Combustion and Flame 174 (2016) 1–15

Contents lists available at ScienceDirect

Combustion and Flame

journal homepage: www.elsevier.com/locate/combustflame

On the merits of extrapolation-based stiff ODE solvers for combustion CFD

A. Imren*, D.C. Haworth

Research Building East, Department of Mechanical and Nuclear Engineering, The Pennsylvania State University, University Park, PA 16802, USA

ARTICLE INFO

Article history:
Received 6 May 2016
Revised 27 June 2016
Accepted 15 September 2016
Available online 28 September 2016

Keywords:
Computational fluid dynamics
Detailed chemistry
Stiff ODE solver
Dynamic adaptive chemistry
Compression-ignition engine

ABSTRACT

In applications including compression-ignition engines, there is a need to accommodate more realistic chemistry in computational fluid dynamics (CFD) simulations. Here, we consider approaches where a chemical mechanism is implemented in an application CFD code, an operator-splitting strategy is used to isolate the chemical source terms, and a stiff ordinary differential equation (ODE) solver is used to compute the changes in composition due to chemical reactions for each computational element. Chemical source terms often dominate the computational effort, and reducing the high computational cost associated with realistic chemistry has been the subject of extensive research. This includes work on improved stiff ODE solvers, which in most cases has centered on backward differentiation formula (BDF) methods. Here a different class of solvers is considered, based on extrapolation methods. Key elements of stiff ODE solvers are reviewed briefly, focusing on differences between BDF methods and extrapolation methods. Issues related to using a stiff ODE solver with operator splitting in a CFD code (where the solver is repeatedly stopped and restarted) are emphasized. Homogeneous-reactor results are presented first. There the relationship between user-specified error tolerances and solution accuracy is explored, tradeoffs between accuracy and CPU time are shown, and close-to-linear increase in CPU time with increasing chemical mechanism size is demonstrated. Engine results are presented next, including both homogeneous-charge compression-ignition engines, and direct-injection (nonhomogeneous) compression-ignition engines. There some results are presented where the stiff ODE solver is combined with a dynamic adaptive chemistry scheme. In all cases, it is found that the extrapolation solver offers significant advantages in accuracy and computational efficiency compared to the BDF solver. While the results presented are for one BDF solver (CVODE) and one extrapolation solver (SEULEX), it is anticipated that the insight into how stiff ODE solvers behave in combustion CFD will be broadly applicable to other solvers in these general classes.

1. Introduction

Across multiple domains of application, there is a pressing need to incorporate more realistic chemical kinetics in computational fluid dynamics (CFD) simulations of chemically reacting flows. For example, increasingly large chemical mechanisms are needed to accurately predict autoignition, heat release and pollutant emissions in multidimensional modeling of in-cylinder processes in compression-ignition engines, especially for the thermochemical environments that are of interest for next-generation engines [1,2]. Here, the focus is on approaches where the chemical mechanism of interest is carried directly in the CFD simulation (versus being pretabulated in flame libraries, for example), an operator-splitting strategy is invoked to isolate the chemical source terms in the governing equations, and the changes in composition due to chemical reactions for each computational element (e.g., each finite-volume cell in the case of Eulerian CFD, or each computational particle in the case of a Lagrangian-particle-based transported probability density function – PDF – method [3]) are computed by solving a system of ordinary differential equations (ODEs), typically using a stiff ODE solver. This does not preclude initial prereduction of a detailed chemical mechanism. Further information on why this approach is of interest is provided in Section 2.

In this scenario, calculation of chemical source terms usually dominates the computational effort, and several strategies have been developed to reduce the high computational overhead of carrying realistic chemistry. These include: on-the-fly mechanism reduction strategies, where the chemical mechanism is reduced locally for each computational element based on local thermochemical conditions (e.g., directed-relation-graph-based dynamic

* Corresponding author.
E-mail address: aui14@psu.edu (A. Imren).

http://dx.doi.org/10.1016/j.combustflame.2016.09.018
0010-2180/

adaptive chemistry – DRG/DAC [4,5]); storage-retrieval (tabulation) strategies, where chemical source terms are stored as they are computed and are reused later in the simulation when similar thermochemical conditions are encountered (e.g., *in situ* adaptive tabulation – ISAT [6]); clustering strategies, where computational elements having similar initial conditions are combined to reduce the number of ODE integrations required (e.g., chemistry coordinate mapping – CCM [7]); hardware-based strategies that take advantage of specific computer architectures (e.g., graphics processing units – GPUs [8]); and combinations of two or more of these strategies (e.g., combining dynamic adaptive chemistry with tabulation [9,10], or dimension reduction with tabulation [11,12]).

At some point, all of these on-the-fly chemistry acceleration strategies (as well as most prereduction strategies) require a stiff ODE solver, and improved stiff ODE solvers, tailored to the combustion chemistry problem, have been the subject of recent investigations [13–15]. Much of the work to date on stiff ODE solvers for combustion chemistry has focused on solver performance for isolated homogeneous reactors, and has not directly considered the implications on solver performance of repeated solver stopping/reinitialization in a CFD code. Also, most work has focused on backward differentiation formula (BDF) methods for solving systems of stiff ODEs. Here, a fresh look is taken at stiff ODE solvers for combustion CFD by explicitly considering the reinitialization problem, and this leads to consideration of a different class of stiff ODE solvers based on extrapolation methods [16,17].

The remainder of the paper is organized as follows. In Section 2, the motivation for operator splitting and stiff ODE solvers in combustion CFD is presented, key elements of stiff ODE solvers are discussed, the reinitialization problem is introduced, a basic DAC method is reviewed (for subsequent application to engine combustion CFD, in combination with the stiff ODE solvers), and other elements of the CFD algorithms that have been used here are described. The chemical mechanisms that have been considered are introduced in Section 3, and results are presented in Section 4. Three general configurations are considered: homogeneous reactors, homogeneous-charge compression-ignition (HCCI) engines and direct-injection compression-ignition engines. The emphasis is on comparisons of computational time required versus solution accuracy for different stiff ODE solver variants. Conclusions and next steps are summarized in the final section.

2. Operator splitting and stiff ODE solvers for combustion CFD

For the class of combustion CFD problems that is of interest here, key elements of the computational strategy are: (1) invoke operator splitting to isolate the chemical source terms in the governing equations; (2) integrate a system of highly nonlinear ODEs to compute the changes in composition due to chemical reactions over a specified computational time step using a stiff ODE solver; and (3) implement other chemistry acceleration strategies (in addition to improved stiff ODE solvers) to further reduce the computational time required for chemistry calculations – here DRG–based DAC is considered, as an example. Each of these is discussed in the following subsections. The importance of solver reinitialization is emphasized. In the final subsection, the specific CFD algorithms that have been used are described briefly.

2.1. Operator splitting

We consider a reacting mixture of N_S chemical species. The mixture composition can be expressed in terms of the species mass fractions (Y_α for species α), and the equations governing the evolution of Y_α can be written as Eulerian partial differential equations (in the case of finite-volume-based CFD, for example) or as Lagrangian ordinary differential equations (in a particle-based transported PDF method [3], for example). In either case, one can invoke operator splitting to isolate the changes in composition due to chemical reactions. Then for each computational element (Eulerian finite-volume cell, or Lagrangian particle), the change in composition due to chemical reactions over a computational time step Δt_{CFD} can be computed as the solution to a nonlinear initial-value problem:

$$\frac{d\mathbf{Y}}{dt} = \mathbf{S}\big(\mathbf{Y}(t)\big) \quad \text{with } \mathbf{Y}(t=t_0) = \mathbf{Y}_0 \ ,$$

$$\text{or} \quad \mathbf{Y}(t_0 + \Delta t_{\mathrm{CFD}}) = \mathbf{Y}_0 + \int_{t_0}^{t_0+\Delta t_{\mathrm{CFD}}} \mathbf{S}\big(\mathbf{Y}(t)\big)dt \ , \tag{1}$$

where $\mathbf{Y}$ and $\mathbf{S}$ are vectors of length $N_\phi = N_S + 1$ (N_S species mass fractions plus temperature), and $\mathbf{S}$ is the mass-based rate of production of species α or of temperature. The method that is chosen to solve Eq. 1 depends on the physical time scales in the problem, compared to the computational time step Δt_{CFD}. For typical hydrocarbon-air chemical mechanisms at engine-relevant conditions, the chemistry is *stiff*: that is, the range of relevant time scales (which can be quantified formally using eigenvalue analysis, or estimated using various approximations) varies over several orders of magnitude. The longest chemical time scale of interest is the time to reach chemical equilibrium from the prescribed initial condition: e.g., $\tau_{\mathrm{chem,max}} = \tau_{\mathrm{equil}} \approx 10^{-4} - 10^{-3}$ s. The value of $\tau_{\mathrm{chem,\ max}}$ can be even larger than this, depending on the chemical mechanism, thermochemical conditions, and the physics of interest (the time required for NO to reach equilibrium can be much longer, for example). The shortest chemical time scales are typically several orders of magnitude smaller: e.g., $\tau_{\mathrm{chem,min}} \approx 10^{-9} - 10^{-8}$ s. The value of $\tau_{\mathrm{chem,\ min}}$ can be even smaller than this, depending on the chemical mechanism and thermochemical conditions. The computational time step that is used in CFD is usually determined based on considerations other than chemistry (e.g., capturing transport or fuel-spray processes, or controlling splitting errors), and a typical value for compression-ignition engine applications is on the order of microseconds: $\Delta t_{\mathrm{CFD}} \approx 10^{-6} - 10^{-5}$ s, say. This is still orders of magnitude larger than the smallest relevant chemical time scale, and in that case, an implicit stiff ODE solver is appropriate to solve the system of ODEs given in Eq. 1.

2.2. Stiff ODE solvers

The nonlinear initial-value problem of Eq. 1 can be solved using variants of Newton iteration, where a succession of linear problems is solved whose solution eventually converges to the solution of the nonlinear problem. This involves subdividing Δt_{CFD} into smaller subintervals Δt_{ODE}. For an explicit solver, or for an implicit solver that reverts to explicit time stepping at some point, the smallest value of Δt_{ODE} can be expected to be of the order of $\tau_{\mathrm{chem,\ min}}$. To advance the solution over a subinterval of the CFD time step from time t_n (where $\mathbf{Y}_n = \mathbf{Y}(t=t_n)$ is known) to time t_{n+1} (where $\mathbf{Y}_{n+1} = \mathbf{Y}(t=t_{n+1})$ is to be found) and with $\Delta t_{\mathrm{ODE}} \equiv t_{n+1} - t_n$, the problem can be linearized as follows:

$$\begin{aligned}\mathbf{Y}_{n+1} &= \mathbf{Y}_n + \int_{t_n}^{t_{n+1}} \Big(\mathbf{S}_n + \mathbf{J}_n\big(\mathbf{Y}_{n+1} - \mathbf{Y}_n\big) + \mathcal{O}(\Delta t_{\mathrm{ODE}}^2)\Big)dt \\ &= \mathbf{Y}_n + \mathbf{S}_n \Delta t_{\mathrm{ODE}} + \mathbf{J}_n\big(\mathbf{Y}_{n+1} - \mathbf{Y}_n\big)\Delta t_{\mathrm{ODE}} + \mathcal{O}(\Delta t_{\mathrm{ODE}}^2) \ ,\end{aligned}$$

so that on rearranging and neglecting higher-order terms ($\mathcal{O}(\Delta t_{\mathrm{ODE}}^2)$):

$$\big(\mathbf{I} - \mathbf{J}_n \Delta t_{\mathrm{ODE}}\big)\big(\mathbf{Y}_{n+1} - \mathbf{Y}_n\big) = \mathbf{S}_n \Delta t_{\mathrm{ODE}} \ . \tag{2}$$

Here, $\mathbf{I}$ is the identity matrix and $\mathbf{J}_n \equiv \frac{d\mathbf{S}}{d\mathbf{Y}}|_n$ is the Jacobian matrix evaluated at time level n. The final equation provides the basis of an iterative procedure that can be used to solve for $\mathbf{Y}_{n+1}$. On each iteration, $\mathbf{S}$ is recomputed based on the most recent $\mathbf{Y}$, $\mathbf{J}$ is updated

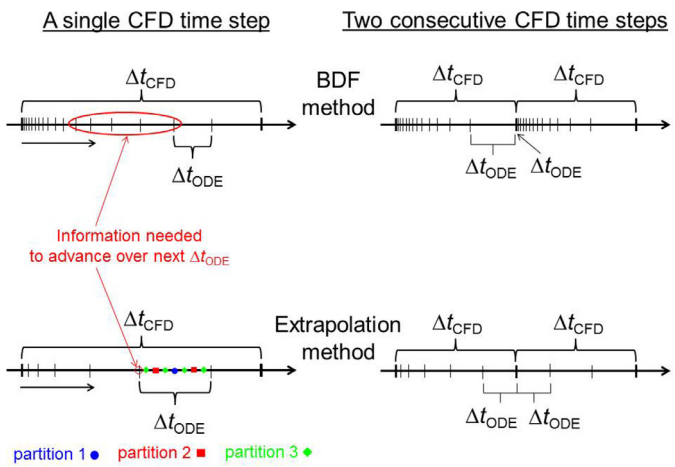

Fig. 1. A schematic illustration of a BDF method (upper) versus an extrapolation method (lower) over a single CFD time step (left) and over two consecutive CFD time steps (right). In a BDF method, information from multiple earlier internal solver time steps (hence "backward") is used to advance the solution over the next internal time step. In an extrapolation method, recursive partitioning and Richardson extrapolation are used to advance the solution over the next internal time step.

