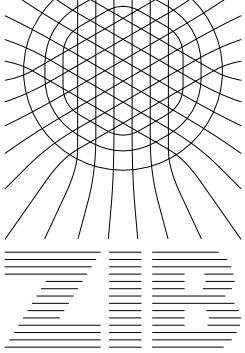
Konrad-Zuse-Zentrum für Informationstechnik Berlin



U. Nowak

Dynamic Sparsing in Stiff Extrapolation Methods

Preprint SC 92–21 (October 1992)

U. Nowak

Dynamic Sparsing in Stiff Extrapolation Methods

Abstract

Based on a simple stability analysis for the semi–implicit Euler discretization a new dynamic sparsing procedure is derived. This procedure automatically eliminates "small" elements of the Jacobian matrix. As a consequence, the amount of work needed to handle the linear algebra within a semi–implicit extrapolation integrator can be reduced drastically. Within the course of integration the sparsing criterion, which decides what "small" means, is dynamically adapted to ensure stability of the discretization scheme. Thus, stepsize restrictions due to instability can be avoided. Numerical experiments for quite different problems show robustness and efficiency of this dynamic sparsing technique. The techniques developed here in the context of stiff extrapolation integrators can, in principle, be applied to W–methods, where exact Jacobians may be replaced by "sufficiently good" approximations.

Keywords: Large scale integration, extrapolation methods, stiff ODEs, W–methods, sparse matrix techniques.

Contents

1. Introduction

Solving an initial value problem

$$y' = f(y) \ , \quad y(t_0) = y_0 \ , \quad y \in \mathbb{R}^n \tag{1.1}$$

with a stiff integrator, a significant part of the overall computing time is spent for the evaluation of the Jacobian matrices

$$J := f_y(y)$$

and for linear algebra computations with these matrices. This time is wasted, if the problem is nonstiff or only mildly stiff. In that case, the use of a nonstiff (or explicit) integrator is more appropriate. However, the degree of inefficiency depends, mainly, on the dimension n of the current problem and on the fashion of the linear algebra procedures used. In the opposite situation, i.e. solving a really stiff problem with an explicit integrator, the effect is more distinct. In the best case, the required computing time increases dramatically, but quite often the problem cannot be solved correctly.

Preliminary Considerations

The main objective of the dynamic sparsing algorithm presented in this paper is to reduce linear algebra computing time when solving large stiff systems. The very simple and not new idea is to drop elements of the Jacobian. In doing so, some essential questions arise. First, how to exploit a reduced number of nonzero elements. Second, which of the elements are really open for dropping. Finally, which stiff methods actually allow a modification of the Jacobian without being strongly affected.

Distinct benefits can be expected, if the sparsing is done under structural aspects only, e.g. dropping all off–diagonal elements or all elements which destroy another nice pattern like band structure or block structure. This technique works quite good, if knowledge of the underlying problem induces the dropping procedure. In the general case, however, neglecting the problem of stiffness and forcing a special structure often leads to an integration with stepsizes of the order of an explicit method.

An alternative is the use of general sparse linear algebra techniques. Structural aspects are no longer of great importance. Dropping a sufficiently large number of elements will speed up the computation. However, a general sparse linear algebra mode only pays off, if the (sparsed) Jacobian is really sparse and the dimension of the problem (1.1) is not too small. Thus, the problem class where the dynamic sparsing algorithm may be applied profitably is restricted.

But other approaches which, ultimately, try to avoid the above mentioned problem of wasting computing time in a stiff integrator, have also structural drawbacks. They will be discussed shortly in the next two subsections. In the last subsection, the question of choosing an appropriate stiff integrator is examined. In Section 2, a properly working sparsing criterion is derived and the dynamic sparsing algorithm for extrapolation methods is presented. The results of some numerical experiments are given in Section 3.

Type insensitive methods

In order to solve (1.1), the problem of specifying the appropriate type of method (stiff or nonstiff), gave rise to the development of so–called type insensitive methods. Such methods aim at deciding for each individual integration interval whether stiff or nonstiff numerical integration should be used exclusively. Especially for multistep methods and Runge–Kutta methods several attempts have been made to develop such integrators, cf. BUTCHER [5], PETZOLD [22], RENTROP [23], SHAMPINE [25], SOTTAS [26] – just to quote a few. Loosely speaking, such a method is able to switch automatically from an explicit to an implicit discretization – and vice versa. For the time being, the term "implicit" is used to indicate the contrast to "explicit" and means "suited for stiff problems". The crucial point is the development of cheap and robust switching criteria, especially for the switch from the explicit to the implicit method – the so–called problem of stiffness detection. Usually, an explicit method can be used efficiently as long as the stepsize needed to achieve the required accuracy in the solution is smaller than the maximum possible stepsize due to stability. Thus, the stability stepsize bound of the explicit method must be estimated. Mostly, this is done by estimating the spectral radius of the Jacobian matrix. But within the phase of explicit integration, the Jacobian is not available and estimates are needed. Several techniques have been developed to get meaningful estimates, but the problem of automatic stiffness detection is still not fully solved.

But, even with a perfectly working switching algorithm and an efficient nonstiff or stiff integrator respectively, type switching methods have, by construction, the following drawback. A substantial gain in the overall computing time can be expected only, if the problem at hand allows to replace a significant number of implicit integration steps by explicit steps of almost the same size. In [22, 23, 26] no benefits, rather losses, are reported for the type switching methods solving the problems of the STIFF DETEST [14].

Partitioning and Projection Methods

If the problem (1.1) can be partitioned into a stiff part and a nonstiff part

$$\begin{aligned} y_s' &= f_s(y_s, y_n) \qquad \text{(stiff)} \\ y_n' &= f_n(y_s, y_n) \qquad \text{(nonstiff)} \end{aligned} \tag{1.2}$$

the system can be treated with an implicit method for the stiff components and an explicit method for the nonstiff part, see e.g. HOFER [20], RENTROP [23], STREHMEL/WEINER [30]. In this approach, the first task is to specify appropriate methods, as, in contrast to the type switching schemes, the explicit and implicit discretization are used simultaneously. For Runge–Kutta methods, the theory of P–series, HAIRER [18], has its origin in the study of the order properties of such methods. If the partitioning (1.2) is not known in advance, or if it changes during the integration, an adaptive partitioning algorithm is required. The development of a cheaply and reliably working method turns out to be a quite difficult task [23]. The general drawback of the method is obvious. The portion of components which can be treated explicitly must be significant in order to save computing time – compared to an overall stiff treatment.

