Implementation of an *in situ* adaptive tabulation technique for efficient computation of combustion chemistry

M.Phil. in Advanced Chemical Engineering

Jean Hélie

Darwin College

August, 2008

University of Cambridge

Department of Chemical Engineering and Biotechnology

Abstract

The ability to accurately model combustion phenomena is crucial in a wide range of engineering applications, from transportation to energy generation. However, realistic numerical simulations of such phenomena are very computationally expensive and progress has been hitherto severely hindered by their computational cost. This is due to both the size and the complexity of the systems to solve since the description of even a simple combustion model typically involves dozens of chemical species, hundreds of chemical reactions and energy and transport equations. In particular, combustion chemistry has proven to be the major bottleneck in the calculations because of the wide range of characteristic time-scales leading to very stiff systems of equations. In order to minimise the associated computational cost, several numerical schemes have been developed and storage and retrieval methods have become a popular approach. Among such methods, the *in situ* adaptive tabulation (ISAT) technique offers the advantage of achieving speed-up factors of up to three orders of magnitude while still providing accurate error control on the solutions retrieved. The work presented here focused on the implementation of ISAT as an independent piece of code which can be called by external software. The mathematical issues associated with the implementation of the ISAT algorithm were exposed and thoroughly addressed. A corresponding C++ program was written and then validated by sending it mock composition queries. The results obtained demonstrated the functionality of the program, thus validating the mathematical work. However, further work is needed in order to characterize the speed-up achieved.

Contents

1 Introduction

Combustion systems are one of the most common source of energy and have a wide range of applications, electricity generation and engines for ground or air transportation being among the most important. Yet, due to the inherent complexity of the phenomenon, many aspects of combustion systems are very often not completely mastered nor understood. An improved understanding of combustion phenomenon is therefore desirable as it could lead to higher efficiency systems. Moreover, with issues such as pollution and global warming likely to put more and more constraints on the design of fuels, reactors and engines, a better understanding of the chemical processes occuring during combustion will help to achieve pollutant reduction.

Accurate numerical simulations of combustion systems are thus a desirable tool, as they are less expensive than experimental investigations and allow for an easy study of the effects of modications in geometry, operating conditions or fuel composition.

However, combustion problems involve many physical and chemical processes occurring simultaneously and are challenging to model: the description of even a simple model will typically involve dozens of chemical species, hundreds of chemical reactions as well as energy and transport equations. Analytical solutions exist for extremely simple situations but these are limited to the case of simple fuel composition in a laminar flame in zero or one-dimensional space while in most industrial applications (e.g. gas turbines or gasoline engines) combustion occurs in a turbulent flow (so as to improve the efficiency of the mixing process between the fuel and oxidizer) and involves many more species. For such complex problems, despite the advances in computing power, the cost of carrying out calculations including a realistic description of the combustion chemistry still hinder the actual feasibility of the simulation in computational flow dynamics. Detailed large scale simulation of complex combustion problems can only be performed on massively parallel computers and are still out of reach for everyday laboratory use, not to mention industrial applications. Though Moore's law seems to predict that ever more computing power will be available, the development of more subtle numerical algorithms, by making better use of the already resources, can considerably speed up progress in those fields whose development is currently hindered by calculation capacity limitations.

The evaluation of chemical source terms is particularly computationally expensive due to the large size of the chemical system and the wide range of characteristic time-scales which span several orders of magnitude: this leads to large stiff systems which have to be solved repeatedly for each cell of the meshing. This issue has motivated the search

for computational techniques that would reduce the burden imposed on CPU by reactive flow calculations when using standard direct integration methods. It is in this context that an *in situ* adaptive tabulation technique (ISAT) was developed by Pope [1]: its goal is to speed up the time necessary to evaluate the reaction term in the system of equations describing the full combustion problem while still providing accurate error control. This computational technique enables researchers as well as the industry to gain access to the chemical composition evolution of combustion systems which were hitherto too complex to be solved with traditional methods, thus helping to improve our understanding of combustion phenomenon as well as facilitating the conception and development of better design for new engines and combustors. Beyond the scope of combustion chemistry calculations, the principle of the ISAT technique could also be applied to other reacting flows or any fields where the solution of large, computationally expensive differential equations systems is required numerous times.

This thesis focuses on the implementation of the ISAT method as an independent piece of code which can be called by external software. A description of the method's principle is presented first, followed by a presentation of the resolution of the key mathematical points associated to it. The architecture and structure of the code written is then described, explained and validated in the following sections.

2 Literature review

Turbulent reactive flows, and combustion processes in particular, are at the heart of a wide variety of engineering disciplines. It is therefore not surprising that there is a huge amount of literature dedicated to the modelling of such complex processes. The available literature shows how much the complexity and accuracy of those models are dependent on the available computing power and a recent review by Westbrook *et al.* [2] of the historical developments in this field provides an excellent overview of the link between the performance of numerical combustion models and that of computers.

The literature review shows evidence that early work was primarily focused on the development of solvers able to compute efficiently the solutions to the set of ordinary differential equations (ODE) describing combustion problems. Indeed, to ensure stability of the numerical scheme, classical explicit methods are limited to time steps no bigger than the fastest chemical time-scale which requires an enormous amount of CPU time and makes them excessively slow for stiff problems (since the time for the system to reach equilibrium is dictated by the largest chemical time-scale). Because time-scales

in combustion chemistry can vary over several orders of magnitude the ability to solve such ODE systems was a prerequisite to the development of accurate models. The search for more efficient techniques led to the production of much literature on stiff ODE solvers (see the work of Byre and Hindmarsh [3] for a detailed review on this topic) and produced results directly applicable to combustion chemistry models [4]. This enabled scientists working on combustion problems to make a big step forward in their understanding of the basic combustion processes [2].

A review of the available literature also highlights the crucial importance of fractional steps, or operator splitting, methods [5, 6] in the development of computational combustion. In these methods, the set of equations describing the different processes occurring during combustion are solved separately, and an approximation of the overall solution is constructed by an adequate combination of the sub-solutions. Operator splitting techniques have become widely popular [2, 7] as they enable scientists to focus on sub-models dealing with only one aspect of the physics present, thereby alleviating the task at hand. From a computational point of view they offer the invaluable advantage of leading to a drastic reduction in terms of memory and CPU time requirements as they reduce the size of the systems to solve by de-coupling the equations and allow for tailor-made integration techniques for each process involved [7]. By breaking the extremely complex turbulent combustion problems in simpler blocks, operator splitting techniques make it possible to compute their solutions. This is a crucial point which can be illustrated as follows.

If $\mathcal{F}(U, \psi, x, t)$ represents the state of the overall system (e.g. U is the flow velocity and ψ its chemical composition) at a given point (x, t) in space and time, then the set of equations describing its evolution can for instance be written as

$$\frac{\partial \mathcal{F}}{\partial t} = (P_1 + P_2 + P_3)\mathcal{F} \tag{1}$$

where P_1, P_2 and P_3 are operators representing different processes (e.g. advection, diffusion and chemical reactions). While these processes all occur simultaneously in reality, it has been shown that the splitting techniques allow to consider their influence on the evolution of $\mathcal{F}$ one at a time. The utilisation and justification of such a scheme is very well documented in the work of Pope on probability density function (PDF) [8].

While the development of stiff ODE solvers and operator splitting techniques allowed for substantial progress in turbulent combustion modelling these methods alone have their limits and their direct application, often referred to as direct integration (DI), to large problems including detailed chemical mechanisms still requires an exceedingly large amount

of computer power and time which makes them unsuitable as an everyday modelling tool. The evaluation of the chemical source terms in realistic chemical kinetics models including many species and reactions is often the most computationally expensive step and the bottleneck of the calculations [1, 9, 10]. This can be explained by the fact that in a typical problem the meshing will contain about 10^6 cells for each of which the solution usually has to be advanced at least a 1000 times therefore meaning the set of chemical equations has to be solved in the order of 10^9 times [1, 9]. The time needed to solve the system formed by the chemical reactions obviously greatly depends on its size and complexity as well as on the machine performing the calculations; however given the number of times the solution is needed it is easy to see how complex problems can quickly lead to computing times of several weeks or more.

This issue has been at the origin of a vast amount of research on numerical schemes minimising the computational cost needed to solve the equations governing the chemical system. In other words, if P_3 in equation (1) represents the reaction term, a lot of recent research focused on efficient computational methods for problems involving the frequent solving of the system

$$\frac{\partial \mathcal{F}}{\partial t} = P_3 \mathcal{F}. \tag{2}$$

Review of the work produced shows that there are essentially two types of techniques: dimension reduction and storage/retrieval methods. Dimension reduction methods are all based on a drastic reduction of the number of degrees of freedom in the description of the chemical system thanks to the introduction of certain assumptions (see the work of Brad *et al.* [11] for a detailed and recent application of such techniques to combustion chemistry). These methods were not within the scope of this thesis and the reader is invited to refer to the extensive literature on this subject for more information on the main available reduction techniques (quasi steady state assumptions (QSSA) [12, 13], rate-controlled constrained equilibrium (RCCE) [14, 15] and intrinsic low-dimensional manifolds (ILDM) [16, 17]).

The ISAT method on which the work presented here is based belongs to the storage and retrieval methods. The review of the literature available on these methods shows that they have become widely used and that many different techniques have been developed among which the principals are structured look-up tabulation (LUT) [18], artificial neural networks (ANN) [19, 20], repro-modelling [21], piecewise reusable implementation of solution mapping (PRISM) [22] and high dimensional model representations (HDMR) [23].

Such techniques are based on the creation of a table of integrated solutions which can then be used for interpolation. The most important criteria when assessing the computational performance of theses storage and retrieval techniques are the amount of memory and CPU time necessary to create the table and to retrieve a solution as well as the accuracy of the retrieved solutions. The main drawbacks traditionally associated to most of these storage based method are due to the memory requirements of the tabulated space. Indeed since the region of the accessible composition space which is actually accessed is *a priori* unknown, the entire accessible space has to be tabulated and this leads to very large tables. Besides the retrieval work in the case of large tables can be resource consuming from a computing point of view. The efficiency of these tabulation techniques thus decreases as the dimension of the composition space (i.e. the number of variables, which is directly linked to the detail level of the chemistry model) increases.

The *in situ* adaptive tabulation method (ISAT) developed by Pope [1] overcomes most of these drawbacks. Besides, along with lighter computer resources demands, one of the great asset of ISAT is the ability to control the error on the solution retrieved. Since its original publication a decade ago the method has had a substantial impact in the world of computational techniques and has triggered an impressive number of publications. Various successful implementations with satisfying error control and speed-up factors ranging from one to three order of magnitudes have been reported [10, 24–26], several improvements have been proposed [27–29] and the combination of ISAT with reduction methods [30] or other storage/retrieval methods [31] have been explored. Successful adaptations of ISAT in other fields than computational combustion have also been reported [9, 32–35].

The work presented here focused on the implementation of the original ISAT algorithm, as described in [1]. The objective was, without discarding any of the modifications which may have been explored in the literature, to write a general functional scheme based on the *in situ* adaptive tabulation idea. It is thus important to bear in mind that the points addressed in this thesis deal with the *practical implementation* of the concepts and ideas constituting the ISAT method.

