

**Babeş–Bolyai University of Cluj-Napoca
Faculty of Mathematics and Informatics
Department of Applied Mathematics**

Doctoral Report

**Scientific conductor:
Dr. Ioan A. Rus**

**PhD student:
Egri Edith**

2006

**Numerical and approximative
methods in some mathematical
models**

Universitatea Babeş–Bolyai, Cluj-Napoca
Facultatea de Matematică și Informatică
Matematică Aplicată

Referat de doctorat

Conducător științific,
Prof. dr. Ioan A. Rus

Doctorand,
Egri Edith

2006

Metode numerice și aproximative în unele modele matematice

Contents

1	Introduction	2
2	The computational singular perturbation method	3
2.1	The theory of CSP	4
2.2	A simple example	6
3	Euler method	8
4	Runge-Kutta methods	10
4.1	Runge-Kutta method of order 2	10
4.2	Runge-Kutta method of order 4	10
4.3	Controlling the step size	10
5	The Adomian's decomposition method	11
5.1	The theory of ADM	11
5.2	Applications	12
6	Stiff systems	15
6.1	Introduction	15
6.2	Stiff systems in chemistry	17
6.3	Handling the stiffness	18
6.4	Rosenbrock methods	20
6.5	Semi-implicit extrapolation method	22
6.6	Gear algorithm	23
6.7	Multistep, multivalued, and predictor-corrector methods	24
7	Positive and conservative numerical methods	26
7.1	Stiff systems related to mass action kinetics	27
7.1.1	Explicit Euler scheme	29
7.1.2	Linearized implicit Euler scheme	29
7.1.3	Runge-Kutta's fourth-order scheme	30
7.1.4	A semi-implicit scheme	30
7.1.5	A fully implicit scheme	32
7.1.6	A second-order positive scheme	34
7.1.7	Two better second-order positive schemes	35
7.1.8	A second-order diagonally implicit Runge-Kutta scheme	36
8	Solving stiff IVP's with Maple	36

1 Introduction

Historically, differential equations have originated in chemistry, physics, and engineering. More recently they have also arisen in models in medicine, biology, anthropology, and the like. Here we mainly restrict our attention to ordinary differential equations; a discussion of partial differential equations is a much more complicated issue. We focus on initial value problems and present some of the more commonly used methods for solving such problems.

It is often difficult to find the analytic or exact solution to many differential equations. This may be because the equation is non-linear or has coefficients that vary with time.

Alternative strategies to approximate the solution of a differential equations would be:

- simplify the ordinary differential equation (ODE) and solve it analytically;
- use methods for directly approximating the solution.

As an application for the first strategy, we will present later the computational singular perturbation (CSP) method. Other well-known methods would be quasi steady state approximation and partial equilibrium approaches.

There are many methods for finding direct approximate solutions to differential equations. These methods are referred to by a variety of different names including: numerical methods, numerical integration, or approximate solutions, among others.

The numerical methods for solving ordinary differential equations are methods of integrating a system of first order differential equations, since higher order ordinary differential equations can be reduced to a set of first order ODEs.

Errors enter into the numerical solution of initial value problems (IVPs) from two sources. The first is discretization error and depends on the method being used. The second is computational error which includes such things as roundoff error, the error in evaluating implicit formulas, etc. In general, roundoff error can be controlled by carrying enough significant figures in the computation. The control of other computational errors again depends on the method being used.

There are two measures of discretization error commonly used in discussing the accuracy of numerical methods for solving IVPs. The first is true or global error. Global error is simply the difference between the true solution and the numerical approximation to it. Even though this is the error in which we are usually interested, it is a relatively difficult and expensive to estimate. The other measure of error is local error. It is the error incurred in taking a single step using a numerical method.

Three major types of practical numerical methods for solving initial value problems for ODEs are:

- Runge-Kutta methods,
- Richardson extrapolation and its particular implementation as the Bulirsch- Stoer method,
- predictor-corrector methods.

A brief description of each of these types follows.

1. Runge-Kutta methods propagate a solution over an interval by combining the information from several Euler-style steps (each involving one evaluation of the right-hand f 's), and then using the information obtained to match a Taylor series expansion up to some higher order.
2. Richardson extrapolation uses the powerful idea of extrapolating a computed result to the value that would have been obtained if the step size had been very much smaller than it actually was. In particular, extrapolation to zero step size is the desired goal. The first practical ODE integrator that implemented this idea was developed by Bulirsch and Stoer, and so extrapolation methods are often called Bulirsch-Stoer methods.

We have to mention here that these techniques are not for differential equations containing nonsmooth functions, and they are not particularly good for differential equations that have singular points inside the interval of integration.

3. Predictor-corrector methods store the solution along the way, and use those results to extrapolate the solution one step advanced; they then correct the extrapolation using derivative information at the new point. These are best for very smooth functions.

In what follows, we mainly will deal with the first type of numerical method, i.e. with the Runge-Kutta methods.

The framework of the report is the following: section 2 is about the computational singular perturbation method illustrated by an example. Then forward and backward Euler methods are presented. A more accurate and more elaborate technique, the Runge-Kutta methods are mentioned in section 4. The next section is meant for the Adomian's decomposition method. After that stiff systems are discussed and numerical methods are given for handling the stiffness. Section 7 deal with positive and conservative numerical integration methods for ODEs which describe chemical kinetics. Then the last section contain a few Maple codes and examples to solve stiff IVPs.

2 The computational singular perturbation method

When an investigator is confronted with an unfamiliar problem in chemical kinetics, the traditional first step is to identify the relevant chemical species and the important elementary reactions which occur among them. After establishing a complete model of the reaction system, it is usually desirable to obtain a simplified model by taking advantage of available approximations. For sufficiently simple problems, conventional analytical methods can be used.

Recently, databases containing extensive, reliable and up-to-date data for certain reaction systems are available. Computations using complete models from such databases can now be routinely carried out. In this new computational era, it is no longer necessary to pick out only the relevant chemical species and the important elementary reactions because inclusion of benign superfluous terms in the formulation is not a problem. An option increasingly available to modern theoreticians is to first generate a complete model numerical solution, examine the resulting data to discern significant and interesting causes

and effects - making additional diagnostic runs if necessary – and then try to propose simplifications and approximations.

The goal of developing a general theory of singular perturbation which can handle any system of nonlinear first order ODEs in a programmable manner seems very ambitious in chemistry. The conventional method, using the partial equilibrium and quasi steady state approximations, is only viable for relatively simple problems for which adequate amount of experience and intuition have been accumulated, and that the algebra involved is manageable. For massively complex problems a better method was found, the computational singular perturbation method (CSP). It exploits the power of the computer to do simplified kinetics modeling. A CSP computation not only generates the numerical solution of the given problem, but also the simplified equations in terms of the given information. This theory was developed by S. H. Lam and published in 1985 in [Lam85] and it can be used to deal with massively complex problems, but only for boundary–layer type problems where all fast modes eventually decay exponentially. Fortunately, most problems in chemical kinetics are of this type. The basic strategy of CSP is to uncouple the fast, exhausted modes from the slower, currently active modes through an intelligent choice of basis vectors. In this way we get a simplified model which can generate approximate solutions to the full chemistry model.

2.1 The theory of CSP

Consider a reaction system of N unknowns¹ denoted by the column state vector $\mathbf{y} = [y_1, y_2, \dots, y_N]^\top$. The governing system of ODE is:

$$\frac{d\mathbf{y}}{dt} = \mathbf{g}(\mathbf{y})$$

where

$$\mathbf{g}(\mathbf{y}) = \sum_{r=1}^R \mathbf{s}_r F^r(\mathbf{y}),$$

R is the number of elementary reactions being included in the reaction system, \mathbf{s}_r and $F^r(\mathbf{y})$ are the stoichiometric vector and the reaction rate of the r -th elementary reactions, respectively. The N -dimensional column vector \mathbf{g} is the overall reaction rate vector, and can be interpreted as the velocity vector of \mathbf{y} in the N -dimensional y -space. For a massively complex problem, N and R can be large numbers, and the accuracy or reliability of the available rate constants is usually less than ideal.

The physical problem is completely specified by $\mathbf{g}(\mathbf{y})$, a non-linear function of \mathbf{y} obtained by summing all the physical processes which contribute to the time rate of change of \mathbf{y} . Since \mathbf{g} is a N -dimensional vector, it can always be expressed in terms of a set of arbitrarily chosen N linearly independent column basis vectors, $\mathbf{a}_i(t)$, $i = 1, 2, \dots, N$. Then the set of inverse row basis vectors, $\mathbf{b}^i(t)$, $i = 1, 2, \dots, N$, can be computed from the orthonormal relations:

$$\mathbf{b}^i \odot \mathbf{a}_j = \delta_j^i, \quad i, j = 1, 2, \dots, N. \quad (2.1)$$

¹The N -dimensional column vector may include temperature, total density, etc. in addition to chemical species

Here the operator \odot denotes the dot product of the N -dimensional vector space.

In this case a column vector \mathbf{g} has an alternative representation:

$$\mathbf{g} = \sum_{i=1}^N \mathbf{a}_i f^i, \quad (2.2)$$

where

$$f^i := \mathbf{b}^i \odot \mathbf{g} = \sum_{r=1}^R B_r^i F^r, \quad i = 1, 2, \dots, N, \quad (2.3)$$

and

$$B_r^i = \mathbf{b}^i \odot \mathbf{s}_r, \quad i = 1, 2, \dots, N. \quad (2.4)$$

Each of the additive terms in (2.2) represents a *reaction mode*, or simply *mode*. The *amplitude* and *direction* of the i th mode are f^i and \mathbf{a}_i , respectively. Eventually, CSP provides an algorithm to compute an approximation to the "ideal" set of basis vectors for the derivation of the simplified models.

The physical representation of \mathbf{g} uses the physically meaningful (and time-independent) stoichiometric vectors as the default column basis vectors.

Now, differentiating (2.3) with respect to time along a solution trajectory $\mathbf{y}(t)$, we obtain:

$$\frac{df^i}{dt} = \sum_{j=1}^N \Lambda_j^i f^j, \quad i = 1, 2, \dots, N, \quad (2.5)$$

where

$$\Lambda_j^i := \left[\frac{d\mathbf{b}^i}{dt} + \mathbf{b}^i \odot \mathbf{J} \right] \odot \mathbf{a}_j, \quad i, j = 1, 2, \dots, N,$$

and \mathbf{J} denotes the Jacobian matrix of \mathbf{g} .

A set of basis vectors $\mathbf{a}_i(t)$ is said to be *ideal* if

- the inverse row vectors $\mathbf{b}^i(t)$ can be accurately computed for all time interval of interest,
- $\Lambda_j^i(t)$ is diagonal,
- the diagonal elements of $\Lambda_j^i(t)$ are ordered in descending magnitudes.

For linear problems where \mathbf{J} is a constant matrix, the ideal basis vectors would be the (constant) ordered eigenvectors of \mathbf{J} . For nonlinear problems, the eigenvectors of \mathbf{J} are time-dependent, and they do not diagonalize Λ_j^i .

The reciprocal of an eigenvalue, called the *time scale*, has the dimension of time, and shall be denoted by $\tau(i)$. Ordering them in increasing magnitudes, we have:

$$|\tau(1)| < \dots < |\tau(i)| < \dots < |\tau(N)|,$$

which provides an approximate speed ranking of the "eigen-modes".

Events whose time scales are shorter than Δt are not of interest. Hence, the group of M modes which satisfy: $|\tau(m)| < \Delta t$, $m = 1, 2, \dots, M$, are considered *fast modes*, and

all others are considered the *slow modes*. The fastest group of active slow modes are the *rate-controlling modes*. Slow modes with negligible amplitudes are called *dormant modes*.

The method of CSP does not attempt to find the ideal set of basis vectors even when \mathbf{g} is linear. Instead, it assumes that, at any moment in time, a *trial* set of ordered basis vectors is somehow available, that the first M fastest modes are exhausted as measured by some criterion, and generates from this trial set a new *refined* set of basis vectors, a_i^0 and b_i^0 , $i = 1, 2, \dots, N$ using a two-step *refinement* procedure (see [Lam93]). When recursively applied, the refinement procedure successively weakens the coupling between the fast modes and the slow modes.

Using the refined basis vectors, the governing system of ODEs become simpler.

For a more detailed description of the method and some other related works of S. H. Lam see [Lam93, Lam95].

2.2 A simple example

Take a simple hypothetical reaction system (see [Lam94]) with state vector $\mathbf{y} = [A, B, C]^\top$, where A and B are chemical concentrations and C is temperature. The elementary reactions are:



where ΔH_1 , ΔH_2 and ΔH_3 are the heats of reaction of the respectively reactions. The (generalized) stoichiometric vectors and the reaction rates are:

$$\begin{aligned} s_1 &= [-2, 1, \Delta H_1]^\top, & F^1 &= k_1(A^2 - K_1B), \\ s_2 &= [-1, 1, \Delta H_2]^\top, & F^2 &= k_2(A - K_2B), \\ s_3 &= [1, -2, \Delta H_3]^\top, & F^3 &= k_3(B^2 - K_3A), \end{aligned}$$

where the reaction rate coefficients k_1, k_2, k_3 and the equilibrium constants K_1, K_2, K_3 are known and their dependence on C is negligible. If we separately identify the forward and reverse reaction rates, i.e. take $F^r = F_+^r - F_-^r$ ($r = 1, 2, \dots, R$), the induced kinetic differential equation system reads:

$$\begin{aligned} \frac{dA}{dt} &= -2F^1 - F^2 + F^3, \\ \frac{dB}{dt} &= F^1 + F^2 - 2F^3, \\ \frac{dC}{dt} &= \Delta H_1 F^1 + \Delta H_2 F^2 + \Delta H_3 F^3, \end{aligned} \tag{2.7}$$

To make things concrete the rate coefficients are given numerical values:

$$\begin{aligned} k_1 &\approx 10^4 \text{cc/mole} \cdot \text{second}, & K_1 &\approx 1.1 \times 10^{-2} \text{mole/cc}, \\ k_2 &\approx 10^{-1} \text{cc/mole} \cdot \text{second}, & K_2 &\approx 1.1 \times 10^2 \text{mole/cc}, \\ k_3 &\approx 10^4 \text{cc/mole} \cdot \text{second}, & K_3 &\approx 0.8 \times 10^{-8} \text{mole/cc}, \end{aligned}$$

and

$$\begin{aligned}\Delta H_1 &\approx +1.1 \times 10^4 \text{cc}\cdot\text{K}/\text{mole}, \\ \Delta H_2 &\approx +1.0 \times 10^5 \text{cc}\cdot\text{K}/\text{mole}, \\ \Delta H_3 &\approx -2.9 \times 10^5 \text{cc}\cdot\text{K}/\text{mole}.\end{aligned}$$

The initial conditions are also given numerical values:

$$A(0) \approx 1.5 \times 10^{-4} \text{mole}/\text{cc}, \quad B(0) \approx 0.1 \times 10^{-6} \text{mole}/\text{cc}, \quad C(0) \approx 300\text{K} \quad (2.8)$$

To obtain the reduced chemistry system, we need to remove reactions which participation is negligible and remove reactants which are not important to the issues at hand.

Experience and intuition (which is an important requirement of partial equilibrium or quasi steady state approximations) can play no role here because the problem is hypothetical, and indeed may not even make chemical sense. Note that concerning our problem, detailed balance would require $K_3 = K_1/(K_2)^3$, thermodynamics would require $\Delta H_3 = \Delta H_1 - 3\Delta H_2$, and the law of mass action would require s_r and F^r to be consistent.

On how to apply conventional methodologies to this example one can consider the paper of S. H. Lam, [Lam94].

Here we only will show the CSP method on this example. For our example, the default set is:

$$\begin{aligned}\mathbf{a}_1 = \mathbf{s}_1 &= [-2, 1, \Delta H_1]^\top, \\ \mathbf{a}_2 = \mathbf{s}_2 &= [-1, 1, \Delta H_2]^\top, \\ \mathbf{a}_3 = \mathbf{s}_3 &= [1, -2, \Delta H_3]^\top,\end{aligned}$$

Using this set, the inverse row vectors can easily be computed:

$$\begin{aligned}\mathbf{b}_1 &= [2\Delta H_2 + \Delta H_3, \Delta H_2 + \Delta H_3, 1]/H, \\ \mathbf{b}_2 &= [-2\Delta H_1 - \Delta H_3, -\Delta H_1 - 2\Delta H_2, -3]/H, \\ \mathbf{b}_3 &= [-\Delta H_1 + \Delta H_2, -\Delta H_1 + 2\Delta H_2, -1]/H,\end{aligned}$$

where $H := \Delta H_1 - 3\Delta H_2 - \Delta H_3$.

Using the given input numerical data, we have: $H = 10^3 \text{cc} - \text{K}/\text{mole}$.

It can be verified that at $t = 0$, the amplitudes of the modes are:

$$\begin{aligned}f^1 = \mathbf{b}_1 \odot \mathbf{g} = F^1 &= 2.14 \times 10^{-4} \text{mole}/\text{cc}\text{-second}, \\ f^2 = \mathbf{b}_2 \odot \mathbf{g} = F^2 &= 1.39 \times 10^{-5} \text{mole}/\text{cc}\text{-second}, \\ f^3 = \mathbf{b}_3 \odot \mathbf{g} = F^3 &= -1.19 \times 10^{-8} \text{mole}/\text{cc}\text{-second},\end{aligned}$$

In terms of these basis vectors, the original reaction system becomes:

$$\frac{d\mathbf{y}}{dt} = \mathbf{a}_1 f^1 + \mathbf{a}_2 f^2 + \mathbf{a}_3 f^3. \quad (2.9)$$

We can rewrite (2.9) in long-hand notation as follows:

$$\begin{aligned}\frac{dA}{dt} &= -2f^1 + f^2 + \Delta H_1 f^3, \\ \frac{dB}{dt} &= -f^1 + f^2 + \Delta H_2 f^3, \\ \frac{dC}{dt} &= f^1 - 2f^2 + \Delta H_3 f^3.\end{aligned} \quad (2.10)$$

The CSP idea is very simple: instead of using the physically meaningful stoichiometric vectors as the default basis vectors, lets exploit the theoreticians prerogative of trying different alternatives-may be something else works better.

