pyJac: analytical Jacobian generator for chemical kinetics

Kyle E. Niemeyer[a,*], Nicholas J. Curtis[b], Chih-Jen Sung[b]

[a] *School of Mechanical, Industrial, and Manufacturing Engineering*
Oregon State University, Corvallis, OR 97331, USA
[b] *Department of Mechanical Engineering*
University of Connecticut, Storrs, CT, 06269, USA

Abstract

Accurate simulations of combustion phenomena require the use of detailed chemical kinetics in order to capture limit phenomena such as ignition and extinction as well as predict pollutant formation. However, the chemical kinetic models for hydrocarbon fuels of practical interest typically have large numbers of species and reactions and exhibit high levels of mathematical stiffness in the governing differential equations, particularly for larger fuel molecules. In order to integrate the stiff equations governing chemical kinetics, generally reactive-flow simulations rely on implicit algorithms that require frequent Jacobian matrix evaluations. Some in situ and a posteriori computational diagnostics methods also require accurate Jacobian matrices, including computational singular perturbation and chemical explosive mode analysis. Typically, finite differences numerically approximate these, but for larger chemical kinetic models this poses significant computational demands since the number of chemical source term evaluations scales with the square of species count. Furthermore, existing analytical Jacobian tools do not optimize evaluations or support emerging SIMD processors such as GPUs. Here we introduce `pyJac`, a Python-based open-source program that generates analytical Jacobian matrices for use in chemical kinetics modeling and analysis. In addition to producing the necessary customized source code for evaluating reaction rates (including all modern reaction rate formulations), the chemical source terms, and the Jacobian matrix, `pyJac` uses an optimized evaluation order to minimize computational and memory operations. As a demonstration, we first establish the correctness of the Jacobian matrices for kinetic models of hydrogen, methane, ethylene, and isopentanol oxidation (number of species ranging 13–360) by showing agreement within 0.001 % of matrices obtained via automatic differentiation. We then demonstrate the performance achievable on CPUs and GPUs using `pyJac` via matrix evaluation timing comparisons; the routines produced by `pyJac` outperformed first-order finite differences by 3–7.5 times and the existing analytical Jacobian software `TChem` by 1.1–2.2 times on a single-threaded basis. It is noted that `TChem` is not thread-safe, while `pyJac` is easily parallelized, and hence can greatly outperform `TChem` on multicore CPUs. The Jacobian matrix generator we describe here will be useful for reducing the cost of integrating chemical source terms with implicit algorithms in particular and algorithms that require an accurate Jacobian matrix in general. Furthermore, the open-source release of the program and Python-based implementation will enable wide

*Corresponding author
Email address: `Kyle.Niemeyer@oregonstate.edu` (Kyle E. Niemeyer)

adoption.

Keywords: Chemical kinetics, Jacobian, SIMD, GPU

PROGRAM SUMMARY

Manuscript Title: `pyJac`: analytical Jacobian generator for chemical kinetics
Authors: Kyle E. Niemeyer, Nicholas J. Curtis, Chih-Jen Sung
Program Title: pyJac
Journal Reference:
Catalogue identifier:
Licensing provisions: MIT License
Programming language: Python
Computer: Windows, Ubuntu, and Mac OS computers
Operating system: Any (Linux, Mac OS X, Windows)
RAM: Problem dependent; typically less than 1 GB
Number of processors used: one
Classification: 16.12, 4.3
External routines/libraries: required: `NumPy`; optional: `Cython`, Cantera, `PyYAML`, `Adept`
Nature of problem: Automatic generation of source code to evaluate Jacobian matrices for chemical kinetic models
Solution method: Chemical kinetic model interpreted from input file(s), and partial derivatives and necessary supporting functions are written to file based on a theoretical derivation.
Additional comments: Current version and support available at https://github.com/SLACKHA/pyjac/

Running time: Problem dependent; from seconds to tens of minutes depending on the model size.

Keywords: Chemical kinetics, Jacobian, SIMD, GPU

1. Introduction

As the need for detailed and accurate chemical kinetic models[1] in predictive reactive-flow simulations has become recognized in recent years, such models describing the oxidation of hydrocarbon fuels simultaneously grew orders of magnitude in size and complexity. For example, a recently developed detailed kinetic model for 2-methylalkanes, relevant for jet and diesel fuel surrogates, consists of over 7000 species and 30000 reactions [1]; similarly large surrogate models exist for gasoline [2, 3] and biodiesel [4]. Since in general the computational cost of solving the associated generally stiff systems of equations scales quadratically with the number of species at best—and at worst, cubically—models of such a large size pose challenges even for lower-dimensional analyses, and cannot practically be used directly in multidimensional reactive-flow simulations.

In an effort to reduce the computational demands of using large, detailed kinetic models, a number of techniques have been developed to reduce their size and complexity while retaining

[1]Note that the term "reaction mechanism" is also used commonly in the literature; we adopted our preferred terminology "chemical kinetic model" here.

predictiveness, as reviewed by Lu and Law [5] as well as Turányi and Tomlin [6]. Major classes of such approaches include skeletal reduction methods that remove unimportant species and reactions [7–10], lumping of species that share similar properties [11–13], and time-scale reduction methods that reduce chemical stiffness [14–17]. Effective reduction strategies either combine multiple methods a priori [18–20] or apply them dynamically during a simulation to achieve greater local savings [21–26]. Techniques such as interpolation/tabulation of expensive terms [27] can also reduce computational costs.

In addition to the aforementioned cost reduction methods that modify the chemical kinetic models, improvements to the integration algorithms that actually solve the governing differential equations can also offer significant gains in computational performance. Due to the chemical stiffness exhibited by most kinetic models, solvers typically rely on robust, high-order implicit integration algorithms based on backward differentiation formulae [28–31]. In order to solve the nonlinear algebraic equations that arise in these methods, the Jacobian matrix must be evaluated and factorized, operations that result in the quadratic and cubic costs mentioned previously. However, by using an analytical formulation for the Jacobian matrix rather than a typical finite difference approximation, the cost of the numerous evaluations can drop from growing with the square of the number of species to a linear dependence [5].

In parallel with the potential improvements in stiff implicit integrators used for chemical kinetics, algorithms tailored for high-performance hardware accelerators offer another route to reducing computational costs. In the past, central processing unit (CPU) clock speeds increased regularly—i.e., Moore's Law—but power consumption and heat dissipation issues disrupted this trend recently, slowing the pace of increases in CPU clock rates. While multicore parallelism continues to raise CPU performance, single-instruction multiple data (SIMD) processors, e.g., graphics processing units (GPUs), recently emerged as a low-cost, low-power consumption, and massively parallel high-performance computing alternative. GPUs—originally developed for graphics/video processing and display purposes—consist of hundreds to thousands of cores, compared to the tens of cores found on a typical CPU. Recognizing that the SIMD parallelism model fits well with the operator-split chemistry integration that forms the basis of many reactive-flow codes [32], a number of studies in recent years [33–38] explored the use of SIMD processors to accelerate the integration of chemical kinetics in reactive-flow codes. Niemeyer and Sung [39] reviewed such efforts in greater detail. While explicit methods offer significant improvements in performance for nonstiff and moderately stiff chemical kinetics [38], experiences thus far suggest that stiff chemistry continues to require the use of implicit or similar algorithms. This provides significant motivation to provide the capability of evaluating analytical Jacobian matrices on GPUs as well as CPUs.

Thus, motivated by the potential cost reductions offered by analytical Jacobian matrix formulations, over the past five years a number of research groups developed analytical Jacobian generators for chemical kinetics, although as will be discussed the software package introduced here offers a number of improvements. The `TChem` toolkit developed by Safta et al. [40] was one of the first software packages developed for calculating the analytical Jacobian matrix, but provides this functionality through an interface rather than generating customized source code for each model. Youssefi [41] recognized the importance of using an analytical Jacobian over a numerical approximation to reduce both computational cost

and numerical error when performing eigendecomposition of the matrix. Bisetti [42] released a utility for producing analytical Jacobian matrix source code based on isothermal and isobaric conditions, with the state vector comprised of species concentrations rather than mass fractions; while incompatible with many existing reactive-flow formulations, this formulation resulted in a significant increase of Jacobian matrix sparsity—such a strategy should be investigated further. Perini et al. [43] developed an analytical Jacobian matrix formulation for constant-volume combustion, and when used in a multidimensional reactive-flow simulation—combined with tabulation of temperature-dependent properties—reported a performance improvement of around 80 % over finite-difference-based approximations. Recently, Dijkmans et al. [44] used a GPU-based analytical Jacobian combined with tabulation of temperature-dependent functions based on polynomial interpolations to accelerate the integration of chemical kinetics equations, similar to the earlier approach of Shi et al. [34]. Unlike the current work, the approach of Dijkmans et al. [44] used the GPU to calculate the elements of a single Jacobian matrix in parallel, rather than a large number of matrices corresponding to different states.

To our knowledge, currently no open-source analytical chemical Jacobian tool exists that is capable of generating code specifically optimized for SIMD processors. To this end, `pyJac` is capable of generating subroutines for reaction rates, species production rates, derivative source terms, and analytical chemical Jacobian matrices both for CPU operation via C/C++ and GPU operation via CUDA [45], a widely used programming language for NVIDIA GPUs. Furthermore, `pyJac` supports newer pressure-dependent reaction formulations (i.e., based on logarithmic or Chebyshev polynomial interpolation).

The rest of the paper is structured as follows. First, in Section 2 we introduce the governing equations for chemical kinetics and then provide the analytical Jacobian matrix formulation in Section 3. Next, Section 4 describes the techniques used to optimize evaluation of the analytical Jacobian on both CPUs and GPUs. Then, in Section 5 we demonstrate the correctness and computational performance of the generated analytical Jacobian matrices for benchmark chemical kinetic models, and discuss the implications of these results. Finally, we summarize our work in Section 6 and outline future research directions.

2. Theory

This section describes the theoretical background of the analytical Jacobian generator, first in terms of the governing equations and then the components of the Jacobian matrix itself. The literature contains more detailed explanations of the governing equations development [46–48], but we include the necessary details here for completeness.

2.1. Governing equations

The initial value problem to be solved, whether in the context of a single homogeneous reacting system (e.g., autoignition, perfectly stirred reactor) or the chemistry portion of an operator-split multidimensional reactive-flow simulation [32], is described using an ordinary differential equation for the thermochemical composition vector

$$\Phi = \left\{T, Y_1, Y_2, \dots, Y_{N_{\text{sp}}-1}\right\}^{\intercal} \ , \tag{1}$$

where T is the temperature, Y_i are the species mass fractions, and N_{sp} is the number of species. The mass fraction of the final species, $Y_{N_{\text{sp}}}$, is determined through conservation of mass:

$$Y_{N_{\text{sp}}} = 1 - \sum_{k=1}^{N_{\text{sp}}-1} Y_k \ , \tag{2}$$

where the most abundant species (e.g., N_2 in air-fed combustion) can be assigned to this role. In multidimensional simulations where the equations for chemical kinetics are coupled to conservation of energy (or enthalpy), temperature can be determined algebraically in a straightforward manner [32]. Completely defining the thermodynamic state also requires either pressure (p) or density (ρ), related to temperature and the mixture composition through the ideal equation of state

$$p = \rho \frac{\mathcal{R}}{W} T = \mathcal{R} T \sum_{k=1}^{N_{\text{sp}}} [X_k] \ , \tag{3}$$

where $\mathcal{R}$ is the universal gas constant, W is the average molecular weight of the mixture, and $[X_k]$ is the molar concentration of the kth species. In the current implementation of `pyJac`, we assume a constant-pressure system[2]; thus, we use Eq. (3) to determine density rather than including it in the differential system given by Eq. (1). The average molecular weight is defined by

$$W = \frac{1}{\sum_{k=1}^{N_{\text{sp}}} Y_k / W_k} = \frac{\rho \mathcal{R} T}{p} \tag{4}$$

and the molar concentrations by

$$[X_k] = \rho \frac{Y_k}{W_k} \ , \tag{5}$$

where W_k is the molecular weight of the kth species.

The system of ODEs governing the time change in thermochemical composition corresponding to Eq. (1) is then $f = \partial \Phi / \partial t$:

$$f = \frac{\partial \Phi}{\partial t} = \left\{ \frac{\partial T}{\partial t}, \frac{\partial Y_1}{\partial t}, \frac{\partial Y_2}{\partial t}, \ldots, \frac{\partial Y_{N_{\text{sp}}-1}}{\partial t} \right\}^{\intercal} \ , \tag{6}$$

where

$$\frac{\partial T}{\partial t} = \frac{-1}{\rho c_p} \sum_{k=1}^{N_{\text{sp}}} h_k W_k \dot{\omega}_k \ , \tag{7}$$

$$\frac{\partial Y_k}{\partial t} = \frac{W_k}{\rho} \dot{\omega}_k \quad k = 1, \ldots, N_{\text{sp}} - 1 \ , \tag{8}$$

where c_p is the mass-averaged constant-pressure specific heat, h_k is the enthalpy of the kth species in mass units, and $\dot{\omega}_k$ is the kth species overall production rate.

[2] In the context of multidimensional reactive flows, the constant-pressure assumption signifies that pressure remains constant during the reaction substep of an operator-split scheme (and is thus an input variable to the kinetics problem), not that the pressure is fixed throughout the entire simulation. This assumption is commonly used in low-speed combustion simulations.

2.2. Thermodynamic properties

The standard-state thermodynamic properties (in molar units) for a gaseous species k is defined using the standard seven-coefficient polynomial of Gordon and McBride [49]:

$$\frac{C^\circ_{p,k}}{\mathcal{R}} = a_{0,k} + T\left(a_{1,k} + T\left(a_{2,k} + T\left(a_{3,k} + a_{4,k}T\right)\right)\right) \tag{9}$$

$$\frac{H^\circ_k}{\mathcal{R}} = T\left(a_{0,k} + T\left(\frac{a_{1,k}}{2} + T\left(\frac{a_{2,k}}{3} + T\left(\frac{a_{3,k}}{4} + \frac{a_{4,k}}{5}T\right)\right)\right)\right) + a_{5,k} \tag{10}$$

$$\frac{S^\circ_k}{\mathcal{R}} = a_{0,k}\ln T + T\left(a_{1,k} + T\left(\frac{a_{2,k}}{2} + T\left(\frac{a_{3,k}}{3} + \frac{a_{4,k}}{4}T\right)\right)\right) + a_{6,k} \tag{11}$$

where $C_{p,k}$ is the constant-pressure specific heat in molar units, H_k is the enthalpy in molar units, S_k is the entropy in molar units, and the $^\circ$ indicates a standard-state property at one atmosphere (equivalent to the property at any pressure for calorically perfect gases).

The mass-based specific heat and enthalpy are then defined as

$$c_{p,k} = \frac{C_{p,k}}{W_k} \quad \text{and} \quad h_k = \frac{H_k}{W_k}\ , \tag{12}$$

and the mixture-averaged specific heat is

$$c_p = \sum_{k=1}^{N_{\text{sp}}} Y_k c_{p,k}\ . \tag{13}$$

2.3. Reaction rate expressions

Next, we define the species rates of production and related kinetic terms as

$$\dot{\omega}_k = \sum_{i=1}^{N_{\text{reac}}} \nu_{ki} q_i\ , \tag{14}$$

where N_{reac} is the number of reactions, ν_{ki} is the overall stoichiometric coefficient for species k in reaction i, and q_i is the rate-of-progress for reaction i. These are defined by

$$\nu_{ki} = \nu''_{ki} - \nu'_{ki} \quad \text{and} \tag{15}$$

$$q_i = c_i R_i\ , \tag{16}$$

where ν''_{ki} and ν'_{ki} are the product and reactant stoichiometric coefficients (respectively) of species k in reaction i. The base rate-of-progress for the ith reversible reaction R_i is given by

$$R_i = R_{f,i} - R_{r,i}\ , \tag{17}$$

$$R_{f,i} = k_{f,i}\prod_{j=1}^{N_{\text{sp}}}[X_j]^{\nu'_{ji}}\ , \text{ and} \tag{18}$$

$$R_{r,i} = k_{r,i}\prod_{j=1}^{N_{\text{sp}}}[X_j]^{\nu''_{ji}}\ , \tag{19}$$

where $k_{f,i}$ and $k_{r,i}$ are the forward and reverse reaction rate coefficients for the ith reaction, respectively, and the third-body/pressure modification c_i is given by

$$c_i = \begin{cases} 1 & \text{for elementary reactions,} \\ [X]_i & \text{for third-body enhanced reactions,} \\ \dfrac{P_{r,i}}{1+P_{r,i}} F_i & \text{for unimolecular/recombination falloff reactions, and} \\ \dfrac{1}{1+P_{r,i}} F_i & \text{for chemically-activated bimolecular reactions,} \end{cases} \tag{20}$$

where for the ith reaction $[X]_i$ is the third-body concentration, $P_{r,i}$ is the reduced pressure, and F_i is the falloff blending factor. These terms are defined in the following sections.