3 The *in situ* adaptive tabulation technique

3.1 Problem formulation

Let us consider a reactive gaseous flow. At any point in time and space its chemical composition can be characterized by the mass fractions Y_k, where $k \in \{1, 2, ..., n_s\}$, of

the n_s species present in the flow. If there are n_l linear relationships between those mass fractions (for instance their sum must be equal to 1) then the number of degrees of freedom D of this chemical system is $D = n_s - n_l$.

At time t the chemical composition of any computational particle (i.e. any cell of the meshing used to discretize the domain occupied by the fluid) can thus be described by the vector $\Phi(t)$ of length D

$$\Phi(t) = [\phi_1, \phi_2, ..., \phi_D]\,. \tag{3}$$

As explained in the previous section, several processes are responsible for change in particle composition but each of them can be treated in a separate sub-step. The evolution of composition due to chemical reactions can be represented by

$$\frac{d\Phi(t)}{dt} = \mathbf{S}(\Phi(t)) \tag{4}$$

where the i^{th} component of $\mathbf{S}$ represents the rate of change of the i^{th} component of $\Phi(t)$ and is determined by the system chemical kinetics. Integration of equation (4) for a time interval Δt from the initial condition $\Phi_{initial} = \Phi(t_0)$ will give $\mathbf{R}(\Phi_{initial}) = \Phi(t_0 + \Delta t)$ which is referred to as the reaction mapping and represents particle composition at time $t_0 + \Delta t$. Solving the system chemistry thus requires to determine $\mathbf{R}(\Phi)$ for every cells of the grid on every time-step.

The task at hand is therefore the efficient computation of the reaction mapping (i.e. integration of equation (4)) for a very large number of different initial conditions. For simplicity sake the time interval Δt is kept constant.

3.2 Method description

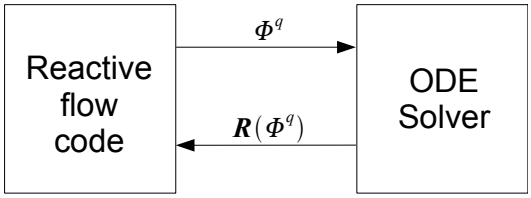
The purpose here is to briefly explain how ISAT works and to introduce the key points of the method which are addressed in detail in section 4. For a more detailed description of how the technique was developed the reader is invited to refer to Pope's original publication [1].

3.2.1 In situ tabulation

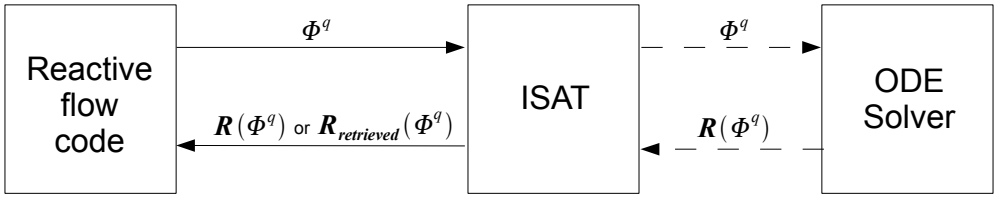
It stems from equation (3) that the composition space is a D-dimensional space. All the physically possible values of Φ (for instance the mass fractions must be within the interval $[0, 1]$) define a subset of the composition space which is referred to as the *realizable region*.

However the set of compositions Φ that actually occur in the flow studied only represents a small region of this realizable region which is referred to as the *accessed region*.

Unlike classic tabulation techniques which tabulate the entire realizable region in a pre-processing stage, ISAT only tabulates the accessed region which is computationally more efficient and has no impact on precision since any point out of the accessed region is irrelevant and useless. However, the accessed region is by definition unknown prior to running the calculations and therefore the ISAT table cannot be constructed during a pre-processing stage. Instead it is progressively built up during the calculation, as illustrated in Figure 1.



(a) *Classic computation of the reactive flow without using any scheme (Direct Integration).*



(b) *Computation of the reactive flow using the ISAT scheme: the ODE solver is not systemically called (which is represented by dashed lines) as solutions can be retrieved from the table built by ISAT.*

Figure 1: *Schematic representation of the interaction between a reactive flow code, an ODE solver and the ISAT algorithm.*

As the calculation proceeds the reactive flow code sends composition queries Φ^q (one on every time-step for every cell of the grid, as explained in 3.1) to the ISAT algorithm which simultaneously builds the table and returns the corresponding reaction mappings $\mathbf{R}(\Phi^q)$. This ensures that the entries of the ISAT table correspond to compositions occurring in the flow and it is referred to as *in situ* tabulation.

3.2.2 Table structure

The table is constructed as a binary tree with each leaf containing information about the reaction mapping for particular compositions that occurred in the flow. The leaves are referred to as *records* and for each record the information is made up of four elements:

- The tabulation point, referred to as Φ^0.
- The corresponding reaction mapping, $\mathbf{R}(\Phi^0)$.
- Information for the retrieval of $\mathbf{R}(\Phi^q)$ for a query Φ^q close to Φ^0.
- Information on the region of the composition space centered on Φ^0 where such a retrieval is considered valid given a specified error tolerance. This region basically contains the queries Φ^q close enough to Φ^0 so that $\mathbf{R}(\Phi^q)$ can be retrieved from $\mathbf{R}(\Phi^0)$ without exceeding the error tolerance. It is referred to as the *region of accuracy* (ROA) and is detailed in 4.1.

The internal nodes of the tree (i.e. all the tree nodes which are not leaves) are referred to as *cutting planes* and contain information on how to correctly traverse the table given a query composition Φ^q. This is explained in 4.2.

3.2.3 Algorithm overview

When the first query is sent to ISAT the ODE solver has to be called as at this stage the table is completely empty. The exact solution is thus always returned for the first query and the tree is initialized with the creation of the corresponding leaf. For the next query however, as well as for all the following ones, ISAT first tries to use the information contained in the table to return an approximate solution: if this cannot be done without exceeding the specified error tolerance the ODE solver is called and the table is updated accordingly.

For all the queries Φ^q but the first one, the ISAT algorithm is thus as follows:

1. The tree is traversed until a leaf (associated to a tabulation point referred to as Φ^0) is reached. The tree is constructed (see 4.4.2 and 4.4.3) and traversed (see 4.2) in such a way that Φ^q and Φ^0 should be close in the composition space.
2. It is determined whether Φ^q is within Φ^0 record ROA (see 4.3.2).
3. If Φ^q is within the ROA, information stored in Φ^0 record is used to calculate and return an approximation of $\mathbf{R}(\Phi^q)$. This scenario is called **Retrieve** (see 4.2.1).
4. If on the contrary Φ^q is outside the ROA, a direct integration is performed: the ODE solver is called and the exact solution $\mathbf{R}(\Phi^q)$ is calculated. The approximate solution is evaluated too and the local error ϵ between the exact and the retrieved solution is calculated (see 4.2.2).

5. If ϵ is smaller than the specified tolerance ϵ_{tol}, the exact solution $\mathbf{R}(\Phi^q)$ is returned and the ROA in Φ^0 record is modified so as to include Φ^q. This scenario is referred to as a **Growth** (see 4.5).

6. If on the contrary $\epsilon > \epsilon_{tol}$, the exact solution $\mathbf{R}(\Phi^q)$ is returned and a record associated to Φ^q is created and the tree is modified accordingly by replacing the Φ^0 record by a cutting plane whose children are the Φ^0 record and the Φ^q record. This scenario is referred to as an **Addition** (see 4.4).

The ISAT algorithm thus basically consists of a tree traversing routine and three main functions (**Retrieve**, **Grow** and **Add**) responsible for returning the reaction mapping for a given composition query and updating the table.

4 Mathematical and technical aspects

In this section we explain how the algorithm is carried out in practice: the mathematical mechanisms used to perform the tasks constituting each scenario are detailed as well as the technical aspects of browsing and updating the binary tree. In this regard, the reader might find valuable to remember that the composition vectors Φ^q or Φ^0 are merely points belonging to a D-dimensional space.

It should be stressed that most of the content of the first three sections (4.1 to 4.3) was developed by Pope in [1]. It is presented here both for completeness and by necessity as it introduces hypotheses required later. On the other hand, sections 4.4 and 4.5 address crucial technical parts of the implementation of ISAT which were mentioned but not covered in [1].

4.1 Browsing of the tree

Each internal node of the tree (i.e. each cutting plane) has exactly two children, referred to as the left child and the right child. A cutting plane is defined by two elements: a vector $\mathbf{v}$ of length D and a scalar a. The couple $(\mathbf{v}, a)$ is used to divide the composition space in two according to the following rule: if $\mathbf{v}^T\Phi^q > a$ then Φ^q is on the right of the cutting plane, otherwise it is on the left. Every time it receives a composition query Φ^q, ISAT traverses the tree starting from its root: the scalar product $\mathbf{v}^T\Phi^q$ is calculated and if it is greater than a then the algorithm proceeds with the right child of the cutting plane and with the left one otherwise.

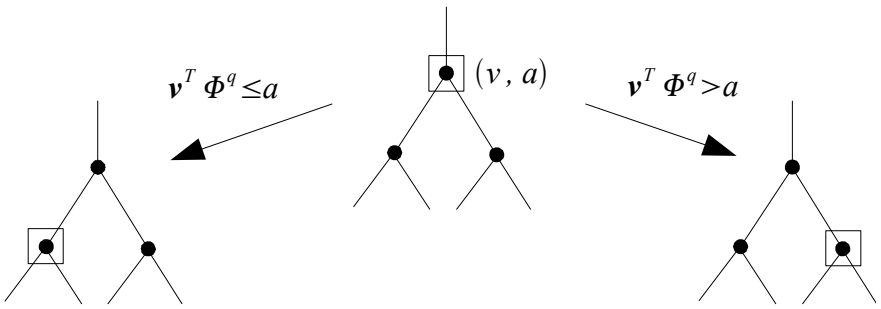


Figure 2: *Schematic representation of how the tree is traversed when ISAT receives a query Φ^q. The nodes in square represents the current position reached in the tree.*

This process, illustrated by Figure 2, is continued until a leaf is reached (that leaf is from now on referred to as the Φ^0 record). The issue of how to determine the couple $(\mathbf{v}, a)$ when a new cutting plane is created is addressed in section 4.4.3.

4.2 Retrieval of a reaction mapping

For a given composition query Φ^q this section describes how the reaction mapping $\mathbf{R}(\Phi^q)$ is approximated in the case that Φ^q belongs to the ROA associated to Φ^0 and how the error associated to this approximation is measured.

4.2.1 Linearized mapping

We note $\delta\Phi$ the displacement between Φ^q and Φ^0:

$$\delta\Phi = \Phi^q - \Phi^q. \tag{5}$$

We also define the *mapping gradient* as the DxD matrix $\mathbf{A}$ representing the sensitivity of the reaction mapping to the components of the composition query:

$$A_{ij}(\Phi) = \frac{\partial R_i(\Phi)}{\partial \Phi_j}. \tag{6}$$

Using a first order development in Taylor series for the reaction mapping we can then write:

$$\mathbf{R}(\Phi^q) = \mathbf{R}(\Phi^0 + \delta\Phi) = \mathbf{R}(\Phi^0) + \mathbf{A}(\Phi^0)\delta\Phi + O(|\delta\Phi|^2). \tag{7}$$

A linear approximation of $\mathbf{R}(\Phi^q)$, noted $\mathbf{R}^l(\Phi^q)$, can then be obtained by neglecting the second order term in equation (7):

$$\mathbf{R}(\Phi^q) \approx \mathbf{R}^l(\Phi^q) = \mathbf{R}(\Phi^0) + \mathbf{A}(\Phi^0)\delta\Phi. \tag{8}$$

Therefore, in addition to Φ^0 and $\mathbf{R}(\Phi^0+\delta\Phi)$, it is sufficient to store $\mathbf{A}(\Phi^0)$ in the record associated to the composition Φ^0 to be able to retrieve the reaction mapping $\mathbf{R}^l(\Phi^q)$ for a composition query Φ^q within Φ^0 ROA.

