Journal of Mathematical Chemistry Vol. 33, No. 1, January 2003 (© 2003)

A novel algorithm for the numerical integration of systems of ordinary differential equations arising in chemical problems

F. Aluffi-Pentini [a], V. De Fonzo [b] and V. Parisi [c,*]

[a] *Dipartimento di Metodi e Modelli Matematici, Università di Roma "La Sapienza", Via A. Scarpa 16, I-00161 Roma, Italy*
[b] *EuroBioPark c/o Parco Scientifico, Università di Roma "Tor Vergata", Via della Ricerca Scientifica 1, I-00133 Roma, Italy*
[c] *Sezione INFM, Dipartimento di Fisica, Università di Roma "Tor Vergata", Via della Ricerca Scientifica 1, I-00133 Roma, Italy*
E-mail: Valerio.Parisi@roma2.infn.it

Received 17 December 2001; revised 12 November 2002

Simulation of large networks of chemical reactions via the numerical integration of large systems of ordinary differential equations is of growing importance in real-world problems. We propose an attractive novel numerical integration method, that is largely independent from ill-conditioning and is suitable for any nonlinear problem; moreover, the method, being exact for linear problems, is especially precise for quasi-linear problems, the most frequent kind in the real world. The method is based on a new approach to the computation of a matrix exponential, includes an automatic correction of rounding errors, is not too expensive computationally, and lends itself to a short and robust software implementation that can be easily inserted in large simulation packages. A preliminary numerical verification has been performed, with encouraging results, on two sample problems. The full source listing (in standard C language) of an academic version of the algorithm is freely available on request (e-mail address: `Valerio.Parisi@roma2.infn.it`), together with a very simple but very stiff chemical problem.

KEY WORDS: algorithms, chemical pathways, differential equations, numerical integration, matrix exponential

1. Introduction

Simulation of the kinetics of chemical reaction networks is becoming a very active field of chemical research with a strong impact on the set-up of appropriate models and their software implementation. Important chemical examples in which such networks are involved are the simulation of industrial chemical processes, of pollution dynamics, and of the kinetics of systems of biochemical reactions.

* Corresponding author.

0259-9791/03/0100-0001/0

One of the most important problems in such models (when spatial aspects such as diffusion or turbulence can be neglected) is the solution of initial value problems for systems of ordinary differential equations. Such problems cannot be generally solved in analytical form, and one must resort to numerical integration techniques.

The main difficulty in numerical integration arises when a system of differential equations is stiff, i.e., when speeds of variation coexist having very different orders of magnitude. Stiffness is obviously more likely, and more harmful, when a great number of variables are involved, as in important practical cases.

In these cases the classical explicit integration methods (e.g., Euler or Runge–Kutta) are instability-prone and require very small, and therefore, too many integration steps, thus causing unacceptable integration times and even an excessive error accumulation.

Implicit methods have the advantage of allowing very large steps that do not introduce instabilities but suffer from two drawbacks: the additional burden of finding at each step the roots of a nonlinear system; and the fact that too large steps may cause spurious damping or oscillations.

The so-called linearly implicit methods (e.g., linearised Backward-Euler or more generally the methods due to Lambert and Sigurdsson [1]) have the same characteristics of implicit methods but are computationally more economical since at each step they require only to find the roots of a linear system.

Another aspect that is to be considered is the nonlinearity of the differential equation system: we note that in chemical reactions the nonlinear functions involved in ordinary differential equations are always polynomials in several variables [2], that can always be reduced to second-order polynomials (Riccati differential equation). We note by the way that also a surprisingly great number of non-polynomial differential equations arising in other fields can always be reduced to a Riccati form [3].

We now note that in most chemical problems the polynomial is already of second degree, and the nonlinearity is mild: most monomials are at most bilinear or even linear. It is therefore natural to consider the linearised problem, that has a simple analytical solution and as such does not suffer from numerical integration problems due to stiffness. The analytical solution of the linearised problem provides a satisfactory local approximation to the exact solution, usually without introducing spurious damping or oscillations; and it can be used as an iterative step in a numerical integration method for the nonlinear case. This idea is not new (for a review of several methods, see [4]), but since the analytical solution of the linearised problem requires the computation of the exponential of the Jacobian matrix [5,6], this approach has mostly been regarded as rather impractical due to the difficulties in implementing it in a reliable and economical way.

The Krylov-subspaces approximation technique [7] has made the matrix exponential computation feasible; we note however that this technique by its very nature neglects the effects of the non-dominant eigenvalues at the expense of again introducing inaccuracies (such as spurious oscillations or dampings) also in linear or quasilinear cases; moreover, the recent software implementation Expokit [8] appears to be rather bulky.

