where the zero function, $f_0(T)$, is based on the given internal energy, e, and a guess for internal energy based on temperature,

$$f_0(T) = e_{guess} - e = \sum_{i=1}^{N} f_i \left(h_i - R_i T_{guess}\right) - e. \tag{2.14}$$

Using the fact that $C_{vi} = \frac{de_i}{dt}$, we can conveniently find the derivative function for the Newton technique by computing the mixture C_v,

$$\frac{df_0(T)}{dT} = \sum_{i=1}^{N} f_i \frac{de_i}{dT} = \sum_{i=1}^{N} f_i C_{vi} = C_v. \tag{2.15}$$

The Newton iteration is set to converge when the accuracy of the temperature value is within $\pm 1.0 \times 10^{-6}$ K. Personal experience has shown that this kind of error tolerance is required when temperature is used in a finite-rate chemistry calculation to compute rates of composition change.

The calculation of mixture transport properties is not as straight forward as the thermodynamic properties. A mixing rule is required to compute the mixture viscosity and thermal conductivity. Wilke's mixing rule [3] has been implemented in the work presented here. Specifically, the mixing rules used by Gordon and McBride [4] in the CEA program are used for calculating mixture transport properties in this work; these rules are a variant of Wilke's original formulation [3].

$$\mu_{mix} = \sum_{i=1}^{N} \frac{x_i \mu_i}{x_i + \sum_{\substack{j=1 \\ j \neq i}}^{N} x_j \phi_{ij}} \tag{2.16}$$

and

$$k_{mix} = \sum_{i=1}^{N} \frac{x_i k_i}{x_i + \sum_{\substack{j=1 \\ j \neq i}}^{N} x_j \psi_{ij}} \tag{2.17}$$

where x_i is the mole fraction of species i.

The interaction potentials, ϕ_{ij} and ψ_{ij}, can be calculated a number of ways. Again, the formulae suggested by Gordon and McBride [4] have been used,

$$\phi_{ij} = \frac{1}{4}\left[1 + \left(\frac{\mu_i}{\mu_j}\right)^{1/2} \left(\frac{M_j}{M_i}\right)^{1/4}\right]^2 \left(\frac{2M_j}{M_i + M_j}\right)^{1/2} \tag{2.18}$$

and

$$\psi_{ij} = \phi_{ij}\left[1 + \frac{2.41(M_i - M_j)(M_i - 0.142M_j)}{(M_i + M_j)^2}\right] \tag{2.19}$$

where M_i and M_j refer to the molecular weights of species i and j respectively.

Once the mixture viscosity and thermal conductivity have been computed, it is possible to compute the mixture Prandtl number from its definition

$$Pr = \frac{\mu C_p}{k}. \tag{2.20}$$

CHAPTER 3

Finite-rate chemistry

3.1 Rates of species change due to chemical reaction

By assuming a collection of simple reversible reactions, the chemically reacting system can be represented as,

$$\sum_{i=1}^{N} \alpha_i X_i \rightleftharpoons \sum_{i=1}^{N} \beta_i X_i, \tag{3.1}$$

where α_i and β_i represent the stoichiometric coefficients for the reactants and products respectively. The case of an irreversible reaction is represented by setting the backward rate to zero. For a given reaction j, the rate of concentration change of species i is given as,

$$\left(\frac{d[X_i]}{dt}\right)_j = \nu_i \left\{ k_f \prod_i [X_i]^{\alpha_i} - k_b \prod_i [X_i]^{\beta_i} \right\}, \tag{3.2}$$

where $\nu_i = \beta_i - \alpha_i$. By summation over all reactions, N_r, the total rate of concentration change is,

$$\frac{d[X_i]}{dt} = \sum_{j=1}^{N_r} \left(\frac{d[X_i]}{dt}\right)_j. \tag{3.3}$$

For certain integration schemes it is convenient to have the production and loss rates available as separate quantities. In this case,

$$\frac{d[X_i]}{dt} = q_i - L_i = \sum_{j=1}^{N_r} \dot{\omega}_{app_{i,j}} - \sum_{j=1}^{N_r} \dot{\omega}_{va_{i,j}} \tag{3.4}$$

The calculation of $\dot{\omega}_{app_{i,j}}$ and $\dot{\omega}_{va_{i,j}}$ depends on the value of ν_i in reach reaction j as shown in Table 3.1.

The calculation of the reaction rate coefficients, k_f and k_b, and the solution methods for the ordinary differential equation system of species concentration changes are discussed in the subsequent sections.

Table 3.1: The form of the chemical production and loss terms based on the value of ν_i

	$\nu_i > 0$	$\nu_i < 0$
$\dot{\omega}_{app_i}$	$\nu_i k_f \prod_i [X_i]^{\alpha_i}$	$-\nu_i k_b \prod_i [X_i]^{\beta_i}$
$\dot{\omega}_{va_i}$	$-\nu_i k_b \prod_i [X_i]^{\beta_i}$	$\nu_i k_f \prod_i [X_i]^{\alpha_i}$

3.2 Reaction rate coefficients

The reaction rate coefficients for a reaction can be determined by experiment (often shock tube studies are used) or from theory. In a great number of cases, estimates of the reaction rate from theory can vary by orders of magnitude from experimentally determined values. For this reason, fits to experimental values are most commonly used.

For the single-temperature gas model discussed in this chapter, the forward reaction rate coefficients are calculated using the generalised Arrhenius form,

$$k_f = AT^n \exp\left(\frac{-E_a}{kT}\right) \tag{3.5}$$

where k is the Boltzmann constant and A, n and E_a are constants of the model.

The backward rate coefficient can also be calculated using a modified Arrhenius form,

$$k_b = AT^n \exp\left(\frac{-E_a}{kT}\right) \tag{3.6}$$

or it can be calculated by first calculating the equilibrium constant for the reaction,

$$k_b = \frac{k_f}{K_c}. \tag{3.7}$$

If the backward rate coefficient is calculated from the equilibrium constant, then a method of calculation of the equilibrium constant is required. The equilibrium constant for a specific reaction can be calculated from curve fits or, as is done in this work, using the principles of thermodynamics. The equilibrium constant based on concentration is related to the equilibrium constant based on pressure by,

$$K_c = K_p \left(\frac{p_{atm}}{\mathcal{R}T}\right)^{\nu} \tag{3.8}$$

where p_{atm} is atmospheric pressure in Pascals, $\mathcal{R}$ is the universal gas constant, $\nu = \sum_i^{N_S} \nu_i$ and

$$K_p = \exp\left(\frac{-\Delta G}{\mathcal{R}T}\right). \tag{3.9}$$

The derivation of the formula for K_p, the equilibrium constant based on partial pressures, can be found in any introductory text on classical thermodynamics which covers chemical equilibrium. The differential Gibbs function for the reaction, ΔG, is calculated using

$$\Delta G = \sum_i^{N_s} \nu_i G_i \tag{3.10}$$

where each G_i is computed from the definition of Gibbs free energy,

$$G_i(T) = H_i(T) - T \times S_i(T) \tag{3.11}$$

and G_i is in units of J/mol. H_i and S_i can be computed in the appropriate units by using the CEA polynomials and multiplying by $\mathcal{R}T$ and $\mathcal{R}$ respectively.

Some caution should be exercised in the selection and use of reaction rates for a specific flow problem. In many cases, a set of reaction rates may only be "tuned" for a specific problem domain. This problem of "tuned" sets of reaction rates and an explanation for why it arises is described by Oran and Boris (p. 38 of Ref. [5]):

> A problem that often arises in chemical reactions is that there are fundamental inconsistencies in a measured reaction rate. For example, there may be experimental measurements of both the forward and reverse rate constants, k_f and k_r. Nonetheless, when either is combined with the equilibrium coefficient for that reaction, the other is not produced. This appears to represent a violation of equilibrium thermodynamics. The explanation is usually that k_f and k_r have been measured at rather different temperatures or pressures, and so there are inconsistencies when they are extrapolated outside the regime of validity of the experiments.

3.3 Solving the chemical kinetic ordinary differential equation

The system represented in Equation 3.3 is a system of ordinary differential equations (ODEs) which can be solved by an appropriate method. For certain chemical systems, the governing ODEs form a stiff system due to rates of change varying by orders of magnitude for certain species. For these systems, special methods for stiff ODEs are required. At present, there are two ODE methods provided in the implementation: one aimed at non-stiff systems, and the other for stiff systems.

1. Runge-Kutta-Fehlberg method (efficient for non-stiff systems)

2. alpha-QSS method (specialised for stiff chemitry systems)

The fourth-order Runge-Kutta method uses a fifth-order error estimate as a means for controlling the timestep used for integration as proposed by Fehlberg [6]. This is particularly efficient for non-stiff systems.

alpha-QSS method The alpha-QSS (quasi-steady-state) method was proposed in Mott's thesis [7]. The thesis provides good detail on the method development, however, a later journal article [8] on the method provides typographical corrections to the update equations. Our implementation uses the equation from this journal article. It is an ODE solver aimed specifically at the problem of stiffness in chemical systems. This ODE solver makes use of the forward and backwards rates of concentration change as calculated by Equation 3.4. This is a predictor-corrector type scheme in which the corrector is iterated upon until a desired convergence is achieved. The

predictor and corrector are,

$$[X_i]^1 = [X_i]^0 + \frac{\Delta t q_i^0}{1 + \alpha_i^0 \Delta t L_i^0} \tag{3.12}$$

$$[X_i]^{n+1} = [X_i]^0 + \frac{\Delta t \left(\bar{q}_i - [X_i]^0 \bar{L}_i\right)}{1 + \bar{\alpha}_i \Delta t \bar{L}_i}. \tag{3.13}$$

In the above equations,

$$\bar{L}_i = \frac{1}{2}\left(L_i^0 + L_i^n\right) \tag{3.14}$$

and

$$\bar{q}_i = \bar{\alpha}_i q_i^n + (1 - \bar{\alpha}_i) Q_i^0. \tag{3.15}$$

The key to the scheme is calculating α correctly. This α parameter controls how the update works on a given species integration. Note that α is defined as

$$\alpha(L\Delta t) \equiv \frac{1 - \left(1 - e^{-L\Delta t}\right)/(L\Delta t)}{1 - e^{-L\Delta t}}. \tag{3.16}$$

Using Pade's approximation,

$$e^x \approx \frac{360 + 120x + 12x^2}{360 - 240x + 72x^2 - 12x^3 + x^4} \tag{3.17}$$

it is possible to write a form of the expression for α which is more amenable to computation as the expensive exponential function evaluation is avoided. The approximation for α becomes,

$$\alpha(L\Delta t) \approx \frac{180r^3 + 60r^2 + 11r + 1}{360r^3 + 60r^2 + 12r + 1} \tag{3.18}$$

where $r \equiv 1/(L\Delta t)$.

The corrector step is iterated until convergence is achieved or a maximum number of corrector steps have been taken. The convergence criterion, given by Mott [7], is based on all species satisying:

$$\left|\left|[X]^C - [X]^P\right|\right| < \varepsilon_1 [X]^C \tag{3.19}$$

Referring to the notation used earlier, $[X]^C = [X]^{n+1}$ and $[X]^P = [X]^1$. Qureshi and Prosser [9] suggest an addition of a δ term to Equation 3.19 to prevent an excessively restrictive criterion when species first appear from zero concentration. This change in critertion also has a follow-on effect in prevernting some needlessly small timesteps. Qureshi and Prosser suggest:

$$\left|\left|[X]^C - [X]^P\right|\right| < \varepsilon_1 \left([X]^C + \delta\right) \tag{3.20}$$

On completion of a step, either successful or unsuccesful, Mott [7] provides a suggestion for the new step size. That algorithm, including the δ addition of Qureshi and Prosser [9], is:

$$\sigma = \max\left(\frac{\left|\left|[X]^C - [X]^P\right|\right|}{\varepsilon_2\left([X]^C + \delta\right)}\right) \tag{3.21}$$

$$(\Delta t)_{\text{new}} = (\Delta t)_{\text{old}}\left(\frac{1}{\sqrt{\sigma^*}} + 0.005\right) \tag{3.22}$$

Note that $\sqrt{\sigma}^*$ is computed as three steps of a Newton's method using σ as the starting value. This is the suggestion of Mott [7] as an efficiency measure in the implementation.

3.4 Coupling chemistry effects to the flow solver

Some details about the coupling of the chemistry effects to the gas dynamics simulation are provided here. In an unsteady, time-accurate flow simulation, the allowable timestep is constrained by the Courant-Friedrichs-Lewy (CFL) criterion. In a viscous compressible flow, the CFL criterion allows one to select an appropriate timestep and limit the propagation of flow information to distances less than one cell-width. The speed at which flow information propagates is a function of inviscid wave speeds and viscous effects.

