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 exhibit both mathematical stiffness in the governing differential equations and large numbers of species and reactions, particularly for larger 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; in addition, various computational combustion diagnostics methods require accurate Jacobian matrices. 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. First, we 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 1×10^{-3} % 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 2.9–5.3 times and the existing analytical Jacobian software `TChem` by 3.9–41 times. 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 adoption.

Keywords: Chemical kinetics, Jacobian, SIMD, GPU

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

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:
Programming language: Python
Computer: Any
Operating system: Any (Linux, OS X, Windows)
RAM: bytes
Keywords: Chemical kinetics, Jacobian
Classification: 16.12, 4.3
External routines/libraries:
Nature of problem:

Solution method:

Additional comments:

Running time:

*Todo list

1. Introduction

As the need for detailed and accurate chemical kinetics models [1] in predictive reactive-flow simulations has become recognized in recent years, simultaneously such models describing the oxidation of hydrocarbon fuels 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 [**?**]; similarly large surrogate models exist for gasoline [**?** **?**] and biodiesel [**?**]. 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 this 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 re-

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

taining predictiveness, as reviewed by Lu and Law [?] as well as Turï£¡nyi and Tomlin [?]. Major classes of such approaches include skeletal reduction methods that remove unimportant species and reactions [? ? ? ?], lumping of species that share similar properties [? ? ?], and time-scale reduction methods that reduce chemical stiffness [? ? ? ?]. Strategies for achieving significant reduction combine such methods a priori in multiple stages [? ? ?], or apply them dynamically during a simulation to achieve greater local savings [? ? ? ? ? ?]. Techniques such as interpolation/tabulation of expensive terms [?] can also reduce computational costs.

In addition to the aforementioned cost reduction methods that modify the chemical kinetic models, focusing on 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, typically solvers rely on robust, high-order implicit integration algorithms based on backward differentiation formulae [? ? ? ?]. 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 drops from growing with the square of the number of species to a linear dependence [?].

In parallel to the potential improvements in stiff implicit integrators used for chemical kinetics, developing algorithms tailored for high-performance hardware accelerators offers 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 multi-core parallelism brought about some CPU performance gains, single-instruction multiple data processors (SIMD), e.g., graphics processing units (GPUs), recently emerged as a low-cost, low-power, 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 [?], a number of studies in recent years [? ? ? ? ? ?] explored the use of SIMD processors to accelerate the integration of chemical kinetics in reactive-flow codes. While explicit methods offer significant improvements in performance for nonstiff and moderately stiff chemical kinetics [?], 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. [?] 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 [?] recognized the importance of using an analytical Jacobian over a numerical approximation both for reduction in computational cost

and numerical error reduction when performing eigendecomposition of the matrix. Bisetti [?] 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 reacting-flow formulations, this formulation resulted in a significant increase of Jacobian matrix sparsity—such a strategy should be investigated further. Perini et al. [?] 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. [?] 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. [?]. Unlike the current work, the approach of Dijkmans et al. [?] 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 species and reaction rates, derivative source terms, and analytical chemical Jacobian matrices both for CPU operation via C/C++ and GPU operation via CUDA [?], a widely used programming language for NVIDIA GPUs. Furthermore, to the our knowledge, unlike `pyJac` none of the previously discussed analytical Jacobian generation implementations support the 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. 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 [? ? ?], 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 [?], is described using an ordinary differential equation for the thermochemical composition vector

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

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 [**?**]. 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; thus, we use Eq. (3) to determine density rather than including it in the differential system (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

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

The system of ODEs governing the 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

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

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

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

2.2. Thermodynamic properties

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

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

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

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

where $C_{p,l}$ 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 standard-state one atmosphere. For a calorically perfect gas, the standard-state specific heats, enthalpies, and internal energies are also the actual values.

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

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

2.3. Reaction rate expressions

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

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

where $N_{\rm reac}$ 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 a reversible reaction is

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

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

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

and the third-body/pressure modification c_i is given by

$$c_i = \begin{cases} 1 & \text{for elementary reactions,} \\ [X]_i & \text{for } 3^{\text{rd}}\text{-body enhanced reactions,} \\ \dfrac{P_{ri}}{1+P_{ri}} F_i & \text{for unimolecular/recombination falloff reactions, and} \\ \dfrac{1}{1+P_{ri}} F_i & \text{chemically-activated bimolecular reactions.} \end{cases} \tag{20}$$

The forward reaction rate coefficient $k_{f,i}$ is given by the 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 [**?**], depending on the value of the Arrhenius parameters, k_f can be calculated in different ways to minimize the computational cost:

$$k_f = \begin{cases} A & \text{if } \beta = 0 \text{ and } T_a = 0 \, , \\ \exp\left(\log A + \beta \log T\right) & \text{if } \beta \neq 0 \text{ and if } T_a = 0 \, , \\ \exp\left(\log A + \beta \log T - T_a/T\right) & \text{if } \beta \neq 0 \text{ and } T_a \neq 0 \, , \\ \exp\left(\log A - T_a/T\right) & \text{if } \beta = 0 \text{ and } T_a \neq 0 \, , \text{ and} \\ A \prod^{b} T & \text{if } T_a = 0 \text{ and } b \in \mathbb{Z} \text{ (integers).} \end{cases} \tag{22}$$

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,

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

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

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

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

$$\begin{aligned} K_{c,i} &= \left(\frac{p_{\text{atm}}}{\mathcal{R}T}\right)^{\sum_{k=1}^{N_{\text{sp}}} \nu_{ki}} \exp\left(\sum_{k=1}^{N_{\text{sp}}} \nu_{ki} \left(\frac{S_k^\circ}{\mathcal{R}} - \frac{H_k^\circ}{\mathcal{R}T}\right)\right) \\ &= \left(\frac{p_{\text{atm}}}{\mathcal{R}}\right)^{\sum_{k=1}^{N_{\text{sp}}} \nu_{ki}} \exp\left(\sum_{k=1}^{N_{\text{sp}}} \nu_{ki} B_k\right) , \end{aligned} \tag{26}$$

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

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

2.3.2. Third-body effects

A three-body reaction contains "+M" in the reaction line description. In this case,

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

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,

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

2.3.3. Falloff reactions

Unlike elementary and third-body reactions, falloff reactions exhibit a pressure dependence described as a blending of rates as 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

$$P_{r,i} = \begin{cases} \dfrac{k_{0,i}}{k_{\infty,i}}[X]_i & \text{if "(+M)", or} \\ \dfrac{k_{0,i}}{k_{\infty,i}}[X_m] & \text{if "(+Y}_m\text{)".} \end{cases} \tag{30}$$

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

$$F_i = \begin{cases} 1 & \text{for Lindemann,} \\ F_{\text{cent}}^{\left(1+(A/B)^2\right)^{-1}} & \text{for Troe, or} \\ dT^e\left(a \cdot \exp\left(-\dfrac{b}{T}\right) + \exp\left(-\dfrac{T}{c}\right)\right)^X & \text{for SRI.} \end{cases} \tag{31}$$

The variables for the Troe representation are given by

$$F_{\text{cent}} = (1-a)\exp\left(-\frac{T}{T^{***}}\right) + a \cdot \exp\left(-\frac{T}{T^*}\right) + \exp\left(-\frac{T^{**}}{T}\right) \,, \tag{32}$$

$$A = \log_{10} P_{r,i} - 0.67\log_{10} F_{\text{cent}} - 0.4 \,, \text{and} \tag{33}$$

$$B = 0.806 - 1.1762\log_{10} F_{\text{cent}} - 0.14\log_{10} P_{r,i} \,, \tag{34}$$

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 expression used in the SRI representation is given by

$$X = \left(1 + (\log_{10} P_{r,i})^2\right)^{-1} \tag{35}$$

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 [? ?], each evaluated using Eq. (21):

$$k_1(T) = A_1 T^{\beta_1} \exp\left(-\frac{T_{a,1}}{T}\right) \text{ at } p_1 \text{ and} \tag{36}$$

$$k_2(T) = A_2 T^{\beta_2} \exp\left(-\frac{T_{a,2}}{T}\right) \text{ at } p_2 \ , \tag{37}$$

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:

$$\log k_{f,i}(T,p) = \log k_1(T) + (\log k_2(T) - \log k_1(T)) \frac{\log p - \log p_1}{\log p_2 - \log p_1} \ . \tag{38}$$

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

$$\log_{10} k_f(T,p) = \sum_{i=1}^{N_T} \sum_{j=1}^{N_p} \alpha_{ij} \phi_i(\tilde{T}) \phi_j(\tilde{p}) \ , \tag{39}$$

where α_{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

$$\phi_n(x) = T_{n-1}(x) = \cos\left((n-1)\arccos(x)\right) \quad \text{for } |x| \le 1 \ . \tag{40}$$