We propose in this paper a numerical integration method of matrix-exponential type that uses a novel algorithm for computing the exponential, and that in our opinion has a number of pleasant features.

Firstly, our method preserves the virtues of a matrix-exponential solver with steplength control, i.e., it is largely independent from ill-conditioning, it is suitable for any nonlinear problem and it gives the exact analytic solution for linear problems. Secondly, our method for the computation of the matrix exponential does not deliberately neglect the effects of any eigenvalue; and thirdly, it includes a novel algorithm for the automatic correction of the accumulation effect of some kind of rounding errors.

Moreover, our method is not too expensive computationally, and it lends itself to a short and robust software implementation that, apart from its obvious standalone use, is a good candidate to be easily inserted in those large programs for chemical simulations where the integration method is often implemented in relatively independent (and sometimes unsophisticated) software modules.

2. Algorithm

2.1. The numerical integration

We consider the autonomous nonlinear ordinary (vector) differential equation for the time function $\underline{X}(t)$, in the explicit form

$$\dot{\underline{X}} = \underline{f}(\underline{X})$$

with initial condition $\underline{X}(t_0) = \underline{X}_0$, where the underlined symbols denote column vectors of length n.

We note that any non-autonomous equation $\dot{\underline{X}} = \underline{f}(\underline{X}, t)$ can be always put in autonomous form by simply adding an auxiliary variable.

Let $\underline{x}(t)$ be the (continuous and piecewise smooth) approximation to the solution, defined below.

For $k = 0, 1, 2, 3, \ldots$, let t_k, t_{k+1} be the initial and final times of the step $k+1$ of the procedure, $h_{k+1} = t_{k+1} - t_k$ the time stepsize, and let $\underline{x}_0 = \underline{X}_0$ and $\underline{x}_k = \underline{x}(t_k)$ for $k > 0$.

We integrate, along the step $k+1$, a linearised form of our differential equation, as follows.

We denote by $J(\underline{x})$ the $n \times n$ Jacobian matrix of $\underline{f}(\underline{x})$, we put, for the sake of simplicity, $\underline{f}_k = \underline{f}(\underline{x}_k)$, $J_k = J(\underline{x}_k)$ and we consider the linearisation $\underline{f}_k + J_k \cdot (\underline{x} - \underline{x}_k)$ of $\underline{f}(\underline{x})$ around $\underline{x}_k$.

The approximate solution $\underline{x}(t)$ along step $k+1$, i.e., from $t = t_k$ to $t = t_{k+1}$, is the solution of the differential equation

$$\frac{\mathrm{d}\underline{x}(t)}{\mathrm{d}t} = \underline{f}_k + J_k \cdot \left(\underline{x}(t) - \underline{x}_k\right)$$

with the initial condition $\underline{x}(t_k) = \underline{x}_k$.

The analytical solution of the differential equation is, within step $k+1$,

$$\underline{x}(t) = \tau \cdot \varphi(\tau \cdot J_k) \cdot \underline{f}_k + \underline{x}_k,$$

where $\tau = t - t_k$ and $\varphi(G) \equiv G^{-1} \cdot (\mathrm{e}^G - I)$.

The initial point of the following step will be, therefore, $\underline{x}_{k+1} = \underline{x}(t_{k+1})$, i.e., the final point of the current step.

To simplify the notation we put, inside the step $k+1$,

$$\underline{\xi}_k(\tau) = \underline{x}(t) - \underline{x}_k,$$

and we drop the subscript k, so that the linearised equation for the generic step can be written $\mathrm{d}\underline{\xi}/\mathrm{d}\tau = J \cdot \underline{\xi} + \underline{f}$ with initial condition $\underline{\xi}(0) = \underline{0}$ and the solution is, for nonsingular J,

$$\underline{\xi}(\tau) = J^{-1} \cdot \left(\mathrm{e}^{\tau \cdot J} - I\right) \cdot \underline{f},$$

where I is the $n \times n$ identity matrix.

To avoid analytical or numerical annoyances when J is singular or nearly singular we consider an augmented problem, which avoids altogether the matrix inversion, and therefore, is simpler and more robust.

