OpenSMOKE++: An object-oriented framework for the numerical modeling of reactive systems with detailed kinetic mechanisms

A. Cuoci *, A. Frassoldati, T. Faravelli, E. Ranzi

Department of Chemistry, Materials, and Chemical Engineering "G. Natta", P.zza Leonardo da Vinci 32, 20133 Milano, Italy

Received 1 September 2014
Received in revised form
19 December 2014
Accepted 17 February 2015
Available online 27 February 2015

1. Introduction

Nowadays it is well recognized that realistic numerical simulations of combustion phenomena necessarily require not only accurate, detailed modeling of fluid dynamic aspects, but also a detailed characterization of the chemical reactions and the physical and chemical properties of the gas mixture [1,2]. In particular, the inaccuracy of simplistic approaches assuming either equilibrium chemistry or global mechanisms has been clearly demonstrated in the modeling of reacting flows. As a consequence, this has led to an increasing effort to develop and incorporate more complex reaction mechanisms in the numerical simulation of combustion processes [3,4]. The result is the availability of a huge number of kinetic mechanisms with different levels of detail and comprehensiveness. Because of the hierarchical nature of the combustion process of hydrocarbons, the larger the fuel molecule (i.e. the number of carbon atoms), the more species and reactions are required to model and describe its combustion. Sufficiently realistic and comprehensive kinetic mechanisms usually consist of thousands reactions and species [5–8]. Obviously, the number of species and reactions increases with the complexity and the size of the fuel molecule. As an example, if the largest kinetic mechanism for n-heptane involves ∼600 species and ∼3000 reactions [9], the oxidation of methyl-decanoate, a methyl-ester used as a biodiesel surrogate, is described by a mechanism with ∼3000 species and ∼9000 reactions [5]. As a matter of fact, the most recent reaction mechanisms for n-alkanes and 2-methyl-alkanes involve more than 7000 species and more than 30,000 elementary reactions [7]. In particular, in Fig. 1a we reported the number of reactions as a functions of the number of species for several detailed kinetic mechanisms freely available on the web in CHEMKIN® format. The mechanisms reported in Fig. 1 are grouped in different families, according to the research group involved in their development:

- POLIMI: CRECK Modeling Group at Politecnico di Milano (Italy) [10];
- LLNL: Lawrence Livermore National Laboratories (USA) [11];
- KinCom (Kinetics of Combustion): Université de Lorraine (France) [12];
- C3 (Combustion Chemistry Center): National University of Ireland, Galway [13];
- GRI: mechanisms related to the GRI-Mech Project [14];
- Others: mechanisms developed by other groups.

The Supplemental material provides the complete list of all the mechanisms summarized in Fig. 1 (Appendix D), together with the bibliographic references. With the exception of schemes belonging to the POLIMI class, the number of reactions increases less than linearly with the number of species, i.e. $N_R \approx 16 N_S^{0.80}$. POLIMI kinetic mechanisms show a different trend, in which there is an approximately linear correlation between reactions and species, $N_R \approx 26 N_S$. The difference is due to the different features of POLIMI mechanisms (based on the lumping technique [15]), resulting in a number of reactions per species which is usually much larger than in conventional mechanisms. Indeed, following the approach already described by Ranzi et al. [16], the H-abstraction reactions on the fuel are systematically considered, taking into account all the different H-abstracting radicals and the number and type of H-atoms in the fuel molecule. This is particular evident from the analysis of Fig. 1b, where the ratio for the same schemes is reported. While for conventional schemes the mean number of reactions per species is between 4 and 6, for the lumped POLIMI mechanisms this number is usually between 30 and 35. As extensively discussed in Section 6, the high number of reactions of lumped kinetic mechanisms has a non negligible impact on the performances of numerical algo-rithms for the simulation of combustion processes. Clearly the size of mechanisms tends to grow with time, thanks to new discoveries in chemical kinetics. Recent trends suggest that in the near future mechanisms with more than ∼10,000 species will be available, es-pecially in order to describe combustion of biofuels.

While such large mechanisms may provide very detailed information about the chemical activities, it is very expensive (in terms of CPU time) to accommodate them in numerical simulations. The computational cost of simulations can be prohibitive, even when ideal reactors (batch, plug-flow, perfectly-stirred reactors, etc.) or one-dimensional laminar flames (flat premixed flames, counterflow diffusion flames, etc.) are numerically modeled. As a consequence, the numerical simulation of most practical combustion systems of industrial interest is practically impossible using detailed kinetic mechanisms, even on massively distributed-memory machines. The substantial reduction of these large mechanisms (i.e. the elimination species and reactions which are unimportant in the specific conditions under investigation) is probably the only viable technique to make them applicable in the near future [17]. Even if the size (i.e. number of species and reactions) of detailed kinetic mechanisms is the most important challenge for the numerical simulations, also stiffness of the non-linear chemical equations governing the evolution of combustion processes plays a fundamental role in controlling the performance and the robustness of numerical algorithms [18]. The stiffness, which is related to the existence of a wide range of characteristic chemical times, increases with increasing the complexity of the fuel molecule, because of the larger number of radical species in quasi-steady state, fast reversible reactions in partial equilibrium, and isomerization processes [19]. As a consequence, in order to apply detailed kinetic mechanisms in complex reacting flows, the stiffness reduction appears as an important step, which is usually performed on-the-fly and involves expensive calculations, like operations on the Jacobian matrix or sensitivity analysis [4].

The huge number of species and reactions in detailed kinetic mechanisms makes it very difficult and complicated to recognize the main reaction paths and to interpret the numerical results from a chemical point of view [20]. This difficulty can be usually overcome by making use of reaction path analysis (RPA) and sensitivity

☆ * Corresponding author.
E-mail address: alberto.cuoci@polimi.it (A. Cuoci).
URL: http://creckmodeling.chem.polimi.it (A. Cuoci).

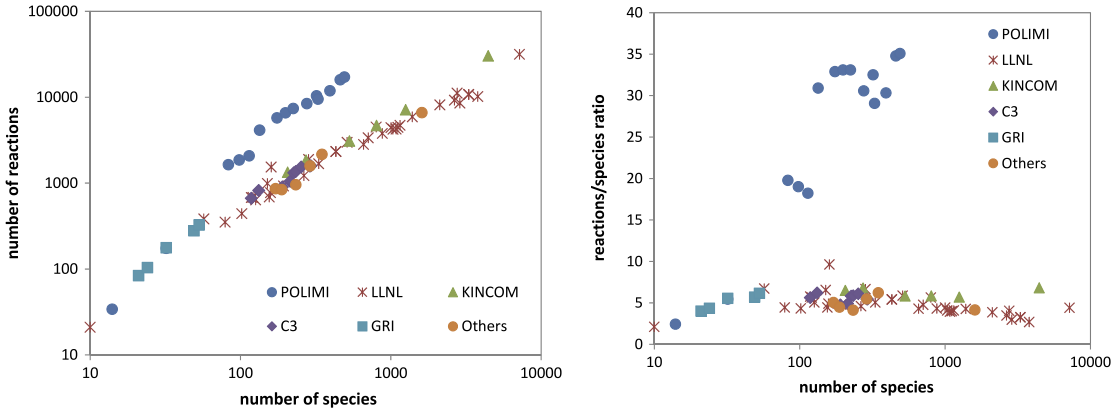

Fig. 1. Size (left) and reactions/species ratios (right) of skeletal and detailed kinetic mechanisms for hydrocarbon fuels studied in the present work. Each point refers to a different kinetic mechanism. The complete list is reported in the Supplemental material (Appendix D).
Source: Adapted from Lu and Law [2].

analysis (SA). The first technique consists in determining the contribution of each reaction to the net production or destruction rates of a species, while the sensitivity analysis allows for the quantitative understanding of how the numerical solution depends on the various reactions in the kinetic mechanism. However, both the numerical techniques are computationally expensive and produce a large number of data which require careful post-processing to be correctly interpreted.

The considerations above reported suggest the need of computational tools able to manage large kinetic mechanisms, with thousands of species and reactions. Two main features are strictly required:

1. the calculations have to be very efficient, i.e. proper numerical technique have to be adopted to reduce the computational cost, without loss of accuracy;
2. the computational tools have to be very user-friendly, i.e. easy to use and to be modified by the user, in order to incorporate them in new or existing numerical codes, to perform also complex operations (reduction of kinetic mechanisms, kinetic analyses, etc.).

This paper describes a suite of numerical tools, known as OpenSMOKE++, designed to provide a flexible, extensible framework for simulations involving thermodynamics, transport processes, and chemical kinetics. The proposed framework is specifically conceived to manage very large detailed kinetic mechanisms, involving thousands of species and reactions. The OpenSMOKE++ framework strongly relies on the so-called Computation Cost Minimization (CCM) techniques [2], i.e. a set of strategies that expedite simulations with little or no accuracy loss through optimization of the computation sequence. Moreover, thanks to its very user-friendly interfaces, based on the most modern features of C++, the OpenSMOKE++ framework can be easily incorporated into new or existing numerical codes to efficiently calculate thermodynamic properties, transport properties, and homogeneous and heterogeneous kinetics rates, and to solve the typical ODE (Ordinary Differential Equations) systems describing combustion processes. The most important features and the current capabilities of OpenSMOKE++, together with benchmark tests and illustrative examples, are presented and discussed in the present paper, which is organized as reported in the following. Section 2 describes the most important technical features of the OpenSMOKE++ framework, together with its innovative aspects with respect to similar frameworks existing in the literature and commonly adopted by the combustion community. In Section 3 the most important computational capabilities of OpenSMOKE++ are presented. Then, in Sections 4 and 5, the OpenSMOKE++ Suite is presented, i.e. a collection of tools to perform numerical simulations of ideal reactors and kinetic analyses, like sensitivity analysis or rate of production analysis. Section 6 is devoted to the analysis of the numerical performances of the proposed framework on a series of examples of interest, in order to show its efficiency and robustness. Then, in Section 7, a series of typical illustrative examples is presented, to better demonstrate the generality and the potential applications of the proposed framework.

Even if the OpenSMOKE++ framework was already applied with satisfactory results for the simulation of a wide range of reacting systems (ideal reactors [21], laminar premixed [4] and counter-flow diffusion flames [22], evaporation and combustion of droplets in microgravity conditions [23], laminar coflow diffusion flames [24,25], catalytic heterogeneous reactors [26,27], reactor networks [28], and internal combustion engines [29]), this is the first time we give a detailed description of its features, the numerical models available, and the rationale behind its design.

2. The OpenSMOKE++ framework

OpenSMOKE++ is a framework specifically conceived to manage large, detailed kinetic mechanisms and perform numerical simulations of combustion processes. The OpenSMOKE++ framework is written in object-oriented C++ and provides classes represent-ing the components of a simulation, gas mixtures, reactors, ki-netics models, equations of state, ODE integrators, reaction path diagrams, and so on. The numerical models are assembled in a physical, intuitive way, by creating and combining the different classes. As better explained below, this modular approach has im-portant advantages in the usability and the extension of the code. OpenSMOKE++ is specifically conceived to handle problems with thousands of species and reactions, which can be imported from the standard CHEMKIN© format [30]. Efficient algorithms are used to evaluate reaction and formation rates.

2.1. The object-oriented features

In most cases the numerical codes for the simulation of reacting flows have been written in procedural languages. The large use of techniques based on procedural programming has led to a concentration on the lower levels of the coding and the implementation of models (in many cases very complicated) has been usually discussed in terms of the manipulation of individual floating-point values [31].

The OpenSMOKE++ framework is based on a different methodology, which is called Object-Oriented Programming (OOP), which is today generally recognized as able to have code that is easier to write, validate, and maintain than procedural techniques [32]. The

feasibility of OOP techniques for writing numerical codes for the solution of reacting flow problems with detailed chemistry will be described and discussed.

The OOP approach involves three main features: abstraction, inheritance, and polymorphism [32]. *Abstraction* is the ability to represent conceptual constructs in the program and to hide details behind an interface. This is achieved by the concept of classes to represent conceptual objects in the code, that encapsulate (i.e. contain and protect) the data that make up the object. Member functions are provided that permit limited, well-defined access to the encapsulated data. Thus, as an example, it is possible to create data types representing chemical species, chemical reactions, a mixture of chemical species, a gas stream, an ideal reactor, etc., hiding the numerical details of the implementation by encapsulation. The class interface is designed to be as simple as possible, while the implementation is not relevant at this level. *Inheritance* enables relationships between the various classes to be expressed, representing commonality between different classes of objects by class extension. By doing this, existing classes can be given new behavior without the necessity of modifying the existing class. For example, the different kind of reactions have some elements in common (e.g. the functions to manage the stoichiometry) which can be inherited by a parent reaction class and specialized for the specific reactions (children) which the user wants to implement. This approach can be used to construct a complicated class by extending base classes that express simpler objects. *Polymorphism* is the ability to provide the same interface to objects with different implementations, thus representing a conceptual equivalence between classes that in practical terms have to be coded differently. Examples of this feature in `OpenSMOKE++` include the implementation of the `Mixture` class (as better explained in the following).

The `OpenSMOKE++` library is written in C++, since this is a good programming language for scientific work. It is widely available on all platforms and, being based on C, is fast. Several studies [33,34] indicate no significant difference in performance between Fortran and the C group of languages.

The `OpenSMOKE++` library is based on *template programming* and strongly relies on the concept of policies and policy classes [35], an important class design technique that enables the creation of flexible, highly reusable libraries. In brief, policy-based class design fosters assembling a class with complex behavior out of many little classes (called *policies*), each of which takes care of only one behavioral or structural aspect. As the name suggests, a policy establishes an interface pertaining to a specific issue. The user can implement policies in various ways as long as the policy interface is respected. Since policies can be mixed and matched, a combinatorial set of behaviors can be achieved by using a small core of elementary components.

2.2. The *OpenSMOKE++* kernel

`OpenSMOKE++` consists of a "kernel" that provides the core capabilities for its integration in existing and new numerical codes and interface packages to external numerical libraries, to solve for example ODE and DAE problems. This kernel is conceived by taking into account that one of the main objective of the `OpenSMOKE++` framework is to make the management of complex reacting mixtures easy. For this purpose the main classes are designed by exploiting policy design and inheritance techniques.

The real core of the kernel is the `Species` class, which manages the properties of a single chemical species. The `Species` class is based on several policies concerning the sub-models used to evaluate the thermodynamic (density, specific heats, enthalpy, entropy, etc.) and transport properties (thermal conductivity, viscosity, mass diffusivity, etc.). The user can easily choose the policies to be used which better fit his needs, i.e. the sub-models to calculate the required properties, or can easily implement a new policy. The only requirement is that the new policy respects the policy interface. The main advantage of this approach is that, if a new policy is needed, no changes to the existing code are needed. As an example, if a new equation of state is needed for evaluating the density, the new corresponding policy can be added to the `OpenSMOKE++` library, but the source code of the original `OpenSMOKE++` library does not have to be changed.

The chemical species described by the `Species` classes can be collected together to create the `Mixture` class, through which the user can easily and efficiently evaluate thermodynamic and transport properties of a mixture. The `Mixture` class is built using a number of policies concerning the mixing rules for the evaluation of mixture properties (e.g. mean specific heat at constant pressure, thermal conductivity, etc.). The `Mixture` class does not need to know the details behind the calculations of properties of every species, but only how to "mix" these properties to evaluate the corresponding mean mixture values. This is easily achieved because the different species involved in the mixture expose a common interface to the `Mixture` class. Moreover, the user can access the properties of single species of the mixture, without knowing what kind of species is being considered, because the exposed interface is always the same, independently of the policy. More importantly, since policies are implemented on the basis of template programming, there is no CPU-time penalty in this approach.

The chemical reactions are described through the `Reaction` class, which manages the data about the stoichiometry and the kinetic parameters of a single reaction. The `Reaction` class is designed as a virtual class, from which a large number of classes is derived, each of them describing a particular kind of reaction (fall-off reactions, Chebyshev, bimolecular activated reactions, etc.).

The `Chemistry` class represents a collection of reactions involved in a kinetic mechanism. This class does not have to know the details about the implementation of every `Reaction` type class. The most important functions which have to be exposed to the `Chemistry` class are the evaluation of the reaction rate and the access to the stoichiometric coefficients. The `Chemistry` class provides all the functions to evaluate the formation rates of every chemical species, which are usually the most important data required in the simulation of reacting flows.

It is quite easy to recognize that the `OpenSMOKE++` C++ Library can be easily extended and adapted to the needs of the user, by adding new thermodynamic and/or transport policies and by introducing new classes of chemical reactions.

2.3. Efficiency

According to Smooke et al. [36], in typical reacting flow calculations with detailed kinetic mechanisms, more than ~80%of CPU time is spent for the numerical evaluation of Jacobian matrices. Since the construction of the Jacobian usually involves the evaluation of thermodynamic and transport properties and the calculation of formation rates of every species a large number of times, the `OpenSMOKE++` library tries to make the calculations of these properties as fast as possible. This goal can be reached by exploiting the object oriented nature of C++. In particular, the `MixtureMap` class is specifically conceived for calculating thermodynamic, transport properties and kinetics data as fast as possible, without using complex interfaces. The idea is quite simple: from every `Mixture` object (working as a sort of C++ Factory [35]) a number of independent `MixtureMap` objects can be created. The user can register in each of these `MixtureMap` objects the properties which have to be calculated. Then, when these properties are needed, the user has only to update the status of the `MixtureMap` object and ask for the property in which he is interested. The details about how calculations are performed

are completely hidden to the user, who does not need to recur to complex interfaces in which a long number of parameters have to be passed or to remember the exact order in which the different functions must be called. This is possible because of the object oriented programming techniques. Moreover the `MixtureMap` objects know the kind of mixture from which they were originated and the `Mixture` object knows every map that it created. In this way if a change is applied to the `Mixture` object (e.g. a new reaction is added, etc.), such changes are automatically applied to all the maps created by the `Mixture` object.

Efficiency is achieved by the `MixtureMap` using several techniques:

- caching: the code is written in order to cache as much as possible, which means storing items for future use in order to avoid retrieving or recalculating them. Only calculations which are strictly necessary are performed "on the fly". If some variables can be calculated only once, they are stored so that they are available for future needs;
- object pools: this is a technique for avoiding the creation and deletion of a large number of objects during the code execution [37]. If the user knows that his code needs a large number of short-lived objects of the same type, he creates a pool of those objects. Whenever he needs an object in his code, he asks the pool for one. When the user is done with the object, he returns it to the pool. The object pool creates the objects only once, so their constructor is called only once, not each time they are used. Thus, object pools are appropriate when the constructor performs some setup actions that apply to many uses of the object, and when the user can set instance-specific parameters on the object through non-constructor method calls. For example, if the user need to solve several ODE systems with different initial conditions, it is possible create a pool of ODE objects, in order to reduce the CPU time for allocating memory, setting the numerical parameters, etc.;
- optimized functions: the numerical algorithms are often reformulated in order to exploit the Intel® MKL Vector Mathematical Functions Library (VML) [38]. VML includes a set of highly optimized functions (arithmetic, power, trigonometric, exponential, hyperbolic, special, and rounding) that operate on vectors of real and complex numbers;
- code reformulation: many parts of the numerical algorithms are reformulated in a less intuitive way in order to minimize the number of flops needed to perform some calculations or to avoid the usage of CPU-expensive functions [2]. As an example, the kinetic constants are not evaluated using the usual direct approach: $k_j = A_j T^{\beta_j} \exp\left(-E_j/RT\right)$. On the contrary, they are calculated using the following formulation: $k_j = \exp\left(\ln\left(A_j\right) + \beta_j - E_j/RT\right)$. The first formulation requires two expensive functions for the calculation of the kinetic constant: a power and an exponential. The second formulation only one expensive exponential function. The $\ln\left(A_j\right)$ coefficient must be evaluated only once and stored and it does not need to be re-evaluated every time. It is quite clear that by using this simple reformulation, the user can save CPU time, considering that detailed kinetic mechanisms involves thousands of reactions.

Moreover, the `OpenSMOKE++` functions always try to handle objects efficiently, applying inline methods and using only pass-by-reference and return-by-reference techniques, without over-using costly language features, such as exceptions, virtual methods and RTTI (Run Time Type Information).

2.4. Integration in new and existing codes

The object-oriented nature of `OpenSMOKE++` library makes its integration in existing numerical codes quite easy. Because of encapsulation of data, the user does not have to know the details about the algorithms internally used by `OpenSMOKE++` and, most importantly, the interface for calling common functions (for evaluation of formation rates, transport properties, etc.) is kept as simple as possible. An example is reported in the following C++ lines of code:

```
1. Mixture *mix = new Mixture(fileName);
2. MixtureMap *map = new MixtureMap(mix);
3. map->SetTemperature(T);
4. map->SetPressure(P);
5. cp_species = map->cp_species();
6. viscosity_mixture = map->viscosity_mixture(x);
7. R = map->formation_rates(c);
8. r = map->reaction_rates(c);
```

In the first line the kinetic mechanism (together the thermodynamic data and the transport properties) is imported from an external file (usually in `CHEMKIN®` format). Then (line 2) a MixtureMap object is created, from the mixture previously imported. Lines 3 and 4 are used to set the temperature and the pressure at which the thermodynamic, transport and kinetic properties have to be calculated. In line 5 the constant specific heats of the species contained in the kinetic mechanism are calculated. In line 6 the dynamic viscosity of the mixture is evaluated. In this case, the composition is provided in terms of mole fractions (`x`). The following point is important to remark. In order to calculate the viscosity of the mixture, the viscosities of the single species have to be firstly evaluated. However, as in this case, the user does not need to calculate them explicitly, because, thanks to the object-oriented nature of the framework, the `MixtureMap` recognizes that the viscosities of single species have to be calculated before returning the viscosity of the mixture. More interestingly, if the user asks for the viscosity of the mixture for a different composition, without changing the temperature and the pressure, the `MixtureMap` class performs only the operations strictly needed. This means that if the viscosities of single species depends only on temperature and pressure, they are not recalculated. So, the big advantage of object-oriented programming in this kind of context is quite evident: the user does not need to know the exact order in which to call the functions to obtain the required data. Moreover, only the strictly necessary calculations are automatically performed, which means that the risk to perform useless calculations is minimized.