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

$$\tilde{T} \equiv \frac{2T^{-1} - T_{\min}^{-1} - T_{\max}^{-1}}{T_{\max}^{-1} - T_{\min}^{-1}} \quad \text{and} \tag{41}$$

$$\tilde{p} \equiv \frac{2\log_{10} p - \log_{10} p_{\min} - \log_{10} p_{\max}}{\log_{10} p_{\max} - \log_{10} p_{\min}} \ , \tag{42}$$

where $T_{\min} \le T \le T_{\max}$ and $p_{\min} \le p \le p_{\max}$ 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

$$\mathcal{J}_{i,j} = \frac{\partial f_i}{\partial \Phi_j} \,. \tag{43}$$

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

$$\mathcal{J}_{1,j} = \frac{\partial \dot{T}}{\partial \Phi_j} \quad j = 1, \ldots, N_{\mathrm{sp}} - 1 \,. \tag{44}$$

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

$$\begin{aligned}
\mathcal{J}_{1,1} &= \frac{\partial f_1}{\partial T} = \frac{\partial}{\partial T}\left(-\frac{\sum_{k=1}^{N_{\mathrm{sp}}} h_k W_k \dot{\omega}_k}{\rho c_p}\right) = -\sum_{k=1}^{N_{\mathrm{sp}}} \frac{\partial}{\partial T}\left(\frac{h_k}{c_p}\frac{W_k\dot{\omega}_k}{\rho}\right) \\
&= -\sum_{k=1}^{N_{\mathrm{sp}}}\left[\frac{\partial}{\partial T}\left(\frac{h_k}{c_p}\right)\frac{W_k\dot{\omega}_k}{\rho} + \frac{h_k}{c_p}\frac{\partial}{\partial T}\left(\frac{W_k\dot{\omega}_k}{\rho}\right)\right] \\
&= \frac{-1}{c_p}\sum_{k=1}^{N_{\mathrm{sp}}}\left[\left(\frac{\partial h_k}{\partial T} - h_k\frac{\partial c_p}{\partial T}\right)\frac{W_k\dot{\omega}_k}{\rho} + \frac{h_k}{c_p}\frac{\partial}{\partial T}\left(\frac{W_k\dot{\omega}_k}{\rho}\right)\right],
\end{aligned} \tag{45}$$

$$\begin{aligned}
\mathcal{J}_{1,j+1} &= \frac{\partial f_1}{\partial Y_j} = \frac{\partial}{\partial Y_j}\left(-\frac{\sum h_k W_k \dot{\omega}_k}{\rho c_p}\right) = -\sum_{k=1}^{N_{\mathrm{sp}}} \frac{\partial}{\partial Y_j}\left(\frac{h_k}{c_p}\frac{W_k\dot{\omega}_k}{\rho}\right) \\
&= -\sum_{k=1}^{N_{\mathrm{sp}}}\left[\frac{\partial}{\partial Y_j}\left(\frac{h_k}{c_p}\right)\frac{W_k\dot{\omega}_k}{\rho} + \frac{h_k}{c_p}\frac{\partial}{\partial Y_j}\left(\frac{W_k\dot{\omega}_k}{\rho}\right)\right] \\
&= \frac{-1}{c_p}\sum_{k=1}^{N_{\mathrm{sp}}}\left[\left(\frac{\partial h_k}{\partial Y_j} - h_k\frac{\partial c_p}{\partial Y_j}\right)\frac{W_k\dot{\omega}_k}{\rho} + \frac{h_k}{c_p}\frac{\partial}{\partial Y_j}\left(\frac{W_k\dot{\omega}_k}{\rho}\right)\right],
\end{aligned} \tag{46}$$

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

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

$$\begin{aligned}
\mathcal{J}_{k+1,1} &= \frac{\partial f_{k+1}}{\partial T} = \frac{\partial}{\partial T}\left(\frac{W_k\dot{\omega}_k}{\rho}\right) \\
&= \frac{W_k}{\rho}\left(\frac{\partial\dot{\omega}_k}{\partial T} - \frac{\dot{\omega}_k}{\rho}\frac{\partial\rho}{\partial T}\right), \\
\mathcal{J}_{k+1,j+1} &= \frac{\partial f_{k+1}}{\partial Y_j} = \frac{\partial}{\partial Y_j}\left(\frac{W_k\dot{\omega}_k}{\rho}\right)
\end{aligned} \tag{47}$$

$$= \frac{W_k}{\rho}\left(\frac{\partial \dot{\omega}_k}{\partial Y_j} - \frac{\dot{\omega}_k}{\rho}\frac{\partial \rho}{\partial Y_j}\right), \tag{48}$$

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

The following partial derivatives need to be evaluated: $\frac{\partial \rho}{\partial T}$, $\frac{\partial \rho}{\partial Y_j}$, $\frac{\partial c_p}{\partial T}$, $\frac{\partial c_p}{\partial Y_j}$, $\frac{\partial h_k}{\partial T}$, $\frac{\partial h_k}{\partial Y_j}$, $\frac{\partial \dot{\omega}_k}{\partial T}$, and $\frac{\partial \dot{\omega}_k}{\partial Y_j}$. The partial derivatives of density are:

$$\begin{aligned}\frac{\partial \rho}{\partial T} &= \frac{\partial}{\partial T}\left(\frac{pW}{\mathcal{R}T}\right) = \frac{pW}{\mathcal{R}}\frac{-1}{T^2} \\ &= -\frac{\rho}{T} \end{aligned} \tag{49}$$

$$\begin{aligned}\frac{\partial \rho}{\partial Y_j} &= \frac{\partial}{\partial Y_j}\left(\frac{pW}{\mathcal{R}T}\right) = \frac{-pW^2}{\mathcal{R}T}\left(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\right) \\ &= -\rho W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\right) \end{aligned} \tag{50}$$

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:

$$\frac{\partial h_k}{\partial T} = c_{p,k} \tag{51}$$

$$\frac{\partial h_k}{\partial Y_j} = 0 \tag{52}$$

Next, the derivatives of specific heat:

$$\frac{\partial c_p}{\partial T} = \sum_{k=1}^{N_{\text{sp}}} Y_k \frac{\partial c_{p,k}}{\partial T} \tag{53}$$

$$\frac{\partial c_{p,k}}{\partial T} = \frac{\mathcal{R}}{W_k}\left(a_{1,k} + T\left(2a_{2,k} + T\left(3a_{3,k} + 4a_{4,k}T\right)\right)\right) \tag{54}$$

$$\frac{\partial c_p}{\partial Y_j} = \frac{\partial}{\partial Y_j}\sum_{k=1}^{N_{\text{sp}}} Y_k c_{p,k} = c_{p,j} - c_{p,N_{\text{sp}}} \tag{55}$$

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

$$\frac{\partial \dot{\omega}_k}{\partial T} = \sum_{i=1}^{N_{\text{reac}}} \nu_{ki}\frac{\partial q_i}{\partial T} = \sum_{i=1}^{N_{\text{reac}}} \nu_{ki}\left(\frac{\partial c_i}{\partial T}R_i + c_i\frac{\partial R_i}{\partial T}\right) \tag{56}$$

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

The partial derivatives of species concentration are

$$\frac{\partial [X_i]}{\partial T} = \frac{Y_i}{W_i}\frac{\partial \rho}{\partial T} = -\frac{Y_i}{W_i}\frac{\rho}{T} = -\frac{[X_i]}{T} \tag{58}$$

$$\frac{\partial [X_i]}{\partial Y_j} = \frac{Y_i}{W_i}\frac{\partial \rho}{\partial Y_j} + \frac{\rho}{W_i}\frac{\partial Y_i}{\partial Y_j} = \frac{\rho}{W_i}\left[-Y_i W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right) + \left(\delta_{ij} - \delta_{iN_{\rm sp}}\right)\right]$$
$$= -[X_i]W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right) + \left(\delta_{ij} - \delta_{iN_{\rm sp}}\right)\frac{\rho}{W_i}\,, \tag{59}$$

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 R_i vary depending on whether the reaction is reversible. For irreversible reactions,

$$\frac{\partial R_{f,i}}{\partial T} = \frac{\partial k_{f,i}}{\partial T}\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}} + k_{f,i}\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}}\right)$$
$$= \frac{\partial k_{f,i}}{\partial T}\frac{1}{k_{f,i}}R_{f,i} + k_{f,i}\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}}\right) \text{ and} \tag{60}$$

$$\frac{\partial R_{f,i}}{\partial Y_j} = k_{f,i}\frac{\partial}{\partial Y_j}\left(\prod_{k=1}^{N_{\rm sp}}[X_k]^{\nu'_{ki}}\right)\,. \tag{61}$$