We define the augmented $(n+1)$-dimensional vectors

$$\underline{\eta} \equiv \begin{pmatrix} \underline{\xi} \\ 1 \end{pmatrix}, \quad \text{and therefore,} \quad \underline{\eta}_0 \equiv \begin{pmatrix} \underline{0} \\ 1 \end{pmatrix},$$

and the augmented $(n+1) \times (n+1)$ matrix

$$A \equiv \begin{pmatrix} J & \underline{f} \\ \underline{0}^{\mathrm{T}} & 0 \end{pmatrix},$$

where the n-vectors $\underline{\xi}$, $\underline{f}$ and $\underline{0}$ are considered column vectors, and superscript T indicates transposition.

We note that A has (apart from an added zero eigenvalue) the same eigenvalue spectrum as J.

The above non-homogeneous differential equation becomes a homogeneous equation

$$\frac{\mathrm{d}\underline{\eta}}{\mathrm{d}\tau} = A \cdot \underline{\eta},$$

with initial condition $\underline{\eta}(0) = \underline{\eta}_0$.

The exact solution of the augmented problem is

$$\underline{\eta}(\tau) = \mathrm{e}^{\tau \cdot A} \cdot \underline{\eta}_0,$$

which only requires the computation of a matrix exponential, and is valid also if J is singular.

Due to the definitions of $\underline{\eta}$ and $\underline{\eta}_0$, the vector $\underline{\eta}$ will be given by the last column of $e^{\tau \cdot A}$, while the vector $\underline{\xi}$ will be given by the first n elements of $\underline{\eta}$. Therefore, at the end of step $k+1$ the value of the solution $\underline{x}(t)$ provided by our method is

$$\underline{x}_{k+1} = \underline{x}_k + \underline{\xi}_k(h_{k+1}),$$

where the n-vector $\underline{\xi}_k(h_{k+1})$ consists of the first n elements of

$$\underline{\eta}(h_{k+1}) = e^{h_{k+1} \cdot A} \cdot \underline{\eta}_0,$$

where A is evaluated at $\underline{x}_k$.

We now consider in some detail the computation of $e^{h \cdot A}$.

2.2. *On the computation of the exponential of a matrix*

We describe here the very simple method we propose.

Let Z be a square matrix and I the corresponding identity matrix. The exact formula

$$e^Z = \lim_{N \to \infty} \left(I + \frac{Z}{N} \right)^N$$

provides the truncated approximation

$$e^Z \cong \left(I + \frac{Z}{N} \right)^N$$

for a suitable value of N.

An improved form is

$$e^Z \cong \left(I + \frac{Z}{2^b} \right)^{2^b}$$

for a suitable value of b, since it can be computed by b iterated squaring operations starting from

$$I + \frac{Z}{2^b}.$$

This idea (which, according to Knuth [9], can be traced back to Pingala in 200 B.C.) is much more economical in terms of the number of matrix multiplications.

2.3. *Control of rounding errors*

In order to counter the loss of significant digits due to rounding we devise a simple method inspired by Kahan's method [10,11] for reducing rounding losses in sums of real numbers.

The idea is to store a cumulated effect of rounding errors, to suitably reintroduce it in a later phase of the computation.

In order to compute the exponential of the $(n+1)\times(n+1)$ matrix $h\cdot A$, we define a new $(n+2)\times(n+2)$ matrix

$$M \equiv \begin{pmatrix} h\cdot A & \underline{0} \\ \underline{s}^{\mathrm{T}} & 0 \end{pmatrix},$$

where the underlined symbols are column vectors of length $(n+1)$ and

$$s_j = -h\cdot\sum_{l=1}^{n+1} a_{lj}, \quad j = 1,\ldots,n+1,$$

so that the sum of all elements of each column is zero.

We note that, due to the structure of M, any power of M contains as a submatrix the corresponding power of $h\cdot A$ in the same position where M contains $h\cdot A$; and therefore, the required matrix $\mathrm{e}^{h\cdot A}$ can be similarly extracted from e^{M}.

Moreover, for the matrix $I + M/2^b$ (to be squared b times to estimate e^{M}), for all its squares, and also for e^{M}, the sum of the elements of each column is equal to 1. Due, however, to rounding errors the value of the above sums will not be exactly equal to 1.

As a measure of the harmful effect of the rounding errors on the matrix obtained at any stage of the successive squaring process, we define for each column the "column error", i.e., the difference between 1 and the actual value of the sum of the elements in the column.

In order to counter the cumulated effect of the errors we suitably smear the column error among the column elements so that their rounded sum is again 1, and we improve the efficacy by means of a simple rescaling.

We choose the smearing law as follows. If we make the reasonable assumption that rounding errors are independent gaussian random variables with zero mean and standard deviation proportional to each element value, the optimal law is a weighted mean with weights proportional to the squared values, as described below. Obviously this does not completely eliminate the rounding errors, but very likely greatly reduces their worst effects, as discussed in the next section.