(although not necessarily on every iteration), and eventually the solution converges to the solution of the nonlinear problem (for appropriate choices of Δt_{ODE} and how often **J** is updated).

From this basic development, key elements of stiff ODE solvers can be identified. These are the time-integration strategy (specification of the internal time step Δt_{ODE}, and a strategy to go to higher order accuracy), a strategy for how often to update **J** and how to compute **J** (approximations that are made in computing **J** do not affect the final converged solution of the nonlinear problem, only the stability of the method and the rate of convergence), and the method that is used to solve the linear system. Several options are available for each.

2.2.1. Time-integration options

In this paper, two different time-integration strategies are considered: a BDF method and an extrapolation method (Fig. 1). BDF methods have been used for computational chemistry problems since the 1970's [18], including in most recent publications on improved stiff ODE solvers for combustion problems [14,15,19]. An example of a BDF method is CVODE [20,21]. In a BDF method, a first-order approximation with very small Δt_{ODE} is used at the beginning of each Δt_{CFD}. As the integration proceeds, a weighted average of information from earlier internal time steps is used to increase the order of the time-integration scheme and to grow Δt_{ODE}, based on a combination of theory and heuristics. Eventually, Δt_{ODE} and the integration order can increase significantly compared to their initial values (up to 5th order, in the case of CVODE), and if the integration is allowed to continue until local equilibrium is reached, BDF methods are extremely efficient. Details of the BDF method that has been used here (CVODE) can be found in [21], and key elements of the algorithm are summarized in Appendix A. Two especially important characteristics of BDF methods for the present discussion are: (1) the computational effort for each Δt_{CFD} tends to be concentrated at the beginning of the CFD time interval; and (2) to advance from solver internal time level n to time level $n+1$, information from multiple earlier internal time levels is required and must be saved (for greater than first-order accuracy).

Extrapolation methods have been available for as long as BDF methods [22], and have been implemented in some combustion CFD codes. However, no specific analysis/development or detailed comparisons to other methods for combustion problems have been published, to the best of the authors' knowledge. An example of an extrapolation method is SEULEX [17]. In an extrapolation method, each Δt_{ODE} is successively partitioned into subintervals, a low-order method is used over each subinterval, and a recursive procedure (a divided difference table and Richardson extrapolation) is used to project to higher-order accuracy over each Δt_{ODE}; up to 12th-order is available in the implementation that has been used here. The sequence of partitions is prescribed based on theory, and partitioning continues until the user-specified level of accuracy is reached for the current Δt_{ODE}. As the time integration proceeds, a combination of theory and heuristics is used to estimate the next allowable Δt_{ODE}, as in a BDF method. Key elements of the SEULEX algorithm that has been used here are summarized in Appendix B. Since the use of an extrapolation-based stiff ODE solver for combustion CFD is central in this paper, somewhat more detail is provided in Appendix B compared to Appendix A, and C++ source code is provided in the Supplementary Material. Two especially important characteristics of extrapolation methods (in contrast to BDF methods) are: (1) the computational effort for each Δt_{CFD} is not concentrated at the beginning of the CFD time interval; and (2) to advance from internal time level n to time level $n+1$, only information from time level n is required.

In both methods, it is necessary to implement strategies to recognize and correct convergence failures, and again, these are based on a combination of theory and heuristics.

2.2.2. Jacobian options

Three basic approaches can be distinguished to compute the elements of the $N_\phi \times N_\phi$ matrix **J**: the difference quotient (finite difference) method, semianalytic methods and fully analytic methods.

In the difference quotient method, each column of **J** is computed by perturbing one element of **Y** by a small amount ϵ and calculating the partial derivatives numerically, usually using first-order forward differences. This requires N_ϕ evaluations of **S**, is computationally expensive, and has been largely abandoned in recent work.

Analytic methods take advantage of the known functional form of the chemical source terms $\mathbf{S} = \mathbf{S}(\mathbf{Y})$ to calculate some (in the case of semianalytic) or all (in the case of fully analytic) of the partial derivatives analytically. This is relatively straightforward for elementary reaction mechanisms, although exact expressions for the partial derivatives often are used only for simple reactions, while approximations are used for pressure-dependent reactions and reactions with third-body collision efficiencies, for example.

As noted earlier, approximations that are made in estimating the Jacobian do not influence the converged solution to the nonlinear problem (as long as it converges), and this can be exploited in various ways. Those include deferring Jacobian updates as long as possible over the course of the integration, invoking approximations in evaluating Jacobian elements analytically as noted above, "filtering" the Jacobian (setting values that are small compared to a user-prescribed threshold to zero to increase sparseness [15]), and zeroing full rows and/or columns. While filtering has been shown to be effective for large mechanisms, it requires judicious specification of tolerances whose optimum values can vary from one mechanism to another.

Finally, it is noted that since only up to three-body reactions are considered in most chemical mechanisms, the Jacobian matrix is inherently sparse (most elements are zeros, when the problem is formulated as described in Appendix C), and the degree of sparseness increases with increasing mechanism size. This can be exploited in the linear system solver, which is discussed next.

2.2.3. Linear system options

Each iteration requires solving a linear system, and linear algebra can dominate the computational effort if inefficient methods are used. Examples of CPU profiling results will be presented later.

Three basic choices to be made are: whether to use a direct solver or an iterative solver; whether to use a sparse or dense matrix representation; and what underlying math libraries to use.

Direct solvers (e.g., LU decomposition) provide (in principle) an exact solution to the linear system, with the exception of the errors that arise from the use of finite-precision floating-point arithmetic ("roundoff errors"), which can accumulate for large systems. The computational effort scales as n^3 for large n, where n is the size of the linear system (here $n = N_\phi$). Iterative solvers (e.g., Krylov methods) give an approximate solution to the linear system to within a user-prescribed tolerance. Failure to converge the solution of the linear system to sufficient accuracy can lead to global convergence to an incorrect solution of the nonlinear problem. A preconditioner is usually used with an iterative method, and incomplete LU decomposition is a common choice. The computational effort for an iterative solver can scale as n to n^2 for large n. Iterative solvers are expected to be more efficient than direct solvers for large systems (large n), but a general statement cannot be made regarding the relative performance of direct versus iterative solvers for a particular value of n in a specific application.

The question here is whether or not, even for a 10,000-species mechanism (say), the advantage of an iterative solver (including the overhead of preconditioning) is significant or not. The effectiveness of iterative solvers relies strongly on effective sparse linear algebra libraries. Filtering the Jacobian increases sparseness for the preconditioner, which improves performance, but overfiltering can increase the number of Newton iterations required and the overall computational effort; care must be taken in specifying the value of the filtering tolerance, and the appropriate value can vary from one mechanism to another. In [15], iterative methods were found to be advantageous for large mechanisms (larger than $\sim$ 1000 species), when used in conjunction with a sparse matrix representation and Jacobian filtering to improve sparseness. On the other hand, similar performance was reported in [14] using a direct solver, and that might be related to other optimizations that were implemented there.

The inherent sparseness of the combustion chemistry problem suggests that a sparse matrix representation should be advantageous, although there is an overhead associated with reorganizing and indexing of arrays. We have found the sparse representation to be faster for all mechanisms and conditions tested, and a sparse representation will be used for all results presented here, unless specified otherwise.

Finally, recent studies have shown the advantages of using highly optimized math libraries (e.g., [15]). Here the latest version of the KLU sparse matrix math libraries that are provided in the SuiteSparse package has been used [23]; the most recent CVODE release provides interface support for KLU [21]. We have found KLU to be faster across the board than any other library tested (including SuperLU, which was used in [15]), and the KLU sparse matrix libraries are used exclusively in all results presented here.

2.2.4. Best results to date from the literature, and solver combinations adopted here

Three keys to reducing the computational effort required to solve Eq. 1 using either BDF or extrapolation methods are: (1) maximize the internal time step Δt_{ODE} and order of accuracy of the scheme; (2) minimize the number of Jacobian updates; and (3) use an efficient linear system solver. The first two of these are subject to stability constraints.

The best available methods reported to date in the literature give close to linear scaling (linear increase in CPU time with increasing mechanism size). In [14] and [15], approximately linear scaling was reported for homogeneous reactors with chemical mechanisms ranging in size over two-to-three orders of magnitude (from 10–30 species to over 7000 species) using BDF methods. In [14], VODE was used with fully analytic Jacobians and a direct sparse solver, while in [15] CVODE was used with semianalytic filtered Jacobians, a Krylov iterative solver, and SuperLU math libraries [24].

Here the main focus is to compare and contrast the behaviors of a BDF method (CVODE) and an extrapolation method (SEULEX) for combustion CFD, taking a state-of-the-art BDF implementation as the basis for comparison. It appears that the best overall performance reported to date using CVODE has been obtained using a Krylov iterative linear system solver with optimized sparse linear algebra libraries and filtered semianalytic Jacobians, and an efficient interface to implement this combination is provided in the CVODE version used here [21]. Our baseline BDF solver is CVODE with an iterative linear system solver, KLU sparse linear algebra libraries, and semianalytic Jacobians without filtering. We have confirmed that this combination reproduces the results presented in [15], with the exception of slightly higher CPU times for the largest mechanisms because of the absence of Jacobian filtering. We did not implement Jacobian filtering, as that requires specification of additional parameters whose optimum values may vary from one mechanism to another, and poor choices can result in higher CPU time, as discussed earlier.

Our baseline extrapolation solver is SEULEX with a direct linear system solver, KLU sparse linear algebra libraries, and semianalytic Jacobians without filtering. A direct solver was chosen based on implementation expediency. This may give an advantage to the BDF method for the larger mechanism sizes and to the extrapolation method for the smaller mechanism sizes (where differences in CPU time are small, in any case), but as will be shown in Section 4, other factors dominate the differences in performance between the two classes of stiff ODE solvers.

2.3. The reinitialization problem

As discussed earlier, the computational time step Δt_{CFD} is often set by considerations other than chemistry, and may be much smaller than the time required to reach local equlibrium based on the beginning-of-time-step conditions for the computational element. The composition and temperature change because of other physical processes (e.g., transport and diffusion), so that in general, the initial conditions for the chemistry calculation for each computational element at the beginning of each CFD time step are not the same as the conditions at the end of the chemistry calculation from the previous CFD time step. In the simplest implementation of either solver, no information is carried from one CFD time step to the next. The stiff ODE solver must be restarted for each computational element at the beginning of each CFD time step, and the extent to which this impacts solver performance is different for a BDF method versus an extrapolation method (Fig. 1).

The BDF computational effort tends to be concentrated at the beginning of each CFD time step, as a history of several internal time steps of information is needed to grow Δt_{ODE} and to enable higher-order accuracy. Even if the composition of the computational element did not change appreciably since the end of the previous chemistry calculation, continuing directly from the end of the previous calculation would require storing the Jacobian, several earlier internal time steps of information, and the estimates for the next internal time step.

In contrast, relatively little computational effort is lost for an extrapolation method at the start of a new CFD time step, and information from earlier internal time steps is not needed to restart the chemistry calculation. Moreover, by saving a single scalar of information for each computational element from the previous CFD time step (the internal estimate for the next Δt_{ODE}), the extrapolation solver can continue where it left off with little overhead other than recomputing the Jacobian. Storing the solver's internal

estimate of Δt_{ODE} for each computational element can be a useful strategy in general, and this feature has been implemented in OpenFOAM [25] for some time, for example.

In general, the extent to which information saved from one chemistry step can be reused effectively on the next chemistry step depends on how much the thermochemical conditions of the computational element changed from the end of the previous chemistry step as a result of physical processes other than chemistry. Even if the changes in composition and temperature due to physical processes other than chemistry are negligible, a BDF solver (as usually implemented) would restart from scratch with a very small time step and first-order accuracy, and considerable computational effort would be required to build back up to higher order with the largest possible internal time step. On the other hand, if the changes in composition and temperature due to physical processes other than chemistry are large, neither solver would be expected to effectively reuse information stored from the end of the previous chemistry step. Each solver deals with this using its internal algorithms (see Appendices A and B). A BDF method would normally revert to a very small time step and first-order accuracy, and considerable computational effort would required to build back up to higher order with the largest possible internal time step. The startup overhead is lower for the extrapolation method, and the maximum order of accuracy is available immediately on the first internal time step. Even without storing and reusing information from the previous chemistry step, the extrapolation method has an advantage when the solver is repeatedly stopped and restarted.