So–called projection methods, see e.g. BJÖRCK [4], GEAR/SAAD [17], WATKINS/HANSONSMITH [31], may be interpreted as generalized partitioning methods. Within such methods, one tries to approximate the stiff subspace of the Jacobian. Somehow, these methods can be viewed as being in between a pure projection method and a simple dropping method. Again, substantial gains are possible only, if the dimension of the subspace is significantly less than the dimension n of (1.1) – a perfectly working separation algorithm assumed.

Extrapolation Methods

Although modern stiff extrapolation methods, see DEUFLHARD [7] for an overview, are quite outstanding candidates for the above mentioned techniques, the investigations made up to now on type insensitive methods concentrate on multistep and Runge–Kutta methods. On the contrary, the dynamic sparsing procedure is derived in the context of extrapolation methods, but may be used also in combination with other discretizations. However, such a method must accept perturbations of the Jacobian. Rosenbrock methods, for example, require the exact Jacobian within their order conditions, whereas so–called W-methods, cf. STEINHAUG/WOLFBRANDT [29], fall into the class of possible methods.

In the present context, one attraction of extrapolation methods is, that pairs of explicit/implicit discretizations are available within a unified frame of efficient extrapolation and control techniques. First, there is the pair explicit/semi–implicit Euler discretization, combined with h–extrapolation. Second, with h^2–extrapolation, there is the pair explicit midpoint rule (the

GRAGG/BULIRSCH /STOER–algorithm)/semi–implicit midpoint rule (due to BADER/DEUFLHARD [2]). So, if the Jacobian is dropped totally, an approved explicit discretization appears. Furthermore, the extrapolation process and the order and stepsize control algorithm due to DEUFLHARD [6] need not be changed. Rather, a quite straightforward realization of the sparsing algorithm is possible, e.g. without indroducing additional heuristic control devices.

2. Derivation of the Algorithm

As pointed out in the preceding section, an appropriate criterion for dropping "unnecessary" elements is required. Such a criterion is now derived by a simple stability analysis of the underlying methods. The new method is then interpreted in terms of recent theoretical results due to DEUFLHARD [8]. So, in what follows, nothing is really new, but the proper collection of some well–known considerations to a reliable algorithm.

The Basic Sparsing Criterion

The explicit Euler discretization applied to the problem (1.1) reads

$$y_{n+1} = y_n + hf(y_n). \tag{2.1}$$

The semi–implicit Euler discretization is given by

$$(I - hA)(y_{n+1} - y_n) = hf(y_n) \tag{2.2}$$

with

$$A := f_y(y_n). \tag{2.3}$$

The integrators which are based on these discretizations (the codes EULEX and EULSIM) use Richardson extrapolation to get approximations of higher and variable order. Furthermore, the codes are equipped with a sophisticated order and stepsize control [6]. Nevertheless, it is sufficient to look on a basic stability property of (2.1, 2.2) to motivate the sparsing criterion used finally. Application of both discretizations to the famous Dahlquist test equation

$$y' = \lambda y \ , \quad y(0) = 1 \ , \quad Re(\lambda) \leq 0 \ , \tag{2.4}$$

yields the well known stability domains for the methods (2.1) and (2.2) respectively, that is,

$$\begin{aligned} S_{EE} &= \{z \in \mathbb{C}; \mid 1 + z \mid \leq 1\} \\ S_{IE} &= \{z \in \mathbb{C}; |(1 - z)^{-1}| \leq 1\} \end{aligned}$$

where

$$z = h\lambda.$$

The requirement for A–stability ($S \supset \mathbb{C}^-$) gives rise to the stability stepsize bound of the explicit Euler method

$$h_{EE}^{max} \leq 2/|\lambda|, \tag{2.5}$$

whereas the semi–implicit Euler method is A–stable. Note that for the linear problem (2.4) the semi–implicit discretization (2.2) coincides with the usual implicit Euler discretization. In terms of A–stability, these methods are sometimes regarded as "superstable" (a notation due to Dahlquist). Applied to nonlinear problems, this disadvantage vanishes for the semi–implicit discretization.

The usual generalization of the stepsize bound (2.5) for nonlinear systems is

$$h_{EE}^{max} \leq 2/\|f_y\|. \tag{2.6}$$

Herein, a suitable norm $\| \, . \, \|$ must be chosen. Note that any type switching method will have to make use of such a norm based criterion.

In order to derive a sparsing criterion, one may look at (2.5) in a special manner. Assume that a stepsize h is given. Then, as long as

$$|\lambda| \leq 2/h,$$

holds, the explicit discretization (2.1) can be used. In other words, one may use the semi–implicit discretization (2.2), but with $A \equiv 0$. Now, as an entry based criterion is required, instead of the generalization (2.6) one may intuitively generalize in a different way. An element $a_{i,j}$ of the Jacobian (2.3) can be dropped, if

$$|a_{i,j}| \leq 2/h \tag{2.7}$$

holds. In a rough sketch, one may interpret the criterion (2.7) in the following way. If the component y_i depends in a non–stiff fashion on the component y_j the element $a_{i,j}$ is not necessary for the stability of the integration. Thus, this entry of the Jacobian may be dropped.

In this form, however, the criterion (2.7) cannot be used. First, the constant 2 in the nominator of the right hand side of (2.7) must be replaced by a more conservative value (or safety factor). There are, indeed, several reasons for that, not only numerical experience.

Recall that the integrators use extrapolation with variable order. Thus, the extrapolated semi–implicit (sometimes called linerly implicit) Euler is $A(\alpha)$–stable solely (α close to 90°). Furthermore, the problem is usually nonlinear. Finally, the stepsize h in (2.7) is an internal estimate. Normally, this stepsize estimate is computed after each successful integration step and is based on the current internal error estimates. If estimated in a stiff integrator, the size of h reflects only the dynamics of the problem and is not limited due to missing stability. Now, underestimating the maximum possible stepsize forces a criterion based on (2.7) to eliminate more elements. As a consequence, the next stepsize estimate, done within an integrator which is now partially explicit (totally if all elements of A have been dropped), may suffer from a

too extensive sparsing of A. This stepsize is again too small (compared to the optimal stepsize of the stiff method) and the effect may strengthen. On the other hand, if this trapping effect can be avoided, one may drop more elements than originally indicated by a criterion of type (2.7). If the linear algebra computations are speeded up so far, that the increasing number of steps is, at least, compensated, one may optimize the dynamic sparsing procedure in that sense.

A second objection to the simple criterion (2.7) is its unscaled form. As the problem of scaling invariance is of great importance for the reliability of a method, this distinctive feature needs to be discussed in more detail.