Now, we assume that at the beginning we have no idea which reaction is fast. The eigenvalues $\lambda(i)$ and eigenvectors of \mathbf{J} at $t = 0$ can be computed numerically. We have:

$$\begin{aligned}\lambda(1) &= -1.27 \times 10^2/\text{second}, \\ \lambda(2) &= -0.173/\text{second}, \\ \lambda(3) &= 0.00/\text{second},\end{aligned}$$

indicating that there is a fast mode with time scale of the order of 10^{-2} seconds, followed by a slower mode with time scale of the order of about 10^1 seconds.

Taking the right (column) eigenvectors α_i and left (row) eigenvectors β^i , ranked in order of decreasing speed, these may be used as our time-independent trial basis vectors for $t \geq 0$ but they diagonalize Λ_j^i only at $t = 0$. Since our time resolution of interest is in seconds, only the first mode can be considered fast. Hence, $M = 1$.

The first eigenmode (which has the largest eigenvalue magnitude) is obviously the fastest. In what follows, we shall choose

$$\mathbf{a}_1 = [-2.00, 1.00, 1.10 \times 10^4]^\top$$

and

$$\mathbf{b}^1 = [-9.00 \times 10^1, -1.90 \times 10^2, 1.00 \times 10^{-3}]$$

instead of trial fast basis vectors for $t \geq 0$.

Since we are not interested in the rapid transient period which lasts tens of milliseconds, the main issue now is to find the adjusted initial conditions for the simplified model which governs the slow evolutionary period.

In the rapid transient period, \mathbf{y} adjusts rapidly in such a way that the amplitude of the fastest mode approaches zero. Here, the amplitude of the fastest mode at $t = 0$ is $f^1 = F^1 = 2.14 \times 10^{-4}$. Making the radical correction using the trial fast basis vectors, we obtain the following adjusted initial condition at $t = 0^+$ (see [Lam94]):

$$\mathbf{y}(0^+) = \mathbf{y}(0) + \Delta y_{rc} = [1.46 \times 10^{-4}, 1.95 \times 10^{-6}, 300.02]^\top,$$

which yields a much smaller amplitude, $f^1 = F^1 = -1.36 \times 10^{-7}$.

From the mathematical point of view, reduced chemistry modeling can be routinely accomplished once an appropriate set of basis vectors which approximately decouples the fast and slow modes is available. In conventional methods, such fast basis vectors are identified by guessing – based primarily on experience and intuition of the investigator. The CSP method simply provides an iterative algorithm to find such decoupling basis vectors using an iterative (refinement) procedure.

3 Euler method

In the 18th century Leonhard Euler invented a simple scheme for numerically approximating the solution to an ODE. It results from ignoring second and higher order terms in Taylor series to approximate the solution.

Thus, to solve an IVP

$$y'(x) = f(x, y), \quad y(x_0) = y_0, \quad (3.1)$$

we have to pick the marching step h and compute

$$\begin{aligned} k_1 &= hf(x_n, y_n), \\ y_{n+1} &= y_n + k_1 + \mathcal{O}(h^2). \end{aligned}$$

It is an explicit method, i.e., y_{n+1} is given explicitly in terms of known quantities.

The formula is unsymmetrical: it advances the solution through an interval h , but uses derivative information only at the beginning of that interval. That means that the steps error is only one power of h smaller than the correction, i.e $\mathcal{O}(h^2)$.

Implicit methods can be used to replace explicit ones in cases where the stability requirements of the latter impose stringent conditions on the time step size. However, implicit methods are more expensive to be implemented for non-linear problems since y_{n+1} is given only in terms of an implicit equation, which have to be solved to find y_{n+1} . One often uses functional iteration or (some modification of) the Newton-Raphson method to achieve this. The implicit analogue of the explicit FE method is the backward Euler (BE) method, which reads as follows:

$$\begin{aligned} k_1 &= hf(x_{n+1}, y_{n+1}), \\ y_{n+1} &= y_n + k_1 + \mathcal{O}(h^2). \end{aligned}$$

There are several reasons that Euler's method is not recommended for practical use. Among them, the method is not very accurate when compared to other methods run at the equivalent step size, and neither is it very stable. Clearly, there is a trade-off between accuracy and complexity of calculation which depends heavily on the chosen value for h . In general as h is decreased the calculation takes longer but is more accurate. However, if h is decreased too much the slight rounding that occurs in the computer (because it cannot represent real numbers exactly) begins to accumulate enough to cause significant errors. For many higher order systems, it is very difficult to make the Euler approximation effective.

One possibility is to use not only the previously computed value y_n to determine y_{n+1} , but to make the solution depend on more past values. This yields a so-called multistep method. Almost all practical multistep methods fall within the family of linear multistep methods, which have the form

$$\begin{aligned} \alpha_k y_{n+k} + \alpha_{k-1} y_{n+k-1} + \cdots + \alpha_0 y_n &= \\ = h[\beta_k f(x_{n+k}, y_{n+k}) + \beta_{k-1} f(x_{n+k-1}, y_{n+k-1}) + \cdots + \beta_0 f(x_n, y_n)]. \end{aligned}$$

Another possibility is to use more points in the interval $[t_n, t_{n+1}]$. This leads to the family of Runge-Kutta methods, named after Carl Runge and Martin Kutta.

4 Runge-Kutta methods

4.1 Runge-Kutta method of order 2

In this case, to solve (3.1) we have to pick the marching step h and compute

$$\begin{aligned}k_1 &= hf(x_n, y_n), \\k_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right), \\y_{n+1} &= y_n + k_2 + \mathcal{O}(h^3).\end{aligned}$$

As indicated in the error term, $\mathcal{O}(h^3)$, this symmetrization cancels out the first-order error term, making the method second order². Thus, the second-order Runge-Kutta method, also known as the *midpoint method*, improves the Euler method by adding a midpoint in the step which increases the accuracy by one order.

There are many ways to evaluate the right-hand side $f(x, y)$ that all agree to first order, but that have different coefficients of higher-order error terms. Adding up the right combination of these, we can eliminate the error terms order by order. That is the basic idea of the Runge-Kutta method.

4.2 Runge-Kutta method of order 4

The fourth-order Runge-Kutta method is by far the ODE solving method most often used. In this case, to solve (3.1) we have to pick the marching step h and compute

$$\begin{aligned}k_1 &= hf(x_n, y_n), \\k_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right), \\k_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right), \\k_4 &= hf(x_n + h, y_n + k_3).\end{aligned}$$

$$\text{Then } y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + \mathcal{O}(h^5).$$

4.3 Controlling the step size

A good ODE integrator should exert some adaptive control over its own progress, making frequent changes in its step size. Usually the purpose of this adaptive step size control is to achieve some predetermined accuracy in the solution with minimum computational effort.

The main principle is that to take each integration step twice, once as a full step, then, independently, as two half steps. In this case the difference between the two numerical

²A method is conventionally called n th order if its error term is $\mathcal{O}(h^{n+1})$.

estimates ($\Delta := y_1 - y_2$) will be a convenient indicator of the truncation error. To keep the desired degree of accuracy (Δ_0), the step size is adjusted according to: new time step = old time step $\times \left(\frac{\Delta_0}{\Delta}\right)^{0.2}$.

More about Runge–Kutta methods with adaptive step size choice, see [Pre88].

5 The Adomian’s decomposition method

As we mentioned earlier, most realistic system of ordinary differential equations do not have analytic solutions so that a numerical technique must be used. But in the case of using the Adomian’s decomposition method (ADM) [Ado94] is that it can provide analytical approximation to the problems. It is a relatively new field of study, but has already found applications in many branches of physics and engineering.

ADM has been acclaimed as a significant powerful method for systems of nonlinear equations. It is useful for obtaining both analytic and numerical approximations of linear or nonlinear differential equations and it is also quite straightforward to write computer codes.

An implementation of the ADM in chemical applications can be found in [Kay04]. In the mentioned work D. Kaya have illustrated the advantages and simplicity of using the decomposition method over traditional methods, namely the simple Taylor series method and fourth-order RungeKutta method in terms of numerical comparisons.

5.1 The theory of ADM

We will present the method to obtain approximate solution of the (kinetic) system

$$\frac{dy_i}{dt} = f_i(t, y_1(t), y_2(t), \dots, y_p(t)), \quad y_i(0) = \alpha_i, \quad i = 1, 2, \dots, p \quad (5.1)$$

where f_1, f_2, \dots, f_p are real continuous functions defined on same domain D ; α_i is a specified constant vector, and $y_i(t)$ is the solution vector.

In the decomposition method, equation (5.1) is approximated by the operators in the following form:

$$Ly_i(t) = f_i(t, y_1(t), y_2(t), \dots, y_p(t)), \quad i = 1, 2, \dots, p \quad (5.2)$$

where L symbolizes $\frac{d}{dt}$.

Assuming the inverse of the operator $L^{-1} = \int_0^t (\cdot) dt$ exists, then applying this inverse to (5.2) yields

$$L^{-1}Ly_i(t) = L^{-1}f_i(t, y_1(t), y_2(t), \dots, y_p(t)), \quad (5.3)$$

where $i = 1, 2, \dots, p$. Therefore, it follows that

$$y_i(t) = y_i(0) + L^{-1}f_i(t, y_1(t), y_2(t), \dots, y_p(t)),$$

$i = 1, 2, \dots, p$. The decomposition method consists of representing $y_i(t)$ in the decomposition series form given by

$$y_i(t) = \sum_{n=0}^{\infty} f_{i,n}(t, y_1(t), y_2(t), \dots, y_p(t)),$$

where the components $y_{i,n}(t)$, $n \geq 1$ and $i = 1, 2, \dots, p$ can be computed readily in a recursive manner. We may define the components $y_{i,n}(t)$, $n \geq 1$ and $i = 1, 2, \dots, p$ of the decomposition series by the following recursive relationship:

$$y_i(0) = \alpha_i, \quad i = 1, 2, \dots, p, \quad (5.4)$$

$$y_{i,n+1}(t) = L^{-1}f_{i,n}(t, y_1(t), y_2(t), \dots, y_p(t)), \quad n \geq 0. \quad (5.5)$$

Since $y_{i,0}$ is known

$$y_{i,1}(t) = L^{-1}f_{i,0}(t, y_1(t), y_2(t), \dots, y_p(t)),$$

$$y_{i,2}(t) = L^{-1}f_{i,1}(t, y_1(t), y_2(t), \dots, y_p(t)),$$

⋮

Then, the series solution is given by

$$y_i(t) = y_{i,0}(t) + \sum_{n=1}^{\infty} \{L^{-1}f_{i,n}(t, y_1(t), y_2(t), \dots, y_p(t))\},$$

$i = 1, 2, \dots, p$. The solution $y_i(t)$ must satisfy the requirements imposed by the initial conditions.

Using the decomposition method it provides a reliable technique that requires less work if compared with the traditional techniques. Moreover, this method does not need discretization of the problem to obtain numerical results. We can evaluate the approximate solution $\phi_{i,m}$, by using the m -term approximation

$$\phi_{i,1} = y_{i,0}(t),$$

$$\phi_{i,2} = y_{i,0}(t) + y_{i,1}(t),$$

$$\phi_{i,3} = y_{i,0}(t) + y_{i,1}(t) + y_{i,2}(t),$$

⋮

$$\phi_{i,m} = y_{i,0}(t) + y_{i,1}(t) + y_{i,2}(t) + \dots + y_{i,m-1}(t).$$

5.2 Applications

In order to illustrate the decomposition technique discussed above, let see an example which arose in a chemistry problem:

$$\begin{aligned} \frac{dy_1}{dt} &= -k_1y_1 + k_2y_2y_3, \\ \frac{dy_2}{dt} &= k_3y_1 - k_4y_2y_3 - k_5y_2^2, \\ \frac{dy_3}{dt} &= k_6y_2^2, \end{aligned}$$

where the values of the reaction rate constants are considered as follows, $k_1 = 0.04$, $k_2 = 0.01$, $k_3 = 400$, $k_4 = 100$, $k_5 = 3000$, $k_6 = 30$. Moreover, take the initial conditions $y_1(0) = 1$, $y_2(0) = 0$, and $y_3(0) = 0$.

In addition to equations (5.4) and (5.5) we define the nonlinear operator $Ny = y_2^2 = \sum_{n=0}^{\infty} A_n(t)$, where A_n is the convenient Adomian polynomial, given as

$$A_n(x_0, \dots, x_n; y_0, \dots, y_n) = \frac{1}{n!} \frac{d^n}{d\lambda^n} \left[F \left(\sum_{k=0}^n \lambda^k x_k, \sum_{k=0}^n \lambda^k y_k \right) \right]_{\lambda=0}, \quad n \geq 0.$$

This formulae is easy to set computer code to get as many polynomial as we need in the calculation of the numerical as well as explicit solutions. For example a Maple code to compute these polynomials can be found in [Bia06]. For $Ny = y_2^2$, we get $A_0 = y_{20}^2$, $A_1 = 2y_{20}y_{21}$, $A_2 = y_{21}^2 + 2y_{20}y_{22}, \dots$

The first terms of decomposition series solutions, by using the zeroth components (5.4) and the recursive relationship (5.5) will be

$$\begin{aligned} y_1 &= 1.0 - k_1 t + \frac{k_1^2}{2} t^2 - \frac{k_1^3}{6} t^3 + \dots, \\ y_2 &= k_3 t - \frac{k_1 k_3}{2} t^2 + \left(\frac{k_1^3 k_3 - k_3^3 k_5}{6} \right) t^3 + \dots, \\ y_3 &= \frac{k_3^2 k_6}{3} t^3 + \dots \end{aligned}$$

Another application for the decomposition method and comparison of it with Taylor series methods, power series methods and Runge–Kutta methods (using MATLAB) can be found in [Edw97], where a predator prey model is discussed.

The predator-prey equations have the following familiar form:

$$\begin{aligned} \frac{dN(t)}{dt} &= N(t) \left[r \left(1 - \frac{N(t)}{K} \right) - \frac{kP(t)}{N(t) + D} \right], \\ \frac{dP(t)}{dt} &= P(t) \left[s \left(1 - \frac{hP(t)}{N(t)} \right) \right], \end{aligned} \tag{5.6}$$

where r, K, k, D, s, h are positive constants; subject to the initial conditions

$$N(t_0) = N_0, \quad P(t_0) = P_0. \tag{5.7}$$

The direct application of ADM to the more complex model equations (5.6) subject to the parameter choice as in [Edw97], provides an approximate solution of reasonable performance. The authors observe that ADM solutions only converge locally to the true solution of (5.6) and are less accurate at similar floating point operational cost than a 4th order Runge–Kutta routine.

The Adomian's decomposition method can also be applied to a large class of system of partial differential equations with approximates that converges rapidly to accurate solutions. The implementation of the method has shown reliable results in that few terms are needed to obtain either exact solution or to find an approximate solution of a reasonable degree of accuracy in real physical models. Moreover, no linearization or

perturbation is required in the method. T. Alabdullatif et al. in [Ala07] have used this method to obtain an analytic approximate solution for nonlinear reaction–diffusion system of Lotka-Volterra type:

$$\begin{aligned} u_t &= (uu_x)_x + u(a_1 + b_1u) + h_1 + c_1v, \\ v_t &= (vv_x)_x + v(a_2 + b_2v) + h_2 + c_2u, \end{aligned} \quad (5.8)$$

$a_1, a_2, b_1, b_2, c_1, c_2, h_1, h_2$ are arbitrary constant such that $b_1b_2 \neq 0$ and $c_1c_2 \neq 0$, i.e. system (5.8) contains quadratic nonlinearity in reaction terms and the two equations are coupled.

In the case of PDEs we define the linear operator (and its inverse operator) in the following way:

$$L_t = \frac{\partial}{\partial t} \quad \text{and} \quad L_t^{-1} = \int_0^t (\cdot) dt. \quad (5.9)$$

Using (5.9), system (5.8) can be written as

$$\begin{aligned} L_t u &= (uu_x)_x + u(a_1 + b_1u) + h_1 + c_1v, \\ L_t v &= (vv_x)_x + v(a_2 + b_2v) + h_2 + c_2u. \end{aligned} \quad (5.10)$$

Applying the inverse operator to both sides of the above system, we get

$$\begin{aligned} u(x, t) &= f(x) + L_t^{-1}[a_1u + c_1v + h_1 + F(u)], \\ v(x, t) &= g(x) + L_t^{-1}[a_2v + c_2u + h_2 + G(v)], \end{aligned} \quad (5.11)$$

where

$$F(u) = (uu_x)_x + b_1u^2, \quad G(v) = (vv_x)_x + b_2v^2 \quad (5.12)$$

are the nonlinear terms in (5.10), $u(x, 0) = f(x)$ and $v(x, 0) = g(x)$.