4.2.2 Measure of the approximation error

The error made when retrieving a reaction mapping from a table entry is characterized by the difference between the exact mapping $\mathbf{R}$ and the linearized mapping $\mathbf{R}^l$.

For a given query Φ^q, if we define δR as the displacement between the two exact reaction mappings $\mathbf{R}(\Phi^q)$ and $\mathbf{R}(\Phi^0)$,

$$\delta R = \mathbf{R}(\Phi^q) - \mathbf{R}(\Phi^0). \tag{9}$$

the difference between $\mathbf{R}(\Phi^q)$ and $\mathbf{R}^l(\Phi^q)$ is, based on equation (8):

$$\mathbf{R}(\Phi^q) - \mathbf{R}^l(\Phi^q) = \mathbf{R}(\Phi^0) + \delta R - (\mathbf{R}(\Phi^0) + \mathbf{A}(\Phi^0)\delta\Phi), \tag{10}$$

i.e.

$$\mathbf{R}(\Phi^q) - \mathbf{R}^l(\Phi^q) = \delta R - \mathbf{A}(\Phi^0)\delta\Phi. \tag{11}$$

The local error ϵ is represented by the magnitude of this difference. However, in order to take into account the possible different order of magnitudes between the components of the reaction mapping a scaling matrix $\mathbf{B}$ is introduced and the local error ϵ is defined and calculated as follows:

$$\epsilon = |\mathbf{B}(\delta R - \mathbf{A}(\Phi^0)\delta\Phi)|, \tag{12}$$

where $|...|$ is the Euclidean norm defined by

$$\forall X \in \mathbb{R}^N, \;\; |X| = \left(\sum_{i=1}^{N} x_i^2 \right)^{\frac{1}{2}}. \tag{13}$$

4.3 Region of accuracy

4.3.1 Hypotheses

The region of accuracy (i.e. the region of the composition space where it is considered legitimate to retrieve the reaction mapping as described in 4.2.1) is assumed to be a hyper-ellipsoid and is now referred to as the ellipsoid of accuracy (EOA). An hyper-ellipsoid is merely an extension of the concept of ellipse to higher dimensional spaces and can be represented in a similar way in a cartesian system of coordinates: if $X = [x_1, x_2, ..., x_n]$ verifies

$$\frac{x_1^2}{l_1^2} + \frac{x_2^2}{l_2^2} + ... + \frac{x_n^2}{l_n^2} \leqslant 1, \tag{14}$$

then X is inside the hyper-ellipsoid centered on the origin and for which the half-lengths of the principal axes are l_i. An hyper-ellipsoid $\mathcal{E}$ can also be defined by a positive-definite matrix $\mathbf{M}$ (i.e. a symmetric matrix for which $\forall X \in \mathbb{R}^N \setminus \{0\},\ X^T\mathbf{M}X > 0$) with the following equation:

$$X \in \mathcal{E} \iff X^T\mathbf{M}X \leqslant 1, \tag{15}$$

In ISAT the matrix $\mathbf{M}$ describing the EOA associated to a given tabulation point Φ^0 is chosen to be:

$$\mathbf{M} = \widetilde{\mathbf{A}}^T\mathbf{B}^T\mathbf{B}\widetilde{\mathbf{A}}/\epsilon_{tol}^2. \tag{16}$$

where $\widetilde{\mathbf{A}}$ is a modification (see 4.3.2 below for explanation) of the mapping gradient matrix defined in 4.2.1, and $\mathbf{B}$ and ϵ are respectively the scaling matrix and the local error defined previously in 4.2.2. The motivation for such an expression is based on an estimation of the local error ϵ in the simplified case where a constant approximation ($\mathbf{R}^c(\Phi^q) = \mathbf{R}(\Phi^0)$) instead of a linear one is used to retrieve an estimate of $\mathbf{R}(\Phi^q)$ (see [1] for details).

4.3.2 Characterization

Since $\widetilde{\mathbf{A}}^T\mathbf{B}^T\mathbf{B}\widetilde{\mathbf{A}}/\epsilon_{tol}^2 = \left(\mathbf{B}\widetilde{\mathbf{A}}/\epsilon\right)^T\left(\mathbf{B}\widetilde{\mathbf{A}}/\epsilon\right)$, $\mathbf{M}$ is a definite positive matrix and therefore according to the spectral theorem there are an *orthogonal* matrix $\mathbf{Q}$ (i.e. a square matrix verifying $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$) and a *diagonal* matrix $\mathbf{\Lambda}$ whose coefficients λ_i are all strictly positive, verifying:

$$\mathbf{Q}^T\mathbf{\Lambda}\mathbf{Q} = \widetilde{\mathbf{A}}^T\mathbf{B}^T\mathbf{B}\widetilde{\mathbf{A}}/\epsilon_{tol}^2. \tag{17}$$

The matrix $\mathbf{Q}$ is a change of basis matrix from $\mathbb{R}^D$ canonic basis to a basis which corresponds to the directions of the principal axes of the EOA while the diagonal coefficients

λ_i of $\mathbf{\Lambda}$ and the half-lenghts l_i of the EOA principle axes are related as follows:

$$l_i = \frac{1}{\sqrt{\lambda_i}} \tag{18}$$

Equation (18) justifies the need to use the modified matrix $\widetilde{\mathbf{A}}$. Indeed, if the mapping gradient matrix $\mathbf{A}$ were to be used directly in equation (16), some of the λ_i could potentially be very small (or even null) thus generating extremely large principal axes (meaning the EOA would stretch unduly in one or several directions of the composition space). To address this issue $\widetilde{\mathbf{A}}$ is constructed by modifying the singular value decomposition (SVD) of $\mathbf{A}$. If the SVD of $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{19}$$

where $\mathbf{\Sigma}$ is a diagonal matrix whose coefficients are noted σ_i, then $\widetilde{\mathbf{A}}$ is defined by:

$$\widetilde{\mathbf{A}} = \mathbf{U}\widetilde{\mathbf{\Sigma}}\mathbf{V}^T \tag{20}$$

where the coefficients of the diagonal matrix $\widetilde{\mathbf{\Sigma}}$ are defined as follows:

$$\widetilde{\sigma_i} = \max\{\sigma_i, \frac{1}{2}\}. \tag{21}$$

Since $\mathbf{B}$ is a scaling matrix (i.e. a diagonal matrix with strictly positive elements), this method not only ensures that $\widetilde{\mathbf{A}}^T\mathbf{B}^T\mathbf{B}\widetilde{\mathbf{A}}/\epsilon_{tol}^2$ is actually a definite positive matrix and thus that equation (18) is legit (since all the λ_i are strictly positive) but also that the EOA can only stretch so much in any direction of the composition space (since the l_i have an upper bound).

The matrices $\mathbf{Q}$ and $\mathbf{\Lambda}$ as defined by equation (17) are therefore sufficient to characterize $\mathcal{E}(\Phi^0)$, the EOA associated to a given tabulation point Φ^0, and provide a simple criterion to determine whether or not a query $\Phi^q = \Phi^0 + \delta\Phi$ is *within* the region of accuracy of Φ^0:

$$\Phi^q \in \mathcal{E}(\Phi^0) \Longleftrightarrow \delta\Phi^T\mathbf{Q}^T\mathbf{\Lambda}\mathbf{Q}\delta\Phi \leqslant 1. \tag{22}$$

4.4 Addition of a record to the ISAT table

This section explains how the addition scenario described in section 3.2.3 has been implemented. As explained in 3.2.3, a new record (designated by Φ^q) has to be created and added to the ISAT table when the one that was reached after traversing the tree (referred to as the *old record* and designated by Φ^0) could not be used to retrieve the reaction mapping

because the query point was outside its EOA and the difference between the linearized and exact mapping was too important (i.e. exceeding ϵ_{tol}) to grow that EOA.

Thanks to the previous sections we know that all of the following elements must be determined in order to create a record:

- Φ, the tabulation point.
- $\mathbf{R}(\Phi)$, the exact reaction mapping.
- $\mathbf{A}(\Phi)$, the mapping gradient.
- $\mathbf{Q}$, an orthogonal matrix associated to the EOA.
- $\mathbf{\Lambda}$, a positive definite diagonal matrix associated to the EOA.

While Φ is supplied to ISAT and $\mathbf{R}(\Phi)$ is determined by solving equation (4) with the ODE solver , $\mathbf{A}(\Phi)$, $\mathbf{Q}$ and $\mathbf{\Lambda}$ must be calculated.

Besides, adding a record to the table also necessitates to determine the parameters of the cutting plane between the new record and the old one as well as the corresponding modification of the tree structure.

4.4.1 Determination of the mapping gradient matrix

Equation (6) alone cannot be used to calculate $\mathbf{A}(\Phi^q)$ as no literal expression of $\mathbf{R}(\Phi^q)$ is available but only a numerical evaluation returned by the ODE solver. Another relation is therefore needed to be able to compute the mapping gradient.

Since $\mathbf{R}(\Phi(t)) = \Phi(t + \Delta t)$ it comes from equation (6):

$$A_{ij}(\Phi(t)) = \frac{\partial \Phi_i(t + \Delta t)}{\partial \Phi_j(t)}. \tag{23}$$

If we note $\Phi_0 = \Phi(t_0)$ and $\Phi = \Phi(t_0 + \Delta t)$, we can reformulate equation (6) in a more compact way:

$$A(\Phi_0) = \frac{\partial \Phi}{\partial \Phi_0}, \tag{24}$$

as well as equation (4):

$$\frac{d\Phi}{dt} = \mathbf{S}(\Phi). \tag{25}$$

If we derive equation (25) with respect to Φ_0 we obtain

$$\frac{\partial}{\partial \Phi_0}\left(\frac{d\Phi}{dt}\right) = \frac{\partial \mathbf{S}(\Phi)}{\partial \Phi_0}, \tag{26}$$

which can be rewritten as

$$\frac{d}{dt}\left(\frac{\partial \Phi}{\partial \Phi_0}\right) = \frac{\partial \mathbf{S}(\Phi)}{\partial \Phi_0}, \tag{27}$$

and then as

$$\frac{d}{dt}\left(\frac{\partial \Phi}{\partial \Phi_0}\right) = \frac{\partial \mathbf{S}(\Phi)}{\partial \Phi}\frac{\partial \Phi}{\partial \Phi_0}, \tag{28}$$

which finally yields:

$$\frac{dA(\Phi^0)}{dt} = \mathbf{J}(\Phi)A(\Phi^0), \tag{29}$$

where $\mathbf{J}(\Phi)$ is the Jacobian of the ODE system and is defined by:

$$J_{ij}(\Phi) = \frac{\partial S_i(\Phi)}{\partial \Phi_j}. \tag{30}$$

The mapping gradient matrix can thus be obtained by using the ODE solver to solve equation (29).