Discontinuities in composition and temperature resulting from physical processes other than chemistry can artificially reactivate fast chemical modes that had been exhausted by the end of the previous chemistry step, such that at the beginning of the next chemistry step, small internal solver time steps are needed to relax back to the slow trajectory. This reactivation of fast chemical modes is a numerical artifact from operator splitting, and in fact, there may be no need to accurately resolve the decay of these artificial fast modes. In any case, if the solver determines (based on the user-specified error tolerances and the solver's internal algorithms) that it needs to use a smaller internal time step, it will do so. The results provided in Section 4 show that across a wide range of engine-relevant conditions, there are net gains in both accuracy and computational efficiency for the extrapolation solver compared to the BDF solver, and there are benefits from saving and reusing the solver's internal time step estimate.

2.4. Directed-relation-graph-based dynamic adaptive chemistry

It is of interest to explore the interaction between the stiff ODE solver and other chemistry acceleration strategies. In general, it is expected that the more efficient the stiff ODE solver, the smaller the benefits of other strategies would be. Even so, non-negligible benefits are found with DAC, as will be shown below.

Here a method that has been applied in earlier modeling studies of compression-ignition engines (DRG-based DAC [26,27]) has been implemented, as an example. Since its introduction [4], several variants of DRG methods have been proposed (see [28] and references therein). Here DRGEP (DRG with error propogation [5]) has been used, following [27]. For each computational element and on every CFD time step, DRGEP is used to reduce the chemical mechanism and the stiff ODE solver is used on the reduced mechanism. Key user-specified parameters are a set of target species and an error tolerance. The baseline error tolerance is set to be 10^{-4}, and the target species are selected dynamically based on the degree of fuel decomposition (quantified by ϕ_l), a progress equivalence ratio (quantified by ϕ_p) and temperature. If ϕ_l and ϕ_p are greater than the tolerance value, the target species are CO, HO_2 and fuel. If a large fuel molecule has decomposed into smaller molecules (containing fewer than three carbon atoms) and ϕ_l is smaller than the tolerance value, fuel is removed from the target species list. When both ϕ_l and ϕ_p are smaller than the tolerance, the main fuel conversion is considered to be essentially complete, and the target species list is updated by replacing CO with CO_2 and HO_2 with H_2O. And if the reactor temperature is greater than a critical temperature (here 1800 K), NO is added to the target list. This gives computed heat-release profiles and emissions values that differ by less than 0.1% from those computed using the same chemical mechanism without DAC for the engine cases that are considered in Section 4. In the remainder of this paper, "DAC" will imply this specific implementation, unless stated otherwise.

2.5. CFD solver

The solver is based on OpenFOAM 2.3.x libraries [25]. A standard pressure-based finite-volume method is used with second-order spatial accuracy and first-order time accuracy. For engine cases, an unsteady RANS formulation is used with a standard two-equation turbulence model and wall functions. In cases with direct in-cylinder liquid fuel injection, standard stochastic Lagrangian parcel spray models are used. No turbulence-chemistry interaction model has been used: chemical source terms are computed using finite-volume cell-level compositions and temperatures with the given chemical mechanism (a locally well-stirred reactor – WSR – model). The limitations of this approach have been demonstrated for engine simulations (e.g., [29]); here the focus is not on simulation accuracy with respect to experimental measurements, but rather on performance comparisons for different stiff ODE solvers and (in some cases) DAC. The general conclusions that are drawn here in the absence of a turbulence-chemistry interaction model are expected to carry over to cases where such a model is used, as long as a detailed chemistry/stiff ODE solution is still needed for each computational element (as in a particle-based transported composition PDF method [3], for example).

The specific equations that are solved to compute the changes in composition due to chemical reactions for each computational element, and how those are coupled with the CFD code, are provided in Appendix C. Here a simple first-order operator-splitting strategy has been used. Splitting errors are important in combustion CFD, but they are not the focus of this paper. A recent discussion can be found in [19], for example. There it was shown that when radical species are present and radical transport is strong, splitting errors could result in large errors in computing ignition unless very small time steps (comparable to those required for an explicit solver) were used.

3. Thermochemical properties and chemical mechanisms

Reacting ideal-gas mixtures are considered, with all thermodynamic property and chemical mechanism files in standard CHEMKIN format [30]. Here, ODE solution approaches have been tested for 12 different chemical mechanisms that range in size from 34 to 7171 species (Table 1). These are mechanisms for real or surrogate fuels that are of interest for piston-engine applications.

For each chemical mechanism, the thermodynamic data file provided with the mechanism has been used. In several cases, discontinuities were found at the midpoint temperature between the low- and high-temperature-range property polynomial coefficients. These were corrected using a constrained linear least squares fitting procedure that enforces C1 continuity (essentially the CHEMKIN FITDAT algorithm [42]). Any property discontinuities larger than 10^{-4} were repaired prior to using the mechanisms, and the new fits were checked to ensure that no new problems were introduced (e.g., property oscillations). This has been found

Table 1
Chemical mechanisms tested. A primary reference fuel (PRF) is a two-component blend of n-heptane and iso-octane; PRF29 is 71% n-heptane and 29% iso-octane. A TRF is a three-component blend of n-heptane, iso-octane and toluene.

Fuel(s) designed for	Fuel simulated in homogeneous reactor	# of species	# of reactions	Ref.
n-Heptane	n-Heptane	34	74	[31]
n-Heptane	n-Heptane	42	168	[32]
n-Heptane	n-Heptane	48	121	[33]
TRF	PRF29	134	731	[34]
PRF	PRF29	171	861	[35]
Gasoline	PRF29	335	1610	[36]
n-Heptane	n-Heptane	654	2827	[37]
Gasoline surrogate	PRF29	679	3479	[38]
Gasoline surrogate	PRF29	1389	5935	[38]
Diesel surrogate	n-Dodecane/m-xylene	2885	11754	[39]
Biodiesel surrogate	Methyldecanoate (C10)	3299	10806	[40]
Methylalkanes to C20	2-Methylnonadecane (C20)	7171	31669	[41]

to be an important step for stiff ODE solvers, to minimize convergence failures and wasted computational effort. It can be especially important for autoignition under engine-relevant conditions, since midpoint temperature values are often around 1000 K, in a temperature range that is of significant interest for hydrocarbon–air autoignition chemistry. In principle, continuity to within machine precision could be enforced, but no convergence problems were encountered with the 10^{-4} tolerance.

4. Results and discussion

In this section, results for homogeneous reactors, HCCI engines and direct-injection compression-ignition engines are presented and discussed. The main focus is on comparisons of computational times and solution accuracy for different stiff ODE solver variants. The relationship between solution accuracy and solver tolerance specification is discussed, and some DAC results are included to illustrate tradeoffs between improved ODE solvers and other chemistry acceleration strategies. All computations were performed using recent Intel-based hardware (Intel Xeon E5-2600 series, 2.5 GHz, DDR3 memory). Homogeneous reactor and HCCI engine simulations were run using a single core, and direct-injection engine simulations were run using eight cores.

4.1. Homogenous reactor with reinitialization

As discussed earlier, reinitialization is a key issue when a stiff ODE solver is used as part of an operator-splitting strategy in combustion CFD. For this reason, we consider computations of a homogeneous reactor with imposed solver reinitialization at prescribed time intervals $\Delta t_{CFD} < \tau_{equil}$. This is more representative of solver performance in combustion CFD, compared to integrating directly from the initial condition to τ_{equil} without solver reinitialization. Specifically, we consider a constant-pressure adiabatic reactor with a stoichiometric fuel-air mixture initially at 50 atm and 800 K. Different fuels were used for different mechanisms, as indicated in Table 1. It has been confirmed that the results are representative of solver performance over a range of engine-relevant conditions. The total integration time is 10^{-3} s (sufficient to reach equilibrium, in all cases), with reinitialization either once every 10^{-6} s or once every 10^{-5} s; these are time intervals that are representative of the values of Δt_{CFD} that are used in engine combustion CFD. For the extrapolation solver, some results with "dynamic time stepping" (using the solver's internal estimate of Δt_{ODE} to begin the next time step) are shown, to point out an advantage of extrapolation solvers.

4.1.1. Error tolerances, solution accuracy and computational efficiency

The relationships among user-specified error tolerances, solution accuracy and CPU time are explored first (Fig. 2). Global solution accuracy is the difference between the numerical solution of the nonlinear ODE system and the exact solution (e.g., for species compositions and temperature), at any instant in time. In practice, the exact solution is not known, and must be estimated using a reference numerical solution, which can be taken as the best available numerical solution, for example. Most stiff ODE solvers (including the ones that have been used here) require the specification of two convergence criteria to determine when the nonlinear problem has been solved to sufficient accuracy on each internal time step Δt_{ODE}: an absolute tolerance (denoted as ATOL) and a relative tolerance (denoted as RTOL). In the case of an iterative linear system solver, a separate tolerance may be specified for convergence of the linear system. The single most important figure-of-merit in comparing different solvers is the CPU time required to realize a specified level of global solution accuracy. However, because of differences in the order of accuracy of the numerical schemes used, details of how the tolerances are used internally in each solver, and differences in accumulation of roundoff errors, it cannot be assumed that all solvers will give solutions of the same global accuracy for the same input values of ATOL and RTOL.

An example for a 42-species n-heptane mechanism with reinitialization once every 10^{-6} s (1000 CFD time steps to integrate to 10^{-3} s) is shown in Fig. 2. In Fig. 2a, normalized error in the computed temperature at one instant in time ($t = 0.51$ ms, corresponding to an instant during the steep temperature rise of the main ignition event – a particularly stringent test of accuracy) is plotted as a function of RTOL ($10^{-14} \le$ RTOL $\le 10^{-4}$) for two different values of ATOL (10^{-14} and 10^{-10}). Results for a BDF solver and an extrapolation solver are plotted, both using a direct linear system solver in this case, to remove iteration errors in solving the linear system from consideration. This does not adversely affect the CPU time results for either solver, as the differences in CPU time for iterative versus direct linear system solvers are small for this mechanism size. The error is defined as $|(T - T_{ref})/T_{ref}|$ at that instant, where the reference value T_{ref} for each solver is from a numerical solution obtained using that solver with ATOL = RTOL = 10^{-15}. The reference solutions are therefore slightly different for the two solvers. The tradeoff between CPU time and global solution accuracy for the same case is shown in Fig. 2b; there the CPU time is the time for a full run to 1 ms of physical time, and the error is defined in the same way as in Fig. 2a.

Several observations can be made. First (Fig. 2a), for the same values of ATOL and RTOL, the extrapolation solver gives two-to-three more significant figures of solution accuracy than the BDF solver. Second (Fig. 2b), the extrapolation solver gives a better CPU time versus accuracy tradeoff compared to the BDF solver. And third (Fig. 2a and b), for a given level of solution accuracy, the CPU time required is lower for the smaller value of ATOL (10^{-14}) than for the larger value of ATOL (10^{-10}), for both solvers. This last result is somewhat counterintuitive, but can be understood in terms of the values of RTOL that are required in each case. For a given level of global solution accuracy, the value of RTOL required decreases as the value of ATOL increases, the net effect being that the solver must do more work for the larger value of ATOL. This suggests that there is no advantage to using a value of ATOL that is larger than 10^{-14} (say), and that RTOL should be used (with ATOL = 10^{-14}) to control the solution accuracy to the desired level in all cases. The lower solution accuracy and higher CPU time for BDF compared to extrapolation is not evident when both solvers are allowed to integrate to the specified end time in one sweep without reinitialization (not shown). The loss of accuracy for the BDF solver with repeated stopping/reinitialization is presumably a consequence of the accumulation of errors associated

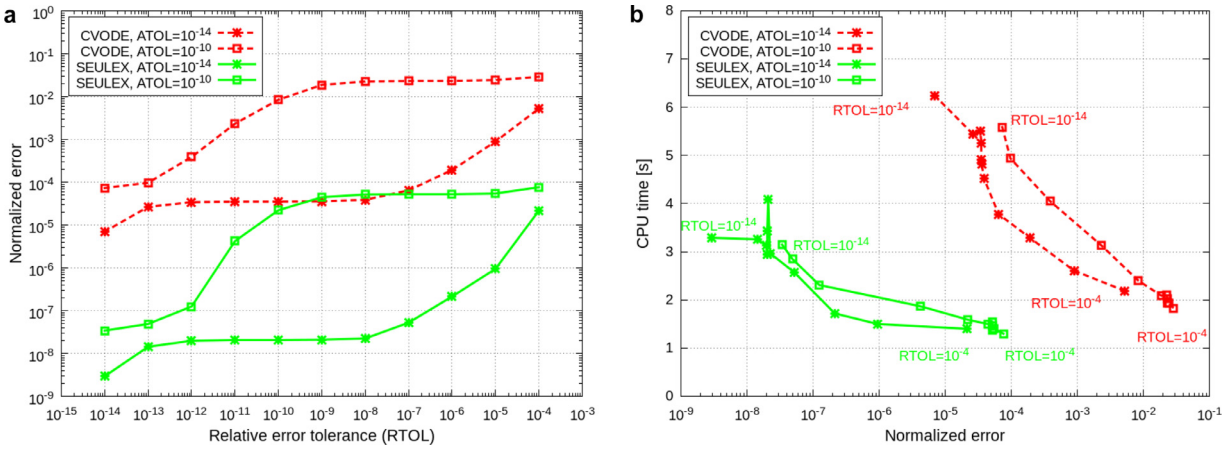

Fig. 2. Relationships among user-specified error tolerances, global solution accuracy and CPU time for a homogeneous adiabatic system with reinitialization once every 10^{-6} s, for a BDF solver (CVODE) and an extrapolation solver (SEULEX). (a) Normalized error in the computed temperature at one instant in time versus RTOL for two different values of ATOL. (b) CPU time versus normalized error in the computed temperature at one instant in time for two different values of ATOL, and the same range of RTOL values as in Fig. 2a.

with solver reinitialization (restarting with a small Δt_{ODE} using a first-order method, and a large number of floating-point operations required to build back up to larger Δt_{ODE} and higher order). This behavior is not peculiar to CVODE or to the specific implementation that has been used here. The same behavior has been confirmed with CHEMKIN/CONP/DVODE [30] and with DASSL/DASAC [43], as expected, since these solvers share a common lineage with CVODE. Reinitialization does not impact the extrapolation solver to the same degree, and the maximum order of accuracy is available even on the first solver internal time step.