These advantages are also evident from lines 7 and 8. In particular, in line 7 the formation rates of all the species are calculated, according to the specified concentrations. Then, in line 8, the user asks for the reaction rates. Apparently, the two lines seem conceptually wrong: in order to calculate the formation rates, the reaction rates for all the reactions are needed, but in this example the order in which these two operations are performed is the opposite. However, what happens is completely hidden to the user: in line 7 the `MixtureMap` object recognizes that the reactions rates are needed and therefore they are calculated before returning the formation rates. In line 8, the reaction rates are not re-calculated, by they are simply returned, since the `MixtureMap` object knows that they were just calculated. If the two lines were switched, something different would happen, leading to the same result:

```
7*. r = map->reaction_rates(c); // reaction rates
8*. R = map->formation_rates(c); // formation rates
```

Now in line 7* the reaction rates are not simply returned, but they have to be calculated. In line 8* the formation rates are assembled from the reaction rates evaluated above, which of course are not recalculated. This is a further example that demonstrates the advantages of object-oriented programming over the procedural programming.

3. Main features

This section briefly describes the computational capabilities of OpenSMOKE++.

3.1. Thermodynamic and transport properties

The thermodynamic properties of single chemical species are calculated following the approach proposed by Gordon and McBride [39]. For each species, the dimensionless properties at constant pressure are specified as functions of temperature as follows:

$$\frac{\widetilde{C}_{P,i}}{R} = a_{i,1} + a_{i,2}T + a_{i,3}T^2 + a_{i,4}T^3 + a_{i,5}T^4 \tag{1}$$

$$\frac{\widetilde{H}_i}{RT} = a_{i,1} + \frac{a_{i,2}}{2}T + \frac{a_{i,3}}{3}T^2 + \frac{a_{i,4}}{4}T^3 + \frac{a_{i,5}}{5}T^4 + \frac{a_{i,6}}{T} \tag{2}$$

$$\frac{\widetilde{S}_i^0}{R} = a_{i,1}\ln T + a_{i,2}T + \frac{a_{i,3}}{2}T^2 + \frac{a_{i,4}}{3}T^3 + \frac{a_{i,5}}{4}T^4 + a_{i,7} \tag{3}$$

where $\widetilde{C}_{P,i}$, $\widetilde{H}_i$, and $\widetilde{S}_i^0$ are the molar specific heat at constant pressure, the molar specific enthalpy and the molar specific entropy at 1 atm of species ith, respectively. T is the temperature, R the universal gas constant, and $a_{i,1} - a_{i,7}$ are the least-squares coefficients of the Gordon and McBride empirical equations [39]. For each species, two sets of coefficients for use on two adjacent temperature intervals are included, resulting in an overall number of parameters equal to 14.

The thermodynamic properties of the mixture are evaluated by applying the Gibbs theorem, which consists in summing up the contributions made by all species. For example, the specific molar enthalpy of the mixture is given by $\widetilde{H} = \sum_{i=1}^{N_S} x_i\widetilde{H}_i$, where x_i is mole fraction of species ith.

The transport properties of the species are computed by using the standard kinetic theory expressions [40,41]. To expedite the evaluation of transport properties in OpenSMOKE++, the temperature dependent parts of the pure species property expressions are fitted. This means that, rather than re-evaluating the complex expressions for the properties, only simple fits need to be evaluated. In particular, following the approach proposed by Kee et al. [30], a polynomial fit of the logarithm of the property versus the logarithm of the temperature is adopted:

$$\ln\eta_i = \sum_{k=1}^{N} b_{i,k}^{\eta}(\ln T)^{k-1} \tag{4}$$

$$\ln\lambda_i = \sum_{k=1}^{N} b_{i,k}^{\lambda}(\ln T)^{k-1} \tag{5}$$

$$\ln\Gamma_{i,j}^0 = \sum_{k=1}^{N} b_{i,j,k}^{\Gamma}(\ln T)^{k-1} \tag{6}$$

where η_i and λ_i are the dynamic viscosity and the thermal conductivity of species ith, respectively. $\Gamma_{i,j}^0$ is the binary mass diffusion coefficient between species ith and jth evaluated at the pressure of 1 bar. The coefficients $b_{i,k}^{\eta}$, $b_{i,k}^{\lambda}$, and $b_{i,j,k}^{\Gamma}$ are the fitting parameters for viscosity, thermal conductivity and mass diffusion, respectively.

OpenSMOKE++ uses third-order polynomial fits (i.e., $N = 4$), as suggested in [30], where it is reported that the average error is well within one percent. Obviously, the fitting procedure has to be carried out for the particular gaseous mixture under investigation, which means that the fitting cannot be done "once and for all" but must be done once at the beginning of each new problem. Moreover, while the viscosities and conductivities of species do not depend on pressure, the binary diffusion coefficients inversely depend on pressure. Eq. (6) refers to unit pressure of 1 bar and therefore the real evaluation of a binary diffusion coefficient $\Gamma_{i,j}$ at pressure p (in bar) is given by $\Gamma_{i,j} = \Gamma_{i,j}^0/p$.

The thermal diffusion ratios are estimated using the simplified procedure described in [30]. Since the dependence on temperature is weaker if compared to the previous transport properties, the fitting procedure is directly applied on the temperature, rather than on its logarithm.

The mixture diffusion coefficient $\Gamma_{i,mix}$ for species ith is calculated using the following expression [42]:

$$\Gamma_{i,mix} = \frac{\sum_{j\neq i}^{N_S} x_j W_j}{W_{mix}\sum_{j\neq i}^{N_S}\frac{x_j}{\Gamma_{ji}}}. \tag{7}$$

The remaining mixture-averaged transport properties are then estimated from the corresponding pure species properties through the application of proper mixing rules. OpenSMOKE++ employs the Wilke formula [43] to calculate the dynamic viscosity:

$$\eta = \sum_{i=1}^{N_S}\frac{x_i\eta_i}{\sum_{j=1}^{N_S} x_j\phi_{i,j}} \tag{8}$$

where:

$$\phi_{i,j} = \frac{1}{\sqrt{8}}\sqrt{\frac{W_j}{W_i + W_j}}\left[1 + \sqrt{\frac{\eta_i}{\eta_j}}\left(\frac{W_j}{W_i}\right)^{1/4}\right]^2 \tag{9}$$

and W is the molecular weight of species ith. For the thermal conductivity the combination averaging formula proposed by Mathur et al. [44] is adopted:

$$\lambda = \frac{1}{2}\left[\sum_{i=1}^{N_S} x_i\lambda_i + \left(\sum_{i=1}^{N_S}\frac{x_i}{\lambda_i}\right)^{-1}\right]. \tag{10}$$

3.2. Kinetic mechanisms

OpenSMOKE++ is able to manage any number of elementary chemical reactions, both reversible and irreversible. For a reaction mechanism involving N_R elementary reactions among N species, the jth elementary step can be written in the following general form:

$$\sum_{i=1}^{N_S}\upsilon_{ij}^f S_i \underset{k_j^b}{\overset{k_j^f}{\leftrightarrow}} \sum_{i=1}^{N_S}\upsilon_{ij}^b S_i \quad j = 1,\ldots,N_R \tag{11}$$

where S_i is the symbol for species ith, υ_{ij}^f is the stoichiometric coefficient of reactant species ith in reaction jth, υ_{ij}^b the stoichiometric coefficient of product species ith in reaction jth. k_j^f and k_j^b are respectively the forward and the reverse kinetic constants of reaction jth. In most cases the forward kinetic constant is given by the Arrhenius' law:

$$k_j^f = A_j T^{n_j} e^{-\frac{E_j}{RT}} \tag{12}$$

where A_j is the pre-exponential factor, n_j the temperature exponent, E_j the activation energy, R the universal gas constant

and T the absolute temperature. For a reversible elementary reaction, the reverse kinetic constant k_j^b can be explicitly specified by the user or calculated within the code through the equilibrium constant K_j^c:

$$K_j^c = e^{\left(\frac{\widetilde{\Delta S}_i^0}{R} - \frac{\widetilde{\Delta H}_i}{RT}\right)} \left(\frac{p_{\text{atm}}}{RT}\right)^{\sum_{i=1}^{N_S}\left(\upsilon_{i,j}^b - \upsilon_{i,j}^f\right)} \tag{13}$$

$$k_j^b = \frac{k_j^f}{K_j^c} \tag{14}$$

where the reaction entropy $\widetilde{\Delta S}_j^0$ and the reaction enthalpy $\widetilde{\Delta H}_j$ are defined as:

$$\widetilde{\Delta S}_j^0 = \sum_{i=1}^{N_S}\left(\upsilon_{i,j}^b - \upsilon_{i,j}^f\right)\widetilde{S}_i^0 \tag{15}$$

$$\widetilde{\Delta H}_j = \sum_{i=1}^{N_S}\left(\upsilon_{i,j}^b - \upsilon_{i,j}^f\right)\widetilde{H}_i. \tag{16}$$

The net reaction rate r_j of reaction jth is given by the difference between the forward and the backward (if any) reaction rates, according to the following plain law of mass action:

$$r_j = k_j^f \prod_{i=1}^{N_S} c^{\upsilon_{ij}^f} - k_j^b \prod_{i=1}^{N_S} c^{\upsilon_{ij}^b}. \tag{17}$$

In some reactions, like dissociation or recombination reactions, a "third body" is required for the reaction to proceed. In these cases the presence of other species than those directly participating in the reaction stoichiometry enhances the reaction rate according to an 'effective' concentration, or molecularity, $M_{eff,j}$:

$$M_{eff,j} = \sum_{i=1}^{N_S}\left(\alpha_{ij} c_i\right) \tag{18}$$

where α is the third-body efficiency of species ith in reaction jth. The resulting, effective reaction rate is then given by $r_j^{eff} = r_j M_{eff,j}$.

OpenSMOKE++ is also able to manage more complex kinetic laws describing the chemical reactions (Pressure-dependent reactions, Chemically Activated Bimolecular Reactions, Chebyshev Polynomials, etc.). The interested reader is referred to A, where detailed descriptions are provided.

3.3. ODE Solvers for stiff problems

The time integration of reacting gas mixtures is the basis of many computational approaches in numerical combustion [18]. Examples include the simulation of ideal reactors (perfectly stirred reactors, plug-flow reactors, shock-tubes, batch reactors, etc.) and the point-wise time integration of the chemical source terms as part of the operator-splitting strategy for the solution of reactive Navier–Stokes equations [24]. Thus, one shall assume that the reacting gas mixture is governed by the following system of coupled, first-order ODEs (Ordinary Differential Equations):

$$\begin{cases} \dfrac{d\mathbf{y}}{d\xi} = \mathbf{f}(\mathbf{y}, \xi) \\ \mathbf{y}(\xi_0) = \mathbf{y}_0. \end{cases} \tag{19}$$

In the equation reported above, $\mathbf{y}$ is the vector of unknowns (species concentrations, temperature, pressure, etc., depending on the specific problem under investigation), ξ is the independent variable (usually the time or a spatial coordinate) and $\mathbf{f}(\mathbf{y}, \xi)$ is a non-linear function of the unknowns. The ODE system (19) is usually stiff [45], since the evolution of chemical species in most cases occurs on a wide range of characteristic times. Moreover, in case of large kinetic mechanisms, the number of equations to be solved can be very large.

The most popular methods for the time integration of stiff reaction mechanisms are based on variable-coefficient backward differentiation formula (BDF) methods [46]. Those techniques are based on modified Newton's methods and are computationally quite expensive, since they require repeated solutions of linear systems involving the Jacobian matrix associated to the system (19). Due to the importance of ODE systems in kinetic analysis of combustion phenomena, the OpenSMOKE++ framework provides the interface to several well-known numerical libraries for the solution of such ODE systems. In particular, solvers for stiff or moderately stiff problems are available, because the best (i.e. more efficient and more robust) solver depends on the intrinsic features of the kinetic mechanism under investigation.

The current version of OpenSMOKE++ provides the interface to the following ODE solvers:

- CVODE [47]: a solver for stiff and non-stiff problems. The methods used in CVODE are variable-order, variable-step multistep methods. For non-stiff problems, CVODE includes the Adams–Moulton formulas, with the order varying between 1 and 12. For stiff problems, CVODE includes the BDFs in so-called fixed-leading coefficient form, with order varying between 1 and 5. For either choice of formula, the resulting nonlinear system is solved (approximately) at each integration step;
- DASPK [48]: a solver for systems of differential–algebraic equations. It includes options for both direct and iterative (Krylov) methods for the solution of the linear systems arising at each (implicit) time step. However, in the present work the linear systems that arise are always solved by direct methods (LU factorization/solution).
- DLSODE [49]: a solver for stiff and non-stiff systems. It uses the Adams methods (predictor–corrector) in the non-stiff case, and the Backward Differentiation Formulas (BDF) methods in the stiff case. The linear systems that arise are solved by direct methods (LU factorization/solution);
- DLSODA [49]: a variant version of the DLSODE package. It switches automatically between stiff and non-stiff methods. This means that the user does not have to determine whether the problem is stiff or not, and the solver will automatically choose the most appropriate method. It always starts with the non-stiff method;
- DVODE [46]: a general purpose solver very similar to DLSODE [49]. However, it uses variable-coefficient methods (fixed-leading coefficient form) instead of the fixed-step-interpolate methods in DLSODE. This and other features make it often more efficient than DLSODE;
- RADAU5 [50]: a solver using an implicit Runge–Kutta method (RadauIIa) of order 5 (three stages) with step size control and continuous output.

In addition to the solvers reported above, a native OpenSMOKE++ ODE solver is also provided. Basically it is based on the same numerical approach of BzzOde [51] (belonging to the BzzMath libraries [51–53]), but all the linear algebra operations are performed through the C++ Eigen library. It is specifically conceived for very stiff problems and it is based on the BDF in the so-called fixed-leading coefficient form, with order varying between 1 and 5.

The complete list of solvers is summarized in Table 1. All the solvers are open-source and freely available for academic purposes. Since smart interfaces are available in the OpenSMOKE++ framework, the user can very easily switch from one solver to another.

Table 1
List of stiff ODE solvers available in the current version of OpenSMOKE++.

Name	Integration method	Order	Linear system	Linear algebra	Ref.
CVODE	Variable-coefficient BDF	Variable	Direct, LU Factorization	BLAS/LAPACK (Intel® MKL)	[47]
DASPK	Variable-coefficient BDF	Variable	Direct, LU Factorization	BLAS/LAPACK (Intel® MKL)	[48]
DLSODA	Fixed-coefficient BDF	Variable	Direct, Gauss Factorization	LINPACK	[49]
DLSODE	Fixed-coefficient BDF	Variable	Direct, Gauss Factorization	LINPACK	[49]
DVODE	Variable-coefficient BDF	Variable	Direct, LU Factorization	BLAS/LAPACK (Intel® MKL)	[46]
RADAU5	Implicit Runge–Kutta	Variable	Direct, Gauss Factorization	LINPACK	[50]
Native	Fixed-coefficient BDF	Variable	Direct, LU Factorization	Eigen (Intel® MKL)	Present work

The current version of OpenSMOKE++ assumes that the Jacobian matrix corresponding to the ODE system is dense, in order to be as general as possible. However, the formation rates of most chemical species are usually dominated by only a small set of reactions. Reflecting this physical behavior, most large chemical kinetic mechanisms in the literature are extremely sparse in terms of interactions between different species. In other words, each species reacts with a small number of the total number of species available in the mixture. Dense techniques adopted by the current version OpenSMOKE++ cannot exploit this inherent sparsity. Several authors demonstrated that, after a proper reformulation of the ODE problem (19), in some cases (for example batch reactors with constant volume) it is possible to take advantage of the sparsity and to adopt ODE solvers specifically conceived for systems with sparse Jacobian matrices, with a significant gain in the computational time [18,54–56]. As a consequence, our near-term development plans do necessarily focus on the development of numerical tools to exploit the sparsity of large kinetic mechanisms.

3.4. Sensitivity analysis

The OpenSMOKE++ framework provides several useful tools to perform sensitivity analysis, both for unsteady and steady-state problems. Sensitivity analysis is very important for kinetic studies, since it allows the quantitative understanding of how the numerical solution of the governing equations depends on the various parameters contained in the model itself [57,58]. Only the first-order sensitivity coefficients with respect to the reaction rate coefficients (pre-exponential factors, activation energy or kinetic constant) can be calculated. The calculation of sensitivity coefficients exploits the linearity of the differential equations governing their evolution, regardless of any non-linearities in the problem itself.

The equations for the sensitivity coefficients can be easily obtained starting from the ODE system describing the system under investigation:

$$\frac{d\mathbf{y}}{d\xi} = \mathbf{f}\left(\mathbf{y}, \xi; \boldsymbol{\alpha}\right). \tag{20}$$

In the equations reported above, we explicitly introduced the idea that the functions on the right hand side do not depend only on the unknowns $\mathbf{y}$ (size N), but also on a number of parameters $\boldsymbol{\alpha}$ (size N_P), corresponding for example to the kinetic parameters of the chemical reactions. The first-order sensitivity coefficients are then defined as:

$$s_{ij} = \frac{\partial y_i}{\partial \alpha_j} \tag{21}$$

where the index j refers to the variable and i to the parameter. For large kinetic mechanisms the number of sensitivity coefficients can be enormous. As an example, for a kinetic mechanism with $N = 1000$ species and $N_P = 10,000$ reactions the number of sensitivity coefficients of species with respect to the pre-exponential factors is equal to 10^8. In order to simplify the next equations we introduced

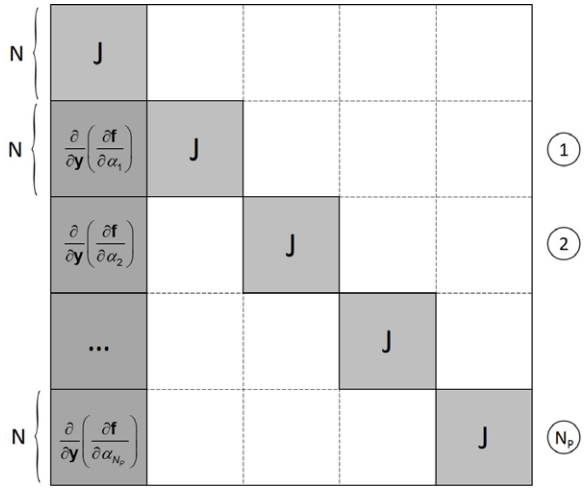

Fig. 2. Sparsity pattern of Jacobian matrix of coupled system of model equations (20) and sensitivity equations (22). N is the number of main unknowns and N_P the number of parameters. Each block is a square matrix, with size equal to $N \times N$. The shaded blocks are usually dense, while the white blocks are null matrices. The blocks on the main diagonal are equal to the Jacobian matrix $\mathbf{J}$ of the model Eqs. (20).

the following vector $\mathbf{s}_j = \left[\frac{\partial y_1}{\partial \alpha_j}, \frac{\partial y_2}{\partial \alpha_j}, \ldots, \frac{\partial y_N}{\partial \alpha_j}\right]$, i.e. the set of sensitivity coefficients of all the N variables y_i with respect to the jth parameter α_j. If we differentiate the differential equations (20) with respect to the parameters (21), we have the following N_P additional ODE systems:

$$\begin{cases} \dfrac{d\mathbf{s}_j}{d\xi} = \mathbf{J}\mathbf{s}_j + \dfrac{\partial \mathbf{f}}{\partial \alpha_j} \\ \mathbf{s}_j\left(\xi_0\right) = \mathbf{0} \end{cases} \tag{22}$$

where $\mathbf{J}$ is the Jacobian matrix of the main ODE system (20), i.e. $J_{ij} = \frac{\partial f_i}{\partial y_j}$.

There are two important points to remark about the above sensitivity equations. The first observation is that the original system (20) is not coupled to the sensitivity equations (22), and can be solved independently of the sensitivity equations, although the sensitivity equations are dependent on the original system. The second point is that the sensitivity equations (22) are linear in the sensitivity coefficients with the same Jacobian matrix employed for the state equations.

The overall system, given by the coupling of Eqs. (20) and (22), can be solved directly only if the number of parameters of interest is relatively small. This is impossible for very large kinetics, with thousands of reactions.

OpenSMOKE++ calculates the sensitivity coefficients also for very large mechanisms using a modified version of the staggered direct method [59,60], which is described in the following. Since the second term in the r.h.s. of Eqs. (22) does not depend on the sensitivity coefficient, it is quite easy to demonstrate that the structure of the Jacobian matrix associated to the overall ODE system is very sparse and block-structured, as reported in Fig. 2.