In practice, because equation (29) requires $\mathbf{J}(\Phi(t))$, both equations (4) and (29) are solved together. A vector Y of length $D + D^2$ containing the components of Φ as well as those of $\mathbf{A}(\Phi)$ in row major order is associated to a vector F of the same length representing the system of ODEs defined by the rows of the vector $\mathbf{S}$ (see 3.1) and the elements of the corresponding Jacobian matrix $\mathbf{J}(\Phi)$:

$$Y = \begin{bmatrix} \Phi_1 \\ \vdots \\ \Phi_D \\ A_{11} \\ \vdots \\ A_{DD} \end{bmatrix}, \quad F = \begin{bmatrix} S_1(\Phi) \\ \vdots \\ S_D(\Phi) \\ J_{11}(\Phi) \\ \vdots \\ J_{DD}(\Phi) \end{bmatrix}. \tag{31}$$

$\mathbf{A}(\Phi^q)$ and $\mathbf{R}(\Phi^q)$ are thus both computed together by asking the external ODE solver to solve the following system whenever a retrieval cannot be performed:

$$\frac{dY(t)}{dt} = F(t). \tag{32}$$

The first D components of Y (corresponding to Φ) are initialized using the query Φ^q while the components corresponding to $\mathbf{A}(\Phi^q)$ are initialized using the identity matrix.

4.4.2 Determination of the EOA parameters

The principle of the calculation of $\mathbf{Q}$ and $\mathbf{\Lambda}$ is quite straightforward and has mostly been described in 4.3.2. Once the mapping gradient has been calculated, the following steps are used to compute the two matrices:

1. the SVD of the mapping gradient $\mathbf{A}$ is calculated (equation (19))
2. the singular values of $\mathbf{\Sigma}$ are modified to obtain $\widetilde{\mathbf{\Sigma}}$ (equation (21))
3. the matrix $\widetilde{\mathbf{A}}$ is calculated (equation (20))
4. $\mathbf{Q}$ and $\mathbf{\Lambda}$ are retrieved from the SVD of the matrix $\widetilde{\mathbf{A}}^T\mathbf{B}^T\mathbf{B}\widetilde{\mathbf{A}}/\epsilon_{tol}^2$ (equation (17))

However a minor technical difficulty appeared due to the external code used to perform the singular value decompositions (*GSL*, see 5.1.1). As a reminder, for a given real DxD matrix $\mathbf{M}$, $\sigma \in \mathbb{R}$ is a singular value if and only if there exist two vectors u and v in $\mathbb{R}^D$ such that:

$$\begin{cases} \mathbf{M}v = \sigma u \\ \mathbf{M}^T u = \sigma v \end{cases} \tag{33}$$

The vectors u and v are respectively referred to as the left-singular and right-singular vectors for σ: if the SVD of $\mathbf{M}$ is $\mathbf{U\Sigma V}^T$ then the diagonal entries of $\mathbf{\Sigma}$ are the singular values and the columns of U and V are the corresponding left- and right-singular vectors. Because of computing efficiency motivations, if $\sigma = 0$ is a singular value *GSL* does not compute the corresponding left-singular vectors thus leading to null columns in the matrix $\mathbf{U}$ obtained (this truncated version of $\mathbf{U}$ is referred to as $\widehat{\mathbf{U}}$). This is usually not an issue as $\mathbf{M} = \widehat{\mathbf{U}}\mathbf{\Sigma V}^T$ is still true.

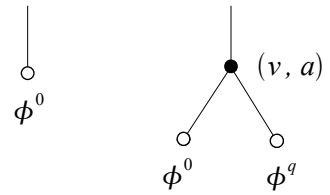
However, if after having replaced the null singular values by $\frac{1}{2}$ (equation (21)) we use $\widehat{\mathbf{U}}$ in the rest of the calculations, the matrix $\widetilde{\mathbf{A}}$ obtained will be incorrect (since the effect of the null columns of $\widehat{\mathbf{U}}$ will not anymore be cancelled out by null diagonal elements in $\widetilde{\mathbf{\Sigma}}$) and as a result so will the matrices $\mathbf{Q}$ and $\mathbf{\Lambda}$, thus leading to the wrong EOA.

In order to avoid this problem it is sufficient to determine those vectors u corresponding to $\sigma = 0$, i.e., according to equation (33), to determine a basis of the kernel of $\mathbf{M}^T$. It appears that the most convenient way to do that with *GSL* is to actually perform the

SVD of $\mathbf{M}^T$. Indeed it is trivial from equation (33) that a matrix and its transpose have the same singular values and that the right-singular vectors of $\mathbf{M}^T$ corresponding to $\sigma = 0$ are the missing left-singular vectors of $\mathbf{M}$ corresponding to that singular value. Therefore, in addition to modifying the singular values of the mapping gradient $\mathbf{A}(\Phi^q)$ according to equation (21), if one of them is null the SVD of $\mathbf{A}^T(\Phi^q)$ is performed so as to complete the columns of $\widehat{\mathbf{U}}$ and obtain the correct matrix $\mathbf{U}$.

4.4.3 Determination of the cutting plane parameters

As explained in 3.2.3 the addition of the record associated to Φ^q involves the creation of a new cutting plane separating the new record from the old one. We have seen in 4.1 that a cutting plane is defined by a vector $\mathbf{v} \in \mathbb{R}^D$ and a scalar a which, based on the scalar product $\mathbf{v}^T\Phi$, are used to separate the composition space in two regions referred to as *left* and *right*. However, while such a convention is simple enough and provides an easy criteria to traverse the tree, the determination of the couple $(\mathbf{v}, a)$ sketched in Figure 3 is not trivial.



(a) *Before* **(b)** *After*

Figure 3: *Schematic representation of how the binary tree is grown when a record is added to it.*

Pope suggests in [1] that the easiest way to do so is in the transformed space defined by applying to the composition space the linear transformation mapping Φ^0 EOA to the unit hypersphere. In this transformed composition space (sketched in Figure 4) the cutting plane is defined as the perpendicular bisector of the line between Φ^0 and Φ^q. We define $\mathbf{T}$ as the matrix associated to this linear transformation: if X is a point in the composition space, than $\mathbf{T}X$ is the corresponding point in the transformed composition space and is referred to as $\widehat{X}$. The algorithm we used to determine $(\mathbf{v}, a)$ can then be described as follows:

1. Calculate $\mathbf{T}$ (as explained below).
2. Apply $\mathbf{T}$ to Φ^q and Φ^0 to calculate the transformed tabulation points $\widehat{\Phi^q}$ and $\widehat{\Phi^0}$.

3. In the transformed composition space, $\widehat{\mathbf{v}} = \widehat{\Phi^q} - \widehat{\Phi^0}$ is an orthogonal vector to the perpendicular bisector of the line between $\widehat{\Phi^q}$ and $\widehat{\Phi^0}$, i.e. an orthogonal vector to the cutting plane in the transformed composition space.

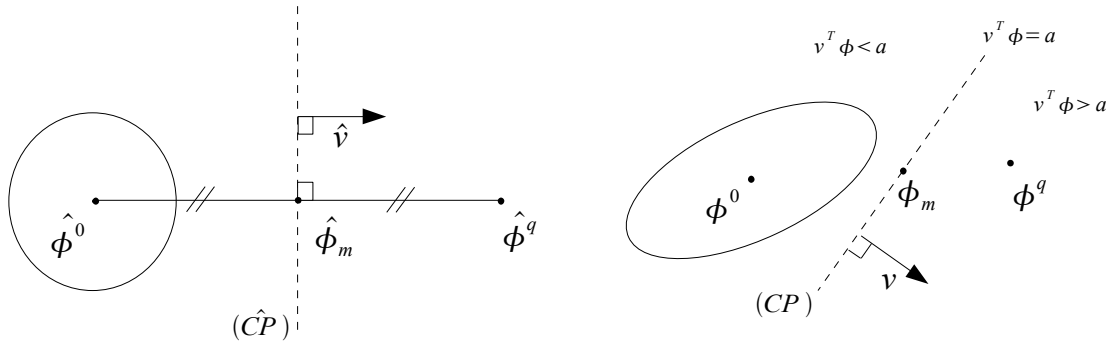
4. Apply $\mathbf{T}^{-1}$ to $\widehat{\mathbf{v}}$ to calculate $\mathbf{v}$.

5. Calculate the coordinates of the point $\widehat{\Phi_m}$, defined as the middle of the line between $\widehat{\Phi^q}$ and $\widehat{\Phi^0}$ (using the Euclidean norm).

6. Apply $\mathbf{T}^{-1}$ to $\widehat{\Phi_m}$ to calculate Φ_m.

7. Calculate a with $\mathbf{v}^T\Phi_m = a$.

8. Calculate $\mathbf{v}^T\Phi^0$ and change $\mathbf{v}$ into $-\mathbf{v}$ if $> a$ (by convention the old record Φ^0 is on the *left* of the cutting plane).



(a) *Transformed composition space* **(b)** *Composition space*

Figure 4: *Schematic representation of the cutting plane* (CP) *in relation with the tabulation points* Φ^0 *and* Φ^q*.*

This algorithm thus essentially relies on the calculation of the matrices $\mathbf{T}$ and $\mathbf{T}^{-1}$. According to equation (22), for any point $\Phi = \Phi^0 + \delta\Phi$ of the composition space located precisely on Φ^0 EOA (i.e. not *inside* the hyper-ellipsoid but *on* its surface) we have:

$$\delta\Phi^T\mathbf{Q}^T\mathbf{\Lambda}\mathbf{Q}\delta\Phi = 1. \tag{34}$$

By definition the linear transformation must transform Φ in a point located on the unit hypersphere centered on $\widehat{\Phi^0}$. Since the unit hypersphere is characterized by the identity matrix $\mathbf{I}$, the matrix $\mathbf{T}$ must therefore verify:

$$(\mathbf{T}\delta\Phi)^T(\mathbf{T}\delta\Phi) = 1, \tag{35}$$

i.e.

$$\delta\Phi^T \mathbf{T}^T \mathbf{T} \delta\Phi = 1. \tag{36}$$

If we call **P** the matrix corresponding to the linear transformation expressed in the basis constituted by the directions of Φ^0 EOA principle axes (see 4.3.2), we have

$$\mathbf{T} = \mathbf{Q}^T \mathbf{P} \mathbf{Q}, \tag{37}$$

and since **Q** is orthogonal equation (36) can then be rewritten as

$$\delta\Phi^T \mathbf{Q}^T \mathbf{P}^T \mathbf{P} \mathbf{Q} \delta\Phi = 1. \tag{38}$$

If we compare equations (38) and (34) it appears that the diagonal matrix

$$\mathbf{P} = \begin{bmatrix} \ddots & & 0 \\ & \sqrt{\lambda_i} & \\ 0 & & \ddots \end{bmatrix}. \tag{39}$$

leads to a suitable solution for **T**. Indeed with this definition:

$$\mathbf{P}^T \mathbf{P} = \mathbf{\Lambda}, \tag{40}$$

and thus:

$$(\mathbf{T}\delta\Phi)^T (\mathbf{T}\delta\Phi) = \delta\Phi^T \mathbf{Q}^T \mathbf{\Lambda} \mathbf{Q} \delta\Phi = 1. \tag{41}$$

The calculation of **T** and $\mathbf{T}^{-1}$ is thus straightforward as all the information needed (i.e. **Q** and $\mathbf{\Lambda}$) is contained in the record associated to Φ^0:

$$\begin{cases} P_{ii} = \sqrt{\lambda_{ij}} \ , \ P_{i \neq j} = 0 & \forall\, i,j \in \{1,...,D\} \\ P_{ii}^{-1} = \dfrac{1}{\sqrt{\lambda_{ij}}} \ , \ P_{i \neq j}^{-1} = 0 & \forall\, i,j \in \{1,...,D\} \\ \mathbf{T} = \mathbf{Q}^T \mathbf{P} \mathbf{Q} \\ \mathbf{T}^{-1} = \mathbf{Q}^T \mathbf{P}^{-1} \mathbf{Q} \end{cases} \tag{42}$$

4.4.4 Update of the tree structure

Once all the elements of the new record have been determined, it must be inserted in the tree as represented in Figure 3. This is done as follows:

1. The left child of the new cutting plane is set to be the old record Φ^0.
2. The right child of the new cutting plane is set to be the new record Φ^q.
3. The cutting plane that used to point to the record Φ^0 is now pointing to the new cutting plane.