When the effects of finite-rate chemistry are 'split' from the flow simulation, the chemical update is solved in a separate step in which the flow is held frozen. (In fact, in true timestep-splitting, *all* other contributing physics is frozen during the chemistry update.) Thus the chemistry problem is to find the updated species composition at the end of the flow timestep.

It may be, and is quite likely, that the flow timestep is not an appropriate timestep to solve the chemical kinetic ODE problem. When the timestep for the chemistry problem is smaller than the flow timestep, the chemistry problem is subcycled a number of times until the total elapsed time equals that of the flow timestep. It is common to have simulations where the chemistry timestep is 100–1000 times smaller than the flow timestep, that is, 100-1000 subcycles are required to solve the chemistry problem. When the timstep for the chemistry problem is larger than the flow timestep, it is simply set to the value of the flow timestep.

During the simulation process, the chemistry timestep is tracked for each finite-volume cell in the simulation. Although the flow 'moves on' in subsequent timesteps, if the change of flow conditions is not large, then the previous chemistry timestep will be a good estimate to begin the new chemistry problem in the subsequent timestep. An exceptional case is when a shock passes through the cell: the change of flow conditions does become large. In this instance, the old chemistry step is disregarded and a new step is selected. The selection procedure for a new step is discussed in the next paragraph. When using either the Runge-Kutta-Fehlberg or the alpha-QSS methods, an estimate of the new chemistry timestep is provided as part of the ODE update routine.

So, during a simulation, the old chemistry step at one iteration is used to begin the new chemistry problem in the next iteration. What is needed is a means to select the chemistry step on the initial iteration, or whenever the old suggestion is not reasonable (as in the case of a shock passing through the cell). In this work, the initial step for the chemistry problem is selected based on the suggestion by Young and Boris [10],

$$dt_{\text{chem}} = \epsilon_1 \min\left(\frac{[X_i](0)}{[\dot{X}_i](0)}\right) \tag{3.23}$$

where ϵ_1 is taken as 1.0×10^{-3} in this work, and the expression is evaulated at the initial values for the chemistry subproblem. Young and Boris [10] suggest that ϵ_1 be

scaled from the convergence criteria. We have found that the fixed value is adequate for the problems of interest to our research group.

CHAPTER 4

The reactions-scheme file

The chemical reactions which may take place in a reacting flow simulation are described in a Lua input file. This input file, prepared by the user, is pre-processed with the `prep-chem` program to produce a detailed chemistry file for subsequent use in the main simulation code or in a custom script. As the input file is Lua-based, the user has access to the full extent of the Lua scripting language when preparing her files. Do not be concerned if you do not know the Lua syntax; the instructions and examples given here should be ample to get you started building reaction schemes.[1]

Let's proceed by looking at an example input file and discussing the keywords and syntax. Listed here is an input file which describes the simple thermal dissociation of nitrogen. There are only two participating species, N_2 and N, and only two reactions.

```
Reaction{
   'N2 + N2 <=> N + N + N2',
   fr={'Arrhenius', A=7.0e21,  n=-1.6, C=113200.0},
   br={'Arrhenius', A=1.09e16, n=-0.5, C=0.0}
}

Reaction{
   'N2 + N <=> N + N + N',
   fr={'Arrhenius', A=3.0e22,  n=-1.6, C=113200.0},
   br={'Arrhenius', A=2.32e21, n=-1.5, C=0.0}
}
```

The first reaction is the dissociation of N_2 by collision with other N_2 molecules. The forward reaction rate coefficient is computed with a generalised Arrhenius model, and the parameters for that model are specified. Similarly, the backward reaction rate coefficient is computed using the Arrhenius expression.

[1] If you are worried about needing to "learn Lua" just to get started, then don't be. First, you may just look at this as an input format for the chemistry, and forget that it has anything to do with Lua altogether. Second, Lua was designed with non-programmers in mind and so it uses a simple syntax, specifically so that those non-programmers could quickly use Lua as a configuration language.

More generally, each reaction is specified within a `Reaction` table.[2] The table is delimited by the opening and closing braces ({ }). The first entry in the table is always a string. That string is the chemical equation for the reaction. The remaining items in the table are denoted by key-value pairs (of the form `key=val`), and may appear in any order. Each item in the table is separated by a comma.[3] This example file contained two `Reaction` tables, hence two reactions are treated in the reaction scheme.

Some final notes before discussing the input file in further depth. There is no explicit mention of the participating species in the reaction file. The participating species are taken from the species that are present in the gas model file for the same flow simulation. In other words, if you list species in the reaction scheme that are not present in the gas model, then you will get an error message.

4.1 Overview of input file format

By leveraging Lua as the input data description language, the input file is almost self-describing, in my opinion. This provides an excellent record of what modelling was used when you performed a simulation. A valid reaction input file will conform to the following rules.

1. Any legal Lua code is acceptable, but you must not rename the following the pre-defined functions:
   - `Reaction`
   - `removeAllReactionsWithLabel`
   - `removeReaction`
   - `selectOnlyReactionsWithLabel`
   - `selectOnlyReactions`
2. Reactions are declared in `Reaction` tables.
3. Comments in the file begin when two dashes (`--`) are encountered and proceed to the end of the line. (This is a repetition of Item 1 in that comments are legal Lua code.)

As the reactions are listed in the file, they are numbered internally beginning from 1. In some cases, it is convenient to list all reactions in a scheme but then only use some of the reactions. This is quite common if you wish to use a reduced mechanism or if you believe that one of the species is inert at your flow conditions of interest, and so would want to remove all reactions involving the transformation of that species. Two convenience functions are provided so that you do not have to hack into your input file to remove the unwanted reactions:

- `removeAllReactionsWithLabel`

[2] If you are well versed in Lua, you will recognise that `Reaction` is a function call with one argument, a table.

[3] Lua also permits the use of semi-colons instead of commas to delimit table entries.

- removeReaction

Both functions will take a single item or an array of items. An array is a special form of Lua table which is bracketed with braces ({ }). The first function accepts strings which correspond to the labels of reactions. The labelling of reactions is explained in the next section. The second function accepts integers which correspond to the internal numbering. The convenience functions must be called *after* the declaration of the associated reactions. Typically, the user would place the calls to these functions at the end of his input file. Two examples follow.

```
removeAllReactionsWithLabel({'r3', 'r5'})
```

This call would remove the reactions labelled 'r3' and 'r5'
from the list of participating reactions.

```
removeReaction(13)
```

In this call, the 13th listed reaction is removed from the list (because we all know that 13 is unlucky, right?)[4]

Similarly, there are two complementary convenience functions that allow for the selection of only certain reactions from the full set:

- selectOnlyReactionsWithLabel
- selectOnlyReactions

They work in reverse to the remove functions: these functions will only select those reactions listed in their arguments for inclusion in the chemistry scheme.

Note, it is not advisable to mix and match the use of the remove and select functions in the one reaction script. The behaviour is untested. Now on to the details of the Reaction table.

4.2 Details of the Reaction table

The Reaction table accepts a number of items — some are mandatory, most are not. The full list of items is shown here, and each item is described below.

```
Reaction{
   'equation string',
   fr={...},
   br={...},
   ec='model name',
   efficiencies={...},
   label='r1'
}
```

[4] Actually, unlike the Americans and their buildings, you don't get rid of 13 that easily. If you have more than 13 reactions, the higher numbered reactions will shuffle up one spot so that the numbering remains continuous from 1. This all happens internally.

'equation string' *(mandatory)*
As mentioned earlier, this string must appear first in the table and has no key associated with it. This string represents the reaction equation. As an example, dissociation of nitrogen may be written as

```
'N2 + N2 <=> N + N + N2'
```

If the reaction involves a collision with a general third body, then this is strictly denoted as species `'M'`. For example, the formation of hydroperoxyl from oxygen and monatomic hydrogen requires the presence of a third body. This reaction is written as

```
'H + O2 + M <=> HO2 + M'.
```

The reactants and products are delimited by direction arrows. The use of <=> indicates that the reaction proceeds in both directions, while => will mean that the reaction proceeds in the forward direction only (no backward rate of conversion will be computed).

fr *(optional, if `br` supplied)*
The `fr` key is used to specify the forward reaction rate coefficient and expects a table value. The format of the table is a string naming the model followed by key-value pairs giving the parameters for the model. The currently implemented reaction rate coefficient models are listed at the end of this section, along with their input format.

br *(optional, if `fr` supplied)*
The `br` key is used to specify the backward reaction rate coefficient. It is used in the same manner as the forward rate key (`fr`).

ec *(optional)*
The `ec` key is used to specify the model for computing the equilibrium constant. It accepts a string naming the model. Currently, there is only one model implemented, `'from thermo'`, which calculates the equilibrium constant based on thermodynamic principles. For reversible reactions, if only one of `fr` or `br` is specified, then the use of the equilibrium constant is assumed and does not need to be declared.

efficiencies *(optional)*
If declaring a third body reaction, all species in the mixture are assumed to react with an efficiency of 1.0. The `efficiencies` key accepts a list of *exceptions* to that assumption of a value of 1.0. The list contains the key-value pairs of the type `species=efficiency_value`. For example, to denote that N_2 has a 6-fold efficiency value and O_2 a value of 3.5, the list would be:

```
efficiencies={N2=6.0, O2=3.5}
```

Remember that all species are assumed to have a value of 1.0 unless otherwise noted in the list. If you have a species that does *not* participate as a third body,

then be sure to set its efficiency value to 0.0 (e.g. `H=0.0`). Also, note how you can use the Lua table constructor to help in the case of ions and the electron: simply enclose these in brackets when putting them in the `efficiencies` table. An example is:

```
efficiencies={['O2+']=9.0, ['e-']=0.0}
```

`label` *(optional)*
The `label` accepts a string allowing the user to give the reaction a name. This is useful if one wishes to later remove certain reactions based on their labels using the `remove_reactions_by_label` convenience function.

Note that if you specify all three of `fr`, `br` and `ec`, you have overspecified the modelling of reaction rate coefficients. In this case, no error is given. Instead, the `ec` model is ignored.

Input for rate coefficient models

The various rate coefficient models are specified for the forwards rate coefficient, the backwards rate coefficient, or both indepdendently. The specification is of the form:
`fr={'ModelName', param1=..., param2=...}`
The available rate coefficient models are:

- generalised Arrhenius
- generalised Arrhenius with pre-exponential as $log(A)$
- pressure-dependent rates in two forms:
  1. Lindemann-Hinshelword form
  2. Troe form

Below we describe the available models, how they are selected and their input parameters.

Generalised Arrhenius
The generalised Arrhenius rate coefficient is computed as

$$k = AT^n \exp(-C/T).$$

This is set in the input file as
`fr={'Arrhenius', A=..., n=..., C=...}`
where:

`A` is the pre-exponential coefficient given in 'cgs' units (because they are most common in the chemistry reaction rate literature).

`n` is the non-dimensional power for T.

`C` is the activation temperature in Kelvin.

Note that the units of the pre-exponential coefficients will depend on the number of colliders, in each direction, for a particular reaction. Considering the nitrogen dissociation reaction

```
'N2 + N2 <=> N + N + N2'
```

we can specialize the rate equation (3.2) to give the rate of production of nitrogen molecules due to this reaction as

$$\frac{d[N_2]}{dt} = -1\left\{k_f[N_2]^2 - k_b[N]^2[N_2]\right\}.$$

With the units of concentration being mole·cm^{-3}, the units of k_f and its associated pre-exponential coefficient, `A`, will need to be mole^{-1}·cm^3·s^{-1} while the units for the backward rate will need to be mole^{-2}·cm^6·s^{-1}.

Generalised Arrhenius with pre-exponential as $log(A)$
The generalised Arrhenius rate coefficient is computed as

$$k = \exp(\log(A) + BT - C/T).$$

This is set in the input file as
`fr={'Arrhenius-logA', logA=..., B=..., C=...}`.

Lindemann-Hinshelwood form, pressure-dependent rate coefficient
The Lindemann-Hinshelwood pressure dependent rate coefficient is a blending of zero-pressure and infinite-pressure limits:

$$k = \frac{k_\infty k_0[M]}{k_\infty + k_0[M]}$$

A discussion on the theory of Lindemann-Hinshelwood pressure-dependent rates is given in Chapter 3 of O'Flaherty's thesis [11]. To set this for a rate coefficient in the input file, use:

```
fr={'pressure dependent',
    kInf={A=..., n=..., C=...},
    k0={A=..., n=..., C=...}}
```

where:

`kInf` is used to set Arrhenius parameters at the infinite pressure limit.

`k0` is used to set Arrhenius parameters at the zero pressure limit.