Further insight into differences in solver behavior can be drawn from Fig. 3. Two-stage ignition is evident in the computed temperature versus time profile (Fig. 3a). The solver internal order of accuracy (the value reached at the end of each CFD time step of 10^{-6} s) and internal time step Δt_{ODE} (the value reached at the end of each CFD time step of 10^{-6} s) as functions of time through the simulation are shown in Fig. 3b, and the cumulative numbers of right-hand-side evaluations (**S** in Eq. 1) and Jacobian updates as functions of time are shown in Fig. 3c. Since 1000 internal time steps are taken here, the Jacobian must be recalculated a minimum of 1000 times. For both solvers, the two-stage ignition process can be seen as a decrease in the solver internal time step at the times corresponding to first-stage ($t \approx 0.33$ ms) and second-stage ($t \approx 0.51$ ms) ignition (Fig. 3b), and also in the slopes of the curves of Fig. 3c. The internal time step for BDF is always smaller than the CFD time step of 10^{-6} s. In contrast, the internal time step for the extrapolation solver is larger than the CFD time step at all times other than during the first- and second-stage ignition. This suggests that a stiff implicit solver may not be needed over much of the simulation. An advantage of the extrapolation solver (that has not been exploited here) is that it can be readily configured as a high-order explicit solver simply by deleting the Jacobian calculation and the linear system solution [16]. For the BDF solver, the maximum order of accuracy (5th order) is used from before the first-stage ignition until after the second-stage ignition, whereas for the extrapolation solver, the maximum order of accuracy used (10th order) is less than the maximum order available (12th order) through the full simulation, and the two-stage ignition signature is clearly evident in the order-of-accuracy-versus-time plot. Fewer right-hand-side evaluations and Jacobian updates are required by the extrapolation solver compared to the BDF solver, and this contributes to the higher CPU time and possibly to the lower solution accuracy for the BDF solver (higher accumulation of roundoff errors).

To determine the extent to which these advantages of the extrapolation solver can be attributed to its higher internal order of accuracy compared to the BDF solver, the same cases shown in Figs. 2 and 3 were repeated using the extrapolation solver with the maximum internal order of accuracy limited to 5th order (the same as that for the BDF solver). The error-versus-RTOL plots are essentially identical to those shown in Fig. 2a for the 12th-order extrapolation method (not shown), but interesting differences can be seen in CPU time versus accuracy (Fig. 4a) and in internal solver behavior (Fig. 4b). For the smaller error values, the CPU time required for the 5th-order extrapolation method is higher than that for the 12th-order extrapolation method, because smaller internal time steps are required during the first- and second-stage ignition events for the lower-order method; but the 5th-order extrapolation method still shows a significant advantage compared to the 5th-order BDF method. This shows that the advantages of extrapolation compared to BDF are not due solely to the higher internal order of accuracy for the extrapolation solver, but higher-order accuracy is beneficial in reducing overall CPU time when high solution accuracy is needed. It is relatively easy to implement higher-order accuracy in an extrapolation method (by going to deeper levels of partitioning) compared to a BDF method (by retaining information from more earlier internal time steps).

The results presented in Figs. 2–4 are for one physical quantity (temperature) at one instant in time (during the main ignition event) for one chemical mechanism and one initial condition. However, qualitatively similar behavior is found for other physical quantities (including major and minor species mass fractions) at other instants in time and for other chemical mechanisms. Examples are provided in the Supplementary Material. Four general conclusions are: (1) for best performance from either solver, a small value of ATOL (e.g., 10^{-14}) should be used, and RTOL should be used to control global solution accuracy; (2) the BDF solver requires smaller values of RTOL and ATOL to realize the same global solution accuracy as the extrapolation solver; (3) for a given level of global solution accuracy, the extrapolation solver is faster; and (4) the global solution error for the BDF solver cannot be driven as low as that for the extrapolation solver for any values of the user-specified tolerances. Again, this is probably a consequence of the BDF stopping/starting issues, in combination with the differences in accumulation of roundoff errors for the two methods. The computational efficiency advantage of extrapolation over BDF for all chemical mechanisms in Table 1 will be discussed further in the following subsection.

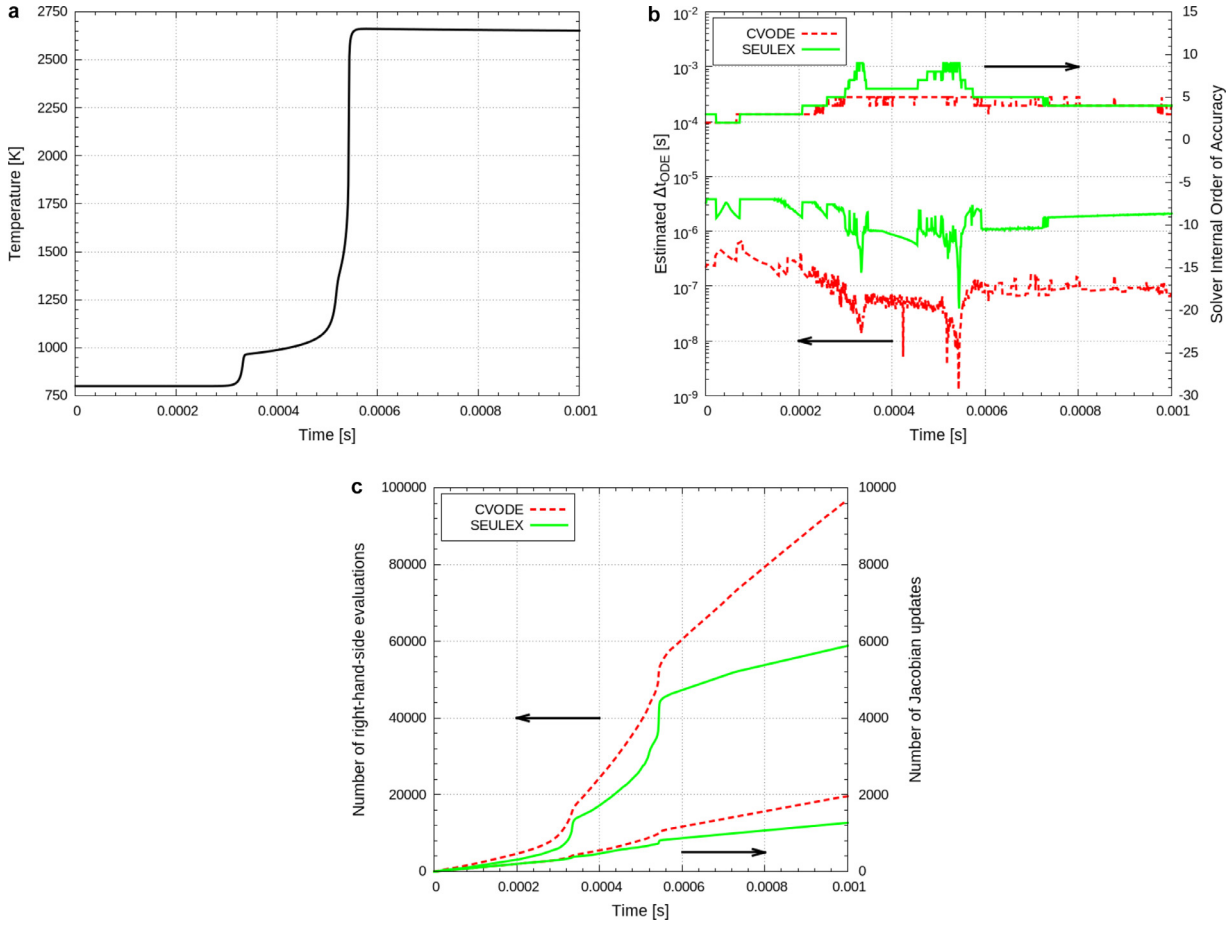

Fig. 3. Internal solver behavior for a homogeneous adiabatic system with reinitialization once every 10^{-6} s, for a BDF solver (CVODE) and an extrapolation solver (SEULEX). (a) Computed temperature versus time. (b) Solver internal order of accuracy and internal time step Δt_{ODE} as functions of time for ATOL=RTOL=10^{-14}. (c) Numbers of right-hand-side evaluations and Jacobian updates as functions of time for ATOL=RTOL=10^{-14}.

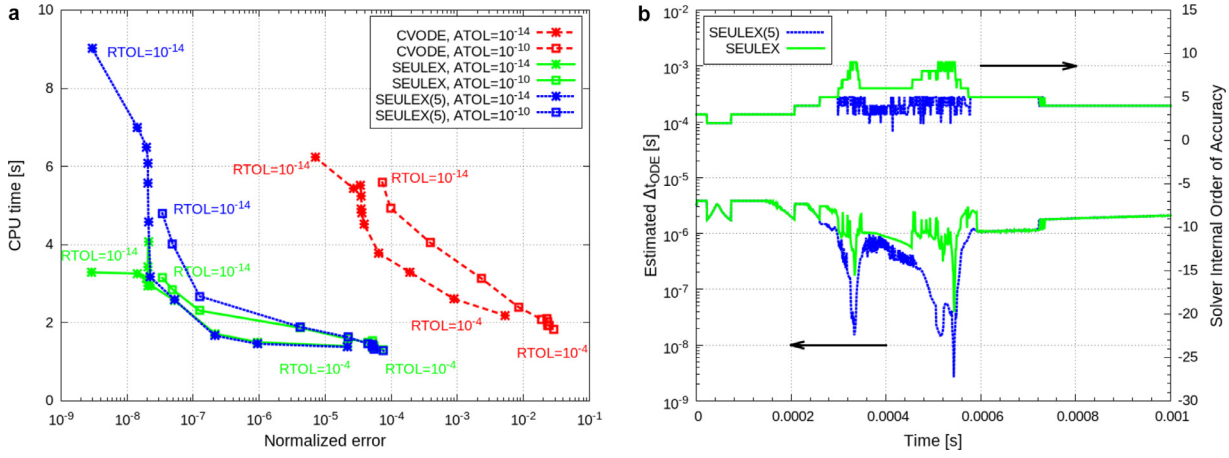

Fig. 4. (a) CPU time versus normalized error in the computed temperature at one instant in time for two different values of ATOL, and the same range of RTOL values as in Fig. 2a, including results for the extrapolation method (SEULEX) limited to 5th-order internal accuracy. (b) Solver internal order of accuracy and internal time step Δt_{ODE} as functions of time for ATOL=RTOL=10^{-14}, for a 5th-order extrapolation method and a 12th-order extrapolation method.

Ideally, solver performance comparisons would be made on the basis of equal global solution accuracy, by using different values of ATOL and RTOL for each solver (for example). That would be feasible for homogeneous systems where it is relatively straightforward to quantify accuracy, but would be difficult for engine simulations. In the results that follow, comparisons are made on the basis of the same values of ATOL and RTOL for each solver, with the understanding that this gives an advantage to BDF, since for the same level of global solution accuracy, looser tolerances could be used with the extrapolation solver. In most cases, results will be shown for ATOL = 10^{-14} and RTOL = 10^{-8}. For reasons that will be discussed later, most of the direct-injection compression-ignition results will be for ATOL = 10^{-14} and RTOL = 10^{-4}, and comparisons with results for different tolerances will be shown to illustrate tradeoffs between accuracy and CPU time for engine simulations.