While this is very simple and does not lead to any issue, when implementing the code it does require to add a *parent* attribute to the record structure (in addition to Φ, $\mathbf{R}(\Phi)$, $\mathbf{A}(\Phi)$, $\mathbf{Q}$ and $\mathbf{\Lambda}$) so that is is possible to go up the tree structure and find the cutting plane pointing to it.

4.5 Growth of a record's EOA

The EOA associated to the tabulation point Φ^0 is grown when for a query $\Phi^q = \Phi^0 + \delta\Phi$ it is found that:

$$\begin{cases} \delta\Phi^T \mathbf{Q}^T \mathbf{\Lambda} \mathbf{Q} \delta\Phi > 1 \\ |\mathbf{B}\left(\mathbf{R}(\Phi^q) - \mathbf{R}^l(\Phi^q)\right)| \leqslant \epsilon_{tol} \end{cases} \tag{43}$$

The growth of Φ^0 EOA, the purpose of which is to encompass the new point Φ^q, must be performed in such a way that it is minimal in order to avoid to include points of the composition space for which the linear approximation may not be within the error tolerance. In [1], Pope describes the new EOA as *the hyper-ellipsoid of minimal volume centered on Φ^0, which encloses both the original EOA and the point Φ^q* but does not elaborate on how to calculate it. This section details how we tackled this issue.

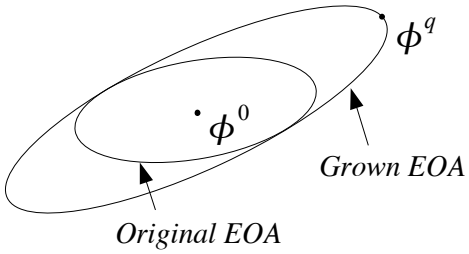


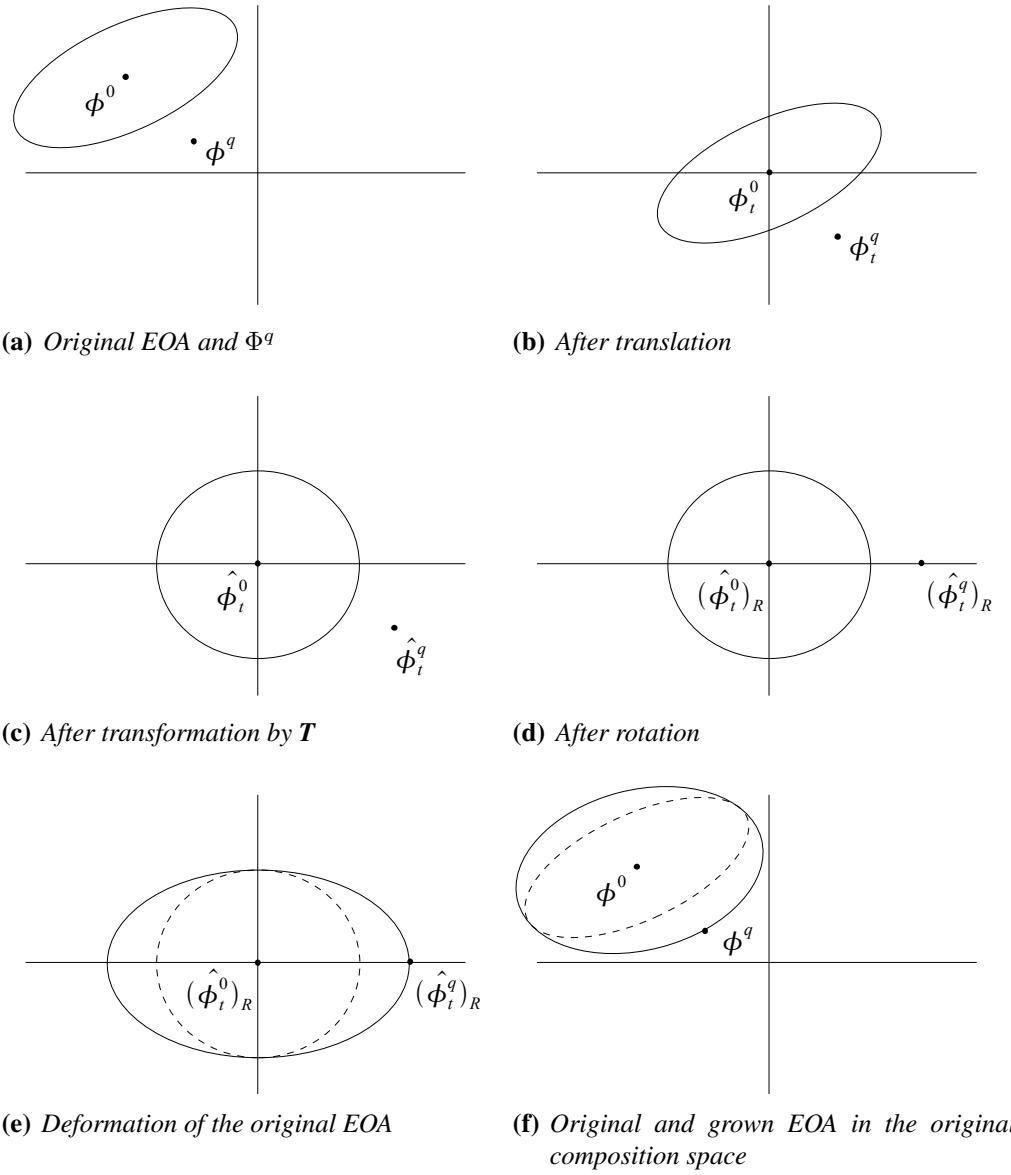
Figure 5: *Schematic representation of the Φ^0 EOA before and after growth in relation with the point Φ^q for which the approximation error ϵ is less than the tolerance ϵ_{tol}.*

4.5.1 Method

As for the determination of the cutting plane parameters (see 4.4.3), the idea is to use a transformed composition space to perform the growth. Indeed, let us consider a linear

transformation which maps the composition space to a space in which Φ^0 EOA is a hypersphere and in which Φ^q has a *non null coordinate in only one direction*. In such a space, stretching Φ^0 EOA in that particular direction so as to exactly reach Φ^q ensures that the growth is both suitable (the grown EOA encompasses the original one and includes Φ^q) and minimal (only the strict necessary deformation of the original EOA is performed).

As illustrated in Figure 6, such a suitable transformation can be obtained by composing the transformation **T** defined in 4.4.3 (mapping Φ^0 EOA to the unit hypersphere) with a translation (to center the EOA) and a rotation (to put Φ^q on a canonic axis).



(a) *Original EOA and* Φ^q

(b) *After translation*

(c) *After transformation by* ***T***

(d) *After rotation*

(e) *Deformation of the original EOA*

(f) *Original and grown EOA in the original composition space*

Figure 6: *Schematic representation of the successive transformations applied to calculate the grown EOA.*

Once the equation of the grown EOA in the transformed composition space has been determined, the corresponding equation in the original composition space can be calculated. In practice, we proceeded according to the following algorithm:

1. Calculate the coordinates of Φ_t^q, the point corresponding to Φ^q after application of the translation of the composition space putting Φ^0 at the origin.
2. Apply $\mathbf{T}$ (as defined in 4.4.3) to Φ_t^q to calculate the transformed tabulation point $\widehat{\Phi_t^q}$.
3. Calculate the rotation matrix $\mathbf{R}$.
4. Apply $\mathbf{R}$ to $\widehat{\Phi_t^q}$ to calculate $\left[\widehat{\Phi_t^q}\right]_R$.
5. Calculate the matrix $\widetilde{\mathbf{M}}_{Grown}$ describing the grown EOA in the transformed composition space.
6. Based on $\mathbf{T}$, $\mathbf{R}$ and $\widetilde{\mathbf{M}}_{Grown}$, calculate the matrix $\mathbf{M}_{Grown}$ describing the grown EOA in the original composition space.
7. Calculate the parameters of the grown EOA, $\mathbf{Q}_{Grown}$ and $\mathbf{\Lambda}_{Grown}$.

4.5.2 Implementation

Calculating Φ_t^q is trivial since by definition we have:

$$\Phi_t^q = \Phi^q - \Phi^0 = \delta\Phi. \tag{44}$$

The matrix $\mathbf{T}$ is calculated as explained in 4.4.3, using equation (42). As for the matrices $\mathbf{Q}_{Grown}$ and $\mathbf{\Lambda}_{Grown}$, their calculation is very straightforward: as described in 4.4.2 they are obtained directly by performing a SVD of $\mathbf{M}_{Grown}$. On the other hand, the determination of $\mathbf{R}$, $\widetilde{\mathbf{M}}_{Grown}$ and $\mathbf{M}_{Grown}$ is more complex and is detailed below.

Determination of the rotation matrix R

As a rotation we know that $\mathbf{R}$ must be a real DxD square matrix verifying:

$$\begin{cases} \mathbf{R}^T\mathbf{R} = \mathbf{I} \\ \det(\mathbf{R}) = 1 \end{cases} \tag{45}$$

However, additional information is needed to characterize $\mathbf{R}$: for the sake of convenience it was chosen to calculate the rotation that leads to only the first coordinate of $\left[\widehat{\Phi_t^q}\right]_R$ being non null. This can be interpreted as rotating $\widehat{\Phi_t^q}$ so that it is aligned with the basis axis $\mathbf{e}_1 = [1, 0, ..., 0]$. The rotation matrix $\mathbf{R}$ must therefore be such that:

$$\mathbf{R}\widehat{\Phi_t^q} = \left[\widehat{\Phi_t^q}\right]_R = \begin{bmatrix} x_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{46}$$

where x_1 is a real number representing $\left[\widehat{\Phi_t^q}\right]_R$ coordinate along $\mathbf{e}_1$.

The easiest way to calculate $\mathbf{R}$ is to construct it as the composition of elementary *Givens* rotations. A Givens rotation is a rotation in the plane spanned by two axes. If we consider the Givens rotation in the $(\mathbf{e}_i, \mathbf{e}_j)$ plane, it is represented by a matrix $\mathbf{G}_{(i,j)}$ identical to the identity matrix except for $g_{ii} = g_{jj} = c$, $g_{ij} = s$ and $g_{ji} = -s$ with $(c, s) \in [0, 1]^2$ verifying $c^2 + s^2 = 1$:

$$\mathbf{G}_{(i,j)} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}. \tag{47}$$

Such rotations are very convenient to introduce zeros in a vector. Indeed, for a given non null vector $\mathbf{Y} = [y_1, y_2]$, if we set

$$\begin{cases} r = \sqrt{y_1^2 + y_2^2} \\ c = \dfrac{y_1}{r} \\ s = \dfrac{y_2}{r} \end{cases} \tag{48}$$

we obtain

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}. \tag{49}$$

The method we used thus consists in successively zeroing all the components of $\widehat{\Phi_t^q}$, starting from the Dth component, proceeding upward and stopping when the second component has been reached and zeroed, i.e. once the form presented in equation (46) has been achieved.