The forward reaction rate coefficient $k_{f,i}$ is given by the three-parameter Arrhenius expression:

$$k_{f,i} = A_i T^{\beta_i} \exp\left(-\frac{T_{a,i}}{T}\right) , \tag{21}$$

where A_i is the pre-exponential factor, β_i is the temperature exponent, and $T_{a,i}$ is the activation temperature given by $T_{a,i} = E_{a,i}/\mathcal{R}$.

As given by Lu and Law [5], depending on the value of the Arrhenius parameters, $k_{f,i}$ can be calculated in different ways to minimize the computational cost:

$$k_{f,i} = \begin{cases} A_i & \text{if } \beta = 0 \text{ and } T_{a,i} = 0 , \\ \exp\left(\log A_i + \beta_i \log T\right) & \text{if } \beta_i \neq 0 \text{ and } T_{a,i} = 0 , \\ \exp\left(\log A_i + \beta_i \log T - T_{a,i}/T\right) & \text{if } \beta_i \neq 0 \text{ and } T_{a,i} \neq 0 , \\ \exp\left(\log A_i - T_{a,i}/T\right) & \text{if } \beta_i = 0 \text{ and } T_{a,i} \neq 0 , \text{ and} \\ A_i \prod^{\beta_i} T & \text{if } T_{a,i} = 0 \text{ and } \beta_i \in \mathbb{Z} , \end{cases} \tag{22}$$

where $\mathbb{Z}$ is the set of integers.

2.3.1. Reverse rate coefficient

By definition, irreversible reactions have a zero reverse rate coefficient $k_{r,i}$, while reversible reactions have nonzero $k_{r,i}$. For reversible reactions, the reverse rate coefficient is determined in one of two ways: (1) via explicit reverse Arrhenius parameters as with the forward rate coefficient—thus following the same expression as Eq. (21)—or (2) via the ratio of the forward rate coefficient and the equilibrium constant,

$$k_{r,i} = \frac{k_{f,i}}{K_{c,i}} , \tag{23}$$

$$K_{c,i} = K_{p,i} \left(\frac{p_{\mathrm{atm}}}{\mathcal{R}T}\right)^{\sum_{k=1}^{N_{\mathrm{sp}}} \nu_{ki}} , \text{ and} \tag{24}$$

$$K_{p,i} = \exp\left(\frac{\Delta S_i^\circ}{\mathcal{R}} - \frac{\Delta H_i^\circ}{\mathcal{R}T}\right) = \exp\left(\sum_{k=1}^{N_{\mathrm{sp}}} \nu_{ki} \left(\frac{S_k^\circ}{\mathcal{R}} - \frac{H_k^\circ}{\mathcal{R}T}\right)\right) , \tag{25}$$

where p_{atm} is the pressure of one standard atmosphere in the appropriate units.

By combining the expressions for $K_{c,i}$ and $K_{p,i}$, we obtain

$$K_{c,i} = \left(\frac{p_{\mathrm{atm}}}{\mathcal{R}}\right)^{\sum_{k=1}^{N_{\mathrm{sp}}} \nu_{ki}} \exp\left(\sum_{k=1}^{N_{\mathrm{sp}}} \nu_{ki} B_k\right) , \tag{26}$$

where, expanding the polynomial expressions for S_k° and H_k° from Eqs. (11) and (10), respectively,

$$\begin{aligned} B_k = {} & a_{6,k} - a_{0,k} + (a_{0,k} - 1)\ln T \\ & + T\left(\frac{a_{1,k}}{2} + T\left(\frac{a_{2,k}}{6} + T\left(\frac{a_{3,k}}{12} + \frac{a_{4,k}}{20}T\right)\right)\right) - \frac{a_{5,k}}{T} . \end{aligned} \tag{27}$$

2.3.2. Third-body effects

For a reaction enhanced (or diminished) by the presence of a third body, the reaction rate is modified by the third-body concentration $[X]_i$ given by

$$[X]_i = \sum_{j=1}^{N_{\mathrm{sp}}} \alpha_{ij}[X_j] , \tag{28}$$

where α_{ij} is the third-body efficiency of species j in the ith reaction. If all species in the mixture contribute equally as third bodies (the default) then $\alpha_{ij} = 1$ for all species. In this case,

$$[X]_i = [M] = \sum_{j=1}^{N_{\mathrm{sp}}} [X_j] = \frac{p}{\mathcal{R}T} = \frac{\rho}{W} . \tag{29}$$

In addition, a single species may act as the third body, in which case

$$[X]_i = [X_m] , \tag{30}$$

where the mth species is the third body.

2.3.3. Falloff reactions

Unlike elementary and third-body reactions, falloff reactions exhibit a pressure dependence described as a blending of rates at low- and high-pressure limits; thus, the rate coefficients depend on a mixture of low-pressure- ($k_{0,i}$) and high-pressure-limit ($k_{\infty,i}$) coefficients, each with corresponding Arrhenius parameters and expressed using Eq. (21). The ratio of the coefficients $k_{0,i}$ and $k_{\infty,i}$, combined with the third-body concentration (based on either the mixture as a whole including any efficiencies $\alpha_{i,j}$, or a specific species), define a reduced pressure $P_{r,i}$ given by

$$P_{r,i} = \begin{cases} \dfrac{k_{0,i}}{k_{\infty,i}}[X]_i & \text{for the mixture as the third body, or} \\ \dfrac{k_{0,i}}{k_{\infty,i}}[X_m] & \text{for a specific species } m \text{ as third body.} \end{cases} \tag{31}$$

The falloff blending factor F_i used in Eq. (20) is determined based on one of three representations: the Lindemann [50], Troe [51], and SRI [52] falloff approaches

$$F_i = \begin{cases} 1 & \text{for Lindemann,} \\ F_{\text{cent}}^{\left(1+(A_{\text{Troe}}/B_{\text{Troe}})^2\right)^{-1}} & \text{for Troe, or} \\ dT^e \left(a \cdot \exp\left(-\dfrac{b}{T}\right) + \exp\left(-\dfrac{T}{c}\right)\right)^X & \text{for SRI.} \end{cases} \tag{32}$$

The variables for the Troe representation are given by

$$F_{\text{cent}} = (1-a)\exp\left(-\frac{T}{T^{***}}\right) + a \cdot \exp\left(-\frac{T}{T^{*}}\right) + \exp\left(-\frac{T^{**}}{T}\right) \ , \tag{33}$$

$$A_{\text{Troe}} = \log_{10} P_{r,i} - 0.67 \log_{10} F_{\text{cent}} - 0.4 \ , \text{and} \tag{34}$$

$$B_{\text{Troe}} = 0.806 - 1.1762 \log_{10} F_{\text{cent}} - 0.14 \log_{10} P_{r,i} \ , \tag{35}$$

where a, T^{***}, T^{*}, and T^{**} are specified parameters. The final parameter T^{**} is optional, and, if it is not used, the final term of F_{cent} is omitted. The exponent used in the SRI representation is given by

$$X = \left(1 + (\log_{10} P_{r,i})^2\right)^{-1} \tag{36}$$

where a, b, and c are required parameters. The parameters d and e are optional; if not specified, $d = 1$ and $e = 0$.

2.3.4. Pressure-dependent reactions

In addition to the falloff approach given previously, two additional formulations can be used to describe the pressure dependence of reactions that do not follow the modification factor c_i approach. The first involves logarithmic interpolation between Arrhenius rates at two pressures [53, 54], each evaluated using Eq. (21):

$$k_1(T) = A_1 T^{\beta_1} \exp\left(-\frac{T_{a,1}}{T}\right) \ \text{at } p_1 \text{ and} \tag{37}$$

$$k_2(T) = A_2 T^{\beta_2} \exp\left(-\frac{T_{a,2}}{T}\right) \ \text{at } p_2 \ , \tag{38}$$

where the Arrhenius coefficients are given for each pressure p_1 and p_2. Then, the reaction rate coefficient at a particular pressure p between p_1 and p_2 can be determined through logarithmic interpolation:

$$\log k_f(T,p) = \log k_1(T) + (\log k_2(T) - \log k_1(T)) \frac{\log p - \log p_1}{\log p_2 - \log p_1} \ . \tag{39}$$

In addition, the pressure dependence of a reaction can be expressed through a bivariate Chebyshev polynomial fit [53–57]:

$$\log_{10} k_f(T,p) = \sum_{i=1}^{N_T} \sum_{j=1}^{N_p} \eta_{ij} \phi_i(\tilde{T}) \phi_j(\tilde{p}) \ , \tag{40}$$

where η_{ij} is the coefficient corresponding to the grid points i and j, N_T and N_p are the numbers of grid points for temperature and pressure, respectively, and ϕ_n is the Chebyshev polynomial of the first kind of degree $n-1$ typically expressed as

$$\phi_n(x) = \mathcal{T}_{n-1}(x) = \cos\left((n-1)\arccos(x)\right) \quad \text{for } |x| \leq 1 \,. \tag{41}$$

The reduced temperature $\tilde{T}$ and pressure $\tilde{p}$ are given by

$$\tilde{T} \equiv \frac{2T^{-1} - T_{\min}^{-1} - T_{\max}^{-1}}{T_{\max}^{-1} - T_{\min}^{-1}} \quad \text{and} \tag{42}$$

$$\tilde{p} \equiv \frac{2\log_{10} p - \log_{10} p_{\min} - \log_{10} p_{\max}}{\log_{10} p_{\max} - \log_{10} p_{\min}} \,, \tag{43}$$

where $T_{\min} \leq T \leq T_{\max}$ and $p_{\min} \leq p \leq p_{\max}$ describe the ranges of validity for temperature and pressure.

3. Jacobian matrix formulation

Next, we detail the construction of the Jacobian matrix, and provide a simple method for evaluating it efficiently by calculating elements in the appropriate order. More sophisticated approaches for reducing the cost of evaluating the matrix will be proposed in Section 4.

3.1. Elements of the Jacobian matrix

Let $\mathcal{J}$ denote the Jacobian matrix corresponding to the vector of ODEs given by Eq. (6). $\mathcal{J}$ is filled by the partial derivatives $\partial f/\partial\Phi$, such that

$$\mathcal{J}_{i,j} = \frac{\partial f_i}{\partial \Phi_j} \,. \tag{44}$$

The first line of $\mathcal{J}$ is filled with partial derivatives of the energy equation, or

$$\mathcal{J}_{1,j} = \frac{\partial \dot{T}}{\partial \Phi_j} \quad j = 1, \ldots, N_{\text{sp}} - 1 \,. \tag{45}$$

The components of $\mathcal{J}_{1,j}$ are:

$$\mathcal{J}_{1,1} = \frac{\partial f_1}{\partial T} = \frac{-1}{c_p} \sum_{k=1}^{N_{\text{sp}}} \left[\left(\frac{\partial h_k}{\partial T} - h_k \frac{\partial c_p}{\partial T} \right) \frac{W_k \dot{\omega}_k}{\rho} + \frac{h_k}{c_p} \frac{\partial}{\partial T} \left(\frac{W_k \dot{\omega}_k}{\rho} \right) \right], \tag{46}$$

$$\mathcal{J}_{1,j+1} = \frac{\partial f_1}{\partial Y_j} = \frac{-1}{c_p} \sum_{k=1}^{N_{\text{sp}}} \left[\left(\frac{\partial h_k}{\partial Y_j} - h_k \frac{\partial c_p}{\partial Y_j} \right) \frac{W_k \dot{\omega}_k}{\rho} + \frac{h_k}{c_p} \frac{\partial}{\partial Y_j} \left(\frac{W_k \dot{\omega}_k}{\rho} \right) \right], \tag{47}$$

for $j = 1, \ldots, N_{\text{sp}} - 1$.

The remaining lines of $\mathcal{J}$ are filled with the partial derivatives of the species equations, with the components

$$\mathcal{J}_{k+1,1} = \frac{\partial f_{k+1}}{\partial T} = \frac{W_k}{\rho} \left(\frac{\partial \dot{\omega}_k}{\partial T} - \frac{\dot{\omega}_k}{\rho} \frac{\partial \rho}{\partial T} \right) \text{ and} \tag{48}$$

$$\mathcal{J}_{k+1,j+1} = \frac{\partial f_{k+1}}{\partial Y_j} = \frac{W_k}{\rho}\left(\frac{\partial \dot{\omega}_k}{\partial Y_j} - \frac{\dot{\omega}_k}{\rho}\frac{\partial \rho}{\partial Y_j}\right), \tag{49}$$

for $k = 1, \ldots, N_{\text{sp}} - 1$ and $j = 1, \ldots, N_{\text{sp}} - 1$.

The partial derivatives of ρ, c_p, h_k, and $\dot{\omega}_k$ with respect to temperature and mass fraction need to be evaluated. The partial derivatives of density are

$$\frac{\partial \rho}{\partial T} = \frac{\partial}{\partial T}\left(\frac{pW}{\mathcal{R}T}\right) \text{ and } = -\frac{\rho}{T} \tag{50}$$

$$\frac{\partial \rho}{\partial Y_j} = \frac{\partial}{\partial Y_j}\left(\frac{pW}{\mathcal{R}T}\right) = -\rho W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\right) . \tag{51}$$

The partial derivative of species specific enthalpy with respect to temperature is simply constant-pressure specific heat, and the partial derivative with respect to species mass fraction is zero:

$$\frac{\partial h_k}{\partial T} = c_{p,k} \tag{52}$$

$$\frac{\partial h_k}{\partial Y_j} = 0 . \tag{53}$$

The derivatives of specific heat are expressed by

$$\frac{\partial c_p}{\partial T} = \sum_{k=1}^{N_{\text{sp}}} Y_k \frac{\partial c_{p,k}}{\partial T} , \tag{54}$$

$$\frac{\partial c_{p,k}}{\partial T} = \frac{\mathcal{R}}{W_k}\left(a_{1,k} + T\left(2a_{2,k} + T\left(3a_{3,k} + 4a_{4,k}T\right)\right)\right) , \text{ and} \tag{55}$$

$$\frac{\partial c_p}{\partial Y_j} = \frac{\partial}{\partial Y_j}\sum_{k=1}^{N_{\text{sp}}} Y_k c_{p,k} = c_{p,j} - c_{p,N_{\text{sp}}} . \tag{56}$$

Next, the derivatives of species rate-of-production $\dot{\omega}_k$ are given by

$$\frac{\partial \dot{\omega}_k}{\partial T} = \sum_{i=1}^{N_{\text{reac}}} \nu_{ki}\frac{\partial q_i}{\partial T} = \sum_{i=1}^{N_{\text{reac}}} \nu_{ki}\left(\frac{\partial c_i}{\partial T}R_i + c_i\frac{\partial R_i}{\partial T}\right) \text{ and} \tag{57}$$

$$\frac{\partial \dot{\omega}_k}{\partial Y_j} = \sum_{i=1}^{N_{\text{reac}}} \nu_{ki}\frac{\partial q_i}{\partial Y_j} = \sum_{i=1}^{N_{\text{reac}}} \nu_{ki}\left(\frac{\partial c_i}{\partial Y_j}R_i + c_i\frac{\partial R_i}{\partial Y_j}\right) . \tag{58}$$

The partial derivatives of R_i vary depending on whether the reaction is reversible. For irreversible reactions,

$$\frac{\partial R_{f,i}}{\partial T} = \frac{\partial k_{f,i}}{\partial T}\frac{1}{k_{f,i}}R_{f,i} + k_{f,i}\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\text{sp}}}[X_j]^{\nu'_{ji}}\right) \text{ and} \tag{59}$$

$$\frac{\partial R_{f,i}}{\partial Y_j} = k_{f,i}\frac{\partial}{\partial Y_j}\left(\prod_{k=1}^{N_{\text{sp}}}[X_k]^{\nu'_{ki}}\right) . \tag{60}$$