According to the method we assume that a series solution of the unknown functions $u(x, t)$ and $v(x, t)$ are given by

$$u(x, t) = \sum_{n=0}^{\infty} u_n(x, t), \quad v(x, t) = \sum_{n=0}^{\infty} v_n(x, t). \quad (5.13)$$

The nonlinear terms $F(u)$ and $G(v)$ can be decomposed into the infinite series of polynomials given as

$$F(x, t) = \sum_{n=0}^{\infty} A_n, \quad G(x, t) = \sum_{n=0}^{\infty} B_n, \quad (5.14)$$

where A_n 's and B_n 's are the Adomian polynomials of u_n 's and v_n 's respectively. These polynomials for nonlinear terms can be calculated via the formula

$$A_n(u_0, u_1, \dots, u_n) = \frac{1}{n!} \left[\frac{d^n}{d\lambda^n} F \left(\sum_{k=0}^n \lambda^k u_k \right) \right]_{\lambda=0}, \quad n \geq 0, \quad (5.15)$$

$$B_n(v_0, v_1, \dots, v_n) = \frac{1}{n!} \left[\frac{d^n}{d\lambda^n} G \left(\sum_{k=0}^n \lambda^k v_k \right) \right]_{\lambda=0}, \quad n \geq 0. \quad (5.16)$$

For example,

$$\begin{aligned}
A_0 &= bu_0^2 + u_0u_{0xx} + (u^2)_x, \\
A_1 &= 2bu_0u_1 + u_1u_{0xx} + u_0u_{1xx} + 2u_{0x}u_{1x}, \\
A_2 &= \frac{1}{2} [b(2u_1^2 + 4u_0u_2) + 2u_2u_{0xx} + u_1u_{1xx} + 2u_0u_{2xx} + 2(u_{1x})^2 + 4u_{0x}u_{2x}] \\
&\quad \vdots
\end{aligned}$$

The components u_n and v_n for $n \geq 0$ are given by the following recursive relationships:

$$\begin{aligned}
u_0 &= u(x, 0) = f(x), \\
v_0 &= v(x, 0) = g(x), \\
u_1 &= L_t^{-1}[a_1u_0 + c_1v_0 + h_1 + A_0], \\
v_1 &= L_t^{-1}[a_2v_0 + c_2u_0 + h_2 + B_0], \\
&\quad \vdots \\
u_{n+1} &= L_t^{-1}[a_1u_n + c_1v_n + h_1 + A_n], \\
v_{n+1} &= L_t^{-1}[a_2v_n + c_2u_n + h_2 + B_n].
\end{aligned}$$

Using the above recursive relationships, we construct the solutions $u(x, t)$ and $v(x, t)$ as

$$u(x, t) = \lim_{n \rightarrow \infty} \psi_n(x, t), \quad v(x, t) = \lim_{n \rightarrow \infty} \varphi_n(x, t),$$

where

$$\psi_n(x, t) = \sum_{i=0}^{n-1} u_i(x, t), \quad \varphi_n(x, t) = \sum_{i=0}^{n-1} v_i(x, t), \quad n \geq 1.$$

This scheme was used in [Ala07] to obtain analytic approximate solution of the nonlinear reaction diffusion system of Lotka-Volterra type. The results obtained by the authors indicate that the method is efficient and accurate.

Beside these we would like to mention here the work of P. Diță and N. Grama, who have shown in [Dit06] that with a few modification the Adomian's method for solving second order differential equations can be used to obtain the known results of the special functions. Their results can also be seen as a good illustration for the effectiveness of the Picard method of successive approximation.

6 Stiff systems

6.1 Introduction

A stiff problem is a particular case of an initial value problem:

$$\begin{cases} y'(x) = f(x, y(x)), & a \leq x \leq b; \\ y(a) = y_a \end{cases}$$

In this case there are N equations and $y : [a, b] \rightarrow \mathbb{R}^N$, $f : [a, b] \times \mathbb{R}^N \rightarrow \mathbb{R}^N$.

The step by step numerical algorithm builds an approximate solution by an interactive procedure starting from $x = a$. Denote $y(x_n)$ the exact solution at the point x_n of the associated interval division

$$a = x_0 < \cdots < x_n < \cdots < x_M = b,$$

and y_n , the approximate value computed with the numerical method.

For the unicity of the exact solution we put the following condition:

- the system function f is continuous on

$$D = \{(x, y) | x \in [a, b], \|y(x) - y\| \leq d\},$$

where $d \in \mathbb{R}^+$;

- f satisfies the Lipschitz's inequality

$$\|f(x, u) - f(x, v)\| \leq L\|u - v\|, \quad a \leq x \leq b, \quad \forall(x, u), (x, v) \in D.$$

For almost all differential equation systems, the value $L(b-a)$ is of hundreds order. For such systems the classical step by step methods, like Runge-Kutta's processes or Adam's formulae are satisfactorily (in the sense of a small error).

Unfortunately, there are some exceptions, for instance, the case when the function variation is very strong. When the value $L(b-a)$ exceeds the thousands order, a variety of restrictions are imposed to the classical methods, especially on the step size, so there are useless. Such a system is classified to be of stiff kind one.

The essence of the stiff phenomenon consists in the fact that the exact solution includes some components with a very fast decrease that can be very hard to be followed by the numerical solution given by step by step iterative process.

We can give some pragmatic definitions, too:

- The stiff equations are the equations for which some implicit methods work better than the explicit ones.
- A problem is stiff in a given integration interval if, for a given numerical code, the step size must be very strongly reduced.
- The stiff differential equations are wrong conditioned in the computational sense.
- An ordinary differential equation system is stiff in a given integration interval $[0, T]$ if, in the exact solution, there is at least one component with a very large variation relative to the value $1/T$.
- An ODEs is defined stiff if the real parts of the Jacobian matrix eigenvalues are negative and vary over many orders of magnitude.

Definition 6.1. *An ordinary differential equation system with $y' = f(x, y)$, numerical integrated on the interval $[a, b]$, starting from the initial value $y(a) = y_a$, is stiff if the function f has a Lipschitz's constant very large relative to the interval length, and the error tolerance.*

The principal characteristics of the stiff problems are:

- the exact solutions are stable in the sense that small perturbations in the initial values are followed only by small perturbations in the exact solutions;
- trying to solve the problem with the standard methods we get some strict restrictions on the step size from the stability conditions.

6.2 Stiff systems in chemistry

It is well-known that for a numerical chemistry model the computational time needed is a function of both the number of species and the number of reactions. A particular problem arising in the solution of these ODEs is that of stiffness, which is related to the existence of widely differing timescales in the solution.

As it was stated earlier, a stiff system can be described mathematically as one in which all the eigenvalues of the Jacobian matrix of the system are negative, and the ratio of the absolute values of the largest to smallest real parts of the eigenvalues is much greater than one. For example, for atmospheric chemistry problems, the ratio is often greater than 1000, making the system very stiff.

The stiffness problem coupled with the fact that these equations must be solved for tens of thousands of cells in a typical modeling application require that special numerical methods be employed. The use of standard explicit methods is often precluded because relatively small time steps are required to maintain numerical stability and obtain accurate solutions. On the other hand, classical implicit methods that are both accurate and stable have not often been used because of high computational demands.

In their paper, see [Ver95], J. G. Verwer and D. Simpson discussed and compared three explicit methods for a selected chemical kinetics system, which lies in the study of air pollution: the first and the second method are of the explicit QSSA type and the third is based on the two-step backward differentiation formula, combined with Gauss-Seidel iteration to approximately solve the implicitly defined solution.

In [Cli96] L. J. Clifford et al. have investigated computationally faster methods of solving finite-rate chemistry than the chemical kinetics methods. One approach to this is to try to reduce a chemical kinetics data set to the lowest possible number of reactions and species using a detailed reduction strategy. The authors have shown that some stiff ODE solving methods are fastest in certain conditions and other methods are faster in other conditions. They also have derived an induction parameter model for reactions involving the oxidation of ethylene in an argon atmosphere.

The system of convection-diffusion equations with stiff source terms are also discussed by R. P. Fedkiw et al. in [Fed97] who developed the framework needed to apply modern high accuracy numerical methods from computational gas dynamics to this extended system. They among others also have shown how to treat the stiff reactions via time splitting (or fractional time step), in which the various operators (transport, diffusion, chemistry, etc.) are treated separately, and in particular how to increase accuracy by avoiding the common practice of approximating the temperature. The authors were based on general consideration that, since the stiff source terms require specialized and costly time integration, it is most practical to use a time splitting to isolate their treatment from the rest of the problem.

6.3 Handling the stiffness

To illustrate the stiffness phenomenon, consider the following set of equations:

$$\begin{aligned} \dot{x} &= 998x + 1998y, \\ \dot{y} &= -999x - 1999y, \end{aligned} \tag{6.1}$$

with boundary conditions

$$x(0) = 1, \quad y(0) = 0. \tag{6.2}$$

Using the transformation

$$x = 2u - v, \quad y = -u + v$$

which is equivalent with

$$u = x + y, \quad v = x + 2y,$$

we find the new system

$$\begin{aligned} \dot{u} &= -u, \\ \dot{v} &= -1000u. \end{aligned} \tag{6.3}$$

Take into account the initial condition $u(0) = 1, v(0) = 1$, we can solve the system (6.3) easily, and get

$$\begin{aligned} u(t) &= e^{-t}, \\ v(t) &= 1000e^{-t}. \end{aligned}$$

Finally, we find the solution of (6.1)

$$\begin{aligned} x(t) &= 2e^{-t} - e^{-1000t}, \\ y(t) &= -e^{-t} + e^{-1000t}. \end{aligned} \tag{6.4}$$

If we integrated the system (6.1) with any of the methods given so far, the presence of the e^{-1000t} term would require a step size $h \ll 1/1000$ for the method to be stable. This is so even though the e^{-1000t} term is completely negligible in determining the values of x and y as soon as one is away from the origin (see fig. 1).

This is the generic disease of stiff equations: we are required to follow the variation in the solution on the shortest length scale to maintain stability of the integration, even though accuracy requirements allow a much larger step size.

Implicit integration methods are the cure of stiffness. For example, having the system $\mathbf{y}' = \mathbf{f}(\mathbf{y})$, implicit differencing gives

$$y_{n+1} = y_n + h\mathbf{f}(y_{n+1}). \tag{6.5}$$

In general this is some nasty set of nonlinear equations that has to be solved iteratively at each step. Suppose we try linearizing the equations, as in Newton's method:

$$y_{n+1} = y_n + h \left[\mathbf{f}(y_n) + \left. \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right|_{y_n} \cdot (y_{n+1} - y_n) \right]. \tag{6.6}$$

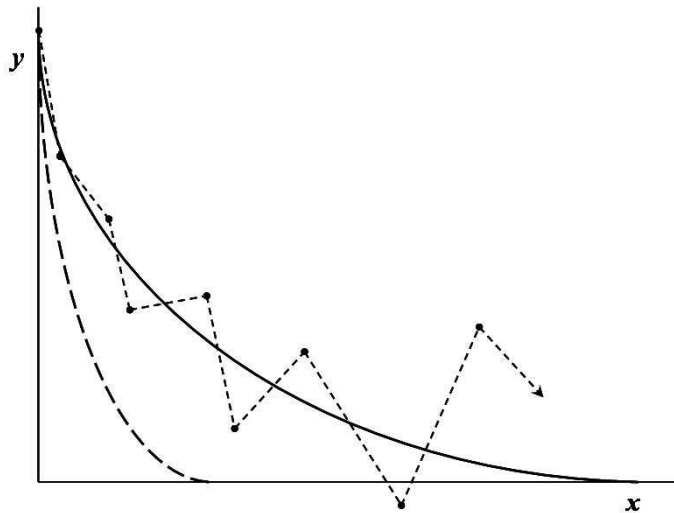


Figure 1: Example of an instability encountered in integrating a stiff equation. Here it is supposed that the equation has two solutions, shown as solid and dashed lines. Although the initial conditions are such as to give the solid solution, the stability of the integration (shown as the unstable dotted sequence of segments) is determined by the more rapidly varying dashed solution, even after that solution has effectively died away to zero.

Here $\partial \mathbf{f} / \partial \mathbf{y}$ is the matrix of the partial derivatives of the right-hand side (the Jacobian matrix). Rearranging the previous equation we obtain

$$\left[1 - h \left. \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right|_{y_n} \right] \cdot y_{n+1} = y_n + h \left[\mathbf{f}(y_n) - \left. \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right|_{y_n} \cdot y_n \right],$$

then

$$\left[1 - h \left. \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right|_{y_n} \right] \cdot y_{n+1} = \left[1 - h \left. \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right|_{y_n} \right] \cdot y_n + h \mathbf{f}(y_n),$$

and finally

$$y_{n+1} = y_n + h \left[1 - h \left. \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right|_{y_n} \right]^{-1} \cdot \mathbf{f}(y_n). \quad (6.7)$$

Solving implicit methods by linearization is called a "semi-implicit" method, so equation (6.7) is the *semi-implicit Euler method*.

So far we have dealt only with implicit methods. While these are very robust, most problems will benefit from higher-order methods. There are three important classes of higher-order methods for stiff systems:

- Generalizations of the Runge-Kutta method, of which the most useful are the Rosenbrock methods. The first practical implementation of these ideas was by Kaps and Rentrop, and so these methods are also called Kaps-Rentrop methods.
- Generalizations of the Bulirsch-Stoer method, in particular a semi-implicit extrapolation method due to Bader and Deuffhard.

- Predictor-corrector methods, most of which are descendants of Gears backward differentiation method.

6.4 Rosenbrock methods

Now consider the ODE system in the autonomous form

$$\dot{y} = f(y), \quad t > t_0, \quad y(t_0) = y_0. \quad (6.8)$$

This places no restriction since every non-autonomous system can be put in the form (6.8) by treating time t also as a dependent variable.

Rosenbrock methods have the advantage of being relatively simple to understand and implement. For moderate accuracies and moderate-sized systems, they are competitive with the more complicated algorithms. For more stringent parameters, these methods remain reliable; they merely become less efficient than competitors like the semi-implicit extrapolation method.

In 1963 Rosenbrock proposed to generalize the linearly implicit approach to methods using more stages, so as to achieve a higher order of consistency. The crucial consideration put forth was to no longer use the iterative Newton method, but instead to derive stable formulas by working the Jacobian matrix directly into the integration formula. His idea has found widespread use and a generally accepted formula (Hairer and Wanner, 1991) for a so-called s -stage Rosenbrock method, is

$$\begin{aligned} y_{n+1} &= y_n + \sum_{i=1}^s b_i k_i \\ k_i &= hf \left(y_n + \sum_{j=1}^{i-1} \alpha_{ij} k_j \right) + h\mathbf{J} \sum_{j=1}^i \gamma_{ij} k_j, \end{aligned} \quad (6.9)$$

where s and the formula coefficients b_i , α_{ij} and γ_{ij} are chosen to obtain a desired order of consistency and stability for stiff problems. We can take the coefficients γ_{ii} equal for all stages, i.e. $\gamma_{ii} = \gamma$ for all $i = 1, 2, \dots, s$. For $s = 1, \gamma = 1$ the linearized implicit Euler formula is recovered. For the non-autonomous system $\dot{y} = f(t, y)$, the definition of k_i is changed to

$$k_i = hf \left(t_n + \alpha_i h, y_n + \sum_{j=1}^{i-1} \alpha_{ij} k_j \right) + \gamma_i h^2 \frac{\partial f}{\partial t}(t_n, y_n) + h\mathbf{J} \sum_{j=1}^i \gamma_{ij} k_j$$

where

$$\alpha_i = \sum_{j=1}^{i-1} \alpha_{ij}, \quad \gamma_i = \sum_{j=1}^i \gamma_{ij}.$$

Like Runge–Kutta methods, Rosenbrock methods successively form intermediate results

$$Y_i = y_n + \sum_{j=1}^{i-1} \alpha_{ij} k_j, \quad 1 \leq i \leq s, \quad (6.10)$$

which approximate the solution at the intermediate time points $t_n + \alpha_i h$. Rosenbrock methods are therefore also called Runge–Kutta–Rosenbrock methods. Observe that if we identify \mathbf{J} with the zero matrix and omit the $\partial f/\partial t$ term, a classical explicit Runge–Kutta method results.

Rosenbrock methods are attractive for a number of reasons. Like fully implicit methods, they preserve exact conservation properties due to the use of the analytic Jacobian matrix. However, they do not require an iteration procedure as for truly implicit methods and are therefore more easy to implement. They can be developed to possess optimal linear stability properties for stiff problems. They are of one-step type, and thus can rapidly change step size.

For an application of a second order Rosenbrock method applied to photochemical dispersion problems see [Ver97].

In [Cab] one can find an investigation on a chemical mechanism which demonstrates stiffness. The authors have compared various explicit and implicit numerical methods to solve the model for the GABA reaction scheme, which is based on the classical bimolecular reaction of Michaelis–Menten, suitably adapted to account for reversible reactions and multiple receptor states.

In [Vos01] parallel Rosenbrock methods are developed for systems with stiff chemical reactions.

The authors first have considered the Robertson chemical kinetics problem:

$$\begin{aligned} \dot{y}_1 &= -0.04y_1 + 10^4 y_2 y_3; \\ \dot{y}_2 &= 0.04y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2; \\ \dot{y}_3 &= 3 \times 10^7 y_2^2 \end{aligned} \tag{6.11}$$

with initial conditions

$$y_1(0) = 1, \quad y_2(0) = 0, \quad y_3(0) = 0.$$

Next they have considered the enzyme kinetics problem of Michaelis–Menten type:

$$\begin{aligned} \frac{\partial u}{\partial t} &= d \frac{\partial^2 u}{\partial x^2} - \frac{u}{1+u}, \quad d > 0, \quad (x, t) \in [0, 1] \times [0, \infty), \\ u(x, 0) &= 1, \quad u(0, t) = 0, \quad u(1, t) = 0, \end{aligned} \tag{6.12}$$

whose steady state solution is $u = 0$.

Unlike classical Runge–Kutta methods, parallel Rosenbrock methods avoid the necessity to iterate at each time step. Parallelism across the method allows for the solution of the linear algebraic systems in essentially backward Euler–like solves on concurrent processors. In addition to possessing excellent stability properties, these methods are computationally efficient while preserving positivity of the solutions. Numerical results applied to problems above, involving stiff chemistry, and enzyme kinetics confirm these characteristics.

Finally, we mention here that for an application on second order Rosenbrock method

one can study the paper of Verwer et al., see [Ver97], where this scheme is defined as:

$$\begin{aligned}
y_{n+1} &= y_n + \frac{3}{2\gamma}k_1 + \frac{1}{2\gamma}k_2, \\
\left(\frac{1}{\gamma h}\mathbf{I} - \mathbf{J}\right)k_1 &= f(t_n, y_n) + \gamma h f_t, \\
\left(\frac{1}{\gamma h}\mathbf{I} - \mathbf{J}\right)k_2 &= f(t_n + h, y_n + \frac{1}{\gamma}k_1) - \frac{2}{\gamma h}k_1 - \gamma h f_t,
\end{aligned} \tag{6.13}$$

with $\gamma = 1 + 1/\sqrt{2}$. In [Ver97] it was noted that the second order Rosenbrock method have favorable positivity properties, and the method is stable for nonlinear problems even with large fixed step sizes.