Troe form, pressure-dependent rate coefficient
The Troe form for rate coefficient is also a pressure dependent rate. It is evaluated as

$$k = \frac{k_\infty k_0[M]}{k_\infty + k_0[M]}F$$

with

$$\log F = \left[1 + \left(\frac{\log P_r + c}{n - d(\log P_r + c)}\right)^2\right]^{-1} \log F_{cent}$$

where

$$\begin{aligned} P_r &= \frac{k0[M]}{k_\infty} \\ c &= -0.4 - 0.67 \log F_{cent} \\ n &= 0.75 - 1.27 \log F_{cent} \\ d &= 0.14 \quad \text{and} \\ F_{cent} &= (1-a)\exp\left(-\frac{T}{T^{***}}\right) + a\exp\left(-\frac{T}{T^{*}}\right) + \exp\left(-\frac{T^{**}}{T}\right) \end{aligned}$$

To set this for a rate coefficient in the input file, use:

```
fr={'pressure dependent',
    kInf={A=..., n=..., C=...},
    k0={A=..., n=..., C=...},
    Troe={a=..., T1=..., T2=..., T3=...}}
```

where:

kInf is used to set Arrhenius parameters at the infinite pressure limit.

k0 is used to set Arrhenius parameters at the zero pressure limit.

Troe is used to set the parameters that appear in the expressions above, with `T1` = T^*, `T2` = T^{**}, and `T3` = T^{***}.

Note: Both the Troe and Lindemann-Hinshelwood forms are selected with the name `'pressure dependent'`. The presence or otherwise of the `Troe` parameter set is used to distinguish between to the two forms.

4.3 Extra control of the chemistry scheme

There are a number of details to do with solving the finite-rate chemistry problem that are set by default for the user. However, all of these parameters may be controlled by the user by setting values in the input file. The user can set these values in a `Config` table. The full list of controllable options is described in what follows. To get started, an example of setting some temperature limits on the rate coefficient evaluations and selecting the Runge-Kutta-Fehlberg ODE integrator is shown here:

```
Config{
   tempLimits={lower=300.0, upper=10000.0},
   odeStep={method="rkf"}
}
```

tightTempCoupling
: By default, tight coupling between the temperature update and the species update is *not* employed. What this means is that in normal operation, the rate constants are evaluated once (the constants, not the rate) and only once at the start of a chemistry update. This is an efficiency measure that is employed quite commonly in finite-rate chemistry modules because the rate constant evaluation is

relatively expensive. It also works well as an approximation in most situations because the temperature in a given computational cell does not change much over the timestep of interest.

However, there can be cases when the chemical composition change is quite rapid over the chosen timestep, and correspondingly the temperature changes a lot. In these cases, it is more robust and accurate to re-evaluate the temperature and the rate constants on every subcycle step in the chemistry update. To set this option active, set this parameter to true:

```
tightTempCoupling = true
```

To summarise, the default is to treat temperature as constant over the chemistry update. This is good for most situations and is a very efficient way to proceed with the calculation. For very energetic chemistry situations, it is more robust to select tight temperature coupling. This will be more accurate and robust, at the cost of being more computationally expensive.

tempLimits

This setting is used to control the temperature limits at which reaction rate coefficients are evaluated. The user provides `lower` and `upper` values for the temperature limits. In the example here, the lower temperature limit is set to 300 K and the upper limit is set to 50 000 K. These happen to be the default values, if not set by the user.

```
tempLimits={lower=300.0, upper-50000.0}
```

These values are used to control the temperature limits at which reaction rate coefficients are evaluated. When the local temperature exceeds the limits (on either side), the rate is simply evaluated at the temperature corresponding to the exceeded limit. As pseudo-code:

```
if T > T_upper
  then T = T_upper
if T < T_lower
  then T = T_lower
eval_rate_coeff(T)
```

maxSubcycles

This setting controls the maximum allowable chemistry subcycles when coupled to a flow solver. This is an integer value and the default is 10,000. This setting is to ensure that the chemistry update remains sensible and does not get caught making almost no progress with extremely small chemistry steps. If this does happen, it is usually a signal that there are bigger issues numerical model setup, or what is being asked of the numerical model.

```
maxSubcycles=10000
```

maxAttempts

Occasionally, a chemistry step will fail. This might be because the step size has grown a little beyond what is numerically stable. This is called a failed attempt and the code will adjust the step size an re-attempt the step. There is a control on how many attempts are made to recover a failed step. This is an integer value and set to 4 by default. The user can control this value. If a value much larger than 4 is required, this is usually a hint that there are larger issues with the simulation.

```
maxAttempts=4
```

odeStep

This setting allows the user to select the method type for the chemical ODE integration. The user passes a table. At a minimum, the table contains a method name. For example:

```
odeStep = {method='alpha-qss'}
```

Optionally, the user might supply some more information specific to the chosen method that controls aspects of the numerics. This is why a table is used: so that the extra information is easy to include. Here is an example with some finer control of the α-QSS method:

```
odeStep = {method='alpha-qss', eps1=1.0e-4, maxIters=5}
```

Presently, the choice of ODE methods is `rkf` and `alpha-qss`. These methods and a description of the associated parameters are given directly below.

4.3.1 Selection and extra control of the ODE methods

As shown above, the selection of an ODE method for integrating the chemistry problem is performed by setting the `odeStep` parameter. Here we describe the finer details of control that are available for each of the methods: `rkf` and `alpha-qss`.

Runge-Kutta-Felhberg method

The Runge-Kutta-Fehlberg has one control parameter available to the user: an error tolerance used when estimating error on each step, `errTol`. The error tolerance is used in the algorithm to estimate error as suggested by Press et al. [12]. In the authors' experience, the default value of 1×10^{-3} has worked well for a variety of non-stiff chemistry systems associated with hypersonic reacting flows over blunt bodies.

The RKF integrator is selected and configured as:

```
odeStep = {method='rkf', errTol=1.0e-3}
```

Mott's α-QSS method

The α-QSS method has four configurable parameters: `eps1`, `eps2`, `delta` and `maxIters`. The first three parameters align with the update and step size estimator equations presented in Section 3.3. The `maxIters` parameter is used to control iterations of the *corrector* step are performed.

An example of selecting and configuring the α-QSS method is shown here (with default values if left unspecified):

```
odeStep = {method='alpha-qss', eps1=0.001, eps2=5.0e-4,
           delta=1.0e-10, maxIters=10}
```

CHAPTER 5

Examples of Use

Although the reacting gas models are developed for use in the Eilmer flow solver, their services are available via a simple applications programming interface. On top of the basic `GasModel` and `GasState` classes discussed in the gas-model user's guide [13], the `ThermochemicalReactor` class provides the `update_state` method. This single method defines the application programming interface (API) for the reacting gas models. The evolution of a charge of perfectly-stirred reactants in a thermally-isolated, fixed-volume reactor is the conceptual model behind this update function.

So, given a particular gas model with more than one chemical species, we may construct a `GasState` object and then pass that to the `update_state` method to allow the chemical reactions to change the species fractions over a specified time interval. During this interval, volume, density and internal energy are fixed. Species mass or mole fractions and other thermodynamic quantities such as pressure ant temperature may change.

The following sections in this chapter provide examples that might be interesting in themselves or they might provide good starting points for building your own analysis tools. The first (and simplest) example shows direct use of the ThermochemicalReactor class to calculate the evolution of an isolated blob of gas over a short time period. This example is minimal but coded in all three of the supported scripting languages (Lua, Python and Ruby). The second example is a small but typical application that evaluates a selection of reaction schemes for hydrogen combustion in air. The final example makes use of the reacting gas model within an Eilmer flow simulation. There, the ThermochemicalReactor's `update_state` method is called by the flow solver code, rather than by the user's input script.

5.1 Fixed-volume nitrogen reactor

Continuing with our minimal reacting gas example of dissociating nitrogen, we will use the API to build a simulation of a fixed-volume reactor, initially containing nitrogen molecules and atoms at high enough temperature for reactions to occur over the reasonable time of 300 microseconds. At least, that's a reasonable time for hypersonics people.

In preparation for running the scripts, we need a detailed-gas-model file and detailed chemistry file set up for use by the simulation classes. The input file (called

nitrogen-2sp.inp) for our nitrogen gas model specifies that we want to use the thermally-perfect gas model and then lists nitrogen molecules and nitrogen atoms as the species of interest.

```
model = 'thermally perfect gas'
species = {'N2', 'N'}
```

Running this input file through the prep-gas program will generate a detailed-gas-model file (nitrogen-2sp.lua) that can be used by the simulation functions.

```
$ prep-gas nitrogen-2sp.inp nitrogen-2sp.lua
```

The chemistry input file differs from the one shown on page 13 in that we omit the backward rate expressions and let the solver provide them from the equilibrium constant.

```
-- nitrogen-2sp-2r-Keq.lua
--
-- This chemical kinetic system provides
-- a simple nitrogen dissociation mechanism.
--
-- Author: Rowan J. Gollan
-- Date: 13-Mar-2009 (Friday the 13th)
-- Place: NIA, Hampton, Virginia, USA
--
-- History:
--   24-Mar-2009 - reduced file to minimum input
--   11-Aug-2015 - updated for dlang module

Reaction{
   'N2 + N2 <=> N + N + N2',
   fr={'Arrhenius', A=7.0e21,  n=-1.6, C=113200.0},
}

Reaction{
   'N2 + N <=> N + N + N',
   fr={'Arrhenius', A=3.0e22,  n=-1.6, C=113200.0},
}
```

This input file can be used to generate a detailed chemistry file with the command:

```
$ prep-chem nitrogen-2sp.lua nitrogen-2sp-2r-Keq.lua chem.lua
```

The detailed chemistry file, chem.lua, may now be used when constructing a reactor object in the simulation script.

5.1.1 Lua programming interface

For the Lua API, we will access the reacting-gas model functionality via the gas-calc program. This Lua programming interface reflects closely the D-language interface that is used within the Eilmer flow simulation program.

The following script is passed to the gas-calc program on the command line:

```
$ gas-calc fvreactor.lua
```

On line 5, we first construct a gas model consisting of our two species for reacting nitrogen: nitrogen molecules and nitrogen atoms. We then construct a ThermochemicalReactor with the call to `ChemistryUpdate:new` (on line 6), passing the nitrogen gas model object and the name of the detailed chemistry file.

```
1 -- fvreactor.lua
2 -- A simple fixed-volume reactor.
3 -- PJ & RJG 2018-04-21.
4
5 gm = GasModel:new{"nitrogen-2sp.lua"}
6 chemUpdate = ChemistryUpdate:new{gasmodel=gm, filename="chem.lua"}
7
8 gs = GasState:new{gm}
9 gs.p = 1.0e5 -- Pa
10 gs.T = 4000.0 -- degree K
11 molef = {N2=2/3, N=1/3}
12 gs.massf = gm:molef2massf(molef)
13 gm:updateThermoFromPT(gs)
14 conc = gm:massf2conc(gs)
15
16 tFinal = 300.0e-6 -- s
17 t = 0.0
18 dt = 1.0e-6
19 dtSuggest = 1.0e-11
20 print("# Start integration")
21 f = assert(io.open("fvreactor.data", 'w'))
22 f:write('# 1:t(s)  2:T(K)  3:p(Pa)  4:massf_N2  5:massf_N  6:conc_N2  7:conc_N\n')
23 f:write(string.format("%10.3e %10.3f %10.3e %20.12e %20.12e %20.12e %20.12e\n",
24                       t, gs.T, gs.p, gs.massf.N2, gs.massf.N, conc.N2, conc.N))
25 while t <= tFinal do
26    dtSuggest = chemUpdate:updateState(gs, dt, dtSuggest, gm)
27    t = t + dt
28    -- dt = dtSuggest -- uncomment this to get quicker stepping
29    gm:updateThermoFromRHOE(gs)
30    conc = gm:massf2conc(gs)
31    f:write(string.format("%10.3e %10.3f %10.3e %20.12e %20.12e %20.12e %20.12e\n",
32                          t, gs.T, gs.p, gs.massf.N2, gs.massf.N, conc.N2, conc.N))
33 end
34 f:close()
35 print("# Done.")
```

With the gas model in hand (as variable `gm`), we can construct a gas state object (line 8) and proceed to set its thermodynamic values (on lines 9 through 13). The species fractions are specified as a table of numbers, with species names being provided as the keys for those values. Only the species with non-zero fractions need appear to be specified, however, those fractions need to sum to a total of 1.0. It is often convenient to work in mole fractions or concentrations for the species but we must set the mass fractions in the gas state. To do this, there are a number of convenience functions to make these conversions easy, as seen on lines 12 and 14. Note that, in this example, the gas starts with a significant dissociation fraction and a high enough temperature for reactions to proceed quickly.