The partial derivatives of species concentration are

$$\frac{\partial [X_i]}{\partial T} = \frac{Y_i}{W_i}\frac{\partial \rho}{\partial T} = -\frac{[X_i]}{T} \tag{61}$$

$$\frac{\partial [X_i]}{\partial Y_j} = \frac{Y_i}{W_i}\frac{\partial \rho}{\partial Y_j} + \frac{\rho}{W_i}\frac{\partial Y_i}{\partial Y_j} = -[X_i]W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right) + \left(\delta_{ij} - \delta_{iN_{\rm sp}}\right)\frac{\rho}{W_i}\ , \tag{62}$$

where $i = 1, \ldots, N_{\rm sp}$, $j = 1, \ldots, N_{\rm sp}-1$, and δ_{ij} is the Kronecker delta. The partial derivatives of the molar concentration product terms are then

$$\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}}\right) = -\frac{1}{T}\left(\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}}\right)\sum_{j=1}^{N_{\rm sp}}\nu'_{ji} \quad \text{and} \tag{63}$$

$$\begin{aligned}\frac{\partial}{\partial Y_j}\left(\prod_{k=1}^{N_{\rm sp}}[X_k]^{\nu'_{ki}}\right) = \sum_{k=1}^{N_{\rm sp}}\nu'_{ki}\Bigg(&-[X_k]^{\nu'_{ki}}W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right)\\ &+ \left(\delta_{kj} - \delta_{kN_{\rm sp}}\right)\frac{\rho}{W_k}[X_k]^{\nu'_{ki}-1}\Bigg)\prod_{\substack{l=1\\ l\neq k}}^{N_{\rm sp}}[X_l]^{\nu'_{li}}\ ,\end{aligned} \tag{64}$$

and the partial derivative of forward reaction rate coefficient is

$$\frac{\partial k_{f,i}}{\partial T} = \frac{k_{f,i}}{T}\left(\beta_i + \frac{T_{a,i}}{T}\right)\ . \tag{65}$$

Inserting these into Eqs. (59) and (60) gives

$$\frac{\partial R_{f,i}}{\partial T} = \frac{R_{f,i}}{T}\left(\beta_i + \frac{T_{a,i}}{T} - \sum_{j=1}^{N_{\rm sp}}\nu'_{ji}\right) \quad \text{and} \tag{66}$$

$$\begin{aligned}\frac{\partial R_{f,i}}{\partial Y_j} = \sum_{k=1}^{N_{\rm sp}}\nu'_{ki}\Bigg(&-R_{f,i}W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right)\\ &+ \left(\delta_{kj} - \delta_{kN_{\rm sp}}\right)k_{f,i}\frac{\rho}{W_k}[X_k]^{\nu'_{ki}-1}\prod_{\substack{l=1\\ l\neq k}}^{N_{\rm sp}}[X_l]^{\nu'_{li}}\Bigg)\ .\end{aligned} \tag{67}$$

For reversible reactions with explicit reverse Arrhenius coefficients,

$$\begin{aligned}\frac{\partial R_i}{\partial T} = &\frac{\partial k_{f,i}}{\partial T}\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}} + k_{f,i}\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}}\right)\\ &- \frac{\partial k_{r,i}}{\partial T}\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu''_{ji}} - k_{r,i}\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu''_{ji}}\right)\end{aligned} \tag{68}$$

$$= \frac{R_{f,i}}{T}\left(\beta_{f,i} + \frac{T_{af,i}}{T} - \sum_{j=1}^{N_{\rm sp}}\nu'_{ji}\right) - \frac{R_{r,i}}{T}\left(\beta_{r,i} + \frac{T_{ar,i}}{T} - \sum_{j=1}^{N_{\rm sp}}\nu''_{ji}\right)\ . \tag{69}$$

Similarly, $\frac{1}{[X_j]}\left(\nu'_{ji}R_{f,i} - \nu''_{ji}R_{r,i}\right)$ should be avoided since $[X_j]$ could be zero.

The partial derivative of the reverse reaction rate coefficient, when evaluated using the forward rate coefficient and equilibrium constant using Eq. (23), is more complicated:

$$\frac{\partial k_{r,i}}{\partial T} = \frac{\partial}{\partial T}\left(\frac{k_{f,i}}{K_{c,i}}\right) = \frac{\frac{\partial k_{f,i}}{\partial T}K_{c,i} - \frac{\partial K_{c,i}}{\partial T}k_{f,i}}{K_{c,i}^2} \tag{70}$$

$$= \frac{k_{f,i}}{K_{c,i}}\frac{1}{T}\left(\beta_i + \frac{T_{a,i}}{T}\right) - \frac{1}{K_{c,i}}\frac{\partial K_{c,i}}{\partial T}\frac{k_{f,i}}{K_{c,i}} \tag{71}$$

$$\frac{\partial K_{c,i}}{\partial T} = K_{c,i}\sum_{k=1}^{N_{\text{sp}}}\nu_{ki}\frac{\partial B_k}{\partial T} \tag{72}$$

$$\therefore \frac{\partial k_{r,i}}{\partial T} = k_{r,i}\left(\frac{1}{T}\left(\beta_i + \frac{T_{a,i}}{T}\right) - \sum_{k=1}^{N_{\text{sp}}}\nu_{ki}\frac{\partial B_k}{\partial T}\right)\ , \tag{73}$$

where

$$\frac{\partial B_k}{\partial T} = \frac{1}{T}\left(a_{0,k} - 1 + \frac{a_{5,k}}{T}\right) + \frac{a_{1,k}}{2} + T\left(\frac{a_{2,k}}{3} + T\left(\frac{a_{3,k}}{4} + \frac{a_{4,k}}{5}T\right)\right)\ . \tag{74}$$

Now, the partial derivative of R_i with respect to temperature is

$$\frac{\partial R_i}{\partial T} = \frac{R_{f,i}}{T}\left(\beta_i + \frac{T_{a,i}}{T} - \sum_{j=1}^{N_{\text{sp}}}\nu'_{ji}\right) - R_{r,i}\left(\frac{1}{T}\left(\beta_i + \frac{T_{a,i}}{T} - \sum_{j=1}^{N_{\text{sp}}}\nu''_{ji}\right) - \sum_{j=1}^{N_{\text{sp}}}\nu_{ji}\frac{\partial B_j}{\partial T}\right)\ . \tag{75}$$

For all reversible reactions,

$$\begin{aligned}\frac{\partial R_i}{\partial Y_j} = \sum_{k=1}^{N_{\text{sp}}}\Bigg(&\nu'_{ki}\Bigg(-R_{f,i}W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\right) \\ &+ \left(\delta_{kj} - \delta_{kN_{\text{sp}}}\right)k_{f,i}\frac{\rho}{W_k}[X_k]^{\nu'_{ki}-1}\prod_{\substack{l=1\\l\neq k}}^{N_{\text{sp}}}[X_l]^{\nu'_{li}}\Bigg) \\ &- \nu''_{ki}\Bigg(-R_{r,i}W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\right) \\ &+ \left(\delta_{kj} - \delta_{kN_{\text{sp}}}\right)k_{r,i}\frac{\rho}{W_k}[X_k]^{\nu''_{ki}-1}\prod_{\substack{l=1\\l\neq k}}^{N_{\text{sp}}}[X_l]^{\nu''_{li}}\Bigg)\Bigg)\ .\end{aligned} \tag{76}$$

The partial derivatives of c_i depend on the type of reaction. For elementary reactions,

$$\frac{\partial c_i}{\partial T} = 0 \text{ and} \tag{77}$$

$$\frac{\partial c_i}{\partial Y_j} = 0\ . \tag{78}$$

For third-body-enhanced reactions,

$$\frac{\partial c_i}{\partial T} = -\frac{c_i}{T} \text{ , and} \tag{79}$$

$$\frac{\partial c_i}{\partial Y_j} = -W c_i\Big(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\Big) + \rho\Big(\frac{\alpha_{ij}}{W_j} - \frac{\alpha_{iN_{\text{sp}}}}{W_{N_{\text{sp}}}}\Big) \,. \tag{80}$$

Note that in the case that all species contribute equally (i.e., $\alpha_{ij} = 1$ for all species j), the latter partial derivative simplifies to

$$\frac{\partial c_i}{\partial Y_j} = \frac{\partial}{\partial Y_j}\left(\frac{p}{\mathcal{R}T}\right) = 0 \,, \tag{81}$$

because $\frac{\partial p}{\partial Y_j} = 0$ (shown in Appendix B).

In the case of unimolecular/recombination fall-off reactions,

$$\frac{\partial c_i}{\partial T} = c_i\left(\frac{1}{P_{r,i}(1+P_{r,i})}\frac{\partial P_{r,i}}{\partial T} + \frac{1}{F_i}\frac{\partial F_i}{\partial T}\right) \tag{82}$$

$$\frac{\partial c_i}{\partial Y_j} = c_i\left(\frac{1}{P_{r,i}(1+P_{r,i})}\frac{\partial P_{r,i}}{\partial Y_j} + \frac{1}{F_i}\frac{\partial F_i}{\partial Y_j}\right) \,. \tag{83}$$

The partial derivatives for $P_{r,i}$ are

$$\frac{\partial P_{r,i}}{\partial T} = \frac{P_{r,i}}{T}\left(\beta_{0,i} - \beta_{\infty,i} + \frac{T_{a0,i} - T_{a\infty,i}}{T} - 1\right) \tag{84}$$

$$\frac{\partial P_{r,i}}{\partial Y_j} = \begin{cases} -P_{r,i}W\Big(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\Big) + \frac{k_{0,i}}{k_{\infty,i}}\rho\Big(\frac{\alpha_{ij}}{W_j} - \frac{\alpha_{iN_{\text{sp}}}}{W_{N_{\text{sp}}}}\Big) \text{ , or} \\ -P_{r,i}W\Big(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\Big) + \frac{k_{0,i}}{k_{\infty,i}}\frac{\rho}{W_m}\left(\delta_{mj} - \delta_{mN_{\text{sp}}}\right) \text{, if } [X]_i = [X_m] \,. \end{cases} \tag{85}$$

Similarly, for chemically-activated bimolecular reactions

$$\frac{\partial c_i}{\partial T} = c_i\left(\frac{-1}{1+P_{r,i}}\frac{\partial P_{r,i}}{\partial T} + \frac{1}{F_i}\frac{\partial F_i}{\partial T}\right) \tag{86}$$

$$\frac{\partial c_i}{\partial Y_j} = c_i\left(\frac{-1}{1+P_{r,i}}\frac{\partial P_{r,i}}{\partial Y_j} + \frac{1}{F_i}\frac{\partial F_i}{\partial Y_j}\right) \tag{87}$$

The partial derivatives of F_i depend on the representation of pressure dependence. For Lindemann reactions,

$$\frac{\partial F_i}{\partial T} = 0 \text{ and} \tag{88}$$

$$\frac{\partial F_i}{\partial Y_j} = 0 \,. \tag{89}$$

For reactions using the Troe falloff formulation,

$$\frac{\partial F_i}{\partial T} = \frac{\partial F_i}{\partial F_{\text{cent}}}\frac{\partial F_{\text{cent}}}{\partial T} + \frac{\partial F_i}{\partial P_{r,i}}\frac{\partial P_{r,i}}{\partial T} \text{ and} \tag{90}$$

$$\frac{\partial F_i}{\partial Y_j} = \frac{\partial F_i}{\partial P_{r,i}}\frac{\partial P_{r,i}}{\partial Y_j} \,, \tag{91}$$

where

$$\frac{\partial F_i}{\partial F_{\text{cent}}} = F_i \left(\frac{1}{F_{\text{cent}} \left(1 + (A_{\text{Troe}}/B_{\text{Troe}})^2\right)} - \ln F_{\text{cent}} \frac{2A_{\text{Troe}}}{B_{\text{Troe}}^3} \frac{\frac{\partial A_{\text{Troe}}}{\partial F_{\text{cent}}} B_{\text{Troe}} - A_{\text{Troe}} \frac{\partial B_{\text{Troe}}}{\partial F_{\text{cent}}}}{\left(1 + (A_{\text{Troe}}/B_{\text{Troe}})^2\right)^2} \right), \tag{92}$$

$$\frac{\partial F_{\text{cent}}}{\partial T} = -\frac{1-a}{T^{***}} \exp\left(\frac{-T}{T^{***}}\right) - \frac{a}{T^*} \exp\left(\frac{-T}{T^*}\right) + \frac{T^{**}}{T^2} \exp\left(\frac{-T^{**}}{T}\right), \tag{93}$$

$$\frac{\partial F_i}{\partial P_{r,i}} = -F_i \ln F_{\text{cent}} \frac{2A_{\text{Troe}}}{B_{\text{Troe}}^3} \frac{\frac{\partial A_{\text{Troe}}}{\partial P_{r,i}} B_{\text{Troe}} - A_{\text{Troe}} \frac{\partial B_{\text{Troe}}}{\partial P_{r,i}}}{\left(1 + (A_{\text{Troe}}/B_{\text{Troe}})^2\right)^2}, \tag{94}$$

$\frac{\partial P_{r,i}}{\partial T}$ is given by Eq. (84), $\frac{\partial P_{r,i}}{\partial Y_j}$ is given by Eq. (85), and

$$\frac{\partial A_{\text{Troe}}}{\partial F_{\text{cent}}} = \frac{-0.67}{F_{\text{cent}} \log 10} \qquad \frac{\partial B_{\text{Troe}}}{\partial F_{\text{cent}}} = \frac{-1.1762}{F_{\text{cent}} \log 10} \tag{95}$$

$$\frac{\partial A_{\text{Troe}}}{\partial P_{r,i}} = \frac{1}{P_{r,i} \log 10} \qquad \frac{\partial B_{\text{Troe}}}{\partial P_{r,i}} = \frac{-0.14}{P_{r,i} \log 10}. \tag{96}$$

Finally, for falloff reactions described with the SRI formulation,

$$\frac{\partial F_i}{\partial T} = F_i \left(\frac{e}{T} + X \frac{\frac{ab}{T^2} \exp\left(\frac{-b}{T}\right) - \frac{1}{c} \exp\left(\frac{-T}{c}\right)}{a \cdot \exp\left(\frac{-b}{T}\right) + \exp\left(\frac{-T}{c}\right)} + \frac{\partial X}{\partial P_{r,i}} \frac{\partial P_{r,i}}{\partial T} \log\left(a \cdot \exp\left(\frac{-b}{T}\right) + \exp\left(\frac{-T}{c}\right)\right) \right) \tag{97}$$

$$\frac{\partial F_i}{\partial Y_j} = F_i \frac{\partial X}{\partial Y_j} \log\left(a \cdot \exp\left(\frac{-b}{T}\right) + \exp\left(\frac{-T}{c}\right)\right) \tag{98}$$

where

$$\frac{\partial X}{\partial P_{r,i}} = -X^2 \frac{2 \log_{10} P_{r,i}}{P_{r,i} \log 10}, \tag{99}$$

$$\frac{\partial X}{\partial Y_j} = \frac{\partial X}{\partial P_{r,i}} \frac{\partial P_{r,i}}{\partial Y_j}, \tag{100}$$

$\frac{\partial P_{r,i}}{\partial T}$ is given by Eq. (84), and $\frac{\partial P_{r,i}}{\partial Y_j}$ is given by Eq. (85). Note that for both falloff and chemically activated bimolecular reactions, regardless of falloff parameterization, if all species contribute equally to the third-body concentration $[X]_i$ then the partial derivative of c_i with respect to species mass fraction Y_j is zero, since $\frac{\partial p}{\partial Y_j} = 0$.

The contributions of the logarithmic and Chebyshev pressure-dependent descriptions to the Jacobian matrix entries require more complete descriptions since they do not follow the falloff formulation (e.g., falloff factor F_i). Instead, alternative partial derivatives of the reaction rate coefficient with respect to temperature ($\frac{\partial k_{f,i}}{\partial T}$) must be provided in place of Eq. (65) when calculating the partial derivatives of R_i using Eqs. (59), (68), or (70). Note that in both cases the partial derivatives with respect to species mass fractions ($\frac{\partial k_{f,i}}{\partial Y_j}$) are

zero, because the partial derivative of pressure with respect with species mass fraction is zero. In addition, since this treatment assumes a constant-pressure system, the partial derivative of pressure with respect to temperature is also zero.