The rescaling is simply performed by a suitable diagonal rescaling matrix, as detailed in section 2.7.

2.4. Justification of the method

We briefly describe here the rationale of our method, while a more detailed analysis is best deferred to a more specific work to be described elsewhere.

It can be easily theoretically predicted and experimentally verified that our method applied to the computation of the exponential of a real number gives a result almost exact within the machine precision, while a straightforward use of the iterated-squaring method provides at best only half of the exact digits. An example of such behaviour is reported in figure 1.

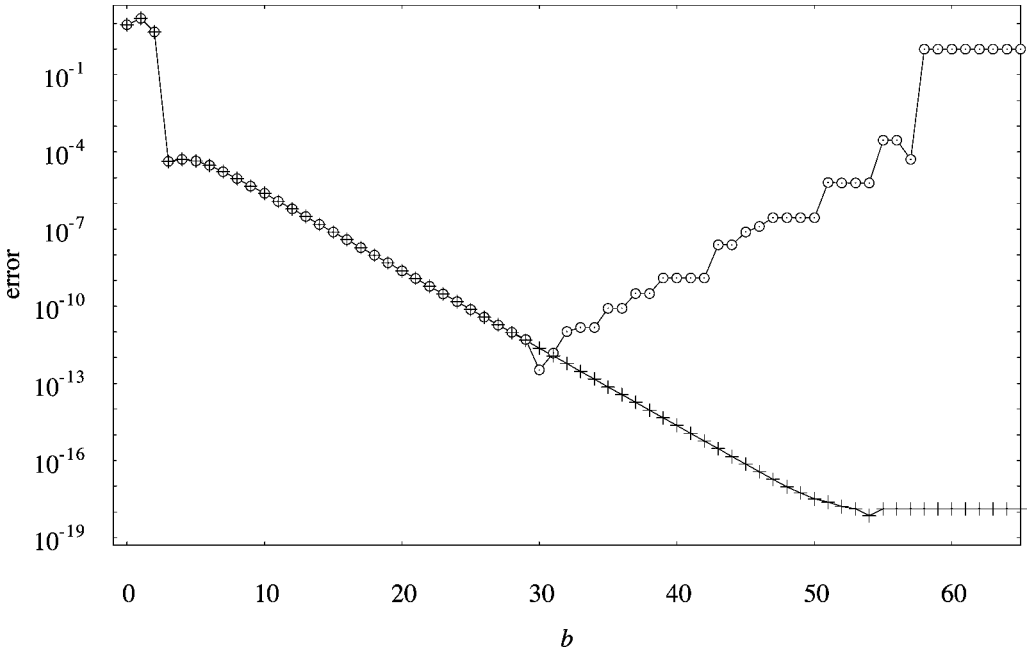

Figure 1. Graph, as a function of the number b of squarings (section 2.5), of the error in the computation of e^a, for $a = -\pi^2$, by a straightforward use of the iterated-squaring method $e^a \approx (1 + a/2^b)^{2^b}$ identified by points, and by our method identified by crosses.

Obviously, the above idea is equally valid also for the exponential of any diagonal matrix, and, from a numerical standpoint, small off-diagonal elements do not affect very much this behaviour.

In the case of a triangular matrix the simple diagonal rescaling can arbitrarily reduce the size of the off-diagonal elements, and the same applies to a block-triangular matrix.

We note that a triangular structure arises very often at least for important submatrices, whenever the problem exhibits a hierarchical structure due to chained causal links; this often occurs in many chemical problems.

More generally it often occurs that large off-diagonal elements can be greatly reduced by the above simple rescaling.

It appears, therefore, that while admittedly our method for controlling rounding errors cannot pretend to cope with any situation, it does provide a remarkably efficient tool, especially if compared to its great simplicity.

2.5. On the choice of the number of squaring operations

In order to obtain a rough estimate of a suitable value for the exponent b to be used in our computation we proceed as follows.

We neglect the numerical error, that should be hopefully sufficiently tamed by our procedure, and we consider the analytical error due to using a finite value for b.