Scaling Invariance

In practical applications the scaling invariance property of a code plays an important role. This means, the current scaling of the problem should not affect the algorithmic performance. In other words, the code should be invariant under rescaling of some or all components of y, say by $y_i \to s_i y_i$. To discuss the general case, consider a scaling transformation

$$y \to Sy =: \bar{y} \ , \quad S = diag(s_1, ..., s_n). \tag{2.8}$$

Insertion into the original problem (1.1) yields the transformed problem

$$\bar{y}' = \bar{f}(\bar{y}) := Sf(S^{-1}\bar{y}) \ , \quad \bar{y}_0 = Sy_0. \tag{2.9}$$

The analytical solution and the numerical approximation of both discretizations (2.1) and (2.2), turn out to be covariant, i.e. $\bar{y}(t) = Sy(t)$ and $\bar{y}_{n+1} = Sy_{n+1}$ hold. So, invariance is lost in the code if unscaled norms enter into the control of the algorithm. Typically, norms of the form $\|\Delta y\|$, where Δy denotes the difference of two internal approximations to $y(t)$, have to be evaluated. Obviously,

$$\|\Delta y\| \neq \|\Delta \bar{y}\| = \|S\Delta y\| \tag{2.10}$$

holds in general. Therefore, to ensure algorithmic invariance, scaled (or weighted) norms are internally used, i.e.

$$\|\Delta y\| \to \|D^{-1}\Delta y\| \ . \tag{2.11}$$

Assume for the moment that for the internal scaling matrix D a choice

$$D = diag(y_1, ..., y_n) \tag{2.12}$$

is possible. Note that the replacement (2.11) with the choice (2.12) forces the usual error check $\|\Delta y\| \leq tol$ to test for the relative error. With (2.12), the internal scaling matrix $\bar{D}$ for the transformed problem (2.9) reads

$$\bar{D} = diag(\bar{y}_1, ..., \bar{y}_n) = SD$$

and, in contrast to (2.10),

$$\|\bar{D}^{-1}\Delta\bar{y}\| = \|D^{-1}S^{-1}S\Delta y\| = \|D^{-1}\Delta y\| \tag{2.13}$$

holds. Thus, the algorithmic performance is now invariant.

The sparsing criterion (2.7) monitors the values of Jacobian elements. Consequently, these elements must be internally scaled to achieve invariance also for the sparsing algorithm. Straightforward calculation shows that the Jacobian of the transformed problem (2.9) is given by

$$\bar{A}' := \bar{f}_{\bar{y}} = Sf_yS^{-1} = SAS^{-1}. \tag{2.14}$$

Inspired by (2.11), an internal scaling of the Jacobian (2.3) of the form

$$A \to D^{-1}AD \tag{2.15}$$

guarantees invariance due to

$$\bar{D}^{-1}\bar{A}\bar{D} = D^{-1}S^{-1}SAS^{-1}SD = D^{-1}AD.$$

However, the choice (2.12) is not always possible. Either as $y_i \to 0$ or if a relaxed relative error control is more adequate for the problem at hand. So, (2.12) is replaced by

$$D = diag(y_1^w, ..., y_n^w). \tag{2.16}$$

Herein, y^w denotes an internal weighting vector which is updated internally within the course of the integration. Details are omitted here. But the fact that the internal weights which enter into the order and stepsize control, are also used in the sparsing criterion is of great importance for the reliability of the sparsing criterion. Combining (2.15),(2.16) and (2.7), the actually implemented sparsing criterion reads as follows

$$if(y_j^w|a_{i,j}|/y_i^w \leq \sigma/H) \quad then \;\; a_{i,j} := 0, \tag{2.17}$$

where

H	:	current basic stepsize of the integrator
σ	:	safety factor, $\sigma \leq 1$
y^w	:	internal weighting vector of the integrator.

Interpretation of the Method

In principle, the new method is nothing else than the semi–implicit Euler method, where the exact Jacobian $A = f_y$ is replaced by a modified one, say $\hat{A}$, i.e.

$$A \to \hat{A} \tag{2.18}$$

with

$$\|A - \hat{A}\| \leq \delta_0. \tag{2.19}$$

If not all elements are dropped, one has a new method which is somehow in between (2.1) and (2.2). Both basic methods, as well as variants and extensions, have been studied extensively – from theoretical and computational points of view, see e.g. [7, 9, 10]. Recall that the semi–implicit Euler discretization can be interpreted as an explicit Euler discretization of the transformed problem

$$y' - Ay = f(y) - Ay. \tag{2.20}$$

As the Jacobian is only needed to ensure stability, a method which eliminates some elements of the Jacobian (simultaneously on both sides of (2.20)) still meets the requirements for consistency and convergence of the method. The extrapolation process and the order and stepsize control can be done as usual. The replacement (2.18) just influences the maximum permitted stepsize. This can nicely be seen by looking to some recent uniqueness theorems due to DEUFLHARD [8]. Herein, stiff ODE problems and implicit discretization methods are characterized by affine invariant Newton–type uniqueness theorems. Within that frame, the basic discretization (2.2) is interpreted as one possible realization of a simplified Newton method in function space. "Simplified" means that from the global, continuous Jacobian information $f_y(y(t)), t \in [t_n, t_{n+1}]$ just the local information $A := f_y(y(t_n))$ is used. Clearly, with the replacement (2.18) the simplified Newton method is replaced by a Newton–like method, again in function space. The local continuation property for the solution of (1.1) is discussed in terms of a characteristic time constant $\bar{\tau}$. Due to (2.19) this constant is reduced according to

$$\bar{\tau} \to \hat{\tau} = \bar{\tau}/(1 + \delta_0 \bar{\tau}) \leq \bar{\tau}. \tag{2.21}$$

Thus, dropping small terms in the Jacobian is a slight modification of the basic method (2.2) which leads to a minor reduction of the maximum permitted stepsize.

Algorithmic Realization

In order to discuss the algorithmic realization of the dynamic sparsing procedure one may look at the following informal algorithm which – very roughly – describes one basic step (from $\bar{t} \to \bar{t} + H$) of the extrapolated semi–implicit Euler method. Herein, H denotes the so–called basic stepsize, q the depth of extrapolation and $\bar{y}$ the extrapolated approximation to $y(\bar{t})$.