6.5 Semi-implicit extrapolation method

The Bulirsch–Stoer method, which discretizes the differential equation using the modified midpoint rule, does not work for stiff problems. Bader and Deuffhard discovered a semi-implicit discretization that works very well and that lends itself to extrapolation exactly as in the original Bulirsch–Stoer method. The starting point is an implicit form of the midpoint rule:

$$y_{n+1} - y_{n-1} = 2h\mathbf{f}\left(\frac{y_{n+1} + y_{n-1}}{2}\right) \tag{6.14}$$

Convert this equation into semi-implicit form by linearizing the right-hand side about $\mathbf{f}(y_n)$. The result is the *semi-implicit midpoint rule*:

$$\left[1 - h\frac{\partial\mathbf{f}}{\partial\mathbf{y}}\right] \cdot y_{n+1} = \left[1 + h\frac{\partial\mathbf{f}}{\partial\mathbf{y}}\right] \cdot y_{n-1} + 2h\left[\mathbf{f}(y_n) - \frac{\partial\mathbf{f}}{\partial\mathbf{y}} \cdot y_n\right]. \tag{6.15}$$

It is used with a special first step, the semi-implicit Euler step (6.7), and a special "smoothing" last step in which the last y_n is replaced by

$$\bar{y}_n := \frac{1}{2}(y_{n+1} + y_{n-1}). \tag{6.16}$$

Bader and Deuffhard showed that the error series for this method once again involves only even powers of h .

For practical implementation, it is better to rewrite the equations using $\Delta_k = y_{k+1} - y_k$. With $h = H/m$, start by calculating

$$\begin{aligned}
\Delta_0 &= \left[\mathbf{1} - h\frac{\partial\mathbf{f}}{\partial\mathbf{y}}\right]^{-1} \cdot h\mathbf{f}(y_0) \\
y_1 &= y_0 + \Delta_0.
\end{aligned} \tag{6.17}$$

Then for $k = 1, 2, \dots, m - 1$, set

$$\begin{aligned}
\Delta_k &= \Delta_{k-1} + 2\left[\mathbf{1} - h\frac{\partial\mathbf{f}}{\partial\mathbf{y}}\right]^{-1} \cdot [h\mathbf{f}(y_k) - \Delta_{k-1}], \\
y_{k+1} &= y_k + \Delta_k.
\end{aligned} \tag{6.18}$$

Finally compute

$$\begin{aligned}\Delta_m &= \left[\mathbf{1} - h \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right]^{-1} \cdot [h\mathbf{f}(y_m) - \Delta_{m-1}], \\ \bar{y}_m &= y_m + \Delta_m.\end{aligned}\tag{6.19}$$

6.6 Gear algorithm

Consider the ODE system

$$\dot{y} = f(y, t), \quad t > t_0, \quad y(t_0) = y_0.\tag{6.20}$$

arising from chemistry.

In what follows we will present here the Gear algorithm in reference to the above system.

The Gear algorithm is one of a class of methods referred to as backward differentiation formulae (BDF). The generalized BDF that forms the basis for Gear's method can be expressed as follows:

$$y_n = h\beta_0 f(y_n, t_n) + \sum_{j=1}^p \alpha_j y_{n-j}\tag{6.21}$$

where n refers to the time step, h is the size of the time step, p is the assumed order, β_0 and α_j are scalar quantities that are functions of the order, and $f(y, t)$ is the function which describe the rate of change of each species concentration in a chemical mechanism. The method is implicit since concentrations at the desired time step n depend on values of the first derivatives contained in $f(y_n, t_n)$ that are functions of the concentrations at the same time. The order of the method corresponds to the number of concentrations at previous time steps that are incorporated in the summation on the right hand side of equation (6.21).

To facilitate changing step size and estimating errors, the multi-step method in equation (6.21) is transformed to a multi-value form in which information from only the previous step is retained, but information on higher order derivatives is now used. In this formulation, the solution is first approximated by predicting concentrations and higher order derivatives at the end of a time step for each species using the following matrix equation:

$$\mathbf{z}_{i,n,(0)} = \mathbf{B}\mathbf{z}_{i,n-1}\tag{6.22}$$

where $\mathbf{z}_i = [y_i, hy'_i, \dots, h^p y_i^{(p)}/p!]^\top$, the subscript $n, (0)$ refers to the prediction at the end of time step n , and the subscript $n-1$ refers to values obtained at the end of previous time step (or the initial conditions when $n = 1$). \mathbf{B} is the Pascal triangle matrix, the columns of which contain the binomial coefficients.

The prediction obtained from equation (6.22) is then corrected by solving for $\mathbf{z}_{i,n}$ such that the following relations hold for all species:

$$\mathbf{z}_{i,n} = \mathbf{z}_{i,n,(0)} + \mathbf{r}[hf_i(y_n, t_n) - hy'_{i,n,(0)}].\tag{6.23}$$

In equation (6.23), \mathbf{r} is a vector of coefficients that is dependent on the order, but r_2 , the element corresponding to the first derivative location in \mathbf{z} , is always equal to one.

Thus, the correct value of $y_{i,n}$ is obtained when the calculated value of $y'_{i,n}$ equals $f_i(y_n, t_n)$ in equation (6.23). An approximate solution for $y_{i,n}$ is obtained by applying Newton's method to the system of equations that correspond to the first equation in (6.23) for all species. This leads to the following corrector iteration equation:

$$y_{n,(m+1)} = y_{n,(m)} + [\mathbf{I} - h\beta_0\mathbf{J}]^{-1}r_1[hf(y_{n,(m)}, t_n) - h\delta y_{n,(m)}] \quad (6.24)$$

where m refers to the Newton iteration number, the vector $f(y_n, t_n)$ is calculated using concentrations computed for the m th iteration, δc is the vector containing the most recent estimates of first derivatives, \mathbf{I} is the identity matrix, and \mathbf{J} is the Jacobian matrix whose entries are defined as:

$$J_{ij} = \frac{\partial f_i(\mathbf{y}, t_n)}{\partial y_j}; \quad i, j = 1, 2, \dots, N. \quad (6.25)$$

At the end of each iteration, the vector containing the first derivatives (δc) is updated, but higher order derivatives in \mathbf{z} need not be computed until convergence is achieved.

Although several variants of the basic Gear algorithm have been developed, the fundamental computational scheme can be described generically as follows. At the beginning of any integration interval, the order is set to one and the starting time step is either calculated or selected by the user. Each time step is initiated by predicting concentrations at the end of the time step using equation (6.22). Corrector iterations are then carried out using equation (6.24) until prescribed convergence criteria are achieved or non-convergence is deemed to have occurred.

We have to mention here that Jacobson and Turco (1994) have modified the Gear algorithm to incorporate additional computational efficiencies that can achieve speedups on the order of 100 on vector computers. About half of the improvement is attributed to enhanced vectorization, and half to improved matrix operations.

6.7 Multistep, multivalued, and predictor-corrector methods

The terms multistep and multivalued describe two different ways of implementing essentially the same integration technique for ODEs. Predictor-corrector is a particular subcategory of these methods—in fact, the most widely used. Accordingly, the name predictor-corrector is often loosely used to denote all these methods.

To advance the solution of $y' = f(x, y)$ from x_n to x , we have

$$y(x) = y_n + \int_{x_n}^x f(x', y) dx'. \quad (6.26)$$

In a multistep method, we approximate $f(x, y)$ by a polynomial passing through several previous points x_n, x_{n-1}, \dots and possibly also through x_{n+1} . The result of evaluating the integral (6.26) at $x = x_{n+1}$ is then of the form

$$y_{n+1} = y_n + h(\beta_0 y'_{n+1} + \beta_1 y'_n + \beta_2 y'_{n-1} + \beta_3 y'_{n-2} + \dots) \quad (6.27)$$

where y'_n denotes $f(x_n, y_n)$, and so on. If $\beta_0 = 0$, the method is explicit; otherwise it is implicit. The order of the method depends on how many previous steps we use to get each new value of y .

Consider how we might solve an implicit formula of the form (6.27) for y_{n+1} . Two methods suggest themselves: functional iteration and Newtons method. In functional iteration, we take some initial guess for y_{n+1} , insert it into the right-hand side of (6.27) to get an updated value of y_{n+1} , insert this updated value back into the right-hand side, and continue iterating. To get an initial guess for y_{n+1} one must use some explicit formula of the same form as (6.27) This is called the predictor step. In the predictor step we are essentially extrapolating the polynomial fit to the derivative from the previous points to the new point x_{n+1} and then doing the integral (6.26) from x_n to x_{n+1} . The subsequent Simpson-like integration, using the prediction steps value of y_{n+1} to interpolate the derivative, is called the corrector step. The difference between the predicted and corrected function values supplies information on the local truncation error that can be used to control accuracy and to adjust step size.

The most popular predictor-corrector methods are probably the Adams- Bashforth-Moulton schemes, which have good stability properties. The Adams- Bashforth part is the predictor.

The Adams-Bashforth formula has the form

$$x_{i+1} = x_i + \sum_{j=1}^n c_j f_j, \text{ where } f_j = f(t_{i-(j-1)}, x(t_{i-(j-1)})).$$

Note the formula uses the function values at points left to t_i , namely, at $t_i, t_{i-1}, \dots, t_{i-(n-1)}$.

To determine the coefficients in the Adams-Bashforth formula one have to solve a linear system for the c_j 's,

$$\int_0^1 r^{(i-1)} dr = \sum_{j=1}^n (c_j (1-j)^{(i-1)})$$

where $1, r, r^2, \dots, r^{(n-1)}$ are the testing polynomials.

For example the the 2nd-order Adams-Bashforth method becomes as follow:

$$x_0 = x_0, x_1 = x_1, x_{i+1} = x_i + \frac{h(3f(t_i, x_i) - f(t_{i-1}, x_{i-1}))}{2}$$

for $i > 0$.

In practice, Adams-Bashforth formula is not used alone. Neither does Adams-Moulton formula. Instead, the Adams-Moulton formula, which is implicit, is modified by approximating x_{i+1} using the solution we get from the Adams-Bashforth method. This avoids solving a nonlinear equation for x_{i+1} in the original Adams-Moulton formula, and improves accuracy of the Adams-Bashforth formula.

For instance, the 2nd-order Adams-Bashforth formula and the 2nd-order Adams-Moulton formula lead us to a second order predictor-corrector method:

$$\begin{aligned} x_0 &= x_0, x_1 = x_1, \\ y_{i+1} &= x_i + \frac{h(3f(t_i, x_i) - f(t_{i-1}, x_{i-1}))}{2}, \\ x_{i+1} &= x_i + \frac{h(f(t_{i+1}, y_{i+1}) + f(t_i, x_i))}{2}. \end{aligned} \tag{6.28}$$

7 Positive and conservative numerical methods

There exist specialized softwares that translate kinetic reactions into differential equations (e.g., KPP [Dam02]). As we mentioned earlier, the ODEs describing chemical kinetics can be nonlinear and very stiff. The stiffness in these systems forces the use of implicit numerical techniques for their solution. The drawback to the use of an implicit numerical technique is the need to solve a large nonlinear algebraic system of equations for each numerical time step at each spatial grid point. The greater the number of chemical species modeled, the larger the resulting system. Thus, the numerical overhead associated with solving these nonlinear algebraic systems can account for a significant portion of the total runtime, for example, in atmospheric sciences (reactive flow problem, combustion, climate modeling, chemical-radiative-transport model).

Several recent studies have proposed techniques that effectively eliminate the need for the matrix algebra typically associated with an implicit numerical method, for instance, the preconditioned implicit methods. That study led to the development of the Chemical Solver for Ordinary Differential Equations (CHEMSODE) package to support the use of these methods. The paradigm of the preconditioned implicit methods combine an implicit ODE integration formula with an iterative technique for solving simultaneous algebraic systems. This idea is illustrated in [Aro96] and have been drawn from two previous studies. The above report documents the CHEMSODE package: a collection of FORTRAN subroutines implementing three different types of preconditioned time differencing techniques along with a choice of step length control.

A comparison between two classical integrators for stiff ordinary differential equations, Quasi Steady State Approximation (QSSA), and Hybrid Solver (HS), and CHEMSODE package is presented in [Lor99]. The results have been compared with an "exact" solution obtained by the Livermore Solver for Ordinary Differential Equations (LSODE).

A comprehensive numerical comparison between some(other) explicit and implicit solvers can also be found in [San97]. For another comparison of numerical methods for the integration of kinetic equations in atmospheric chemistry and transport models, see [Say95].

When fast reversible reactions precede slower ones in a mechanism, in order to derive approximate analytical expressions, beside the QSSA classical method we can use another useful approximation method, the pre-equilibrium approximation (PEA), also called equilibrium approximation. Both approximations can also be used to simplify the numerical integration of complex reaction schemes, reducing the dimensionality and decoupling time scales. In their work, see [Rae02], M. Rae and N. Berberan-Santos were presented a general view of the pre-equilibrium approximation via several photophysical systems. They also have shown that the kinetic behavior of systems subject to pre-equilibration can be obtained by the application of perturbation theory.

Air quality models solve the convection-diffusion reaction set of partial differential equations which describe the atmospheric physical and chemical processes. Usually an operator-split approach is taken: chemical equations and convectiondiffusion equations are solved in alternative steps. In this setting the integration of chemical kinetic equations is a demanding computational task. The chemical integration algorithm should be stable in the presence of stiffness; ensure a modest level of accuracy, typically 1%; preserve mass; and keep the concentrations positive. Since chemical kinetics conserves mass and renders

nonnegative solutions a good numerical simulation would ideally produce mass-balanced, positive concentration vectors. Many time-stepping methods are mass conservative (multistep, Runge-Kutta, Rosenbrock); however, unconditional positivity restricts the order of a traditional method to one. Clipping (setting the negative concentrations to zero) enhances stability but artificially adds mass to the system. In [San97] was presented a projection method which ensure mass conservation and positivity, alleviating the order and step-size restrictions. The solutions computed at each step by a standard integration method are "projected" back onto the reaction simplex. The resulting vectors better approximate the true solution itself.

Another approach to accelerate reacting flow calculations was proposed by O. Knöth and R. Wolke in [Kno98]. The authors gave implicit-explicit time integration schemes which use explicit higher order Runge-Kutta schemes for the integration of the horizontal advection. The stiff chemistry and all vertical transport processes (turbulent diffusion, advection, deposition) are integrated in an implicit and coupled manner by a higher order backward differentiation formula method.

7.1 Stiff systems related to mass action kinetics

Numerical schemes that maintain numerical analogous of physical properties such as nonnegativity of the solution and atomic mass conservation without time step restrictions are also described in [Ber96]. Classical explicit schemes maintain these properties with too strong time-step restrictions to be useful. Classical implicit schemes maintain these properties with weaker time-step restrictions, but require the solution of an algebraic system at each time step. In the above mentioned paper, E. Bertolazzi proposed and discussed the solution of these algebraic systems without the use of Jacobian matrices, but by the repeated inversion of M-matrices that can be easily constructed, thus considerably simplifying and accelerating the computer implementation of the schemes.

To illustrate the idea consider a reaction mechanism

$$\sum_{i=1}^m \alpha(i, r) \mathcal{X}_i \rightleftharpoons \sum_{i=1}^m \beta(i, r) \mathcal{X}_i \quad r = 1, 2, \dots, k$$

which can be written in the form

$$\sum_{i=1}^m \sigma(i, r) \mathcal{X}_i = 0, \quad r = 1, 2, \dots, k. \quad (7.1)$$

The associated system of polynomial differential equation describing the time evolution of the species concentrations can be written as

$$\frac{d\mathbf{x}}{dt} = \sigma \mathbf{w}(\mathbf{x}), \quad (7.2)$$

where $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$, $\sigma = \begin{bmatrix} \sigma(1, 1) & \sigma(1, 2) & \dots & \sigma(1, k) \\ \sigma(2, 1) & \sigma(2, 2) & \dots & \sigma(2, k) \\ \vdots & \vdots & \dots & \vdots \\ \sigma(m, 1) & \sigma(m, 2) & \dots & \sigma(m, k) \end{bmatrix}$, $\mathbf{w}(\mathbf{x}) = \begin{bmatrix} w_1(\mathbf{x}) \\ w_2(\mathbf{x}) \\ \vdots \\ w_k(\mathbf{x}) \end{bmatrix}$ are the

molar concentration vector, the stoichiometric matrix and the reaction velocity vector, respectively.

If \mathcal{M}_i is the atomic mass of the i th species, then $\rho_i = \mathcal{M}_i x_i$ is its density. Defining

$$\begin{aligned}\mathbf{D} &= \text{diag}(\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_m), \\ \rho &= \mathbf{D}\mathbf{x}, \\ \mathbf{f}(\rho) &= \mathbf{w}(\mathbf{D}^{-1}\rho), \\ \mathcal{V} &= \mathbf{D}\sigma,\end{aligned}$$

chemical system (7.1) can be written in term of densities as follows:

$$\frac{d\rho}{dt} = \mathcal{V}\mathbf{f}(\rho). \quad (7.3)$$

We have to mention here that in mass action kinetic reaction rates often have the form $f(\rho) = c\rho_1^{\alpha_1}\rho_2^{\alpha_2}\dots\rho_m^{\alpha_m}$, where c is a constant and $\alpha_i \geq 0$.

Related to equation (7.3), it is important that, there exist two general requirements (see [Ber96]):

- one for the reaction rate f with its stoichiometric vector \mathcal{V}
 - (i) $f \in C(\overline{\mathbb{R}}_+^m, \overline{\mathbb{R}}_+)$,
 - (ii) if $\nu_i < 0$, then the function $q(\rho) := f(\rho)/\rho_i$ is such that $q \in C(\overline{\mathbb{R}}_+^m, \overline{\mathbb{R}}_+)$
- the other is connected with the nonnegativity of the solution:

$$\text{if } \rho_0 \geq 0 \implies \forall t > 0, \rho(t; \rho_0) \geq 0 \quad (7.4)$$

where $\rho(t; \rho_0)$ denotes the solution of system (7.3) with ρ_0 as initial value.