It can be easily shown that the (analytical) relative matrix error E_t defined by

$$\left(1+\frac{Z}{2^b}\right)^{2^b} \equiv \mathrm{e}^Z \cdot (1+E_t)$$

is given approximately by

$$E_t \approx -\frac{1}{2}\cdot\frac{Z^2}{2^b},$$

and therefore, for any matrix norm,

$$\|E_t\| \approx \frac{1}{2}\cdot\frac{\|Z^2\|}{2^b} \leqslant \frac{1}{2}\cdot\frac{\|Z\|^2}{2^b}.$$

A reasonable estimate of the value of b needed to have $\|E_t\| < \varepsilon$, where ε is a preassigned tolerance (maximum tolerable value) for $\|E_t\|$, is, therefore, given by

$$b^* \equiv \mathrm{int}_+\left(\log_2\left(\frac{\|Z\|^2}{2\varepsilon}\right)\right),$$

where $\mathrm{int}_+(x)$ is the lowest positive integer greater or equal to x.

For the sake of simplicity we shall use the norm

$$\|Z\| = \max_k \sum_j |z_{kj}|,$$

and for sake of safety we shall take for b the value $b = b^* + 3$.

2.6. On the choice of the tolerance to compute the number of squaring operations

It is obviously meaningless to look for an analytical precision greater than the machine precision μ, and therefore, to take a value for the error tolerance ε smaller than μ.

On the other hand in most cases, as discussed above, the numerical error should be, thanks to our procedure, not too far from the machine precision μ.

Since we do not want to risk to have an analytical error greater than the numerical error, we choose as a reasonable value for the maximum admissible error ε just the machine precision μ.

2.7. Rescaling

Our simple diagonal rescaling aims at minimising the sum of the squares of all elements of the rescaled matrix: since diagonal element values are not affected by such a rescaling, the net effect is to reduce, as desired, the values of the harmful off-diagonal elements.

In more detail, if

$$\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_{n+1} \end{bmatrix}$$

is the rescaling matrix, and $\widetilde{A} = \Lambda \cdot A \cdot \Lambda^{-1}$ is the rescaled matrix, we choose Λ with the aim of minimising

$$S = \sum_{l,m} (\widetilde{a}_{lm})^2.$$

For the sake of simplicity, we limit ourselves to a truncated minimisation based on successive one-dimensional minimisations.

Obviously, at each step the required matrix exponential is obtained by reversing the rescaling

$$\mathrm{e}^{h\cdot A} = \Lambda^{-1} \cdot \mathrm{e}^{h\cdot \widetilde{A}} \cdot \Lambda.$$

2.8. The algorithm for the matrix exponential

A schematic representation of the implementation of the matrix exponential computation is given by the following sequential steps:

- Compute the initial matrix $P^{(0)} = I + 2^{-b} \cdot M$.
- Perform the b iterations (loop for $j = 1, \ldots, b$):
  - ∗ Compute the square $Q^{(j)} = P^{(j-1)} \cdot P^{(j-1)}$.
  - ∗ Compute the column errors

$$d_m^{(j)} = 1 - \sum_{k=1}^{n+2} q_{km}^{(j)}, \quad m = 1, \ldots, n+2.$$

  - ∗ Compute the smearing weights

$$w_{lm}^{(j)} = \frac{(q_{lm}^{(j)})^2}{\sum_{k=1}^{n+2} (q_{km}^{(j)})^2}, \quad l, m = 1, \ldots, n+2.$$

  - ∗ Perform the error smearing

$$p_{lm}^{(j)} = q_{lm}^{(j)} + w_{lm}^{(j)} \cdot d_m^{(j)}, \quad l, m = 1, \ldots, n+2.$$

  (end of the loop).
- The final matrix $P^{(b)}$ is the estimate of e^M, and contains the estimate of $\mathrm{e}^{h\cdot A}$ as explained above.

The required vector $\underline{\xi}$ is given by the first n elements of the column $n+1$ of $P^{(b)}$.

We note that this method for computing e^M requires $2bn^3 + O(n^2)$ floating point operations, and its computational cost is therefore comparable with, say, Lambert–Sigurdsson or Krylov methods.

We also note that in this section, for simplicity's sake, we have not explicitly considered the rescaling.

3. The software

We have produced an academic software implementation of our algorithm, in standard C language. The aim is twofold: to allow an immediate use of the software "as it is", and to provide a power user with a model software that can be easily modified and adapted to specific needs; a typical use could be, as suggested in the introduction, as an alternative integration module in complex chemical software packages.

We stress the fact that we have deliberately pursued program simplicity rather then maximum efficiency: our software is simple (i.e., it is based on few simple formulas), short (about 200 source code lines) and robust (i.e., it prevents boundary or numerical exceptions), and can therefore be easily translated into any programming language.

The complete source code, including the sample problem described in section 4.2, is available on request (e-mail address: `Valerio.Parisi@roma2.infn.it`).