Theoretically, $\frac{\partial R_{f,i}}{\partial [X_j]}$ could be further simplified to $\frac{\nu'_{ji}}{[X_j]}R_{f,i}$, but practically this should be avoided since $[X_j]$ could be zero. The partial derivatives of the molar concentration product terms above are

$$\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}}\right) = \sum_{j=1}^{N_{\rm sp}}\nu'_{ji}[X_j]^{\nu'_{ji}-1}\left(\frac{-[X_j]}{T}\right)\prod_{\substack{k=1\\k\neq j}}^{N_{\rm sp}}[X_k]^{\nu'_{ki}}$$
$$= -\frac{1}{T}\left(\prod_{j=1}^{N_{\rm sp}}[X_j]^{\nu'_{ji}}\right)\sum_{j=1}^{N_{\rm sp}}\nu'_{ji} \quad \text{and} \tag{62}$$

$$\frac{\partial}{\partial Y_j}\left(\prod_{k=1}^{N_{\rm sp}}[X_k]^{\nu'_{ki}}\right) = \sum_{k=1}^{N_{\rm sp}}\nu'_{ki}[X_k]^{\nu'_{ki}-1}\left(\frac{\partial [X_k]}{\partial Y_j}\right)\prod_{\substack{l=1\\l\neq k}}^{N_{\rm sp}}[X_l]^{\nu'_{li}}$$
$$= \sum_{k=1}^{N_{\rm sp}}\nu'_{ki}\Bigg(-[X_k]^{\nu'_{ki}}W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right)$$
$$+ \left(\delta_{kj} - \delta_{kN_{\rm sp}}\right)\frac{\rho}{W_k}[X_k]^{\nu'_{ki}-1}\Bigg)\prod_{\substack{l=1\\l\neq k}}^{N_{\rm sp}}[X_l]^{\nu'_{li}}\,, \tag{63}$$

and the partial derivative of forward reaction rate coefficient is

$$\frac{\partial k_{f,i}}{\partial T} = \frac{\partial}{\partial T}\left(A_i \exp\left(\beta_i \ln T - \frac{T_{a,i}}{T}\right)\right)$$
$$= \frac{k_{f,i}}{T}\left(\beta_i + \frac{T_{a,i}}{T}\right)\,. \tag{64}$$

Inserting these into Eqs. (60) and (61) gives

$$\begin{aligned}\frac{\partial R_{f,i}}{\partial T} &= \frac{1}{T}\left(\beta_i \frac{T_{a,i}}{T}\right) R_{f,i} - \frac{k_{f,i}}{T}\left(\prod_{j=1}^{N_{\rm sp}} [X_j]^{\nu'_{ji}}\right)\sum_{j=1}^{N_{\rm sp}} \nu'_{ji} \\ &= \frac{R_{f,i}}{T}\left(\beta_i + \frac{T_{a,i}}{T} - \sum_{j=1}^{N_{\rm sp}} \nu'_{ji}\right) \text{ and} \qquad (65)\\ \frac{\partial R_{f,i}}{\partial Y_j} &= \sum_{k=1}^{N_{\rm sp}} \nu'_{ki}\Bigg(-R_{f,i} W \left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right) \\ &\quad + \left(\delta_{kj} - \delta_{kN_{\rm sp}}\right) k_{f,i} \frac{\rho}{W_k} [X_k]^{\nu'_{ki}-1} \prod_{\substack{l=1 \\ l\neq k}}^{N_{\rm sp}} [X_l]^{\nu'_{li}}\Bigg) . \qquad (66)\end{aligned}$$

For reversible reactions with explicit reverse Arrhenius coefficients,

$$\begin{aligned}\frac{\partial R_i}{\partial T} &= \frac{\partial k_{f,i}}{\partial T}\prod_{j=1}^{N_{\rm sp}} [X_j]^{\nu'_{ji}} + k_{f,i}\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\rm sp}} [X_j]^{\nu'_{ji}}\right) \\ &\quad - \frac{\partial k_{r,i}}{\partial T}\prod_{j=1}^{N_{\rm sp}} [X_j]^{\nu''_{ji}} - k_{r,i}\frac{\partial}{\partial T}\left(\prod_{j=1}^{N_{\rm sp}} [X_j]^{\nu''_{ji}}\right) \qquad (67)\\ &= \frac{R_{f,i}}{T}\left(\beta_{f,i} + \frac{T_{af,i}}{T} - \sum_{j=1}^{N_{\rm sp}} \nu'_{ji}\right) - \frac{R_{r,i}}{T}\left(\beta_{r,i} + \frac{T_{ar,i}}{T} - \sum_{j=1}^{N_{\rm sp}} \nu''_{ji}\right) . \qquad (68)\end{aligned}$$

(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:

$$\begin{aligned}\frac{\partial k_{r,i}}{\partial T} &= \frac{\partial}{\partial T}\left(\frac{k_{f,i}}{K_{c,i}}\right) = \frac{\frac{\partial k_{f,i}}{\partial T} K_{c,i} - \frac{\partial K_{c,i}}{\partial T} k_{f,i}}{K_{c,i}^2} \qquad (70)\\ &= \frac{k_{f,i}}{K_{c,i}}\frac{1}{T}\left(\beta_i + \frac{T_{a,i}}{T}\right) - \frac{1}{K_{c,i}}\frac{\partial K_{c,i}}{\partial T}\frac{k_{f,i}}{K_{c,i}} \\ \frac{\partial K_{c,i}}{\partial T} &= \left(\frac{p_{\rm atm}}{\mathcal{R}}\right)^{\sum \nu_{ki}}\left(\sum_{k=1}^{N_{\rm sp}} \nu_{ki}\frac{\partial B_k}{\partial T}\right)\exp\left(\sum_{k=1}^{N_{\rm sp}} \nu_{ki} B_k\right) \\ &= K_{c,i}\sum_{k=1}^{N_{\rm sp}} \nu_{ki}\frac{\partial B_k}{\partial T} \qquad (71)\\ \therefore \frac{\partial k_{r,i}}{\partial T} &= k_{r,i}\left(\frac{1}{T}\left(\beta_i + \frac{T_{a,i}}{T}\right) - \sum_{k=1}^{N_{\rm sp}} \nu_{ki}\frac{\partial B_k}{\partial T}\right) , \qquad (72)\end{aligned}$$

where

$$\frac{\partial B_k}{\partial T} = \frac{1}{T}\left(a_{0,k} - 1 + \frac{a_{5,k}}{T}\right) + \frac{a_{1,k}}{2} + T\left(\frac{a_{2,k}}{3} + T\left(\frac{a_{3,k}}{4} + \frac{a_{4,k}}{5}T\right)\right) . \qquad (73)$$

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

$$\frac{\partial R_i}{\partial T} = \frac{R_{f,i}}{T}\left(\beta_i + \frac{T_{a,i}}{T} - \sum_{j=1}^{N_{\text{sp}}} \nu'_{ji}\right) - R_{r,i}\left(\frac{1}{T}\left(\beta_i + \frac{T_{a,i}}{T} - \sum_{j=1}^{N_{\text{sp}}} \nu''_{ji}\right) - \sum_{j=1}^{N_{\text{sp}}} \nu_{ji}\frac{\partial B_j}{\partial T}\right) . \tag{74}$$

For all reversible reactions,

$$\begin{aligned}\frac{\partial R_i}{\partial Y_j} = \sum_{k=1}^{N_{\text{sp}}}\Bigg(&\nu'_{ki}\Bigg(-R_{f,i}W\Big(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\Big) \\ &+ \left(\delta_{kj} - \delta_{kN_{\text{sp}}}\right) k_{f,i}\frac{\rho}{W_k}[X_k]^{\nu'_{ki}-1}\prod_{\substack{l=1\\l\neq k}}^{N_{\text{sp}}}[X_l]^{\nu'_{li}}\Bigg) \\ &- \nu''_{ki}\Bigg(-R_{r,i}W\Big(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\Big) \\ &+ \left(\delta_{kj} - \delta_{kN_{\text{sp}}}\right) k_{r,i}\frac{\rho}{W_k}[X_k]^{\nu''_{ki}-1}\prod_{\substack{l=1\\l\neq k}}^{N_{\text{sp}}}[X_l]^{\nu''_{li}}\Bigg)\Bigg) .\end{aligned} \tag{75}$$

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

$$\frac{\partial c_i}{\partial T} = 0 \text{ and} \tag{76}$$

$$\frac{\partial c_i}{\partial Y_j} = 0 . \tag{77}$$

For third-body-enhanced reactions,

$$\frac{\partial c_i}{\partial T} = \frac{-1}{T}\sum_{j=1}^{N_{\text{sp}}}\alpha_{ij}[X_j] = -\frac{c_i}{T} \text{ , and} \tag{78}$$

$$\begin{aligned}\frac{\partial c_i}{\partial Y_j} &= \sum_{k=1}^{N_{\text{sp}}}\alpha_{ik}\left(-[X_i]W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\right) + \left(\delta_{ij} - \delta_{iN_{\text{sp}}}\right)\frac{\rho}{W_i}\right) \\ &= -Wc_i\Big(\frac{1}{W_j} - \frac{1}{W_{N_{\text{sp}}}}\Big) + \rho\Big(\frac{\alpha_{ij}}{W_j} - \frac{\alpha_{iN_{\text{sp}}}}{W_{N_{\text{sp}}}}\Big) .\end{aligned} \tag{79}$$