From this Figure it is not difficult to demonstrate that instead of solving the whole ODE system, one can solve N_P independent ODE systems, given by the coupling of Eqs. (20) and (22) for a specific parameter α_j. However, even exploiting this kind of decoupling, the computational cost could be prohibitively large for detailed mechanisms. A further simplification is performed. In particular, the sensitivity equations (22) are solved separately from, but sequentially with, the model equations (20). This means that the model equations are advanced over the step $\Delta\xi = \xi_{n+1} - \xi_n$ independently of the sensitivity equations, without any loss of accuracy, since they do not depend on sensitivity coefficients. Then, the sensitivity equations are solved over the same step, adopting the backward Euler method:

$$\left(\mathbf{I} - \Delta\xi \mathbf{J}^{n+1}\right) \mathbf{s}_j^{n+1} = \mathbf{s}_j^{n+1} + \Delta\xi \left.\frac{\partial \mathbf{f}}{\partial \alpha_j}\right|^{n+1} \qquad j = 1, \ldots, N_P \tag{23}$$

where $\mathbf{I}$ is the identity matrix. The equations reported above correspond to N_P independent linear systems, where the unknowns are the sensitivity coefficients $\mathbf{s}_j^{n+1}$. It is interesting to observe that the matrix $\left(\mathbf{I} - \Delta\xi \mathbf{J}^{n+1}\right)$ is always the same, regardless of the specific parameter. This means that it can be factorized only once per step, regardless of the number of parameters, which is usually very large.

The numerical procedure described above is computationally very convenient, since the stiff ODE solver is adopted to solve only the model equations, while the sensitivity equations are transformed in a set of decoupled linear systems sharing the same matrix. Since the systems are decoupled, they can be also solved in parallel, either on distributed or shared memory machines (this feature is not yet available in `OpenSMOKE++`).

The main limitation of this approach is that the integration step is dictated exclusively by the model equations. This could lead to some inaccuracies in the estimation of sensitivity coefficients. Based on our experience, the relative error with respect to a fully-coupled approach is usually within 5%. As the relative values of the sensitivity coefficients, rather than their absolute values, are really important in sensitivity analysis, the approximations introduced by the proposed method can be considered acceptable. However, if more accuracy is needed, `OpenSMOKE++` allows to split the integration step $\Delta\xi$ in a number of sub-steps and to solve the sensitivity equations (23) over them in sequence. In this procedure the Jacobian matrix $\mathbf{J}$ and the $\frac{\partial \mathbf{f}}{\partial \alpha_j}$ vector are assumed to vary linearly along the whole integration step. Obviously the computational cost increases linearly with the number of sub-steps adopted.

The raw sensitivity coefficients are normalized in the form of logarithmic derivatives, in order to make them more useful for analyses and comparisons:

$$\tilde{s}_{ij} = \frac{\partial \ln y_i}{\partial \ln \alpha_j} = \frac{\alpha_j}{y_i} s_{ij}. \tag{24}$$

In some cases, the sensitivity coefficients are more useful when they are normalized by the maximum value of each dependent variable:

$$\widehat{s}_{ij} = \frac{\alpha_j}{\max y_i} s_{ij}. \tag{25}$$

This normalization is particularly useful when y_i is a mass fraction, since it avoids artificially high sensitivity coefficients in regions where the mass fractions are approaching zero, and thus affected by numerical errors.

3.5. Rate of production analysis (ROPA)

The rate of production analysis (ROPA) is another useful tool to better understand kinetic aspects in the simulation of reaction flows. Basically, the rate of production analysis determines the contribution of each reaction to the production or destruction rates of a species.

The ROPA is performed according to the procedure described in [30]. For each species i and each reaction j it is possible to define a normalized production contribution C_{ij}^p and a normalized destruction contribution C_{ij}^d:

$$C_{ij}^p = \frac{\max\left(\upsilon_{ij}^f - \upsilon_{ij}^b, 0\right) r_j}{\sum_{k=1}^{N_R} \max\left(\upsilon_{ij}^f - \upsilon_{ij}^b, 0\right) r_k} \tag{26}$$

$$C_{ij}^d = \frac{\min\left(\upsilon_{ij}^f - \upsilon_{ij}^b, 0\right) r_j}{\sum_{k=1}^{N_R} \min\left(\upsilon_{ij}^f - \upsilon_{ij}^b, 0\right) r_k}. \tag{27}$$

The normalized contributions to production and destruction sum to 1 (i.e. $\sum_{j=1}^{N_R} C_{ij}^p = 1$ and $\sum_{j=1}^{N_R} C_{ij}^d = 1$). They compare the relative importance of each reaction to the production or destruction rates of a species.

3.6. Reaction path analysis (RPA)

Reaction path analysis (RPA) in `OpenSMOKE++` is performed following the guidelines proposed by Grcar et al. [20]. Here we present only the basic ideas behind the implementation of RPA in `OpenSMOKE++`. Reaction path analysis is an accounting of the exchange of material among species in a chemically reacting system, which can be conveniently visualized by a reaction path diagram. In mathematical terms a reaction path diagram is a directed graph whose nodes are the chemical species. An edge connects two species if a reaction moves material from one to the other. The edge is drawn as an arrow from the reactant to the product. The thickness of an arrow may indicate the rate of material exchange among species. In order to avoid any ambiguity in determining the rate of exchange among species (i.e. the thickness of the arrows), `OpenSMOKE++` is based on the concept of reaction fluxes of conserved scalars.

In other words, the thickness of the arrows is evaluated according to the reaction flux of a conserved scalar through species due to reactions. The conserved scalar is an atomic element, so each reaction path analysis is specific, typically, to either carbon, hydrogen, nitrogen, or oxygen. With this in mind, it is straightforward to determine the mass flux involved in a reaction. Over the region of interest, atoms of element e move from species A to species B at the rate:

$$R(e, A, B) = \sum_{j=1}^{N_R} \int_V n_j(e, A, B)\, r_j dV \tag{28}$$

where the summation is over all reactions, and V is the whole region of space. $n_j(e, A, B)$ is the number of atoms of elements e that a single forward instance of reaction jth moves from A to B. The magnitude of $R(e, A, B)$ determines the width of the edge between species A and B. The sign determines the direction of the arrow: if positive then $A \rightarrow B$, if negative $B \rightarrow A$. The conserved scalar approach gives reaction path diagrams the following properties:

1. the amount of material removed from the species at the base of any path equals the amount contributed to the species at the head;
2. the sum of the thicknesses of all paths into a species equals the sum of the thicknesses of all paths going out.

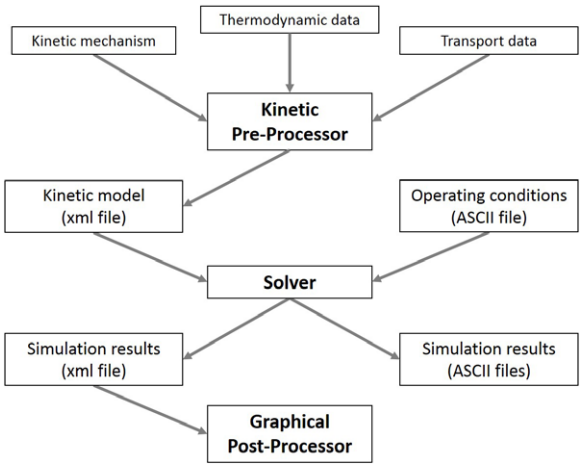

Fig. 3. Structure of the OpenSMOKE++ Suite.

4. The OpenSMOKE++ suite

Being open-source, the OpenSMOKE++ framework offers users complete freedom to customize and extend existing functionalities, to add new classes and models, and to create specific solvers for reacting systems. Since it follows a highly modular code design with sub-libraries devoted to specific functionalities (e.g. thermodynamic models, kinetic models, mathematical functions, numerical methods, physical models, etc.) the customization process is quite easy, without requiring a deep knowledge of the whole framework.

The current version of OpenSMOKE++ is already distributed with a collection of "standard solvers" for solving very common problems, which are typically met when chemical kinetic analyses have to be performed. Here the word "solver" has to be intended as an independent program, built with the aim to perform a specific task (for example to simulate a batch reactor, or to model a shock-wave, etc.). In the following, the OpenSMOKE++ Suite definition will be used to refer to the collection of OpenSMOKE++ standard solvers. The list of available solvers (which is continuously growing) includes:

1. a kinetic pre-processor, a solver which is able to read, check, and analyze kinetic mechanisms written in the CHEMKIN® format. It is used to rewrite the kinetic mechanism in a XML format which can be efficiently used by every other OpenSMOKE++ solvers;
2. a collection of solvers to simulate chemical reactors like batch reactors, perfectly stirred reactors, shock-tubes, etc. These solvers are completely independent from each other, but need the same pre-processed kinetic mechanism in XML format, generated by the kinetic pre-processor mentioned above;
3. a graphical post-processor, to easily post-process the results of the numerical simulations performed using the OpenSMOKE++ solvers.

Fig. 3 shows a schematic diagram of the links among the different solvers included in the OpenSMOKE++ Suite.

4.1. Kinetic pre-processor

The Kinetic Pre-Processor is a program that reads a symbolic description of a reaction mechanism and then extracts the needed thermodynamic and transport data for each species and the kinetic data for each reaction. The primary output from the Kinetic Pre-Processor is the `kinetics.xml` file written in XML (Extensible Markup Language) format. This file contains all the required data about the atomic elements, the chemical species, and the reactions included in the kinetic mechanism under investigation. The XML format was chosen because of several reasons:

1. accessibility: separation of data makes very easy and computationally efficient to extract them. Moreover, since the formatting instruction are incorporated in the data, if changes are applied in the organization of data, they can be easily managed by the XML reader;
2. standardization: XML is an international standard, which means that it is quite easy to read the data also with third-party software.
3. portability: XML provides a robust and durable format for information storage and transmission. Robust because it is based on a proven standard, and can thus be tested and verified. Durable (i.e. persistent) because it uses plain-text file formats which will live longer than proprietary binary ones.

Once the Kinetic Pre-Processor has been executed and the `kinetics.xml` file created, the user is ready to use the any OpenSMOKE++ solver.

The Kinetic Pre-Processor checks the consistency of thermodynamic coefficients provided by the user to calculate the specific heat, the enthalpy and the entropy of each species. In particular, two sets of 7 coefficients each are needed, one for a low-temperature range $[T_{\min} \div T_{com}]$ and the other one for the high-temperature range $[T_{com} \div T_{\max}]$. At T_{com}, i.e. the common temperature, the thermodynamic properties calculated using the two sets of coefficients must be the same, for consistency reasons.

The Kinetic Pre-Processor allows also to adjust the thermodynamic coefficients in order to ensure not only the continuity of the thermodynamic functions at T_{com}, but also the continuity of first-, second-, and third-order derivatives. This adjustment can result in positive effects during the integration of the ODE systems describing many chemical reactors (see Section 5). The procedure is described in Appendix B.

Additional tests to check the consistency of kinetic data are performed by the Kinetic Pre-Processor (for example about the stoichiometry of the reactions, the existence of duplicated reactions, etc.).

4.2. Reactor solvers

The list of available solvers for chemical reactors is reported in Table 2. The corresponding mathematical models are described in detail in the next paragraph. Typically, the desired initial (or inlet) conditions, including temperature, pressure, and mixture composition (which may be expressed by means of mole fractions, mass fractions, or fuel-oxidizer equivalence ratio) should be specified. If sensitivity analysis is required, the problem data must also include lists of the parameters with respect to which sensitivity coefficients have to be calculated and the dependent variables of interest. For normal code usage, the user will set only two integration parameters: the relative and absolute error tolerances for all the variables. Obviously, the magnitudes of the two error tolerances control the accuracy of the numerical solution, and the computational cost of the simulation usually increases with increased accuracy requirements. Some numerical experiments may be necessary to optimize the local error tolerances (see Section 6).

4.3. Graphical post-processor

The OpenSMOKE++ outputs are not written in binary format, in order to give the user the freedom to use them with any kind of software. However, the OpenSMOKE++ framework is

Table 2
List of solvers available in the current version of OpenSMOKE++ Suite for the simulation of ideal reactors.

Reactor type	Description
Batch reactor	Transient, homogeneous system closed to exchange of mass, open to exchange of heat.
Perfectly stirred reactor	Transient or steady-state perfectly stirred reactor, also known as continuously stirred tank reactor (CSTR).
Plug flow reactor	Tubular reactor with flat radial profiles, without axial dispersion.
Shock tube reactor	Normal incident or reflected shock, used to simulate shock-tube experiments.

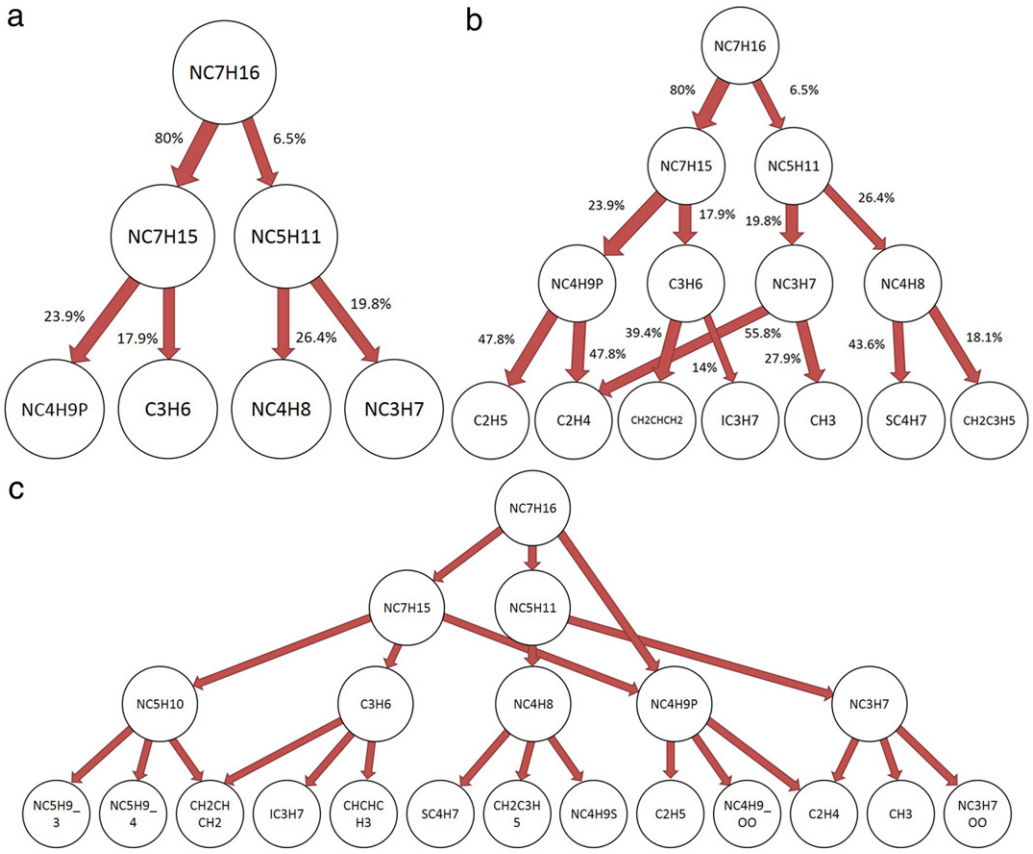

Fig. 4. Reaction path diagrams automatically generated by the Graphical Post-Processor using the GraphViz open-source graph visualization software [61]. The diagrams correspond to the simulation of a constant volume, adiabatic batch reactor at the initial pressure and temperature of 1 atm and 1200 K, respectively, burning a mixture of n-heptane and air with equivalence ratio of 1. The results are obtained using the POLIMI_PRF_PAH_LTHT_1311 kinetic mechanism (276 species and 8439 reactions) [62]. The impact of width (w) and depth (d) parameters on the complexity and the level of accuracy of the diagram is shown: (a) $w = 2, d = 1$; (b) $w = 2, d = 2$; (c) $w = 3, d = 2$.

also equipped with a Graphical Post-Processor, i.e. a software to conveniently post-process and visualize the results of the numerical simulations, which is especially useful for the kinetic analysis of very large mechanisms. The most interesting and useful features are reported in the following:

1. Post-processing of sensitivity analysis: the raw sensitivity coefficients s_{ij}, are normalized according to different options (local versus global normalization), sorted according to their values and plotted using bar charts. Moreover, profiles of most important sensitivity coefficients can be also plotted in 2D charts.
2. Rate of production analysis: the production and destruction coefficients (Eqs. (26) and (27)) are automatically calculated (either locally or over a user-defined region), sorted and plotted using bar charts.
3. Reaction path analysis: the reaction path analysis is automatically performed (either locally or integrated over a user-defined region) for any species with respect to any atomic element available in the kinetic mechanism and the corresponding reaction path diagram is automatically generated and drawn, using the open-source GraphViz graph visualization software [61]. Starting from the requested species and atomic element, the reaction path diagram is drawn according to the width and the depth parameters specified by the user, which are used to control its complexity and the level of detail. In particular, the first parameter is the maximum number of arrows leaving from each node. The depth parameter is an integer which specifies the maximum number of levels to be drawn. Fig. 4 shows an example to better illustrate the capabilities of the reaction path diagram generator.

5. Standard solvers for ideal reactors

In this Section we present the mathematical models behind the solvers reported in Table 2. Additional details about the derivation of the governing equations reported below can be found in [63].

5.1. Batch reactors

A variety of batch reactors can be solved, either at constant pressure, assigned or variable volume, in isothermal or adiabatic conditions or with heat exchange with the external environment. In particular, the conservation equations of species are solved in

terms of mass fractions:

$$\rho\frac{d\omega_i}{dt} = W_i\dot{\Omega}_i \quad i = 1,\dots,N_S \tag{29}$$

while the conservation equation of energy is written directly in terms of temperature. For constant-pressure reactors:

$$\rho\widehat{C}_P\frac{dT}{dt} = -\sum_{i=1}^{N_S}\dot{\Omega}_i\tilde{H}_i + \frac{\dot{Q}}{V} \tag{30}$$

while for constant or variable volume reactors:

$$\rho\widehat{C}_V\frac{dT}{dt} = -\sum_{i=1}^{N_S}\dot{\Omega}_i\left(\tilde{H}_i - RT\right) - \frac{P}{V}\frac{dV}{dt} + \frac{\dot{Q}}{V}. \tag{31}$$

In the expression reported above, $\dot{Q}$ is the power exchanged with the external environment, typically expressed using the global heat exchange coefficient U, the exchange surface area A and the temperature of the environment T_{env}: $\dot{Q} = UA\,(T_{env} - T)$.

5.2. Plug flow reactors

Plug flow reactors (PFR) are commonly used to validate detailed kinetic mechanisms and to perform kinetic analyses. The species and the temperature may vary along the reactor, but it is assumed that there are neither variations in radial direction, neither diffusive transport along the length of the channel, i.e. in the flow direction. The OpenSMOKE++ framework allows the simulation of such reactors, in isothermal or adiabatic conditions or with prescribed heat exchange with the external environment. The conservation equations of species and energy are written with respect to the spatial coordinate ξ. An additional equation is written to reconstruct the residence time τ. Thanks to the assumption reported above, the steady-state equations governing the PFR constitute a ODE system with initial conditions (prescribed on the inlet section):

$$\begin{cases} \rho v\dfrac{d\omega_i}{d\xi} = W_i\dot{\Omega}_i \quad i = 1,\dots,N_S \\ \rho v\left(\hat{C}_P + \dfrac{v^2}{T}\right)\dfrac{dT}{d\xi} = -\displaystyle\sum_{i=1}^{N_S}\dot{\Omega}_i\tilde{H}_i \\ \qquad - v^2 W_{mix}\displaystyle\sum_{i=1}^{N_S}\dot{\Omega}_i + \dfrac{UP_c}{A_c}\,(T_{env} - T) \\ \dfrac{d\tau}{d\xi} = \dfrac{1}{v}. \end{cases} \tag{32}$$

In the equations reported above, P_c and A_c are the perimeter and the area of the cross section surface.

5.3. Transient stirred reactors

The Perfectly Stirred Reactor (PSR) or Continuously Stirred Tank Reactor (CSTR) is an idealization that proves useful in describing laboratory experiments and can often be used in modeling practical devices. Gases enter the reactor with mass flow rate $\dot{m}$, temperature T^{inlet} and composition ω_i^{inlet}. Once inside the reactor, the gases mix instantaneously and perfectly, which means that temperature and composition within the reactor are uniform. The OpenSMOKE++ framework also includes the possibility to study chemical reactions in such kind of reactors, both in steady-state and unsteady conditions. Moreover, the reactor can be adiabatic, isothermal or it can exchange heat with the external environment. The conservation equations of species and energy are reported in the following in the unsteady form:

$$\begin{cases} \rho\dfrac{d\omega_i}{dt} = \rho\dfrac{\omega_i^{inlet} - \omega_i}{\tau} + W_i\dot{\Omega}_i \quad i = 1,\dots,N_S \\ \rho\hat{C}_P\dfrac{dT}{dt} = \rho\dfrac{\displaystyle\sum_{i=1}^{N_S} x_i^{inlet}\left(\tilde{H}_i^{inlet} - \tilde{H}_i\right)}{W_{mix}^{inlet}\tau} - \displaystyle\sum_{i=1}^{N_S}\dot{\Omega}_i\tilde{H}_i + \dfrac{\dot{Q}}{V}. \end{cases} \tag{33}$$

In the equations reported above, τ is the residence time inside the reactor, which is defined as:

$$\tau = \frac{\rho V}{\dot{m}}. \tag{34}$$

In steady-state conditions, Eqs. (33) become a system of nonlinear algebraic equations, since the unsteady terms on the l.h.s. are identically equal to zero. The solution of these equations is obtained using the Newton's method or modified Newton's methods. Unfortunately, especially for large kinetic mechanisms, the Newton's methods are not sufficiently robust, i.e. they require a good first-guess solution to converge. In order to circumvent this problem, even if steady-state simulations are required, OpenSMOKE++ starts solving the conservation equations in the unsteady form, to approach a better first guess solution. Then, after solving for a specified time interval, OpenSMOKE++ attempts to solve the steady-state problem by the Newton's method. If the Newton's method fails, the unsteady equations are solved for an additional time interval to improve the first-guess estimation. This procedure is then repeated up the convergence of the Newton's method. In the worst scenario, the steady-state solution is obtained through the integration of the ODE system (33) for a sufficiently long time interval.