The procedure starts at the initial time from the initial point and consists of one or more steps. The length of each step is obtained by successively halving the residual duration of the observation interval, until the requested tolerances are met for each variable. A step is accepted on the basis of the equivalence (within the preassigned tolerances) of the final points reached by a couple of two attempted consecutive steps of equal duration and by a test step of double duration.

For each step the matrix exponential is computed as described in section 2.8.

The required input parameters are:

- the C function defining the system of differential equations,
- the number of differential equations (that must be not greater than the "C-language #define constant" MAX_DIM; the constant MAX_DIM should be suitably set by the user, otherwise its default value is 10),
- the initial time and the length of the observation period,
- the minimum and maximum values of the time integration steplength to avoid respectively too long computing times and too rough simulation of transients,
- the requested tolerances (maximum admissible values) for the absolute and relative errors,
- the initial point,
- the pointer to the file where intermediate results are written.

The function overwrites the final point on the initial point and returns a value that normally is 1, and is 0 only if at least in one accepted substep the estimated error has not met the required tolerance.

The Jacobian matrix J must be computed by a user-supplied function. The user can obviously choose to compute J analytically or numerically; in the second case the user can use a function included in our software, that computes J by central differences: we note that for the Riccati equation the central-difference derivatives give the exact analytical value.

4. Numerical verification

In order to perform a preliminary numerical verification of our method, we have tested our software on the two sample problems described below. We have selected the first problem in a standard test set in order to perform a rough preliminary comparison with the state-of-the-art software. We have devised the second problem in order to have a very stiff realistic problem.

We feel that a thorough testing is best deferred to a more specific work to be described elsewhere.

4.1. Sample problem 1

This problem is taken from the standard Test Set for Initial Value Problems originally produced by CWI (the National Research Institute for Mathematics and Computer Science in the Netherlands) and now available at the address `http://hilbert.dm.uniba.it/~testset/` of the University of Bari, Italy. The test set includes documentation of the test problems, experimental results from a number of proven solvers, and FORTRAN subroutines providing a common interface to the defining problem functions. The first problem of the set concerning ordinary differential equations in explicit form (number 2 in the list) is problem HIRES [12,13], a chemical problem originating from plant physiology, and modeling a case of "High Irradiance Responses" (HIRES) by means of a chemical reaction involving eight reactants: it is a classical 8-variables stiff problem of Riccati form.

As it can be seen from the test set documentation, the solution is approached after a transient of about 300 time units, while the shortest characteristic time is of some tenth of time unit, with a ratio of the order of 10^3.

Table 1 reports the final values of the eight variables both for the reference solution (provided in the test set) and for the solutions obtained by our method, with three different values (10^{-4}, 10^{-5} and 10^{-6}) for the tolerance (having taken the same tolerance for the absolute and the relative error): our final values appear to be well compatible with the assumed tolerances.

For the above three tolerances our software performed respectively 62, 153 and 382 accepted integration steps, while the number of function evaluations was respectively

Table 1
Sample problem 1. Final values of the eight variables both for the reference solution (provided in the test set) and for three solutions obtained by our method, for three different tolerance values (having taken the same tolerance for the absolute and the relative error).

Reference solution	Tolerance 10^{-4}	Tolerance 10^{-5}	Tolerance 10^{-6}
$0.7371312573325668 \cdot 10^{-3}$	$0.73646291 \cdot 10^{-3}$	$0.73701731 \cdot 10^{-3}$	$0.73711309 \cdot 10^{-3}$
$0.1442485726316185 \cdot 10^{-3}$	$0.14411726 \cdot 10^{-3}$	$0.14422614 \cdot 10^{-3}$	$0.14424500 \cdot 10^{-3}$
$0.5888729740967575 \cdot 10^{-4}$	$0.58760503 \cdot 10^{-4}$	$0.58865913 \cdot 10^{-4}$	$0.58883878 \cdot 10^{-4}$
$0.1175651343283149 \cdot 10^{-2}$	$0.11744388 \cdot 10^{-2}$	$0.11754414 \cdot 10^{-2}$	$0.11756180 \cdot 10^{-2}$
$0.2386356198831331 \cdot 10^{-2}$	$0.23650970 \cdot 10^{-2}$	$0.23828468 \cdot 10^{-2}$	$0.23857924 \cdot 10^{-2}$
$0.6238968252742796 \cdot 10^{-2}$	$0.61592162 \cdot 10^{-2}$	$0.62257505 \cdot 10^{-2}$	$0.62370855 \cdot 10^{-2}$
$0.2849998395185769 \cdot 10^{-2}$	$0.28460476 \cdot 10^{-2}$	$0.28495060 \cdot 10^{-2}$	$0.28496910 \cdot 10^{-2}$
$0.2850001604814231 \cdot 10^{-2}$	$0.28539524 \cdot 10^{-2}$	$0.28504940 \cdot 10^{-2}$	$0.28503090 \cdot 10^{-2}$

Table 2
Sample problem 1. Comparison of the best and worst values of some run characteristics reported in the CWI test set documentation (possibly obtained by different solvers) with the values obtained with our method, with the same tolerance value 10^{-4}.