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

$$\frac{\partial c_i}{\partial Y_j} = \frac{\partial}{\partial Y_j}\left(\frac{p}{\mathcal{R}T}\right) = 0 , \tag{80}$$

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

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

$$\frac{\partial c_i}{\partial T} = c_i \left(\frac{1}{P_{r,i}(1+P_{r,i})} \frac{\partial P_{r,i}}{\partial T} + \frac{1}{F_i} \frac{\partial F_i}{\partial T} \right) \tag{81}$$

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

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

$$\begin{aligned} \frac{\partial P_{r,i}}{\partial T} &= \frac{1}{k_{\infty,i}} \left(\frac{\partial k_{0,i}}{\partial T}[X]_i + k_{0,i} \frac{\partial [X]_i}{\partial T} - \frac{k_{0,i}}{k_{\infty,i}}[X]_i \frac{\partial k_{\infty,i}}{\partial T} \right) \\ &= \frac{P_{r,i}}{T} \left(\beta_{0,i} - \beta_{\infty,i} + \frac{T_{a0,i} - T_{a\infty,i}}{T} - 1 \right) \end{aligned} \tag{83}$$

$$\frac{\partial P_{r,i}}{\partial Y_j} = \begin{cases} -P_{r,i} W \left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}} \right) + \frac{k_{0,i}}{k_{\infty,i}} \rho \left(\frac{\alpha_{ij}}{W_j} - \frac{\alpha_{iN_{\rm sp}}}{W_{N_{\rm sp}}} \right), & \text{if ``(+M)'', or} \\ -P_{r,i} W \left( \frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}} \right) + \frac{k_{0,i}}{k_{\infty,i}} \frac{\rho}{W_m} \left( \delta_{mj} - \delta_{mN_{\rm sp}} \right), & \text{if ``(+Y}_m\text{)''.} \end{cases} \tag{84}$$

Similarly, for chemically-activated bimolecular reactions

$$\frac{\partial c_i}{\partial T} = c_i \left(\frac{-1}{1+P_{r,i}} \frac{\partial P_{r,i}}{\partial T} + \frac{1}{F_i} \frac{\partial F_i}{\partial T} \right) \tag{85}$$

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

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

$$\frac{\partial F_i}{\partial T} = 0 \text{ and} \tag{87}$$

$$\frac{\partial F_i}{\partial Y_j} = 0 \ . \tag{88}$$

For reactions using the Troe falloff formulation,

$$\frac{\partial F_i}{\partial T} = \frac{\partial F_i}{\partial F_{\rm cent}} \frac{\partial F_{\rm cent}}{\partial T} + \frac{\partial F_i}{\partial P_{r,i}} \frac{\partial P_{r,i}}{\partial T} \text{ and} \tag{89}$$

$$\frac{\partial F_i}{\partial Y_j} = \frac{\partial F_i}{\partial F_{\rm cent}} \cancelto{0}{\frac{\partial F_{\rm cent}}{\partial Y_j}} + \frac{\partial F_i}{\partial P_{r,i}} \frac{\partial P_{r,i}}{\partial Y_j} = \frac{\partial F_i}{\partial P_{r,i}} \frac{\partial P_{r,i}}{\partial Y_j} \ , \tag{90}$$

where

$$\frac{\partial F_i}{\partial F_{\rm cent}} = F_i \left(\frac{1}{F_{\rm cent}\left(1+(A/B)^2\right)} - \ln F_{\rm cent} \frac{2A}{B^3} \frac{\frac{\partial A}{\partial F_{\rm cent}} B - A \frac{\partial B}{\partial F_{\rm cent}}}{\left(1+(A/B)^2\right)^2} \right), \tag{91}$$

$$\frac{\partial F_{\rm cent}}{\partial T} = -\frac{1-a}{T^{***}} \exp\left(\frac{-T}{T^{***}} \right) - \frac{a}{T^*} \exp\left(\frac{-T}{T^*} \right) + \frac{T^{**}}{T^2} \exp\left(\frac{-T^{**}}{T} \right), \tag{92}$$

$$\frac{\partial F_i}{\partial P_{r,i}} = -F_i \ln F_{\text{cent}} \frac{2A}{B^3} \frac{\frac{\partial A}{\partial P_{r,i}} B - A \frac{\partial B}{\partial P_{r,i}}}{\left(1 + (A/B)^2\right)^2}, \tag{93}$$

$\frac{\partial P_{r,i}}{\partial T}$ is given by Eq. (83), $\frac{\partial P_{r,i}}{\partial Y_j}$ is given by Eq. (84), and

$$\frac{\partial A}{\partial F_{\text{cent}}} = \frac{-0.67}{F_{\text{cent}} \ln 10} \qquad \frac{\partial B}{\partial F_{\text{cent}}} = \frac{-1.1762}{F_{\text{cent}} \ln 10} \tag{94}$$

$$\frac{\partial A}{\partial P_{r,i}} = \frac{1}{P_{r,i} \ln 10} \qquad \frac{\partial B}{\partial P_{r,i}} = \frac{-0.14}{P_{r,i} \ln 10}. \tag{95}$$

Finally, for falloff reactions described with the SRI formulation,

$$\frac{\partial F_i}{\partial T} = F_i \left(\frac{e}{T} + X \frac{\frac{ab}{T^2} \exp\left(\frac{-b}{T}\right) - \frac{1}{c} \exp\left(\frac{-T}{c}\right)}{a \cdot \exp\left(\frac{-b}{T}\right) + \exp\left(\frac{-T}{c}\right)} + \frac{\partial X}{\partial P_{r,i}} \frac{\partial P_{r,i}}{\partial T} \ln\left(a \cdot \exp\left(\frac{-b}{T}\right) + \exp\left(\frac{-T}{c}\right) \right) \right) \tag{96}$$

$$\frac{\partial F_i}{\partial Y_j} = F_i \frac{\partial X}{\partial Y_j} \ln\left(a \cdot \exp\left(\frac{-b}{T}\right) + \exp\left(\frac{-T}{c}\right) \right) \tag{97}$$

where

$$\frac{\partial X}{\partial P_{r,i}} = -X^2 \frac{2 \log_{10} P_{r,i}}{P_{r,i} \ln 10}, \tag{98}$$

$$\frac{\partial X}{\partial Y_j} = \frac{\partial X}{\partial P_{r,i}} \frac{\partial P_{r,i}}{\partial Y_j}, \tag{99}$$

$\frac{\partial P_{r,i}}{\partial T}$ is given by Eq. (83), and $\frac{\partial P_{r,i}}{\partial Y_j}$ is given by Eq. (84). 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 alternative pressure-dependence descriptions, logarithmic and Chebyshev, 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. (64) when calculating the partial derivatives of R_i using Eqs. (60), (67), 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

$$\frac{\partial k_{f,i}}{\partial T} = \left[\frac{\partial(\log k_1)}{\partial T} + \left(\frac{\partial(\log k_2)}{\partial T} - \frac{\partial(\log k_1)}{\partial T} \right) \frac{\log p - \log p_1}{\log p_2 - \log p_1} + \left(\log k_2 - \log k_1\right) \frac{\frac{\partial(\log p)}{\partial T}}{\log p_2 - \log p_1} \right] \cdot k_{f,i}, \tag{100}$$

where k_1 and k_2 are given by Eqs. (36) and (37), respectively. Note that terms such as k_1, k_2, p_1, and p_2 are associated with the ith reaction alone; we omitted the subscript for clarity, and continue this was other reaction-specific terms in the following. The necessary components can be determined as

$$\frac{\partial(\log k_1)}{\partial T} = \frac{1}{k_1}\frac{\partial k_1}{\partial T} = \frac{1}{T}\left(\beta_1 + \frac{T_{a,1}}{T}\right) , \tag{101}$$

$$\frac{\partial(\log k_2)}{\partial T} = \frac{1}{T}\left(\beta_2 + \frac{T_{a,2}}{T}\right) , \text{and} \tag{102}$$

$$\frac{\partial(\log p)}{\partial T} = \frac{1}{p}\frac{\partial p}{\partial T} = 0 . \tag{103}$$

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

$$\frac{\partial k_{f,i}}{\partial T} = \frac{k_{f,i}}{T}\left[\beta_1 + \frac{T_{a,1}}{T} + \left(\beta_2 - \beta_1 + \frac{T_{a,2} - T_{a,1}}{T}\right)\frac{\log p - \log p_1}{\log p_2 - \log p_1}\right] . \tag{104}$$

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:

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

where

$$\frac{\partial}{\partial T}\left(T_{i-1}(\tilde{T})\,T_{j-1}\,(\tilde{p})\right) = (i-1)U_{i-2}(\tilde{T})\,T_{j-1}(\tilde{p})\,\frac{\partial \tilde{T}}{\partial T} + (j-1)T_{i-1}(\tilde{T})\,U_{j-2}(\tilde{p})\,\frac{\partial \tilde{p}}{\partial T} , \tag{105}$$

$$\frac{\partial \tilde{T}}{\partial T} = \frac{-2T^{-2}}{T_{\max}^{-1} - T_{\min}^{-1}} , \tag{106}$$

$$\frac{\partial \tilde{p}}{\partial T} = \frac{\frac{2}{p\log(10)}\frac{\partial p}{\partial T}}{\log_{10} p_{\max} - \log_{10} p_{\min}} = 0 , \tag{107}$$

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

$$U_n(x) = \frac{\sin\left((n+1)\arccos x\right)}{\sin(\arccos x)} . \tag{108}$$

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

$$\frac{\partial k_f}{\partial T} = \frac{k_f}{T}\log(10)\sum_{i=1}^{N_T}\sum_{j=1}^{N_p}\alpha_{ij}\left((i-1)U_{i-2}(\tilde{T})\,T_{j-1}(\tilde{p})\frac{-2T^{-1}}{T_{\max}^{-1} - T_{\min}^{-1}}\right) . \tag{109}$$

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. Efficient 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 $k, j = 1, \ldots, N_{\rm sp} - 1$):

$$\begin{aligned}\mathcal{J}_{k+1,1} &= \frac{W_k}{\rho}\left(\frac{\partial \dot{\omega}_k}{\partial T} + \frac{\dot{\omega}_k}{T}\right) \\ &= \frac{W_k}{\rho}\sum_{i=1}^{N_{\rm reac}} \nu_{ki}\left[\frac{\partial c_i}{\partial T}\left(R_{f,i} - R_{r,i}\right) + c_i\left(\frac{\partial R_i}{\partial T} + \frac{R_{f,i} - R_{r,i}}{T}\right)\right], \end{aligned} \tag{110}$$

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

$$\begin{aligned}\mathcal{J}_{k+1,j+1} &= \frac{W_k}{\rho}\left(\frac{\partial \dot{\omega}_k}{\partial Y_j} + \dot{\omega}_k W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right)\right) \\ &= \frac{W_k}{\rho}\Bigg(\dot{\omega}_k W\left(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\right) + \sum_{i=1}^{N_{\rm reac}} \nu_{ki}\Bigg(\frac{\partial c_i}{\partial Y_j}(R_{f,i} - R_{r,i}) \\ &\quad + c_i \sum_{l=1}^{N_{\rm sp}}\Bigg(\nu'_{li}\Bigg(-R_{f,i} W\Big(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\Big) \\ &\quad + \left(\delta_{lj} - \delta_{lN_{\rm sp}}\right) k_{f,i}\frac{\rho}{W_l}[X_l]^{\nu'_{li}-1}\prod_{\substack{n=1\\ n\neq l}}^{N_{\rm sp}}[X_n]^{\nu'_{ni}}\Bigg) \\ &\quad - \nu''_{li}\Bigg(-R_{r,i} W\Big(\frac{1}{W_j} - \frac{1}{W_{N_{\rm sp}}}\Big) \\ &\quad + \left(\delta_{lj} - \delta_{lN_{\rm sp}}\right) k_{r,i}\frac{\rho}{W_l}[X_l]^{\nu''_{li}-1}\prod_{\substack{n=1\\ n\neq l}}^{N_{\rm sp}}[X_n]^{\nu''_{ni}}\Bigg)\Bigg)\Bigg)\Bigg). \end{aligned} \tag{111}$$

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

$$\begin{aligned}\mathcal{J}_{1,1} &= -\sum_{k=1}^{N_{\rm sp}}\left[\left(\frac{1}{c_p}\frac{\partial h_k}{\partial T} - \frac{h_k}{c_p^2}\frac{\partial c_p}{\partial T}\right)\frac{W_k \dot{\omega}_k}{\rho} + \frac{h_k}{c_p}\frac{\partial}{\partial T}\left(\frac{W_k \dot{\omega}_k}{\rho}\right)\right] \\ &= \frac{-1}{c_p}\sum_{k=1}^{N_{\rm sp}}\left[\left(c_{p,k} - \frac{h_k}{c_p}\frac{\partial c_p}{\partial T}\right)\frac{W_k \dot{\omega}_k}{\rho} + h_k \mathcal{J}_{k+1,1}\right] \end{aligned} \tag{112}$$

$$\begin{aligned}\mathcal{J}_{1,j+1} &= -\sum_{k=1}^{N_{\rm sp}}\left[\frac{-h_k}{c_p^2}\frac{\partial c_p}{\partial Y_j}\frac{W_k \dot{\omega}_k}{\rho} + \frac{h_k}{c_p}\frac{\partial}{\partial Y_j}\left(\frac{W_k \dot{\omega}_k}{\rho}\right)\right] \\ &= \frac{-1}{c_p}\left(-\frac{c_{p,j} - c_{p,N_{\rm sp}}}{\rho c_p}\sum_{k=1}^{N_{\rm sp}} h_k W_k \dot{\omega}_k + \sum_{k=1}^{N_{\rm sp}} h_k \mathcal{J}_{k+1,j+1}\right), \end{aligned} \tag{113}$$

where $\frac{\partial c_p}{\partial T}$ is given by Eq. (53).

3.3. Jacobian matrix sparsity

Although others discuss sparsity of chemical kinetics Jacobian matrices (Lu and Law, Perini et al. [?]); however, while many assume that large kinetic models exhibit a sparse Jacobian matrix, the Jacobian is actually dense for the governing equations of constant-pressure kinetics using mass fractions. This results from the partial derivative of density with respect to species mass fraction, $\partial\rho/\partial Y_j$, that appears in Eq. (48). As a result, the partial derivatives of all species k rate-of-change $\dot{Y}_k$ with respect to the jth species' mass fraction of species 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. [?]. In that case, off-diagonal elements of the Jacobian matrix arise due to direct species interactions and falloff/pressure-dependent reactions. However, simulations of low-speed open combustion systems typically rely on the constant-pressure assumption [?], with constant volume applicable to enclosed systems (e.g., internal combustion engine cylinder).

finish and revise discussion

Alternatively, the governing equations could be formulated in terms of species molar concentrations rather than mass fractions [? ?]. This would eliminate the presence of ρ in the cross-species Jacobian matrix elements and thus significantly increase matrix sparsity for constant-pressure combustion.

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 straight-forward implementation, it minimizes 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 [?] (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 occasionally crashes of the compiler itself. This section will lay 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.

4.1. Accelerating evaluation via reuse of temporary products

First, examining Eqs. (110) and (111), we see that large portions of the Jacobian entries are constant for a single reaction. For instance, if we define the temporary variable for reaction i:

$$T_{\partial T,i} = \frac{1}{\rho}\left[\frac{\partial c_i}{\partial T}\left(R_{f,i} - R_{r,i}\right) + c_i\left(\frac{\partial R_i}{\partial T} + \frac{R_{f,i} - R_{r,i}}{T}\right)\right] \tag{114}$$

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

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

Eq. (110) can be rewritten as:

$$\mathcal{J}_{k+1,1} = W_k \sum_{i=1}^{N_{\text{reac}}} \nu_{ki} T_{\partial T,i} \tag{115}$$

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:

$$\mathcal{J}_{k+1,1} \mathrel{+}= \nu_{ki} W_k T_{\partial T,i} \quad k = 1, \ldots, N_{\text{sp}} \tag{116}$$

In doing so, the temporary product $T_{\partial T,i}$ must only be evaluated a single time for each reaction, rather than once for each species in the reaction with a non-zero net production or consumption rate.

Similarly, for Eq. (111) 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:

$$T_{\partial Y,i,\text{ind}} = -\frac{W c_i}{\rho} \left(R_{f,i} \sum_{l=1}^{N_{sp}} \nu'_{li} - R_{r,i} \sum_{l=1}^{N_{sp}} \nu''_{li} \right) \tag{117}$$

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

$$\begin{aligned} \mathcal{J}_{k+1,j+1} \mathrel{+}= \nu_{ki} \frac{W_k}{W_j} \Bigg[& T_{\partial Y,i,\text{ind}} \left(1 - \frac{W_j}{W_{N_{\text{sp}}}} \right) + c_i \left(\left(k_{f,i} S'_j - k_{r,i} S''_j \right) \right. \\ & \left. - \frac{W_j}{W_{N_{\text{sp}}}} \left(k_{f,i} S'_{N_{\text{sp}}} - k_{r,i} S''_{N_{\text{sp}}} \right) \right) \Bigg] \quad k, j = 1, \ldots, N_{\text{sp}} \end{aligned} \tag{118}$$

where S'_l and S''_l are defined as:

$$\begin{aligned} S'_l &= \nu'_{li} [X_l]^{\nu'_{li}-1} \prod_{\substack{n=1 \\ n \neq l}}^{N_{\text{sp}}} [X_n]^{\nu'_{ni}} \\ S''_l &= \nu''_{li} [X_l]^{\nu''_{li}-1} \prod_{\substack{n=1 \\ n \neq l}}^{N_{\text{sp}}} [X_n]^{\nu''_{ni}} \end{aligned} \tag{119}$$

Note that either one or both of ν'_{ji} and ν''_{ji} can be zero, while $\nu'_{N_{\text{sp}}i}$ and $\nu''_{N_{\text{sp}}i}$ are typically both zero if the last species is an inert abundant gas, e.g. N_2 in air-fed combustion. This means that the added S'_l and S''_l terms are often zero, and may be omitted in many cases.