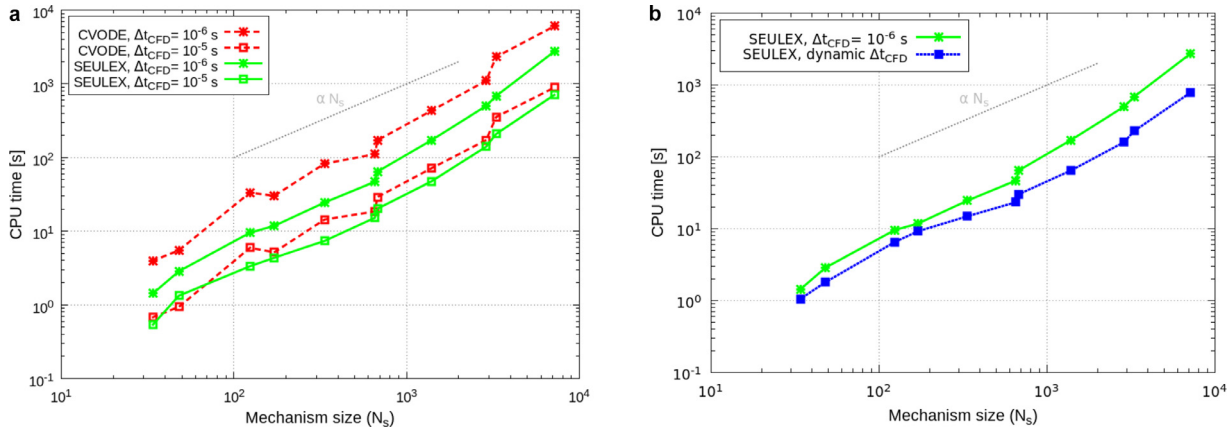

Fig. 5. CPU time versus mechanism size for a homogeneous adiabatic system, without DAC. (a) Results for a BDF solver (CVODE) and an extrapolation solver (SEULEX) with reinitialization once every 10^{-6} s and once every 10^{-5} s. (b) Results for an extrapolation solver (SEULEX) with reinitialization once every 10^{-6} s and with dynamic time stepping.

4.1.2. Mechanism size sweep

CPU time required to integrate to 1 ms of physical time for the same homogeneous adiabatic system considered in the previous subsection is plotted in Fig. 5a. There results for all chemical mechanisms in Table 1 are shown, for BDF and extrapolation methods, each with two different reinitialization intervals (10^{-6} s and 10^{-5} s), and with ATOL = 10^{-14} and RTOL = 10^{-8} in all cases. Nearly linear increase in CPU time with increasing mechanism size is realized, as has been reported in earlier studies without consideration of reinitialization [14,15]. There is some departure from linearity for the largest mechanisms, which might be improved by implementing an iterative solver with Jacobian filtering [15]. Perfectly linear scaling is not expected in any case, as different mechanisms have different ratios of number of reactions to number of species, and different degrees of stiffness, among other things. The computational efficiency advantage of extrapolation over BDF increases with decreasing reinitialization time interval Δt_{CFD}, because the number of solver reinitializations required is proportional to $1/\Delta t_{\mathrm{CFD}}$, and the reinitialization overhead is smaller for extrapolation compared to BDF. A further factor of two-to-three reduction in CPU time for the extrapolation method (compared to using a fixed value of $\Delta t_{\mathrm{CFD}} = 10^{-6}$ s) is realized by using the solver's internal time step estimation (Δt_{ODE}) as the next reinitialization time interval Δt_{CFD} (Fig. 5b). This "dynamic time stepping" would not benefit the BDF solver, unless the relatively large amount of information that would need to be saved to restart the solver (for each computational element, in the case of a multidimensional engine CFD simulation) is saved from one time step to the next.

Figure 6 shows CPU profiling results versus mechanism size for the extrapolation solver with $\Delta t_{\mathrm{CFD}} = 10^{-6}$ s. The fraction of total CPU time required for everything other than the chemistry calculations decreases from approximately 35% for the 34-species mechanism to less than 5% for the 7171-species mechanism. The chemistry (stiff ODE solver) CPU time is broken down into contributions from Jacobian updates, right-hand-side evaluations (**S** in Eq. 1), the KLU direct sparse linear solver (further subdivided to break out the times for KLU analyze, KLU factorization and KLU solve substeps), and the time for other solver-related operations. The latter include other stiff ODE solver operations such as data kind conversions and array copying/reindexing operations. The fraction of CPU time required by the direct solver remains approximately constant across all mechanism sizes, while the fraction required for right-hand-side evaluations decreases with increasing mechanism size, and the fractions required for Jacobian updates and for other operations increase with increasing mechanism size. This suggests that to realize further improvements for large mechanisms, the focus should be on further reducing the number of Jacobian updates and the overhead required for other internal operations. These results can be compared to those reported in [14], for example, where the linear system solver time dominated.

4.2. HCCI engine

Here, we consider the compression ignition of initially homogeneous premixed reactants, where inhomogeneity develops as the simulation proceeds as a consequence of heat transfer to a fixed-temperature wall. This more fully exercises the chemistry-CFD coupling, compared to the homogeneous cases considered earlier. The configuration adopted is one that was used in an earlier simulation study [27]; there are no corresponding experimental measurements. A 16.6:1 compression ratio pancake engine operates at 1500 rpm, with an initial (at intake-valve closure – IVC) quiescent mixture of n-heptane, air and recirculated exhaust gas (equivalence ratio $\Phi = 0.5$, 18% exhaust-gas recirculation – EGR) at 350 K and 1 bar. The fixed wall temperature is 450 K. A coarse two-dimensional (axisymmetric) mesh of 136 finite-volume cells is used, and all simulations were run using a single core with a fixed computational time step of $\Delta t_{\mathrm{CFD}} = 10^{-6}$ s. Baseline solver tolerances are ATOL = 10^{-14} and RTOL = 10^{-8}.

Computed heat-release traces for the baseline solver tolerances show no discernable differences between the BDF method and the extrapolation method (not shown), but closer examination reveals differences in solution accuracy and CPU time. For BDF, a small temporal phase shift is observed between the CO and OH global volume fraction solutions obtained using the tightest tolerances and the solutions obtained using what would probably be the loosest acceptable value of RTOL (RTOL = 10^{-3}, not shown); no further changes are evident when this is combined with DAC. For extrapolation, there are no apparent differences between the solutions obtained using different values of the user-specified tolerances, or with versus without DAC.

Solution accuracy and CPU time for a 48-species chemical mechanism [33] with variations in RTOL and with versus without DAC are explored in Fig. 7. There the CPU time required for the full run (from 137 crank angle degrees before piston top dead center – TDC – to 112 crank angle degrees after TDC) is plotted as a function of the normalized error in the computed global OH volume fraction at the instant of peak heat-release rate and the error in the computed global NO volume fraction at the end of the simulation; the

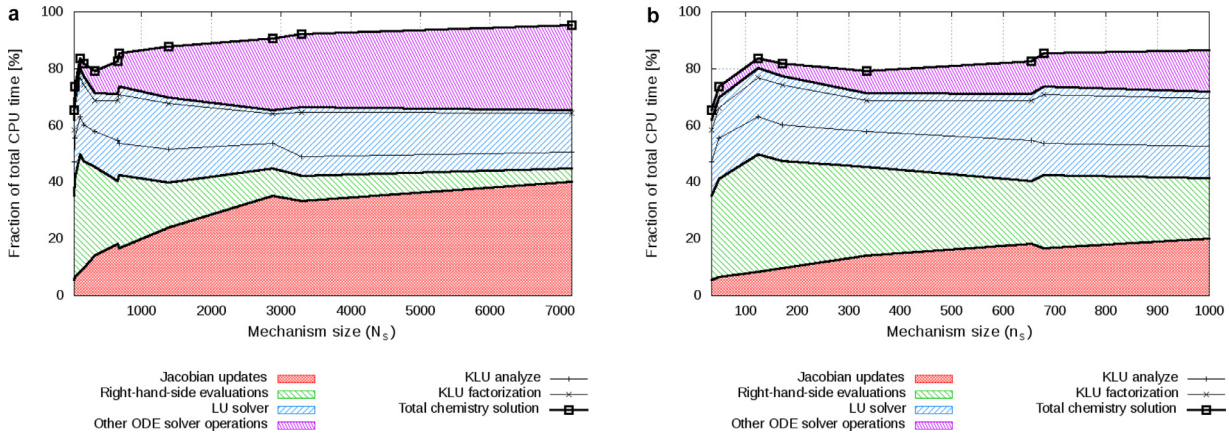

Fig. 6. Fractions of total simulation CPU time for various operations for a homogeneous reactor with reinitialization once every 10^{-6} s using SEULEX, versus mechanism size. The white space above the uppermost curve (the solid line with square symbols) corresponds to the fraction of CPU time required for everything other than chemistry. (a) 34-species to 7171-species mechanisms. (b) Expanded view of Fig. 6a from 34 species to 1000 species.

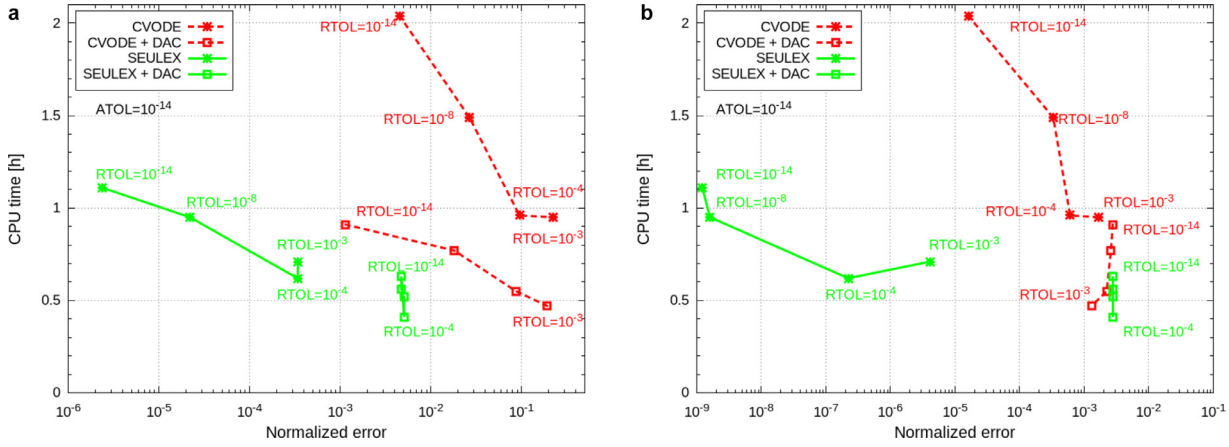

Fig. 7. CPU time versus normalized error for a HCCI engine with ATOL = 10^{-14} and values of RTOL from 10^{-14} to 10^{-3} for BDF (CVODE) and extrapolation (SEULEX) solvers with and without DAC. (a) Error in global OH volume fraction at instant of peak heat-release rate. (b) Error in global NO volume fraction at end of simulation.

normalized errors are computed with respect to a reference solution with ATOL = RTOL = 10^{-15}, as before. The results without DAC are qualitatively similar to those shown earlier for temperature in a homogeneous reactor (Fig. 4): extrapolation gives a more favorable CPU time versus error tradeoff, and allows errors to be driven to lower levels. When combined with DAC, overall CPU time is lower, and accuracy is limited by the DAC tolerance (here 10^{-4}) in most cases. An exception is that for BDF with DAC, the error in global OH volume fraction decreases with decreasing RTOL (Fig. 7a); in that case, the solver error is not well below the DAC error. In general, it is difficult to cleanly isolate and quantify individual sources of error in a complex nonlinear system, where different sources of error (stiff ODE solver errors, DAC errors, and splitting errors, including the phase shift noted earlier), can interact and compensate in various ways.