5.4. Shock tubes

The Shock Tube Model (STM) is used to model the chemical kinetics behind a normal incident or a reflected shock. In particular, the interest is especially to simulate the behavior of a shock tube experiment for studying reaction kinetics.

The post-shock conditions, needed to follow the evolution of chemical species after a shock has passed over, are automatically determined by solving the following equations, corresponding to the conservation of mass, momentum, and energy in steady, one-dimensional, inviscid flow of an ideal gas mixture:

$$\begin{cases} p_1 v_1 = p_2 v_2 \\ p_1 + p_1 v_1^2 = p_2 + p_2 v_2^2 \\ \hat{H}_1 + \dfrac{v_1^2}{2} = \hat{H}_2 + \dfrac{v_2^2}{2}. \end{cases} \tag{35}$$

In these equations, which assume that the coordinate system is attached to the shock, the subscripts 1 and 2 indicate conditions upstream and downstream of the shock, respectively. p, v, and $\hat{H}$ are the pressure, the velocity and the mass specific enthalpy of the gas stream, respectively. Eqs. (35) are solved using a Newton–Raphson iteration procedure. The set of equations describing the mass fraction, velocity and temperature profiles downstream of the shock, can be derived from conservation laws of mass, momentum and energy. The flow is assumed to be adiabatic and mass diffusion, thermal conductivity, and viscous effects are assumed to be negligible. Since the typical test times behind a shock wave are on the order of 10^{-4}–10^{-3} s, neglecting transport phenomena does not impact on the accuracy of the results. The conservation equations constitute a ODE system with initial conditions, as reported

in the following:

$$\begin{cases} \rho\dfrac{d\omega_i}{dt} = W_i\dot{\Omega}_i \quad i = 1,\ldots,N_S \\ \rho\hat{C}_P\dfrac{dT}{dt} = v^2\dfrac{d\rho}{dt} - \displaystyle\sum_{i=1}^{N_S}\dot{\Omega}_i\tilde{H}_i \\ \quad + \rho v^3\beta\left[p\left(1+\dfrac{v^2}{\hat{C}_P T}\right) - \rho v^2\right] \\ \dfrac{d\rho}{dt} = -pW_{mix}\displaystyle\sum_{i=1}^{N_S}\dot{\Omega}_i + \dfrac{p}{\hat{C}_P T}\sum_{i=1}^{N_S}\dot{\Omega}_i\tilde{H}_i \\ \quad + \rho^2 v^3\left(1-\dfrac{p}{\rho\hat{C}_P T}\right)\beta \\ \rho\dfrac{dv}{dt} = -v\dfrac{d\rho}{dt} - \rho v^2\beta. \end{cases} \tag{36}$$

The equations reported above are written using the time t as the independent variable instead of distance z, since in typical shock tube experiments, the usual measurable quantities are density, species concentration, velocity and temperature as functions of time. The relationship between time and distance can be obtained by adding the following equation to the ODE system (36):

$$\frac{dz}{dt} = v. \tag{37}$$

In Eqs. (36) β is a correction coefficient which takes into account the boundary layer effects, causing the shock to decelerate, the contact surface to accelerate and the flow behind the shock to be non uniform. In the current version of OpenSMOKE++ β is calculated following the approach proposed by Mirels [64,30].

6. Performances

In this section we analyze and discuss the numerical performances of the OpenSMOKE++ framework for solving problems which are typically faced to perform kinetic analysis. In addition a comparison with the Cantera framework [65] is also presented, in order to better demonstrate the reliability of the OpenSMOKE++ framework and its numerical efficiency. This choice has been made since Cantera is a widely used open-source general purpose code, applied to a large variety of combustion cases with detailed kinetic mechanisms and therefore it provides a suitable solution against which to compare the present code. Moreover, since the Cantera source code is available, it is possible to directly compare its performances through the direct control of the numerical parameters governing the ODE integration, the initialization overhead, and the solution output.

An Intel Xeon E5320 CPU, running at 1.86 GHz with 8 MB of L2 cache memory and 36 Gb of RAM was adopted for running all the simulations presented in this Section. Even if 12 cores were available, the simulations were always performed on a single core. We used the GNU compiler collection (g++) version 4.8.1 to compile both the OpenSMOKE++ and the Cantera solvers (with the compiler options: -O3 -m64). In order to ensure the best performances of Cantera solvers, the Sundials libraries [66] were adopted for solving the stiff ODE systems. The BLAS/LAPACK support was provided through the Intel® MKL Libraries [38], both for the OpenSMOKE++ and the Cantera solvers.

6.1. Evaluation of thermodynamic and kinetic data

The performances of the OpenSMOKE++ framework were firstly evaluated in terms of CPU time needed to calculate thermodynamic (specific heats and enthalpies of species, reaction enthalpies, etc.) and kinetic (reaction and formation rates) properties of mixtures

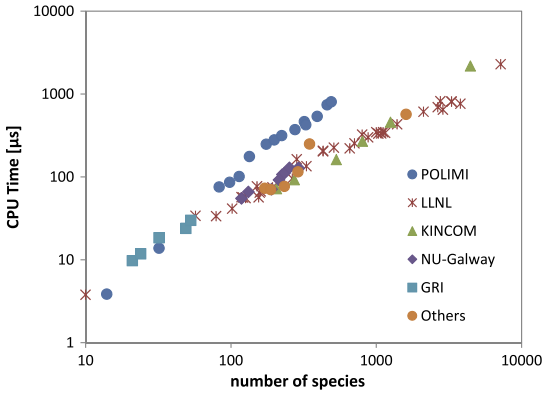

Fig. 5. Overall CPU time for calculating thermodynamic and kinetic data for different kinetic mechanisms versus the number of species.

of ideal gases with different number of species and reactions. In particular the following operations were analyzed:

1. calculation of specific heat of species;
2. calculation of enthalpies of species and reaction enthalpies for all the reactions;
3. calculation of kinetic constants for all the reactions;
4. calculation of reaction rates;
5. calculation of formation rates for all the species.

This choice has been made since the operations reported above are typically required at each time step for the simulation of ideal reactors (see next Section). Fig. 5 reports the overall CPU time to perform the 5 operations reported above for the different kinetic mechanisms already analyzed in Fig. 1. All the kinetic mechanisms, with the exception of POLIMI family, tend to collapse on the same line, showing that the CPU time increases with the number of species with a power of ~0.9 (i.e. less than linearly). This is due to the fact that the number of reactions tends to increase slower than the number of species (i.e. the ratio reactions/species tends to decrease with increasing the number of species). The kinetic mechanisms belonging to the POLIMI family show a different trend: in particular the CPU time is 2–3 times larger than kinetic mechanisms with the same number of species and the rate of increase occurs with a power of ~1.4. In both cases the explanation is simply related to the larger reactions/species ratio of POLIMI kinetic mechanisms (about 35 for the largest scheme against a mean value of ~4 for the other kinetic mechanisms).

Fig. 6 reports the relative weights of the five operations reported above for the main families of kinetic mechanisms. Independently of the kinetic mechanism, the computational cost for evaluating the specific heat is negligible (less than 2%). The evaluation of kinetic constants and formation rates for all the species require usually 15%–20% and 10%–15% of total CPU time, respectively. The evaluation of reaction rates is the most consuming part, with a relative weight of 35%–40% for POLIMI family or 45%–50% for the other families. The calculation of reaction enthalpies is quite time consuming (up to 35%) for the kinetic mechanisms belonging to the POLIMI family, because of the high number of reactions involved. In other case, the cost is smaller (around 20%).

6.2. Evaluation of transport properties

Transport properties (thermal conductivity, dynamic viscosity, mass diffusivities and thermal diffusion ratios) are required for the simulation of combustion systems. In particular, the OpenSMOKE++ framework calculates the transport properties of ideal gases using the technique described in Section 2 in order to minimize the computational cost, which would be prohibitive

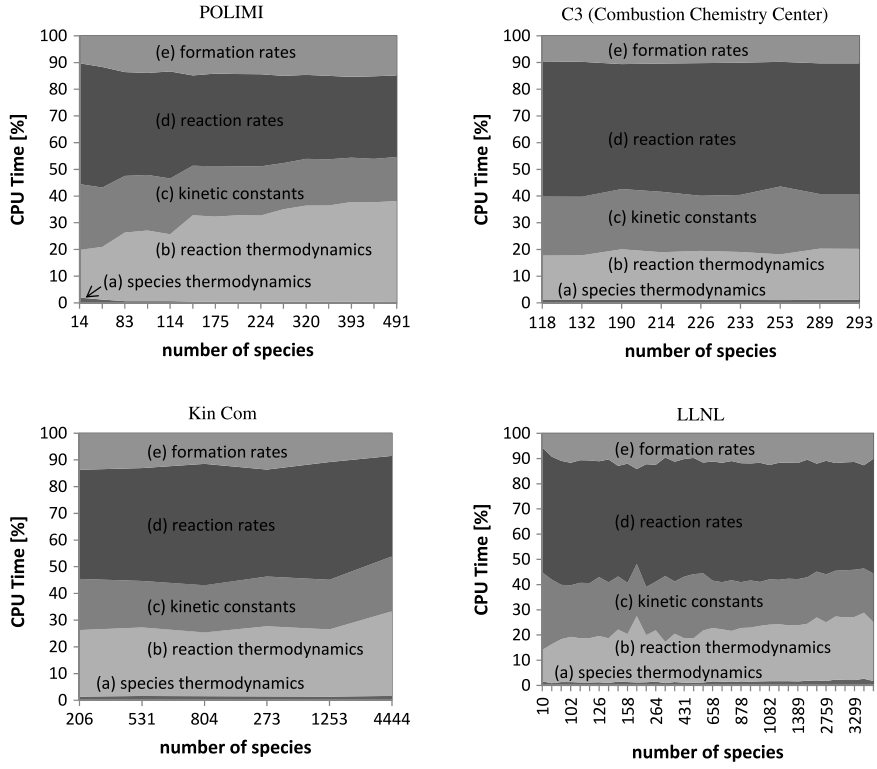

Fig. 6. Relative weight of the different operations required to calculate thermodynamic and kinetic properties for different families of kinetic mechanisms.

for large kinetic mechanism. A difference with respect to the thermodynamic and kinetic properties is that the transport properties are only a function of the number of species and do not depend on the number of reactions involved in the kinetic mechanism. In analyzing the CPU cost required for evaluating the transport properties, we split the cost for calculating the properties of single species from the cost for calculating the properties for the whole mixture (i.e. the cost behind the application of proper mixture-averaging rules).

Since the thermal diffusion ratios are calculated only for species with very low molecular weight (less than 10 kg/kmol), the cost for their evaluation is negligible with respect the other transport properties, especially for large detailed mechanisms with hundreds of species. As a consequence, in the following analysis they are neglected.

The transport properties for the single species are calculated using the same polynomial approximation (see Eqs. (4)–(6)). Thus, the computational cost for evaluating the single transport property of a single species is the same. However, since in a mixture of N_S species, $N_S \times \frac{N_S}{2}$ binary diffusion coefficients $\Gamma^0_{i,j}$ are needed (the factor 2 is due to the symmetry of $\boldsymbol{\Gamma}^0$), the cost for evaluating them is $\frac{N_S}{2}$ times higher than the cost associated to thermal conductivities or dynamic viscosities. Thus the cost of calculation of single species transport properties is dominated by the mass diffusion coefficients and scales quadratically with the number of species. The CPU time required for evaluating the mixture averaged properties depends on the mixing rule, as better explained in the following.

Fig. 7 reports the CPU time needed to calculate the mixture averaged properties (including the cost to calculate the corresponding properties for single species) for kinetic mechanisms with different number of species. Moreover, a comparison with respect to the CPU time required to calculate the thermodynamic and transport properties is also reported. From the analysis of Fig. 7 it is evident that the computational cost for transport properties

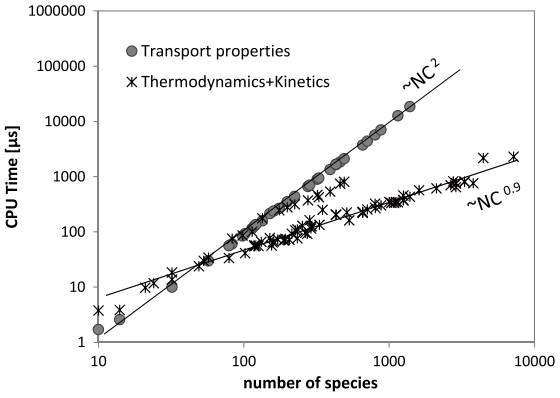

Fig. 7. Comparison between CPU times required for the evaluation of mixture averaged transport properties (thermal conductivity, dynamic viscosity, mass diffusion coefficients) and thermodynamic and kinetic data as a function of the number of species in the kinetic mechanism. The points refer only to the mechanisms reported in Fig. 1.

increases very fast with the number of species and becomes systematically higher than the cost for thermodynamic and kinetic properties when the number of species is larger than ∼200. The CPU time for transport properties is very unbalanced among the different operations required.

Fig. 8 shows the relative weights of such operations versus the number of species. First of all it is evident that for sufficiently large schemes the relative weights becomes independent of the number of species. The time for calculating the species thermal conductivities and viscosities is negligible, as expected. Also the evaluation of the mixture thermal conductivity (Eq. (10)) has a negligible weight. On the contrary, the calculation of binary diffusion coefficients requires ∼40% of total CPU time, and the evolution of the corresponding mixture averaged diffusivities ∼20%. Surprisingly, the application of the mixing rule for viscosity (Eq. (8)) is very expensive, requiring ∼40% of total CPU time.

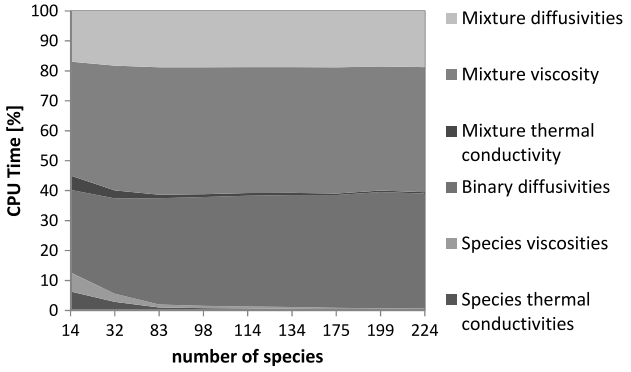

Fig. 8. Relative weights of different operations for the evaluation of mixture-averaged transport properties as a function of the number of species in the kinetic mechanism. Due to the asymptotic behavior of relative weights of CPU times with the number of species, only the smallest kinetic mechanisms were taken into account among those reported in Fig. 1.

In order to reduce the computational cost spent for calculating the mixture viscosity, the mixing rule proposed by Herning and Zipperer [67] was also implemented in the OpenSMOKE++ framework and tested:

$$\eta = \frac{\sum_{i=1}^{N_S} x_i \eta_i \sqrt{W_i}}{\sum_{i=1}^{N_S} x_i \sqrt{W_i}}. \tag{38}$$

The formula reported above is less accurate than the original Wilke formula, especially at high pressures. However, we found that the relative error between the two formulations was always smaller than ∼4% in a wide range of operating conditions of temperature and composition. The advantage of the Herning and Zipperer formula is the computational cost, which increases only linearly with the number of species, as reported in Fig. 8.

6.3. Adiabatic, constant-volume batch reactors fed with n-heptane

Following the work performed by Perini et al. [68], the first benchmark was chosen as a set of 18 constant-volume, adiabatic batch reactors, burning a mixture of n-heptane and air. The 18 problems have different initial conditions, corresponding to a matrix of cases involving two initial pressure values (2 and 20 bar), three temperature values (750, 1000 and 1500 K), and three initial compositions of the fuel–air mixture (λ equal to 0.5, 1, and 2). Each case is integrated for a time interval equal to 0.01 s. Table 3 reports the full details about the initialization matrix.

The simulations were carried out with all the available solvers in OpenSMOKE++ (to compare their relative performances) and with the Cantera framework [65]. The same absolute and relative tolerances, respectively equal to 10^{-14} and 10^{-8}, were used for all the simulations, regardless the adopted solver. Four different kinetic mechanisms were adopted to perform the simulations, with increasing number of species: a skeletal mechanism from Lu et al. [69] (188 species and 842 reactions), the detailed scheme of Herbinet et al. [70] (273 species and 1853 species), the POLIMI_TOT_1311 kinetic mechanisms (460 species and 16,000 reactions) [4], and the latest version of the LLNL n-heptane mechanism (658 species and 2827 reactions) [9].

Fig. 9 reports the calculated temperature profiles with the four mechanisms reported above for Case 2 (see Table 3). This is an initial low temperature, low pressure case, where the system's stiffness is at its peak [68], and therefore it is particularly interesting for numerical analysis. The predicted ignition delay

Table 3
Initial conditions for the constant-volume batch reactors.

Case	T_0[K]	P_0[bar]	Mixture λ
1	750	2	0.5
2	750	2	1
3	750	2	2
4	1000	2	0.5
5	1000	2	1
6	1000	2	2
7	1500	2	0.5
8	1500	2	1
9	1500	2	2
10	750	20	0.5
11	750	20	1
12	750	20	2
13	1000	20	0.5
14	1000	20	1
15	1000	20	2
16	1500	20	0.5
17	1500	20	1
18	1500	20	2

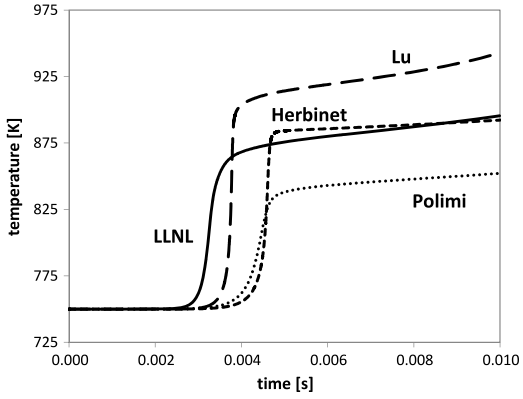

Fig. 9. Predicted temperature profiles versus time in a batch reactors fed with n-heptane and air. Case 2: initial composition λ = 1, initial pressure p_0 = 2 bar, initial temperature T_0 = 750 K (Table 3).

times range from ∼3 to ∼4.5 ms, according to the employed kinetic mechanism.

Fig. 10 shows the comparison between the OpenSMOKE++ and Cantera solutions for Case 2 (only the results referring to the LLNL mechanism are reported). Excellent agreement between the two solutions is evident. A similar agreement was also observed for all the other cases, which are not here reported for sake of brevity.

The numerical performances of the OpenSMOKE++ framework were tested by solving the 18 cases summarized in Fig. 15, comparing the 8 different ODE solvers for stiff problems reported in Table 1. Moreover, as a further comparison with existing external codes, the same simulations were repeated using the Cantera framework. The results are summarized in Fig. 11, where the overall CPU time (i.e. summed up over all the 18 cases) is reported for the 4 kinetic mechanisms reported above. Moreover, Tables 4 and 5 report the overall number of time steps and Jacobian matrix evaluations for the 7 ODE solvers, respectively. From the analysis of such results, it is evident that the OpenSMOKE++ performances are strongly affected by the adopted ODE solver. In particular, as evident from Fig. 11, the best performances are always associated with the CVODE solver. Also the DASPK solver shows very good results in terms of CPU time, but it resulted less robust than CVODE: as an example it failed in solving 4 cases when the POLIMI_TOT_1311 kinetic mechanism is adopted. The DVODE and DLSODE showed similar performances: they are very robust (no failures were observed), but slower than CVODE (up to ∼5 times slower). The RADAU5 implicit solver is very

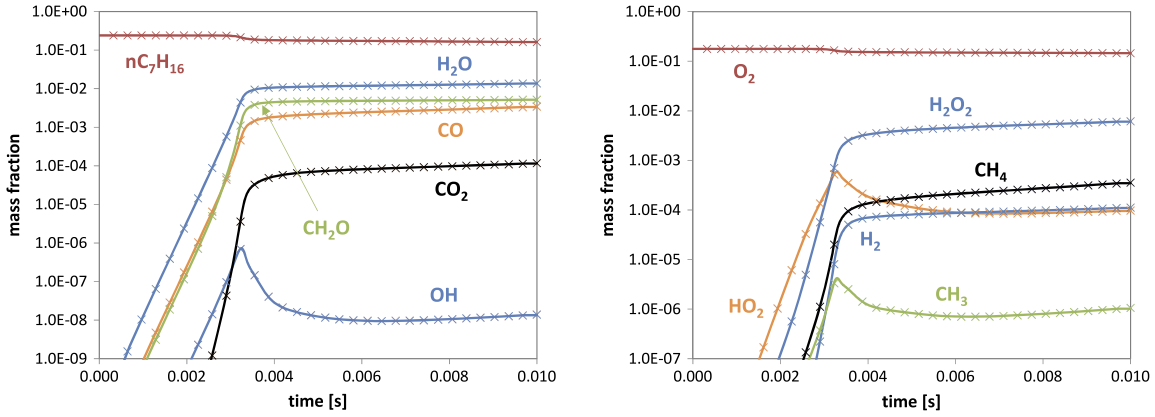

Fig. 10. Mass fraction profiles of selected species versus the time: comparison between the OpenSMOKE++ simulation (lines) and the Cantera simulation (points) performed using the detailed LLNL n-heptane kinetic mechanism. Case 2: initial composition $\lambda = 1$, initial pressure $p_0 = 2$ bar, initial temperature $T_0 = 750$ K (Table 3).