For third-body enhanced or falloff reactions, the $\frac{\partial c_i}{\partial Y_j}$ term in Eq. (111) 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. (80), and the updating scheme defined in Eq. (118) can be used. Otherwise, the appropriate update schemes described below must be employed

For third body-enhanced reactions, we define:

$$\begin{aligned} T_{\partial Y,i,\text{third-body}} &= T_{\partial Y,i,\text{ind}} + \frac{-Wc_i}{\rho} R_i \\ &= -\frac{Wc_i}{\rho}\left(R_{f,i}\left(1+\sum_{l=1}^{N_{sp}}\nu'_{li}\right) - R_{r,i}\left(1+\sum_{l=1}^{N_{sp}}\nu''_{li}\right)\right) \end{aligned} \tag{120}$$

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

$$\begin{aligned} \mathcal{J}_{k+1,j+1} \mathrel{+}= \nu_{ki}\frac{W_k}{W_j}\Bigg[& T_{\partial Y,i,\text{third-body}}\left(1-\frac{W_j}{W_{N_\text{sp}}}\right) + \left(\alpha_{ij} - \alpha_{iN_\text{sp}}\frac{W_j}{W_{N_\text{sp}}}\right) R_i \\ & + c_i\left(\left(k_{f,i}S'_j - k_{r,i}S''_j\right) - \frac{W_j}{W_{N_\text{sp}}}\left(k_{f,i}S'_{N_\text{sp}} - k_{r,i}S''_{N_\text{sp}}\right)\right)\Bigg] \quad k,j = 1,\ldots,N_\text{sp} \end{aligned} \tag{121}$$

For reactions with falloff dependence, we define a temporary term corresponding to the P_r polynomial in Eqs. (82) and (86):

$$T_{P_{r,i}} = \begin{cases} 0 & \text{if Lindemann falloff,} \\ \dfrac{1.0}{1.0+P_r} & \text{if falloff reaction (Troe/SRI), or} \\ \dfrac{-P_r}{1.0+P_r} & \text{if chemically activated reaction (Troe/SRI).} \end{cases} \tag{122}$$

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

$$T_{F_i} = \begin{cases} 0 & \text{if Lindemann,} \\ \ln F_\text{cent}\dfrac{2A}{B^3}\dfrac{0.14A+B}{\ln 10(1+(A/B)^2)^2} & \text{if Troe, or} \\ X^2\dfrac{2\log 10 P_{r,i}}{\ln 10}\ln\left(a\cdot\exp\dfrac{-b}{T}+\exp\dfrac{-T}{c}\right) & \text{if SRI.} \end{cases} \tag{123}$$

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

$$\begin{aligned} T_{\partial Y,i,\text{falloff}} &= T_{\partial Y,i,\text{ind}} + c_i\left(T_{P_{r,i}} + T_{F_i}\right)\left(\frac{-W}{\rho}\right)R_i \\ &= \frac{-Wc_i}{\rho}\left(R_{f,i}\left(\sum_{l}^{N_{sp}}\nu'_{li}\right) - R_{r,i}\left(\sum_{l}^{N_{sp}}\nu''_{li}\right) + R_i\left(T_{P_{r,i}}+T_{F_i}\right)\right) \end{aligned} \tag{124}$$

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

$$\begin{aligned} \mathcal{J}_{k+1,j+1} \mathrel{+}= \nu_{ki}\frac{W_k}{W_j}\Bigg[& T_{\partial Y,i,\text{falloff}}\left(1-\frac{W_j}{W_{N_\text{sp}}}\right) \\ & + \frac{\frac{k_0}{k_\infty}F_i}{1+Pr_i}\left(T_{P_{r,i}}+T_{F_i}\right)\left(\delta_M\left(\alpha_{ij} - \alpha_{iN_\text{sp}}\frac{W_j}{W_{N_\text{sp}}}\right)\right. \end{aligned}$$

$$+\ (1-\delta_{\mathrm{M}})\left(\delta_{mj}-\delta_{mN_{\mathrm{sp}}}\frac{W_j}{W_{N_{\mathrm{sp}}}}\right)\Bigg)R_i$$
$$+\ c_i\left(\left(k_{f,i}S'_j-k_{r,i}S''_j\right)-\frac{W_j}{W_{N_{\mathrm{sp}}}}\left(k_{f,i}S'_{N_{\mathrm{sp}}}-k_{r,i}S''_{N_{\mathrm{sp}}}\right)\right)\Bigg] \quad k,j=1,\ldots,N_{\mathrm{sp}} \tag{125}$$

where δ_{M} is defined as:

$$\delta_M=\begin{cases}1 & \text{if ``(+M)'' or}\\ 0 & \text{if ``(+Y}_m\text{)''.}\end{cases} \tag{126}$$

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

$$\mathcal{J}_{k+1,j+1}\ +\!=\frac{W_k}{W_j}\frac{\dot{\omega}_k W}{\rho}\left(1-\frac{W_j}{W_{N_{\mathrm{sp}}}}\right)\ . \tag{127}$$

As with the update scheme used in Eq. (116), the strategy presented in Eqs. (118), (121), and (125) 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.

5. Results and discussion

The Python [?] package `pyJac` implements the methodology for producing analytical Jacobian matrices described in the previous sections, which we released openly online [?] under a modified BSD license. `pyJac` requires the Python module `NumPy` [?]. The module used to test the correctness of `pyJac` is included in this release, and additionally requires `Cython` [?], Cantera [?], `PyYAML` [?], and `Adept` [?]; these are not required for Jacobian/rate subroutine generation however. In addition, interpreting Cantera-format models [?] requires that module, although interpretation of Chemkin-format models do not require any additional components.

Get DOI for v1 release

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. [?], GRI-Mech 3.0 [?], USC-Mech version II [?], and the isopentanol model of Sarathy et al. [?].

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 B 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/model, with the parameters given in Table 2. Each simulation ran for 10 residence times (τ_{res}) to ensure reaching statistical steady state.

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

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. [?] model since neither Cantera nor `pyJac` support these.

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_p	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_p 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.

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 sizes [?]. For example, when initially attempting to evaluate derivatives using sixth-order central differences with increments calculated similarly to that used by the implicit CVODE integrator in the SUNDIALS suite [?], we encountered large errors in some partial derivatives under certain conditions. Adjusting the relative and absolute tolerances reduced some of these errors, but consistent values could not be found that sufficed over all 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 SUNDIALS; such implicit solvers only require approximations to the Jacobian matrix. However, based on these 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 [? ? ? ?] and chemical explosive mode analysis [? ? ?] that rely on eigendecomposition of the Jacobian matrix.

As a result of the aforementioned difficulties with finite differences, we used the `Adept` software library [? ?] to obtain accurate Jacobian matrices via automatic differentiation using expression templates. In our experience, numerical Jacobians obtained using multiple-term Richardson extrapolants [? ? ?] 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 [? ? ?]. For all test cases, the correctness of the species and reaction rate subroutines were established through comparison with Cantera [?].

In reporting the discrepancy between the analytical and automatically differentiated Jacobian matrices, denoted by $\mathcal{J}$ and $\hat{\mathcal{J}}$ respectively, we used the norm of the relative errors of matrix elements $E_{\text{rel}} = \left\| (\hat{\mathcal{J}}_{ij} - \mathcal{J}_{ij})/\hat{\mathcal{J}}_{ij} \right\|_p$, where $p = 2$ (the Frobenius norm), to quantify error. This differs somewhat from the error metric suggested by Anderson et al. [?] for use quantifying the error of matrices in LAPACK: the relative error p-norm $E_{\text{norm}} = \left\| \hat{\mathcal{J}} - \mathcal{J} \right\|_p / \left\| \hat{\mathcal{J}} \right\|_p$. The latter indicates overall error in the matrix but can be dominated by large elements, so for the current purposes the we believed the former error measure 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., magnitude $\sim 10^{-22}$) likely due to roundoff errors; thus, error statistics comprised only matrix elements $\mathcal{J}_{ij}$ where $|\mathcal{J}_{ij}| \geq \|\mathcal{J}\|_p / 10^{20}$. For all benchmark cases, the analytical Jacobian matrices matched closely with those obtained via automatic differentiation, with the largest discrepancy below 1×10^{-3} %.