We define $\mathbf{G}_i = \mathbf{G}_{(i,i-1)}$ (with $i \in \{2, ..., D\}$) the matrix associated to the Givens rotation that for a given vector Φ zeroes its ith component. For instance:

$$\mathbf{G}_D \widehat{\Phi_t^q} = \begin{bmatrix} \widehat{\phi_t^q}_1 \\ \vdots \\ \widehat{\phi_t^q}_{D-2} \\ r \\ 0 \end{bmatrix} \tag{50}$$

The determination of all the matrices $\mathbf{G}_i$ only requires the knowledge of $\widehat{\Phi_t^q}$ and is done successively according to the following algorithm:

1. Knowing the initial vector $\widehat{\Phi_t^q}$, $\mathbf{G}_D$ is calculated first based on equations (47) and (48).
2. The vector $\mathbf{G}_D \widehat{\Phi_t^q}$ is calculated.
3. Steps 1 and 2 are repeated, this time using the vector obtained in step 2 as the initial vector.

The product of two rotation matrices being a rotation matrix, the overall matrix $\mathbf{R}$ that verifies both equations (45) and (46) is then simply calculated as the result of the left-side multiplication of all the $\mathbf{G}_i$ matrices:

$$\mathbf{R} = \prod_{i=2}^{D} \mathbf{G}_i. \tag{51}$$

Determination of $\widetilde{\mathbf{M}}_{Grown}$

To determine the grown EOA in the transformed space we must remember that we only have to stretch the original EOA in the direction $\mathbf{e}_1$ until we reach $\left[\widehat{\Phi_t^q}\right]_R$. Therefore the expression of $\widetilde{\mathbf{M}}_{Grown}$ can easily be deducted from the matrix $\mathbf{D}$ corresponding to this deformation.

Indeed, if $\mathbf{D}$ corresponds to the transformation that stretches the hypersphere in order to include the point $\left[\widehat{\Phi_t^q}\right]_R$, the image of this point by the inverse transformation (represented by $\mathbf{D}^{-1}$) belongs to the unit hypersphere. This means $\mathbf{D}^{-1}$ verifies:

$$\left(\mathbf{D}^{-1}\left[\widehat{\Phi_t^q}\right]_R\right)^T\left(\mathbf{D}^{-1}\left[\widehat{\Phi_t^q}\right]_R\right)=1, \tag{52}$$

i.e.

$$\left[\widehat{\Phi_t^q}\right]_R^T\left(\mathbf{D}^{-1}\right)^T\mathbf{D}^{-1}\left[\widehat{\Phi_t^q}\right]_R=1, \tag{53}$$

Besides, the original EOA being represented by the identity matrix in the transformed composition space, the deformation matrix $\mathbf{D}$ has to be equal to the identity matrix except for one diagonal element which represents the deformation performed in the direction $\mathbf{e}_1$. Given our choice $\mathbf{e}_1 = [1, 0, ..., 0]$, this element is the first diagonal element and we therefore have:

$$\mathbf{D}=\begin{bmatrix} d & & 0 \\ & 1 & \\ & \ddots & \\ 0 & & 1 \end{bmatrix}, \quad d\in\,]1;+\propto[. \tag{54}$$

The condition $d > 1$ is guaranteed by the fact that since Φ^q is not within Φ_0 EOA, $\left[\widehat{\Phi_t^q}\right]_R$ has to be outside the unit hypersphere in the transformed composition space. It follows from equation (54) that:

$$\left(\mathbf{D}^{-1}\right)^T=\mathbf{D}^{-1}=\begin{bmatrix} \frac{1}{d} & & 0 \\ & 1 & \\ & \ddots & \\ 0 & & 1 \end{bmatrix}, \tag{55}$$

We then deduct from equations (53) and (55) that

$$\widetilde{\mathbf{M}}_{Grown}=\mathbf{D}^{-1}\mathbf{D}^{-1}=\begin{bmatrix} \frac{1}{d^2} & & 0 \\ & 1 & \\ & \ddots & \\ 0 & & 1 \end{bmatrix} \tag{56}$$

since $\mathbf{D}^{-1}\mathbf{D}^{-1}$ represents the hyper-ellipsoid of minimal volume which englobes the original EOA and include $\left[\widehat{\Phi_t^q}\right]_R$: its principle axes are identical to those of the original EOA (they all have a half-length of 1 in the transformed space) except for one which has been

stretched precisely to reach $\left[\widehat{\Phi_t^q}\right]_R$ (the half-length is now equal to $d > 1$). It is easily deducted from equations (46) and (56) that $d = x_1$, which is a very intuitive result.

Determination of the grown EOA matrix $\mathbf{M}_{Grown}$

Let us consider a point X of the original composition space. This point X belongs to the grown EOA surface if and only if the corresponding point $\widetilde{X}$ in the transformed space belongs to the surface of the EOA defined by $\widetilde{\mathbf{M}}_{Grown}$. This can be written as:

$$\forall X, \quad X^T \mathbf{M}_{Grown} X = 1 \Longleftrightarrow \widetilde{X}^T \widetilde{\mathbf{M}}_{Grown} \widetilde{X} = 1. \tag{57}$$

Since by construction we have

$$\widetilde{X} = \mathbf{RT} X, \tag{58}$$

equation (57) can be rewritten as

$$\forall X, \quad X^T \mathbf{M}_{Grown} X = 1 \Longleftrightarrow (\mathbf{RT}X)^T \widetilde{\mathbf{M}}_{Grown} (\mathbf{RT}X) = 1, \tag{59}$$

i.e.

$$\forall X, \quad X^T \mathbf{M}_{Grown} X = 1 \Longleftrightarrow X^T \left(\mathbf{T}^T \mathbf{R}^T \mathbf{D}^{-1} \mathbf{D}^{-1} \mathbf{RT}\right) X = 1. \tag{60}$$

The matrix $\mathbf{M}_{Grown}$ describing the grown EOA associated to Φ_0 in the original composition space is thus easily determined from equation (60) and is calculated as follows:

$$\mathbf{M}_{Grown} = \mathbf{T}^T \mathbf{R}^T \mathbf{D}^{-1} \mathbf{D}^{-1} \mathbf{RT}. \tag{61}$$

5 Implementation: the ISAT program

5.1 Architecture

5.1.1 Components

We have seen in 3.2 that ISAT primarily acts as an intelligent intermediate between a reactive flow code and an ODE solver in order to minimize the number of times the latter has to be called. Besides, as explained in the first section, combustion chemistry ODE systems are usually very stiff and the solver used must therefore have been specifically built to handle such systems efficiently. In addition to the need for such an adequate ODE

solver, sections 4.4.2 and 4.5.1 demonstrated the required ability to perform singular value decompositions (SVD).

However, the writing of such an ODE solver and SVD algorithm is far from trivial and is irrelevant given the focus of the work presented here. It was thus decided to use existing external codes instead. The program written was therefore based on *three components*:

- **The ISAT code**. This is the core of the program realised. It corresponds to the implementation of the algorithm presented in 3.2.3 based on the techniques developed in the fourth section. It was written in C++, using Microsoft Visual C++ 2008, and its characteristics are explained in 5.2.
- **GSL**. The GNU Scientific Library (GSL) is a C/C++ numerical library which is freely available under the GNU General Public License [36]. GSL linear algebra environment was used to manipulate all matrices and vectors and perform all linear algebra operations, including SVD. We used a Microsoft Windows port of version 1.8 [37].
- **CVODE**. This is a widely used stiff ODE solver, first developed in 1996 [38], which is part of the freely available SUNDIALS package [39]. We used version 2.5.0.

5.1.2 Structure

As previously illustrated in Figure 1b, the ISAT code is the interface with the reactive flow code: it receives queries Φ^q from it as inputs and returns either $\mathbf{R}(\Phi^q)$ or the approximation $\mathbf{R}^l(\Phi^q)$. However, before sending queries to ISAT the reactive flow must first send the information specifying what are the problem characteristics. These are the system equations represented by the vector F (as defined by equation (31) in 4.4.1), the integration time Δt, the error tolerance ϵ_{tol} and the scaling matrix $\mathbf{B}$.

If in the process of determining the reaction mapping a SVD has to be performed, ISAT uses the GLS module: it sends the adequate GSL routine the matrix $\mathbf{M}_{EOA}$ (as defined by equation (16) or (61)) and gets the corresponding matrices $\mathbf{Q}$ and $\mathbf{\Lambda}$ in return.

Similarly, in the event where the calculation of the reaction mapping requires to solve the ODE system, ISAT calls the CVODE solver. Provided with the system of ODEs to solve (F), the integration time (Δt) and the initials conditions (Φ^q and the identity matrix as explained in 4.4.1), CVODE then returns $\mathbf{R}(\Phi^q)$ and $\mathbf{A}(\Phi^q)$ to ISAT.

Figure 7 summarizes all these interactions and illustrates the structure of the program written.

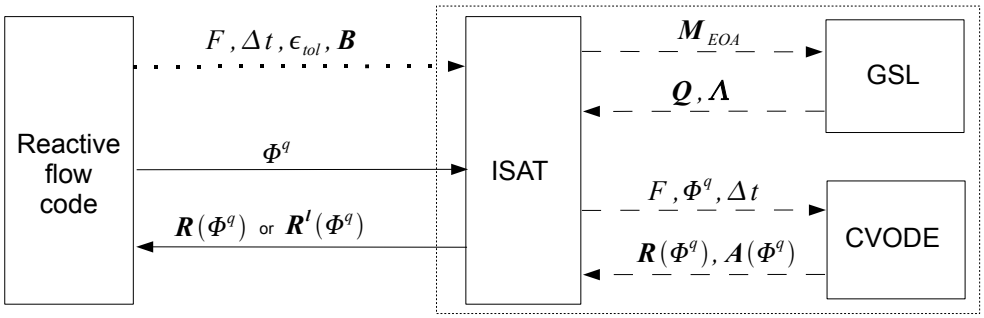


Figure 7: *Schematic representation of the interactions between a reactive flow code, the ISAT code written and the external codes used. The dotted arrow represents a one-time only interaction, the dashed arrows represent interactions that may occur for a query* Φ^q *and the solid ones interactions occurring for each query* Φ^q.

5.2 The ISAT code

The purpose of this section is to briefly outline the key characteristics of the ISAT code so as to illustrate how the algorithms and techniques described in the previous sections were ported into a computer program. At the same time, it also introduces the necessary information to understand what had to be tested in the ISAT code in order to validate it (see section 6).

It is *not* intended to be a listing of the source code or to address programming technicalities (in this regard, the C++ code in the source file and the associated header file has been extensively commented so as to make it as clear as possible and facilitate any future use).

5.2.1 Classes

The structure of the ISAT code is based on three main classes: *Record*, *Cutting_Plane* and *ISAT*. The purpose of the Record and Cutting_Plane classes is merely to *store* the ISAT table (each instance of theses classes represents a node of the binary tree), while the purpose of the ISAT class is to *retrieve* the reaction mapping from this table.