Informal Algorithm:

Basic step:

Given: $\bar{t}$, $\bar{y}$, H, q

$$
\begin{aligned}
t_0 &:= \bar{t} \\
\eta_0 &:= \bar{y} \\
f_0 &:= f(\eta_0) \\
A &:= f_y(\eta_0) \\
\hat{A} &:\approx A \qquad \text{(Sparsing)}
\end{aligned}
$$

Extrapolation loop:

do $j = 1, \ldots, q+1$

$$
\begin{aligned}
h_j &:= H/j \\
LU &:= (I - h_j \hat{A}) \qquad \text{(LU–Decomposition)}
\end{aligned}
$$

Internal steps:

do $l = 0, \ldots, j-1$

$$
\begin{aligned}
f_l &:= f(\eta_l) \\
\eta_{l+1} &:= \eta_l + (LU)^{-1} h_j f_l \qquad \text{(Solve)}
\end{aligned}
$$

enddo

for $j > 1$:

extrapolation

convergence monitor

enddo

order and stepsize selection: H^{new}, q^{new}

extrapolation: $\bar{y}^{\text{new}}$

do next basic step $(\bar{t} := \bar{t} + H)$

Obviously, the dynamic sparsing procedure is a very local addition to the usual algorithm. A more interesting question is, in which way the linear algebra routines are affected. In the sparse version of the code EULSIM, the code EULSIS, the linear system solution is done with the aid of the sparse matrix package MA28 from Harwell [11, 12]. The compilation follows the lines presented in Duff/Nowak [13].

Within the MA28 package there are two routines to factor a given matrix. The expensive Analyse/Factorize routine MA28A analyses the matrix and tries to minimize the number of fill–in elements in its LU–decomposition. Besides, there is the fast Factorize routine MA28B which factors a matrix with the same nonzero pattern as from a previous call to MA28A. But the values may have changed, thus the now prescribed pivot sequence may become not appropriate. This is internally checked and one may restart with a factorization by MA28A, if numerical instability is indicated. Provided that the structural nonzero pattern is known exactly, after a first decomposition with MA28A, the fast factor routine may be used, in principle, throughout

the whole integration. Numerical experience shows, that during an integration quite often just 1–3 expensive ANALYSE/FACTORIZE calls are necessary. All other decompositions can be done by the fast FACTORIZE routine.

Things change, if the sparsing algorithm is added and the criterion (2.17) really removes elements from the Jacobian. Now, the pattern may change from step to step. So, doing dynamic sparsing requires, at least, one ANALYSE/FACTORIZE call per step (to be precise, after each Jacobian evaluation). Thus, fixing the Jacobian for more than one step will pay off especially in combination with dynamic sparsing. Note that typically $q \in \{2, 3, 4\}$ holds, thus one can get over this additional amount of work.

As an interesting variant, one may combine the sparsing procedure directly with the Jacobian evaluation. This means, that the exact Jacobian A must never be stored. Only the nonzero elements of the sparsed Jacobian $\hat{A}$ need to be hold.

3. Numerical Experiments

In this section some numerical results, illustrating the performance of the dynamic sparsing procedure (2.17), are presented. Most of the experiments have been carried out in FORTRAN double precision on a SUN SPARC1+ Workstation, using the Sun FORTRAN Compiler with standard options except for the optimization level "-O1" (due to a sometimes buggy compilation for higher levels). Some of the computations were performed on the CRAY Y–MP (in single precision) of the Konrad–Zuse–Zentrum für Informationstechnik Berlin. The following discussion will focus on the question of reliability and efficiency of the new method within the semi–implicit Euler discretization. Testing the procedure within the frame of the semi–implicit midpoint rule (code METAN1) yields similar results.

The Oregonator problem

This rather small problem, describing an oscillating chemical reaction system, is a quite popular test problem for stiff/nonstiff switching methods. The underlying chemical problem is the Belousov-Zhabotinsky reaction system with the kinetic model of FIELD/NOYES [15]. The equations for the dimensionless concentrations, taken from SEIDER/WHITE III/PROKOPAKIS [24], read

$$\begin{aligned} y_1' &= 77.27(y_2 - y_1 y_2 + y_1 - 8.375 \cdot 10^{-6} y_1^2) \\ y_2' &= (-y_2 - y_1 y_2 + y_3)/77.27 \\ y_3' &= 0.161(y_1 - y_3), \end{aligned}$$

with initial values , again from [24],

$$y_1(0) = 4. \,, \quad y_2(0) = 1.1 \,, \quad y_3(0) = 4.$$

First, to depict the oscillatory nature of this problem, the solution and stepsizes for a simulation over more than 3 cycles ($t_0 = 0,\ t_{end} = 1000$) with the standard code EULSIM are shown in Figure 3.1. Note that the values are in logarithmic scale. The cyclic behavior of the solution carries over to the behavior of the stepsizes. Now, equipping the code EULSIM with the dynamic sparsing procedure, one may check the robustness of this algorithm. Of course, running the new code (DYSEUL) on the *Oregonator* problem, no gain can be expected, as the arising linear systems are still solved with the standard full mode solver DGEFA/DGESL from LINPACK. Rather, the safety factor σ is varied and the changes in the performance of DYSEUL are monitored by checking the usual indicators *nstep* (number of integration steps), *nfcn* (number of function evaluations), *ndec* (number of