We also made comparisons with the `TChem` package [?] for validation purposes. However, while many quantities calculated via `TChem` agreed closely with those from both Cantera and

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.

`pyJac`, we observed discrepancies in the `TChem` species net production rates that correlated with 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.As a result, we do not present detailed Jacobian matrix comparisons between `TChem` and `pyJac` here.

was the error low near equilibrium? I thought it was in these states where it popped up but I could be wrong

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` where possible. The CPU Jacobian subroutines were compiled using `gcc` 4.4.7 [**?**], and run on two six-core Xeon X5650 processors. Although not strictly necessary, we accelerated the performance testing by evaluating Jacobian matrices in parallel via OpenMP [**?**], where each OpenMP thread calculated a single Jacobian matrix. Reported mean evaluation times were computed from the wall-clock run times of ten individual calculations; the results were not normalized by number of OpenMP threads (all cases used twelve threads). The GPU Jacobian subroutines were compiled using `nvcc` 7.5.17 [**?**] 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 [**?**] 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 [**?**]. (`TChem` does not support logarithmic pressure-log or Chebyshev pressure-dependent reaction expressions, so its performance with the isopentanol model could not be evaluated.) Table 4 summarizes the performance ratio between Jacobian matrices calculated using `pyJac` and finite differences/`TChem`; `pyJac`-based routines perform approximately 4–41 $\times$ faster than `TChem`, depending on the size of the model. `pyJac` also performs 3–6 $\times$ 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 0.86, 0.72, 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` is nearly

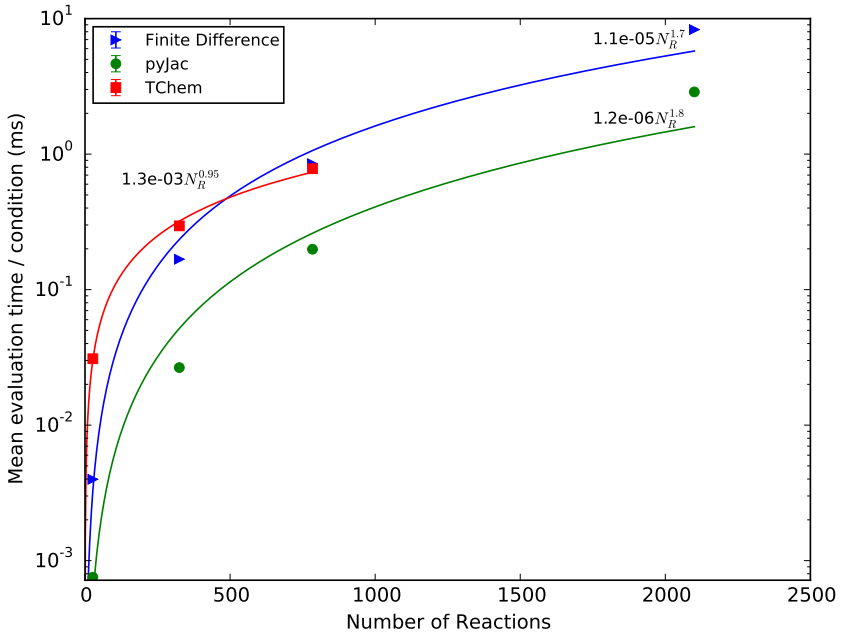


Figure 1: Comparison of Jacobian matrix evaluation times based on finite difference, `pyJac`, and `TChem` for the four kinetic models. The symbols indicate performance data, while the solid lines represent least-squares best fits based on the number of reactions N_R in the models.

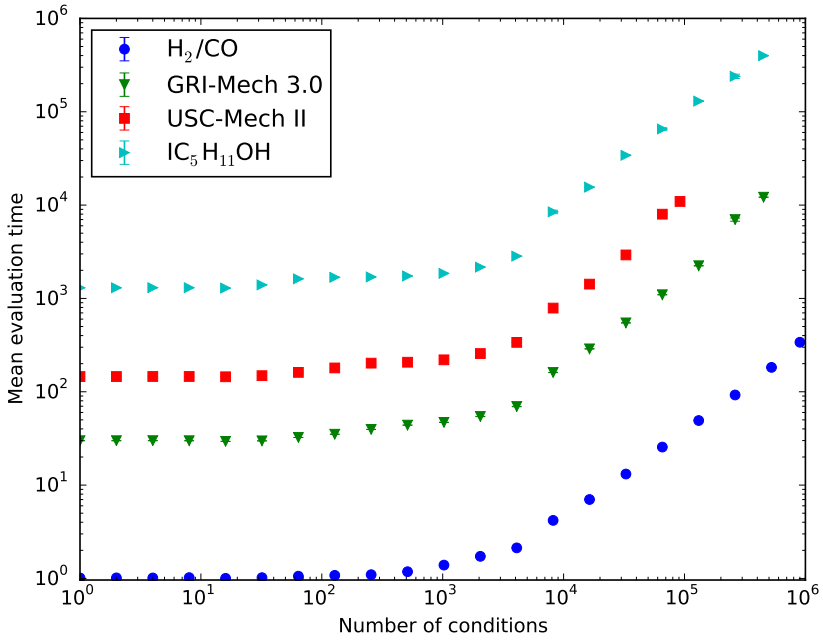
Model	$\bar{R}_{\texttt{TChem}}/\bar{R}_{\texttt{pyJac}}$	$\bar{R}_{\text{FD}}/\bar{R}_{\texttt{pyJac}}$
H_2/CO	40.95	5.28
GRI-Mech 3.0	11.11	6.30
USC-Mech II	3.93	4.28
$iC_5H_{11}OH$	–	2.89

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.

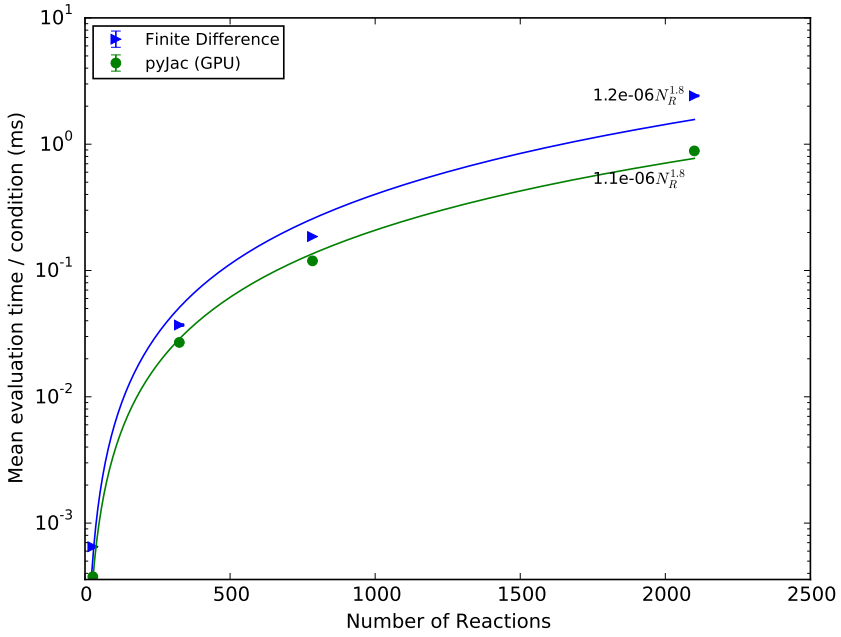
linear in the number of reactions. This is expected for the finite difference approach, but warrants explanation for the analytical Jacobian methods. Lu and Law argued [**?**] that the cost of analytical Jacobian evaluation should scale linearly with the number of reactions in a model. However, this 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. It is unclear whether the nearly linear scaling of `TChem` is an artifact, e.g. due to only having three data-points for fit generation, or the large apparent overhead of `TChem` masking the runtime growth for the smaller mechanisms (`TChem` is outperformed by the finite-difference method for the two smallest mechanisms), or some other cause. The current performance, while faster than either a typical finite difference approach or `TChem`, provides significant incentive for the development of a sparse concentration-based, constant-pressure Jacobian system in future versions of `pyJac`. Using the developed best-fit lines, we can also obtain a rough order-of-magnitude estimate for the crossover points between the three CPU-based approaches: `pyJac` is expected to outperform `TChem` and finite differences for models with up to $\sim$2500 reactions and 8×10^7 reactions, respectively.

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. A "per-thread" GPU parallelization model was adopted in this work, in which a single GPU thread evaluates of 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 remains to be seen if this would translate into an overall increase in 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—are of similar polynomial orders as those of 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. 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



(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.

Model	$\bar{R}_{\text{FD}}/\bar{R}_{\texttt{pyJac}}$
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.