Now, consider the kernel \mathcal{V}^\top , i.e., $\text{Ker}(\mathcal{V}^\top) = \{\mathbf{z} | \mathbf{z}\mathcal{V}^\top = 0\}$. If $\mathbf{z} \in \text{Ker}(\mathcal{V}^\top)$ then from (7.3), it follows that $\mathbf{z}^\top \frac{d\rho}{dt} = \mathbf{z}^\top \mathcal{V}\mathbf{f}(\rho) = 0$, so that the function

$$g_{\mathbf{z}}(\rho) = \mathbf{z}^\top \rho = z_1\rho_1 + \dots + z_m\rho_m, \quad \mathbf{z} \in \text{Ker}(\mathcal{V}^\top) \quad (7.5)$$

is a first integral of (7.3). In the case of a conservative system, we have $\|\rho\|_1 = \rho_1 + \rho_2 + \dots + \rho_m = \text{constant}$ so that $g_{\mathbf{e}}$ is a first integral of the system (7.3), or equivalently, $\mathbf{e} \in \text{Ker}(\mathcal{V}^\top)$, where $\mathbf{e} = [1, 1, \dots, 1]^\top$.

In what follows we will take a survey through some classical numerical schemes to establish whether mass conservation and positivity preservation is realized.

A general numerical one-step scheme can be written as

$$\rho^{n+1} = \mathbf{G}(\rho^n), \quad n = 1, 2, \dots \quad (7.6)$$

For this scheme the conservation and nonnegativity properties (7.5) can be written as

$$\mathbf{G}(\rho) \geq 0, \quad \forall \rho \geq 0, \quad (7.7a)$$

$$\mathbf{z}^\top(\rho - \mathbf{G}(\rho)) = 0, \quad \forall \mathbf{z} \in \text{Ker}(\mathcal{V}^\top), \forall \rho \geq 0. \quad (7.7b)$$

We will consider a single reaction system

$$\frac{d\rho}{dt} = \mathcal{V}f(\rho), \quad (7.8)$$

where the reaction rate f is an homogeneous function of degree 1, for example $f(\rho) = c\rho_1^{\alpha_1}\rho_2^{\alpha_2}\dots\rho_m^{\alpha_m}$, with $\alpha_i = \begin{cases} 1, & \text{if } \nu_i < 0, \\ 0, & \text{if } \nu_i \geq 0, \end{cases}$ and discretize (7.8) with some classical schemes.

7.1.1 Explicit Euler scheme

This scheme can be written as

$$\rho^{n+1} = \rho^n + h\mathcal{V}f(\rho^n),$$

for which the map \mathbf{G} becomes

$$\mathbf{G}(\rho) = \rho + h\mathcal{V}f(\rho)$$

and conditions (7.7a), resp. (7.7b) become

- for nonnegativity it is necessary and sufficient that

$$h \leq \frac{\rho_i^n}{-\nu_i f(\rho^n)}, \quad \text{if } \nu_i < 0,$$

- the conservation is always satisfied, in fact

$$\mathbf{z}^\top(\rho - \mathbf{G}(\rho)) = -h\mathbf{z}^\top\mathcal{V}f(\rho^n) = 0, \quad \forall \mathbf{z} \in \text{Ker}(\mathcal{V}^\top).$$

7.1.2 Linearized implicit Euler scheme

Consider the implicit Euler scheme

$$\rho^{n+1} = \rho^n + h\mathcal{V}f(\rho^{n+1}),$$

and use the Newton–Raphson step to approximate ρ^{n+1}

$$\begin{aligned} \rho^{n+1} &\approx \overline{\rho^{n+1}} = \rho^n - [I - h\mathcal{V}\nabla f(\rho^n)]^{-1}[\rho^n - h\mathcal{V}f(\rho^n) - \rho^n] \\ &= \rho^n [I - h\mathcal{V}\nabla f(\rho^n)]^{-1} [I - h\mathcal{V}\nabla f(\rho^n)] - \\ &\quad - [I - h\mathcal{V}\nabla f(\rho^n)]^{-1} [-h\mathcal{V}f(\rho^n)] \\ &= [I - h\mathcal{V}\nabla f(\rho^n)]^{-1} [\rho^n (I - h\mathcal{V}\nabla f(\rho^n)) + h\mathcal{V}f(\rho^n)] \\ &= [I - h\mathcal{V}\nabla f(\rho^n)]^{-1} [\rho^n + h\mathcal{V}(f(\rho^n) - \nabla f(\rho^n)\rho^n)] \\ &= [I - h\mathcal{V}\nabla f(\rho^n)]^{-1} \rho^n. \end{aligned} \tag{7.9}$$

In the last equality it was used the homogeneity of f . Using $\overline{\rho^{n+1}}$ instead of ρ^{n+1} , we obtain the linearized implicit Euler scheme, for which the solution step is

$$\rho^{n+1} = [I - h\mathcal{V}\nabla f(\rho^n)]^{-1} \rho^n,$$

and applying the Sherman–Morrison formula

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\top\mathbf{A}^{-1}}{I + \mathbf{v}^\top\mathbf{A}^{-1}\mathbf{u}}, \tag{7.10}$$

where $\mathbf{u}\mathbf{v}^\top$ denotes the dyadic product, then the map \mathbf{G} becomes

$$\mathbf{G}(\rho) = \rho + \frac{h\mathcal{V}\nabla f(\rho)\rho}{I - h\nabla f(\rho)\mathcal{V}} \tag{7.11}$$

and conditions (7.7a) and (7.7b) become

- for the nonnegativity it is necessary and sufficient that

$$h \leq \frac{\rho_i^n (I - h \nabla f(\rho^n) \mathcal{V})}{-\nu_i \nabla f(\rho^n) \rho^n}, \quad \text{if } \nu_i < 0,$$

- the conservation is always satisfied, in fact from (7.11), it follows that

$$\mathbf{z}^\top (\rho - \mathbf{G}(\rho)) = -h \frac{\mathbf{z}^\top \mathcal{V} \nabla f(\rho) \rho}{I - h \nabla f(\rho) \mathcal{V}} = 0, \quad \forall \mathbf{z} \in \text{Ker}(\mathcal{V}^\top).$$

7.1.3 Runge–Kutta’s fourth-order scheme

The Runge–Kutta method of order 4 for system (7.3) takes the form

$$\begin{aligned} \mathbf{k}_1 &= hf(\rho^n) \mathcal{V}, \\ \mathbf{k}_2 &= hf\left(\rho^n + \frac{\mathbf{k}_1}{2}\right) \mathcal{V}, \\ \mathbf{k}_3 &= hf\left(\rho^n + \frac{\mathbf{k}_2}{2}\right) \mathcal{V}, \\ \mathbf{k}_4 &= hf(\rho^n + \mathbf{k}_3) \mathcal{V}, \\ \rho^{n+1} &= \rho^n + \frac{\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4}{6}. \end{aligned}$$

- A necessary condition for the applicability is $\rho^n + \frac{\mathbf{k}_1}{2} \geq 0$, so that

$$h \leq \frac{2\rho_i^n}{-\nu_i f(\rho^n)}, \quad \text{if } \nu_i < 0.$$

- The conservation (7.7a) is always satisfied, in fact

$$\mathbf{z}^\top \mathbf{k}_1 = hf(\rho^n) \mathbf{z}^\top \mathcal{V} = 0, \quad \forall \mathbf{z} \in \text{Ker}(\mathcal{V}^\top),$$

and analogously for $\mathbf{k}_2, \mathbf{k}_3, \mathbf{k}_4$.

7.1.4 A semi-implicit scheme

All the schemes previously considered have discrete analogs of the first integrals (7.5), but they suffer a time-step limitation for nonnegativity. Here we will present a semi-implicit method to avoid these limitations. We will use the following results:

Lemma 7.1. *Let $\mathbf{v} \in \mathbb{R}^m$ be a vector such that $\sum_{i=1}^m v_i = 0$, and let $f : \overline{\mathbb{R}}_+^m \rightarrow \overline{\mathbb{R}}_+$ be a continuous function, such that, for those indices i for which $v_i < 0$, $f(\mathbf{x})/x_i$ is also continuous on $\overline{\mathbb{R}}_+$.*

Then $f(\mathbf{x})\mathbf{v}$ can be written as

$$\mathbf{C}(\mathbf{x}, \mathbf{v})\mathbf{x} = f(\mathbf{x})\mathbf{v},$$

where $\mathbf{C}(\mathbf{x}, \mathbf{v})$ is $m \times m$ matrix with continuous entries such that

$$\begin{aligned} C_{i,i}(\mathbf{x}, \mathbf{v}) &\leq 0, & i = 1, 2, \dots, m, \\ C_{i,j}(\mathbf{x}, \mathbf{v}) &\geq 0, & i \neq j, \\ \sum_{i=1}^m C_{i,j}(\mathbf{x}, \mathbf{v}) &= 0, & i = 1, 2, \dots, m. \end{aligned} \tag{7.12}$$

Theorem 7.1. Let \mathcal{V} be a $m \times k$ matrix and $\mathbf{f} : \overline{\mathbb{R}}_+^m \rightarrow \overline{\mathbb{R}}_+^k$ be a continuous map; then if for those indices i, j for which $v_{i,j} < 0$,

$$\frac{f_i(\mathbf{x})}{x_j},$$

is also continuous on $\overline{\mathbb{R}}_+^n$, and

$$\sum_{i=1}^m v_{i,j} = 0, \quad j = 1, 2, \dots, k,$$

then $\mathcal{V}\mathbf{f}(\rho)$ can be written in the form $\mathbf{C}(\rho, \mathcal{V})\rho$, where $\mathbf{C}(\rho, \mathcal{V})$ is a $m \times m$ matrix with continuous entries, such that

- i) $C_{i,i}(\rho, \mathcal{V}) \leq 0, \quad i = 1, 2, \dots, m,$
- ii) $C_{i,j}(\rho, \mathcal{V}) \geq 0, \quad i \neq j,$
- iii) $\sum_{i=1}^m C_{i,j}(\rho, \mathcal{V}) = 0, \quad i = 1, 2, \dots, m.$

For the proof of the lemma and the theorem above, the definition of matrix $\mathbf{C}(\mathbf{x}, \mathbf{v})$ see [Ber96].

With the matrix \mathbf{C} , it is possible to define a semi-implicit numerical scheme for (7.3) as follows:

$$\rho^{n+1} = \rho^n + h\mathbf{C}(\rho^n, \mathcal{V})\rho^{n+1},$$

which results in the following advancing step:

$$\rho^{n+1} = [I - h\mathbf{C}(\rho^n, \mathcal{V})]^{-1}\rho^n.$$

The matrix $I - h\mathbf{C}(\rho^n, \mathcal{V})$ is strictly diagonally dominant with elements positive on the diagonal and nonnegative elsewhere, consequently, it is a M -matrix. By definition of a M -matrix, it follows that $I - h\mathbf{C}(\rho^n, \mathcal{V})^{-1} \geq 0$, and therefore, $\rho^{n+1} \geq 0$ for arbitrarily large h . Unfortunately, this scheme has not a numerical analogs of first integral (7.5).

7.1.5 A fully implicit scheme

All the previous considered schemes cannot have both a discrete analogs of first integral (7.5) and nonnegativity preservation for arbitrarily large time step. The implicit Euler scheme

$$\rho^{n+1} = \rho^n + h\mathcal{V}\mathbf{f}(\rho^{n+1}) \quad (7.13)$$

can be used to avoid time-step restrictions, but the advancing steps involve the solution of a nonlinear system as follows:

$$\rho^{n+1} = \text{the solution of } \mathbf{x} - h\mathcal{V}\mathbf{f}(\mathbf{x}) - \rho^n = 0. \quad (7.14)$$

For system (7.14), there is the question of existence and uniqueness of the solution and an iterative procedure is needed to find the solution.

Existence of a solution: A nonnegative solution of the nonlinear system

$$\mathbf{x} - h\mathcal{V}\mathbf{f}(\mathbf{x}) - \rho^n = 0, \quad (7.15)$$

is also, by Theorem 7.1, a fixed point of the map

$$\Phi(\mathbf{x}) := (I - h\mathbf{C}(\mathbf{x}, \mathcal{V}))^{-1}\rho^n. \quad (7.16)$$

Theorem 7.2. *The map Φ admits at least one nonnegative fixed point (i.e., with nonnegative components). Moreover if \mathbf{x}^* is a nonnegative fixed point, then $\|\mathbf{x}^*\|_1 = \|\rho^n\|_1$.*

Proof. The map Φ has the property $\forall \rho \geq 0 \implies \Phi(\rho) \geq 0$, and from (7.16), we can write

$$\Phi(\mathbf{x}) - h\mathbf{C}(\mathbf{x}, \mathcal{V})\Phi(\mathbf{x}) = \rho^n, \quad (7.17)$$

multiplying (7.17) by \mathbf{e}^\top and using the fact $\mathbf{e}^\top \mathbf{C}(\mathbf{x}, \mathcal{V})$, it follows

$$\mathbf{e}^\top \Phi(\mathbf{x}) = \mathbf{e}^\top \rho^n \implies \|\Phi(\mathbf{x})\|_1 = \|\rho^n\|_1, \quad \forall \mathbf{x} \geq 0.$$

Consequently, the image $\Phi(\overline{\mathbf{R}}_+^n)$ is contained into the convex compact

$$\mathcal{K} = \{x \geq 0 \mid \|\mathbf{x}\|_1 = \|\rho^n\|_1\}, \quad (7.18)$$

so that, if \mathbf{x}^* is a fixed point, it must be contained in \mathcal{K} and $\|\mathbf{x}^*\|_1 = \|\rho^n\|_1$. Moreover, the map Φ can be viewed as a continuous map from \mathcal{K} into \mathcal{K} , and by the Brouwer fixed point theorem it follows that Φ has at least one fixed point. \square

Observe that the proof is independent of the magnitude of h , so that the nonlinear system (7.15) has a nonnegative solution no matter how large the time step is.

The question of uniqueness: In general, it is not possible to see if system (7.15) has a unique solution for arbitrarily large h . However, there are some special cases for which we have also uniqueness. It is the case, for instance, when the system consists of only one reaction and the reaction rate f satisfies

$$\nu_i \partial f \rho_i \leq 0, \quad i = 1, 2, \dots, m,$$

which exclude auto-inhibition in the reaction. In this case, if \mathbf{x}^* and \mathbf{y}^* are two solutions of (7.15), it follows

$$0 = \mathbf{x}^* - \mathbf{y}^* - h\mathcal{V}[f(\mathbf{x}^*) - f(\mathbf{y}^*)] = [I - h\mathcal{V}\nabla f(\xi)](\mathbf{x}^* - \mathbf{y}^*), \quad (7.19)$$

and using the Sherman-Morrison formula (7.10)

$$[I - h\mathcal{V}\nabla f(\xi)]^{-1} = I + \frac{h\mathcal{V}\nabla f(\xi)}{I - h\mathcal{V}\nabla f(\xi)}, \quad (7.20)$$

so that by (7.19) and (7.20), it follows that $\mathbf{x}^* = \mathbf{y}^*$. In the case of more than one reaction, it is possible to prove uniqueness for small h .

Theorem 7.3. *Let $\mathcal{V}\mathbf{f}(\rho)$ be a $m \times k$ matrix and $\mathbf{f} \in C^1(\overline{\mathbb{R}}^m_+, \overline{\mathbb{R}}_+)$, then if for those indices i, j for which $\nu_{i,j} < 0$,*

$$\frac{f_i(\mathbf{x})}{x_j} \in C^1(\overline{\mathbb{R}}^m_+, \overline{\mathbb{R}}_+),$$

then for all h satisfying

$$h < \frac{1}{m\|\rho^n\|_1 \max_{\mathbf{z} \in \mathcal{K}} \|\nabla \mathbf{C}_{i,j}(\mathbf{z}, \mathcal{V})\|_\infty},$$

the map $\Phi : \overline{\mathbb{R}}_+^m \rightarrow \overline{\mathbb{R}}_+^m$ defined in (7.16) is a contraction, where \mathcal{K} is given by (7.18).

Proof. Observe that $\forall \mathbf{z} \geq 0$

$$\mathbf{e}^\top [I - h\mathbf{C}(\mathbf{z}, \mathcal{V})] = \mathbf{e}^\top \implies \mathbf{e}^\top = \mathbf{e}^\top [I - h\mathbf{C}(\mathbf{z}, \mathcal{V})]^{-1},$$

so that it follows $\|[I - h\mathbf{C}(\mathbf{z}, \mathcal{V})]^{-1}\|_1 = 1$. Next

$$\begin{aligned} \Phi(\mathbf{x}) - \Phi(\mathbf{y}) &= [I - h\mathbf{C}(\mathbf{x}, \mathcal{V})]^{-1}\rho^n - [I - h\mathbf{C}(\mathbf{y}, \mathcal{V})]^{-1}\rho^n = \\ &= h[I - h\mathbf{C}(\mathbf{x}, \mathcal{V})]^{-1}[\mathbf{C}(\mathbf{x}, \mathcal{V}) - \mathbf{C}(\mathbf{y}, \mathcal{V})][I - h\mathbf{C}(\mathbf{y}, \mathcal{V})]^{-1}\rho^n, \end{aligned}$$

and taking the $\|\cdot\|_1$ norm on both sides

$$\|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\|_1 \leq h\|\mathbf{C}(\mathbf{x}, \mathcal{V}) - \mathbf{C}(\mathbf{y}, \mathcal{V})\|_1\|\rho\|_1 \leq hL\|\mathbf{x} - \mathbf{y}\|_1,$$

where

$$L = m\|\rho^n\|_1 \max_{\mathbf{z} \in \mathcal{K}} \|\nabla \mathbf{c}_{i,j}(\mathbf{z}, \mathcal{V})\|_\infty,$$

so that if $h < \frac{1}{L}$ the map Φ becomes a contraction. \square

Obviously, if the map Φ is a contraction, then it has a unique fixed point.