For the logarithmic pressure-dependent Arrhenius rate, the partial derivative with respect to temperature is

$$\frac{\partial k_{f,i}}{\partial T} = \left[\frac{\partial(\log k_1)}{\partial T} + \left(\frac{\partial(\log k_2)}{\partial T} - \frac{\partial(\log k_1)}{\partial T} \right) \frac{\log p - \log p_1}{\log p_2 - \log p_1} + (\log k_2 - \log k_1) \frac{\frac{\partial(\log p)}{\partial T}}{\log p_2 - \log p_1} \right] \cdot k_{f,i} \ , \tag{101}$$

where k_1 and k_2 are given by Eqs. (37) and (38), respectively. Note that terms such as k_1, k_2, p_1, and p_2 are associated with the ith reaction alone; we omit the subscript for clarity, and continue this practice with other reaction-specific terms in the following discussion. The necessary components can be determined as

$$\frac{\partial(\log k_1)}{\partial T} = \frac{1}{T} \left(\beta_1 + \frac{T_{a,1}}{T} \right) \ , \tag{102}$$

$$\frac{\partial(\log k_2)}{\partial T} = \frac{1}{T} \left(\beta_2 + \frac{T_{a,2}}{T} \right) \ , \text{and} \tag{103}$$

$$\frac{\partial(\log p)}{\partial T} = \frac{1}{p} \frac{\partial p}{\partial T} = 0 \ . \tag{104}$$

The final simplified expression can next be constructed, appropriate for use when pressure falls between p_1 and p_2:

$$\frac{\partial k_{f,i}}{\partial T} = \frac{k_{f,i}}{T} \left[\beta_1 + \frac{T_{a,1}}{T} + \left(\beta_2 - \beta_1 + \frac{T_{a,2} - T_{a,1}}{T} \right) \frac{\log p - \log p_1}{\log p_2 - \log p_1} \right] \ . \tag{105}$$

Finally, the partial derivatives of the rate coefficient for the ith reaction with a Chebyshev rate expression can be evaluated; again, we omit the subscript i for clarity:

$$\frac{\partial k_f}{\partial T} = \log(10) \cdot k_f \sum_{i=1}^{N_T} \sum_{j=1}^{N_p} \eta_{ij} \frac{\partial}{\partial T} \left(\mathcal{T}_{i-1}(\tilde{T}) \, \mathcal{T}_{j-1}(\tilde{p}) \right) \ ,$$

where

$$\frac{\partial}{\partial T} \left(\mathcal{T}_{i-1}(\tilde{T}) \, \mathcal{T}_{j-1}(\tilde{p}) \right) = (i-1) U_{i-2}(\tilde{T}) \, \mathcal{T}_{j-1}(\tilde{p}) \frac{\partial \tilde{T}}{\partial T} + (j-1) \mathcal{T}_{i-1}(\tilde{T}) \, U_{j-2}(\tilde{p}) \frac{\partial \tilde{p}}{\partial T} \ , \tag{106}$$

$$\frac{\partial \tilde{T}}{\partial T} = \frac{-2T^{-2}}{T_{\max}^{-1} - T_{\min}^{-1}} \ , \tag{107}$$

$$\frac{\partial \tilde{p}}{\partial T} = \frac{\frac{2}{p \log(10)} \frac{\partial p}{\partial T}}{\log_{10} p_{\max} - \log_{10} p_{\min}} = 0 \ , \tag{108}$$

and U_n is the Chebyshev polynomial of the second kind of degree n, expressed as

$$U_n(x) = \frac{\sin\left((n+1)\arccos x\right)}{\sin(\arccos x)} \,. \tag{109}$$

Thus, the partial derivative of the forward rate coefficient can be expressed as

$$\frac{\partial k_f}{\partial T} = \frac{k_f}{T}\log(10)\sum_{i=1}^{N_T}\sum_{j=1}^{N_p}\eta_{ij}\left((i-1)U_{i-2}(\tilde{T})\,\mathcal{T}_{j-1}(\tilde{p})\frac{-2T^{-1}}{T_{\max}^{-1}-T_{\min}^{-1}}\right) \,. \tag{110}$$

The partial derivative of the reverse rate coefficient $k_{r,i}$ with respect to temperature can be found using Eq. (70) for both pressure-dependent reaction classes. The partial derivative of $k_{r,i}$ with respect to species mass fractions is zero, for the same reason as the forward rate coefficient.

3.2. Evaluation of Jacobian matrix

The Jacobian matrix can be efficiently filled by arranging the evaluation of elements in an appropriate order. First, evaluate the Jacobian entries for partial derivatives of species equations ($\mathcal{J}_{k+1,1}$ and $\mathcal{J}_{k+1,j+1}$ for $k,j = 1,\ldots,N_{\text{sp}}-1$):

$$\mathcal{J}_{k+1,1} = \frac{W_k}{\rho}\sum_{i=1}^{N_{\text{reac}}}\nu_{ki}\left[\frac{\partial c_i}{\partial T}\left(R_{f,i}-R_{r,i}\right)+c_i\left(\frac{\partial R_i}{\partial T}+\frac{R_{f,i}-R_{r,i}}{T}\right)\right] \,, \tag{111}$$

where all terms were expressed previously; note that $c_i = 1$ and $\frac{\partial c_i}{\partial T} = 0$ for pressure-dependent reactions expressed via logarithmic interpolations or Chebyshev polynomials. The remaining columns are given by

$$\begin{aligned}
\mathcal{J}_{k+1,j+1} = \frac{W_k}{\rho}\Bigg(&\dot{\omega}_k W\left(\frac{1}{W_j}-\frac{1}{W_{N_{\text{sp}}}}\right)+\sum_{i=1}^{N_{\text{reac}}}\nu_{ki}\Bigg(\frac{\partial c_i}{\partial Y_j}\left(R_{f,i}-R_{r,i}\right)\\
&+c_i\sum_{l=1}^{N_{\text{sp}}}\Bigg(\nu'_{li}\Bigg(-R_{f,i}W\Big(\frac{1}{W_j}-\frac{1}{W_{N_{\text{sp}}}}\Big)\\
&+\left(\delta_{lj}-\delta_{lN_{\text{sp}}}\right)k_{f,i}\frac{\rho}{W_l}[X_l]^{\nu'_{li}-1}\prod_{\substack{n=1\\n\neq l}}^{N_{\text{sp}}}[X_n]^{\nu'_{ni}}\Bigg)\\
&-\nu''_{li}\Bigg(-R_{r,i}W\Big(\frac{1}{W_j}-\frac{1}{W_{N_{\text{sp}}}}\Big)\\
&+\left(\delta_{lj}-\delta_{lN_{\text{sp}}}\right)k_{r,i}\frac{\rho}{W_l}[X_l]^{\nu''_{li}-1}\prod_{\substack{n=1\\n\neq l}}^{N_{\text{sp}}}[X_n]^{\nu''_{ni}}\Bigg)\Bigg)\Bigg)\Bigg) \,.
\end{aligned} \tag{112}$$

Next, the Jacobian entries for partial derivatives of the energy equation ($\mathcal{J}_{1,1}$ and $\mathcal{J}_{1,j+1}$ for $j = 1,\ldots,N_{\text{sp}}-1$) are given by

$$\mathcal{J}_{1,1} = \frac{-1}{c_p}\left\{\sum_{k=1}^{N_{\text{sp}}-1}\left[\left(c_{p,k}-\frac{h_k}{c_p}\frac{\partial c_p}{\partial T}\right)\frac{W_k\dot{\omega}_k}{\rho}+h_k\mathcal{J}_{k+1,1}\right]\right.$$

$$+ \left(c_{p,N_{\text{sp}}} - \frac{h_{N_{\text{sp}}}}{c_p} \frac{\partial c_p}{\partial T} \right) \frac{W_{N_{\text{sp}}} \dot{\omega}_{N_{\text{sp}}}}{\rho} + h_{N_{\text{sp}}} \hat{\mathcal{J}}_{N_{\text{sp}}+1,1} \Bigg\} \tag{113}$$

$$\mathcal{J}_{1,j+1} = \frac{-1}{c_p} \left(-\frac{c_{p,j} - c_{p,N_{\text{sp}}}}{\rho c_p} \sum_{k=1}^{N_{\text{sp}}} h_k W_k \dot{\omega}_k + \sum_{k=1}^{N_{\text{sp}}-1} h_k \mathcal{J}_{k+1,j+1} + h_{N_{\text{sp}}} \hat{\mathcal{J}}_{N_{\text{sp}}+1,j+1} \right), \tag{114}$$

where $\frac{\partial c_p}{\partial T}$ is given by Eq. (54); $\hat{\mathcal{J}}_{N_{\text{sp}}+1,1}$ and $\hat{\mathcal{J}}_{N_{\text{sp}}+1,j+1}$ are placeholder variables for the last species N_{sp} calculated in a similar manner as Eqs. (111) and (112).

3.3. Jacobian matrix sparsity

While some discuss the sparsity of analytically evaluated Jacobian matrices for chemical kinetics [5, 43], for the governing equations of constant-pressure kinetics using mass fractions the Jacobian is actually dense. This results from the partial derivative of density with respect to species mass fraction, $\partial\rho/\partial Y_j$, that appears in Eq. (49). As a consequence, the partial derivatives of all species rate-of-change $\dot{Y}_k$ with respect to the jth species' mass fraction Y_j, or $\mathcal{J}_{k+1,j+1}$, are potentially nonzero. However, if the constant-volume assumption instead holds true, then $\partial\rho/\partial Y_j = 0$ and Jacobian matrices exhibit the sparsity shown by Perini et al. [43]. In that case, off-diagonal elements of the Jacobian matrix arise due to direct species interactions and falloff/pressure-dependent reactions. Unfortunately, simulations of low-speed combustion systems typically rely on the constant-pressure assumption [46].

Alternatively, the governing equations can be formulated in terms of species molar concentrations rather than mass fractions [5, 42]. This eliminates the presence of ρ in the cross-species Jacobian matrix elements and thus significantly increases matrix sparsity for constant-pressure combustion. We plan to explore this approach for future versions of `pyJac`.

4. Optimization of Jacobian evaluation

Algorithm (1) presents pseudocode for a simple implementation of the outlined Jacobian evaluation approach based on the equations in Section 3.2. Although attractive as a straightforward implementation, it neglects potential reuse of computed products and thus unnecessarily recomputes terms. Furthermore, this formulation produces source code with substantial numbers of lines even for small kinetic models such as GRI-Mech 3.0 [58] (e.g., ∼100,000 lines).

Without a strategy to enable the reuse of temporary computed products even reasonably modern compilers (e.g., `gcc` 4.4.7), struggle to compile the resulting code, resulting in long compilation times, slow execution, and even occasional crashes of the compiler itself. This section lays out a restructuring of Algorithm (1), presented in Algorithm (2), that greatly accelerates Jacobian evaluation via reuse of temporary products to reduce computational overhead. As a side-benefit of this technique, compilation times are greatly reduced and compiler crashes can be avoided.

Algorithm 1 A pseudo-code for simple/naive generation of the chemical Jacobian.

for i in $1, \ldots, N_{\text{sp}} - 1$ **do**
  for k in $1, \ldots, N_{\text{reac}}$ **do**
    if $\nu_{i,k} \neq 0$ **then**
      Generate code for reaction k's contribution to $\frac{\partial \dot{Y}_i}{\partial T}$
  for j in $1, \ldots, N_{\text{sp}} - 1$ **do**
    for k in $1, \ldots, N_{\text{reac}}$ **do**
      if $\nu_{j,k} \neq 0$ **then**
        Generate code for reaction k's contribution to $\frac{\partial \dot{Y}_k}{\partial Y_j}$
for i in $1, \ldots, N_{\text{sp}} - 1$ **do**
  Generate code for $\frac{\partial \dot{T}}{\partial Y_i}$
Generate code for $\frac{\partial \dot{T}}{\partial T}$

Algorithm 2 A restructured pseudo-code to enable efficient chemical Jacobian evaluation.

for k in $1, \ldots, N_{\text{reac}}$ **do**
  Define reusable products for temperature derivatives
  for i in $1, \ldots, N_{\text{sp}} - 1$ **do**
    if $\nu_{i,k} \neq 0$ **then**
      Generate code for reaction k's contribution to $\frac{\partial \dot{Y}_i}{\partial T}$
  Define reusable products for species derivatives
  for j in $1, \ldots, N_{\text{sp}} - 1$ **do**
    for i in $1, \ldots, N_{\text{sp}} - 1$ **do**
      if $\nu_{i,k} \neq 0$ **then**
        Generate code for $\frac{\partial \dot{Y}_i}{\partial Y_j}$
for i in $1, \ldots, N_{\text{sp}} - 1$ **do**
  Generate code for $\frac{\partial \dot{T}}{\partial Y_i}$
Generate code for $\frac{\partial \dot{T}}{\partial T}$

4.1. Accelerating evaluation via reuse of temporary products

First, examining Eqs. (111) and (112), we see that large portions of the Jacobian entries are constant for a single reaction. For instance, if we define a temporary variable for the ith reaction

$$\Theta_{\partial T,i} = \frac{1}{\rho}\left[\frac{\partial c_i}{\partial T}\left(R_{f,i} - R_{r,i}\right) + c_i\left(\frac{\partial R_i}{\partial T} + \frac{R_{f,i} - R_{r,i}}{T}\right)\right] , \tag{115}$$

then Eq. (111) can be rewritten as

$$\mathcal{J}_{k+1,1} = W_k \sum_{i=1}^{N_{\text{reac}}} \nu_{ki}\Theta_{\partial T,i} . \tag{116}$$

Next, instead of summing over all reactions for a single species, we transform Algorithm (1) to compute the temporary product for a single reaction i and add to all relevant species:

$$\mathcal{J}_{k+1,1} \mathrel{+}= \nu_{ki} W_k \Theta_{\partial T,i} \quad k = 1, \ldots, N_{\text{sp}} - 1 . \tag{117}$$

and—as introduced in Sec. (3.2)—the placeholder variable $\hat{\mathcal{J}}_{N_{\text{sp}}+1,1}$ may be updated analogously to Eq. (117). In doing so, the temporary product $\Theta_{\partial T,i}$ must only be evaluated once for each reaction, rather than once for every species in each reaction with a non-zero net production or consumption rate.

Similarly, for Eq. (112) we can define similar temporary products, although more care must be taken to ensure the correctness for all of the different reaction types. For all reactions i, the following temporary product can be defined:

$$\Theta_{\partial Y,i,\text{ind}} = -\frac{W c_i}{\rho}\left(R_{f,i}\sum_{l=1}^{N_{\text{sp}}}\nu'_{li} - R_{r,i}\sum_{l=1}^{N_{\text{sp}}}\nu''_{li}\right) . \tag{118}$$

The Jacobian entries $\mathcal{J}_{k+1,j+1}$ for a pressure-independent reaction i can then be updated with

$$\begin{aligned}\mathcal{J}_{k+1,j+1} \mathrel{+}= \nu_{ki}\frac{W_k}{W_j}\Bigg[\Theta_{\partial Y,i,\text{ind}}\left(1 - \frac{W_j}{W_{N_{\text{sp}}}}\right) + c_i\Bigg(&\left(k_{f,i}S'_j - k_{r,i}S''_j\right) \\ &- \frac{W_j}{W_{N_{\text{sp}}}}\left(k_{f,i}S'_{N_{\text{sp}}} - k_{r,i}S''_{N_{\text{sp}}}\right)\Bigg)\Bigg] \quad k, j = 1, \ldots, N_{\text{sp}} - 1 ,\end{aligned} \tag{119}$$

where S'_l and S''_l are defined (for $l = \{j, N_{sp}\}$) as

$$S'_l = \nu'_{li}[X_l]^{\nu'_{li}-1}\prod_{\substack{n=1\\n\neq l}}^{N_{\text{sp}}}[X_n]^{\nu'_{ni}} \text{ and} \tag{120}$$

$$S''_l = \nu''_{li}[X_l]^{\nu''_{li}-1}\prod_{\substack{n=1\\n\neq l}}^{N_{\text{sp}}}[X_n]^{\nu''_{ni}} . \tag{121}$$

and the placeholder variable $\hat{\mathcal{J}}_{N_{\rm sp}+1,j+1}$ may be updated as in Eq. (119). Note that either one or both of ν'_{ji} and ν''_{ji} can be zero, while $\nu'_{N_{\rm sp}i}$ and $\nu''_{N_{\rm sp}i}$ are typically both zero if the last species is an inert abundant gas, e.g., N_2 for air-fed combustion. This means that the added S'_l and S''_l terms are often zero, and may be omitted in such cases.

For third-body enhanced or falloff reactions, the $\frac{\partial c_i}{\partial Y_j}$ term in Eq. (112) is inherently dependent on the jth species; however, certain simplifications can still be made. First, for any third-body enhanced or falloff reaction where all third-body efficiencies $\alpha_{i,j}$ are unity, the derivative $\frac{\partial c_i}{\partial Y_j}$ is identically zero as seen in Eq. (81), and the updating scheme defined in Eq. (119) can be used. Otherwise, the appropriate update schemes described below must be employed.