The calculation of the gas state evolution proceeds in a number of discrete time steps (of duration `dt`) from time zero to `tFinal`. This is the loop starting at line 25, with the actual update function being called on line 26. The arguments given to the update function are:

1. `gs`, the gas state object, whose thermodynamic values will be updated,
2. `dt`, the time interval over which the reactions will occur,
3. `dtSuggest`, a suggested time step size for the internal iterations of the thermochemical reactor, and
4. `gm`, the gas model object

On completing its work, the update function returns a new value of `dtSuggest`, which we may retain for subsequent use. Depending on the rates of reactions and the internal stepping scheme, this value may be significantly different to the value that we initially provided to the update function. After the thermochemical reactor has updated the species mass fractions held by the gas state, we need to update the other thermodynamic properties on line 29.

Some of the results of the simulation are plotted in Figure 5.1. As might be expected, some of the nitrogen atoms recombine into nitrogen molecules, raising the static temperature and pressure as the reactions proceed. Eventually, the reactor approaches an equilibrium state. The final state in this transient simulation, which has pressure $p = 145.5\,\text{kPa}$, temperature $T = 6177.4\,\text{K}$, and mass fractions $massf_{N2} = 0.86928$, $massf_N = 0.13072$, can be checked against the expected equilibrium state, as computed by CEA2. (See the Section 5.1.4.)

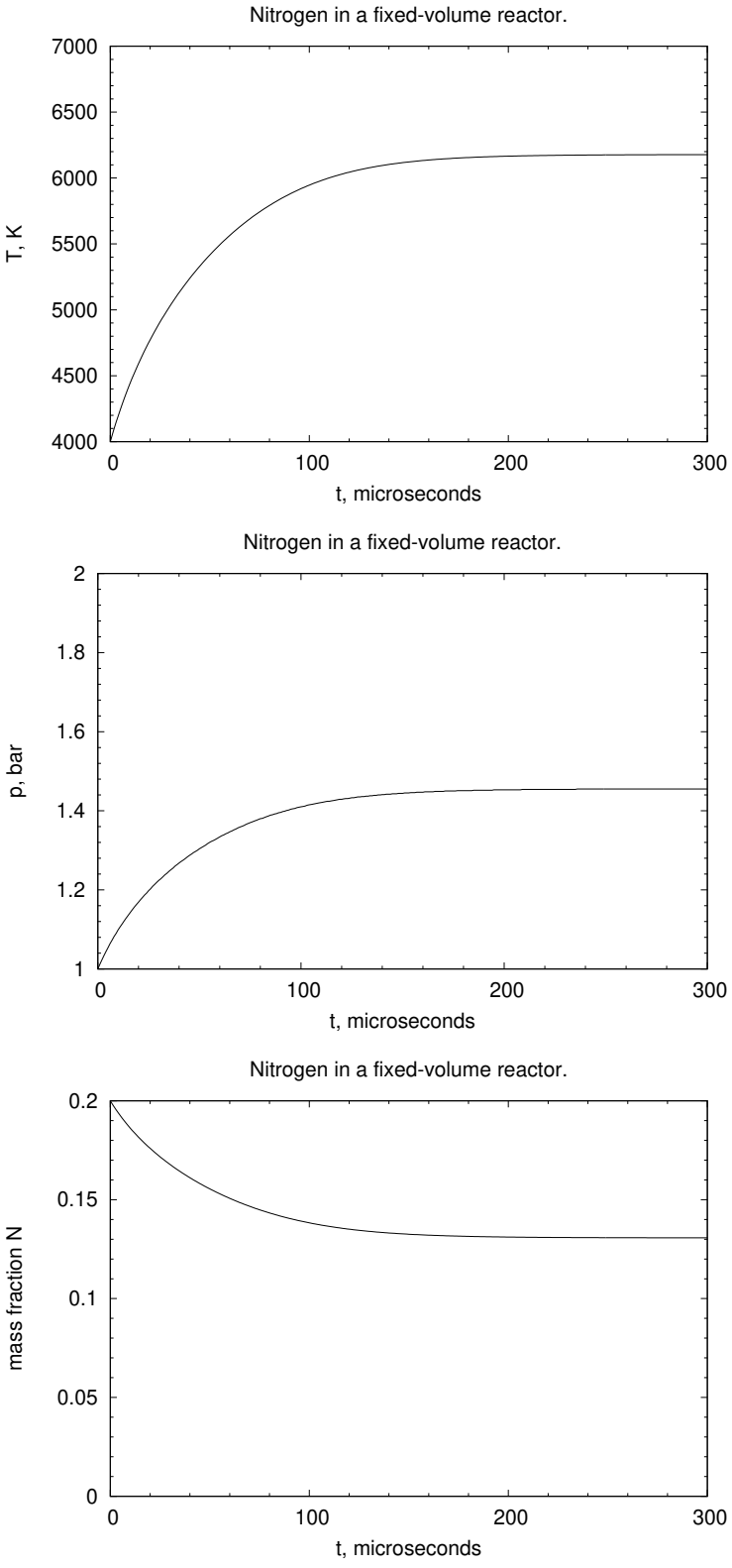

Figure 5.1: Evolution of temperature, pressure and atom mass fraction in the isolated reactor.

5.1.2 Python programming interface

A similar programming interface is available from Python3 scripting language. The Eilmer software package provides the D-language gas module as a loadable library, together with Python and Ruby wrappers that each access this loadable library via CFFI.

The following script is a re-implementation of the fixed-volume reactor calculation, this time in the Python3 programming language. We start by loading this shared library into the Python interpreter, via the import statement on line 12.

```
1 # fvreactor.py
2 # A simple fixed-volume reactor.
3 # PJ & RJG 2019-11-25
4 #
5 # To prepare:
6 #   $ prep-gas nitrogen-2sp.inp nitrogen-2sp.lua
7 #   $ prep-chem nitrogen-2sp.lua nitrogen-2sp-2r.lua chem.lua
8 #
9 # To run:
10 #   $ python3 fvreactor.py
11
12 from eilmer.gas import GasModel, GasState, ThermochemicalReactor
13
14 gm = GasModel("nitrogen-2sp.lua")
15 reactor = ThermochemicalReactor(gm, "chem.lua")
16
17 gs = GasState(gm)
18 gs.p = 1.0e5 # Pa
19 gs.T = 4000.0 # degree K
20 gs.molef = {'N2':2/3, 'N':1/3}
21 gs.update_thermo_from_pT()
22
23 tFinal = 300.0e-6 # s
24 t = 0.0
25 dt = 1.0e-6
26 dtSuggest = 1.0e-11
27 print("# Start integration")
28 f = open("fvreactor.data", 'w')
29 f.write('# 1:t(s)  2:T(K)  3:p(Pa)  4:massf_N2  5:massf_N  6:conc_N2  7:conc_N\n')
30 f.write("%10.3e %10.3f %10.3e %20.12e %20.12e %20.12e %20.12e\n" %
31         (t, gs.T, gs.p, gs.massf[0], gs.massf[1], gs.conc[0], gs.conc[1]))
32 while t <= tFinal:
33     dtSuggest = reactor.update_state(gs, dt, dtSuggest)
34     t = t + dt
35     # dt = dtSuggest # uncomment this to get quicker stepping
36     gs.update_thermo_from_rhou()
37     f.write("%10.3e %10.3f %10.3e %20.12e %20.12e %20.12e %20.12e\n" %
38             (t, gs.T, gs.p, gs.massf[0], gs.massf[1], gs.conc[0], gs.conc[1]))
39 f.close()
40 print("# Done.")
```

Although Python programming interface is almost the same as for the Lua scripting language, there are a few extra conveniences. On line 20, we specify the mole fractions directly, without having to convert to mass fractions first. Concentrations are

also directly available from the gas state, as shown on line 38. Finally, the gas-state object retains a reference to the gas model that was used for its construction (on line 17). The little extra convenience here is that calls to update the thermo properties (on lines 21 and 36) do not need to receive the gas model reference explicitly.

5.1.3 Ruby programming interface

Whatever you do in Python, you can do similarly in Ruby. There are a few language differences, but the Ruby script is very similar to the Python script. The Ruby programming interface to the reacting-gas model may be used in the automated test scripts in the Eilmer code repository.

```
1 # fvreactor.rb
2 # A simple fixed-volume reactor.
3 # PJ & RJG 2019-11-27
4 #
5 # To prepare:
6 #   $ prep-gas nitrogen-2sp.inp nitrogen-2sp.lua
7 #   $ prep-chem nitrogen-2sp.lua nitrogen-2sp-2r.lua chem.lua
8 #
9 # To run:
10 #   $ ruby fvreactor.py
11 $LOAD_PATH << '~/dgdinst/lib'
12 require 'eilmer/gas'
13
14 gm = GasModel.new("nitrogen-2sp.lua")
15 reactor = ThermochemicalReactor.new(gm, "chem.lua")
16
17 gs = GasState.new(gm)
18 gs.p = 1.0e5 # Pa
19 gs.T = 4000.0 # degree K
20 gs.molef = {'N2'=>2.0/3, 'N'=>1.0/3}
21 gs.update_thermo_from_pT()
22
23 tFinal = 300.0e-6 # s
24 t = 0.0
25 dt = 1.0e-6
26 dtSuggest = 1.0e-11
27 puts "# Start integration"
28 f = open("fvreactor.data", 'w')
29 f.write("# 1:t(s)  2:T(K)  3:p(Pa)  4:massf_N2  5:massf_N  6:conc_N2  7:conc_N\n")
30 f.write("%10.3e %10.3f %10.3e %20.12e %20.12e %20.12e %20.12e\n" %
31         [t, gs.T, gs.p, gs.massf[0], gs.massf[1], gs.conc[0], gs.conc[1]])
32 while t <= tFinal do
33   dtSuggest = reactor.update_state(gs, dt, dtSuggest)
34   t = t + dt
35   # dt = dtSuggest # uncomment this to get quicker stepping
36   gs.update_thermo_from_rhou()
37   f.write("%10.3e %10.3f %10.3e %20.12e %20.12e %20.12e %20.12e\n" %
38           [t, gs.T, gs.p, gs.massf[0], gs.massf[1], gs.conc[0], gs.conc[1]])
39 end
40 f.close()
41 puts "# Done."
```

5.1.4 CEAgas model

To get that equilibrium gas estimate of the final gas composition, we may make use of the `CEAgas` gas model as shown below. Behind the scene, the GasModel object is calling up the CEA2 program [4] to do the detailed calculations so, to use this model, you need to have the CEA2 executable program somewhere on your `PATH`.

```
# n2-n-eq-check.py
# Usage: python3 n2-n-eq-check.py
from eilmer.gas import GasModel, GasState
gmodel = GasModel('cea-n2-gas-model.lua')
gs = GasState(gmodel)
gs.p = 1.455e+05
gs.T = 6177.424
gs.update_thermo_from_pT()
print("eq gas state=", gs)
print("ceaSavedData=", gs.ceaSavedData)
```

The transcript, below, has been lightly edited to break the lines into shorter lengths than in the original output.

```
$ python3 n2-n-eq-check.py
eq gas state= GasState(rho=0.070243, p=145500, T=6177.42,
                       u=1.01128e+07, a=1530.8,
                       id=0, gmodel.id=0)
ceaSavedData= {'p': 145500.0, 'rho': 0.070243, 'u': 10112800.0,
               'h': 12184200.0, 'T': 6177.42, 'a': 1530.8,
               'Mmass': 0.024796, 'Rgas': 335.31658331989036,
               'gamma': 1.1313, 'Cp': 8483.7, 's': 11208.3,
               'k': 0.0, 'mu': 0.0,
               'massf': {'N': 0.12976, 'N2': 0.87024}}
```

The content of the gas model file (`cea-n2-gas-model.lua`) is tailored to this particular exercise because we need to specify the fractions of the N2 and N reactants.

```
model = "CEAGas"

CEAGas = {
  mixtureName = 'n2-n',
  speciesList = {"N2", "N"},
  reactants = {N2=2/3, N=1/3},
  inputUnits = "moles",
  withIons = false,
  trace = 1.0e-6
}
```

5.2 Ignition times for hydrogen

As stated earlier (on page 23), the evolution of a charge of perfectly-stirred reactants in a thermally-isolated, fixed-volume reactor is the model behind the main chemical-update function. We can use this basic calculation in a numerical experiment to compare the ignition times for a number of proposed reaction schemes for the combustion of hydrogen. These schemes are:

- a Stanford (2011) scheme with only hydrogen and oxygen participating in the reactions (Appendix A.1),
- an Evans and Schexnayder scheme with only hydrogen and oxygen participating in the reactions (Appendix A.2) and
- a Rogers and Schexnayder scheme that includes hydrogen, oxygen and nitrogen species in the reactions (Appendix A.3).