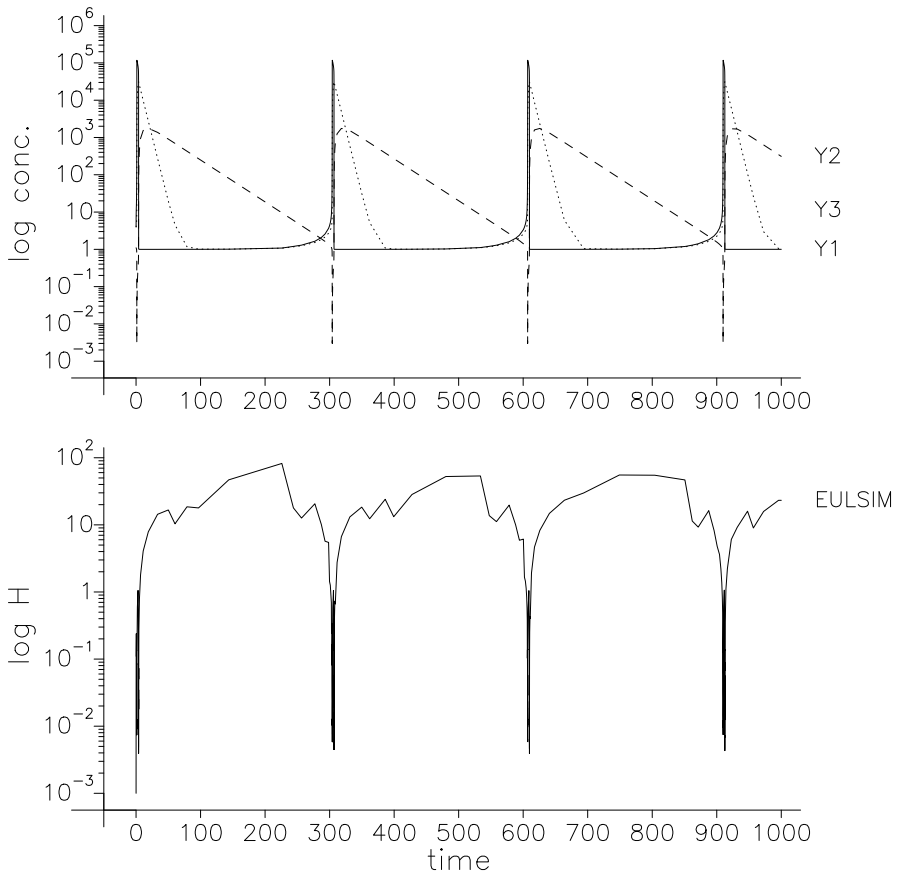


Figure 3.1: Solution and stepsize for the *Oregonator* problem

LU-decompositions) and *nsol* (number of forward/backward substitutions). Note that the count *nfcn* includes also the function evaluations required for the internal Jacobian generation by a finite differences approximation. The integration interval is now $[0, 300]$ and the required relative accuracy was set to $rtol=10^{-3}$ and $rtol=10^{-5}$ respectively. The internal scaling of DYSEUL was forced to ensure a relative error check for all components over the whole integration interval.

The results of this experiment, summarized in Table 3.1 and Table 3.2, are very satisfactory. Up to a value $\sigma = 0.25$ the runs with the dynamic sparsing option switched on show a good coincidence with the standard EULSIM run, i.e. DYSEUL with $\sigma = 0$. Even for values of σ, where the stability of the discretization may degenerate, the problem is solved correctly – but with smaller stepsizes and an increased amount of work. This increase of *nstep* nicely agrees with the theoretical result (2.21).

The last two columns of the tables contain the results of an additional experiment. Using only the diagonal of the Jacobian, indicated by $\hat{A}$=D, the problem is still solved – but extremely inefficient. A pure explicit integration with the code EULEX ($\hat{A} = 0$) fails, as the internal maximum step number bound $nstep_{max} = 20000$ is met.

σ	0	10^{-2}	10^{-1}	0.25	0.5	1	10	100	$\hat{A}$=D	$\hat{A}$=0
nstep	127	129	126	131	168	290	387	fail	5733	fail
nfcn	1761	1730	1671	1715	1930	2608	3301	-	32401	-
ndec	684	676	658	685	779	1188	1524	-	17539	-
nsol	2093	2046	1980	2039	2232	3004	3777	-	31055	-

Table 3.1: Dynamic sparsing for different values of σ ($rtol$=10^{-3}).

σ	0	10^{-2}	10^{-1}	0.25	0.5	1	10	100	$\hat{A}$=D	$\hat{A}$=0
nstep	173	173	176	171	214	297	643	fail	15451	fail
nfcn	4033	4070	4065	3862	4273	5527	10515	-	113621	-
ndec	1245	1240	1265	1192	1378	1851	3951	-	49235	-
nsol	4780	4821	4840	4571	5038	6532	12745	-	117237	-

Table 3.2: Dynamic sparsing for different values of σ ($rtol$=10^{-5}).

In the second picture of Figure 3.2, for the standard value $\sigma = 0.25$, the internally selected stepsizes $H_{i,i=1,..,nstep}$ of DYSEUL are compared with the series H_i of EULSIM. Again, the first picture of Figure 3.2 shows the 3 solution components. In addition, the number of nonzero elements of the sparsed Jacobian ($nne(\hat{A})$) is plotted in the last picture of Figure 3.2. Obviously, the behavior of the H_i and of $nne(\hat{A}_i)$ is nicely coupled. Large H_i and $nne(\hat{A}_i)$ in the stiff, smooth phase, smaller steps and less nonzero elements in the dynamic, transient phase, where, perhaps, an explicit discretization may be used.

In order to study the behavior during this phase in more detail, Figure 3.3 shows the magnified region [0,5] of Figure 3.2. Now one can see, that the transient phase consists of two sharp transient regions, where an explicit method may work efficiently. But these non–stiff regions are separated by a small stiff region, where an explicit method would waste computing time.