For third-body-enhanced reactions, we define

$$\begin{aligned}\Theta_{\partial Y,i,3^{\rm rd}} &= \Theta_{\partial Y,i,\rm ind} + \frac{-Wc_i}{\rho}R_i \\ &= -\frac{Wc_i}{\rho}\left(R_{f,i}\left(1+\sum_{l=1}^{N_{\rm sp}}\nu'_{li}\right) - R_{r,i}\left(1+\sum_{l=1}^{N_{\rm sp}}\nu''_{li}\right)\right) .\end{aligned} \tag{122}$$

The Jacobian entries $\mathcal{J}_{k+1,j+1}$ for a third-body reaction without falloff dependence can then be updated with

$$\begin{aligned}\mathcal{J}_{k+1,j+1} \mathrel{+}= \nu_{ki}\frac{W_k}{W_j}\Bigg[&\Theta_{\partial Y,i,3^{\rm rd}}\left(1-\frac{W_j}{W_{N_{\rm sp}}}\right) \\ &+ \left(\alpha_{ij} - \alpha_{iN_{\rm sp}}\frac{W_j}{W_{N_{\rm sp}}}\right)R_i \\ &+ c_i\Bigg(\left(k_{f,i}S'_j - k_{r,i}S''_j\right) \\ &- \frac{W_j}{W_{N_{\rm sp}}}\left(k_{f,i}S'_{N_{\rm sp}} - k_{r,i}S''_{N_{\rm sp}}\right)\Bigg)\Bigg]\end{aligned} \tag{123}$$

for $k,j = 1,\ldots,N_{\rm sp}-1$, and the placeholder variable $\hat{\mathcal{J}}_{N_{\rm sp}+1,j+1}$ may be updated using Eq. (123).

For reactions with falloff dependence, we define a temporary term corresponding to the P_r polynomial in Eqs. (83) and (87):

$$\Theta_{P_{r,i}} = \begin{cases} 0 & \text{if Lindemann falloff or chemically activated reaction,} \\ \dfrac{1.0}{1.0+P_r} & \text{if Troe/SRI falloff reaction, or} \\ \dfrac{-P_r}{1.0+P_r} & \text{if Troe/SRI chemically activated reaction.} \end{cases} \tag{124}$$

Following this, we define a temporary term for the falloff function derivative $\frac{\partial F_i}{\partial Y_j}$:

$$\Theta_{F_i} = \begin{cases} 0 & \text{if Lindemann,} \\ \ln F_{\text{cent}} \dfrac{2A_{\text{Troe}}}{B_{\text{Troe}}^3} \dfrac{0.14A_{\text{Troe}} + B_{\text{Troe}}}{\log 10(1 + (A_{\text{Troe}}/B_{\text{Troe}})^2)^2} & \text{if Troe, or} \\ X^2 \dfrac{2\log_{10} P_{r,i}}{\log 10} \log\left(a \cdot \exp\dfrac{-b}{T} + \exp\dfrac{-T}{c}\right) & \text{if SRI.} \end{cases} \tag{125}$$

Using the above components, we then define the temporary products for falloff reactions:

$$\begin{aligned} \Theta_{\partial Y,i,\text{falloff}} &= \Theta_{\partial Y,i,\text{ind}} + c_i \left(\Theta_{P_{r,i}} + \Theta_{F_i}\right) \left(\frac{-W}{\rho}\right) R_i \\ &= \frac{-W c_i}{\rho} \left(R_{f,i} \left(\sum_l^{N_{\text{sp}}} \nu'_{li}\right) - R_{r,i} \left(\sum_l^{N_{\text{sp}}} \nu''_{li}\right) + R_i \left(\Theta_{P_{r,i}} + \Theta_{F_i}\right)\right) . \end{aligned} \tag{126}$$

The Jacobian entries $\mathcal{J}_{k+1,j+1}$ for a falloff dependent reaction i can then be updated as

$$\begin{aligned} \mathcal{J}_{k+1,j+1} \mathrel{+}= \nu_{ki} \frac{W_k}{W_j} \Bigg[& \Theta_{\partial Y,i,\text{falloff}} \left(1 - \frac{W_j}{W_{N_{\text{sp}}}}\right) \\ & + \frac{\frac{k_0}{k_\infty} F_i}{1 + Pr_i} \left(\Theta_{P_{r,i}} + \Theta_{F_i}\right) \left(\delta_M \left(\alpha_{ij} - \alpha_{iN_{\text{sp}}} \frac{W_j}{W_{N_{\text{sp}}}}\right)\right. \\ & \left. + \ (1 - \delta_{\text{M}}) \left(\delta_{mj} - \delta_{mN_{\text{sp}}} \frac{W_j}{W_{N_{\text{sp}}}}\right)\right) R_i \\ & + c_i \left(\left(k_{f,i} S'_j - k_{r,i} S''_j\right) - \frac{W_j}{W_{N_{\text{sp}}}} \left(k_{f,i} S'_{N_{\text{sp}}} - k_{r,i} S''_{N_{\text{sp}}}\right)\right)\Bigg] \end{aligned} \tag{127}$$

for $k, j = 1, \ldots, N_{\text{sp}} - 1$, where δ_{M} is defined as:

$$\delta_M = \begin{cases} 1 & \text{if the mixture acts as third body, or} \\ 0 & \text{if species } m \text{ acts as third body.} \end{cases} \tag{128}$$

The placeholder variable $\hat{\mathcal{J}}_{N_{\text{sp}}+1,j+1}$ may be updated as in Eq. (127)

Finally, all Jacobian entries $\mathcal{J}_{k+1,j+1}$ and placeholder term $\hat{\mathcal{J}}_{N_{\text{sp}}+1,j+1}$ must be finished with the addition of the species rate term:

$$\mathcal{J}_{k+1,j+1} \mathrel{+}= \frac{W_k}{W_j} \frac{\dot{\omega}_k W}{\rho} \left(1 - \frac{W_j}{W_{N_{\text{sp}}}}\right) \tag{129}$$

$$\hat{\mathcal{J}}_{N_{\text{sp}}+1,1} \mathrel{+}= \frac{W_{N_{\text{sp}}}}{W_j} \frac{\dot{\omega}_{N_{\text{sp}}} W}{\rho} \left(1 - \frac{W_j}{W_{N_{\text{sp}}}}\right) \tag{130}$$

As with the update scheme used in Eq. (117), the strategy presented in Eqs. (119), (123), and (127) reduces the computational overhead of Jacobian evaluation. The bulk of the computation is performed once per reaction, and only minor sub-products must be computed for each species-species pair for a given reaction.

Model	N_{sp}	N_{reac}	Lind.	Troe	SRI	P-log	Cheb.	Ref.
H_2/CO	13	27		×				[65]
GRI-Mech. 3.0	53	325	×	×				[58]
USC-Mech II	111	784	×	×				[66]
$iC_5H_{11}OH$	360	2172	×	×	×	×	×	[67]

Table 1: Summary of chemical kinetic models used as benchmark test cases. All models contain third-body reactions with enhanced third bodies. The "Lind.", "Troe", "SRI", "P-log", and "Cheb." columns indicate the presence of Lindemann, Troe, and SRI falloff; logarithmic pressure interpolation; and Chebyshev pressure-dependent reactions, respectively. Two reactions with photons were removed from the Sarathy et al. [67] model since neither Cantera nor `pyJac` supports such reactions.

5. Results and discussion

The Python [59] package `pyJac` implements the methodology for producing analytical Jacobian matrices described in the previous sections, which we released openly online [60] under the MIT license. `pyJac` requires the Python module `NumPy` [61]. The modules used to test the correctness and performance of `pyJac` are included in this release, and additionally require `Cython` [62], Cantera [54], `PyYAML` [63], and `Adept` [64]; however, these are not required for Jacobian/rate subroutine generation. In addition, interpreting Cantera-format models [54] requires installing the Cantera module for any purpose, while `pyJac` includes native support for interpreting Chemkin-format models [53].

In order to demonstrate the correctness and computational performance of the generated analytical Jacobian matrices, we chose four chemical kinetic models as test cases, selected to represent a wide spectrum of sizes, classes of fuel species, and types of reaction rate pressure dependence formulations (e.g., Lindemann/Troe/SRI falloff formulations, Chebyshev, pressure-log). Table 1 summarizes the chemical kinetics models used as benchmarks in this work, including the H_2/CO model of Burke et al. [65], GRI-Mech 3.0 [58], USC-Mech version II [66], and the isopentanol model of Sarathy et al. [67].

For both the correctness and performance tests, stochastic partially stirred reactor (PaSR) simulations generated thermochemical composition data covering a wide range of temperatures and species mass fractions. Appendix C contains a detailed description of the PaSR methodology and implementation; in addition, the `pyJac` package contains the PaSR code used in the present study. We performed nine premixed PaSR simulations for each fuel, with the parameters given in Table 2. Each simulation ran for five (isopentanol) or ten (hydrogen, methane, and ethylene) residence times (τ_{res}) to ensure reaching statistical steady state.

5.1. Validation

In order to test the correctness of the Jacobian matrices produced by `pyJac`, we initially compared the resulting analytical matrices against numerical approximations based on finite differences. However, while these commonly provide numerical approximations to Jacobian matrices, potential scaling issues due to large disparities in species mass fractions and associated net production rates can cause challenges in selecting appropriate differencing step

Parameter	H_2	CH_4	C_2H_4	$iC_5H_{11}OH$
ϕ	1			
T_{in}	400, 600, and 800 K			
p	1, 10, and 25 atm			
N_{part}	100			
τ_{res}	10 ms	5 ms	100 µs	100 µs
τ_{mix}	1 ms	1 ms	10 µs	10 µs
τ_{pair}	1 ms	1 ms	10 µs	10 µs
N_{res}	10	10	10	5

Table 2: PaSR parameters used for hydrogen/air, methane/air, ethylene/air, and isopentanol/air premixed combustion cases, where ϕ indicates equivalence ratio, T_{in} is the temperature of the inflowing particles, p is the pressure, N_{part} is the number of particles, τ_{res} is the residence time, τ_{mix} is the mixing time, τ_{pair} is the pairing time, and N_{res} is the number of residence times simulated.

sizes [68]. For example, we encountered large errors in some partial derivatives under certain conditions when initially attempting to evaluate derivatives using sixth-order central differences with increments calculated similarly to the implicit CVODE integrator [31]. Adjusting the relative and absolute tolerances reduced some of these errors, but we could not find consistent, effective values for all the states considered—in particular, states near equilibrium required unreasonably small tolerances (e.g., 1×10^{-20}), but these same tolerances caused larger errors at other states. Note that the discussion of our experiences are not intended to imply issues with the numerical Jacobian approach used in CVODE or other implicit integrators; such solvers only require approximations to the Jacobian matrix. However, based on our experiences, we recommend taking care when attempting to obtain highly accurate Jacobian matrices, as in the current effort and for analysis techniques such as computational singular perturbation [15, 69–71] and chemical explosive mode analysis [72–74] that rely on eigendecomposition of the Jacobian matrix.

As a result of the aforementioned difficulties with finite differences, we obtained accurate Jacobian matrices via automatic differentiation through expression templates using the `Adept` software library [64, 75]. In our experience, numerical Jacobians obtained using multiple-term Richardson extrapolants [76–78] are of similar accuracy but have a much higher computational cost due to the large number of function evaluations required. We did not explore other options for obtaining highly accurate numerical Jacobian approximations using, e.g., complex-step derivatives [79–81]. For all test cases, the correctness of the species and reaction rate subroutines were established through comparison with Cantera [54].

In reporting the discrepancy between the analytical and automatically differentiated Jacobian matrices, denoted by $\mathcal{J}$ and $\hat{\mathcal{J}}$ respectively, we used the Frobenius norm of the relative errors of matrix elements

$$E_{\text{rel}} = \left\| \frac{\hat{\mathcal{J}} - \mathcal{J}}{\hat{\mathcal{J}}} \right\|_F , \tag{131}$$

to quantify error. This differs somewhat from the error metric suggested by Anderson et

Model	Sample size	Mean (%)	Maximum (%)
H_2/CO	900,900	3.170×10^{-8}	9.828×10^{-4}
GRI-Mech 3.0	450,900	2.946×10^{-8}	1.240×10^{-4}
USC-Mech II	91,800	9.046×10^{-9}	2.356×10^{-4}
$iC_5H_{11}OH$	450,900	8.686×10^{-10}	1.680×10^{-5}

Table 3: Summary of Jacobian matrix correctness study results. Error statistics are based on the norm of the relative error percentages E_{rel} for each sample. The sample size for each case depended on the number of particles and relevant timescales used.

al. [82] for use quantifying the error of matrices in LAPACK: the relative error Frobenius norm

$$E_{\text{norm}} = \frac{\left\| \hat{\mathcal{J}} - \mathcal{J} \right\|_F}{\left\| \hat{\mathcal{J}} \right\|_F} . \tag{132}$$

E_{norm} indicates overall error in the matrix but can be dominated by large elements, so we believe the error measure E_{rel} given by Eq. (131) to be more useful in identifying discrepancies in both larger and smaller matrix elements.

Table 3 presents the correctness testing results for each benchmark case. For certain conditions, we observed large relative errors in small nonzero elements (e.g., magnitudes of $\sim 10^{-22}$) likely due to roundoff errors; thus, error statistics comprised only matrix elements $\mathcal{J}_{ij}$ where $|\mathcal{J}_{ij}| \geq \|\mathcal{J}\|_F / 10^{20}$. For all benchmark cases, the analytical Jacobian matrices match closely with those obtained via automatic differentiation, with the largest discrepancy below 1×10^{-3} %.

We also initially made comparisons with the `TChem` package [40, 83] for validation purposes, using the H_2/CO model. However, while many quantities calculated via `TChem` agreed closely with those from both Cantera and `pyJac`, we observed discrepancies in the `TChem` species net production rates that correlated with significant errors in the derivative source term and Jacobian matrix. These discrepancies occurred mainly for low species production rates, where the (larger) rates for other species agreed closely, and in general for near-equilibrium states. Furthermore, during our testing it became apparent that `TChem` is not thread-safe when parallelized with OpenMP. We discuss this issue further in Appendix D. As a result, we do not present detailed Jacobian matrix comparisons between `TChem` and `pyJac` here.

5.2. Performance analysis

The performance of `pyJac`-generated subroutines for CPU and GPU execution was tested by evaluating Jacobian matrices for the four kinetic models using the previously discussed PaSR thermochemical composition data. In both cases, the performance of `pyJac` was compared with that of a finite-difference-based Jacobian; in addition, the CPU routines were compared with `TChem` [83] for the H_2/CO, GRI-Mech 3.0, and USC-Mech II models. The CPU Jacobian subroutines were compiled using `gcc` 4.8.5 [84], and run on four ten-core Intel Xeon E5-4640 v2 processors with 20 MB of L3 cache memory, installed on an Ace

Model	$\bar{R}_{\texttt{TChem}}/\bar{R}_{\texttt{pyJac}}$	$\bar{R}_{\text{FD}}/\bar{R}_{\texttt{pyJac}}$
H_2/CO	2.16	4.40
GRI-Mech 3.0	1.49	6.98
USC-Mech II	1.07	7.51
$iC_5H_{11}OH$	–	2.96

Table 4: Ratio between CPU-based Jacobian matrix evaluation times of `TChem` and finite differences (FD), and `pyJac`. $\bar{R}$ indicates the mean evaluation time.

Powerworks PW8027R-TRF+ with a Supermicro X9QR7-TF+/X9QRi-F+ baseboard. As previously mentioned, our tests found that `TChem` v0.2 is not thread safe (c.f., Appendix D). Therefore, all CPU performance comparisons between `pyJac`, `TChem`, and the finite-difference method were carried in a single-threaded manner. Reported mean evaluation times were computed from the wall-clock run times of ten individual calculations. The GPU Jacobian subroutines were compiled using `nvcc` 7.5.17 [85] and tested on a single NVIDIA Tesla C2075 GPU. The mean GPU evaluation times were again computed from ten individual runs. For the CPU and GPU finite-difference-based Jacobian calculations, we used a simple first-order forward difference adapted from CVODE [31] to give a realistic comparison with a commonly used finite-difference technique.

Figure 1 shows the performance of the CPU-based `pyJac` Jacobian matrix evaluations for the four kinetic models, compared with the performance of finite difference calculations and the `TChem` package [40]. (`TChem` does not support pressure-log or Chebyshev pressure-dependent reaction expressions, so we could not evaluate its performance with the isopentanol model.) Table 4 summarizes the performance ratio between Jacobian matrices calculated using `pyJac` and finite differences/`TChem`; `pyJac`-based routines perform approximately 1.1–2.2 $\times$ faster than `TChem` on a single-threaded basis, depending on the size of the model. However, we again note that `TChem`'s lack of thread-safety implies that calculation of Jacobians for many different thermo-chemical states on a multicore CPU may be much faster using `pyJac` as the calculations may be accelerated easily using OpenMP. `pyJac` also performs 3–7.5 $\times$ faster than the first-order finite difference technique.