	CWI min value	CWI max value	Our method
Accepted steps	33	131	62
Function evaluations	168	665	369
Jacobian evaluations	10	31	369

369, 915 and 2289; the number of evaluations of the Jacobian matrix is in our software always equal to the number of function evaluations.

For the first tolerance value (10^{-4}) we compare in table 2 our run characteristics against the corresponding run characteristics obtained by the six solvers considered in the CWI Test Set. We feel that our performance (as measured by precision of the final values, number of accepted steps and number of function evaluations) can be well considered satisfactory. As for our number of Jacobian evaluations, which is clearly higher, we note that often the computation of the Jacobian is exceedingly cheap and its cost is negligible with respect to the overall computational cost. This especially applies to the case of a sparse Jacobian matrix of second order polynomial functions with only linear or at most bilinear terms (as in our example and more generally in almost all chemical problems, as discussed in the introduction).

For this reason we think that it is definitely not worthwhile to burden the software with sophisticated improvements aimed at reducing the number of Jacobian evaluations without an appreciable advantage (and possible performance degradation), and the more so in a software that is deliberately designed aiming at a great simplicity, readability, ease of use and of translation.

We finally note that clearly the significance of the required tolerances and of the number of integration steps is unavoidably different from package to package, and therefore, any comparison based only on such quantities has ultimately only an orientation value.

4.2. Sample problem 2

We consider here another chemical problem: a catalyser acts on a reactant with an input flow due to diffusion while the reactant deteriorates the catalyser. Such a case can be modelled by a simple two-variable system of differential equations with numerical values of the coefficients chosen in order to have a very stiff realistic problem, i.e., to have the reaction much quicker, and the degradation much slower, than the diffusion,

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 1 - x_1 - 10^3 \cdot x_1 x_2 \\ -10^{-3} \cdot x_1 x_2 \end{pmatrix},$$

where x_1 and x_2 are the concentrations of the reactant and of the catalyser within a compartment. Assuming unitary external concentration and diffusion coefficient, the reactant input flow is $1 - x_1$, while the reaction velocity is taken as $10^3 \cdot x_1 x_2$, and the deterioration rate of the catalyser is taken as $10^{-3} \cdot x_1 x_2$.

The asymptotic values $x_1 = 1,\ x_2 = 0$ of the solution, with the initial conditions $x_1 = 0,\ x_2 = 1$ are approached after a transient of about 10^6 time units, while the shortest characteristic time is about 10^{-3}, with a ratio of the order of 10^9 that gives a reasonable indication of a great stiffness.

In order to get a clear picture of the transient we have observed the numerical solution from 0 to $2 \cdot 10^6$ time units. To meet the required absolute and relative error tolerances (respectively 10^{-4} and 10^{-2}) our software required 156 time integration steps, showing a good insensitivity to stiffness. The behaviour of x_1 and x_2 versus time over the integration interval is depicted in figure 2.

For this problem we provide both the source code and the corresponding output produced by our software.

Since comparison performances are not available, we have solved the same problem with a simple Euler integrator, both to check our solution and to compare the performances. Obviously the Euler method can be hardly considered a state-of-the-art method, but is nevertheless still widely used (see, for example, [14]). We note that any explicit method would require a number of steps of the order of 10^{10}.

5. Conclusions

We have examined various methods for the numerical integration of initial value problems for systems of ordinary differential equations and we may briefly summarise the main features as follows:

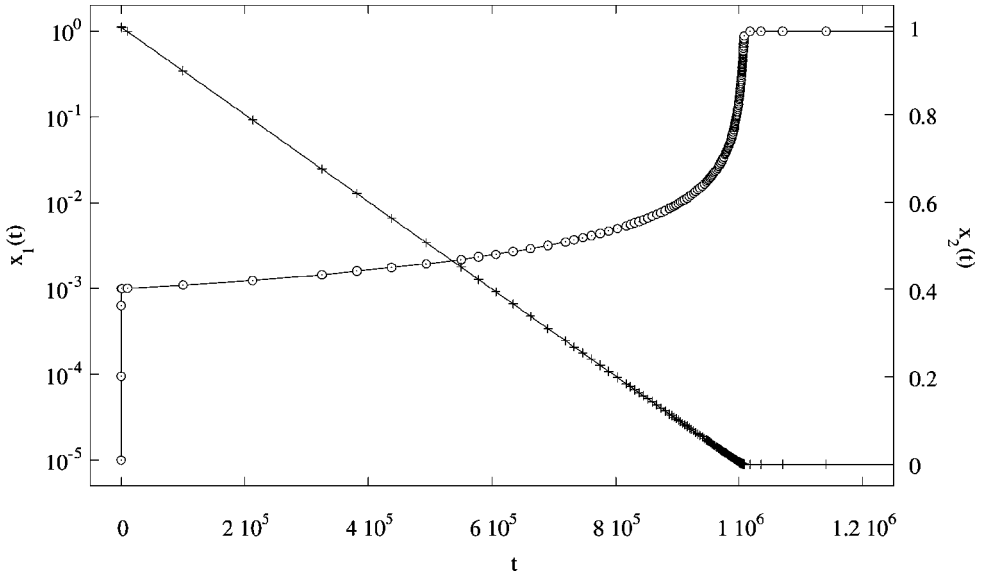

Figure 2. Sample problem 2. Numerical solutions versus time. Points: x_1 values (log scale on the left vertical axis), crosses: x_2 values (linear scale on the right vertical axis).

- explicit methods are instability-prone in ill-conditioned problems,
- implicit methods (or their linearised versions as Lambert–Sigurdsson methods) overcome the above drawback but at the expense of risking spurious oscillations or dampings,
- naïve exponential methods (i.e., methods not based on Krylov subspaces approximations) are exact for linear problems and overcome both of the above drawbacks but use methods for evaluating the matrix exponential that are often unpractical,
- improved Krylov-based exponential methods overcome all the preceding drawbacks but at the expense of neglecting almost all eigenvalues, and may, therefore, again introduce inaccuracies (such as spurious oscillations or dampings) also in linear or quasilinear cases.

We have proposed an attractive novel method that is largely independent from ill-conditioning and is especially precise for quasi-linear problems, since it is exact for linear problems. The method includes an automatic correction of rounding errors, is not too expensive computationally, and has been implemented as a simple and robust C-language program (freely available from the authors) that can be easily inserted in large simulation packages.

Of course, our method is also suitable for any nonlinear problem, since, as most other methods, it includes an automatic control of the time-integration steplength. We note, however, that unlike in other methods, in our method the steplength control is

aimed only at coping with the nonlinearity, allowing therefore longer steps for a preassigned accuracy.

A preliminary numerical verification has been performed, with encouraging results, on two sample problems.

References

[1] J.D. Lambert and S.T. Sigurdsson, SIAM J. Numer. Anal. 9 (1972) 715.
[2] G.R. Gavalas, *Nonlinear Differential Equations of Chemically Reacting Systems* (Springer-Verlag, Berlin, 1968).
[3] E.H. Kerner, J. Math. Phys. 22 (1981) 1366.
[4] M. Hochbruck, C. Lubich and H. Selhofer, SIAM J. Sci. Comput. 19 (1998) 1552.
[5] C. Moler and C. van Loan, SIAM Review 20 (1978) 801.
[6] G.H. Golub and C.F. van Loan, *Matrix Computations* (Johns Hopkins University Press, Baltimore, 1996).
[7] Y. Saad, SIAM J. Numer. Anal. 29 (1992) 209.
[8] R.B. Sidje, ACM TOMS 24 (1998) 130.
[9] D.E. Knuth, *The Art of Computer Programming*, Vol. 2, *Seminumerical Algorithms*, 2nd ed. (Addison-Wesley, Reading, MA, 1981).
[10] W. Kahan, CACM 8 (1965) 40.
[11] W. Kahan, in: *Proceedings of the IFIP Congress, Information Processing 71*, Ljubljana, August 1971, eds. C.V. Freiman, J.E. Griffith and J.L. Rosenfeld (North-Holland, Amsterdam, 1972) pp. 1214–1239.
[12] E. Schäfer, J. Math. Biol. 2 (1975) 41.
[13] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, 2nd ed. (Springer-Verlag, New York, 1996).
[14] M. Tomita, K. Hashimoto, K. Takahashi, T.S. Shimizu, Y. Matsuzaki, F. Miyoshi, K. Saito, S. Tanida, K. Yugi, J.C. Venter and C.A. Hutchison III, Bioinformatics 15 (1999) 72.