7.1.6 A second-order positive scheme

The implicit Euler scheme has no stability restriction, but it is only first-order accurate. A second-order scheme can be the following:

$$\rho^{n+1} = \rho^n + h\nu \frac{\mathbf{f}(\rho^{n+1}) + \mathbf{f}(\rho^n)}{2}.$$

The solution step, as in (7.13), involves the solution of a nonlinear system in the unknown \mathbf{x}

$$\mathbf{x} - h \frac{\mathbf{f}(\mathbf{x})}{2} = \rho^n + h \frac{\nu \mathbf{f}(\rho^n)}{2}.$$

In order to guarantee the existence of a nonnegative solution, it is sufficient that:

$$\rho^n + h \frac{\nu \mathbf{f}(\rho^n)}{2} \geq 0.$$

This condition introduces a time-step bound. To avoid time-step restriction, an alternative approach is to switch to second-order accuracy when reactions are slow and first-order accuracy when reactions are fast. The scheme becomes

$$\rho^{n+1} = \rho^n + h\nu[(1 - \alpha)\mathbf{f}(\rho^{n+1}) + \alpha\mathbf{f}(\rho^n)], \quad (7.21)$$

where α is such that

$$\max_{\alpha \in [0, 1/2]} [\rho^n + \alpha h \nu \mathbf{f}(\rho^n)] \geq 0.$$

If the reaction rates have very different orders of magnitudes, this scheme can lose too much accuracy for slow reactions. A better result can be obtained by using different α 's for each reaction as follows:

$$\rho^{n+1} = \rho^n + h \sum_{j=1}^k \nu_{j,j} [(1 - \alpha_j) f_j(\rho^{n+1}) + \alpha_j f_j(\rho^n)]. \quad (7.22)$$

With (7.22), the solution step involves the solution of the nonlinear system in the unknown \mathbf{x}

$$\mathbf{x} - h \sum_{j=1}^k (1 - \alpha_j) \nu_{j,j} f_j(\mathbf{x}) = \rho^n + h \sum_{j=1}^k \alpha_j \nu_{j,j} f_j(\rho^n).$$

For positivity preservation, it suffices that

$$\rho^n + \sum_{j=1}^k \alpha_j \nu_{j,j} f_j(\rho^n) \geq 0. \quad (7.23)$$

In order to satisfy condition (7.23), we set

$$\alpha_j = \min \left(\frac{1}{2}, \left\{ -\frac{\beta_j \rho_i^n}{h \nu_{i,j} f_j(\rho^n)} \mid \nu_{i,j} < 0, i = 1, 2, \dots, m \right\} \right),$$

where $\beta_j > 0$ are weighting parameters with $\sum_{j=1}^k \beta_j = 1$. The weighting factors are useful; if, for example, the first reaction is very slow, we can use very small β_1 and maintain $\alpha_1 = 1/2$. This permits us to increase the α 's for the fast reactions. The weighting factors can be chosen in many ways. A simple choice could be $\beta_i = 1/k$.

7.1.7 Two better second-order positive schemes

The scheme (7.21) has the disadvantage of slowing down the accuracy for time steps that are large compared to the reaction rates. An alternative can be an implicit version of Collatz scheme which takes the form

$$\rho^{n+1} = \rho^n + h\mathcal{V}\mathbf{f}\left(\rho^{n+1} - \frac{h}{2}\mathcal{V}\mathbf{f}(\rho^{n+1})\right). \quad (7.24)$$

Equation (7.24) involves the solution of the nonlinear system in the unknown \mathbf{x}

$$\mathbf{x} - h\mathcal{V}\mathbf{f}\left(\mathbf{x} - \frac{h}{2}\mathcal{V}\mathbf{f}(\mathbf{x})\right) = \rho^n. \quad (7.25)$$

To solve (7.25), we will introduce a new variable

$$\mathbf{y} = \mathbf{x} - \frac{h}{2}\mathcal{V}\mathbf{f}(\mathbf{x}),$$

so that system (7.25) is equivalent to the new system

$$\mathbf{x} - h\mathcal{V}\mathbf{f}(\mathbf{y}) = \rho^n, \quad \mathbf{x} - \frac{h}{2}\mathcal{V}\mathbf{f}(\mathbf{x}) = \mathbf{y}. \quad (7.26)$$

This system, by Theorem 7.1, can be written in the following equivalent form:

$$\lambda\mathbf{y} - h\mathbf{C}(\mathbf{y}, \mathcal{V})\mathbf{y} = \rho^n - \mathbf{x} + \lambda\mathbf{y}, \quad \mathbf{x} - \frac{h}{2}\mathbf{C}(\mathbf{x}, \mathcal{V})\mathbf{x} = \mathbf{y},$$

where λ is a free parameter. This relation suggests the following iterative procedure to solve (7.26)

$$\begin{aligned} \mathbf{y}^{l+1} &= \left(I - \frac{h}{\lambda}\mathbf{C}(\mathbf{y}^l, \mathcal{V})\right)^{-1} \left(\frac{\rho^n - \mathbf{x}^l}{\lambda} + \mathbf{y}^l\right), \\ \mathbf{x}^{l+1} &= \left(I - \frac{h}{2}\mathbf{C}(\mathbf{x}^l, \mathcal{V})\right)^{-1} \mathbf{y}^{l+1}. \end{aligned} \quad (7.27)$$

Another scheme can be an implicit version of Heun scheme which takes the form

$$\rho^{n+1} = \rho^n + h\frac{1}{2}\mathcal{V}[\mathbf{f}(\rho^{n+1} - h\mathcal{V}\mathbf{f}(\rho^{n+1})) + \mathbf{f}(\rho^{n+1})]. \quad (7.28)$$

Equation (7.28) involves the solution of the nonlinear system in the unknown \mathbf{x}

$$\mathbf{x} - \frac{h}{2}\mathcal{V}[\mathbf{f}(\mathbf{x} - h\mathcal{V}\mathbf{f}(\mathbf{x})) + \mathbf{f}(\mathbf{x})] = \rho^n. \quad (7.29)$$

To solve (7.29), it is convenient to introduce a new variable

$$\mathbf{y} = \mathbf{x} - h\mathcal{V}\mathbf{f}(\mathbf{x}),$$

so that the system is equivalent to the new system

$$\mathbf{y} - h\mathcal{V}\mathbf{f}(\mathbf{y}) = 2\rho^n - \mathbf{x}, \quad \mathbf{x} - h\mathcal{V}\mathbf{f}(\mathbf{x}) = \mathbf{y}.$$

The equivalent form of this system applying Theorem 7.1 will be

$$(1 + \lambda)\mathbf{y} - h\mathbf{C}(\mathbf{y}, \mathcal{V})\mathbf{y} = 2\rho^n - \mathbf{x} + \lambda\mathbf{y}, \quad \mathbf{x} - h\mathbf{C}(\mathbf{x}, \mathcal{V})\mathbf{x} = \mathbf{y}.$$

This suggests the following iterative procedure to solve (7.26)

$$\begin{aligned} \mathbf{y}^{l+1} &= ((1 + \lambda)I - h\mathbf{C}(\mathbf{y}^l, \mathcal{V}))^{-1} (2\rho^n - \mathbf{x}^l + \lambda\mathbf{y}^l), \\ \mathbf{x}^{l+1} &= (I - h\mathbf{C}(\mathbf{x}^l, \mathcal{V}))^{-1} \mathbf{y}^{l+1}. \end{aligned} \tag{7.30}$$

7.1.8 A second-order diagonally implicit Runge–Kutta scheme

The schemes (7.24) and (7.28) have the disadvantage of a slow rate of convergence of the procedures (7.27) and (7.30). An alternative can be the use of diagonally implicit Runge–Kutta schemes, so that, instead of solving a large nonlinear system, we solve a series of smaller nonlinear systems. For example the S -stable two-stage diagonally implicit Runge–Kutta scheme of Alexander after some manipulation takes the form

$$\begin{aligned} \mathbf{x} - \alpha h\mathcal{V}\mathbf{f}(\mathbf{x}) &= \rho^n, \\ \mathbf{y} - \alpha h\mathcal{V}\mathbf{f}(\mathbf{x}) &= \frac{(2\alpha - 1)\rho^n + (1 - \alpha)\mathbf{x}}{\alpha}, \\ \rho^{n+1} &= \mathbf{y}, \end{aligned} \tag{7.31}$$

where $\alpha = 1 \pm \sqrt{2}/2$.

8 Solving stiff IVP's with Maple

Maple is a procedural programming language. It is a computer algebra software that can perform symbolic computation as well as numerical computation, graphics, programming, and more. It can handle differential equations, too. Sometimes Maple can solve them symbolically, finding the general solution. If the equation can not be solved symbolically, one can always find a numerical answer.

If it takes too long to solve an initial value problem (IVP) with the default numerical solver (for example, *rkf45*, which stands for Fehlberg fourth-fifth order Runge-Kutta method) then the IVP might be stiff. A stiff IVP can be solved efficiently with either the default solver using the *stiff=true* option (*rosenbrock*) in **dsolve** command or any of the back- options in *lsode*. Obtained numerical solutions may be plotted using the **odeplot** command in the **plots** package.

First we initialize Maple.

```
> restart;
with(DEtools):
with(plots):
with(linalg):
```

The equation we wish to solve a second order ODE $y''(x) + (\lambda + 1)y'(x) + \lambda y = 0$ with the initial conditions $y(0) = 1, y'(0) = -1$.

By introducing auxiliary variables we can convert it into a set of first order ODEs, where λ is a constant:

$$\begin{aligned}y'(x) &= z(x) \\z'(x) &= -\lambda y(x) - (\lambda + 1)z(x).\end{aligned}$$

We shall investigate how varying λ affects the numerical solution. The above is an example of a stiff equation. If we solve the above analytically, we find that: $y(x) = e^{-x}$. We also have an unwanted solution where $y(x) = e^{-\lambda x}$.

We will show that under certain circumstances forward Euler fails to converge to the real solution, and instead blows up to infinity. In fact forward Euler is only stable for the step sizes that satisfy $h < 2\lambda^{-1}$.

Now we will present two implementations of forward and backward Euler to model the stiff equation:

```
> GeneralForwardEuler := proc(vars, f, init, h, interval)
local makeeqn, ans, solveEqns, assignList, diffValue, newICSet,
i,j,k, ic; ic := init;
makeeqn := nops(u);
ans := [];
ans := [op(ans), ic];

while op(1,ic) < interval do
solveEqns := [];
assignList := [];
for j from 1 by 1 to makeeqn do
assignList := [op(assignList), op(j, u) = op(j, ic)];
end do;
for i from 1 by 1 to makeeqn do
diffValue := eval(op(i, f), assignList);
solveEqns := [op(solveEqns), op(i, u) = op(i, ic) + h * (diffValue)];
end do;
newICSet := solve(convert(solveEqns, set), convert(u, set));
for k from 1 by 1 to makeeqn do
ic := subsop(k = eval(op(k, u), newICSet), ic);
end do;
ans := [op(ans), ic];
end do;

return matrix(ans);
end proc:

> GeneralBackwardEuler := proc(vars, f, init, h, interval)
local makeeqn, ans, solveEqns, assignList, diffValue, newICSet, i, j, k,
ic;
ic := init;
makeeqn := nops(u);
ans := [];
ans := [op(ans), ic];

while op(1,ic) < interval do
solveEqns := [];
assignList := [];
for j from 1 by 1 to makeeqn do
```

```

assignList := [op(assignList), op(j, u) = op(j, ic)];
end do;
for i from 1 by 1 to makeeqn do
diffValue := eval(op(i, f), assignList);
solveEqns := [op(solveEqns), op(i, u) = op(i, ic) + h * (op(i, f))];
end do;
newICSet := solve(convert(solveEqns, set),convert(u, set));
for k from 1 by 1 to makeeqn do
ic := subsop(k = eval(op(k, u), newICSet), ic);
end do;
ans := [op(ans), ic];
end do;

return matrix(ans);
end proc:

```

To investigate how varying the value of λ changes the nature of the numerical solution, first take $\lambda = 1000$ and $h = 0.1$.

```

> u := [x,y,z];
f := [1,z,-1a103*y - (1a103 + 1)*z];
ic := [0,1,-1];

          u := [x,y,z]

          f := [1,z,-1000y - 1001z]

          ic := [0,1,-1]

> 1a103 := 10^3;
          1a103 := 1000

```

If we limit the range for $x = [0, 1.5]$, there doesn't seem to be a problem, until $x = 1.5$ where we have a sudden jump in value.

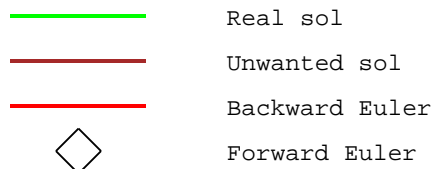
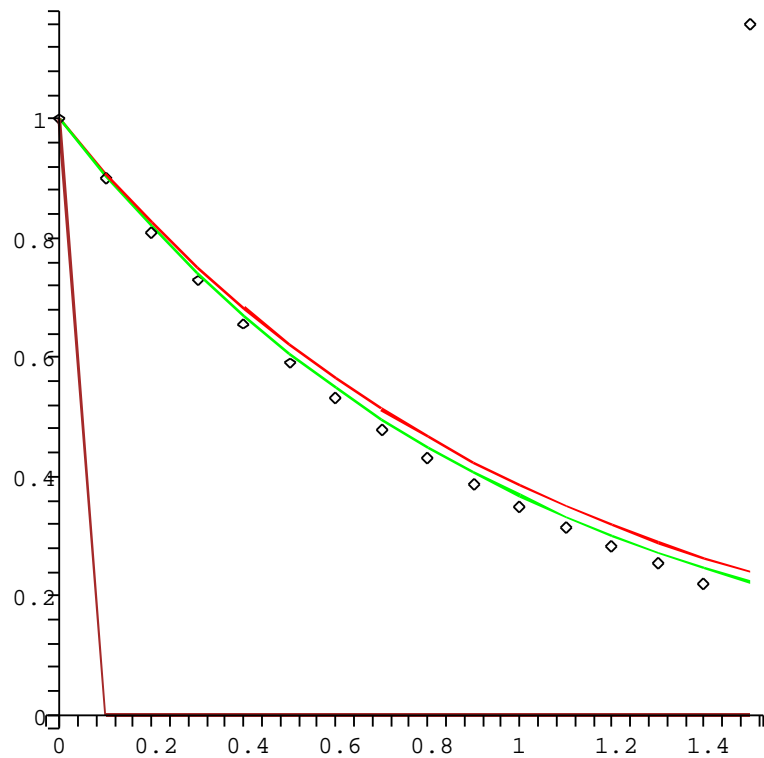
```

> solBE1 := GeneralBackwardEuler(u,f,ic,0.1,1.5):
solFE1 := GeneralForwardEuler(u,f,ic,0.1,1.5):
plot_yz := []:
plot_yzLa := []:
plot_yz1 := []:
plot_yz2 := []:

for i from 1 by 1 to 16 do
plot_yz := [op(plot_yz),[solBE1[i,1], exp(-solBE1 [i,1])]]:
plot_yzLa := [op(plot_yzLa),[solBE1[i,1], exp(-1a103*(solBE1 [i,1])]]:
plot_yz1 := [op(plot_yz1),[solBE1[i,1], solBE1[i,2]]]:
plot_yz2 := [op(plot_yz2),[solFE1[i,1],solFE1[i,2]]]:
end do:

plotsetup("ps", plotoptions="color");
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],
style=[line,line,line, point]);
plotsetup(default);
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],
style=[line,line,line, point]);

```



However, when increasing the range to $x = 1.6$, we find that forward Euler breaks down far more significantly. Increasing x further, and forward Euler becomes very unstable, blowing up to infinity.

```

> solBE2 := GeneralBackwardEuler(u,f,ic,0.1,1.6):
solFE2 := GeneralForwardEuler(u,f,ic,0.1,1.6):
plot_yz := []:
plot_yzLa := []:
plot_yz1 := []:
plot_yz2 := []:

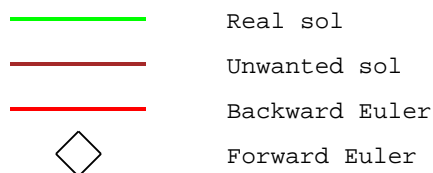
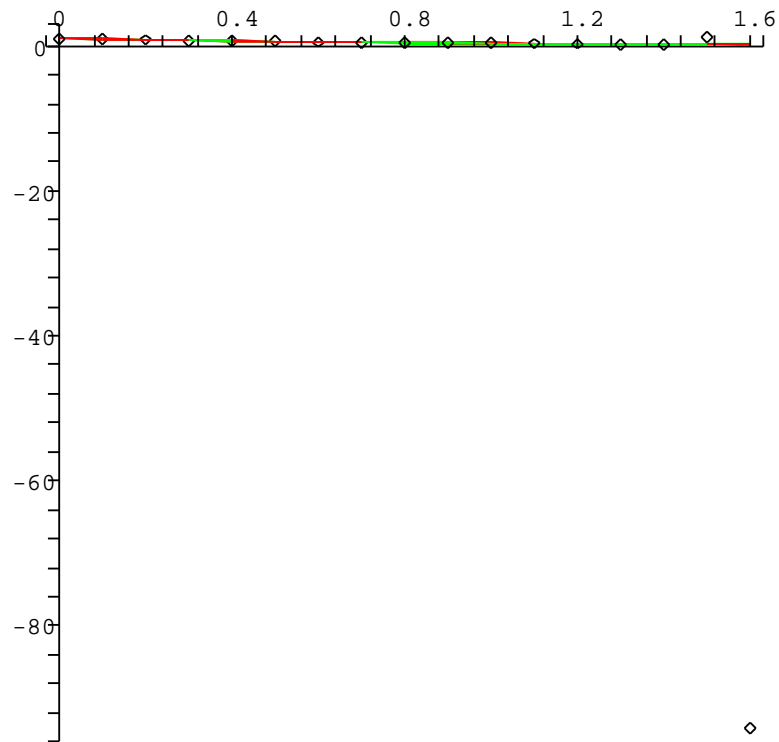
for i from 1 by 1 to 17 do
plot_yz := [op(plot_yz),[solBE2[i,1], exp(-solBE2 [i,1])]]:
plot_yzLa := [op(plot_yzLa),[solBE2[i,1], exp(-1a*(solBE2 [i,1]))]]:
plot_yz1 := [op(plot_yz1),[solBE2[i,1], solBE2[i,2]]]:
plot_yz2 := [op(plot_yz2),[solFE2[i,1],solFE2[i,2]]]:
end do:

```

```

plotsetup("ps", plotoptions="color");
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],
style=[line,line,line, point]);
plotsetup(default);
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],

```



Now we can try $h=0.01$. Here we find that forward Euler breaks down far quicker, in spite of having a step size.

```

> solBE3 := GeneralBackwardEuler(u,f,ic,0.01,0.2):
sol6 := GeneralForwardEuler(u,f,ic,0.01,0.2):
plot_yz := []:
plot_yzLa := []:
plot_yz1 := []:
plot_yz2 := []:

```

```

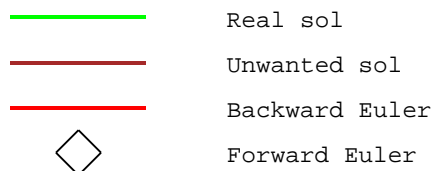
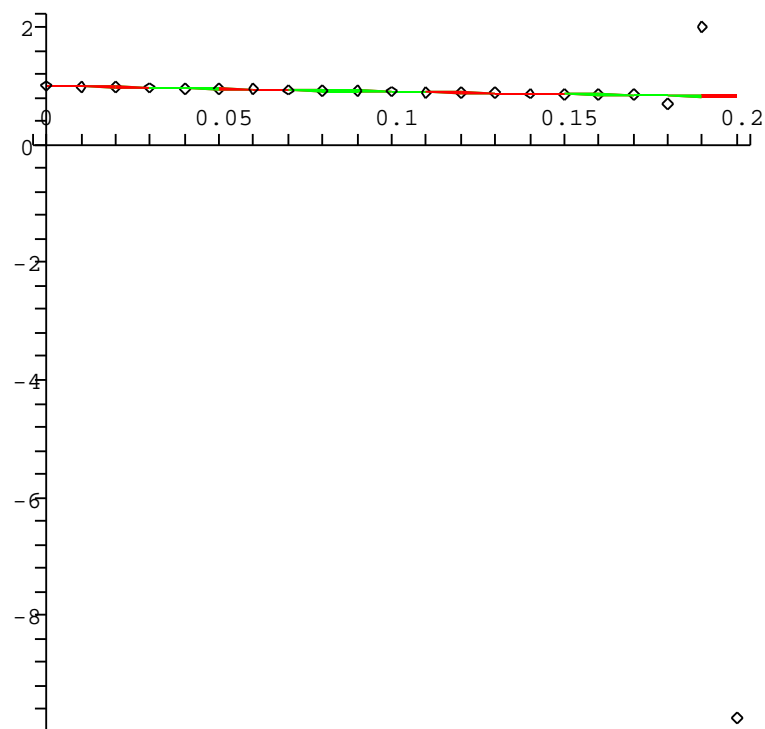
for i from 1 by 1 to 21 do

```

```

plot_yz := [op(plot_yz),[solBE3[i,1], exp(-solBE3 [i,1])]]:
plot_yzLa := [op(plot_yzLa),[solBE3[i,1], exp(-la*(solBE3 [i,1]))]]:
plot_yz1 := [op(plot_yz1),[solBE3[i,1], solBE3[i,2]]]:
plot_yz2 := [op(plot_yz2),[solFE3[i,1],solFE3[i,2]]]:
end do:
plotsetup("ps", plotoptions="color");
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],
style=[line,line,line, point]);
plotsetup(default);
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],

```



Now we can try $h = 0.001$. Here we find that forward Euler is stable. This is not surprising as $h < 2\lambda^{-1}$ where $2\lambda^{-1} = 0.002$.

```

> solBE4 := GeneralBackwardEuler(u,f,ic,0.001,10):
sol8 := GeneralForwardEuler(u,f,ic,0.001,10):
plot_yz := []:

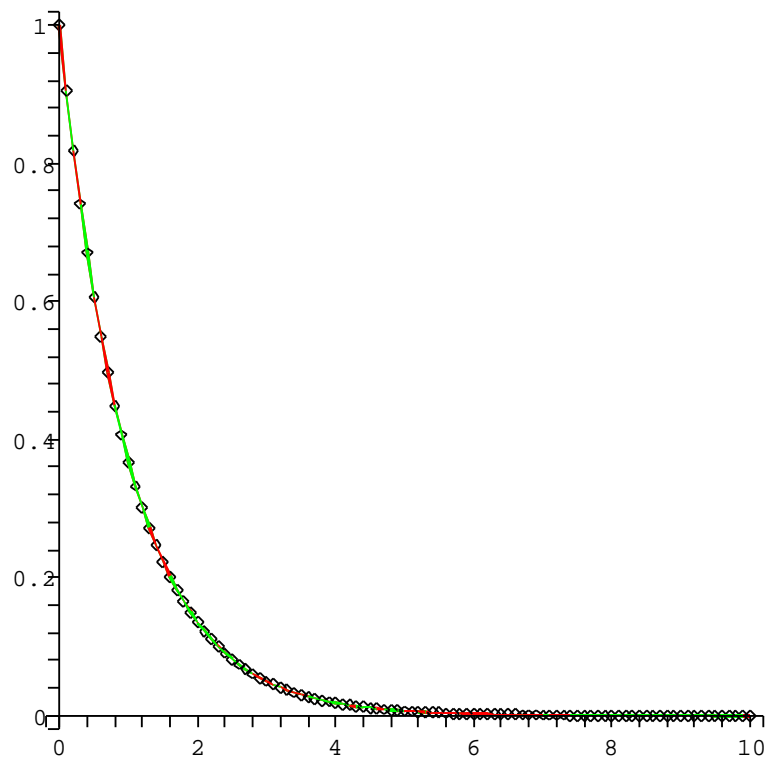
```





```

plot_yzLa := []:
plot_yz1 := []:
plot_yz2 := []:

for i from 1 by 100 to 10001 do
plot_yz := [op(plot_yz),[solBE4[i,1], exp(-solBE4 [i,1])]]:
plot_yzLa := [op(plot_yzLa),[solBE4[i,1], exp(-la*(solBE4 [i,1]))]]:
plot_yz1 := [op(plot_yz1),[solBE4[i,1], solBE4[i,2]]]:
plot_yz2 := [op(plot_yz2),[solFE4[i,1],solFE4[i,2]]]:
end do:
#print(plot_yz);
plotsetup("ps", plotoptions="color");
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],
style=[line,line,line, point]);
plotsetup(default);
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],

```



	Real sol
	Unwanted sol
	Backward Euler
	Forward Euler

Next we can try $\lambda = 1000000$. We shall first let $h = 0.1$.

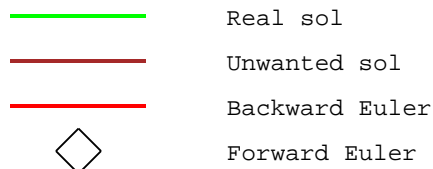
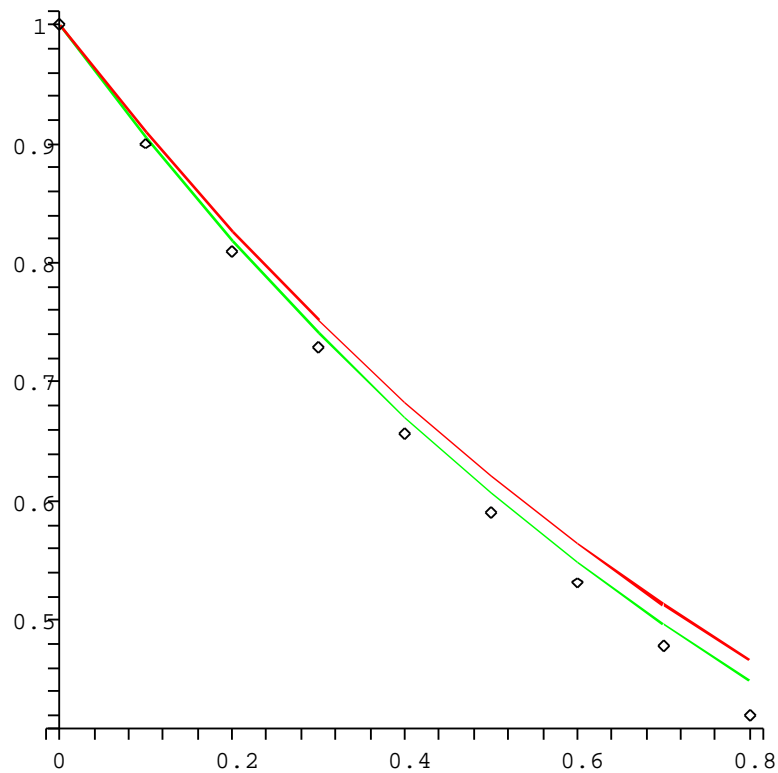
```
> u := [x,y,z];
f := [1,z,-1a106*y - (1a106 + 1)*z];
ic := [0,1,-1];
          u := [x,y,z]
          f := [1,z,-1a106 y - (1a106 + 1) z]
          ic := [0,1,-1]

> la106 := 10^6;
          la106 := 1000000
```

If we limit the range for $x = [0, 0.8]$, there are no sudden "jumps".

```
> solBE5 := GeneralBackwardEuler(u,f,ic,0.1,0.8):
solFE5 := GeneralForwardEuler(u,f,ic,0.1,0.8):
plot_yz := []:
plot_yzLa := []:
plot_yz1 := []:
plot_yz2 := []:

for i from 1 by 1 to 9 do
plot_yz := [op(plot_yz),[solBE5[i,1], exp(-solBE5 [i,1])]]:
plot_yzLa := [op(plot_yzLa),[solBE5[i,1], exp(-la*(solBE5 [i,1]))]]:
plot_yz1 := [op(plot_yz1),[solBE5[i,1], solBE5[i,2]]]:
plot_yz2 := [op(plot_yz2),[solFE5[i,1],solFE5[i,2]]]:
end do:
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],
style=[line,line,line, point]);
```



However, when increasing the range to $x = 0.9$, we encounter the same problem seen earlier with forward Euler.

```

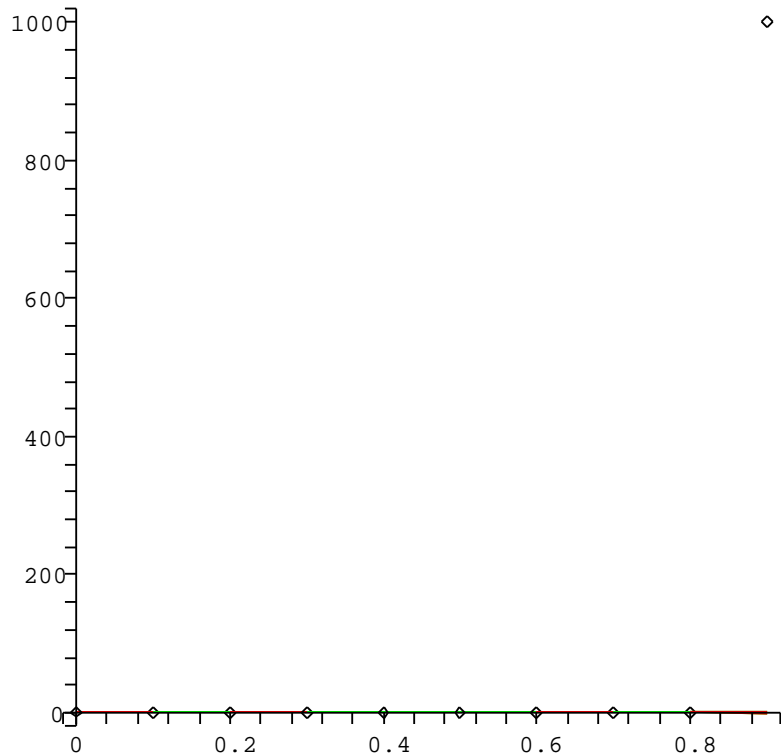
> solBE6 := GeneralBackwardEuler(u,f,ic,0.1,0.9):
solFE6 := GeneralForwardEuler(u,f,ic,0.1,0.9):
plot_yz := []:
plot_yzLa := []:
plot_yz1 := []:
plot_yz2 := []:





for i from 1 by 1 to 10 do
plot_yz := [op(plot_yz),[solBE6[i,1], exp(-solBE6 [i,1])]]:
plot_yzLa := [op(plot_yzLa),[solBE6[i,1], exp(-la*(solBE6 [i,1]))]]:
plot_yz1 := [op(plot_yz1),[solBE6[i,1], solBE6[i,2]]]:
plot_yz2 := [op(plot_yz2),[solFE6[i,1],solFE6[i,2]]]:
end do:
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted

```



```
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],
style=[line,line,line, point]);
```



	Real sol
	Unwanted sol
	Backward Euler
	Forward Euler

Now we can try $h = 0.01$ The solution generated by forward Euler very quickly becomes unstable.

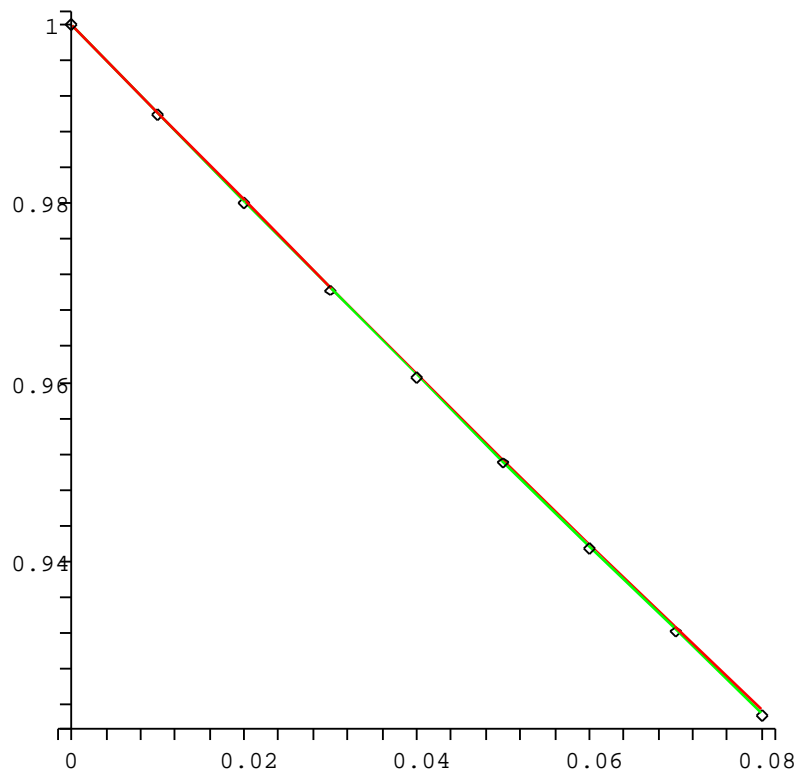
```
> solBE7 := GeneralBackwardEuler(u,f,ic,0.01,0.08):
solFE7 := GeneralForwardEuler(u,f,ic,0.01,0.08):
plot_yz := []:
plot_yzLa := []:
plot_yz1 := []:
plot_yz2 := []:
```

```
for i from 1 by 1 to 9 do
plot_yz := [op(plot_yz),[solBE7[i,1], exp(-solBE7 [i,1])]]:
plot_yzLa := [op(plot_yzLa),[solBE7[i,1], exp(-la*(solBE7 [i,1]))]]:
plot_yz1 := [op(plot_yz1),[solBE7[i,1], solBE7[i,2]]]:
```

```

plot_yz2 := [op(plot_yz2), [solFE7[i,1], solFE7[i,2]]]:
end do:
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler", "Forward Euler"], colour=[green, brown, red, black],
style=[line, line, line, point]);

```



—	Real sol
—	Unwanted sol
—	Backward Euler
◇	Forward Euler

We could try a step size that satisfies the rule for stability say

$$h = 0.000001$$

However, this is computationally very demanding, if we could run this, it would be stable. Note that

$$h < 2\lambda^{-1}$$

where

$$2\lambda^{-1} = 0.000002$$

```

> # procedure for running with a "stable" stepsize
# in practice to takes too long to deliver a result
stepsize := 0.000001:
solBE8 := GeneralBackwardEuler(u,f,ic,stepsize,0.1):
solFE8 := GeneralForwardEuler(u,f,ic,stepsize,0.1):
plot_yz := []:
plot_yzLa := []:
plot_yz1 := []:
plot_yz2 := []:

for i from 1 by 100 to 10001 do
plot_yz := [op(plot_yz),[solBE8[i,1], exp(-solBE8 [i,1])]]:
plot_yzLa := [op(plot_yzLa),[solBE8[i,1], exp(-la*(solBE8 [i,1]))]]:
plot_yz1 := [op(plot_yz1),[solBE8[i,1], solBE8[i,2]]]:
plot_yz2 := [op(plot_yz2),[solFE8[i,1],solFE8[i,2]]]:
end do:
plot([plot_yz, plot_yzLa, plot_yz1, plot_yz2], legend=["Real sol", "Unwanted
sol", "Backward Euler","Forward Euler"], colour=[green, brown, red, black],
style=[line,line,line, point]);

```

Now, consider the IVP arising from a chemistry problem and try to solve it with some of the Maple's incorporated methods:

$$\begin{aligned}
\dot{y}_1 &= -0.04y_1 + 10000y_2y_3, \\
\dot{y}_2 &= 0.04y_2 - 1000y_1y_2 - 3 \cdot 10^7 y_2^2, \\
\dot{y}_3 &= 3 \cdot 10^7 y_2^2,
\end{aligned} \tag{8.32}$$

with initial conditions $y_1(0) = 1, y_2(0) = 0, y_3(0) = 0$.

```

> odesys :=
diff(y1(t),t)=-0.04*y1(t)+10000*y2(t)*y3(t),
diff(y2(t),t)=0.04*y1(t)-10000*y2(t)*y3(t)-30000000*(y2(t))^2,
diff(y3(t),t)=30000000*(y2(t))^2,
y1(0)=1,y2(0)=0,y3(0)=0:
> sol := dsolve(odesys);
sol :=