Figure 1 also shows best-fit lines based on least-squares regressions for the available data (the corresponding R^2 values were 0.89, 0.57, and 0.99 for the finite difference, `pyJac` and `TChem` results, respectively). These fits suggest that `pyJac` and the finite difference method scale superlinearly—nearly quadratically—with number of reactions, while `TChem` scales nearly linearly with the number of reactions. This is expected for the finite difference approach, but warrants explanation for the analytical Jacobian methods. Lu and Law argued [5] that the cost of analytical Jacobian evaluation should scale linearly with the number of reactions in a model. However, this argument is predicated on the assumption of a sparse Jacobian resulting from a molar-concentration-based system; namely, that only changes in species concentrations that participate in a reaction affect the resulting reaction rate. As discussed in Section 3.3, the constant-pressure/mass-fraction-based system used in `pyJac`—along with the majority of reactive-flow modeling descriptions—results in a dense Jacobian and thus execution times that exhibit a higher-order dependence on the number of species/reactions. The current performance, while faster than either a typical finite difference

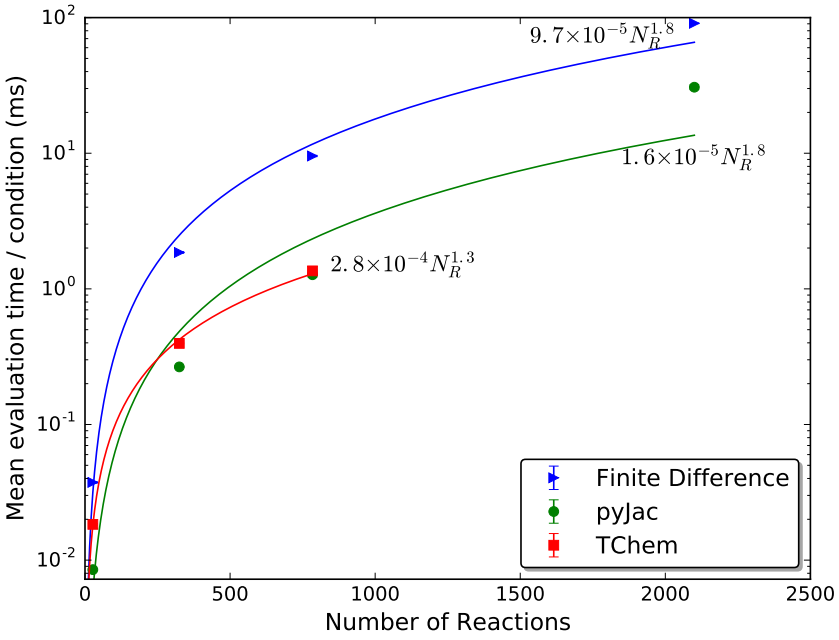


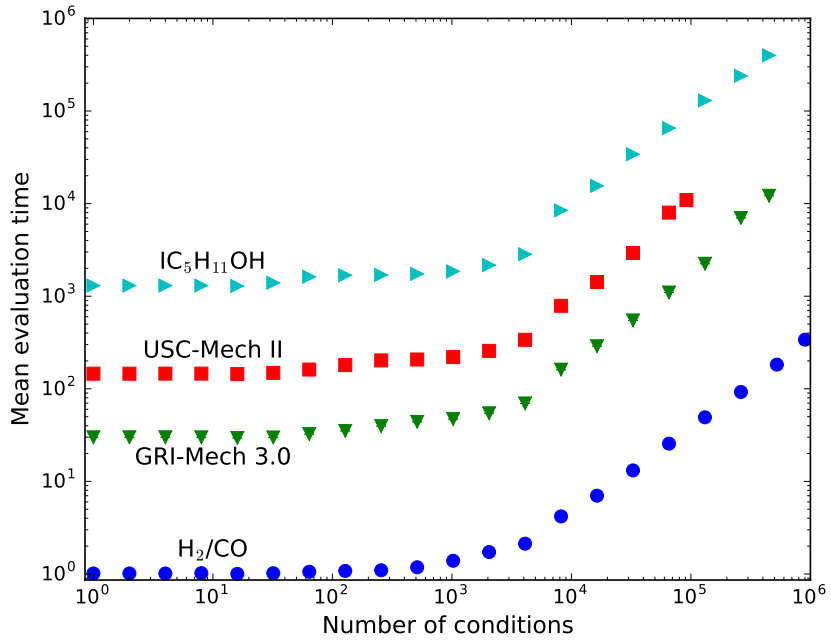
Figure 1: CPU-based Jacobian matrix evaluation times using finite differences, `pyJac`, and `TChem` for the four kinetic models. No performance data were available for the isopentanol model using `TChem` due to the presence of unsupported reaction types. The symbols indicate performance data, while the solid lines represent least-squares best fits based on the number of reactions N_R in the models. Error bars are present, but too small to detect.

approach or `TChem`, motivates developing a sparse-concentration-based constant-pressure Jacobian system in future versions of `pyJac`.

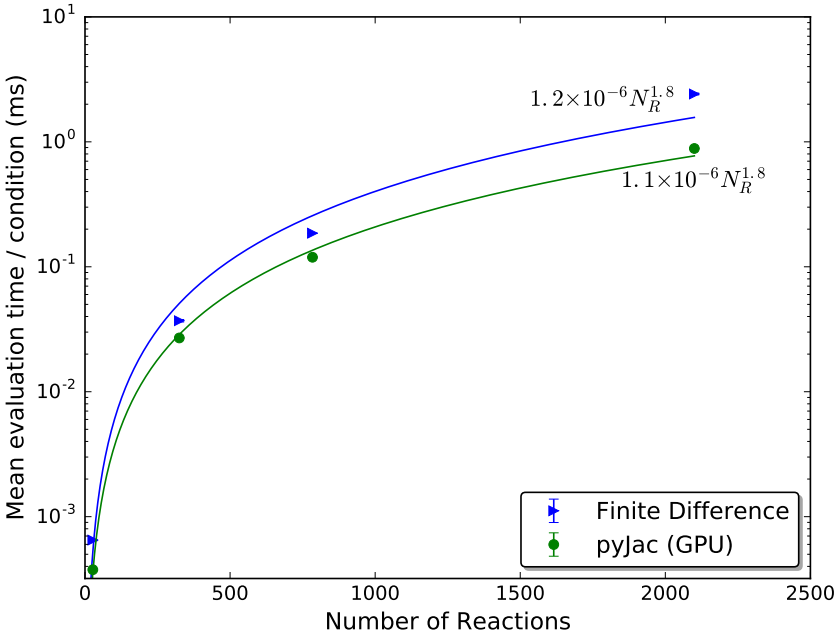
Figure 2 demonstrates the performance of the GPU Jacobian matrix implementations. Figure 2a shows the mean runtime of the GPU Jacobian matrix evaluations against the number of conditions—i.e., the number of thermochemical composition states, with one Jacobian matrix evaluated per state. As the number of conditions increases, the GPU becomes fully utilized and the growth rate of the evaluation time begins increasing linearly (here displayed on a log-log plot). This thread saturation point occurs at nearly the same number of conditions for each model. We adopted a "per-thread" GPU parallelization model in this work, where a single GPU thread evaluates a single Jacobian matrix, rather than a "per-block" model where GPU threads in a block cooperate to evaluate a Jacobian. The per-thread approach simplifies generation of highly optimized code for SIMD processors and provides a higher theoretical bound on the number of Jacobian matrices that can be evaluated in parallel. However, the choice of GPU parallelization merits further investigation because memory bandwidth limits the current per-thread implementation due to small cache sizes available on GPU streaming multiprocessors. A per-block implementation might utilize this small cache more effectively, but it is unclear if this would increase overall performance. Figure 2b shows the longest `pyJac` and forward finite-difference evaluation times—normalized by the number of conditions—for each kinetic model plotted against the number of reactions in the model. As with the CPU matrix evaluations shown in Fig. 1, we observed a superlinear (but subquadratic) scaling of the performance with number of reactions. The least-squares best-fit lines for the `pyJac` and finite-difference results—with R^2 values of 0.98 and 0.82, respectively—exhibit similar polynomial orders to the CPU-based Jacobian evaluation, suggesting dependence on the methods rather than hardware.

Table 5 shows the ratios of average evaluation times between finite difference and `pyJac` on the GPU for the longest runtimes (i.e., the full number of conditions for each model). Interestingly, `pyJac` performs noticeably better than the finite-difference technique with the isopentanol model ($\sim 3\times$) than the three smaller models (1–$2\times$). This likely occurs due to the small cache size of the GPU; in this case, the larger model size (360 species compared to, e.g., 111 species for USC-Mech II) makes it difficult to store a majority of species concentrations in the cache, forcing more loads from global memory. Thus, using an analytical Jacobian formulation increases performance for larger models by a factor of 1.6–2.0 over finite difference methods. A sparse Jacobian formulation would similarly greatly benefit GPU evaluation due to the greatly reduced memory traffic requirements (i.e., reduced number of global reads and writes).

Lastly, Figure 3 shows the scaling of performance with the number of cores, ranging from 1 to 32, and the number of conditions fixed at the values given for each model in Table 3. Figure 3a shows the mean evaluation time, which exhibits a power-law dependence on the number of cores. The functional relationship between evaluation time and number of cores is similar for each model, with an intercept increasing with model size. Figure 3b shows the strong scaling efficiency with number of cores. Interestingly, the strong scaling efficiency drops most significantly for the smallest and largest models (H_2/CO and $iC_5H_{11}OH$, respectively) with increasing number of processors. We hypothesize that the reasons for lower scaling efficiency of these two models actually differ. The cost of evaluation H_2/CO is low enough, particularly at increasing numbers of CPU cores, that the relative overhead for

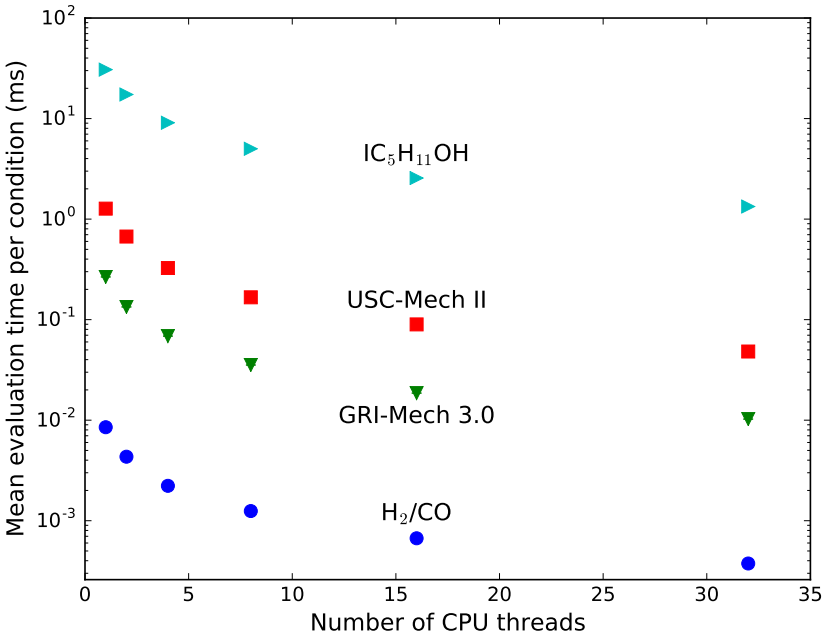


(a) Mean GPU runtime versus the number of conditions evaluated.

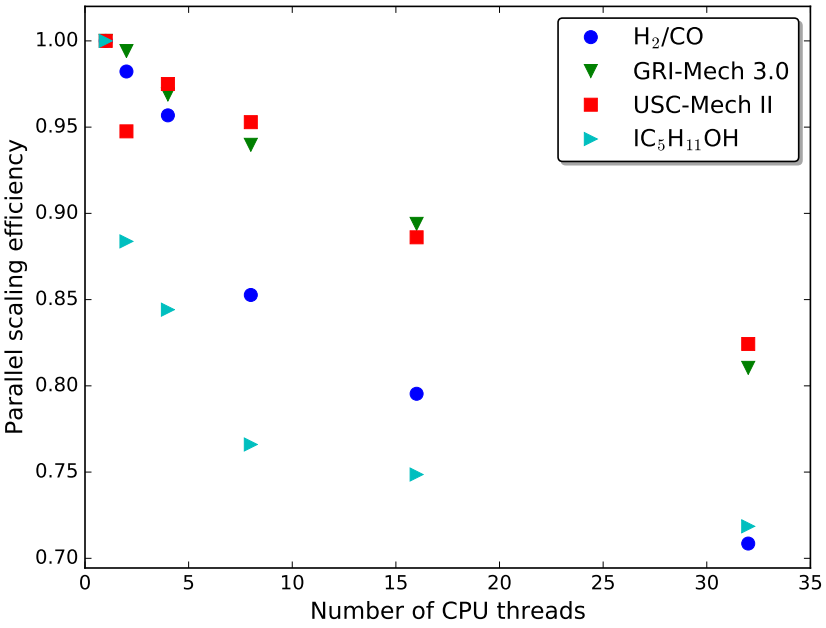


(b) Comparison of normalized GPU `pyJac` and finite difference Jacobian matrix evaluation times versus the number of reactions in the model. Symbols indicate performance data, while solid lines represent least-squares best fits based on the number of reactions N_R in the models.

Figure 2: Performance of the GPU-based `pyJac`. Note the logarithmic scales of the ordinate axes. Error bars are present, but too small to detect.



(a) Mean CPU runtime versus the number of CPU cores. Error bars are present, but too small to detect. Note the logarithmic scales of the vertical axis.



(b) Strong scaling efficiency versus number of CPU cores.

Figure 3: Parallel performance scaling of the CPU-based `pyJac`, with the number of CPU cores varying and the number of states fixed at the values associated with each model given in Table 3.

Model	$\bar{R}_{\mathrm{FD}}/\bar{R}_{\texttt{pyJac}}$
H_2/CO	1.72
GRI-Mech 3.0	1.37
USC-Mech II	1.56
$iC_5H_{11}OH$	2.73

Table 5: Ratio between finite difference (FD) and `pyJac` Jacobian matrix evaluation times on the GPU. $\bar{R}$ indicates the mean evaluation time.

launching OpenMP threads is larger and thus the efficiency drops. Conversely, the much larger size of the $iC_5H_{11}OH$ model requires frequent accesses of larger amounts of memory, and thus possibly also more frequent cache misses.

The results presented above raise a number of issues that warrant further study. The superlinear scaling of `pyJac` matrix evaluation time with number of reactions in the kinetic models results from the density of the Jacobian matrix, and suggests the potential benefits of a sparse molar-concentration-based Jacobian formulation. However, while `pyJac` outperforms `TChem` in all cases, the performance ratio between them reduces with increasing number of reactions, suggesting that the performance benefit of a hard-coded subroutine may decrease with increasing model size. Although it appears from these results that the performance of a hard-coded Jacobian subroutine can surpass that of an interface-based Jacobian evaluation, the hard-coded method has additional difficulties due to the size and number of files created. For example, the CPU Jacobian evaluation subroutine for USC-Mech II comprises over 360,000 total lines spread across 22 files (not including the supporting files for, e.g., species and reaction rate evaluations)—attempting to incorporate the entire matrix evaluation in a single file crashed an older version of the `gcc` compiler (4.4.7). This not only causes longer source code compilation times (on the order of ten minutes to an hour when using 24 parallel compiler instances), but could cause compiler crashes and inexplicable behavior, particularly for the less mature `nvcc` compiler. While we expect a sparse Jacobian formulation would significantly alleviate these issues, future work should directly compare the performance of hard-coded and interface-based Jacobian evaluation approaches.

6. Conclusions

This work developed the theory behind an analytical Jacobian matrix evaluation approach for constant-pressure chemical kinetics, including new derivations of partial derivatives with respect to modern reaction pressure-dependence formulations. In addition, we detailed strategies for efficient evaluation of the Jacobian matrix, including reordering matrix element evaluations in order to enable the reuse of temporary products. We implemented the presented methodology in the open-source software package `pyJac` [60], which generates custom source code files for evaluating chemical kinetics Jacobian matrices on both CPU and GPU systems. We established the correctness of the resulting Jacobian matrices through comparison with matrices obtained by automatic differentiation. The results agree within 0.001 % for kinetic models describing the oxidation of hydrogen, methane, ethylene, and isopentanol, with 13 to 360 species (and 27 to 2172 reactions). Finally, we investigated the

performance of the CPU and GPU matrix evaluation subroutines by evaluating the matrix calculation time for the same kinetic models. The `pyJac` CPU-based Jacobian evaluation performs 3–7.5 times faster than a forward-finite-difference approach, while a more modest improvement speedup of 1.07–2.16 × was found compared to the existing `TChem` software on a single-threaded basis. We note again that `TChem` is **not** thread-safe when parallelized with OpenMP, while `pyJac` is easily parallelized with the same approach; for multicore CPUs `pyJac` is expected to greatly outperform `TChem`.