The script below shows the use of the Stanford-2011 scheme. The only changes required to use the other schemes are the file names provided on lines 1 through 3.

```
1  spFile = "Stanford-2011-gas-model.lua"
2  reacFile = "Stanford-2011-reac-file.lua"
3  outFile = "Stanford-ignition-delay.dat"
4
5  tFinal = 1500.0e-6 -- s
6  pInit = P_atm
7  Tlow = 900.0 -- K
8  Thigh = 1300.0 -- K
9  dT = 10.0
10
11 igCriteria = 5.0e-3 -- mol/m^3 : OH
12
13 function ignition_delay(T, gm, chemUpdate)
14    local Q = gm:createGasState()
15    Q.p = pInit
16    Q.T = T
17    local total = 2 + 1 + 3.76
18    local molef = {H2=2/total, O2=1/total, N2=3.76/total}
19    Q.massf = gm:molef2massf(molef)
20    gm:updateThermoFromPT(Q)
21
22    local t = 0.0
23    local dt = 1.0e-6
24    local dtSuggest = 1.0e-11
25    while t <= tFinal do
26       dtSuggest = chemUpdate:updateState(Q, dt, dtSuggest, gm)
27       t = t + dt
28       dt = dtSuggest
29       gm:updateThermoFromRHOE(Q)
30       local conc = gm:massf2conc(Q)
31       if conc.OH > igCriteria then
32          return t
33       end
34    end
35    return false
```

```
36 end
37
38 function main()
39    local gm = GasModel:new{spFile}
40    local chemUpdate = ChemistryUpdate:new{filename=reacFile, gasmodel=gm}
41
42    local f = assert(io.open(outFile, 'w'))
43    f:write('# 1:T(K)  2:t(s)\n')
44
45    for T=Tlow,Thigh,dT do
46       local tIg = ignition_delay(T, gm, chemUpdate)
47       if tIg then
48          f:write(string.format("%20.12e %20.12e\n", T, tIg))
49       else
50          print("No ignition at T= ", T)
51       end
52    end
53
54 end
55
56 main()
```

The script runs a number of simulations, each starting with the same mixture of hydrogen, oxygen and nitrogen molecules, but with different initial mixture temperatures, `T`. Only one value of pressure is considered in this numerical experiment and that is the standard atmosphere pressure provided as a special constant by the gas module (`P_atm` on line 6).

The `main` function (starting on line 38) coordinates the setting of the initial temperatures and the recording of the ignition times, while the detailed code for each simulation is in the `ignition_delay` function (lines 13 through 36). The simulation code is much like that for the nitrogen reactor discussed in the previous example. An initial gas state is constructed (lines 14 through 20) and the reactions are allowed to proceed over small intervals of time, until ignition occurs. On line 31, the ignition time is defined as the time at which the concentration of OH reaches a specified value, with that value having been given on line 11.

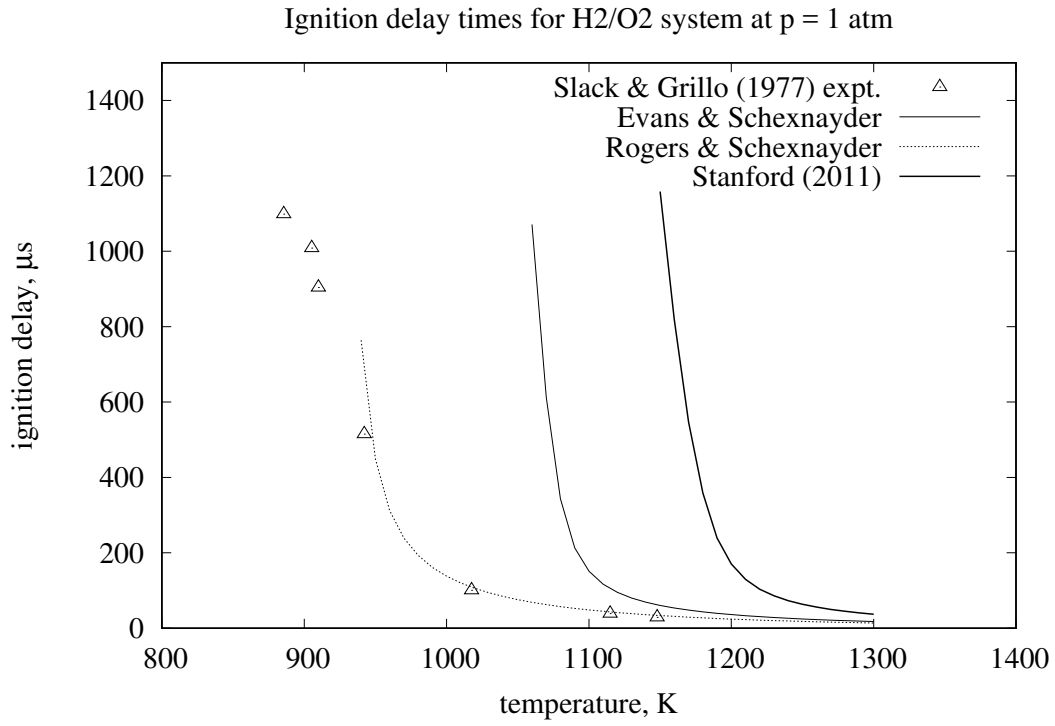

Figure 5.2: Time to reach ignition of a hydrogen-air mixture.

Looking at the plotted times compared with experiment in Figure5.2, we can conclude that nitrogen reactions are an important contributor to hydrogen ignition, especially for temperatures below 1100 K. The two reaction schemes that have only hydrogen and oxygen participating in the reactions miss the experimental mark completely.

5.3 Dissociating nitrogen flow over a 2D cylinder

So far, we have been using the thermochemical models in stand-alone calculations. Now, we look at an example of using the models within a compressible flow simulation done with Eilmer. This example shows the construction of a simple flow domain around a circular cylinder and the set up of a finite-rate reacting model for dissociating nitrogen. High speed flow of nitrogen over a 2D cylinder is a signature experiment for shock tunnel and expansion tube facilities and the data for comparison has come from our colleagues at the DLR-Göttingen shock tunnel laboratory.

The left part of Figure 5.3 shows a representation of the flow region about part of a circular cylinder, with the east boundary representing the surface of the cylinder. We consider just a two-dimensional flow in the simulation, however, the real flow in the shock tunnel would be genuinely three-dimensional, with significant out-of-plane flow over each end of the finite length cylinder. This is okay because our focus here is on the strong thermochemical effects that come with allowing finite-rate reactions of teh nitrogen molecules and atoms.

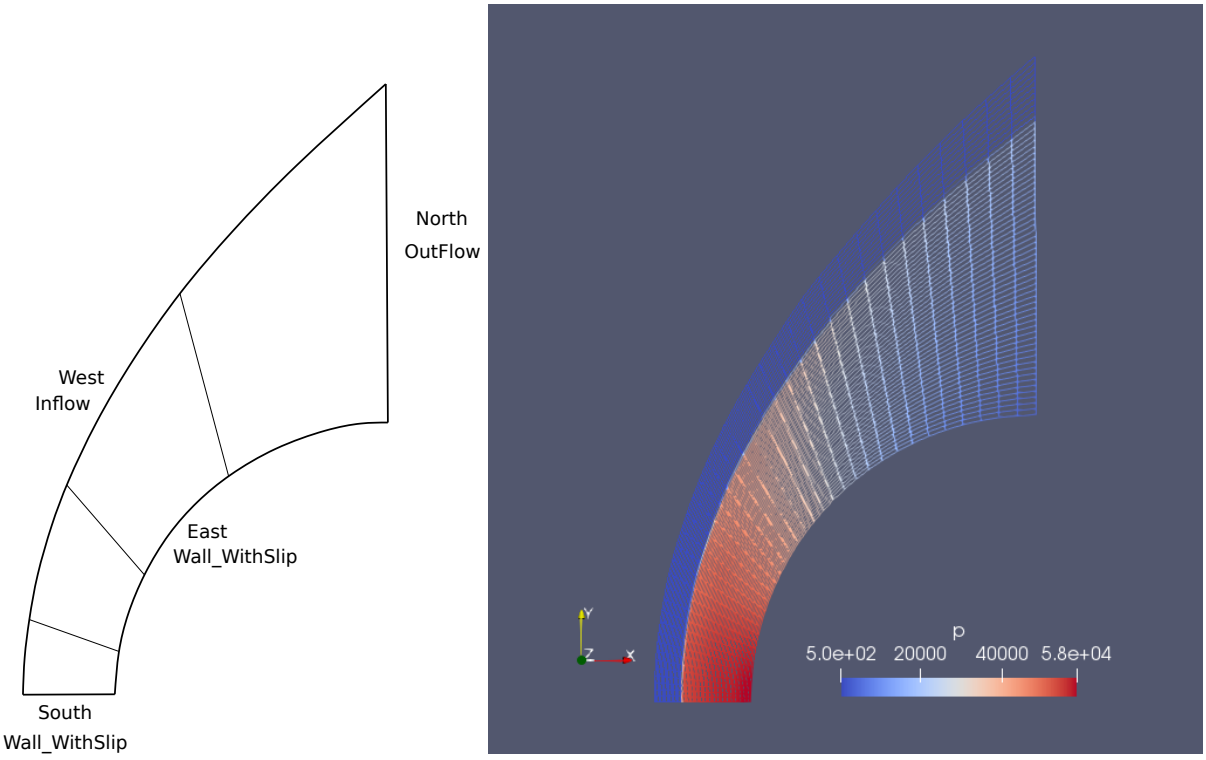

Figure 5.3: Schematic diagram of the flow-domain geometry for the front part of the cylinder (left) with mesh, coloured by pressure on the right.

In our two-dimensional flow domain, the south boundary is along the stagnation line and the south-east intersection is located at the stagnation point on the cylinder. The west boundary sees a supersonic inflow, as would be produced by the shock tunnel nozzle. The north boundary truncates the flow domain over the top of the cylinder and is specified to be a simple ouflow boundary that the gas crosses at (presumably) supersonic conditions. The other radial lines represent the boundaries between the

multiple blocks that make up the full domain grid, which is shown in the right part of the figure. Our motivation for bothering to have multiple blocks is to get the simulation done quickly. The calculation takes less than a minute on a good workstation, using 4 processor cores.

5.3.1 Eilmer input script

The best guidance for setting up flow simulations with Eilmer is found in the user guides [13], [14] and [15], so the material in those reports is a prerequisite to fully understanding the simulation input script shown below. Beyond the set up required for a non-reacting simulation, our essential tasks are to specify the initial and inflow mass fractions for the participating chemical species, and to specify a detailed chemistry file.

```
1 -- n90.lua
2 -- RG & PJ  2015-03-09
3 -- 2015-04-22 build an original grid in this script
4 -- 2019-05-25 Billig shock-shape correlation used to shape grid.
5 --
6 config.title = "Cylinder in dissociating nitrogen flow."
7 print(config.title)
8
9 nsp, nmodes, gmodel = setGasModel('nitrogen-2sp.lua')
10 inflow = FlowState:new{p=500.0, T=700.0, velx=5000.0, massf={N2=1.0}}
11 initial = FlowState:new{p=5.0, T=300.0, massf={N2=1.0}}
12 Minf = inflow.velx / inflow.a
13 print("Minf=", Minf)
14
15 config.reacting = true
16 config.reactions_file = 'e4-chem.lua'
17
18 require "billig_patch"
19 R = 0.045 -- m
20 bp = billig_patch.make_patch{Minf=Minf, R=R, xc=R, scale=0.95}
21 cf = RobertsFunction:new{end0=true, end1=false, beta=1.1}
22 grid = StructuredGrid:new{psurface=bp.patch, niv=61, njv=41,
23                           cfList={west=cf, east=cf}}
24
25 -- We can leave east and south as slip-walls.
26 blks = FBArray:new{grid=grid, initialState=initial, label="blk",
27                    bcList={west=InFlowBC_Supersonic:new{flowState=inflow},
28                            north=OutFlowBC_Simple:new{}},
29                    nib=1, njb=4}
30
31 -- Set a few more config options.
32 -- To get a reasonable start, we needed to set dt_init.
33 config.max_time = 100.0e-6
34 config.max_step = 40000
35 config.dt_plot = 20.0e-6
36 config.dt_init = 5.0e-9
```

Line 9 is where the detailed gas model file is specified. Only one gas model may be specified for a simulation, however, other gas models may be used within the input

script, using the Lua programming interface described in Section 5.1.1. Note that three items are returned by the `setGasModel` function, the third being a reference to the initialized gas model. This is now available for use within the Lua input script and could be used to initialize a ThermochemicalReactor object, for example. Here, we do not happen to make any direct use of the returned gas model but it is good to know that it is available.

Lines 15 and 16 specify that we want the chemical reactions to be active during the simulation and specify the detailed chemistry file. Later, when the simulation is run, the ThermochemicalReactor will be initialized from the information in this file.

Lines 10 and 11, specify the inflow and initial gas states to be undissociated nitrogen, with the mass fractions being specified as tables containing an entry for nitrogen molecules only. In each case, the unspecified mass fraction of `N` atoms is presumed to be zero.