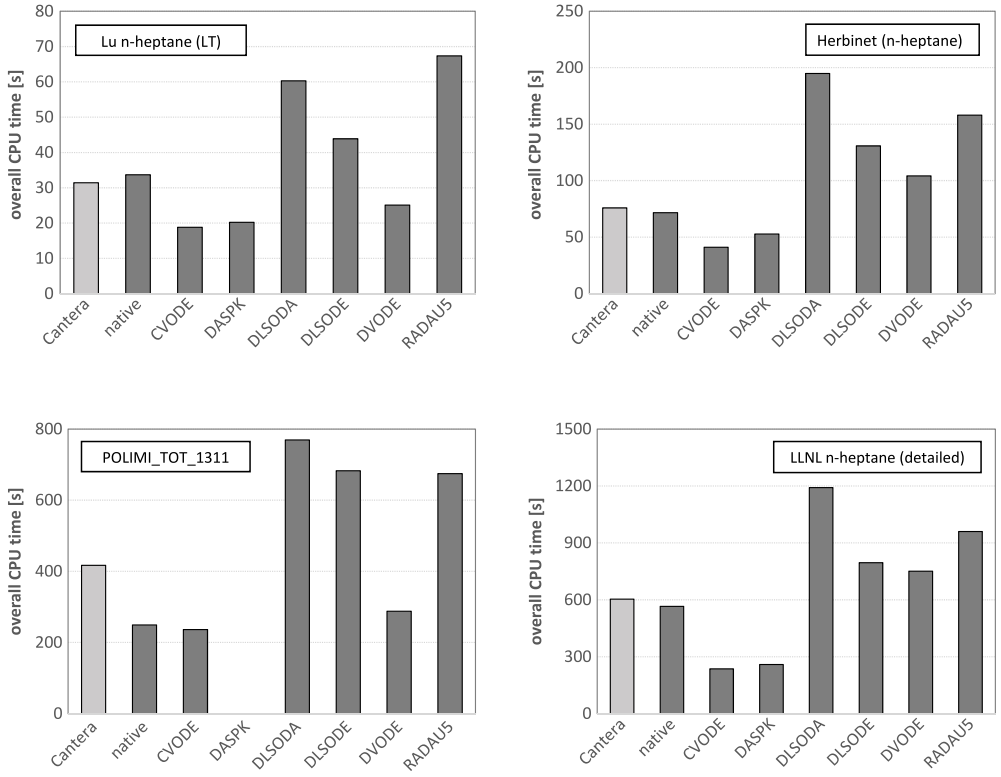

Fig. 11. Performance comparison between Cantera and the OpenSMOKE++ framework: overall CPU times for the 18 cases reported in Fig. 15. The batch reactors are fed with a mixture of n-heptane and air. The OpenSMOKE++ framework was tested using the 8 different stiff ODE solvers summarized in Table 1. If a ODE solver failed in integrating at least one of the cases reported in Table 3, the corresponding bar is not reported.

accurate, but this limits its computational efficiency, since many function evaluations per step are needed. As expected, the worst performances were obtained with the DLSODA solver: it switches automatically between stiff and non-stiff methods, but, since it always starts with the non-stiff method, this can result in a penalty in the CPU time. Moreover, several failures were observed when the POLIMI_TOT_1311 kinetic mechanism was employed. The native OpenSMOKE++ solver resulted very robust (no failures were observed) and sufficiently fast (only the CVODE solver is systematically faster). Some interesting observations can be made from Tables 4 and 5:

- the CVODE and DVODE have a very similar behavior in terms of number of time steps and Jacobian evaluations. This is expected, since the CVODE is the C version of DVODE (which is written in FORTRAN). However, the performances in terms of CPU time are very different. In our opinion, this can be mainly attributed to the different solvers for the linear systems: the CVODE uses the BLAS/LAPACK libraries (through the Intel® MKL implementation), while the less recent LINPACK libraries are used by the DVODE solver;
- the native OpenSMOKE++ solver requires a smaller number of Jacobian evaluations with respect to the CVODE solver, but the overall number of time steps is much larger. This could explain its worse performances in terms of CPU time;
- the number of time steps of DLSODE is similar to CVODE, but the number of Jacobian evaluations is much larger, resulting in quite slow performances;
- the number of steps required by the RADAU5 solver is very small if compared to the other solvers, but the number of Jacobian evaluations is huge. Moreover, since it is based on implicit

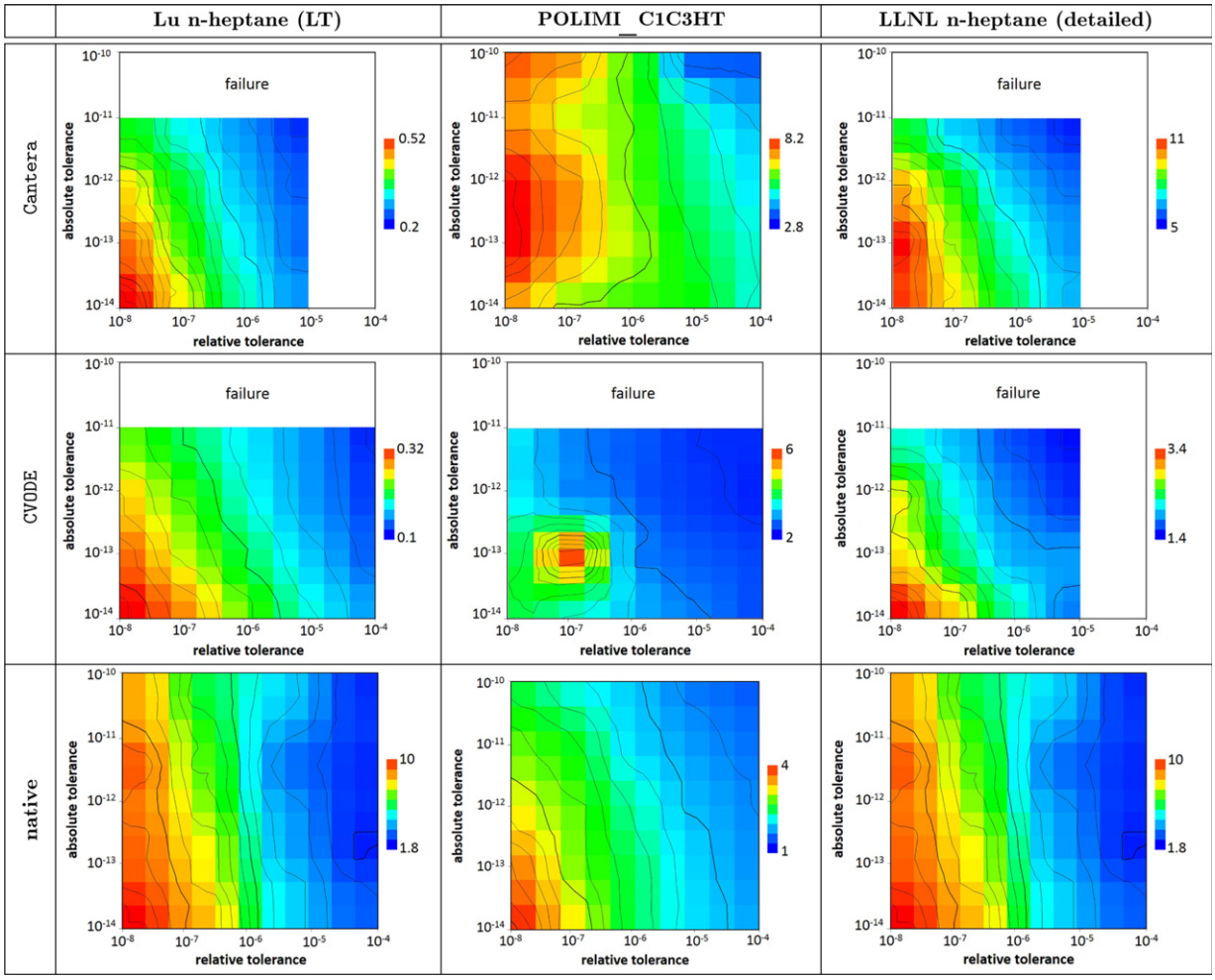

Fig. 12. Comparison of overall computational times (in seconds) required for the numerical integration of the 18 cases reported in Table 3 as a function of the relative and absolute tolerances. The CPU times reported in the maps are in minutes. The analysis is limited to the most effective ODE solvers available in OpenSMOKE++ (i.e. the native solver and CVODE), which are compared to the Cantera framework [65].

Runge–Kutta method, the solution of a larger linear system is required, which further increases the computational time.

The comparison with respect to Cantera is quite encouraging: when the CVODE or the native ODE solvers are employed, the OpenSMOKE++ framework is significantly faster, especially for large number of species (as an example, when the LLNL mechanism is adopted, the CVODE solvers is ∼3 times faster than Cantera).

It is important to remark that the computational cost of the simulations of the batch reactors described in this section is strongly affected by the tolerances employed in the ODE solver, as summarized through the maps reported in Fig. 12. The maps show the overall CPU time needed for integrating the 18 cases (see Table 3) for different values of relative tolerances (from 10^{-4} to 10^{-8}) and absolute tolerances (from 10^{-10} to 10^{-14}). The analysis is limited to the Cantera framework and to the two fastest ODE integrators in OpenSMOKE++, i.e. CVODE and the native solver. In some cases, when the tolerances are too large, the integration failed. As expected, the required CPU time increases when the tolerances are decreased in a monotonic way. However, in some cases there are combinations of values of relative and absolute tolerances which result in a maximum in the CPU time.

6.4. Performances with large kinetic mechanisms

The tests performed in the previous sections were performed with medium-size kinetic mechanisms. Additional analyses were performed using large kinetic mechanisms, with thousands of species, through the simulation of the same 18 adiabatic, constant volume batch reactors already described in Section 6.3.

First of all an inlet mixture of methyl-decanoate and air was considered. Three kinetic mechanisms were compared: the POLIMI_TOT_1311 kinetic mechanisms (460 species and 16,000 reactions) [4], the mechanism of Glaude et al. [71] (1253 species and 7146 reactions), and the latest version of the LLNL methyl-decanoate mechanism (2878 species and 8855 reactions) [3]. The performances of OpenSMOKE++ and Cantera are summarized in Fig. 13. The results are analogous to the n-heptane case. In particular, the best performances are always obtained using the CVODE solver, while the DLSODA solver gives the worst performances, not only in terms of CPU time, but also in terms of reliability, since in many cases failures in the integration of the ODE system were observed. The native OpenSMOKE++ solver results in satisfactory performances, especially if compared with common ODE solvers, like DLSODE, DVODE and RADAU5. The overall performances of OpenSMOKE++ appears very promising, especially if compared with Cantera: in particular, for the largest kinetic mechanism (2878 species) the OpenSMOKE++ calculations (based on the CVODE integrator) resulted ∼8 times faster than Cantera.

As a final example, the kinetic mechanism of Westbrook et al. (7175 species and 31,669 reactions) [6] was taken into account for simulating a Diesel surrogate ($nC_7H_{16}/nC_{16}H_{34}/C_{14}H_{30}$, 40/10/50%

Table 4
Solver performance comparison for stiff ODE solvers available in OpenSMOKE++: overall number of steps for the 18 cases reported in Table 3.

	Native	BzzOde	CVODE	DASPK	DLSODE	DLSODA	DVODE	RADAU5
Lu	68771	67330	34216	35513	29031	40854	34050	6714
KinCom	106879	99874	45400	45389	36375	53470	44601	8164
Polimi	104217	97092	41856	20632	32930	47052	42576	5991
LLNL	110820	106702	36625	36475	30369	46857	37471	6445

Table 5
Solver performance comparison for stiff ODE solvers available in OpenSMOKE++: overall number of evaluations of Jacobian matrix for the 18 cases reported in Table 3.

	native	BzzOde	CVODE	DASPK	DLSODE	DLSODA	DVODE	RADAU5
Lu	286	288	695	1381	4257	5747	693	5251
KinCom	359	356	919	1879	4968	7003	904	6085
Polimi	357	359	950	Failed	4678	Failed	943	4898
LLNL	321	322	720	1518	4331	6393	729	4769

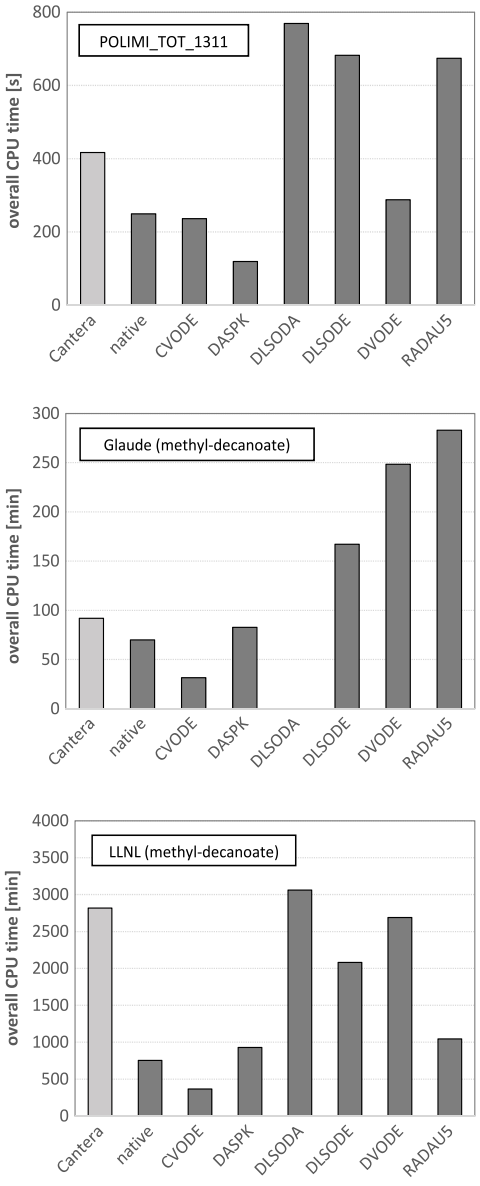

Fig. 13. Performance comparison between Cantera and the OpenSMOKE++ framework: overall CPU times for the 18 cases reported in Table 3. The batch reactors are fed with a mixture of methyl-decanoate and air. The OpenSMOKE++ framework was tested using the 8 different stiff ODE solvers summarized in Table 1. If a ODE solver failed in integrated at least one of the cases reported in Table 3, the corresponding bar is not reported.

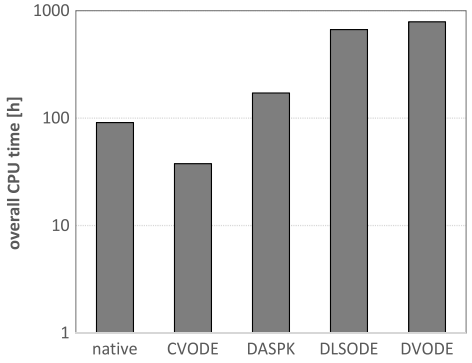

Fig. 14. Batch reactors fed with a Diesel surrogate ($nC_7H_{16}/nC_{16}H_{34}/C_{14}H_{30}$, 40/10/50% mol) and air: performance comparison between among different ODE solvers available in OpenSMOKE++.

mol) burning in air [55]. This last test is computationally very expensive, because of the large number of species and reactions in the mechanism. The numerical analyses were performed using only the most effective ODE solvers available in OpenSMOKE++, namely the native solver, CVODE, DASPK, DLSODE, and DVODE solvers. The results are summarized in Fig. 14, where the overall CPU time to simulate the complete set of 18 batch reactors is reported. The important role played by the ODE solver is very evident from this example: as usual, the best performance is obtained using the CVODE solver, which is also ∼20 times faster than DLSODE and DVODE solvers.

6.5. Oscillating cool-flames in a jet-stirred reactor

The oxidation chemistry in the low- and intermediate-temperature regimes (600–900 K) plays a significant role in combustion processes, such as autoignition in diesel engines, end-gas autoignition and knock phenomena in SI engines are initiated at these low temperatures [72]. In the low-temperature regime the oxidation of hydrocarbons is a very complex process, involving different propagation and chain branching reactions, which can lead to a large variety of phenomena: oscillatory cool-flame, single-stage, two-stage, and multistage ignitions [73].

In this section we focus the attention on the numerical modeling of jet-stirred reactors dominated by the low-temperature mechanism, in which the formation of oscillating cool flames can be observed. The system is quite challenging from a numerical point of view and represents an additional test to analyze the robustness of the OpenSMOKE++ framework.

The numerical tests were performed by considering a spherical, ideal, jet-stirred flow reactor (see Eqs. (33)) with internal vol-

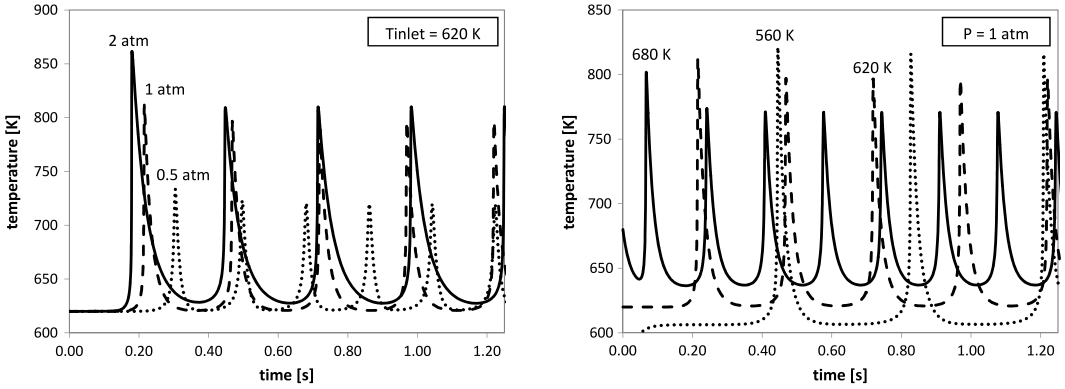

Fig. 15. Temperature profiles in a jet-stirred reactor exchanging heat with the external environment (at 620 K), fed with a mixture of n-heptane and air in stoichiometric amounts, for different values of pressure and inlet temperature. Reactor parameters: $V = 100$ cm^3, $\tau = 200$ ms, $U = 0.5$ kcal/m^2/s. The results refers to the POLIMI_TOT_1311 kinetic mechanisms (460 species and 16,000 reactions) [4].

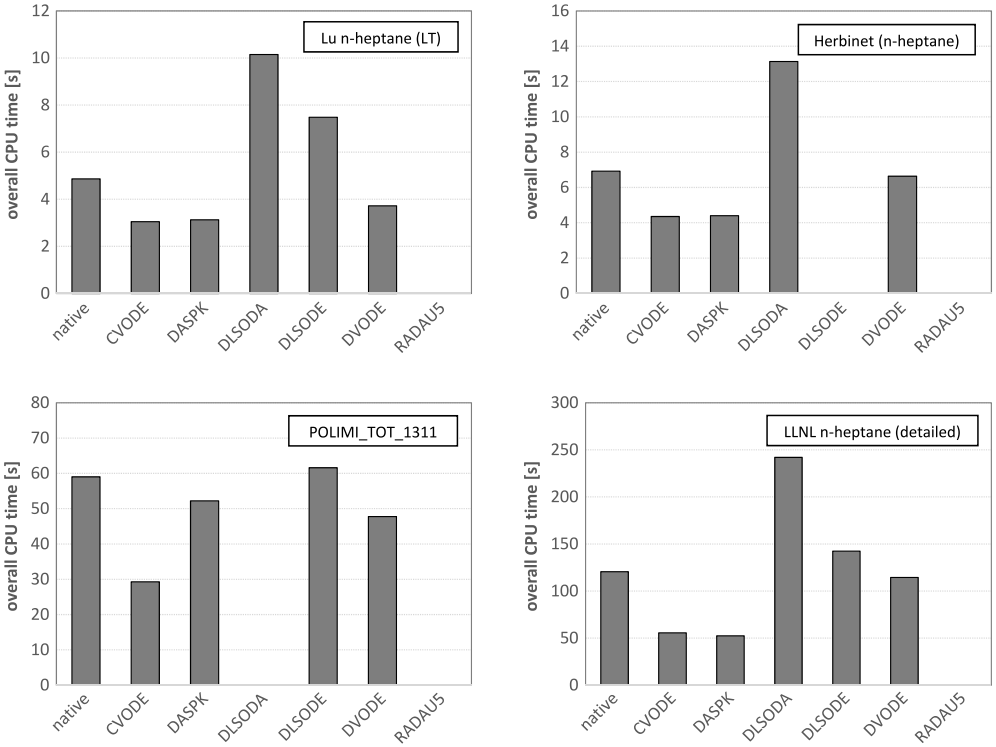

Fig. 16. Computational performances of ODE solvers available in OpenSMOKE++: CPU times for the simulation of a dynamic jet-stirred reactor with thermal exchange with the external environment (at 620 K), fed with a mixture of n-heptane and air in stoichiometric amounts. Reactor parameters: $V = 100$ cm^3, $\tau = 200$ ms, $T_{in} = 620$ K, $P = 2$ atm, $U = 0.5$ kcal/m^2/s. The performances are reported for the following kinetic mechanisms: a skeletal mechanism from Lu et al. [69] (188 species and 842 reactions), the detailed scheme of Herbinet et al. [70] (273 species and 1853 species), the POLIMI_TOT_1311 kinetic mechanisms (460 species and 16,000 reactions) [4], and the latest version of the LLNL n-heptane mechanism (658 species and 2827 reactions) [9].

ume of 100 cm^3 and residence time of 200 ms, fed with a mixture of n-heptane and air (with equivalence ratio equal to 1) at different temperatures and pressures. Heterogeneous reactions are neglected and heat exchange with the external environment (at the fixed temperature of 620 K) is accounted for, assuming a global heat-transfer coefficient equal to 0.5 kcal/m^2/°C (as suggested by Lignola et al. [74]).