```

The equation can not be solved symbolically. We will try to solve it symbolically, by using the classical numerical 4th order Runge-Kutta method, which is a build-in method in Maple.

```

> rk4_sol:=dsolve(odesys, y1(t),y2(t),y3(t), type=numeric,
method=classical[rk4]);
rk4_sol := proc(x_classical) ... end proc

```

The system returns a procedure. We can try to plot the solution, but we only get an empty domain.

```

> plots[odeplot](rk4_sol, [t,y1(t)], 0..3):

```

The applied method was not sufficient to solve our problem. Now, let us try the Runge-Kutta-Fehlberg method and attempt to plot the solution obtained with this method:

```

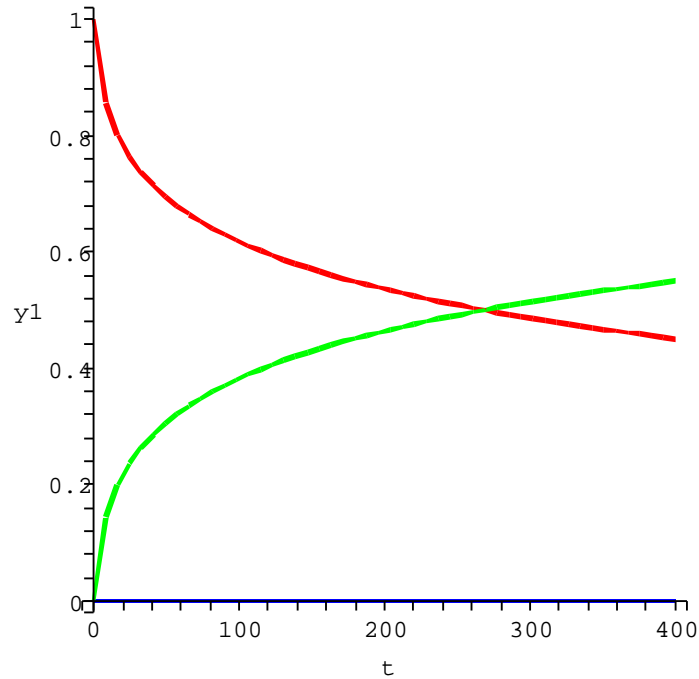
> stiffsystem := dsolve(odesys, [y1(t), y2(t), y3(t)], type=numeric,
method=rkf45, maxfun=10000000);
odeplot(stiffsystem, [[t,y2(t)]], 0..300, color=red, thickness=2);

```

```

stiffsystem := proc(x_rkf45)...end proc
> first:=odeplot(stiffsystem, [[t,y1(t)]], 0..400, color=red, thickness=2):
second:=odeplot(stiffsystem, [[t,y2(t)]], 0..400, color=blue, thickness=2):
third:=odeplot(stiffsystem, [[t,y3(t)]], 0..400, color=green, thickness=2):
display(fir

```

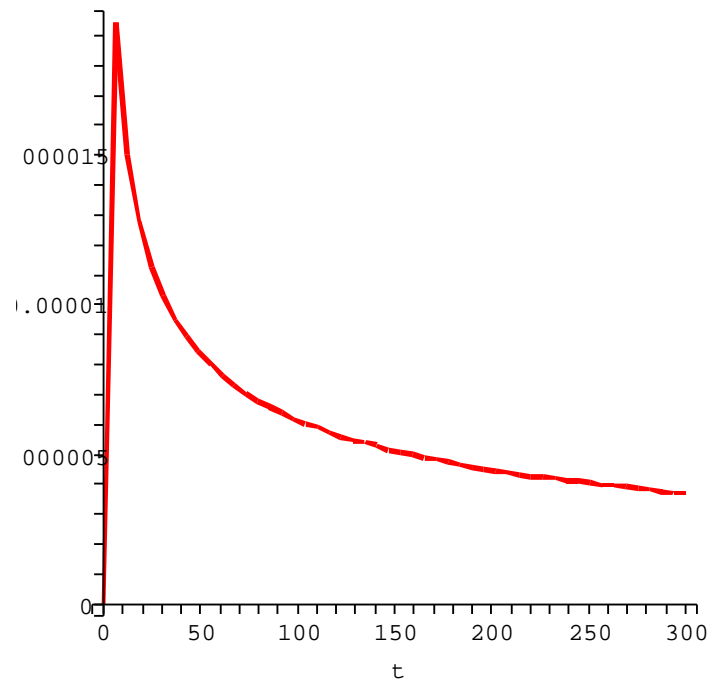


Now we will represent y_2 in a separate chart, too.

```

> odeplot(                                     hickness=2);

```



Let's print the solution as t increases from 0 to 1 by 0.1:

```
> for dt from 0 by .1 to 1 do
  stiffsystem(dt);
od;
```

[t = 0.0, y1 (t) = 1.0, y2 (t) = 0.0, y3 (t) = 0.0]

[t = 0.1, y1 (t) = 0.9960777477208, y2 (t) = 0.00003583031133739, y3 (t) = 0.00388642196785]

[t = 0.2, y1 (t) = 0.9923059495004, y2 (t) = 0.00003508382572584, y3 (t) = 0.00765896667381]

[t = 0.3, y1 (t) = 0.9886739438331, y2 (t) = 0.00003448771948954, y3 (t) = 0.01129156844737]

[t = 0.4, y1 (t) = 0.9851721212508, y2 (t) = 0.00003387917354246, y3 (t) = 0.01479399957556]

[t = 0.5, y1 (t) = 0.9817917893285, y2 (t) = 0.00003324752784505, y3 (t) = 0.01817496314361]

[t = 0.6, y1 (t) = 0.9785250479311, y2 (t) = 0.00003275930440285, y3 (t) = 0.02144219276444]

[t = 0.7, y1 (t) = 0.9753647478928, y2 (t) = 0.00003217254578857, y3 (t) = 0.02460307956140]

[t = 0.8, y1 (t) = 0.9723043293659, y2 (t) = 0.00003170294102878, y3 (t) = 0.02766396769306]

[t = 0.9, y1 (t) = 0.9693378325648, y2 (t) = 0.00003121807773588, y3 (t) = 0.03063094935742]

[t = 1.0, y1 (t) = 0.9664597871114, y2 (t) = 0.00003071473027384, y3 (t) = 0.03350949815826]

Now choose a different numerical method suitable for stiff equations: Rosenbrock-type method.

```
> rosenbrock_sol:=dsolve(odesys, y1(t),y2(t),y3(t), type=numeric,
method=rosenbrock,stiff=true,range=0..300,abserr=0.005):
```

We can review the solution obtained in this way, as follows:

```
> for dt from 0 by .1 to 1 do
  rosenbrock_sol(dt);
od;
```

[t = 0.0, y1 (t) = 1.0, y2 (t) = 0.0, y3 (t) = 0.0]

[t = 0.1, y1 (t) = 0.9960780086289, y2 (t) = 0.00003581995877207, y3 (t) = 0.00388617141229]

[t = 0.2, y1 (t) = 0.9923063308204, y2 (t) = 0.00003511810288788, y3 (t) = 0.00765855107670]

[t = 0.3, y1 (t) = 0.9886748264692, y2 (t) = 0.00003447623539990, y3 (t) = 0.01129069729531]

[t = 0.4, y1 (t) = 0.9851753555818, y2 (t) = 0.00003385980894708, y3 (t) = 0.01479078460918]

[t = 0.5, y1 (t) = 0.9817921205970, y2 (t) = 0.00003327528197591, y3 (t) = 0.01817460412099]

[t = 0.6, y1 (t) = 0.9785202166784, y2 (t) = 0.00003272028403325, y3 (t) = 0.02144706303755]

[t = 0.7, y1 (t) = 0.9753547389896, y2 (t) = 0.00003219244466597, y3 (t) = 0.02461306856572]

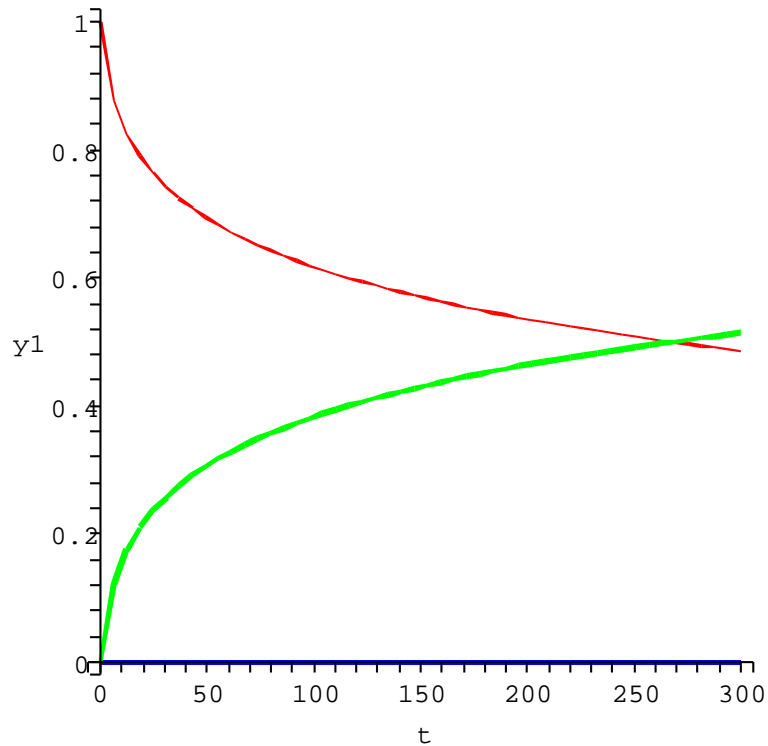
[t = 0.8, y1 (t) = 0.9722907826942, y2 (t) = 0.00003168939342095, y3 (t) = 0.02767752791231]

[t = 0.9, y1 (t) = 0.9693234429559, y2 (t) = 0.00003120875984504, y3 (t) = 0.03064534828415]

[t = 1.0, y1 (t) = 0.9664478149384, y2 (t) = 0.00003074817348513, y3 (t) = 0.03352143688808]

A graphical representation of the solution can be obtained via the following commands:

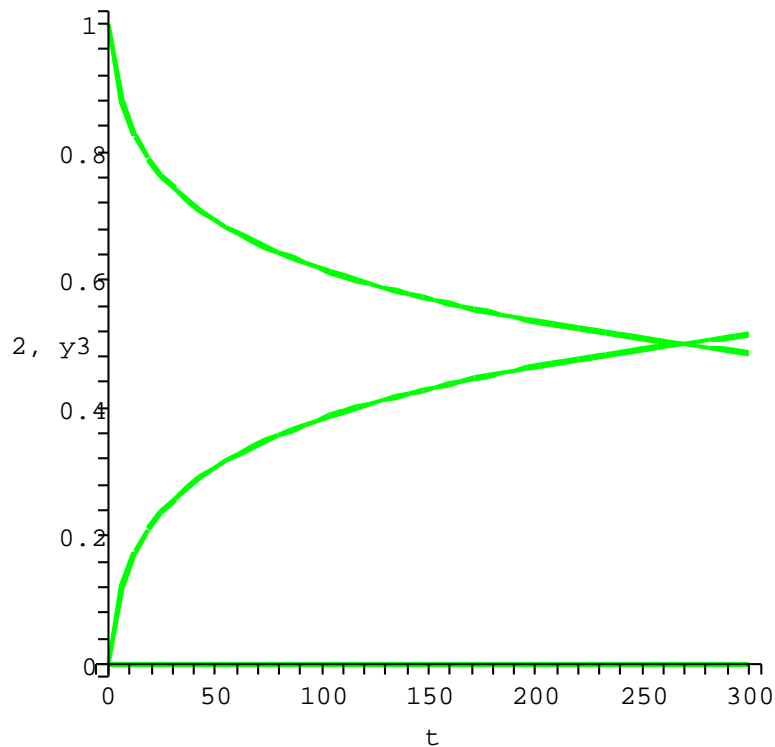
```
> first':=plots[odeplot](rosenbrock_sol, [t,y1(t)], 0..300, color=red):  
> second':=plots[odeplot](rosenbrock_sol, [t,y2(t)], 0..300, color=blue,  
thickness=2):  
> third':=plots[odeplot](rosenbrock_sol, [t,y3(t)], 0..300, color=green,  
thickness=2):  
> plots[
```



Another alternative to solve the stiff system may be the use of the Livermore ODE solver based on backward differentiation formulas. The solution was depicted in this case as well.

```
> stiff_back := dsolve(odesys, y1(t), y2(t), y3(t), type=numeric,  
method=lsode[backfull]);  
odeplot(stiff_back, [[t,y1(t)],[t,y2(t)],[t,y3(t)]], 0..300, color=green,  
thickness=2);
```

```
stiff_back := proc(x_lsode) ... end proc
```



References

- [Ado94] G. Adomian, *Solving Frontier Problems of Physics: The Decomposition Method*, Kluwer Academic Publishers, Boston, MA, 1994.
- [Ala07] M. Alabdullatif, H. A. Abdusalam, E. S. Fahmy, *Adomian decomposition method for nonlinear reaction diffusion system of Lotka-Volterra type*, International Mathematical Forum, 2, 2007, no. 2, 87–96.
- [Aro96] C. J. Aro, *CHEMSODE: a stiff ODE solver for the equations of chemical kinetics*, Computer Physics Communications 97 (1996) 304–314.
- [Ber96] E. Bertolazzi, *Positive and Conservative Schemes for Mass Action Kinetics*, Computers Math. Applic. Vol. 32, No. 6, pp. 29–43, 1996.
- [Bia06] J. Biazar, M. Pourabd, *A Maple program for computing Adomian polynomials*, International Mathematical Forum, 1, 2006, no. 39, 1919–1924.
- [Cab] M. Caberlin, N. Nigam, S. Qazi, *Invariant manifolds in a model for the GABA receptor*
http://www.math.mcgill.ca/nigam/MAIN/pub/caberlin_nigam_qazi.pdf

- [Cli96] L. J. Clifford, A. M. Milne, B. A. Murray, *Numerical modeling of Chemistry an Gas Dynamics During Shock-Induced Ethylene Combustion* Combustion and Flame 104; 311–327, 1996, Elsevier Science Inc.
- [Dam02] V. Damian, A. Sandu, M. Damian, F. Potra, G. R. Carmichael, *The kinetic preprocessor KPP—a software environment for solving chemical kinetics*, Computers and Chemical Engineering 26 (2002) 1567–1579.
- [Dit06] P. Diță, N. Grama, *On Adomians Decomposition Method for Solving Differential Equations*, arXiv:solv-int/9705008 1, May 24, 2006, http://arxiv.org/PS_cache/solv-int/pdf/9705/9705008.pdf
- [Edw97] J. T. Edwards, J. A. Roberts, N. J. Ford, *A comparison of Adomian’s decomposition method and Runge Kutta methods for approximate solution of some predator prey model eqautions*, Numerical Analysis Report no. 309, October 1997.
- [Fed97] R. P. Fedkiw, B. Merriman, S. Osher, *High Accuracy Numerical Methods for Thermally Perfect Gas Flows with Chemistry*, Journal of Computational Physics, 132, 175-190 (1997) article no. CP965622.
- [Kay04] D. Kaya *A reliable method for the numerical solution of the kinetics problems*, Appl. Math. Comput. 156 (2004) 261-270.
- [Kno98] O. Knoth, R. Wolke, *Implicit-explicit Runge-Kutta methods for computing atmospheric reactive flows*, Applied Numerical Mathematics 28 (1998) 327–341.
- [Lam85] S. H. Lam, *Singular perturbation for stiff equations using numerical methods*, Recent Advances in the Aerospace Sciences, Corrado Casci, Ed., Plenum Press, New York and London, pp.3–20, 1985.
- [Lam93] S. H. Lam, *Using CSP to Understand Complex Chemical Kinetics*, Combustion Science and Technology, 89, 5–6, pp. 375, 1993.
- [Lam94] S. H. Lam, D. A. Goussis, *The CSP Method for Simplifying Kinetics*, International Journal of Chemical Kinetics 26, pp. 461–486, 1994.
- [Lam95] S. H. Lam, *Reduced Chemistry Modeling and Sensitivity Analysis*, Lecture notes for Aerothermochemistry for Hypersonic Technology, 1994-1995, <http://www.princeton.edu/~lam/SHL/VKI.pdf>
- [Lor99] R. Lorenzini, L. Passoni, *Test of numerical methods for the integration of kinetic equations in tropospheric chemistry*, Computer Physics Communications 117 (1999) 241–249.
- [Pre88] <http://www.library.cornell.edu/nr/cbookcpdf.html>
- [Rae02] M. Rae, M. N. Berberan-Santos, *Pre-equilibrium approximation in chemical and photophysical kinetics*, Chemical Physics 280 (2002) 283-293.

- [San97] A. Sandu, J. G. Verwer, M. Van Loon, G. R. Carmichael, F. A. Potra, D. Dabdub, J. H. Seinfeld, *Benchmarking stiff ode solvers for atmospheric chemistry problems-I. Implicit vs explicit*, Atmospheric Environment, Vol. 31, No. 19, pp. 3151–3166, 1997.
- [Say95] R. D. Saylor, G. D. Ford, *On the comparison of numerical methods for the integration of kinetic equations in atmospheric chemistry and transport models*, Atmospheric Environment, Vol. 29, No. 19, pp. 2585-2593, 1995.
- [Ver95] J. G. Verwer, D. Simpson, *Explicit methods for stiff ODEs from atmospheric chemistry*, Applied Numerical Mathematics 18 (1995) 413–430.
- [Ver97] J. G. Verwer, E.J. Spee, J.G. Blom, W. Hundsdorfer, *A Second Order Rosenbrock Method Applied to Photochemical Dispersion Problems*, Report MAS-R9717, August 31, 1997,
<http://ftp.cwi.nl/CWIreports/MAS/MAS-R9717.pdf>
- [Vos01] D. A. Voss, A. Q. M. Khaliq, *Parallel Rosenbrock methods for chemical systems*, Computers and Chemistry 25 (2001) 101-107.