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, for larger models, a relatively larger performance benefit (compared to smaller models) is realized by use of an analytical Jacobian formulation versus a finite difference method. 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).

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 is expected due to the density of the Jacobian matrix, but points to the large 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 model size, suggesting a reduced performance benefit from a hard-coded Jacobian matrix evaluation subroutine. 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 the `gcc` 4.4.7 compiler. This not only causes longer source code compilation times (on the order of ten minutes to an hour when using a 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 analytic 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, strategies for efficient evaluation were detailed, including reordering matrix element evaluations in order to enable the reuse of temporary products. The presented methodology was implemented in the open-source software package `pyJac` [**?**], which generates custom source code files for evaluating chemical kinetics Jacobian matrices on both CPU and GPU systems. The

correctness of the resulting Jacobian matrices was established through comparison with matrices obtained by automatic differentiation. The results agreed within 1×10^{-3} % for kinetic models describing the oxidation of hydrogen, methane, ethylene, and isopentanol, with 13 to 360 species (and 27 to 2172 reactions). Finally, the performance of the CPU and GPU matrix evaluation subroutines was investigated by evaluating the matrix calculation time for the same kinetic models. The `pyJac` CPU-based Jacobian evaluation performed 2.9–5.3 times faster than a forward finite difference approach and 3.9–41 times faster than the existing `TChem` software.

Planned work includes further development of `pyJac` to handle the assumption of constant volume, and support for generating Fortran and Matlab source code. Further study will be performed into the benefits of the current hard-coded, 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 for helpful discussions on appropriate formulation of state vectors.

Appendix A. Proof of partial derivative of pressure

$$
\begin{aligned}
\frac{\partial p}{\partial Y_j} &= \frac{\partial}{\partial Y_j}\left(\mathcal{R}T\sum_{k=1}^{N_{\text{sp}}}[X_k]\right) = \mathcal{R}T\sum_{k=1}^{N_{\text{sp}}}\frac{\partial [X_k]}{\partial Y_j} \\
&= \mathcal{R}T\sum_{k=1}^{N_{\text{sp}}}\left[-[X_k]W\left(\frac{1}{W_j}-\frac{1}{W_{N_{\text{sp}}}}\right)+\left(\delta_{kj}-\delta_{kN_{\text{sp}}}\right)\frac{\rho}{W_k}\right] \\
&= -\mathcal{R}TW\left(\frac{1}{W_j}-\frac{1}{W_{N_{\text{sp}}}}\right)\sum_{k=1}^{N_{\text{sp}}}[X_k]+\mathcal{R}T\rho\sum_{k=1}^{N_{\text{sp}}}\frac{\delta_{kj}-\delta_{kN_{\text{sp}}}}{W_k} \\
&= -pW\left(\frac{1}{W_j}-\frac{1}{W_{N_{\text{sp}}}}\right)+\rho\mathcal{R}T\left(\frac{1}{W_j}-\frac{1}{W_{N_{\text{sp}}}}\right) \\
&= (-pW+pW)\left(\frac{1}{W_j}-\frac{1}{W_{N_{\text{sp}}}}\right) \\
\therefore \frac{\partial p}{\partial Y_j} &= 0\,. \qquad \text{(A.1)}
\end{aligned}
$$

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

Appendix B. Partially stirred reactor implementation

Here, we describe our partially stirred reactor (PaSR) implementation for completeness, based on prior descriptions [? ? ? ? ? ?]. The reactor model consists of an even number N_p of particles, each with a time-varying composition $\phi(t)$. Unlike the composition vector described previously (Eq. (1)), here we use mixture enthalpy and species mass fractions:

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

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 $N_\mathrm{p}\Delta t/\tau_\mathrm{res}$ 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 $0.85 : 0.05 : 0.1$. Then, $\frac{1}{2}N_\mathrm{p}\Delta t/\tau_\mathrm{pair}$ pairs of particles not including the inflowing particles, where τ_pair is the pairing timescale, are randomly selected for pairing and then randomly shuffled with the inflowing particles to exchange partners.

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

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

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

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:

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

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

$$\alpha = \frac{\phi_0^p - \phi_0^q}{2}\left[1 - \exp\left(\frac{-2\delta t}{\tau_\mathrm{mix}}\right)\right] \ , \tag{B.6}$$

where ϕ_0^p and ϕ_0^q are the particle compositions at the beginning of the mixture fractional step. The reaction fractional step consists of the enthalpy evolving by

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

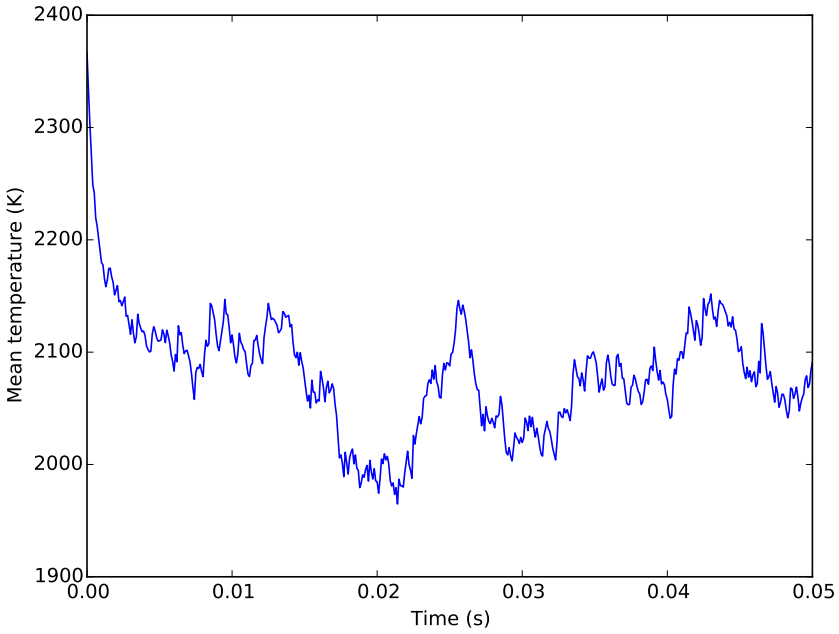


Figure B.1: Mean temperature of premixed PaSR combustion for stoichiometric methane/air with an unburned temperature of 600 K and at 1 atm, $\tau_{\text{res}} = 5\,\text{ms}$, $\tau_{\text{mix}} = \tau_{\text{pair}} = 1\,\text{ms}$, and using 100 particles.

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 [?] `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 and the sub-time step size separating mixing/reaction fractional steps, both held constant in the current implementation, via

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

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

adopted from Pope [?].

Figures B.1 and B.2 demonstrate sample results from premixed PaSR combustion of methane/air, using GRI-Mech 3.0 [?]; Fig. B.1 shows the mean temperature evolution over time, while Fig. B.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.

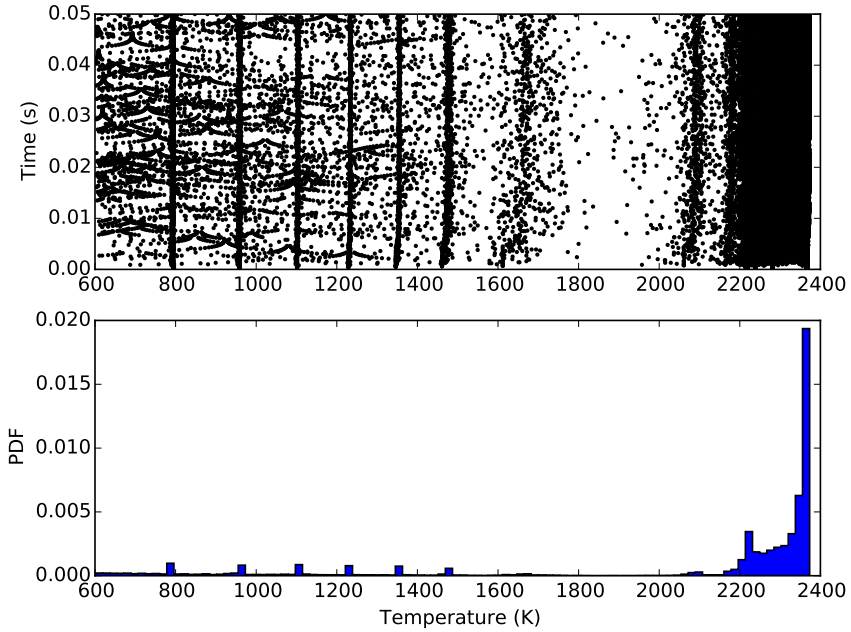


Figure B.2: Scatterplot of temperature over time (top) and PDF of temperature (bottom) of premixed PaSR combustion for stoichiometric methane/air with an unburned temperature of 600 K and at 1 atm, $\tau_{\text{res}} = 5\,\text{ms}$, $\tau_{\text{mix}} = \tau_{\text{pair}} = 1\,\text{ms}$, and using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