- **Record class**. The attributes of the Record class are the elements necessary to characterize the record associated to a tabulation point, as listed in 3.2.2, and a pointer towards the instance of the Cutting_Plane class which represents the parent node of this record (see 4.4.4). One instance of the Record class is created for each record within the ISAT table.

- **Cutting_Plane class**. Similarly, based on section 4.1, the Cutting_Plane class contains the couple $(\mathbf{v}, a)$ (as calculated in 4.4.3) and two pointers towards the two instances of the Record class which represent the left and right children of the cutting plane (as illustrated in Figure 3b). One instance of the Cutting_Plane class is created for each cutting plane within the tree.

- **ISAT class**. This class contains the methods necessary to manipulate the Record and Cutting_Plane instances constituting the ISAT table (see 5.2.2 below for a presentation of the main ones). Its attributes are composed of the information specifying the problem characteristics (listed in 5.1.2 and corresponding to the dotted line in Figure 7) and of a pointer towards the root of the binary tree (which is initially an instance of the Record class and becomes an instance of the Cutting_Plane class when a second record is added to the table, see 4.4.4). A unique instance of this class is created.

5.2.2 ISAT class methods

The principal methods of the ISAT class are listed below, along with their relation with the algorithms described in sections 3 and 4:

- **Query_Find**. Given a query Φ^q this routine is responsible for traversing the tree until it reaches a record, as explained in 4.1.

- **Query_Calculate**. This routine contains the overall ISAT algorithm described in 3.2.3. Given a query Φ^q and the record Φ^0 returned by the Query_Find routine, it is responsible for determining which of the three scenarios (Retrieve, Grow or Add) is relevant and calling the appropriate routine.

- **Solve_ODE**. Given a query Φ^q this routine calculates the exact reaction mapping $\mathbf{R}(\Phi^q)$ as well as the corresponding mapping gradient $\mathbf{A}(\Phi^q)$ by calling CVODE.

- **Retrieve**. Given a query Φ^q and a record Φ^0, this routines calculates the linearized reaction mapping $\mathbf{R}^l(\Phi^q)$ as explained in 4.2.1.

- **Grow**. Given a query Φ^q and a record Φ^0, this routines performs the growths of Φ^0 EOA so as to include Φ^q as explained in 4.5.

- **Add**. Given a query Φ^q and a record Φ^0, this routine creates the record corresponding to Φ^q and adds it to the table as explained in 4.4.

6 Validation and discussion

6.1 Prerequisite

In order to be able to demonstrate the validity of the ISAT program written we must first validate the Solve_ODE routine. It must indeed be verified that the CVODE code is called correctly and that, for a given system of ODEs, the solution returned by Solve_ODE is correct.

To do so, we used an ODE system provided by the SUNDIALS team as a test case for the CVODE code. This ODE system represents a 3-species chemical kinetics problem and is as follows:

$$\begin{cases} \dot{\phi}_1 = -0.04 \cdot \phi_1 + 10^4 \cdot \phi_2 \cdot \phi_3 \\ \dot{\phi}_2 = 0.04 \cdot \phi_1 - 10^4 \cdot \phi_2 \cdot \phi_3 - 3 \cdot 10^7 \cdot \phi_2^2 \\ \dot{\phi}_3 = 3 \cdot 10^7 \cdot \phi_2^2 \end{cases} \tag{62}$$

In the test case, the system was integrated with the initial condition $\Phi(0) = [1, 0, 0]$ and the solutions $\Phi(t = 4 \cdot 10^k)$ for $k \in \{0, 1, ..., 10\}$ were included. The routine Solve_ODE was thus used to integrate the query $\Phi^q = [1, 0, 0]$ for various Δt and the results were then compared to those provided by SUNDIALS.

Table 1: *Comparison of the results provided by the Solve_ODE routine with those included in SUNDIALS test case.*

Δt	SUNDIALS	Solve_ODE
$4 \cdot 10^2$	$\phi_1 = 4.505420 \cdot 10^{-1}$ $\phi_2 = 3.222963 \cdot 10^{-6}$ $\phi_3 = 5.494548 \cdot 10^{-1}$	$\phi_1 = 4.50519 \cdot 10^{-1}$ $\phi_2 = 3.2229 \cdot 10^{-6}$ $\phi_3 = 5.49456 \cdot 10^{-1}$
$3 \cdot 10^4$	$\phi_1 = 1.831878 \cdot 10^{-1}$ $\phi_2 = 8.941319 \cdot 10^{-7}$ $\phi_3 = 8.168113 \cdot 10^{-1}$	$\phi_1 = 1.83203 \cdot 10^{-1}$ $\phi_2 = 8.94142 \cdot 10^{-7}$ $\phi_3 = 8.16797 \cdot 10^{-1}$
$4 \cdot 10^4$	$\phi_1 = 3.897868 \cdot 10^{-2}$ $\phi_2 = 1.621567 \cdot 10^{-7}$ $\phi_3 = 9.610212 \cdot 10^{-1}$	$\phi_1 = 3.8984 \cdot 10^{-2}$ $\phi_2 = 1.62177 \cdot 10^{-7}$ $\phi_3 = 9.61017 \cdot 10^{-1}$

As can be seen from Table 1, the results returned by the Solve_ODE routine satisfyingly matched those provided by SUNDIALS.

6.2 Objectives

The validation of the ISAT program written (simply referred to as ISAT in the rest of this section) can be reduced to three points which must be verified for every query Φ^q:

1. **Table structure and traversing**. The ISAT table must be built and traversed correctly and a tabulation point close to the query must be reached.
2. **Scenario recognition**. The correct scenario (either Retrieve, Grow or Add) must be recognized and the corresponding solution returned.
3. **Precision**. The solution returned must satisfy the error tolerance specified.

The first point validates the routines Query_Find and Query_Calculate, while the second one corresponds to the validation of the routines Retrieve, Grow and Add (see 5.2.2).

6.3 Method

The idea behind the method used was to send ISAT a set of queries for which we knew the exact reaction mapping (to assess the precision) as well as which scenario should be chosen by ISAT (to assess the traverse of the table and the scenario recognition). For instance, if the first query sent to ISAT is $\Phi^1 = [1, 0, 0]$, then if the next one is $\Phi^2 = [0, 0.1, 0.9]$ we expect, given the strong difference in the initial compositions, an addition event to occur and the two reaction mappings to be quite different.

In practice, we wrote a short piece of code, referred to as the *calling program*, to simulate the reactive flow code represented in Figure 7 and send queries to ISAT. The ODE system described by equation (62) was used to represent the chemical kinetics problem and integrated over $\Delta t = 4 \cdot 10^4$. In order to determine which queries would be relevant to send to ISAT we considered the query $\Phi^1 = [1, 0, 0]$ as a starting point: based on sections 4.2.2 and 4.4.2 we estimated three queries Φ^2, Φ^3, and Φ^4 (see equation (63)) which should respectively lead to an Add event, a Grow event and a Retrieve event if each was to be sent to ISAT just after the initial query Φ^1.

$$\Phi^2 = \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \end{bmatrix}, \Phi^3 = \begin{bmatrix} 1.001 \\ 0 \\ 0 \end{bmatrix}, \Phi^4 = \begin{bmatrix} 1.0001 \\ 0 \\ 0 \end{bmatrix}. \tag{63}$$

The estimation of these queries first required to specify an error tolerance and for our validation purpose we decided to set $\epsilon_{tol} = 1.0 \cdot 10^{-3}$ (for a given set of queries it is

important to remember that a tolerance too strict will lead to a disproportionate number of *additions*, whereas a value of ϵ_{tol} too big would result essentially in *retrievals*).

For the first validation run (Table 2), two queries q_1 and q_2 chosen from this set of Φ^i, $i \in \{1, ..., 4\}$, were sent successively to ISAT. The scalar $(\delta\Phi)^T \mathbf{Q}^T \mathbf{\Lambda Q} (\delta\Phi)$ (see 4.3.2) was retrieved to assess whether or not the second query belonged to the EOA associated to the tabulation point created when the first query was sent. In order to know which event occurred, we added to the ISAT code a routine monitoring the number of records in the table as well as the number of retrievals, additions and growths performed. The precision of the reaction mappings returned by ISAT was assessed by retrieving the local error defined in 4.2.2 for each query (for the sake of simplicity, given our validation only purpose, the scaling matrix **B** was set to the identity matrix).

We then conducted a second validation run (Table 3) consisting in sending two sets of queries to ISAT which differed only by the order in which the queries were sent. For each set we monitored the final number of records in the table as well as the number of times each of the three possible scenario had occurred.

6.4 Results

Table 2: *Influence of the composition query on scenario recognition.*

Case	Inputs		$(\delta\Phi)^T \mathbf{Q}^T \mathbf{\Lambda Q} (\delta\Phi)$	Event	Local error ϵ
	q_1	q_2			
1	Φ^1	Φ^2	$3.75 \cdot 10^5$	Addition	$5.19984 \cdot 10^{-2}$
2	Φ^1	Φ^3	1.03575	Growth	$9.32272 \cdot 10^{-4}$
3	Φ^1	Φ^4	$1.03575 \cdot 10^{-2}$	Retrieve	$9.33026 \cdot 10^{-5}$
4	Φ^4	Φ^3	0.838946	Retrieve	$8.3896 \cdot 10^{-4}$

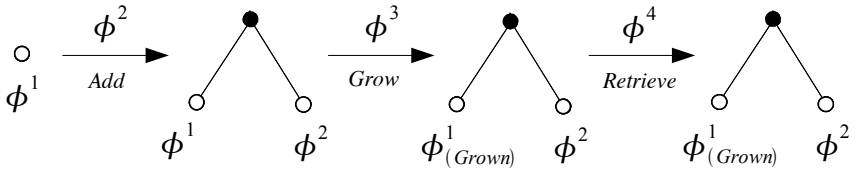
Table 3: *Influence of the table content on the treatment of queries.*

Case	Inputs				Records	Events		
	q_1	q_2	q_3	q_4		Retrieve	Grow	Addition
5	Φ^1	Φ^2	Φ^3		2	0	1	1
	Φ^1	Φ^2	Φ^3	Φ^4	2	1	1	1
6	Φ^4	Φ^1	Φ^2		2	1	0	1
	Φ^4	Φ^1	Φ^2	Φ^3	2	2	0	1

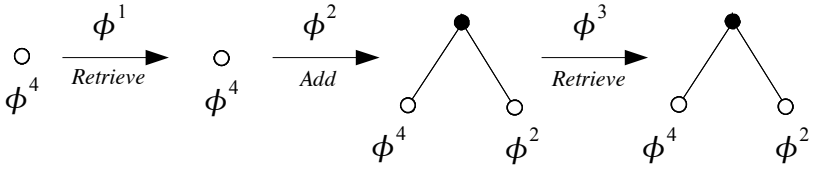
6.5 Discussion

The results contained in Table 2 show that both the precision and scenario recognition requirements are fulfilled. Indeed, we can see that retrievals (cases 3 and 4) occurred only when the second query was within the EOA of the tabulation point associated to the initial query ($\delta\Phi^T\mathbf{Q}^T\mathbf{\Lambda}\mathbf{Q}\delta\Phi \leqslant 1$) and that the associated local errors are within the specified tolerance ($\epsilon \leqslant 1 \cdot 10^{-3}$). Case 2 shows that the precision associated to a growth event is satisfying: indeed, $\epsilon \leqslant 1 \cdot 10^{-3}$ even though the query was outside the tabulation point EOA ($\delta\Phi^T\mathbf{Q}^T\mathbf{\Lambda}\mathbf{Q}\delta\Phi > 1$). It can also be seen from this case that a growth event does correspond to the situation where the query is close to the tabulation point since $\delta\Phi^T\mathbf{Q}^T\mathbf{\Lambda}\mathbf{Q}\delta\Phi \approx 1$. Finally, case 1 shows that an addition does occur when the query is outside the tabulation point EOA and that the local error associated to a retrieve would be greater than the specified tolerance.