Fig. 15 shows the calculated temperature profiles versus time for different inlet conditions. The formation of oscillating cool flames sustained by the continuous feed is evident for all the inlet temperature and pressures investigated. In particular, the frequency of these cool flames increases with the inlet temperature and the pressure of the reactor. Moreover, the oscillation amplitudes, both in the reactor temperature and fuel conversion, are higher at reduced conditions, that is, lower temperatures, pressures, and/or short contact time (not here reported). The predicted behavior qualitatively reproduces the experimental observations with satisfactory agreement [75].

The performances of the ODE solvers available in OpenSMOKE++ are summarized in Fig. 16, where the CPU time to solve the case at T = 620 K and p = 2 atm is reported. Different kinetic mechanisms were employed and compared (see Section 6.3). Also in this case, the best performances are obtained using the CVODE solver and satisfactory CPU times were achieved by the native OpenSMOKE++ solver, especially if compared with common ODE solvers, like DLSODE and DVODE. The RADAU5 solver gave the worst performances, since it was never able to give a solution.

6.6. Sensitivity analysis

The calculation of sensitivity coefficients is a very CPU intensive operation when large kinetic mechanisms are taken into account.

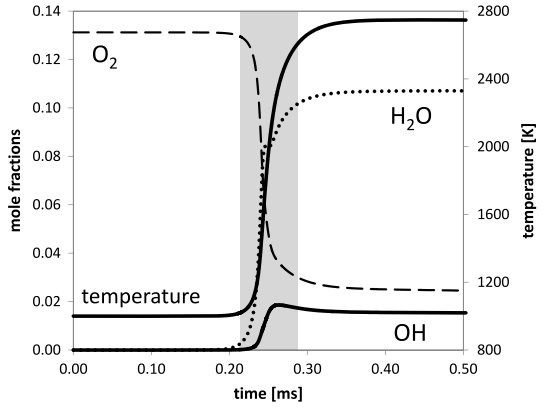

Fig. 17. Calculated profiles of mole fractions of selected species and temperature in a constant-volume batch reactor fed with a mixture of $CO/H_2/N_2$ (40/30/30% mol) and air in stoichiometric conditions at initial pressure $P_0 = 1$ atm and temperature $T_0 = 1000$ K. The simulation was performed using the POLIMI_H2CO_1311 kinetic mechanism [76].

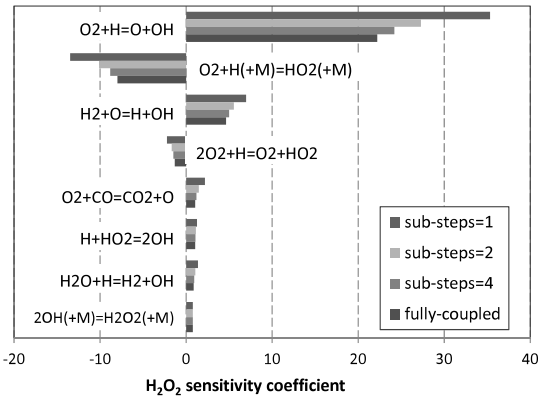

Fig. 19. Calculated sensitivity coefficients of H_2O_2 for the batch reactor described in Fig. 17. The coefficients are calculated at time $t = 0.24$ ms.

This is due to the huge number of equations which have to be solved simultaneously. As reported in Section 3.4, OpenSMOKE++ uses a modified version of the direct staggered algorithm, since the fully-coupled solution of Eqs. (20) and (22) is possible only when the number of species and reactions is small.

In order to show the accuracy and the reliability of the staggered algorithm, sensitivity analyses were performed through the simulation of a constant-pressure batch reactor, fed with a mixture of $CO/H_2/N_2$ (40/30/30% mol) and air in stoichiometric conditions at the initial pressure and temperature of 1 atm and 1000 K, respectively. A kinetic mechanism, with 14 species and 34 reactions, was employed [76]. Fig. 17 shows the calculated temperature profile, together with the mole fraction profiles of selected species. Because of the small dimensions of the kinetic mechanism, the sensitivity coefficients were calculated using both the fully-coupled approach (here assumed as the reference solution) and the staggered technique. Fig. 18 reports the calculated H_2O_2 sensitivity coefficient profiles for the first two most important reactions: the continuous line represents the fully-coupled solution, while the remaining lines the solutions obtained using the staggered algorithm with different number of sub-steps. The selected time interval (from 0.22 to 0.28 ms) corresponds to the shaded region reported in Fig. 17. From these results it is evident that the number of sub-steps affect the absolute values of sensitivity coefficients, but the shape of the curves is correctly captured also using only 1 sub-step. Actually, since in performing the sensitivity analysis the real interest is often in the relative weights of sensitivity coefficients and less in their absolute values, also the staggered solution with only 1 sub-step could be considered acceptable in most cases. This is more evident from Fig. 19, where the H_2O_2 sensitivity coefficients are reported for the first 8 reactions at $t = 0.24$ ms.

The computational performances of the staggered algorithm were evaluated through the simulations of constant-volume batch reactors, already described in Section 6.3. First of all the attention was focused on n-heptane/air mixtures and cases 2 and 11 (see Table 3). The CPU times required to perform the simulations, without and with the sensitivity analysis, are summarized in Table 6.

From these results it is evident, as expected, that the additional cost of sensitivity analysis can be orders of magnitude larger than the time for the simulation of the reactor. Most of the time is spent to factorize the Jacobian matrix and to solve the corresponding linear system through the backward-substitution. In particular at each time step, the Jacobian matrix is factorized only once (independently of the number of reactions), but the backward substitution is performed N_R times, where N_R is the number of reactions. This easily explains why the CPU time to perform the sensitivity analysis with the POLIMI_TOT_1311 kinetic mechanism (16,000 reactions) is much larger than the time required by the LLNL mechanism (2827 reactions), despite the number of species of the latter (658) is larger than the first (460).

Larger kinetic mechanisms were also investigated. In particular, cases 2 and 11 (see Table 3) were simulated for methyldecanoate/air mixtures using the LLNL kinetic mechanism (2878 species and 8555 reactions). This kinetic mechanism is particularly challenging for sensitivity analysis, since at each time step 24.6

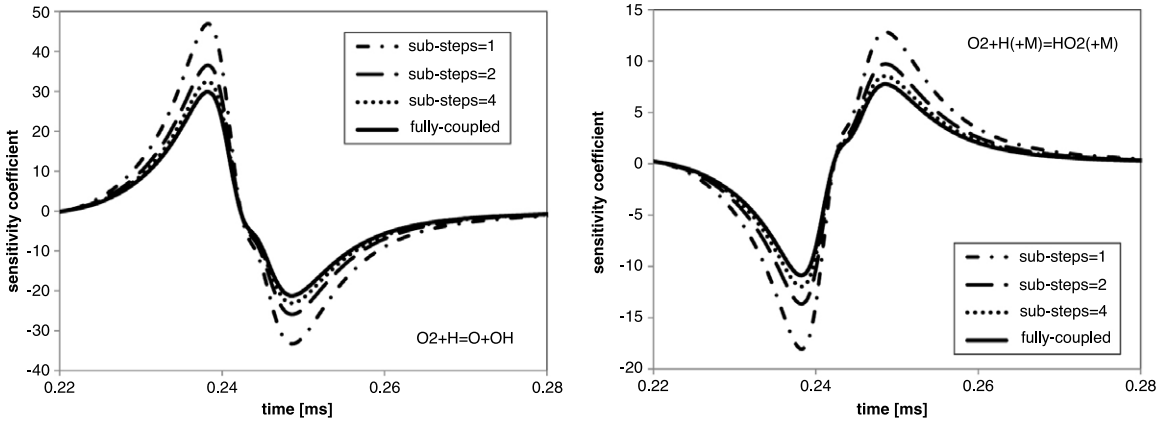

Fig. 18. Calculated sensitivity coefficient profiles of H_2O_2 for the batch reactor analyzed in Section 6.6. The panels refer to the first two most sensitive reactions. The calculations were done using the fully-coupled approach (continuous line) or using the staggered method with different numbers of time sub-steps (1, 2 or 4).

Table 6
Constant-volume batch reactors fed with n-heptane/air mixtures: performances of sensitivity analysis. The CPU times, reported only for 2 cases (Table 3), representative of low- and high-pressure conditions, compare the simulations performed without and with the sensitivity analysis to better show the additional cost due to the calculation of sensitivity coefficients.

	N_S	N_R	N_P	Case 2: CPU time [s]		Case 11: CPU time [s]	
				Without	With	Without	With
Lu	188	842	$0.16 \cdot 10^6$	0.52	72.5	0.98	133
Herbinet	273	1853	$0.50 \cdot 10^6$	0.84	209	2.21	497
LLNL	658	2827	$1.86 \cdot 10^6$	6.43	1129	17.4	3148
Polimi	460	16000	$7.36 \cdot 10^6$	6.17	7045	21.8	18790

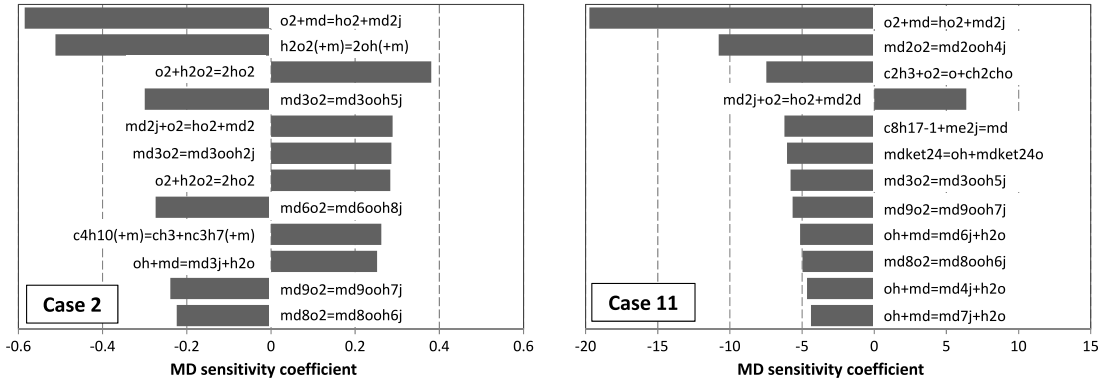

Fig. 20. Calculated sensitivity coefficients of methyl-decanoate (MD) for Cases 2 and 11 described in Table 3.

millions of sensitivity coefficients has to be calculated. The required CPU times were in line with the expected trends from the previous analysis: 28.3 and 65 h for case 2 and 11, respectively, while the corresponding CPU times without the calculation of sensitivity coefficients were 6.7 and 16.5 min.

The huge number of sensitivity coefficients means that it is impossible to store all of the sensitivity results at each time-step. In this particular example, assuming double precision, the result would be a ∼187 MB file for each time-step. For this reason, even if the sensitivity analysis is performed for all the species and all the reactions, the user has to select a limited number of species for which all the sensitivity coefficients have to be writ-ten in a file. Fig. 20 reports the calculated sensitivity coefficients of methyl-decanoate (MD) for the two cases. The numbers re-ported above suggest the fundamental importance of proper post-processing tools to analyze the calculated sensitivity coefficients. The `OpenSMOKE++` Suite includes a graphical post-processor for the efficient treatment of sensitivity analysis results: normaliza-tion of coefficients with different options; identification of most important sensitivity coefficients for each species (on a local or global basis); automatic generation of bar-charts (see Fig. 20); automatic plotting of sensitivity profiles (see Fig. 18).

7. Illustrative examples

In this Section we present a series of illustrative examples which can be typically be solved using the `OpenSMOKE++` framework. The purpose is to demonstrate that large kinetic mechanisms can be efficiently and easily managed, also on complex systems, like multi-dimensional flames.

7.1. Ideal reactors

The design of combustion systems relies on the availability of accurate chemical kinetic mechanisms. In particular, ignition delay time data at elevated pressures and low-to-intermediate temperatures are important for the validation of chemical kinetics models at practical engine conditions. However, it has been noticed that shock-tube ignition data can significantly differ from rapid compression machine (RCM) data especially at high pressures and low temperatures. Ignition delay times obtained from shock-tube experiments are typically shorter than data obtained in RCM experiments [77]. Sung and Curran [78] recently discussed the typical operating ranges of shock tube and RCM experiments. RCMs typically give accessibility to study auto-ignition chemistry at elevated pressure under conditions in which reactivity may be too slow for shock tubes. The complementary combination of RCM and shock tube experiments allows to develop and validate kinetic mechanisms over a very wide range of conditions.

As an example of the combination of shock tube and RCM experiments, Nakamura et al. [79] recently carried out a comprehensive experimental and theoretical study of n-butylbenzene oxidation in air over a wide range of conditions. In particular, the ignition delay times at low temperature were measured using the rapid compression machine at NUI Galway, which adopts a creviced piston, while a shock tube was used to investigate the reactivity at higher temperatures.

The `OpenSMOKE++` code includes a solver for the simulation of a transient closed homogeneous batch reactor, which can be used to perform calculations of ignition delay times experimentally obtained in Shock Tubes (ST) and RCM devices.

It is important to notice that both systems (ST and RCM) typically exhibit a non-ideal behavior, which limits the applicability of a simple adiabatic constant-volume simulation. For example, for the simulations of RCM experiments it is important to include in the simulation the variable volume-time histories, to reproduce in the simulation facility effects such as reaction during compression and heat loss. These variable volume-time histories are nowadays generally experimentally obtained from non-reactive pressure traces [78,79] and can be directly adopted in the simulation. Fig. 21 shows an example of simulation results obtained using the n-butyl-benzene kinetic mechanism of Nakamura et al. [79] (which includes about 960 species and 4330 reactions) and adopting both constant volume simulations and the proper volume time history for each RCM simulation. As expected, and as already observed in [79], the RCM simulations obtained including the facility effects result in a significantly better agreement with the experimental data. Ignition delays are significantly under-predicted if

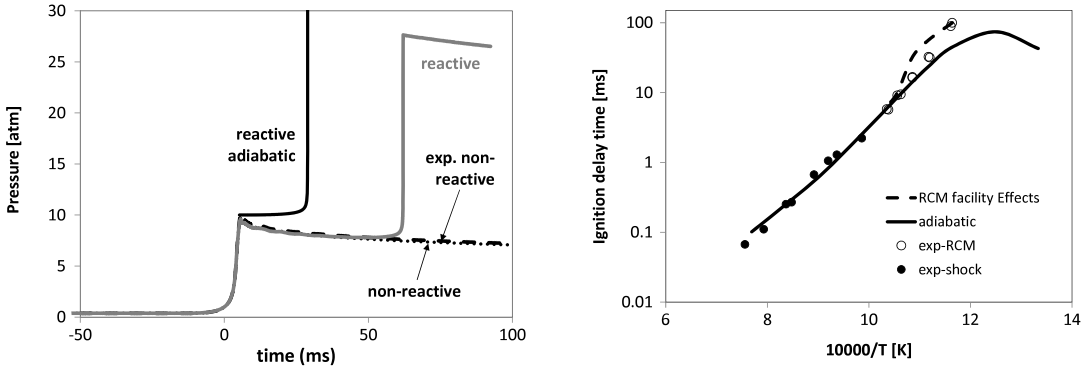

Fig. 21. RCM and Shock tube simulation using OpenSMOKE++ and the kinetic mechanism of Nakamura et al. [79]. Effect of constant-volume and variable volume simulations. Panel (a) Comparison between experimental [79] and predicted pressure profiles during a RCM experiment (butyl-benzene/air at $\Phi = 1$, compressed gas $T =$ 893 K, $P = 10$ atm). Panel (b) Predicted and measured [79] ignition delay times of n-butyl-benzene in shock tube and RCM experiments. The continuous line refers to an isochoric adiabatic batch reactor simulation ($P0 = 10$ atm), the dashed line includes in the simulation the full volume history. Initial mole fractions: fuel = 1.53%, $O2 =$ 20.68%, diluent = 77.79%.

the effect of heat loss is not properly taken into account. For shock tube simulations, constant volume conditions can be usu-ally adopted, but in many cases it is necessary to take into account the pressure rise before ignition which is of the order of $dP/dt = 1 \div 5$ [78–81]. As already discussed by several authors [82,83], these facility effects, which are the consequence of incident-shock attenuation and boundary layer growth, result in an increase in pressure, which can complicate the interpretation of ignition time measurements made at long test times. It is therefore necessary to include the effect of the pre-ignition pressure rise in the simulations when the delay time is longer than ∼1–5 ms [83]. This effect obviously tends to make the calculated ignition delays shorter than a corresponding value obtained without considering the facility-dependent pressure rise, as temperature rises with pressure due to compression [78]. It is also important to underline that the facility-dependent effects on long ignition delay times are opposite when comparing shock tube and RCM ignition delays [78]. Also in the case of a shock tube experiment, it is possible to include facil-ity effects in the simulation using volume profiles that have been deduced from pressure profiles measured in non-reacting experi-ments [78] and assuming an adiabatic compression/expansion pro-cess. The OpenSMOKE++ code supports both alternatives, i.e. it is possible to use the volume history, as in the RCM example of Fig. 21, or to explicitly specify the value of the pressure derivative. This second option can be used only when the pressure rise prior to ig-nition is due to a reasonably constant dP/dt [78]. A more complete discussion on the effect of the pressure rise on the ignition delays can be found in the work of Chaos and Dryer [84].

It is important to underline that it is not possible to reproduce the effects of the roll-up vortex and the thermo-kinetic interactions due to the resulting temperature non-homogeneity which is typically present when non-creviced pistons are adopted. As discussed in detail by Sung and Curran [78], creviced pistons are able to suppress the boundary layer, preventing its entrainment into the reaction chamber of the RCM via a roll-up vortex. Unfortunately, a significant portion of the available ignition delay times in literature were measured with RCMs that did not employ a creviced piston to contain the roll-up vortex. The adequacy of the homogeneous modeling of RCMs without creviced pistons during reactive conditions has been recently investigated by Mittal and Chomier [85] using CFD simulations. They concluded that, since temperature non-homogeneity induced by the roll-up vortex leads to diminution of the NTC behavior, the experimental data from RCMs without creviced piston needs to be taken with caution for quantitative validation and refinement of kinetic mechanisms. Chaos and Dryer [84] concluded that a proper characterization of each apparatus is needed in order to help generate an accurate chemical kinetic models. For this reason it is particularly important to have open codes, such as OpenSMOKE++, where it is possible to introduce the proper reactor model for each apparatus in a flexible way.

7.2. Multi-dimensional laminar flames

As a further illustrative example, the OpenSMOKE++ framework was incorporated into the laminarSMOKE solver [24] for the simulation of multi-dimensional laminar flames. The code has been already used for the simulation of laminar coflow diffusion flames [86–89], premixed flames [25], and to investigate the probe effects on the structure of a low-pressure laminar flame and to assess the deviations from results obtained in unperturbed flames [90]. Here we present only the results obtained for a coflow, axisymmetric, laminar diffusion flame burning prevaporized n-heptane. Additional examples and validation cases are presented in [25]. The fuel stream is a mixture of 3.67% n-heptane and 96.33% nitrogen, by volume. The outer stream is oxygen-vitiated air with 31% of oxygen. The simulated burner (whose geometric configuration is presented in Fig. 22) consists of two concentric nozzles, with diameters of 2.41 mm and 25.4 mm for the fuel and oxidizer streams, respectively. A parabolic, fully developed velocity profile is assumed for the fuel (inner) stream, with an average value of 79 cm/s. A top-hat profile is imposed for the oxidizer (outer) stream, with average speed of 68.7 cm/s. The fuel stream is injected at 470 K, while the oxidizer stream at ambient temperature. Tosatto et al. [91] adopted the same burner geometry to perform experimental and numerical studies of coflow flames fed with JP8 surrogates. In the present work the JP8 was simply replaced with n-heptane, while all the remaining boundary conditions were assumed unchanged. Several detailed kinetic mechanisms, with different number of species and reactions, were adopted to perform the numerical simulations, as reported in Table 7.

As better presented in the following, the investigated laminar flame show a significant lift-off height, H_{lift}. Mohammed et al. [92] demonstrated that the H_{lift} can be correctly predicted only with accurate, detailed kinetic mechanisms. Therefore, the present simulations appear a convenient choice to better point out the different predictive capabilities of the detailed kinetic mechanisms taken into account.