The remaining lines in the input script describe the flow domain (lines 18-20), specify how to discretize it (lines 21-23), and set up an array of FluidBlocks consisting of grid and boundary condition information (lines 26-29). The final few lines of the input script (lines 33-36) set a few simulation control parameters. Again, refer to the flow solver user guide [14] for details.

5.3.2 Results

By the end of the simulation, a shock layer of high-temperature and high-pressure gas has developed over the cylinder. This shock layer can be seen coloured by pressure value in the right part of Figure 5.3. Temperature and mass-fraction of nitrogen atoms is shown in Figure 5.4. Following the gas as it flows from left to right, after it has been processed by the shock, you can see increasing dissociation and corresponding decrease in temperature as the gas approaches the cylinder's surface.

One of the effects of finite-rate chemistry is to reduce the shock layer thickness, relative to that for a non-reacting gas. If the reactions are suppressed, the shock layer would be larger than allowed by the presently defined domain. If you want to experiment with a non-reacting simulation, set `config.reacting = false` on line 15 of the input script. Also, change the value of `scale=0.95` (on line 20) to something more like `1.1` to accommodate the thicker shock layer.

Figure 5.5 shows the temperature along the stagnation line. Moving from left to right across the figure, the shock processing results in a sudden jump in temperature, and then the temperature relaxes as the dissociation reaction absorbs thermal energy during the time that the gas takes to approach the cylinder surface. A non-reacting case would have the gas temperature jumping at the shock and then continuing to increase slowly as the gas approaches the stagnation point. Comparison of the Eilmer data with the reference data provided by Sebastian Karl (DLR Göttingen) is reasonably good, given that Eilmer simulation is done on a very low-resolution grid and the boundary layer on the cylinder is not modelled at all.

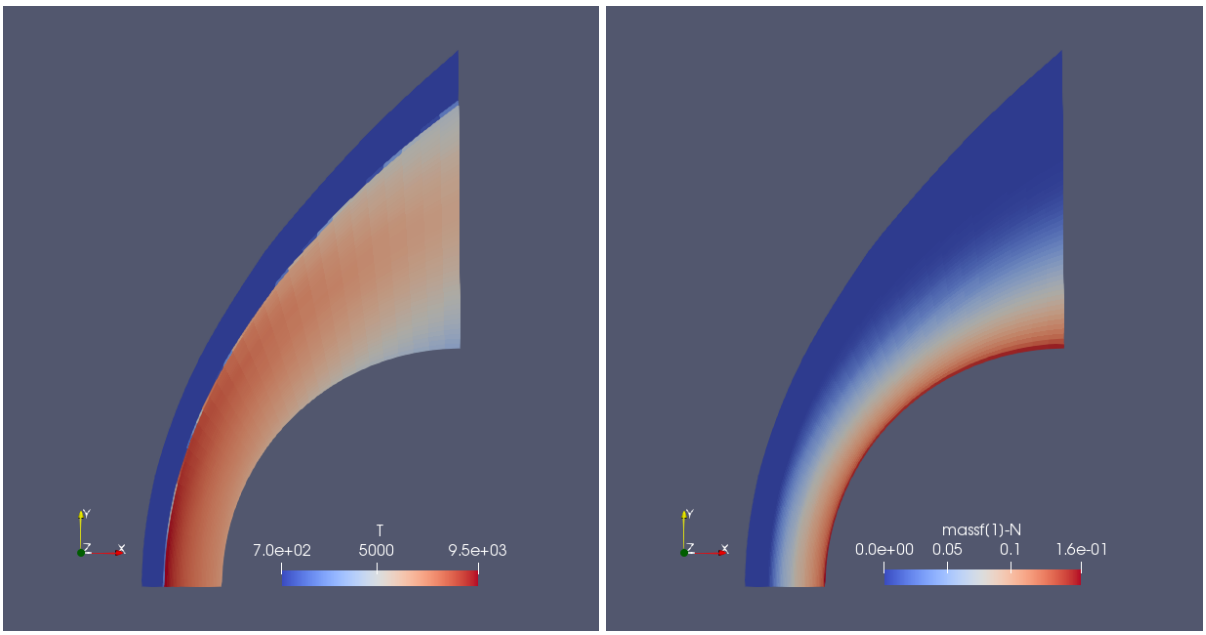

Figure 5.4: Temperature and mass fraction of nitrogen atoms for the n90 bluff body exercise.

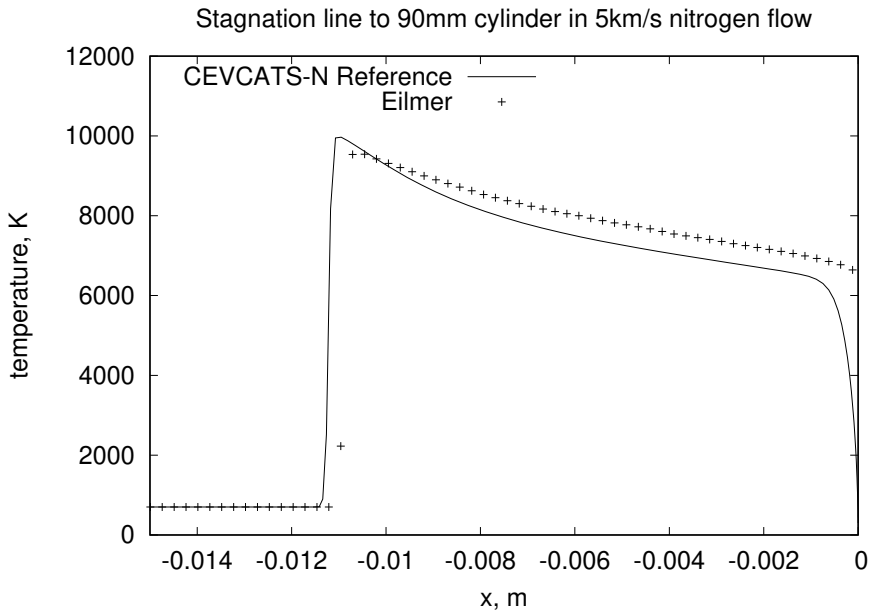

Figure 5.5: Temperature data along the stagnation streamline. The position x=0 is at the cylinder surface.

References

[1] B. J. McBride and S. Gordon. Computer program for calculation of complex chemical equilibrium compositions and applications. Part 2: Users manual and program description. Reference Publication 1311, NASA, 1996.

[2] R. N. Gupta, J. M. Yos, R. A. Thompson, and K.-P. Lee. A review of reaction rates and thermodynamic and transport properties for an 11-species air model for chemical and thermal nonequilibrium calculations to 30 000 k. Reference Publication 1232, NASA, 1990.

[3] C.R. Wilke. A viscosity equation for gas mixtures. *Journal of Chemical Physics*, 18:517–519, 1950.

[4] S. Gordon and B. J. McBride. Computer program for calculation of complex chemical equilibrium compositions and applications. Part 1: Analysis. Reference Publication 1311, NASA, 1994.

[5] E.S. Oran and J.P. Boris. *Numerical Simulation of Reactive Flow*. Cambridge University Press, New York, USA, 2nd edition, 2001.

[6] E. Fehlberg. Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems. Technical Report R-315, NASA, 1969.

[7] D. R. Mott. *New Quasi-Steady-State and Partial-Equilibrium Methods for Integrating Chemically Reacting Systems*. PhD thesis, University of Michigan, 1999.

[8] D. R. Mott, E. S. Oran, and B. van Leer. A quasi-steady-state solver fot the stiff ordinary differential equations of reaction kinetics. *Journal of Computational Physics*, 164:407–428, 2000.

[9] S. R. Qureshi and R. Prosser. Implementation of α-qss stiff integrations methods for solving the detailed combustion chemistry. In *Proceedings of the World Congress on Engineering 2007*, volume II, pages 1352 – 1357, 2007.

[10] T.R. Young and J.P. Boris. A numerical technique for solving stiff ordinary differential equations associated with the chemical kinetics of reactive-flow problems. *Journal of Physical Chemistry*, 81(25):2424–2427, 1977.

[11] B. T. O'Flaherty. *Reducing the Global Warming of Coal Mine Ventilation Air by Combustion in a Free-Piston Engine*. PhD thesis, The University of Queensland, 2012.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, USA, 3rd edition, 2007.

[13] R. J. Gollan and P. A. Jacobs. The Eilmer 4.0 flow simulation program: Guide to the basic gas models package, including gas-calc and the API. School of Mechanical and Mining Engineering Technical Report 2017/27, The University of Queensland, Brisbane, Australia, February 2018.

[14] Peter A. Jacobs and Rowan J. Gollan. The Eilmer 4.0 flow simulation program: Guide to the transient flow solver, including some examples to get you started. School of Mechanical and Mining Engineering Technical Report 2017/26, The University of Queensland, Brisbane, Australia, February 2018.

[15] Peter A. Jacobs, Rowan J. Gollan, and Ingo Jahn. The Eilmer 4.0 flow simulation program: Guide to the geometry package, for construction of flow paths. School of Mechanical and Mining Engineering Technical Report 2017/25, The University of Queensland, Brisbane, Australia, February 2018.