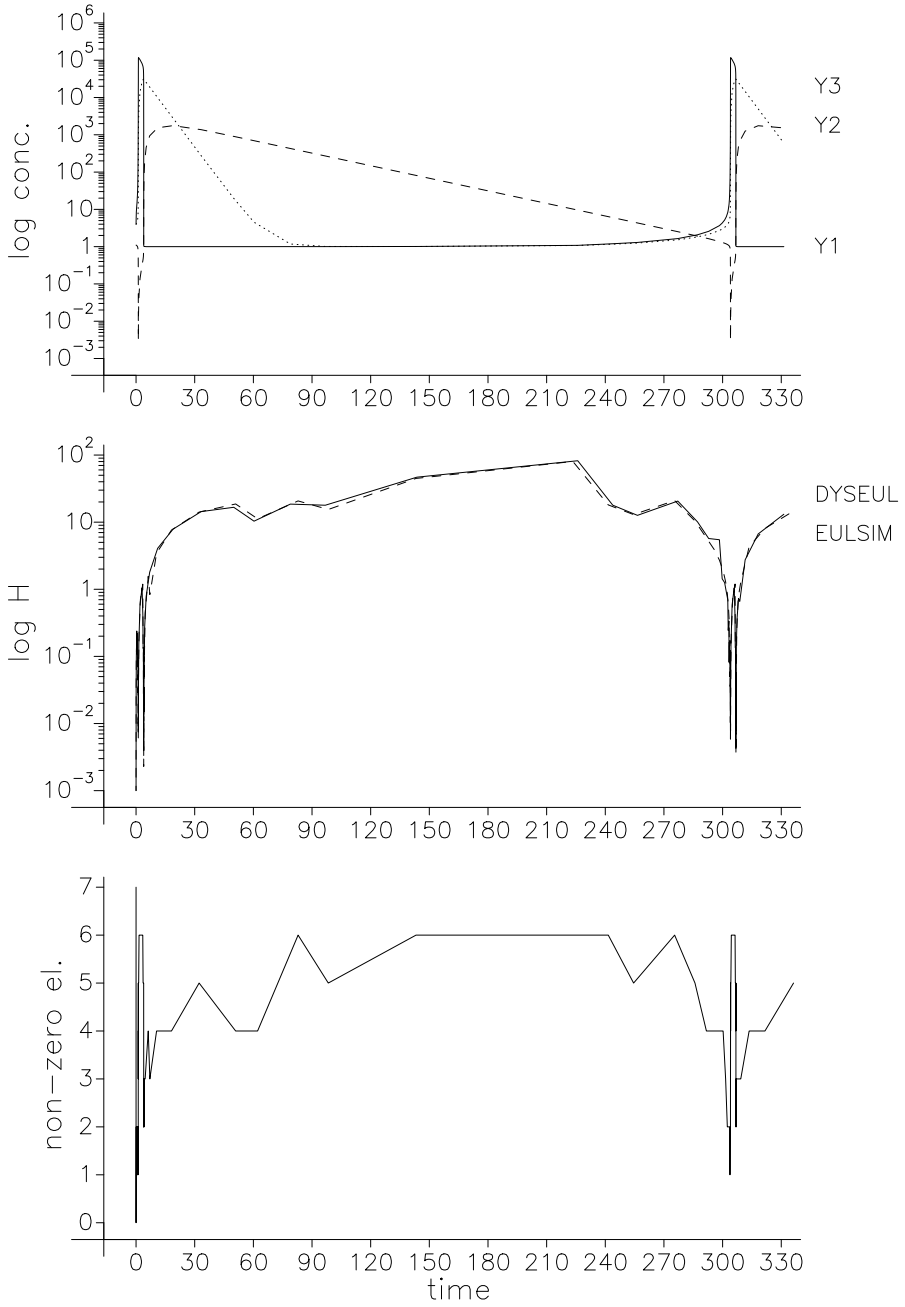


Figure 3.2: Solution, stepsizes and $nne(\hat{A})$ for the *Oregonator* (first zoom)

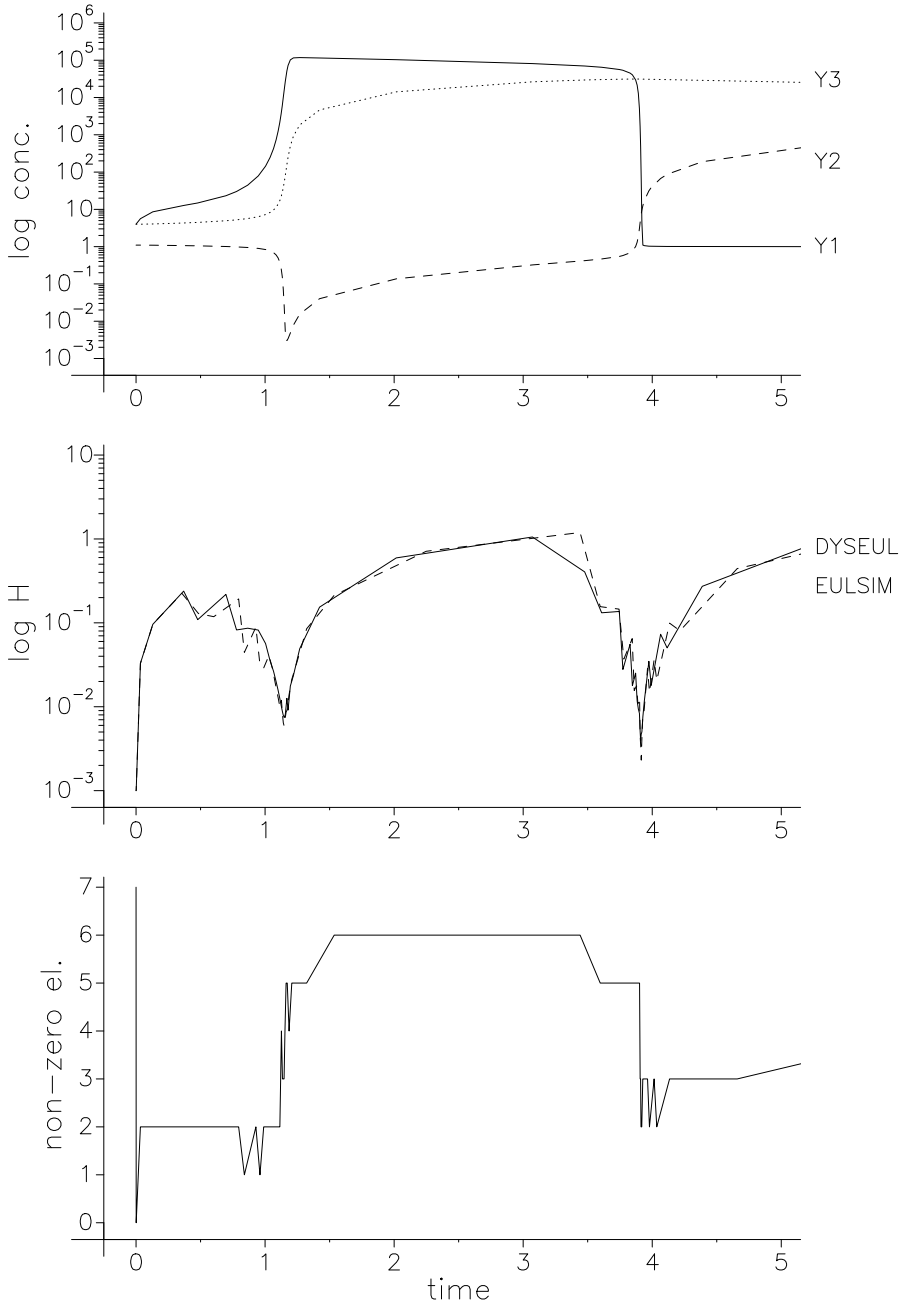


Figure 3.3: Solution, stepsize and $nne(\hat{A})$ for the *Oregonator* (second zoom)

Thus, a type switching method should use the stiff integration method not only in the large smooth region but also for the just mentioned small stiff region, see e.g [26]. Gratifyingly, this small local stiff region is recognized by the sparsing procedure also. The number of remaining nonzero elements in the sparsed Jacobian increases locally. Large time steps are possible and are not prevented by a too insensitive sparsing criterion.

The fact, that the dynamic sparsing procedure works very satisfyingly on this nontrivial test problem is encouraging. But, in order to exploit the reduced number of nonzero elements in the Jacobian, special linear algebra routines are necessary. For an example of dimension $n = 3$ this certainly will not pay off.

Quite interesting is the result of another experiment. Running the type switching code LSODA due to PETZOLD/HINDMARSH [19, 22] on the *Oregonator* problem shows that the code works fastest, if the switching to the nonstiff Adams method is inhibited. Likewise, no gain is reported for the Runge–Kutta based switching method developed by SOTTAS [26].