The transport equations of mass, momentum, species and energy are solved in a 2D axisymmetric computational domain. Proper boundary conditions are needed by the equations reported above. In particular, at the inlets of fuel and oxidizer streams, Dirichlet conditions are imposed, to fix the velocity, the temperature and the composition. At the centerline, symmetry conditions

Table 7
Kinetic mechanisms adopted for the numerical simulation of the laminar co-flow flame described in Section 7.2.

Name	Number of species	Peak temp. [K]	Lift-off height [mm]	Ref.
Mauss-47	47	2161	4.8	[93]
Mauss-110	110	2170	4.5	[93]
Lu-88	88	2175	3.6	[17]
Lu-188	188	2136	5.1	[17]
JetSurF 2.0	348	2210	5.8	[94]

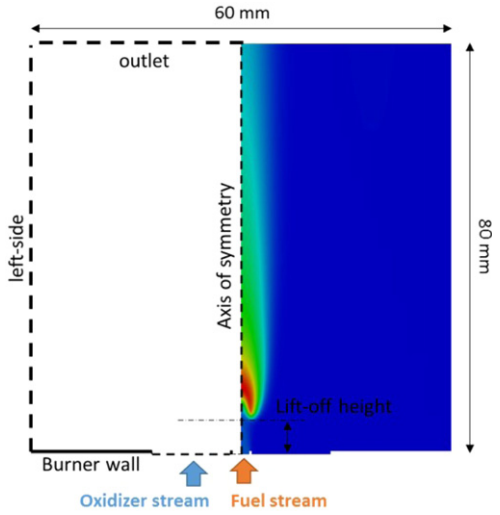

Fig. 22. Coflow, laminar diffusion flames: schematic of the diffusion flame set-up, together with dimensions of the computational domain.

are adopted. Neumann conditions are imposed to model the outflow conditions at the top of the computational domain.

A 2D, 30 × 80 mm rectangular computational domain was adopted, which is sufficiently large to ensure that the boundary conditions do not affect the flame region [91]. A non-uniform, structured mesh with 150 points along the axis and 80 points along the radial coordinate was chosen to discretize the computational domain. This grid was found sufficiently large to ensure that the numerical solution does not depend on the spatial discretization of the computational domain.

The `laminarSMOKE` solver is based on the operator-splitting algorithm, which requires the solution of a transport step (accounting only for convection and diffusion) and of a chemical step (accounting only for the reactions). Here we summarize only its main features, but additional details are presented and discussed in [25]. The transport (convection and diffusion) step is solved using the backward (or implicit) Euler's method. The chemical (or reaction) step (corresponding to a set of independent, stiff ODE systems) is solved using the native `OpenSMOKE++` ODE solver. Only the species and energy equations are solved through the operator-splitting algorithm. Since the continuity and the momentum equations are solved in a segregated approach, in order to ensure the conservation of mass at each time step, the PISO algorithm is applied [95].

Even if the flames here simulated are in steady-state conditions, the `laminarSMOKE` code solves the transport equations in their unsteady form, in order to ensure higher robustness. Thus, the simulations are run over a time interval sufficiently large to reach steady-state conditions. In order to avoid stability issues, the time-step is automatically adjusted during the calculations to have a sufficiently small maximum Courant number. `OpenSMOKE++` was incorporated into the `laminarSMOKE` framework to manage both the transport step (calculation of thermodynamic and transport properties) and the chemical step (integration of independent stiff ODE systems, representing constant-volume adiabatic reactors). An Infiniband platform was used for running all the simulations. In its current configuration, it is made up of 16 nodes, each of them having 36 GB of RAM memory and 12 Intel® Xeon® X5675 (12 Mb cache, 3.06 GHz, 6.40 GT/s Intel® QPI) processors. The simulations reported in the following were performed using a centered spatial scheme and a maximum Courant number of 0.1.

Fig. 23 reports the maps of temperature and mass fractions of selected species, calculated using the JetSurF 2.0 kinetic mechanism (348 species and 2163 reactions). To our knowledge, this is one of the largest kinetic mechanism (in terms of number of species) adopted for the simulation of a laminar coflow flame. The typical features of non-premixed flames can be observed, with fuel inside the flame front and oxidizer (O_2) outside it, together with conversion to CO and H_2 and then to CO_2 and H_2O across the flame front. In Fig. 24 we reported the mass fraction profiles of selected species along the axis of the flame. Because of the flame lift-off, a significant amount of oxygen is able to penetrate into the flame, as evident from the peak at ∼10 mm.

The numerical simulations performed with the other kinetic mechanisms show similar results, and they are not here reported for sake of brevity. The flame lift-off height (here defined as the lowest axial location where the flame reaches the temperature of 1000 K) is a very important property to test the reliability and the accuracy of a kinetic mechanism. In particular, it was demonstrated that the lift-off height is usually strongly dependent on the extinction strain rate of the mechanism [92]. As reported in Table 7, the four kinetic mechanisms employed for the present analysis show different lift-off heights.

The CPU time to perform the simulations reported above, based on the operator-splitting technique, is strongly affected by the complexity (especially the number of species) of the kinetic mechanism adopted. In particular, the total CPU time is split into three main parts: (i) the reaction (or chemical) step, in which the uncoupled, stiff ODE systems are integrated over the chosen time-step; (ii) the evaluation of transport properties (mass diffusion coefficients, thermal conductivity, dynamic viscosity and thermal diffusion coefficients); (iii) the transport (convection and diffusion) step. Fig. 25 reports the computational costs for simulating (on a single processor) a time interval of 2.5 μs for the different kinetic mechanisms in Table 7. The reaction step (i.e. integration of independent stiff ODE systems) results to be a very consuming part of the code, independently of the kinetics, requiring ∼45%–60%of the total computational time. The evaluation of the transport properties covers only ∼1%, while the transport step has a weight which is comparable to the reaction step. The remaining operations represent less than ∼4% of total CPU time. In particular, from Fig. 25 it is evident that the CPU times required by the reaction and transport steps are comparable and increases quadratically (∼1.9) with respect to the number of species. The reaction step and the evaluation of transport properties are entirely performed by the `OpenSMOKE++` library. Since they cover more than 50% of total CPU time, this further explains the interest of a numerical tool like `OpenSMOKE++` able to manage detailed kinetic mechanism with minimization of computational time.

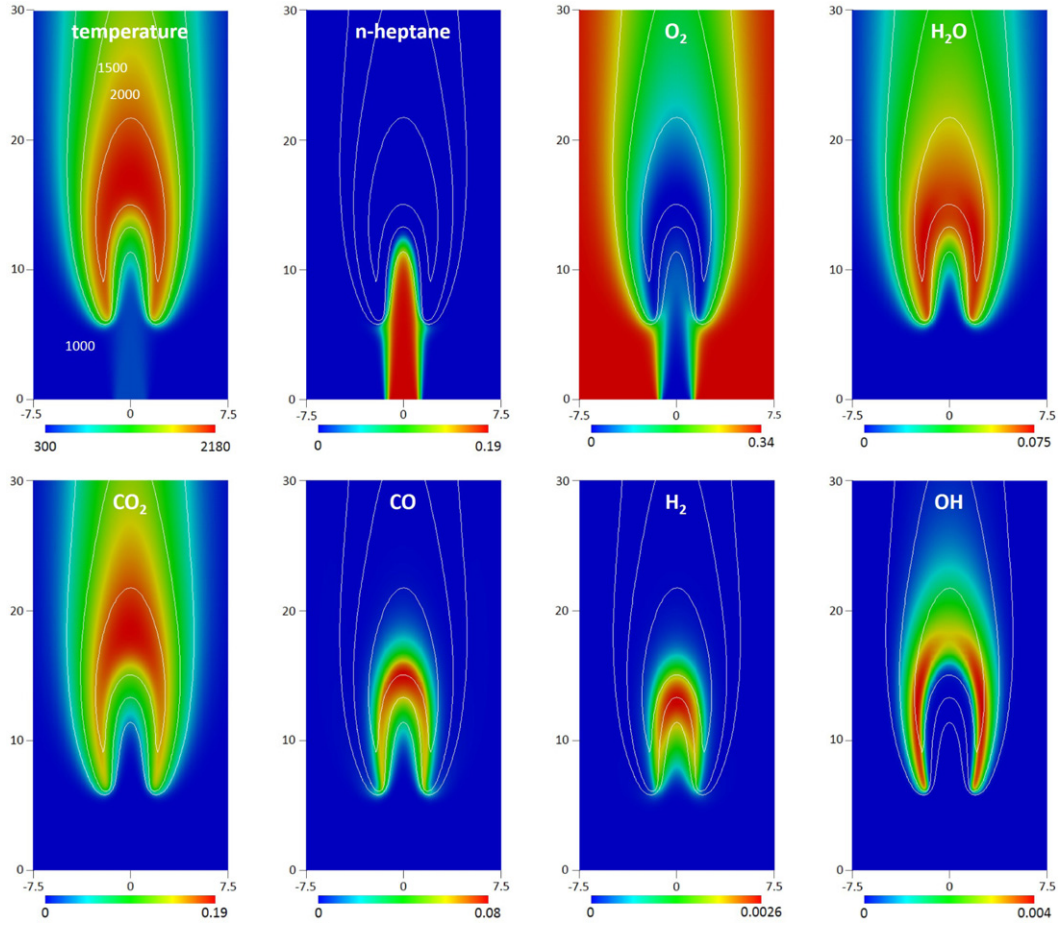

Fig. 23. Laminar, coflow diffusion flame burning n-heptane: calculated maps of temperature and mass fractions of selected species. Iso-contour lines of temperature (1000, 1500 and 2000 K) are reported on each map. The coordinates along the axial and radial directions are in mm. The calculations were performed using the JetSurF 2.0 kinetic mechanism [94].

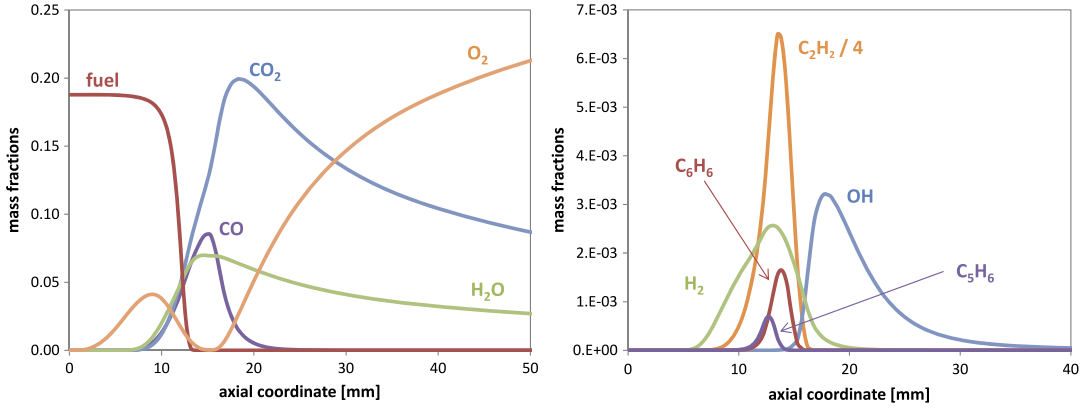

Fig. 24. Laminar, coflow diffusion flame fed burning n-heptane: calculated mass fraction profiles of selected species along the axial coordinate. The calculations were performed using the JetSurF 2.0 kinetic mechanism [94].

8. Concluding remarks

We have presented OpenSMOKE++, a general framework for numerical simulations of reacting systems with detailed kinetic mechanisms. The code, which is entirely written in object-oriented C++, was specifically designed to manage very large detailed kinetic mechanisms (involving thousands of species and reactions) and to be easily customized by the user. The most important technical features of the OpenSMOKE++ framework, together with its innovative aspects with respect to similar codes commonly adopted by the combustion community, were presented and discussed. The computational performances of the proposed framework were discussed through a series of examples, which demonstrated the robustness and the efficiency of OpenSMOKE++ calculations.

We have made the OpenSMOKE++ code freely available, because we believe this will be of great benefit for the scientific community, by giving the researchers the possibility to focus their

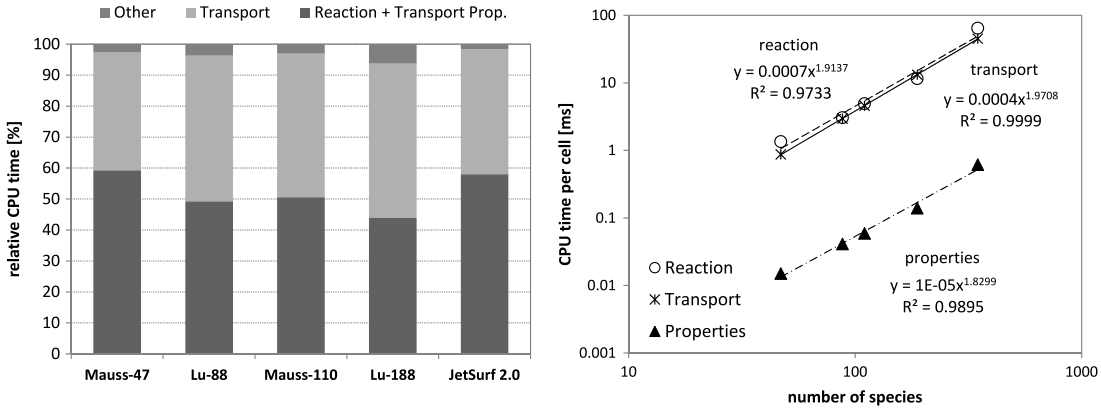

Fig. 25. Laminar, coflow diffusion flame burning n-heptane: analysis of the CPU times for different kinetic mechanisms. Left: relative weights of CPU times for the different phases required by the operator-splitting technique. Right: CPU times per computational cell versus the number of species.

effort on the design of new models, rather than spending time in re-implementing a well-known method.

Acknowledgments

This work has been supported by Regione Lombardia and CILEA Consortium through the LISA Initiative (Laboratory for Interdisciplinary Advanced Simulation) 2013 Grant POLARYS (POllutants in LAminar Reacting sYStems). The authors acknowledge Prof. G. Buzzi-Ferraris (Politecnico di Milano) for his suggestions about the numerical solution of stiff ODE systems.

Appendix A. Reactions available in OpenSMOKE++

In addition to the conventional reaction laws described in Section 3.2, the current version of OpenSMOKE++ offers the possibility to manage a large number of alternative formulations.

Pressure-dependent reactions

Pressure-dependent reactions show a behavior which is directly dependent on the pressure. The effective reaction rate requires the calculation of two modified Arrhenius' laws describing the high- $\left(k^f_{0,j}\right)$ and low-pressure $\left(k^f_{\infty,j}\right)$ limits, which are properly combined together. In particular, the forward kinetic constant is given by:

$$k^f_j = k^f_{\infty,j}\frac{P^j_r}{1+P^j_r}F^{PD}_j \tag{A.1}$$

where F^{PD}_j is a blending function and P^j_r is the reduced pressure $P^j_r = \frac{k^f_{0,j}M_{eff,j}}{k^f_{\infty,j}}$. OpenSMOKE++ provides three different kinetic laws for estimating F^{PD}_j, namely the Lindemann, the Troe and the SRI models. In the Lindemann model F^{PD}_j is simply equal to 1. In the Troe form, F^{PD}_j is given by:

$$\log F^{PD}_j = \left[1+\left(\frac{\log P^j_r + c_j}{n_j - d_j\left(\log P^j_r + c_j\right)}\right)^2\right]^{-1}\log F^{cent}_j \tag{A.2}$$

where the constants c_j, n_j, and d_j have the following expressions:

$$\begin{cases} c_j = -0.4 - 0.67\log F^{cent}_j \\ n_j = 0.75 - 1.27\log F^{cent}_j \\ d_j = 0.14 \end{cases} \tag{A.3}$$

and

$$F^{cent}_j = \left(1-\alpha_j\right)e^{-\frac{T}{T^{***}_j}} + \alpha_j e^{-\frac{T}{T^{*}_j}} + e^{-\frac{T^{**}_j}{T}}. \tag{A.4}$$

The four parameters α_j, T^*_j, T^{**}_j, and T^{***}_j are specific of each pressure-dependent reaction in the Troe form. In the SRI approach the blending function is approximated differently:

$$F^{PD}_j = d_j\left[a_j e^{-\frac{b_j}{T}} + e^{-\frac{T}{c_j}}\right]^{X_j} T^{e_j} \tag{A.5}$$

where

$$X_j = \frac{1}{1+\left(\log P^j_r\right)^2}. \tag{A.6}$$

The five parameters a_j, b_j, c_j, d_j, and e_j are specific of each pressure-dependent reaction in the SRI form.

Chemically activated bimolecular reactions

The kinetic constant of Chemically Activated Bimolecular Reactions is described by the following function:

$$k^f_j = k^f_{0,j}\frac{1}{1+P^j_r}F^{PD}_j. \tag{A.7}$$

The expression reported above is very similar to Eq. (A.1). The main difference is that $k^f_{0,j}$ is the pressure-independent factor, while in (A.1) it is $k^f_{\infty,j}$. The three choices for F^{PD}_j are exactly the same as for the unimolecular fall-off reactions, i.e. the Lindemann, Troe and SRI models.

Pressure-dependent reactions through logarithmic interpolation

The formulation proposed by Miller and Lutz [30] describes the pressure dependence of a reaction based on the direct interpolation of reaction rates specified at individual pressures. In particular, for a given reaction, the usual Arrhenius' parameters are supplied for a set of P_k pressures. Then for a pressure P between P_k and P_{k+1}, the kinetic constant k^f_j is obtained through the following interpolation:

$$\ln k^f_j = \ln k^f_{j,k} + \left(\ln k^f_{j,k+1} - \ln k^f_{j,k}\right)\frac{\ln P - \ln P_k}{\ln P_{k+1} - \ln P_k}. \tag{A.8}$$

The method reported above can be used as an alternative approach to describe any type of pressure dependence (including the formulations reported in the previous sections).

Chebyshev polynomials

The Chebyshev formulation consists in approximating the logarithm of the kinetic constant of a given reaction as a truncate bivariate Chebyshev series in the reverse temperature and logarithm of pressure [30]. First of all, the normalized temperature $\widetilde{T}$ and pressure $\widetilde{P}$ are calculated as reported in the following:

$$\begin{cases} \widetilde{T} = \dfrac{2T^{-1} - T_{\min}^{-1} - T_{\max}^{-1}}{T_{\max}^{-1} - T_{\min}^{-1}} \\ \widetilde{P} = \dfrac{2\log P - \log P_{\min} - \log P_{\max}}{\log P_{\max} - \log P_{\min}} \end{cases} \tag{A.9}$$

where the minimum and maximum values of temperature and pressure have to be established by the user. Then, the corresponding kinetic constant (the subscript j, referring to the index of reaction, is omitted for simplicity) is given by:

$$\log k^f = \sum_{n=1}^{N}\sum_{m=1}^{M} a_{nm}\varphi_n\left(\widetilde{T}\right)\varphi_m\left(\widetilde{P}\right) \tag{A.10}$$

where the Chebyshev polynomials are defined as:

$$\varphi_n(x) = \cos\left(\frac{n-1}{\cos(x)}\right). \tag{A.11}$$

The integers N and M represents the number of basis functions for temperature and pressure, respectively. The $N \times M$ coefficients a of the Chebyshev expansion have to be provided by the user.

Landau–Teller formulation

The generalized Landau–Teller formulation [30] calculates the kinetic constant as:

$$k_j^f = A_j T^{\beta_j} e^{-\frac{E_j^a}{RT} + \frac{B_j}{T^{1/3}} + \frac{C_j}{T^{2/3}}}. \tag{A.12}$$

If the B_j and C_j parameters are set equal to zero, the conventional Arrhenius' expression is recovered.

Janev–Langer–Evans–Post formulation

The formulation proposed by Janev, Langer, Evans and Post [96] is based on a polynomial fit to the logarithm of the temperature and assumes that the kinetic constant can be expressed as reported in the following:

$$k_j^f = A_j T^{\beta_j} e^{\frac{E_j}{T} + \sum_{n=1}^{9} b_{j,n}(\ln T)^{n-1}}. \tag{A.13}$$

The nine $b_{j,n}$ parameters have to be specified by the user.

Power-series formulation

An alternative formulation can be applied, based on a power series within the exponential of a modified Arrhenius' expression [30]:

$$k_j^f = A_j T^{\beta_j} e^{\sum_{n=1}^{4} \frac{b_{j,n}}{T^n}}. \tag{A.14}$$

The $b_{j,n}$ four parameters have to be specified by the user.

Appendix B. Adjustment of thermodynamic coefficients

The thermodynamic properties are calculated using the standard approach already described in Section 3.1. In particular, for each species the constant pressure specific heat is evaluated using a fourth-order polynomial (Eq. (1)), which is defined on two adjacent temperature intervals, respectively from $T_{\min}$ to T_{com} and from T_{com} to $T_{\max}$. Therefore, 10 thermodynamic coefficients are required (5 per interval). Obviously the two sets of coefficients must ensure the continuity of the specific heat at T_{com}. `OpenSMOKE++` checks the continuity, which must be satisfied within a relative error of 0.1%. If the relative error is larger, the user is asked to modify the sets of thermodynamic coefficients to meet the `OpenSMOKE++` requirements.

For most detailed kinetic mechanisms available in the literature, only the continuity of $\widetilde{C}_P$ is ensured at T_{com}. `OpenSMOKE++` allows to apply small adjustments to the thermodynamic coefficients in order to ensure also the continuity of first-, second-, and/or third-order derivatives. This adjustment can result in positive effects during the integration of the ODE systems describing many chemical reactors (see Section 6). The procedure, which can be automatically applied only if the provided thermodynamic coefficients satisfy the continuity requirements described above, is described below.