APPENDIX A

Input files for hydrogen combustion

A.1 Stanford, 2011

```
model = 'thermally perfect gas'
species = {'H2', 'O2', 'H2O', 'HO2', 'H2O2', 'OH', 'O', 'H', 'N2'}
```

```
-- Author: Rowan J. Gollan
-- Date: 2015-03-17
--
-- Reference:
-- Hong, Z., Davidson, D.F. and Hanson, R.K. (2011)
-- An improved H2/O2 mechanism based on recent
-- shock tube/laser absorption measurements.
-- Combustion and Flame, 158, pp. 633--644
--
-- NOTE:
-- Table 1 in Hong et al contains the suggested
-- reaction mechanism with reaction rates.
-- However, there is also a Chemkin input file
-- available online where the article is hosted.
-- The input file has some differences with regards
-- to efficiency values. I have adopted those
-- values from the supplied input file.
--
-- Updated: 2016-06-23
--          Use new format for Eilmer4

S = 1.0/1.987

Config{
   odeStep = {method='alpha-qss'},
}

Reaction{
   'H + O2 <=> OH + O',
   fr={'Arrhenius', A=1.04e14, n=0.0, C=15286*S},
   label='r1'
}
```

```
33
34 Reaction{
35    'H + O2 (+ M) <=> HO2 (+ M)',
36    fr={'pressure dependent',
37         kInf={A=5.59e13, n=0.2, C=0.0},
38         k0={A=3.70e19, n=-1.0, C=0.0},
39         Troe={F_cent=0.8}
40    },
41    efficiencies={H2O=1.0,H=0.0,O2=0.0,OH=0.0,O=0.0,HO2=0.0,H2O2=0.0},
42    label='r2b'
43 }
44
45 Reaction{
46    'H + O2 (+ M) <=> HO2 (+ M)',
47    fr={'pressure dependent',
48         kInf={A=5.59e13, n=0.2, C=0.0},
49         k0={A=5.69e18, n=-1.1, C=0.0},
50         Troe={F_cent=0.7}
51    },
52    efficiencies={O2=1.0,H2O=0.0,H=0.0,OH=0.0,O=0.0,HO2=0.0,H2O2=0.0},
53    label='r2c'
54 }
55
56 Reaction{
57    'H + O2 (+ M) <=> HO2 (+ M)',
58    fr={'pressure dependent',
59         kInf={A=5.59e13, n=0.2, C=0.0},
60         k0={A=2.65e19, n=-1.3, C=0.0},
61         Troe={F_cent=0.7}
62    },
63    efficiencies={H2=2.5,H2O2=12.0,H2O=0.0,O2=0.0},
64    label='r2d'
65 }
66
67 Reaction{
68    'H2O2 (+ M) <=> 2OH (+ M)',
69    fr={'pressure dependent',
70         kInf={A=8.59e14, n=0.0, C=48560*S},
71         k0={A=9.55e15, n=0.0, C=42203*S},
72         Troe={F_cent=1.0}
73    },
74    efficiencies={N2=1.5,H2=2.5,H2O=15,H2O2=15},
75    label='r3'
76 }
77
78 -- Reaction 4 appears twice.
79 -- The reaction rate constants are added together
80 -- as proposed by Hong et al in Section 2.5.
81 -- To achieve the same effect, the reaction can just
82 -- be listed twice with different reaction rates.
83 Reaction{
84    'OH + H2O2 <=> H2O + HO2',
85    fr={'Arrhenius', A=1.74e12, n=0.0, C=318*S},
86    label='r4a'
87 }
88 Reaction{
89    'OH + H2O2 <=> H2O + HO2',
```

```
90     fr={'Arrhenius', A=7.59e13, n=0.0, C=7269*S},
91     label='r4b'
92 }
93
94 Reaction{
95     'OH + HO2 <=> H2O + O2',
96     fr={'Arrhenius', A=2.89e13, n=0.0, C=-500*S},
97     label='r5'
98 }
99
100 Reaction{
101     'HO2 + HO2 <=> H2O2 + O2',
102     fr={'Arrhenius', A=1.30e11, n=0.0, C=-1603*S},
103     label='r6a'
104 }
105 Reaction{
106     'HO2 + HO2 <=> H2O2 + O2',
107     fr={'Arrhenius', A=4.20e14, n=0.0, C=11980*S},
108     label='r6b'
109 }
110
111 Reaction{
112     'H2O + M <=> H + OH + M',
113     fr={'Arrhenius', A=6.06e27, n=-3.31, C=120770*S},
114     efficiencies={O2=1.5,H2=3.0,H2O=0.0},
115     label='r7a'
116 }
117 Reaction{
118     'H2O + H2O <=> H + OH + H2O',
119     fr={'Arrhenius', A=1.00e26, n=-2.44, C=120160*S},
120     label='r7b'
121 }
122
123 Reaction{
124     'OH + OH <=> H2O + O',
125     fr={'Arrhenius', A=3.57e4, n=2.4, C=-2111*S},
126     label='r8'
127 }
128
129 Reaction{
130     'O + H2 <=> H + OH',
131     fr={'Arrhenius', A=3.82e12, n=0.0, C=7948*S},
132     label='r9a'
133 }
134 Reaction{
135     'O + H2 <=> H + OH',
136     fr={'Arrhenius', A=8.79e14, n=0.0, C=19170*S},
137     label='r9b'
138 }
139
140 Reaction{
141     'H2 + OH <=> H2O + H',
142     fr={'Arrhenius', A=2.17e8, n=1.52, C=3457*S},
143     label='r10'
144 }
145
146 Reaction{
```

```
147     'H + HO2 <=> OH + OH',
148     fr={'Arrhenius', A=7.08e13, n=0.0, C=300*S},
149     label='r11'
150 }
151
152 Reaction{
153     'H + HO2 <=> H2O + O',
154     fr={'Arrhenius', A=1.45e12, n=0.0, C=0.0},
155     label='r12'
156 }
157
158 Reaction{
159     'H + HO2 <=> H2 + O2',
160     fr={'Arrhenius', A=3.66e6, n=2.087, C=-1450*S},
161     label='r13'
162 }
163
164 Reaction{
165     'O + HO2 <=> OH + O2',
166     fr={'Arrhenius', A=1.63e13, n=0.0, C=-445*S},
167     label='r14'
168 }
169
170 Reaction{
171     'H2O2 + H <=> HO2 + H2',
172     fr={'Arrhenius', A=1.21e7, n=2.0, C=5200*S},
173     label='r15'
174 }
175
176 Reaction{
177     'H2O2 + H <=> H2O + OH',
178     fr={'Arrhenius', A=1.02e13, n=0.0, C=3577*S},
179     label='r16'
180 }
181
182 Reaction{
183     'H2O2 + O <=> OH + HO2',
184     fr={'Arrhenius', A=8.43e11, n=0.0, C=3970*S},
185     label='r17'
186 }
187
188 Reaction{
189     'H2 + M <=> H + H + M',
190     fr={'Arrhenius', A=5.84e18, n=-1.1, C=104380*S},
191     efficiencies={H2O=14.4,H2O2=14.4,H2=0.0,O2=0.0},
192     label='r18a'
193 }
194 Reaction{
195     'H2 + H2 <=> H + H + H2',
196     fr={'Arrhenius', A=9.03e14, n=0.0, C=96070*S},
197     label='r18b'
198 }
199 Reaction{
200     'H2 + O2 <=> H + H + O2',
201     fr={'Arrhenius', A=4.58e19, n=-1.4, C=104380*S},
202     label='r18c'
203 }
```

```
204
205 Reaction{
206     'O + O + M <=> O2 + M',
207     fr={'Arrhenius', A=6.16e15, n=-0.5, C=0.0},
208     efficiencies={H2=2.5,H2O=12,H2O2=12},
209     label='r19'
210 }
211
212 Reaction{
213     'O + H + M <=> OH + M',
214     fr={'Arrhenius', A=4.71e18, n=-1.0, C=0.0},
215     efficiencies={H2=2.5,H2O=12,H2O2=12},
216     label='r20'
217 }
```

A.2 Evans-Schexnayder

```
model = 'thermally perfect gas'
species = {'H2', 'O2', 'H2O', 'HO2', 'OH', 'O', 'H', 'N2'}
```

```
-- Author: Rowan J. Gollan
-- Date: 02-Feb-2010
-- Place: Poquoson, Virginia, USA
--
-- Adapted from Python file: evans_schexnayder.py
--
-- This file provides four chemical kinetic descriptions
-- of hydrogen combustion.  You can select between the various
-- options below by setting the 'model' variable below to one of
-- the strings listed below.
--
-- REDUCED  : a 7-species, 8-reactions description of hydrogen
--            combustion in pure oxygen
-- PURE_O2  : a 7-species, 16-reactions description of hydrogen
--            combustion in pure oxygen
-- IN_AIR   : a 12-species, 25-reactions description of hydrogen
--            combustion in air (N2 and O2)
-- INERT_N2 : an 8-species, 16-reactions description of hydrogen
--            combustion in air with inert N2 (acting as diluent only).
--
-- The numbering of reactions in this file corresponds to
-- Table 1 in Evans and Schexnayder (1980).
--
-- Reference:
--  Evans, J.S. and Shexnayder Jr, C.J. (1980)
--  Influence of Chemical Kinetics and Unmixedness
--  on Burning in Supersonic Hydrogen Flames
--  AIAA Journal 18:2 pp 188--193
--
-- History:
--  07-Mar-2006 -- first prepared
--
-- Species used in REDUCED: O, O2, H, H2, H2O, OH, N2
-- Species used in PURE_O2: O, O2, H, H2, H2O, OH, HO2
-- Species used in IN_AIR: O, O2, N, N2, H, H2, H2O, HO2, OH, NO, NO2, HNO2
-- Species used in INERT_N2: O, O2, H, H2, H2O, OH, HO2, N2

options = {
   REDUCED=true,
   PURE_O2=true,
   IN_AIR=true,
   INERT_N2=true
}

-- User selects model here
model = 'INERT_N2'

-- Check that selection is valid
if options[model] == nil then
```

```
50    print("User selected model: ", model)
51    print("is not valid.")
52    print("Valid models are:")
53    for m,_ in pairs(options) do
54       print(m)
55    end
56 end
57
58 Config{
59    odeStep = {method='alpha-qss'},
60 --   tightTempCoupling = true
61 }
62
63 Reaction{
64    'HNO2 + M <=> NO + OH + M',
65    fr={'Arrhenius', A=5.0e17, n=-1.0, C=25000.0},
66    br={'Arrhenius', A=8.0e15, n=0.0, C=-1000.0},
67    label='r1'
68 }
69
70 Reaction{
71    'NO2 + M <=> NO + O + M',
72    fr={'Arrhenius', A=1.1e16, n=0.0, C=32712.0},
73    br={'Arrhenius', A=1.1e15, n=0.0, C=-941.0},
74    label='r2'
75 }
76
77 Reaction{
78    'H2 + M <=> H + H + M',
79    fr={'Arrhenius', A=5.5e18, n=-1.0, C=51987.0},
80    br={'Arrhenius', A=1.8e18, n=-1.0, C=0.0},
81    label='r3'
82 }
83
84 Reaction{
85    'O2 + M <=> O + O + M',
86    fr={'Arrhenius', A=7.2e18, n=-1.0, C=59340.0},
87    br={'Arrhenius', A=4.0e17, n=-1.0, C=0.0},
88    label='r4'
89 }
90
91 Reaction{
92    'H2O + M <=> OH + H + M',
93    fr={'Arrhenius', A=5.2e21, n=-1.5, C=59386.0},
94    br={'Arrhenius', A=4.4e20, n=-1.5, C=0.0},
95    label='r5'
96 }
97
98 Reaction{
99    'OH + M <=> O + H + M',
100   fr={'Arrhenius', A=8.5e18, n=-1.0, C=50830.0},
101   br={'Arrhenius', A=7.1e18, n=-1.0, C=0.0},
102   label='r6'
103 }
104
105 Reaction{
106   'HO2 + M <=> H + O2 + M',
```

```
107    fr={'Arrhenius', A=1.7e16, n=0.0, C=23100.0},
108    br={'Arrhenius', A=1.1e16, n=0.0, C=-440.0},
109    label='r7'
110 }
111
112 Reaction{
113    'H2O + O <=> OH + OH',
114    fr={'Arrhenius', A=5.8e13, n=0.0, C=9059.0},
115    br={'Arrhenius', A=5.3e12, n=0.0, C=503.0},
116    label='r8'
117 }
118
119 Reaction{
120    'H2O + H <=> OH + H2',
121    fr={'Arrhenius', A=8.4e13, n=0.0, C=10116.0},
122    br={'Arrhenius', A=2.0e13, n=0.0, C=2600.0},
123    label='r9'
124 }
125
126 Reaction{
127    'O2 + H <=> OH + O',
128    fr={'Arrhenius', A=2.2e14, n=0.0, C=8455.0},
129    br={'Arrhenius', A=1.5e13, n=0.0, C=0.0},
130    label='r10'
131 }
132
133 Reaction{
134    'H2 + O <=> OH + H',
135    fr={'Arrhenius', A=7.5e13, n=0.0, C=5586.0},
136    br={'Arrhenius', A=3.0e13, n=0.0, C=4429.0},
137    label='r11'
138 }
139
140 Reaction{
141    'H2 + O2 <=> OH + OH',
142    fr={'Arrhenius', A=1.7e13, n=0.0, C=24232.0},
143    br={'Arrhenius', A=5.7e11, n=0.0, C=14922.0},
144    label='r12'
145 }
146
147 Reaction{
148    'H2 + O2 <=> H + HO2',
149    fr={'Arrhenius', A=1.9e13, n=0.0, C=24100.0},
150    br={'Arrhenius', A=1.3e13, n=0.0, C=0.0},
151    label='r13'
152 }
153
154 Reaction{
155    'OH + OH <=> H + HO2',
156    fr={'Arrhenius', A=1.7e11, n=0.5, C=21137.0},
157    br={'Arrhenius', A=6.0e13, n=0.0, C=0.0},
158    label='r14'
159 }
160
161 Reaction{
162    'H2O + O <=> H + HO2',
163    fr={'Arrhenius', A=5.8e11, n=0.5, C=28686.0},
```

```
164     br={'Arrhenius', A=3.0e13, n=0.0, C=0.0},
165     label='r15'
166 }
167
168 Reaction{
169     'OH + O2 <=> O + HO2',
170     fr={'Arrhenius', A=3.7e11, n=0.64, C=27840.0},
171     br={'Arrhenius', A=1.0e13, n=0.0, C=0.0},
172     label='r16'
173 }
174
175 Reaction{
176     'H2O + O2 <=> OH + HO2',
177     fr={'Arrhenius', A=2.0e11, n=0.5, C=36296.0},
178     br={'Arrhenius', A=1.2e13, n=0.0, C=0.0},
179     label='r17'
180 }
181
182 Reaction{
183     'H2O + OH <=> H2 + HO2',
184     fr={'Arrhenius', A=1.2e12, n=0.21, C=39815.0},
185     br={'Arrhenius', A=1.7e13, n=0.0, C=12582.0},
186     label='r18'
187 }
188
189 Reaction{
190     'O + N2 <=> N + NO',
191     fr={'Arrhenius', A=5.0e13, n=0.0, C=37940.0},
192     br={'Arrhenius', A=1.1e13, n=0.0, C=0.0},
193     label='r19'
194 }
195
196 Reaction{
197     'H + NO <=> N + OH',
198     fr={'Arrhenius', A=1.7e14, n=0.0, C=24500.0},
199     br={'Arrhenius', A=4.5e13, n=0.0, C=0.0},
200     label='r20'
201 }
202
203 Reaction{
204     'O + NO <=> N + O2',
205     fr={'Arrhenius', A=2.4e11, n=0.5, C=19200.0},
206     br={'Arrhenius', A=1.0e12, n=0.5, C=3120.0},
207     label='r21'
208 }
209
210 Reaction{
211     'NO + OH <=> H + NO2',
212     fr={'Arrhenius', A=2.0e11, n=0.5, C=15500.0},
213     br={'Arrhenius', A=3.5e14, n=0.0, C=740.0},
214     label='r22'
215 }
216
217
218 Reaction{
219     'NO + O2 <=> O + NO2',
220     fr={'Arrhenius', A=1.0e12, n=0.0, C=22800.0},
```

```
221    br={'Arrhenius', A=1.0e13, n=0.0, C=302.0},
222    label='r23'
223 }
224
225 Reaction{
226    'NO2 + H2 <=> H + HNO2',
227    fr={'Arrhenius', A=2.4e13, n=0.0, C=14500.0},
228    br={'Arrhenius', A=5.0e11, n=0.5, C=1500.0},
229    label='r24'
230 }
231
232 Reaction{
233    'NO2 + OH <=> NO + HO2',
234    fr={'Arrhenius', A=1.0e11, n=0.5, C=6000.0},
235    br={'Arrhenius', A=3.0e12, n=0.5, C=1200.0},
236    label='r25'
237 }
238
239 reactions_list = {}
240
241 if model == 'REDUCED' then
242    reactions_list = {'r3', 'r4', 'r5', 'r6', 'r8', 'r9', 'r10', 'r11'}
243 end
244
245 if model == 'PURE_O2' or model == 'INERT_N2' then
246    reactions_list = {'r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10',
247                      'r11', 'r12', 'r13', 'r14', 'r15', 'r16', 'r17', 'r18'}
248 end
249
250
251 if model ~= 'IN_AIR' then
252    -- For all other models we select only a subset.
253    selectOnlyReactionsWithLabel(reactions_list)
254 end
```