A sweep of CPU time versus mechanism size (up to 1389 species) is shown in Fig. 8 for BDF and extrapolation solvers, each with and without DAC. Without DAC, the CPU time increases faster than linearly with increasing mechanism size. In contrast to the homogeneous reactor case, here the temperature changes as a result of piston motion as well as chemistry, and the temperature (and eventually composition) become nonhomogeneous because of wall heat transfer. This exacerbates the reinitialization problem even for the extrapolation solver, since the solver's internal estimate of Δt_{ODE} from the end of the previous time step is not as good of an estimate for the beginning of the next time step as it is in the homogeneous case. More Jacobian updates are required,

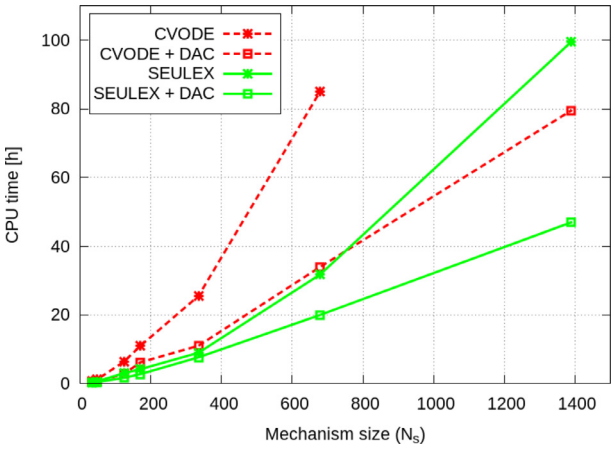

Fig. 8. CPU time versus mechanism size for a HCCI engine for BDF (CVODE) and extrapolation (SEULEX) solvers with ATOL = 10^{-14}, RTOL = 10^{-8}, with and without DAC.

and the Jacobian calculation effort increases faster than linearly with increasing mechanism size. Close to linear scaling is recovered with DAC. The reduction in CPU time with DAC is not as dramatic as what has been reported in earlier studies [26,27]; in general, the more efficient the stiff ODE solver, the less the advantages of other acceleration strategies are expected to be. Even so, there are non-negligible benefits from using DAC with both solvers

even for the smallest mechanisms, and approximately a factor of two reduction in CPU time for the largest mechanism tested.

4.3. Direct-injection (nonhomogeneous-charge) compression-ignition engines

The final configuration is highly nonhomogeneous combustion in a direct-injection compression-ignition engine. Compared to the HCCI engine, this is expected to stress the stiff ODE solver even more severely, because of the highly nonuniform mixture and stronger contributions of physical processes other than chemistry. For this purpose, a simplified model of the Sandia heavy-duty optical engine [44] is considered. A coarse (8784-cell) 45° sector mesh is centered on one of the eight spray plumes, and includes the piston top-ring-land crevice. Simulations begin after IVC with prescribed initial in-cylinder conditions, and end before exhaust-valve opening. A computational time step of 10^{-6} s is used, in all cases. Chemistry is calculated at the finite-volume cell level using chemical mechanisms that range in size from 42 to 1389 species (Table 1), thereby neglecting turbulence-chemistry interactions. The focus here is not on quantitative comparisons with experiment, but rather on the relative performance of different ODE solver variants and DAC.

Simulations have been performed for two different operating conditions: a conventional high-temperature combustion (HTC) diesel operating condition where autoignition occurs before the end of fuel injection, and a low-temperature combustion (LTC) operating condition, where the ignition delay is longer and autoignition occurs after the end of fuel injection. In both cases, the engine operates with a 11.2:1 compression ratio at 1200 rpm with a swirl ratio of 0.5. For HTC, the IVC pressure and temperature are 233 kPa and 384 K, respectively, the initial oxygen volume fraction is 21% (no EGR), and 61 mg of fuel is injected at 1200 bar injection pressure from 7° before TDC to 3° after TDC. Results for six chemical mechanisms (42, 134, 171, 335, 679 and 1389 species) are presented. The fuel is taken to be PRF10 in all cases, with the exception of the smallest mechanism (42 species), where it is taken to be n-heptane.

For LTC, the IVC pressure and temperature are 202 kPa and 343 K, respectively, there is 12.7% O_2 initially, and 56 mg of fuel is injected at 1600 bar injection pressure from TDC to 7° after TDC. LTC results are presented for only three chemical mechanisms (171, 679 and 1389 species). The fuel is taken to be PRF10, to ensure ignition.

An example illustrating the CPU time versus accuracy tradeoff for highly nonhomogeneous combustion is shown in Fig. 9. There the CPU time versus normalized error in the end-of-simulation global NO volume fraction (as defined earlier) is shown for BDF and extrapolation solvers with and without DAC, with ATOL = 10^{-14}, and with values of RTOL ranging from 10^{-14} to 10^{-3}, for a 42-species mechanism. One must exercise caution in trying to draw conclusions regarding the errors associated with the stiff ODE solver and with DAC, when other numerical errors (e.g., stochastic spray models, splitting errors) are non-negligible and there are potentially complex interactions among the different sources of error. With this caveat, it is noted that most of the general trends in Fig. 9 are consistent with those in Fig. 7. Without DAC, the extrapolation solver continues to show better accuracy than the BDF solver for the same values of ATOL and RTOL, although the behavior for extrapolation with varying RTOL is non-monotonic. And the extrapolation solver continues to give a better CPU time versus accuracy tradeoff compared to the BDF solver, both with and without DAC.

Finally, sweeps of CPU time versus mechanism size (up to 1389 species) for HTC and LTC are shown in Figs. 10 and 11, respectively. There extrapolation results with and without DAC are

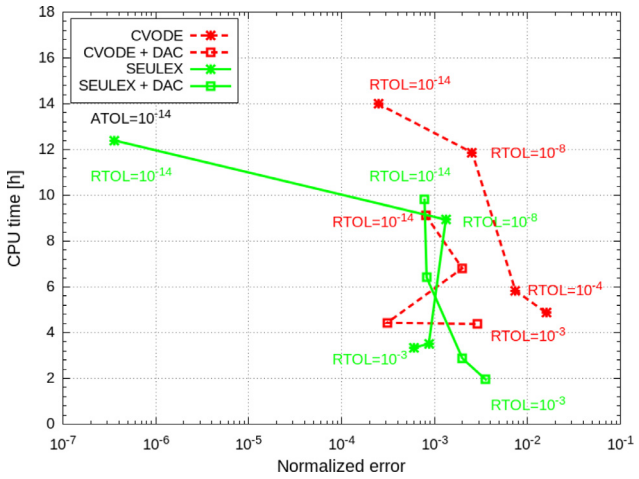

Fig. 9. CPU time versus normalized error in global NO volume fraction at end of simulation for a HTC engine with ATOL = 10^{-14} and values of RTOL from 10^{-14} to 10^{-3} for BDF (CVODE) and extrapolation (SEULEX) solvers with and without DAC.

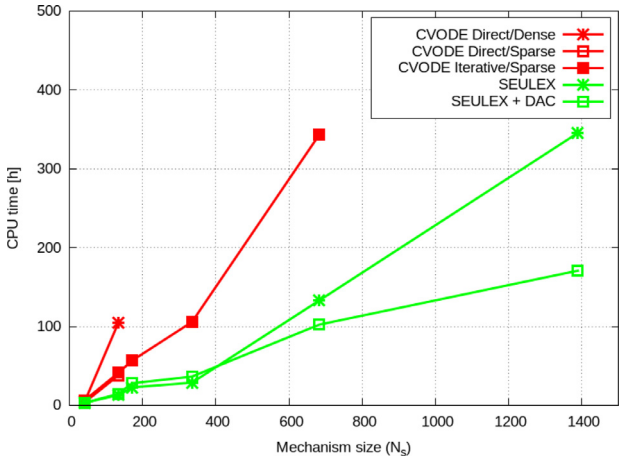

Fig. 10. CPU time versus mechanism size for a conventional (high-temperature combustion) diesel engine. Results are shown for a BDF solver (CVODE) for up to a 654-species mechanism. Results for an extrapolation solver (SEULEX) and an extrapolation solver with DAC are shown for up to 1389-species mechanisms.

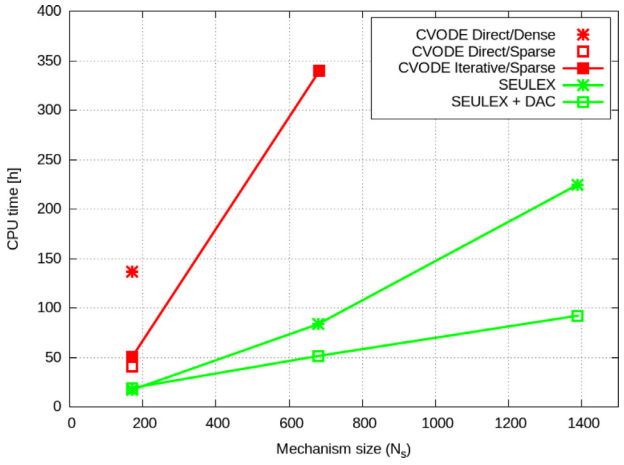

Fig. 11. CPU time versus mechanism size for a low-temperature combustion engine. Results are shown for a BDF solver (CVODE) for up to a 654-species mechanism. Results for an extrapolation solver (SEULEX) and an extrapolation solver with DAC are shown for up to 1389-species mechanisms.

shown for mechanisms up to 1389 species, while BDF results are not shown for the largest mechanism. The BDF results include the baseline BDF/iterative solver/sparse matrix combination and two other combinations: BDF/direct solver/sparse matrix and BDF/direct solver/dense matrix. For this mechanism size, the direct solver has a modest advantage over the iterative solver, but the dense matrix representation is not competitive. A full mechanism sweep was not run for BDF, as the benefits of extrapolation are already evident, and its advantage increases with increasing mechanism size, as shown earlier. Again, extrapolation with DAC gives approximately linear increase in CPU time with increasing mechanism size. In this case, the overhead of DAC is not recovered for mechanisms smaller than approximately 200–400 species (depending on the engine configuration), but DAC gives at least a factor of two improvement for the largest mechanism tested (1389 species).

5. Conclusions

A fresh look has been taken at stiff ODE solvers for combustion CFD, by considering the reinitialization problem that is inherent in operator-splitting strategies. This leads to consideration of a class of stiff ODE solvers based on extrapolation methods [16,17]. Performance and accuracy comparisons between a BDF method (CVODE) and an extrapolation method (SEULEX) have been presented for chemical mechanisms ranging in size from 34 to 7171 species, and for three different configurations: homogeneous reactors, HCCI engines, and direct-injection compression-ignition engines. Differences in internal solver behavior between the BDF solver and the extrapolation solver have been demonstrated and explained, and the relationships among user-specified error tolerances, solution accuracy, and CPU time have been explored.

The single most important conclusion is that extrapolation methods appear to offer significant advantages in accuracy and computational efficiency compared to BDF methods for computational combustion, and warrant further investigation in this regard. The advantages are especially pronounced because of the operator-splitting strategies that are used for combustion CFD (where solver reinitialization is a key issue), and because of the range of relevant chemical time scales compared to the CFD time steps that are used in practice. Specific findings are as follows.

- The extrapolation solver yields higher solution accuracy than the BDF solver for the same values of ATOL and RTOL. The extrapolation solver also gives a more favorable tradeoff between CPU time and solution accuracy. These advantages remain even when the extrapolation solver is limited to the same formal internal order of accuracy as the BDF solver. The reasons for the differences appear to be related to BDF stopping/restarting issues, and differences in accumulations of roundoff errors.
- For both solvers, solution accuracy should be controlled using RTOL, together with a small value of ATOL (e.g., ATOL $= 10^{-14}$). The value of RTOL required depends on the solver type and the application. For engine applications, RTOL $= 10^{-4}$ is sufficient for the extrapolation solver.
- Even for the same values of the error tolerances (implying higher solution accuracy for extrapolation compared to BDF), the extrapolation solver shows significant reductions in computational time compared to the BDF solver across all chemical mechanisms and all configurations considered. The advantages of the extrapolation method increase with decreasing CFD time step, and further benefits can be realized by storing the internal solver time-step estimate and using that at the beginning of the next CFD time step.
- The speedups that can be realized with other chemistry acceleration strategies (here DRGEP-based DAC) diminish with increasing efficiency of the underlying stiff ODE solver. Depending on the configuration, the overhead of DAC may not be recovered for smaller chemical mechanisms, but a speedup of at least a factor of two has been found in engine simulations for the largest mechanism tested (1389 species).
- Approximately linear increase in CPU time with increasing mechanism size is found using the extrapolation method for homogeneous reactors (with reinitialization), as has been reported earlier for BDF methods without consideration of reinitialization [14,15].
- In HCCI and direct-injection engine simulations, the rate of increase in total simulation time with increasing mechanism size is somewhat faster than linear. When combined with DAC, overall approximately linear scaling with increasing mechanism size is achieved.
- There is potential for further significant improvements in computational efficiency for extrapolation solvers. This includes (for example) using second-order midpoint-rule integration to calculate the first column of the divided difference table [45], and taking advantage of what has been learned in BDF-based studies of Jacobian sparsity with iterative solvers [15].

Further improvements in extrapolation solvers for computational chemistry are the subject of ongoing work, including implementation in particle-based transported PDF methods.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.combustflame.2016.09.018

Acknowledgments

Funding for this research was provided by ExxonMobil Research and Engineering Company, and the authors are grateful for technical support from our ExxonMobil contacts Dr. Kenneth Kar and Mr. Walter Weissman. The authors also are grateful for technical discussions on various aspects of stiff ODE solvers with Drs. Matthew McNenly and Russell Whitesides of Lawrence Livermore National Laboratory.

Appendix A. CVODE algorithm

Details of the BDF method that has been used here (CVODE) can be found in [21]. Since BDF methods have been widely used for combustion CFD, only a few key elements of the algorithm are highlighted here.