After defining an intermediate temperature $\overline{T}$ (not necessarily equal to T_{com}), the adjusted specific heats $\widetilde{C}_P^{adj}$ are written as:

$$\begin{cases} \dfrac{\widetilde{C}_P^{adj}}{R} = a_1^{LT} + a_2^{LT}T + a_3^{LT}T^2 + a_4^{LT}T^3 + a_5^{LT}T^4 \\ \quad \text{if } T_{\min} \le T \le \overline{T} \\ \dfrac{\widetilde{C}_P^{adj}}{R} = a_1^{LT} + a_2^{LT}T + a_3^{LT}T^2 + a_4^{LT}T^3 + a_5^{LT}T^4 \\ \quad + \alpha_1\left(T-\overline{T}\right)^4 + \cdots + \alpha_2\left(T-\overline{T}\right)^3 + \alpha_3\left(T-\overline{T}\right)^2 \\ \quad \text{if } \overline{T} < T \le T_{\max}. \end{cases} \tag{B.1}$$

The objective is to find the coefficients $a_1^{LT} - a_5^{LT}$ and $\alpha_1 - \alpha_3$ which best fit (in the least square sense) the original specific heat over the range $T_{\min} \div T_{\max}$. The formulation (B.1) automatically satisfies the continuity of the $\widetilde{C}_P^{adj}$ function and of its first derivative. If α_3 is assumed equal to zero, the resulting formulation ensures also the continuity of the second derivative. Eventually, if both α_2 and α_3 are set equal to zero, the continuity of first-, second- and third-order derivatives is automatically satisfied. Once the coefficients are calculated using the method of least squares, the specific heat in the high-temperature region can be recast in the usual form:

$$\frac{\widetilde{C}_P^{adj}}{R} = a_1^{HT} + a_2^{HT}T + a_3^{HT}T^2 + a_4^{HT}T^3 + a_5^{HT}T^4$$
$$\text{if } \overline{T} < T \le T_{\max} \tag{B.2}$$

where:

$$\begin{cases} a_1^{HT} = a_1^{LT} + \alpha_1\overline{T}^4 - \alpha_2\overline{T}^3 + \alpha_3\overline{T}^2 \\ a_2^{HT} = a_2^{LT} - 4\alpha_1\overline{T}^3 + 3\alpha_2\overline{T}^2 - 2\alpha_3\overline{T} \\ a_3^{HT} = a_3^{LT} + 6\alpha_1\overline{T}^2 - 3\alpha_2\overline{T} + \alpha_3 \\ a_4^{HT} = a_4^{LT} - 4\alpha_1\overline{T} + \alpha_2 \\ a_5^{HT} = a_5^{LT} + \alpha_1. \end{cases} \tag{B.3}$$

The intermediate temperature $\overline{T}$ is chosen in order to minimize the error between the original $\widetilde{C}_P$ and the reformulated $\widetilde{C}_P^{adj}$ specific heats:

$$e = \int_{T_{\min}}^{T_{\max}} \left(\widetilde{C}_P - \widetilde{C}_P^{adj}\right)^2 dT. \tag{B.4}$$

A simple iterative procedure is applied to search the best value.

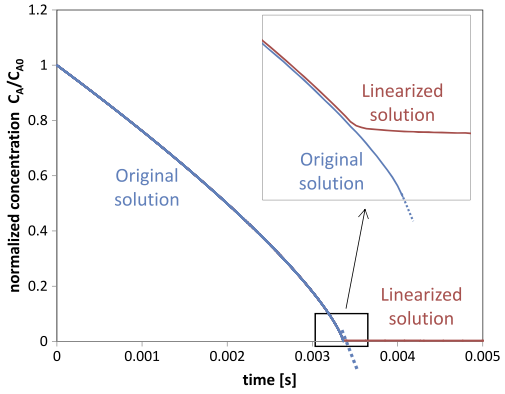

Fig. C.1. Concentration of A with (blue line) and without (red line) the linearization ($C_{A0} = 0.09$ mol/l; $\alpha = -0.3$; $k = 10$ mol$^{0.3}$/l$^{0.3}$/s). The small figure zooms in the zone where the functions approach zero. The constants σ and τ are chosen equal to 23 and 17 respectively.

Appendix C. Reaction orders lower than 1

Non-elementary reactions with arbitrary reaction orders might cause numerical problems when orders lower than 1 are adopted, because of possible negative values of the concentration of the species. The solution which is adopted in `OpenSMOKE++` to circumvent this problem is the linearization of the rate expression when the concentration of the reactants becomes lower than a certain specified value. A simple example can show this approach. The species A is consumed with a reaction rate $r = kC_A^\alpha$, where α is less than one. The mass balance can be then written as:

$$\frac{dC_A}{dt} = -kC_A^\alpha \tag{C.1}$$

whose solution is:

$$C_A = \left[C_{A0}^{1-\alpha} - (1-\alpha)\, kt\right]^{1/(1-\alpha)} \tag{C.2}$$

being C_{A0} the initial concentration of species A. When the time is higher than:

$$t = \frac{C_{A0}^{1-\alpha}}{(1-\alpha)\, k}. \tag{C.3}$$

C_A becomes lower than zero with several problems arising if a numerical solution is adopted. To overcome these difficulties it is possible to identify a small threshold value (C_{AT}) of C_A, below which an order one reaction is assumed: $r = \tilde{k}C_A^\alpha$, where $\tilde{k}$ is estimated making equal the two reaction rates for $C_A = C_{AT}$: $kC_{AT}^\alpha = \tilde{k}C_{AT} \Rightarrow \tilde{k} = kC_{AT}^{\alpha-1}$. The transition between the two reaction rates is obtained through an expression, able to avoid discontinuities in the function and in its derivatives. The final rate constant expression covering the whole time range is then:

$$r = \xi kC_A^\alpha + (1-\xi)\, kC_{AT}^{\alpha-1} C_A \tag{C.4}$$

where ξ is a proper function based on hyperbolic tangent, which allows the continuous transition:

$$\xi = \frac{1}{2}\left[\tanh\left(\sigma\frac{C_A}{C_{AT}} - \tau\right) + 1\right] \tag{C.5}$$

where σ and τ are two constants. Fig. C.1 shows the impact of this approach on the solution. The two results are very similar and only zooming at very low concentrations, it is possible to observe the entity of the correction introduced by the linearization.

Appendix D. Supplementary material

Supplementary material related to this article can be found online.

References

[1] K. Radhakrishnan, AIAA J. 41 (5) (2003) 848–855.
[2] T. Lu, C. Law, Prog. Energy Combust. Sci. 35 (2) (2009) 192–215.
[3] K. Seshadri, T. Lu, O. Herbinet, S. Humer, U. Niemann, W.J. Pitz, R. Seiser, C.K. Law, Proc. Combust. Inst. 32 I (2009) 1067–1074.
[4] E. Ranzi, A. Frassoldati, R. Grana, A. Cuoci, T. Faravelli, A. Kelley, C. Law, Prog. Energy Combust. Sci. 38 (4) (2012) 468–501.
[5] O. Herbinet, W.J. Pitz, C.K. Westbrook, Combust. Flame 154 (3) (2008) 507–528.
[6] C.K. Westbrook, W.J. Pitz, O. Herbinet, H.J. Curran, E. Silke, Combust. Flame 156 (1) (2009) 181–199.
[7] 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, Combust. Flame (2011).
[8] C.K. Westbrook, C.V. Naik, O. Herbinet, W. Pitz, M. Mehl, S.M. Sarathy, H.J. Curran, Combust. Flame 158 (4) (2011) 742–755.
[9] H.J. Curran, P. Gaffuri, W.J. Pitz, C.K. Westbrook, Combust. Flame 114 (1–2) (1998) 149–177.
[10] CRECK Modeling Group 2014 Detailed kinetic mechanisms. URL http://creckmodeling.chem.polimi.it/.
[11] Lawrence Livermore National Laboratories, 2014, Combustion chemistry. URL https://www-pls.llnl.gov/?url=science_and_technology-chemistry-combustion.
[12] Combustion Kinetics (KinCom) 2014 Detailed kinetic models. URL http://lrgp.univ-lorraine.fr/en/research/axe-4-reactions-et-reacteurs/inetique-de-la-combustion.html.
[13] Combustion Chemistry Center, 2014, Reaction mechanisms. http://c3.nuigalway.ie/.
[14] Gas Research Institute, 2014, Gri-mech project. URL http://www.me.berkeley.edu/gri_mech/.
[15] E. Ranzi, M. Dente, A. Goldaniga, G. Bozzano, T. Faravelli, Prog. Energy Combust. Sci. 27 (1) (2001) 99–139.
[16] E. Ranzi, M. Dente, T. Faravelli, G. Pennati, Combust. Sci. Technol. 95 (1994).
[17] T. Lu, C.K. Law, Combust. Flame 144 (2005) 24–36.
[18] F. Bisetti, Combust. Theory Model. 16 (3) (2012) 387–418.
[19] S. Lam, Combust. Sci. Technol. 89 (5–6) (1993) 375–404.
[20] J.F. Grcar, M.S. Day, J.B. Bell, Combust. Theory Model. 10 (4) (2006) 559–579.
[21] A. Stagni, A. Cuoci, A. Frassoldati, T. Faravelli, E. Ranzi, Ind. Eng. Chem. Res. 53 (22) (2014) 9004–9016.
[22] A. Cuoci, A. Frassoldati, T. Faravelli, E. Ranzi, Combust. Flame 156 (10) (2009) 2010–2022.
[23] A. Cuoci, A. Frassoldati, T. Faravelli, E. Ranzi, Proc. Combust. Inst. 35 (2) (2015) 1621–1627. http://dx.doi.org/10.1016/j.proci.2014.06.035.
[24] A. Cuoci, A. Frassoldati, T. Faravelli, E. Ranzi, Combust. Flame 160 (5) (2013) 870–886.
[25] A. Cuoci, A. Frassoldati, T. Faravelli, E. Ranzi, Energy & Fuels 27 (12) (2013) 7730–7753.
[26] M. Maestri, A. Cuoci, Chem. Eng. Sci. 96 (7) (2013) 106–117.
[27] S. Matera, M. Maestri, A. Cuoci, K. Reuter, ACS Catal. 4 (11) (2014) 4081–4092.
[28] A. Cuoci, A. Frassoldati, A. Stagni, T. Faravelli, E. Ranzi, G. Buzzi-Ferraris, Energy and Fuels 27 (2) (2013) 1104–1122.
[29] M. Bissoli, A. Cuoci, A. Frassoldati, T. Faravelli, E. Ranzi, T. Lucchini, G. D'Errico, F. Contino, Detailed kinetic analysis of hcci combustion using a new multi-zone model and cfd simulations, 6 (3), 1594–1609, 2013.
[30] R. Kee, F. Rupley, E. Meeks, J. Miller, Chemkin-III: A Fortran chemical kinetics package for the analysis of gas phase chemical and plasma kinetics, Technical Report, 1996.
[31] H. Weller, G. Tabor, H. Jasak, C., F., Comput. Phys. 12 (6) (1998) 620–631.
[32] B. Stroustrup, The C++ Programming Language, third ed., Addison Wesley, Reading, (MA), 1997.
[33] Y. Dubois-Pelerin, T. Zimmermann, Comput. Methods Appl. Mech. Engrg. 108 (1993) 165.
[34] J. Cary, S. Shasharina, J. Cummings, J. Reynders, P. Hinker, Comput. Phys. Commun. 105 (1997) 20–36.
[35] A. Alexandrescu, Modern C++ Design: Generic Programming and Design Patterns Applied, Addison Wesley Professional, 2001.
[36] M. Smooke, H. Rabitz, Y. Reuven, F.L. Dryer, Combust. Sci. Technol. 59 (1983) 295–319.
[37] M. Gregoire, N. Solter, S. Kleper, Professional C++, Wrox Pr Inc., 2011.
[38] Intel©2014 User's Guide for Intel©Math Kernel Library 11.1.
[39] S. Gordon, B. McBride, Computer Program for Calculation of Complex Chemical Equilibrium Compositions, Rocket Performance, Incidentand Reflected Shocks and Chapman-Jouguet Detonations, Technical Report, 1971.
[40] C. Curtiss, J. Hirschfelder, J. Chem. Phys. 17 (1949) 550.
[41] J. Hirschfelder, C. Curtiss, R.B. Bird, The Molecular Theory of Gases and Liquids, Wiley-Interscience, 1964, Revised edition.
[42] R.B. Bird, W.E. Stewart, E.N. Lightfoot, Transport Phenomena, second ed., J. Wiley, New York, 2002.
[43] C. Wilke, J. Chem. Phys. 18 (1950) 517.
[44] S. Mathur, P. Tondo, S. Saxena, Mol. Phys. 12 (6) (1967) 569–579.
[45] L. Shampine, Numerical Solution of Ordinary Differential Equations, first ed., Chapman and Hall/CRC, 1994.
[46] P. Brown, G. Byrne, A. Hindmarsh, SIAM J. Sci. Stat. Comput. 10 (1989) 1038–1051.
[47] S. Cohen, A. Hindmarsh, Comput. Phys. 10 (2) (1996) 138–143.
[48] P. Brown, A. Hindmarsh, L. Petzold, SIAM J. Sci. Comput. 15 (1994) 1467–1488.
[49] A. Hindmarsh, ODEPACK, A Systematized Collection of ODE Solvers, North-Holland, Amsterdam, 1983, p. 55/64.

[50] E. Hairer, G. Wanner, Solving ordinary differential equations. Stiff and differential-algebraic problems, 2nd ed., in: Springer Series in Computational Mathematics, 14, 1996.
[51] G. Buzzi-Ferraris, D. Manca, Comput. Chem. Eng. 22 (11) (1998) 1595–1621.
[52] G. Buzzi-Ferraris, F. Manenti, Fundamentals and Linear Algebra for the Chemical Engineer: Solving Numerical Problems, Wiley VCH, 2010.
[53] G. Buzzi-Ferraris, F. Manenti, Nonlinear Systems and Optimization for the Chemical Engineer: Solving Numerical Problems, Wiley VCH, 2013.
[54] F. Perini, E. Galligani, G. Cantore, R. Reitz, SAE Technical Papers 8 (2012).
[55] F. Perini, E. Galligani, R.D. Reitz, Combust. Flame 161 (5) (2014) 1180–1195.
[56] M. McNenly, R. Whitesides, D. Flowers, Proc. Combust. Inst. (2014).
[57] J. Miller, M. Branch, W. McLean, D. Chandler, M. Smooke, R. Kee, Proceedings of the Twentieth Symposium (International) on Combustion, The Combustion Institute, 673, 1985.
[58] A. Lutz, R. Kee, J. Miller, Senkin: a Fortran program for predicting homogeneous gas phase chemical kinetics with sensitivity analysis, Technical Report, 1997.
[59] H. Rabitz, M. Kramer, D. Dacol, Annu. Rev. Phys. Chem. 34 (1983) 419–461.
[60] M. Caracotsios, W. Stewart, Comput. Chem. Eng. 9 (4) (1985) 359–365.
[61] E. Gansner, S. North, Softw. Pract. Exp. 30 (11) (2000) 1203–1233.
[62] E. Ranzi, Energy and Fuels 20 (2006) 1024–1032.
[63] R. Kee, M. Coltrin, P. Glarborg, Chemical Reacting Flows: Theory and Practice, Wiley-Interscience, 2003.
[64] H. Mirels, Phys. Fluids 6 (1963) 1201.
[65] D. Goodwin, Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes, 2014. URL http://code.google.com/p/cantera.
[66] A. Hindmarsh, P. Brown, K. Grant, S. Lee, R. Serban, D. Shumaker, C. Woodward, ACM Trans. Math. Softw. 31 (3) (2005) 363–396.
[67] E. Herning, L. Zipperer, Gas und Wasserfach, 79 (1936) 69–73.
[68] F. Perini, E. Galligani, R.D. Reitz, Energy and Fuels 26 (8) (2012) 4804–4822.
[69] T. Lu, C.K. Law, Combust. Flame 154 (1) (2008) 153–163.
[70] O. Herbinet, B. Husson, Z. Serinyel, M. Cord, V. Warth, R. Fournet, P.A. Glaude, B. Sirjean, F. Battin-Leclerc, Z. Wang, M. Xie, Z. Cheng, F. Qi, Combust. Flame 159 (12) (2012) 3455–3471.
[71] P.A. Glaude, O. Herbinet, S. Bax, J. Biet, V. Warth, F. Battin-Leclerc, Combust. Flame 157 (11) (2010) 2035–2050.
[72] N. Cernansky, R. Green, W. Pitz, C. Westbrook, Combust. Sci. Technol. 50 (1) (1986) 3.
[73] P. Gaffuri, T. Faravelli, E. Ranzi, N. Cernansky, D. Miller, A. D'Anna, A. Ciajolo, AIChE J. 43 (5) (1997) 1278–1286.
[74] P. Lignola, F. Di Maio, A. Marzocchella, R. Mercogliano, E. Reverchon, Proc. Combust. Inst. 22 (1988) 1625–1633.
[75] E. Ranzi, T. Faravelli, P. Gaffuri, A. Sogaro, A. D'Anna, A. Ciajolo, Combust. Flame 108 (1–2) (1997) 24–42.
[76] A. Frassoldati, T. Faravelli, E. Ranzi, Int. J. Hydrog. Energy 32 (15) (2007) 3471–3485. (spec. iss.).
[77] E. Petersen, M. Lamnaouer, J. De Vries, H. Curran, J.M. Simmie, M. Fikri, C. Schulz, G. Bourque, Discrepancies between shock tube and rapid compression machine ignition at low temperatures and high pressures, Springer, Berlin (Germany), 2009, pp. 739–744.
[78] C.J. Sung, H. Curran, Prog. Energy Combust. Sci. 44 (2014) 1–18. http://dx.doi.org/10.1016/j.pecs.2014.04.001.
[79] H. Nakamura, D. Darcy, M. Mehl, C. Tobin, W.K. Metcalfe, W. Pitz, C.K. Westbrook, H. Curran, Combust. Flame 161 (2014) 49–64.
[80] A.L. Lapidus, N.A. Gaidai, N.V. Nekrasov, L.A. Tishkova, Y.A. Agafonov, T.N. Myshenkova, Pet. Chem. 47 (2) (2007) 75–82.
[81] H. Wang, S. Warner, M.A. Oehlschlaeger, R. Bounaceur, J. Biet, P.A. Glaude, F. Battin-Leclerc, Combust. Flame 157 (2010) 1976–1988.
[82] H. Li, Z. Owens, D. Davidson, R. Hanson, Int. J. Chem. Kinet. 40 (2008) 189–198.
[83] K. Heufer, R.X. Fernandes, H. Olivier, J. Beeckmann, O. Roehls, N. Peters, Proc. Combust. Inst. 33 (2010) 359–366.
[84] M. Chaos, F.L. Dryer, Int. J. Chem. Kinet. 42 (2010) 143–150.
[85] G. Mittal, M. Chomier, Combust. Flame 161 (1) (2014) 75–83.
[86] A. Cuoci, A. Frassoldati, T. Faravelli, H. Jin, Y. Wang, K. Zhang, P. Glarborg, F. Qi, Proc. Combust. Inst. 34 (1) (2013) 1811–1818.
[87] H. Jin, A. Cuoci, A. Frassoldati, T. Faravelli, Y. Wang, Y. Li, F. Qi, Combust. Flame 161 (3) (2013) 657–670.
[88] H. Jin, A. Cuoci, A. Frassoldati, T. Faravelli, Y. Wang, Y. Li, F. Qi, Combust. Flame 161 (3) (2014) 657–670.
[89] H. Jin, W. Yuan, Y. Wang, Y. Li, F. Qi, A. Cuoci, A. Frassoldati, T. Faravelli, Proc. Combust. Inst. 35 (1) (2015) 863–871. http://dx.doi.org/10.1016/j.proci.2014.05.128.
[90] V. Gururajana, F. Egolfopoulos, K. Kohse-Hoinghaus, Proc. Combust. Inst. 35 (1) (2015), 821–829. http://dx.doi.org/10.1016/j.proci.2014.06.046.
[91] L. Tosatto, B. Bennett, M. Smooke, Combust. Flame 160 (2013) 1572–1582.
[92] R. Mohammed, M. Tanoff, M. Smooke, A. Shaffer, Proc. Combust. Inst. 27 (1998) 693–702.
[93] T. Zeuch, G. Moreac, S.S. Ahmed, F. Mauss, Combust. Flame 155 (2008) 651–674.
[94] H. Wang, E. Dames, B. Sirjean, D. Sheen, R. Tangko, A. Violi, J. Lai, F.N. Egolfopoulos, D. Davidson, R. Hanson, C. Bowman, C.K. Law, W. Tsang, N. Cernansky, D. Miller, R. Lindstedt, Jetsurf version 2.0: A high-temperature chemical kinetic model of n-alkane (up to n-dodecane), cyclohexane, and methyl-, ethyl-, n-propyl and n-butyl-cyclohexane oxidation at high temperatures, 2010. http://melchior.usc.edu/JetSurF/JetSurF2.0.
[95] R. Issa, J. Comput. Phys. 62 (1986) 40–65.
[96] R. Janev, W. Langer, J. Evans, J. Post, Elementary Processesin Hydrogen-Helium Plasmas, Springer-Verlag, New York, 1987.