A.3 Rogers-Schexnayder

```
model = 'thermally perfect gas'
species = {'O', 'O2', 'N', 'N2', 'H', 'H2',
        'H2O', 'HO2', 'OH', 'NO', 'NO2',
        'HNO2', 'HNO3', 'O3', 'H2O2', 'HNO' }
```

```
-- Author: Rowan J. Gollan
-- Date: 29-Mar-2009
-- Place: Poquoson, Virginia, USA
--
-- Adapted from Python file: rogers_schexnayder.py
--
-- This file provides a reaction scheme for
-- hydrogen combustion in air.
-- NOTE: This scheme does not include carbonaceous compounds
-- or Argon (or any of the associated reactions).
--
-- Reference:
-- Rogers, R.C. and Schexnayder, Jr., C.J. (1981)
-- Chemical Kinetic Analysis of Hydroden-Air
-- Ignition and Reaction Times
-- NASA Technical Paper 1856
--
-- Species used: O, O2, N, N2, H, H2, H2O, HO2, OH, NO, NO2, HNO2, HNO3, O3, H2O2, HNO
--
-- Updated 2016-07-31
--    Updated for eilmer4

Config{
   odeStep = {method='alpha-qss'},
   tightTempCoupling = true
}

Reaction{
   'O2 + M <=> O + O + M',
   fr={"Arrhenius", A=0.72e19, n=-1.0, C=59340.0},
   efficiencies={O2=4.0, O=10.0, H2O=2.0},
   label='r1'
}

Reaction{
   'M + H2 <=> H + H + M',
   fr={'Arrhenius', A=0.55e19, n=-1.0, C=51987.0},
   efficiencies={H=5.0, H2=2.0, H2O=8.0},
   label='r2'
}

Reaction{
   'M + H2O <=> H + OH + M',
   fr={'Arrhenius', A=0.52e22, n=-1.5, C=59386.0},
   efficiencies={H2O=6.0},
   label='r3'
}
```

```
48
49 Reaction{
50    'H + O2 + M <=> HO2 + M',
51    fr={'Arrhenius', A=0.23e16, n=0.0, C=-403.0},
52    efficiencies={H2=2.0, H2O=13.0},
53    label='r4'
54 }
55
56 Reaction{
57    'M + NO2 <=> NO + O + M',
58    fr={'Arrhenius', A=0.11e17, n=0.0, C=32710.0},
59    label='r5'
60 }
61
62 Reaction{
63    'M + NO <=> N + O + M',
64    fr={'Arrhenius', A=0.41e19, n=-1.0, C=75330.0},
65    label='r6'
66 }
67
68 -- not included, ignoring the carbonaceous compounds
69 r7 = 'M + O + CO <=> CO2 + M'
70
71 Reaction{
72    'M + H + NO <=> HNO + M',
73    fr={'Arrhenius', A=0.54e16, n=0.0, C=-300.0},
74    efficiencies={H2O=3.0},
75    label='r8'
76 }
77
78 Reaction{
79    'M + H2O2 <=> OH + OH + M',
80    fr={'Arrhenius', A=0.12e18, n=0.0, C=22899.0},
81    efficiencies={H2O=6.0},
82    label='r9'
83 }
84
85 Reaction{
86    'M + OH + NO <=> HNO2 + M',
87    fr={'Arrhenius', A=0.80e16, n=0.0, C=-1000.0},
88    label='r10'
89 }
90
91 Reaction{
92    'M + OH + NO2 <=> HNO3 + M',
93    fr={'Arrhenius', A=0.13e17, n=0.0, C=-1107.0},
94    label='r11'
95 }
96
97 Reaction{
98    'M + O3 <=> O2 + O + M',
99    fr={'Arrhenius', A=0.13e22, n=-2.0, C=12800.0},
100   efficiencies={O2=1.5},
101   label='r12'
102 }
103
104 r13 = 'M + HCO <=> CO + H + M'
```

```
105
106 Reaction{
107     'M + O + H <=> OH + M',
108     fr={'Arrhenius', A=0.71e19, n=-1.0, C=0.0},
109     label='r14'
110 }
111
112 Reaction{
113     'H2O + O <=> OH + OH',
114     fr={'Arrhenius', A=0.58e14, n=0.0, C=9059.0},
115     label='r15'
116 }
117
118 Reaction{
119     'H2 + OH <=> H2O + H',
120     fr={'Arrhenius', A=0.20e14, n=0.0, C=2600.0},
121     label='r16'
122 }
123
124 Reaction{
125     'O2 + H <=> OH + O',
126     fr={'Arrhenius', A=0.22e15, n=0.0, C=8455.0},
127     label='r17'
128 }
129
130 Reaction{
131     'H2 + O <=> OH + H',
132     fr={'Arrhenius', A=0.75e14, n=0.0, C=5586.0},
133     label='r18'
134 }
135
136 Reaction{
137     'H2 + O2 <=> OH + OH',
138     fr={'Arrhenius', A=0.10e14, n=0.0, C=21641.0},
139     label='r19'
140 }
141
142 Reaction{
143     'H + HO2 <=> H2 + O2',
144     fr={'Arrhenius', A=0.24e14, n=0.0, C=350.0},
145     label='r20'
146 }
147
148 Reaction{
149     'H2 + O2 <=> H2O + O',
150     fr={'Arrhenius', A=0.41e14, n=0.0, C=25400.0},
151     label='r21'
152 }
153
154 Reaction{
155     'H + HO2 <=> OH + OH',
156     fr={'Arrhenius', A=0.24e15, n=0.0, C=950.0},
157     label='r22'
158 }
159
160 Reaction{
161     'H2O + O <=> H + HO2',
```

```
162     fr={'Arrhenius', A=0.58e12, n=0.5, C=28686.0},
163     label='r23'
164 }
165
166 Reaction{
167     'O + HO2 <=> OH + O2',
168     fr={'Arrhenius', A=0.5e14, n=0.0, C=503.0},
169     label='r24'
170 }
171
172 Reaction{
173     'OH + HO2 <=> O2 + H2O',
174     fr={'Arrhenius', A=0.3e14, n=0.0, C=0.0},
175     label='r25'
176 }
177
178 Reaction{
179     'H2 + HO2 <=> H2O + OH',
180     fr={'Arrhenius', A=0.2e14, n=0.0, C=12582.0},
181     label='r26'
182 }
183
184 Reaction{
185     'HO2 + H2 <=> H + H2O2',
186     fr={'Arrhenius', A=0.73e12, n=0.0, C=9400.0},
187     label='r27'
188 }
189
190 Reaction{
191     'H2O2 + H <=> OH + H2O',
192     fr={'Arrhenius', A=0.32e15, n=0.0, C=4504.0},
193     label='r28'
194 }
195
196 Reaction{
197     'HO2 + OH <=> O + H2O2',
198     fr={'Arrhenius', A=0.52e11, n=0.5, C=10600.0},
199     label='r29'
200 }
201
202 Reaction{
203     'HO2 + H2O <=> OH + H2O2',
204     fr={'Arrhenius', A=0.28e14, n=0.0, C=16500.0},
205     label='r30'
206 }
207
208 Reaction{
209     'HO2 + HO2 <=> H2O2 + O2',
210     fr={'Arrhenius', A=0.2e13, n=0.0, C=0.0},
211     label='r31'
212 }
213
214 Reaction{
215     'O + O3 <=> O2 + O2',
216     fr={'Arrhenius', A=0.10e14, n=0.0, C=2411.0},
217     label='r32'
218 }
```

```
219
220 Reaction{
221     'O3 + NO <=> NO2 + O2',
222     fr={'Arrhenius', A=0.54e12, n=0.0, C=1200.0},
223     label='r33'
224 }
225
226 Reaction{
227     'O3 + H <=> OH + O2',
228     fr={'Arrhenius', A=0.70e14, n=0.0, C=560.0},
229     label='r34'
230 }
231
232 Reaction{
233     'O3 + OH <=> O2 + HO2',
234     fr={'Arrhenius', A=0.90e12, n=0.0, C=1000.0},
235     label='r35'
236 }
237
238 Reaction{
239     'O + N2 <=> NO + N',
240     fr={'Arrhenius', A=0.50e14, n=0.0, C=37940.0},
241     label='r36'
242 }
243
244 Reaction{
245     'H + NO <=> OH + N',
246     fr={'Arrhenius', A=0.17e15, n=0.0, C=24500.0},
247     label='r37'
248 }
249
250 Reaction{
251     'O + NO <=> O2 + N',
252     fr={'Arrhenius', A=0.15e10, n=1.0, C=19500.0},
253     label='r38'
254 }
255
256 Reaction{
257     'NO2 + H <=> NO + OH',
258     fr={'Arrhenius', A=0.35e15, n=0.0, C=740.0},
259     label='r39'
260 }
261
262 Reaction{
263     'NO2 + O <=> NO + O2',
264     fr={'Arrhenius', A=0.10e14, n=0.0, C=302.0},
265     label='r40'
266 }
267
268 Reaction{
269     'NO2 + H2 <=> HNO2 + H',
270     fr={'Arrhenius', A=0.24e14, n=0.0, C=14595.0},
271     label='r41'
272 }
273
274 Reaction{
275     'HO2 + NO <=> NO2 + OH',
```

```
276     fr={'Arrhenius', A=0.30e13, n=0.5, C=1208.0},
277     label='r42'
278 }
279
280 Reaction{
281     'NO2 + H2O <=> HNO2 + OH',
282     fr={'Arrhenius', A=0.32e13, n=0.0, C=22000.0},
283     label='r43'
284 }
285
286 Reaction{
287     'NO2 + OH <=> HNO2 + O',
288     fr={'Arrhenius', A=0.21e13, n=0.0, C=12580.0},
289     label='r44'
290 }
291
292 r45 = 'CO + OH <=> CO2 + H'
293 r46 = 'CO2 + O <=> O2 + CO'
294 r47 = 'H2O + CO <=> HCO + OH'
295 r48 = 'OH + CO <=> HCO + O'
296 r49 = 'H2 + CO <=> HCO + H'
297 r50 = 'HO2 + CO <=> CO2 + OH'
298
299 Reaction{
300     'HNO + H <=> H2 + NO',
301     fr={'Arrhenius', A=0.48e13, n=0.0, C=0.0},
302     label='r51'
303 }
304
305 Reaction{
306     'HNO + OH <=> H2O + NO',
307     fr={'Arrhenius', A=0.36e14, n=0.0, C=0.0},
308     label='r52'
309 }
310
311 r53 = 'NO + CO <=> CO2 + N'
312 r54 = 'NO2 + CO <=> NO + CO2'
313
314 Reaction{
315     'NO + HO2 <=> HNO + O2',
316     fr={'Arrhenius', A=0.72e13, n=0.5, C=5500.0},
317     label='r55'
318 }
319
320 Reaction{
321     'HNO + O <=> NO + OH',
322     fr={'Arrhenius', A=0.5e12, n=0.5, C=0.0},
323     label='r56'
324 }
325
326 Reaction{
327     'HNO3 + O <=> HO2 + NO2',
328     fr={'Arrhenius', A=0.10e12, n=0.0, C=0.0},
329     label='r57'
330 }
331
332 Reaction{
```

```
333     'HO2 + NO2 <=> HNO2 + O2',
334     fr={'Arrhenius', A=0.20e12, n=0.0, C=0.0},
335     label='r58'
336 }
337
338 r59 = 'HCO + O2 <=> CO + HO2'
339
340 Reaction{
341     'O3 + HO2 <=> 2 O2 + OH',
342     fr={'Arrhenius', A=0.10e12, n=0.0, C=1409.0},
343     label='r60'
344 }
```