Large Problems from Chemical Kinetics

A quite extensive and interesting problem class are large systems of stiff ODEs coming from a detailed modeling of chemical reaction systems. For the numerical simulation of such systems, the program package LARKIN [3] turned out to be a powerful tool. Now, one may compare the efficiency of solving some of these problems with the unchanged integrators of LARKIN (special versions of the semi–implicit Euler method and the semi–implicit midpoint rule) with variants, where the sparsing procedure is added. Three examples are considered:

- $n - hexane$[21] $n = 59$ $nne(A) = 523$ $n^2 = 3481$
- *soot*[16] $n = 138$ $nne(A) = 2880$ $n^2 = 19044$
- RNA[27] $n = 352$ $nne(A) = 2642$ $n^2 = 123904$

Recall that dynamic sparsing means to change the nonzero pattern of the Jacobian within the course of the integration. Thus, one has an a priori loss of efficiency for the sparse linear system solution. Testing again for different required tolerances and various values for σ, the distinct reliability of the sparsing algorithm is the most important result. Once more, up to $\sigma = 1$ only minor changes in the integration behavior can be observed. Regarding the efficiency, the results are summarized quickly. The number of elements which can be removed is not dramatic but sufficient to balance, at least, the a priori loss of efficiency (for $\sigma = 0.25$). For the example *soot*, which has the comparatively densest Jacobian, the overall computing time is reduced by 15% .

The Epidemic Spread of AIDS

A detailed model for the epidemic spread of the HIV/AIDS disease, including the dynamics of the development of the disease, the population size, the gender and age structure, has been derived by J.Weyer and B.Ch.Schmidt. Their fine modeling leads to a set of

$$n = 1650$$

nonlinearly coupled equations. Depending on the selected values for the medical and sociological parameters, the system can become very stiff. A description of the model, the numerical solution and the outcome of some simulations can be found in SCHMIDT/WEYER/DEUFLHARD/NOWAK/PÖHLE [28].

Analyzing the Jacobian A of the system, the number of structural nonzero elements turns out to be

$$nne(A) = 128268 \quad (n^2 = 2,722,500).$$

Hence, the matrix cannot be considered as extremely sparse. Consequently, one may first study the general question whether sparse matrix techniques will still pay off on a vector machine. So, as a first experiment, the problem has been solved with the standard full mode code EULSIM and with the general sparse mode code EULSIS on a CRAY machine. The advantage of using sparse matrix techniques is evident. Although the full mode linear system solver vectorizes nearly optimal, the (badly vectorizing) sparse mode solution is drastically faster. These results, as well as all other test results for the present test problem, are put together in Table 3.3.

Looking into more details, one can see that for the solution with EULSIS the portion of CPU time which is required for the linear system solution is still not in balance with the portion for the function and Jacobian evaluation. Furthermore, the amount of work for the matrix decompositions dominates the one for the forward/backward substitutions too strongly.

Now, solving the problem with dynamic sparsing (code SEULSP, $\sigma = 0.25$) yields another drastic speed up factor. The results are arranged in Table 3.3. Most of the elements of the Jacobian can be dropped. The resulting matrix $\hat{A}$ is extremely sparse, i.e.

$$nne(\hat{A}) < 8000$$

hold for the whole integration. This dramatic sparsing effect is illustrated in Figure 3.4. Again, the choice of the specific value for σ is not critical. Confirming the experience of the experiments presented so far, the performance of the integrator is just slightly changed. The different portions of the computing time are now nicely balanced. Rerunning the problem on a

	EULSIM (CRAY)	EULSIS (CRAY)	SEULSP (CRAY)	SEULSP (SUN)	SEULSP-V (SUN)
nstep	11	11	10	10	10
nfcn	70	70	72	72	72
njac	10	10	9	9	9
ndec	41	41	39	39	39
nsol	110	110	110	110	110
CPU(s)	734.1	67.3	6.2	45.2	30.1
Storage(MB)	46.2	9.5	6.5	5.5	3.0
f-time	0.1 %	0.5 %	6.0 %	10.9%	16.3 %
J-time	0.3 %	4.5 %	45.7 %	55.4%	35.9 %
Dec-time	99.2 %	92.2 %	29.1 %	13.8%	20.6 %
Sol-time	0.3 %	1.8 %	16.1 %	7.8%	11.6 %
other-time	0.1 %	1.0 %	3.1 %	12.1%	15.6 %

Table 3.3: Performance of EULSIM variants for the *Epidemic* problem.

workstation (SUN SPARC1+), yields another load balance. This is due to the fact, that the loss of vectorization shows up mainly in the code for the evaluation of the functions and Jacobians, respectively. Now, the costs for the Jacobian evaluations are dominant.

As pointed out in the preceding section, one may combine the sparsing procedure and the Jacobian generation, mainly to save storage. For the present problem a skillful matching saves, furthermore, about 30% computing time. The results for this variant are shown in the last column of Table 3.3.

A final question to be discussed is, whether the problem can be solved efficiently with an explicit integrator. With the current parameter values and model assumptions of the underlying epidemic model, the problem turns out to be only mildly stiff. A simulation with EULEX requires more steps and function evaluations but the overall computing time is in the same range as for SEULSP. Though, using other parameter values the system may become extremely stiff. An explicit integration is impossible, whereas the behavior of SEULSP remains nearly unchanged. The stepsizes of SEULSP and EULEX for such a stiff problem are plotted in Figure 3.5. This picture indicates that a type switching method would not help, as the stiffness increases with time.