The stiff ODE solver is called for each computational element to solve the initial value problem of Eq. 1, given $Y(t_0)$ and Δt_{CFD}. Based on the values of the user-specified tolerances RTOL and ATOL, an initial value of the solver internal time step, $\Delta t_{ODE,\,1}$, is determined. For the chemical mechanisms and conditions that are of interest here, typical values of $\Delta t_{ODE,\,1}$ range between 10^{-12} s and 10^{-8} s. A first-order method is used to obtain an estimate of $Y(t_0 + \Delta t_{ODE,1})$. The general idea is then to continue to integrate to the end of the current CFD time step, taking internal time steps Δt_{ODE} that are as large as possible, and using as high of an order of accuracy in the time integration (denoted as k) as possible (up to an algorithm-prescribed upper limit k_{max}; here $k_{max} = 5$), where Δt_{ODE} and k for each internal time step are selected to minimize the overall computational time while ensuring stability and maintaining the user-specified level of accuracy.

Specifically, to advance from internal solver time level $n-1$ to time level n, a kth-order multistep BDF method can be written as:

$$\mathbf{Y}_n = \sum_{i=1}^{k}\left(a_i \mathbf{Y}_{n-i}\right) + \Delta t_{\mathrm{ODE,n}} b_0 \mathbf{S}_n \ . \tag{A.1}$$

Here, the values of $\mathbf{Y}_{n-i}$ are known (from previous internal time steps), and a_i $(i = 1, \ldots, k)$ and b_0 are the coefficients associated with the BDF method. From Eq. A.1, it can be seen that solutions from a minimim of k previous internal time steps must be available to achieve kth-order accuracy in the time integration. For $b_0 = 0$, the method is explicit in time. In general, $b_0 \neq 0$ and Newton iteration is used to solve the implicit form of Eq. A.1. In that case, the linear system that is solved to advance from iteration level m to iteration level $m+1$ on internal time step n can be written as,

$$\left(\mathbf{I} - \Delta t_{\mathrm{ODE,n}} b_0 \mathbf{J}\right)\left(\mathbf{Y}_{n,m+1} - \mathbf{Y}_{n,m}\right) = -\mathbf{R}(\mathbf{Y}_{n,m}) \ , \tag{A.2}$$

where $\mathbf{J}$ is the Jacobian matrix and $\mathbf{R}$ is the iteration-level-m residual for Eq. A.1:

$$\mathbf{R}(\mathbf{Y}_{n,m}) = \mathbf{Y}_{n,m} - \sum_{i=1}^{k}\left(a_i \mathbf{Y}_{n-i}\right) - \Delta t_{\mathrm{ODE,n}} b_0 \mathbf{S}_{n,m} \ . \tag{A.3}$$

Convergence is checked as follows. The change in each dependent variable α for iteration m is computed as $\Delta_{\alpha,m} = |Y_{\alpha,m} - Y_{\alpha,m-1}|$ $(\alpha = 1, \ldots, N_\phi)$, and a scaled error is computed as:

$$err_m \equiv \sqrt{\frac{1}{N_\phi}\sum_{\alpha=1}^{N_\phi}\left(\frac{\Delta_{\alpha,m}}{scale_\alpha}\right)^2} \quad \text{where}$$
$$scale_\alpha \equiv \mathrm{ATOL} + |Y_{\alpha,n,m}| \cdot \mathrm{RTOL}. \tag{A.4}$$

The solution has converged when $err_m \leq 1$. If the solution does not converge after three iterations, $\Delta t_{\mathrm{ODE,\,n}}$ is reduced, the order k is reset to one, and internal time step n is retried using the smaller time step.

A change in the order of accuracy k is considered after $k+1$ internal time steps at order k. The order used for the next internal time step (k') can then be equal to $k-1$, k, or $k+1$ (such that $1 \leq k' \leq k_{max}$), and both the new order of accuracy and the new time step are selected based on an estimate of the combination that is expected to minimize the work required to achieve a converged solution.

Appendix B. SEULEX algorithm and C++ implementation

The SEULEX algorithm is presented in Chapter 9 of [17] and in Chapter 17 of [46], and a C++ implementation has been provided with recent versions of OpenFOAM [25]. Here we started with an OpenFOAM implementation, and modified that to use the KLU sparse linear algebra libraries and to improve robustness in applications to combustion problems. Because extrapolation-based solvers are not as familiar in combustion CFD compared to BDF methods, more detail is provided here compared to Appendix A. The main ingredients of the algorithm are provided in the following, and OpenFOAM v2.3.x C++ code is provided in the Supplementary Material. Computer files of source code are available on request from the authors.

The stiff ODE solver is called for each computational element to solve the initial value problem of Eq. 1, given $\mathrm{Y}(t_0)$ and Δt_{CFD}. The solver requires the user-specified tolerances RTOL and ATOL, an initial estimate of the solver internal time step Δt_{ODE}, and an initial target order of accuracy for the time integration (denoted as k_{targ}). As discussed in the main text, Δt_{ODE} is taken to be the solver's internal estimate of Δt_{ODE} from the end of the previous CFD time step for the computational element. The value of k_{targ} can be specified directly, or can be computed based on the values of the user-specified error tolerances RTOL and ATOL, and on the maximum available order k_{max} (here $k_{\mathrm{max}} = 12$) as: $k_{\mathrm{targ}} = \min\{k_{\mathrm{max}}, k_{\mathrm{est}}\}$, where $k_{\mathrm{est}} = \max\{3, \mathrm{int}(0.5 - 0.6\log_{10}(\mathrm{RTOL} + \mathrm{ATOL}) + 1)\}$, so that $3 \leq k_{\mathrm{targ}} \leq k_{\mathrm{max}}$.

$$\begin{matrix}
\mathbf{Y}_{1,1} & & & & \\
\mathbf{Y}_{2,1} & \mathbf{Y}_{2,2} & & & \\
\mathbf{Y}_{3,1} & \mathbf{Y}_{3,2} & \mathbf{Y}_{3,3} & & \\
\mathbf{Y}_{4,1} & \mathbf{Y}_{4,2} & \mathbf{Y}_{4,3} & \mathbf{Y}_{4,4} & \\
\mathbf{Y}_{5,1} & \mathbf{Y}_{5,2} & \mathbf{Y}_{5,3} & \mathbf{Y}_{5,4} & \mathbf{Y}_{5,5}
\end{matrix}$$

Fig. B1. SEULEX extrapolation table structure for $k_{\mathrm{targ}} = 5$. A maximum of k_{targ} rows will be computed. The first column is a succession of first-order approximations to $\mathbf{Y}(t_0 + \Delta t_{\mathrm{ODE}})$ obtained using a semi-implicit Euler method with progressively finer (on successive rows k) uniform partitions of Δt_{ODE} (Eq. B.1). The remaining columns are higher-order approximations to $\mathbf{Y}(t_0 + \Delta t_{\mathrm{ODE}})$ obtained algebraically using Eq. B.2, where the order of accuracy increases by one in each successive column. In general, $\mathbf{Y}_{k,j}$ is a jth-order-accurate estimate of $\mathbf{Y}(t_0 + \Delta t_{\mathrm{ODE}})$ based on partitions $k, k-1, \ldots, k-j+1$.

The general idea on each internal solver time step Δt_{ODE} is to construct a sequence of progressively more accurate approximations to $\mathbf{Y}(t_0 + \Delta t_{\mathrm{ODE}})$, until a solution is found that is accurate to within the user-specified tolerances RTOL and ATOL. Then Δt_{ODE} and k_{targ} are updated, and integration continues over the next solver internal time step, until eventually the end of the current CFD time step Δt_{CFD} is reached. The following sequence of operations is performed for each solver internal time step Δt_{ODE}.

The internal time step Δt_{ODE} is subdivided into a sequence of k_{max} uniform partitions of substeps $h_k = \Delta t_{\mathrm{ODE}}/n_k$, where the sequence of partitions for $k_{\mathrm{max}} = 12$ is:

$$n_k = 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96 \quad (k = 1, 2, \ldots, 12) \ . \tag{B.1}$$

A first-order semi-implicit Euler method is used to solve Eq. 2 to obtain two estimates of $\mathbf{Y}(t_0 + \Delta t_{\mathrm{ODE}})$ using time steps $h_1 = \Delta t_{\mathrm{ODE}}/n_1 = \Delta t_{\mathrm{ODE}}/2$ and $h_2 = \Delta t_{\mathrm{ODE}}/n_2 = \Delta t_{\mathrm{ODE}}/3$; these solutions are denoted as $\mathbf{Y}_{1,1}$ and $\mathbf{Y}_{2,1}$, respectively. A "Jacobian stability" algorithm is applied to ensure that the time step is sufficiently small for the method to eventually converge. The basic approach was suggested by Deuflhard [16,17], based on convergence of the Newton iterations. For the linear implicit Euler method, one solves Eq. 2 for $\boldsymbol{\Delta}_0 \equiv \mathbf{Y}_{n+1} - \mathbf{Y}_n$, updates the right-hand side based on the new $\mathbf{Y}_{n+1}$ (without updating the Jacobian), and solves Eq. 2 again for $\boldsymbol{\Delta}_1$. The computational cost is that of a single right-hand-side evaluation, since the factored matrix is available from the first solution of the linear system. A monotonicity check is then performed to ensure that $\theta \equiv |\boldsymbol{\Delta}_1|/|\boldsymbol{\Delta}_0| < 1$. If $\theta \geq 1$, Δt_{ODE} is reduced by a specified factor f (here $f = 2$), and the above procedure is repeated using the reduced time step. When the time step is sufficiently small that $\theta < 1$, $\mathbf{Y}_{1,1}$ and $\mathbf{Y}_{2,1}$ are taken to be the first two entries in a divided difference table (Fig. B.12), and the algorithm proceeds as follows.

The table of successive approximations to $\mathbf{Y}(t_0 + \Delta t_{\mathrm{ODE}})$ is constructed row by row. For each row k in the table, the first column entry is computed using a semi-implicit Euler method with $h_k = \Delta t_{\mathrm{ODE}}/n_k$ (including the Jacobian stability test described above for the first two entries). Then the Aitken–Neville algorithm (or Richardson extrapolation) is used to calculate columns 2-k to yield

successively higher-order approximations to $\mathbf{Y}(t_0+\Delta t_{\mathrm{ODE}})$, where the order of accuracy increases by one in each successive column:

$$\mathbf{Y}_{k,j}=\mathbf{Y}_{k,j-1}+\frac{\mathbf{Y}_{k,j-1}-\mathbf{Y}_{k-1,j-1}}{\left(\frac{n_k}{n_{k-j+1}}\right)-1}\quad(k=2,\ldots,k_{\mathrm{targ}};\,j=2,\ldots,\ k). \tag{B.2}$$

This formula is used to compute $\mathbf{Y}_{k,\,j}$ for $2\le j\le k$, which is a jth-order approximation to $\mathbf{Y}(t_0+\Delta t_{\mathrm{ODE}})$. As each row of the table is computed, an estimate of the work required for that row (denoted as cpu_k, proportional to the CPU time used to form the row) is computed and stored, based on the numbers of Jacobian updates, linear system solves, and right-hand-side updates. After each row is computed, convergence is checked using criteria that are essentially the same as those used in CVODE. Here, the difference between $\mathbf{Y}_{k,\,k}$ and $\mathbf{Y}_{k,k-1}$ is used to determine whether or not $\mathbf{Y}_{k,\,k}$ is a sufficiently accurate estimate of $\mathbf{Y}(t_0+\Delta t_{\mathrm{ODE}})$. For each dependent variable α in table row k, $\Delta_{\alpha,k}=|Y_{\alpha,k,k}-Y_{\alpha,k,k-1}|$ $(\alpha=1,\ldots,N_\phi)$ is calculated, and a scaled error is computed as:

$$err_k\equiv\sqrt{\frac{1}{N_\phi}\sum_{\alpha=1}^{N_\phi}\left(\frac{\Delta_{\alpha,k}}{scale_\alpha}\right)^2}\quad\text{where}$$
$$scale_\alpha\equiv\mathrm{ATOL}+|Y_{\alpha,k,k}|\cdot\mathrm{RTOL}\ . \tag{B.3}$$

A provisional value of Δt_{ODE} for each row, $\Delta t'_{\mathrm{ODE},k}$, is computed and stored based on the expected relationship between the scaled error and the time step size:

$$\left(\frac{\Delta t'_{\mathrm{ODE},k}}{\Delta t_{\mathrm{ODE}}}\right)^{k+1}=\frac{1}{err_k}\ , \tag{B.4}$$

from which a provisional value for the next internal time step is:

$$\Delta t'_{\mathrm{ODE},k}=S_1\Delta t_{\mathrm{ODE}}\left(\frac{S_2}{err_k}\right)^{\frac{1}{k+1}}\ , \tag{B.5}$$

where S_1 and S_2 are heuristic "safety factors" whose values are close to, but less than, one. Convergence corresponds to $err_k<1$. If $err_k\ge1$ and $k<k_{\mathrm{targ}}$, the algorithm continues to the next value of k (to compute the next row of the table). If $err_k\ge1$ and $k=k_{\mathrm{targ}}$, Δt_{ODE} is reduced by a specified factor f (here $f=2$), and the time step starts over using the smaller value of Δt_{ODE}; the Jacobian is updated if $\theta>\min\{10^{-4},\mathrm{RTOL}\}$. If $err_k<1$, then k_{conv} (the order at which a converged solution was found) is set to k, Δt_{ODE} and k_{targ} are updated as described below, and integration proceeds to the next solver internal time step.