Table 2 data are however not enough to validate that the ISAT table is constructed and traversed correctly since there is only one tabulation point in it when the test query is submitted to ISAT. This was the purpose of the second validation run and the results contained in Table 3 show that it is actually the case. Indeed, according to those results, in case 5 Φ^3 was sent to ISAT after Φ^1 and Φ^2 and led to a growth (since we know from Table 2 that Φ^2, sent just after Φ^1, led to the addition) whereas in case 6 Φ^3 was sent to ISAT after Φ^4, Φ^1 and Φ^2 had been sent and led to a retrieval from Φ^4 (since case 5 shows it cannot be retrieved from either Φ^1 or Φ^2). This means that the traverse of the table for the query Φ^3 led to the tabulation point associated to the query Φ^4 which proves the table is both constructed and traversed correctly. The construction of the binary tree representing the ISAT table in cases 5 and 6 can be illustrated by Figure 8.



(a) *Case 5.*



(b) *Case 6.*

Figure 8: *Evolution of the binary tree as queries are sent to ISAT.*

These tests and results showed that the ISAT program written behaved as expected which proved its *functionality*. However the *efficiency* of the algorithm implemented, i.e. the speed gain over a traditional direct integration technique, could not be tested because of time constraints. Indeed, many technical issues not mentioned here were encountered (due to platforms compatibility and correct integration of the external codes) and slowed down the implementation of the algorithm. The time gain due to ISAT should only become apparent over a large number of queries and therefore any attempt at quantifying the program efficiency will require at least a couple thousands different inputs. This would also be the opportunity to study how the occurrence of each scenario (Retrieve, Add, Grow) evolves: as the number of queries becomes more important it is expected that the number of additions and growths decrease while retrievals becomes preponderant. These points provide ground for future work on the ISAT program written.

7 Conclusion

Amongst the numerical schemes which have been developed to alleviate the computational cost of combustion chemistry, the main drawbacks traditionally associated with storage based methods are often due to the fact that the entire *realizable region* is tabulated. Indeed, because the *accessed region* is likely to be much smaller this does not allow for the most efficient use of computer resources. This issue is addressed by the ISAT method by tabulating only the *accessed region*. The ISAT table is built *in situ* and is progressively constructed in such a way that the retrieved solutions stay within the specified error tolerance. To do so, the ISAT algorithm builds a binary tree whose leaves contain the reaction mapping for particular compositions that occured in the reactive flow, information on how to approximate the reaction mapping for similar compositions and, most importantly, information on the region of the composition space where this approximation is legitimate (ROA). The internal nodes of the tree contain information on how to find the relevant leaf for a given query. The construction of this binary tree essentially depends on two functions: an *Add* function, which adds a new leaf to the tree, and a *Grow* function, which expands the ROA of a leaf. The queries received by ISAT can be considered as points in a D-dimensional space and the implementation of the ISAT algorithm can thus be considered as an an algebra problem. The determination of the elements characterizing the Add and Grow functions was then addressed by using a *transformed composition space*. The validity of this approach and of the solutions suggested was proven by sending mock queries to the program written and demonstrating it had the expected behaviour.

References

[1] S.B. Pope. Computationally efficient implementation of combustion chemistry using *in situ* adaptive tabulation. *Combustion Theory and Modelling*, 1(1):41–63, 1997. doi:10.1080/713665229.

[2] C.K. Westbrook. Computational combustion. *Proceedings of the Combustion Institute*, 30(1):125–157, 2005. doi:10.1016/j.proci.2004.08.275.

[3] G.D. Byre and A.C. Hindmarsh. Stiff ODE slovers: a review of current and coming attractions. *Journal of Computational Physics*, 70(1):1–62, 1987.

[4] K. Radhakrishnan. Comparison of numerical techniques for integration of stiff ordinary differential equations arising in combustion chemistry. *NASA TP-2372*, 1984.

[5] N.N. Yanenko. *The Method of Fractional Steps*. Springer, Berlin, 1971.

[6] G. Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, 1968.

[7] B. Sportisse. An analysis of operator splitting techniques in the stiff case. *Journal of Computational Physics*, 161(1):140–168, 2000.

[8] S.B. Pope. PDF methods for turbulent reactive flows. *Progress in Energy and Combustion Science*, 11:119–192, 1985. doi:10.1016/0360-1285(85)90002-4.

[9] S. Mazumder. Adaptation of the *in situ* adaptive tabulation (ISAT) procedure for efficient computation of surface reactions. *Computers and Chemical Engineering*, 30(1):115–124, 2005. doi:10.1016/j.compchemeng.2005.08.008.

[10] M.A. Singer and S.B. Pope. Exploiting ISAT to solve the reaction–diffusion equation. *Combustion Theory and Modelling*, 8(2):361–383, 2004.

[11] R.B. Brad, A.S. Tomlin, M. Fairweather, and J.F. Griffiths. The application of chemical reduction methods to a combustion system exhibiting complex dynamics. *Proceedings of the Combustion Institute*, 31(1):455–463, 2007. doi:10.1016/j.proci.2006.07.026.

[12] T. Turányi, A.S. Tomlin, and M.J. Pilling. On the error of the quasi-steady-state approximation. *The Journal of Physical Chemistry*, 97(1):163–172, 1993.

[13] M.D. Smooke (eds). Reduced Kinetic Mechanisms and Asymptotic Approximations for Methane-Air Flames: A Topical Volume. *Lecture Notes in Physics, Berlin Springer Verlag*, 384, 1991.

[14] S. Rigopoulos. Reduced flame kinetics via rate-controlled constrained equilibrium. In *Computational Science ICCS 2006*, pages 18–25. Springer Berlin / Heidelberg, 2006. doi:10.1007/11758525.

[15] J.C. Keck. Rate-controlled constrained-equilibrium theory of chemical reactions in complex systems. *Progress in Energy and Combustion Science*, 16(2):125–154, 1990.

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

[17] M. Maas. Efficient calculation of intrinsic low-dimensional manifolds for the simplification of chemical kinetics. *Computing and Visualization in Science*, 1(2):69–81, 1998. doi:10.1007/s007910050007.

[18] J.Y. Chen, W. Kollmann, and R.W. Dibble. PDF modeling of turbulent nonpremixed methane jet flames. *Combustion Science and Technology*, 64(4-6):315–346, 1989. doi:10.1080/00102208908924038.

[19] F.C. Christo, A.R. Masri, and E.M. Nebot. Artificial neural network implementation of chemistry with pdf simulation of H_2/CO_2 flames. *Combustion and Flame*, 106(4):406–427, 1996. doi:10.1016/0010-2180(95)00250-2.

[20] J.A. Blasco, N. Fueyo, C. Dopazo, and J. Ballester. Modelling the temporal evolution of a reduced combustion chemical system with an artificial neural network. *Combustion and Flame*, 113(1-2):38–52, 1998. doi:10.1016/S0010-2180(97)00211-3.

[21] T. Turányi. Parameterization of reaction mechanisms using orthonormal polynomials. *Computers & chemistry*, 18(1):45–54, 1994. doi:10.1016/0097-8485(94)80022-7.

[22] S.R. Tonse, N.W. Moriarty, N.J. Brown, and M. Frenklach. PRISM: piecewise reusable implementation of solution mapping. An economical strategy for chemical kinetics. *Israeli Journal of Chemistry*, 39(1), 1998.

[23] H. Rabitz and Ö.F. Aliş. General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25(2):197–233, 1999.

[24] M.A. Singer, S.B Pope, and H.N. Najm. Modeling unsteady reacting flow with operator splitting and ISAT. *Combustion and Flame*, 147(1-2):150–162, 2006. doi:10.1016/j.combustflame.2006.06.007.

[25] S. James, M. Anand, M. Razdan, and S.B. Pope. *In situ* detailed chemistry calculations in combustor flow analyses. *Journal of Engineering for Gas Turbines and Power*, 123:747–756, 2001. doi:10.1115/1.1384878.

[26] B. Liu and S.B. Pope. The performance of *in situ* adaptive tabulation in computations of turbulent flames. *Combustion Theory and Modelling*, 9(4):549–568, 2005. doi:10.1080/13647830500307436.

[27] B. Yang and S.B. Pope. Treating chemistry in combustion with detailed mechanisms—In situ adaptive tabulation in principal directions—Premixed combustion. *Combustion and Flame*, 112(1-2):85–112, 1998. doi:10.1016/S0010-2180(97)81759-2.

[28] J.Y. Chen. Analysis of *in situ* adaptive tabulation performance for combustion chemistry and improvement with a modified search algorithm. *Combustion Science and Technology*, 176(7):1153–1169, 2004. doi:10.1080/00102200490426488.

[29] L. Lu and S.B. Pope. A systematic investigation of *in situ* adaptive tabulation for combustion chemistry. *FDA 07-01*, 2007.

[30] Q. Tang and S.B. Pope. Implementation of combustion chemistry by *in situ* adaptive tabulation of rate-controlled constrained equilibrium manifolds. *Proceedings of the Combustion Institute*, 2002. doi:10.1016/S1540-7489(02)80173-0.

[31] J.Y. Chen, J.A. Blasco, N. Fueyo, and C. Dopazo. An economical strategy for storage of chemical kinetics: Fitting in situ adaptive tabulation with artificial neural networks. *Proceedings of the Combustion Institute*, 28:115–121, 2000. doi:10.1016/S0082-0784(00)80202-7.

[32] J.D. Hedengren and T.F. Edgar. In situ adaptive tabulation for real-time control. *Industrial and Engineering Chemistry Research*, 44(8):2716–2724, 2005. doi:10.1021/ie049322s.

[33] A. Arsenlis, N.R. Barton, R. Becker, and R.E. Rudd. Generalized in situ adaptive tabulation for constitutive model evaluation in plasticity. *Computer Methods in Applied Mechanics and Engineering*, 196(1-3):1–13, 2006. doi:10.1016/j.cma.2005.10.031.

[34] L. Wang and R.O. Fox. Application of in situ adaptive tabulation to CFD simulation of nano-particle formation by reactive precipitation. *Chemical Engineering Science*, 58(19):4387–4401, 2003. doi:10.1016/S0009-2509(03)00321-X.

[35] A. Varshney and A. Armaou. Multiscale optimization using hybrid PDE/kMC process systems with application to thin film growth. *Chemical Engineering Science*, 60(23):6780–6794, 2005. doi:10.1016/j.ces.2005.05.055.

[36] GSL, available at "`http://www.gnu.org/software/gsl/`", accessed on August 15, 2008.

[37] GSL for Windows, available at "`http://gnuwin32.sourceforge.net/packages/gsl.htm`", accessed on August 15, 2008.

[38] S.D. Cohen and A.C. Hindmarsh. CVODE, a stiff/nonstiff ODE solver in C. *Computers in Physics*, 10(2):138–143, 1996.

[39] CVODE, available at "`https://computation.llnl.gov/casc/sundials/download/download.html`", accessed on August 15, 2008.