Planned development work for `pyJac` will include support for constant-volume assumption, and the capability to generate Fortran and Matlab source codes. Further study will be performed into the benefits of the current custom source-code, compiled Jacobian evaluation subroutine approach versus a loop/branch-based approach, in order to determine how to obtain the best performance scaling with kinetic model size. Finally, we will investigate the performance benefits of converting from a differential system based on species mass fractions to molar concentrations, which offers a marked improvement in matrix sparsity.

Acknowledgments

This material is based upon work supported by the National Science Foundation under grants ACI-1535065 and ACI-1534688. In addition, we thank James Sutherland and Mike Hansen of the University of Utah for helpful discussions on appropriate formulation of state vectors.

Appendix A. Supplementary material

The most recent version of `pyJac` can be found at its GitHub repository https://github.com/SLACKHA/pyJac. In addition, the repository contains detailed documentation and an issue-tracking system. The full source for this paper is also available at https://github.com/Niemeyer-Research-Group/pyJac-paper, including the data and scripts needed to reproduce the figures here.

Appendix B. Proof of partial derivative of pressure

$$
\begin{aligned}
\frac{\partial p}{\partial Y_j} &= \frac{\partial}{\partial Y_j}\left(\mathcal{R}T\sum_{k=1}^{N_{\mathrm{sp}}}[X_k]\right) = \mathcal{R}T\sum_{k=1}^{N_{\mathrm{sp}}}\frac{\partial[X_k]}{\partial Y_j} \\
&= \mathcal{R}T\sum_{k=1}^{N_{\mathrm{sp}}}\left[-[X_k]W\left(\frac{1}{W_j}-\frac{1}{W_{N_{\mathrm{sp}}}}\right)+\left(\delta_{kj}-\delta_{kN_{\mathrm{sp}}}\right)\frac{\rho}{W_k}\right] \\
&= -\mathcal{R}TW\left(\frac{1}{W_j}-\frac{1}{W_{N_{\mathrm{sp}}}}\right)\sum_{k=1}^{N_{\mathrm{sp}}}[X_k]+\mathcal{R}T\rho\sum_{k=1}^{N_{\mathrm{sp}}}\frac{\delta_{kj}-\delta_{kN_{\mathrm{sp}}}}{W_k} \\
&= -pW\left(\frac{1}{W_j}-\frac{1}{W_{N_{\mathrm{sp}}}}\right)+\rho\mathcal{R}T\left(\frac{1}{W_j}-\frac{1}{W_{N_{\mathrm{sp}}}}\right) \\
&= \left(-pW+pW\right)\left(\frac{1}{W_j}-\frac{1}{W_{N_{\mathrm{sp}}}}\right)
\end{aligned}
$$

$$\therefore \frac{\partial p}{\partial Y_j} = 0 \,. \tag{B.1}$$

Note that this holds without any assumption of constant pressure or volume.

Appendix C. Partially stirred reactor implementation

Here, we describe our partially stirred reactor (PaSR) implementation for completeness, based on prior descriptions [27, 86–90]. The reactor model consists of an even number N_{part} of particles, each with a time-varying composition $\phi(t)$. Unlike the composition vector given previously (Eq. (1)), here we use mixture enthalpy and species mass fractions to describe the state of a particle:

$$\phi = \left\{h, Y_1, Y_2, \ldots, Y_{N_{\text{sp}}}\right\}^{\intercal} \,. \tag{C.1}$$

At discrete time steps of size Δt, events including inflow, outflow, and pairing cause certain particles to change composition; between these time steps, mixing and reaction fractional steps separated by step size Δt_{sub} evolve the composition of all particles.

Inflow and outflow events at the discrete time steps comprise the inflow stream compositions replacing the compositions of $N_{\text{part}}\Delta t/\tau_{\text{res}}$ randomly selected particles, where τ_{res} is the residence time. For premixed combustion cases, the inflow streams consist of a fresh fuel/air mixture stream at a specified temperature and equivalence ratio, and a pilot stream consisting of the adiabatic equilibrium products of the fresh mixture stream, with the mass flow rates of these two streams in a ratio of $0.95 : 0.05$. Non-premixed cases consist of three inflow streams: air, fuel, and a pilot consisting of the adiabatic equilibrium products of a stoichiometric fuel/air mixture at the same unburned temperature as the first two streams; the mass flow rates of these streams occur in a ratio of $0.85 : 0.05 : 0.1$. Following inflow/outflow (for both premixed and non-premixed cases), $\frac{1}{2}N_{\text{part}}\Delta t/\tau_{\text{pair}}$ pairs of particles (not including the inflowing particles) are randomly selected for pairing and then randomly shuffled with the inflowing particles to exchange partners, where τ_{pair} is the pairing timescale.

Although multiple mixing models exist [89], the current mixing fractional step consists of a pair of particles ϕ^p and ϕ^q exchanging compositional information and evolving by

$$\frac{d\phi^p}{dt} = -\frac{\phi^p - \phi^q}{\tau_{\text{mix}}} \quad \text{and} \tag{C.2}$$

$$\frac{d\phi^q}{dt} = -\frac{\phi^q - \phi^p}{\tau_{\text{mix}}} \,, \tag{C.3}$$

where τ_{mix} is a characteristic mixing timescale. The analytical solution to this system of equations determines the particle compositions ϕ^p and ϕ^q after a mixture fractional step:

$$\phi^p = \phi_0^p - \alpha \,, \tag{C.4}$$

$$\phi^q = \phi_0^q + \alpha \,, \text{ and} \tag{C.5}$$

$$\alpha = \frac{\phi_0^p - \phi_0^q}{2}\left[1 - \exp\left(\frac{-2\Delta t_{\text{sub}}}{\tau_{\text{mix}}}\right)\right] \,, \tag{C.6}$$

where ϕ_0^p and ϕ_0^q are the particle compositions at the beginning of the mixture fractional step and Δt_{sub} is the mixing sub-time-step size. The reaction fractional step consists of the

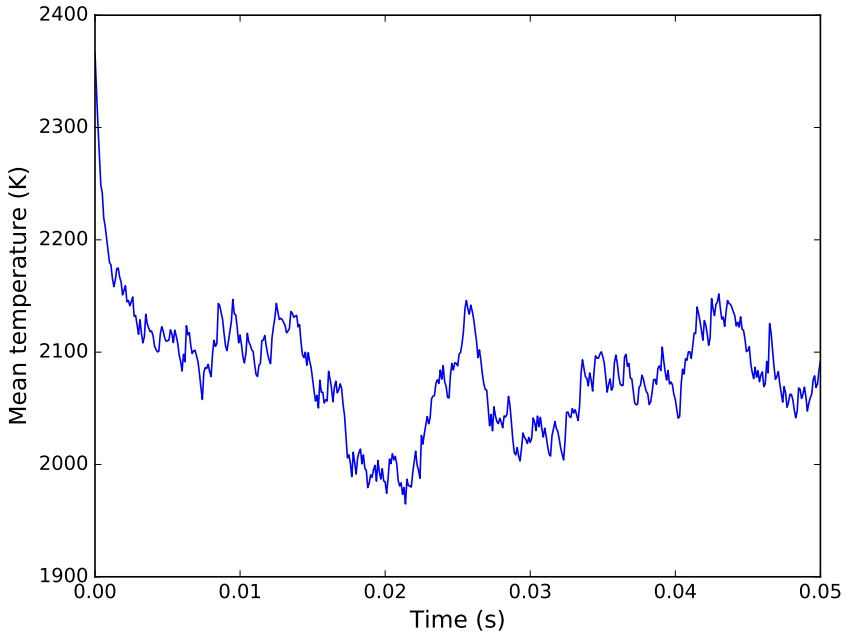


Figure C.1: Mean temperature of premixed PaSR combustion for stoichiometric methane/air with an unburned temperature of 600 K and at 1 atm, $\tau_{\text{res}} = 5\,\text{ms}$, $\tau_{\text{mix}} = \tau_{\text{pair}} = 1\,\text{ms}$, and using 100 particles.

enthalpy evolving by

$$\frac{dh}{dt} = \frac{-1}{\rho} \sum_{k=1}^{N_{\text{sp}}} h_k W_k \dot{\omega}_k \tag{C.7}$$

and the species mass fractions evolving according to Eq. (8). However, in practice our implementation handles the reaction fractional step by advancing in time a Cantera [54] `ReactorNet` that contains a `IdealGasConstPressureReactor` object, rather than integrating the above equations directly.

The time integration scheme implemented in our approach determines the discrete time step between inflow/outflow and pairing events Δt and the sub-time step size Δt_{sub} separating mixing/reaction fractional steps, both held constant in the current implementation, via

$$\Delta t = 0.1 \, \min\left(\tau_{\text{res}}, \tau_{\text{pair}}\right) \text{ and} \tag{C.8}$$

$$\Delta t_{\text{sub}} = 0.04 \, \tau_{\text{mix}} \ , \tag{C.9}$$

adopted from Pope [27].

Figures C.1 and C.2 demonstrate sample results from premixed PaSR combustion of methane/air, using GRI-Mech 3.0 [58]; Fig. C.1 shows the mean temperature evolution over time, while Fig. C.2 shows the temperature distribution among all particles. Although a large number of particles reside near the equilibrium temperature of 2367 K, the wide distribution in particle states is evident.

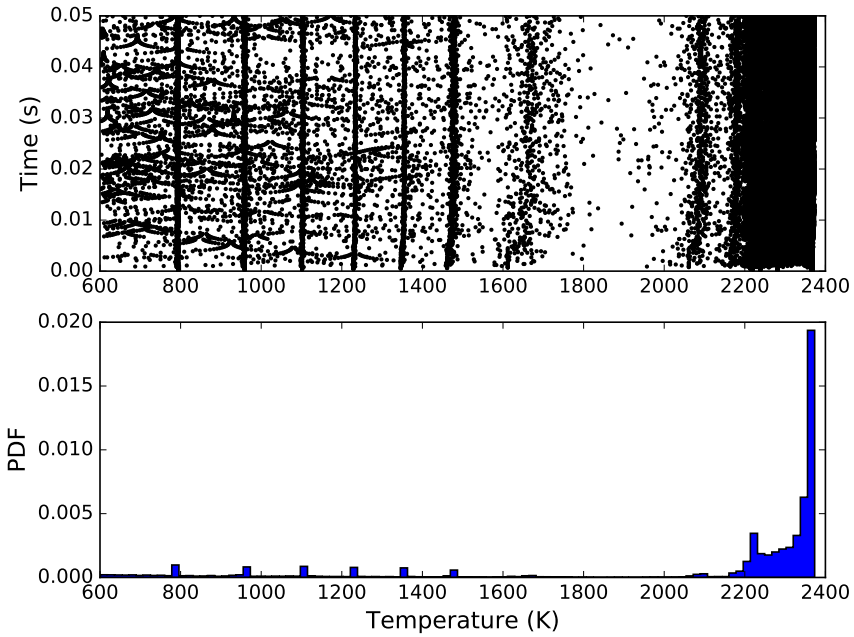


Figure C.2: Scatterplot of temperature over time (top) and probability density function (PDF) of temperature (bottom) of premixed PaSR combustion for stoichiometric methane/ air with an unburned temperature of 600 K and at 1 atm, $\tau_{\text{res}} = 5\,\text{ms}$, $\tau_{\text{mix}} = \tau_{\text{pair}} = 1\,\text{ms}$, and using 100 particles.

Appendix D. Discussion of TChem's thread-safety

During testing, it became evident that TChem v0.2 [83] is not thread-safe for parallelization via OpenMP. In other words, when using OpenMP to parallelize the evaluation of multiple Jacobian matrices, the results differ compared with those obtained from evaluating the same matrices in serial.

To investigate this observed phenomenon further, we created a straightforward test using the H_2/CO model of Burke et al. [65] and a subset of the PaSR data discussed elsewhere in this paper. In short, the testing program first uses TChem to evaluate the Jacobian matrices in serial (i.e., in a single-threaded manner) for 100,100 states sampled from the PaSR data described in Table 2. Next, this process is repeated, but with OpenMP parallelization of the outer loop enabled. To determine whether TChem is thread safe, the computed Jacobian matrices from these two approaches are compared.

We found significant error between the Jacobian matrices computed in serial and with multiple threads for all cases, demonstrating that the current version of TChem is not thread safe. This self-contained test is available openly [91].

References

[1] S. M. Sarathy, C. K. Westbrook, M. Mehl, W. J. Pitz, C. Togbe, P. Dagaut, H. Wang, M. A. Oehlschlaeger, U. Niemann, K. Seshadri, P. S. Veloo, C. Ji, F. N. Egolfopoulos, T. Lu, Comprehensive chemical kinetic modeling of the oxidation of 2-methylalkanes from C_7 to C_{20}, Combust. Flame 158 (12) (2011) 2338–2357. doi:10.1016/j.combustflame.2011.05.007.

[2] M. Mehl, W. J. Pitz, C. K. Westbrook, H. J. Curran, Kinetic modeling of gasoline surrogate components and mixtures under engine conditions, Proc. Combust. Inst. 33 (1) (2011) 193–200. doi:10.1016/j.proci.2010.05.027.

[3] M. Mehl, J.-Y. Chen, W. J. Pitz, S. M. Sarathy, C. K. Westbrook, An approach for formulating surrogates for gasoline with application toward a reduced surrogate mechanism for CFD engine modeling, Energy Fuels 25 (11) (2011) 5215–5223. doi:10.1021/ef201099y.

[4] O. Herbinet, W. J. Pitz, C. K. Westbrook, Detailed chemical kinetic mechanism for the oxidation of biodiesel fuels blend surrogate, Combust. Flame 157 (5) (2010) 893–908. doi:10.1016/j.combustflame.2009.10.013.

[5] T. Lu, C. K. Law, Toward accommodating realistic fuel chemistry in large-scale computations, Prog. Energy Comb. Sci. 35 (2) (2009) 192–215. doi:10.1016/j.pecs.2008.10.002.

[6] T. Turányi, A. S. Tomlin, Analysis of Kinetic Reaction Mechanisms, Springer-Verlag, Berlin Heidelberg, 2014. doi:10.1007/978-3-662-44562-4.

[7] T. Lu, C. K. Law, Linear time reduction of large kinetic mechanisms with directed relation graph: *n*-heptane and iso-octane, Combust. Flame 144 (1-2) (2006) 24–36. doi:10.1016/j.combustflame.2005.02.015.

[8] P. Pepiot-Desjardins, H. Pitsch, An efficient error-propagation-based reduction method for large chemical kinetic mechanisms, Combust. Flame 154 (1–2) (2008) 67–81. doi:10.1016/j.combustflame.2007.10.020.

[9] V. Hiremath, Z. Ren, S. B. Pope, A greedy algorithm for species selection in dimension reduction of combustion chemistry, Combust. Theor. Model. 14 (5) (2010) 619–652. doi:10.1080/13647830.2010.499964.

[10] K. E. Niemeyer, C. J. Sung, M. P. Raju, Skeletal mechanism generation for surrogate fuels using directed relation graph with error propagation and sensitivity analysis, Combust. Flame 157 (9) (2010) 1760–1770. doi:10.1016/j.combustflame.2009.12.022.

[11] T. Lu, C. K. Law, Diffusion coefficient reduction through species bundling, Combust. Flame 148 (3) (2007) 117–126. doi:10.1016/j.combustflame.2006.10.004.

[12] S. S. Ahmed, F. Mauß, G. Moréac, T. Zeuch, A comprehensive and compact n-heptane oxidation model derived using chemical lumping, Phys. Chem. Chem. Phys. 9 (9) (2007) 1107–1126. doi:10.1039/b614712g.

[13] P. Pepiot-Desjardins, H. Pitsch, An automatic chemical lumping method for the reduction of large chemical kinetic mechanisms, Combust. Theor. Model. 12 (6) (2008) 1089–1108. doi:10.1080/13647830802245177.

[14] U. Maas, S. B. Pope, Simplifying chemical kinetics: intrinsic low-dimensional manifolds in composition space, Combust. Flame 88 (3-4) (1992) 239–264. doi:10.1016/0010-2180(92)90034-M.

[15] S.-H. Lam, D. A. Goussis, The CSP method for simplying kinetics, Int. J. Chem. Kinet. 26 (4) (1994) 461–486. doi:10.1002/kin.550260408.