At the conclusion of a successful (converged) internal time step, the values of Δt_{ODE} and k_{targ} for the next internal time step are set in a manner that is expected to yield a converged solution for the least CPU time, based on the values of $\Delta t'_{\mathrm{ODE},k}$ and cpu_k that were computed and stored for the current time step. Specifically, $\Delta t_{\mathrm{ODE}}=\Delta t'_{\mathrm{ODE},k_{\mathrm{conv}}}\cdot cpu_{k_{\mathrm{conv}}}/cpu_{k_{\mathrm{conv}}-1}$, and k_{targ} can decrease by one if $cpu_{k_{\mathrm{conv}}-1}/h_{k_{\mathrm{conv}}-1}<0.7cpu_{k_{\mathrm{conv}}}/h_{k_{\mathrm{conv}}}$ and can increase by one if $cpu_{k_{\mathrm{conv}}-1}/h_{k_{\mathrm{conv}}-1}>0.9cpu_{k_{\mathrm{conv}}}/h_{k_{\mathrm{conv}}}$.

Additional tests have been implemented for robustness in combustion applications. The temperature value is checked to ensure that it remains between the lower and upper limits specified in the thermodynamic property data, and the magnitude of the scaled change in the solution vector over an iteration is checked to make sure that it is not too large. A failure of either of these checks also results in a reduction in Δt_{ODE}.

Appendix C. Structure of chemistry equations

For each computational element, a system of nonlinear coupled ODEs is solved to give the change in composition due to chemical reactions over a computational time step Δt_{CFD}. Here we consider a reacting ideal-gas mixture containing N_S species, and work in terms of molar concentrations. The following system of N_S+1 ODEs is solved:

$$\frac{dc_\alpha}{dt}=\dot\omega_\alpha\quad(\alpha=1,\ldots,N_S)\ ,$$
$$\frac{dT}{dt}=-\sum_{\alpha=1}^{N_S}\left(h_\alpha\dot\omega_\alpha\right)\Big/\sum_{\alpha=1}^{N_S}\left(c_\alpha c_{p\alpha}\right)\ . \tag{C.1}$$

Here c_α is the species-α molar concentration, T is temperature, $\dot\omega_\alpha$ is the species-α molar chemical production rate (from the given chemical mechanism), h_α is the species-α molar specific enthalpy, and $c_{p\alpha}$ is the species-α molar specific constant-pressure specific heat.

This approach has been found to be more convenient and robust compared to solving for species mass fractions, since the chemical source terms are usually given on a molar basis. It does require a conversion between species mass fractions and mole fractions on entering and exiting the solver, in the case where the underlying CFD code solves equations for species mass fractions (as is the case here). Since approximations that are made in calculating the Jacobian are of secondary importance, and a continuity equation is solved in the CFD code, the mass density ρ is kept constant over the stiff ODE solution, and thermodynamic properties are updated by the CFD code in a manner that is consistent with the underlying CFD algorithm at the end of the chemistry calculations. It has been confirmed that computed pressure, temperature and species composition time evolutions from this implementation converge to the solutions obtained using a dedicated homogeneous reactor model (e.g., CHEMKIN-based [30] adiabatic constant-pressure or adiabatic constant-volume models) as Δt_{CFD} decreases toward zero. It also has been confirmed that the general conclusions that have been drawn here regarding the behavior of BDF and extrapolation methods remain valid when alternative formulations are used (e.g., solving for mass fractions rather than mole fractions; updating properties over the course of the stiff ODE solver iterations).

References

[1] A workshop to identify research needs and impacts in predictive simulation for internal combustion engines (PreSICE), Office of Science and Office of Energy and Renewable Energy, U.S. Department of Energy, 2011. Available at https://www1.eere.energy.gov/vehiclesandfuels/pdfs/presice_rpt.pdf

[2] R.D. Reitz, Directions in internal combustion engine research, Combust. Flame 160 (2013) 1–8.

[3] D.C. Haworth, Progress in probability density function methods for turbulent reacting flows, Prog. Energy Combust. Sci. 36 (2010) 168–259.

[4] T.F. Lu, C.K. Law, A directed relation graph method for mechanism reduction, Proc. Combust. Inst. 30 (2005) 1333–1341.

[5] P. Pepiot-Desjardins, H. Pitsch, An efficient error-propagation-based reduction method for large chemical kinetic mechanisms, Combust. Flame 154 (2007) 67–81.

[6] L. Lu, S.B. Pope, An improved algorithm for *in situ* adaptive tabulation, J. Comput. Phys. 228 (2009) 361–386.

[7] M. Jangi, X.-S. Bai, Multidimensional chemistry coordinate mapping approach for combustion modelling with finite-rate chemistry, Combust. Theory Model. 16 (2012) 1109–1132.

[8] Y. Shi, W.H. Green, 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 (2011) 836–847.

[9] F. Contino, H. Jeanmart, T. Lucchini, G. D'Errico, Coupling of in situ adaptive tabulation and dynamic adaptive chemistry: an effective method for solving combustion in engine simulations, Proc. Combust. Inst. 33 (2011) 3057–3064.

[10] F. Contino, J.-B. Masurier, F. Foucher, T. Lucchini, G. D'Errico, P. Dagaut, CFD simulations using the TDAC method to model iso-octane combustion for a large range of ozone seeding and temperature conditions in a single cylinder HCCI engine, Fuel 137 (2014) 179–184.

[11] Z. Ren, G.M. Goldin, V. Hiremath, S.B. Pope, Simulations of a turbulent non-premixed flame using combined dimension reduction and tabulation for combustion chemistry, Fuel 105 (2013) 636–644.

[12] V. Hiremath, S.R. Lantz, H. Wang, S.B. Pope, Large-scale parallel simulations of turbulent combustion using combined dimension reduction and tabulation of chemistry, Proc. Combust. Inst. 34 (2013) 205–215.
[13] D.A. Schwer, J.E. Tolsma, W.H. Green, P.I. Barton, On upgrading the numerics in combustion chemistry codes, Combust. Flame 128 (2002) 270–291.
[14] F. Perini, E. Galligani, R.D. Reitz, A study of direct and Krylov iterative sparse solver techniques to approach linear scaling of the integration of chemical kinetics with detailed combustion mechanisms, Combust. Flame 161 (2014) 1180–1195.
[15] M.J. McNenly, R.A. Whitesides, D.L. Flowers, Faster solvers for large kinetic mechanisms using adaptive preconditioners, Proc. Combust. Inst. 35 (2015) 581–587.
[16] P. Deuflhard, Recent progress in extrapolation methods for ordinary differential equations, SIAM Rev. 27 (1985) 505–535.
[17] E. Hairer, G. Wanner, Solving ordinary differential equations II: Stiff and differential-algebraic problems, second revised edition, Springer-Verlag, Berlin, 1996 pp. 131–142.
[18] G.D. Byrne, A.C. Hindmarsh, A polyalgorithm for the numerical solution of ordinary differential equations, ACM Trans. Math. Softw. 1 (1975) 71–96.
[19] Y. Gao, Y. Liu, Z. Ren, T. Lu, A dynamic adaptive method for hybrid integration of stiff chemistry, Combust. Flame 162 (2015) 287–295.
[20] S.D. Cohen, A.C. Hindmarsh, CVODE, a stiff/nonstiff ODE solver in C, Comput. Phys. 10 (1996) 138–143.
[21] A.C. Hindmarsh, R. Serban, User Documentation for CVODE v2.8.2 (SUNDIALS v2.6.2),, Report number UCRL-SM-208108 (2015), 2015. Available at computation.llnl.gov/casc/sundials/documentation/cv_guide.pdf.
[22] H.J. Stetter, Analysis of discretization methods for ordinary differential equations, Springer, Berlin, 1973.
[23] SuiteSparse sparse matrix algorithms, 2015, Available at http://faculty.cse.tamu.edu/davis/suitesparse.html.
[24] X.S. Li, J.W. Demmel, J.R. Gilbert, L. Grigori, M. Shao, I. Yamazaki, SuperLU users' guide (2011). (2011). Available at http://crd-legacy.lbl.gov/~xiaoye/SuperLU/superlu_ug.pdf.
[25] OpenFOAM open source software for computational fluid dynamics, 2015. Available at http://www.openfoam.org/index.php.
[26] L. Liang, J.G. Stevens, S. Raman, J.T. Farrell, The use of dynamic adaptive chemistry in combustion simulation of gasoline surrogate fuels, Combust. Flame 156 (2009) 1493–1502.
[27] 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. Theory Model. 14 (2010) 69–89.
[28] L. Tosatto, B.A.V. Bennett, M.D. Smooke, Comparison of different DRG-based methods for the skeletal reduction of JP-8 surrogate mechanisms, Combust. Flame 160 (2013) 1572–1582.
[29] V. Raj Mohan, D.C. Haworth, Turbulence-chemistry interactions in a heavy–duty compression-ignition engine, Proc. Combust. Inst. 35 (2015) 3053–3060.
[30] R.J. Kee, F.M. Rupley, E. Meeks, J.A. Miller, CHEMKIN-III: a FORTRAN chemical kinetics package for the analysis of gas-phase chemical and plasma kinetics, Technical Report SAND96-8216, Sandia National Laboratories, Livermore, CA, 1996.
[31] A. Patel, S.C. Kong, R.D. Reitz, Development and validation of a reduced reaction mechanism for HCCI engine simulations, SAE Technical Paper No. 2004-01-0558(2004).
[32] V. Golovitchev, 2015. Available at http://www.tfd.chalmers.se/~valeri/MECH.html.
[33] S. Liu, J.C. Hewson, J.H. Chen, H. Pitsch, Effects of strain rate on high-pressure nonpremixed n-heptane autoignition in counterflow, Combust. Flame 137 (2004) 320–339.
[34] C. Huang, V. Golovitchev, A. Lipatnikov, Chemical model of gasoline-ethanol blends for internal combustion engine applications, SAE Technical Paper No. 2010-01-0543(2010).
[35] M.B. Luong, Z. Luo, T.F. Lu, S.H. Chung, C.S. Yoo, Direct numerical simulations of the ignition of lean primary reference fuel/air mixtures with temperature inhomogeneities, Combust. Flame 160 (2013) 2038–2047.
[36] L. Cai, H. Pitsch, Optimized chemical mechanism for combustion of gasoline surrogate fuels, Combust. Flame 162 (2015) 1623–1637.
[37] H.J. Curran, P. Gaffuri, W.J. Pitz, C.K. Westbrook, A comprehensive modeling study of n-heptane oxidation, Combust. Flame 114 (1998) 149–177.
[38] 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 (2011) 193–200.
[39] Y. Pei, M. Mehl, W. Liu, T. Lu, W.J. Pitz, S. Som, A multi-component blend as a diesel fuel surrogate for compression ignition engine applications, J. Eng. Gas Turbines Power 137 (2015) 111502–1–11502–9.
[40] O. Herbinet, W.J. Pitz, C.K. Westbrook, Detailed chemical kinetic mechanism for the oxidation of biodiesel fuels blend surrogate, Combust. Flame 157 (2010) 893–908.
[41] 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 C7 to C20, Combust. Flame 158 (2011) 2338–2357.
[42] R.J. Kee, F.M. Rupley, J.A. Miller, M.E. Coltrin, J.F. Grcar, E. Meeks, H.K. Moffat, A.E. Lutz, G. Dixon-Lewis, M.D. Smooke, J. Warnatz, G.H. Evans, R.S. Larson, R.E. Mitchell, L.R. Petzold, W.C. Reynolds, M. Caracotsios, W.E. Stewart, P. Glarborg, C. Wang, O. Adigun, CHEMKIN collection, release 3.6, reaction design, inc., Technical Report, 2000.
[43] L.R. Petzold, A description of DASSL: a differential/algebraic system solver, Technical Report SAND82-8637, Sandia National Laboratories, Livermore, CA, 1982.
[44] M.P.B. Musculus, P.C. Miles, L.M. Pickett, Conceptual models for partially premixed low-temperature diesel combustion, Prog. Energy Combust. Sci. 39 (2013) 246–283.
[45] G. Bader, P. Deuflhard, A semi-implicit mid-point rule for stiff systems of ordinary differential equations, Numer. Math. 41 (1983) 373–398.
[46] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical recipes: the art of scientific computing, third edition (C++), Cambridge University Press, UK, 2007.