[16] T. Lu, Y. Ju, C. K. Law, Complex CSP for chemistry reduction and analysis, Combust. Flame 126 (1–2) (2001) 1445–1455. doi:10.1016/S0010-2180(01)00252-8.

[17] X. Gou, W. Sun, Z. Chen, Y. Ju, A dynamic multi-timescale method for combustion modeling with detailed and reduced chemical kinetic mechanisms, Combust. Flame 157 (6) (2010) 1111–1121. doi:10.1016/j.combustflame.2010.02.020.

[18] T. Lu, C. K. Law, Strategies for mechanism reduction for large hydrocarbons: n-heptane, Combust. Flame 154 (1–2) (2008) 153–163. doi:10.1016/j.combustflame.2007.11.013.

[19] K. E. Niemeyer, C. J. Sung, Mechanism reduction for multicomponent surrogates: A case study using toluene reference fuels, Combust. Flame 161 (11) (2014) 2752–2764. doi:10.1016/j.combustflame.2014.05.001.

[20] K. E. Niemeyer, C. J. Sung, Reduced chemistry for a gasoline surrogate valid at engine-relevant conditions, Energy Fuels 29 (2) (2015) 1172–1185. doi:10.1021/ef5022126.

[21] I. Banerjee, M. Ierapetritou, An adaptive reduction scheme to model reactive flow, Combust. Flame 144 (3) (2006) 619–633. doi:10.1016/j.combustflame.2005.10.001.

[22] L. Liang, J. Stevens, J. T. Farrell, A dynamic adaptive chemistry scheme for reactive flow computations, Proc. Combust. Inst. 32 (1) (2009) 527–534. doi:10.1016/j.proci.2008.05.073.

[23] Y. Shi, L. Liang, H.-W. Ge, R. D. Reitz, Acceleration of the chemistry solver for modeling DI engine combustion using dynamic adaptive chemistry (DAC) schemes, Combust. Theor. Model. 14 (1) (2010) 69–89. doi:10.1080/13647830903548834.

[24] X. Gou, Z. Chen, W. Sun, Y. Ju, A dynamic adaptive chemistry scheme with error control for combustion modeling with a large detailed mechanism, Combust. Flame 160 (2) (2013) 225–231. doi:10.1080/13647830.2012.733825.

[25] H. Yang, Z. Ren, T. Lu, G. M. Goldin, Dynamic adaptive chemistry for turbulent flame simulations, Combust. Theor. Model. 17 (1) (2013) 167–183. doi:10.1080/13647830.2012.733825.

[26] N. J. Curtis, K. E. Niemeyer, C. J. Sung, An automated target species selection method for dynamic adaptive chemistry simulations, Combust. Flame 162 (2015) 1358–1374. doi:10.1016/j.combustflame.2014.11.004.

[27] S. B. Pope, Computationally efficient implementation of combustion chemistry using in situ adaptive tabulation, Combust. Theor. Model. 1 (1) (1997) 41–63. doi:10.1080/713665229.

[28] C. F. Curtiss, J. O. Hirschfelder, Integration of stiff equations, PNAS 38 (3) (1952) 235–243.

[29] G. D. Byrne, A. C. Hindmarsh, Stiff ODE solvers: a review of current and coming attractions, J. Comput. Phys. 70 (1) (1987) 1–62. doi:10.1016/0021-9991(87)90001-5.

[30] P. N. Brown, G. D. Byrne, A. C. Hindmarsh, VODE: a variable-coefficient ODE solver, SIAM J. Sci. Stat. Comput. 10 (5) (1989) 1038–1051. doi:10.1137/0910062.

[31] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, C. S. Woodward, SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers, ACM T. Math. Software 31 (3) (2005) 363–396. doi:10.1145/1089014.1089020.

[32] E. S. Oran, J. P. Boris, Numerical Simulation of Reactive Flow, 2nd Edition, Cambridge University Press, Cambridge, 2001.

[33] K. Spafford, J. Meredith, J. Vetter, J. Chen, R. Grout, R. Sankaran, Accelerating S3D: A GPGPU case study, in: Euro-Par 2009 Parallel Processing Workshops, LNCS 6043, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 122–131. doi:10.1007/978-3-642-14122-5_16.

[34] Y. Shi, W. H. G. Jr, H.-W. Wong, O. O. Oluwole, Redesigning combustion modeling algorithms for the graphics processing unit (GPU): Chemical kinetic rate evaluation and ordinary differential equation integration, Combust. Flame 158 (5) (2011) 836–847. doi:10.1016/j.combustflame.2011.01.024.

[35] K. E. Niemeyer, C. J. Sung, C. G. Fotache, J. C. Lee, Turbulence-chemistry closure method using graphics processing units: a preliminary test, in: 7th Fall Technical Meeting of the Eastern States Section of the Combustion Institute, 2011.

[36] Y. Shi, W. H. Green, H.-W. Wong, O. O. Oluwole, Accelerating multi-dimensional combustion simulations using GPU and hybrid explicit/implicit ODE integration, Combust. Flame 159 (7) (2012) 2388–2397. doi:10.1016/j.combustflame.2012.02.016.

[37] C. P. Stone, R. L. Davis, Techniques for solving stiff chemical kinetics on graphical processing units, J. Propul. Power 29 (4) (2013) 764–773. doi:10.2514/1.B34874.

[38] K. E. Niemeyer, C. J. Sung, Accelerating moderately stiff chemical kinetics in reactive-flow simulations using GPUs, J. Comput. Phys. 256 (2014) 854–871. doi:10.1016/j.jcp.2013.09.025.

[39] K. E. Niemeyer, C. J. Sung, Recent progress and challenges in exploiting graphics processors in computational fluid dynamics, J. Supercomput. 67 (2) (2014) 528–564. doi:10.1007/s11227-013-1015-7.

[40] C. Safta, H. N. Najm, O. M. Knio, TChem - a software toolkit for the analysis of complex kinetic models, Tech. Rep. SAND2011-3282, Sandia National Laboratories (May 2011).

[41] M. R. Youssefi, Development of analytic tools for computational flame diagnostics, Master's thesis, University of Connecticut, http://digitalcommons.uconn.edu/gs_theses/145/ (Aug. 2011).

[42] F. Bisetti, Integration of large chemical kinetic mechanisms via exponential methods with Krylov approximations to Jacobian matrix functions, Combust. Theor. Model. 16 (3) (2012) 387–418. doi:10.1080/13647830.2011.631032.

[43] F. Perini, E. Galligani, R. D. Reitz, An analytical Jacobian approach to sparse reaction kinetics for computationally efficient combustion modeling with large reaction mechanisms, Energy Fuels 26 (8) (2012) 4804–4822. doi:10.1021/ef300747n.

[44] T. Dijkmans, C. M. Schietekat, K. M. Van Geem, G. B. Marin, GPU based simulation of reactive mixtures with detailed chemistry in combination with tabulation and an analytical Jacobian, Comput. Chem. Eng. 71 (2014) 521–531. doi:10.1016/j.compchemeng.2014.09.016.

[45] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with CUDA, Queue 6 (2) (2008) 40–53. doi:10.1145/1365490.1365500.

[46] C. K. Law, Combustion Physics, Cambridge University Press, New York, 2006.

[47] J. Warnatz, U. Maas, R. W. Dibble, Combustion, 4th Edition, Physical and Chemical Fundamentals, Modeling and Simulation, Exeriments, Pollutant Formation, Springer, Berlin Heidelberg, 2006.

[48] I. Glassman, R. A. Yetter, Combustion, 4th Edition, Academic Press, Burlington, MA, 2008.

[49] S. Gordon, B. J. McBride, Computer program for calculation of complex chemical equilibrium compositions, rocket performance, incident and reflected shocks, and Chapman-Jouguet detonations, Tech. Rep. NASA-SP-273, NASA Lewis Research Center (Mar. 1976).

[50] F. A. Lindemann, S. Arrhenius, I. Langmuir, N. R. Dhar, J. Perrin, W. C. McC Lewis, Discussion on "the radiation theory of chemical action", Trans. Faraday Soc. 17 (1922) 598–606. doi:10.1039/TF9221700598.

[51] R. G. Gilbert, K. Luther, J. Troe, Theory of thermal unimolecular reactions in the fall-off range. II. Weak collision rate constants, Ber. Bunsenges. Phys. Chem. 87 (2) (1983) 169–177. doi:10.1002/bbpc.19830870218.

[52] P. H. Stewart, C. W. Larson, D. M. Golden, Pressure and temperature dependence of reactions proceeding via a bound complex. 2. Application to $2CH_3 \rightarrow C_2H_5 + H$, Combust. Flame 75 (1) (1989) 25–31. doi:10.1016/0010-2180(89)90084-9.

[53] Reaction Design: San Diego, CHEMKIN-PRO 15113 (2012).

[54] D. G. Goodwin, H. K. Moffat, R. L. Speth, Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes, http://www.cantera.org, version 2.2.0 (2015).

[55] P. K. Venkatesh, A. M. Dean, M. H. Cohen, R. W. Carr, Chebyshev expansions and sensitivity analysis for approximating the temperature- and pressure-dependence of chemically-activated reactions, Rev. Chem. Eng. 13 (1) (1997) 1–67. doi:10.1515/REVCE.1997.13.1.1.

[56] P. K. Venkatesh, A. Y. Chang, A. M. Dean, M. H. Cohen, R. W. Carr, Parameterization of pressure- and temperature-dependent kinetics in multiple well reactions, AICHe J. 43 (5) (1997) 1331–1340. doi:10.1002/aic.690430522.

[57] P. K. Venkatesh, Damped pseudospectral functional forms of the falloff behavior of unimolecular reactions, J. Phys. Chem. A 104 (2) (2000) 280–287. doi:10.1021/jp991458u.

[58] G. P. Smith, D. M. Golden, M. Frenklach, N. W. Moriarty, B. Eiteneer, M. Goldenberg, C. T. Bowman, R. K. Hanson, S. Song, W. C. Gardiner, V. V. Lissianski, Z. Qin, GRI-Mech 3.0, http://www.me.berkeley.edu/gri_mech/.

[59] Python Software Foundation, Python language reference, version 2.7.10, Available at http://www.python.org (2015).

[60] K. E. Niemeyer, N. J. Curtis, `pyJac` v1.0.2 (Jan. 2017). doi:10.5281/zenodo.251144.

[61] S. van der Walt, S. C. Colbert, G. Varoquaux, The NumPy array: A structure for efficient numerical computation, Comput. Sci. Eng. 13 (2011) 22–30. doi:10.1109/MCSE.2011.37.

[62] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, K. Smith, Cython: The best of both worlds, Comput. Sci. Eng. 13 (2011) 31–39. doi:10.1109/MCSE.2010.118.

[63] K. Simonov, PyYAML version 3.11, Available at http://pyyaml.org (Mar. 2014).

[64] R. J. Hogan, `Adept` v1.1, Available at https://github.com/rjhogan/Adept (Jun. 2015).

[65] M. P. Burke, M. Chaos, Y. Ju, F. L. Dryer, S. J. Klippenstein, Comprehensive H_2/O_2 kinetic model for high-pressure combustion, Int. J. Chem. Kinet. 44 (7) (2011) 444–474. doi:10.1002/kin.20603.

[66] H. Wang, X. You, A. V. Joshi, S. G. Davis, A. Laskin, F. Egolfopoulos, C. K. Law, USC Mech Version II. High-temperature combustion reaction model of $H_2/CO/C_1$–C_4 compounds, http://ignis.usc.edu/USC_Mech_II.htm (May 2007).

[67] S. M. Sarathy, S. Park, B. W. Weber, W. Wang, P. S. Veloo, A. C. Davis, C. Togbe, C. K. Westbrook, O. Park, G. Dayma, Z. Luo, M. A. Oehlschlaeger, F. N. Egolfopoulos, T. Lu, W. J. Pitz, C. J. Sung, P. Dagaut, A comprehensive experimental and modeling study of iso-pentanol combustion, Combust. Flame 160 (12) (2013) 2712–2728. doi:10.1016/j.combustflame.2013.06.022.

[68] L. F. Shampine, Numerical Solution of Ordinary Differential Equations, Chapman & Hall, New York, 1994.

[69] S.-H. Lam, D. A. Goussis, Understanding complex chemical kinetics with computational singular perturbation, Symp. (Int.) Combust. 22 (1988) 931–941. doi:10.1016/S0082-0784(89)80102-X.

[70] S.-H. Lam, Using CSP to understand complex chemical kinetics, Combust. Sci. Technol. 89 (5-6) (1993) 375–404. doi:10.1080/00102209308924120.

[71] S.-H. Lam, Reduced chemistry-diffusion coupling, Combust. Sci. Technol. 179 (4) (2007) 767–786. doi:10.1080/00102200601093498.

[72] T. Lu, C. S. Yoo, J. H. Chen, C. K. Law, Three-dimensional direct numerical simulation of a turbulent lifted hydrogen jet flame in heated coflow: a chemical explosive mode analysis, J. Fluid Mech. 652 (2010) 45–64. doi:10.1017/S002211201000039X.

[73] Z. Luo, C. S. Yoo, E. S. Richardson, J. H. Chen, C. K. Law, T. Lu, Chemical explosive mode analysis for a turbulent lifted ethylene jet flame in highly-heated coflow, Combust. Flame 159 (1) (2012) 265–274. doi:10.1016/j.combustflame.2011.05.023.

[74] R. Shan, C. S. Yoo, J. H. Chen, T. Lu, Computational diagnostics for n-heptane flames with chemical explosive mode analysis, Combust. Flame 159 (10) (2012) 3119–3127. doi:10.1016/j.combustflame.2012.05.012.

[75] R. J. Hogan, Fast reverse-mode automatic differentiation using expression templates in C++, ACM Trans. Math. Software 40 (4) (2014) 26. doi:10.1145/2560359.

[76] L. F. Richardson, The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam, Phil. Trans. R. Soc. A 210 (1911) 307–357.

[77] J. N. Lyness, C. B. Moler, Van der Monde systems and numerical differentiation, Numer. Math. 8 (5) (1966) 458–464. doi:10.1007/BF02166671.

[78] J. N. Lyness, C. B. Moler, Generalized Romberg methods for integrals of derivatives, Numer. Math. 14 (1) (1969) 1–13. doi:10.1007/BF02165095.

[79] J. R. R. A. Martins, P. Sturdza, J. J. Alonso, The complex-step derivative approximation, ACM Trans. Math. Software 29 (3) (2003) 245–262. doi:10.1145/838250.838251.

[80] L. F. Shampine, Accurate numerical derivatives in MATLAB, ACM T. Math. Software 33 (4) (2007) 26. doi:10.1145/1268776.1268781.

[81] M. S. Ridout, Statistical applications of the complex-step method of numerical differentiation, Am. Stat. 63 (1) (2009) 66–74. doi:10.1198/tast.2009.0013.

[82] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, 3rd Edition, SIAM, Philadelphia, PA, 1999.

[83] C. Safta, H. N. Najm, O. M. Knio, TChem v0.2, Available at http://www.sandia.gov/tchem/index.html (2011).

[84] Free Software Foundation, gcc-4.8.5, Available at https://gcc.gnu.org/onlinedocs/gcc-4.8.5/gcc/ (Jun. 2015).

[85] NVIDIA, CUDA Toolkit 7.5, Available at https://developer.nvidia.com/cuda-downloads (Sep. 2015).

[86] S. M. Correa, Turbulence-chemistry interactions in the intermediate regime of premixed combustion, Combust. Flame 93 (1–2) (1993) 41–60. doi:10.1016/0010-2180(93)90083-F.

[87] J.-Y. Chen, Stochastic modeling of partially stirred reactors, Combust. Sci. Technol. 122 (1–6) (1997) 63–94. doi:10.1080/00102209708935605.

[88] A. Bhave, M. Kraft, Partially stirred reactor model: Analytical solutions and numerical convergence study of a PDF/Monte Carlo method, SIAM J. Sci. Comput. 25 (5) (2004) 1798–1823. doi:10.1137/S1064827502411328.

[89] Z. Ren, S. B. Pope, An investigation of the performance of turbulent mixing models, Combust. Flame 136 (1-2) (2004) 208–216. doi:10.1016/j.combustflame.2003.09.014.

[90] Z. Ren, Y. Liu, T. Lu, L. Lu, O. O. Oluwole, G. M. Goldin, The use of dynamic adaptive chemistry and tabulation in reactive flow simulations, Combust. Flame 161 (1) (2014) 127–137. doi:10.1016/j.combustflame.2013.08.018.

[91] N. J. Curtis, K. E. Niemeyer, Fileset for testing thread-safety of TChem, figshare (Jan. 2017). doi:10.6084/m9.figshare.4563982.v1.