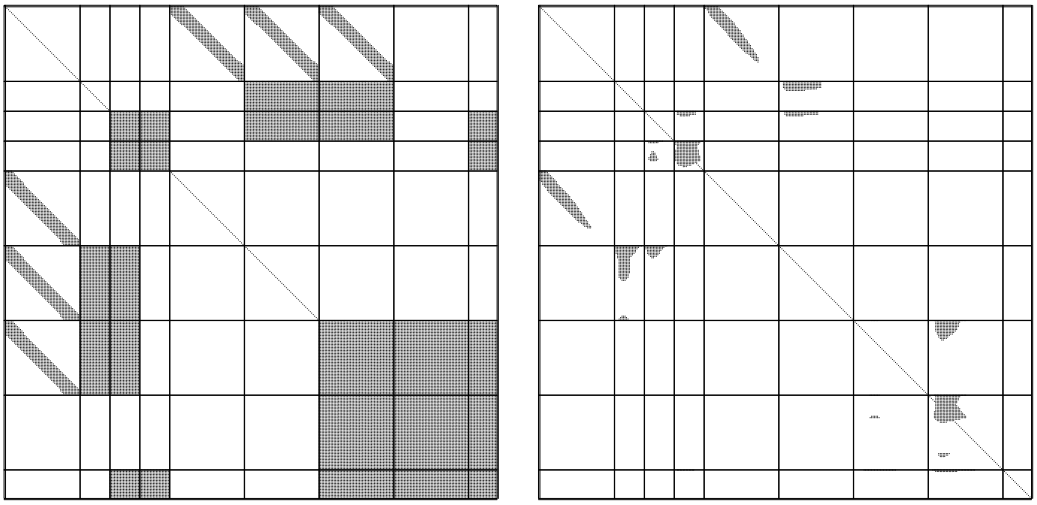


Figure 3.4: Original and sparsed Jacobian for the *Epidemic* problem

Figure 3.5: Performance of EULEX and SEULSP for the *Epidemic* problem

Conclusion

The dynamic sparsing algorithm for semi–implicit extrapolation methods turns out to be a theoretically backed and robust device. It can easily be inserted into existing software. Depending on the problem at hand, the gain in efficiency can be drastic, whereas the potential loss is small. The latter fact is mainly subject to the realization of the sparse linear algebra method, as the computational costs for the sparsing algorithm itself are neglectable. But the efficient application of the dynamic sparsing procedure is not restricted to ODE solvers with sparse linear algebra mode. Especially for very large scale problems, where the linear system solution is done with an iterative method, the benefit from an a priori sparsing of the Jacobian may be substantial.

Acknowledgments

The author is deeply indebted to P. Deuflhard for stimulating and encouraging this work. Thanks to B. Weyer and B.Ch. Schmidt whose model motivated me to develope the described techniques. Special thanks to U. Pöhle for his computational assistance.

Bibliography

[1] R. Aiken (ed.): *Stiff Computation*. Oxford Univerty Press (1985)

[2] G. Bader, P. Deuflhard: *A Semi–Implicit Mid–Point Rule for Stiff Systems of Ordinary Differential Equations*. Numerische Mathematik **41**, p. 373–398 (1983)

[3] G. Bader, U. Nowak, P. Deuflhard: *An Advanced Simulation Package for Large Chemical Reaction Systems*. In [1], p. 255–264

[4] A. Björck: *Some Methods for Separating Stiff Components in Initial Value Problems*. In: Numerical Analysis, Dundee, D. F. Griffiths (ed.), Lecture Notes in Math. **1066**, Springer, p. 30–43 (1983)

[5] C.J. Butcher: *Order, Stepsize and Stiffness Switching*. Computing **44**, p. 209–220 (1990)

[6] P. Deuflhard: *Order and Stepsize Control in Extrapolation Methods*. Numer. Math. **41**, p. 399–422 (1983)

[7] P. Deuflhard: *Recent Progress in Extrapolation Methods for ODE's*. SIAM Review **27**, p. 505–535 (1985)

[8] P. Deuflhard: *Uniqueness Theorems for Stiff ODE Initial Value Problems*. In: Numerical Analysis, Dundee 1989, D.F. Griffith, G.A. Watson (eds.), Pitman Research Notes in Mathematics Series 288, p. 74–88 (1989)

[9] P. Deuflhard, E. Hairer, J. Zugck: *One–step and Extrapolation Methods for Differential – Algebraic Systems*. Num. Math. **51** p. 501–516 (1987)

[10] P. Deuflhard, U. Nowak: *Extrapolation Integrators for Quasilinear Implicit ODE's*. In: P. Deuflhard, B. Enquist (eds.): Large Scale Scientific Computing. Progress in Scientific Computing, Birkhaeuser **7**, p. 37–50 (1987)

[11] I.S. Duff: *MA28 – A Set of Fortran Subroutines for Sparse Unsymmetric Linear Equations*. AERE Report **R. 8730**, HMSO, London (1977)

[12] I.S. Duff: *Direct Methods for Solving Sparse Systems of Linear Equations*. SIAM J. Stat. Comput. **5**, p. 605–619 (1982)

[13] I.S. Duff, U. Nowak: *On Sparse Solvers in a Stiff Integrator of Extrapolation Type*. IMA Journal of Numerical Analysis **7**, p. 391–405 (1987)

[14] W.H. Enright, T.E. Hull, B. Lindberg: *Comparing Numerical Methods for Stiff Systems of ODE's*. BIT **15**, p. 10–48 (1975)

[15] R.J. Field, R.M. Noyes: *Oscillations in Chemical Systems. IV. Limit Cycle Behavior in a Model of a Real Chemical Reaction*. J. Chem. Phys. **60**, p. 1877–1884 (1974)

[16] M. Frenklach et al.: *Mechanism of Soot Formation in Acetylene–Oxygene Mixtures*. In: Combust. Sci. Technol. (1986)

[17] C.W. Gear, Y. Saad: *Iterative Solution of Linear Equations in ODE Codes*. SIAM J. Sci. Stat. Comput. **4**, p. 583–601 (1983)

[18] E. Hairer: *Order conditions for Numerical Methods for Partitioned Ordinary Differential Equations*. Numer. Math. **36**, p. 431–445 (1981)

[19] A.C. Hindmarsh: *ODEPACK, A Systematized Collection of ODE Solvers*. In Scientific Computing, R. S. Stepleman et al. (eds.), North–Holland, Amsterdam, p. 55–64 (1983)

[20] E. Hofer: *A Partially Implicit Method for Large Stiff Systems of ODE's with only Few Equations Introducing Small Time–Constants*. SIAM J. Numer. Anal. **9**, p. 603–637 (1976)

[21] G. Isbarn, H.J. Ederer, K.H. Ebert: *The Thermal Decomposition of n–Hexane: Kinetics, Mechanism and Simulation*. In: K.H. Ebert, P. Deuflhard, W. Jäger (eds.): Modelling of Chemical Reaction Systems. Springer Series in Chemical Physics **18**, p. 235–248 (1981)

[22] L. Petzold: *Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations*. SIAM J. Sci. Stat. Comput. **4**, p. 136–148 (1983)

[23] P. Rentrop: *Partitioned Runge–Kutta Methods with Stiffness Detection and Stepsize Control*. Numerische Mathematik **47**, p. 545–564 (1985)

[24] W.D. Seider, C. W. White III, G. J. Prokopakis: *Chemical Reaction Systems*. In [1], p. 48–52

[25] L.F. Shampine: *Implementation of Rosenbrock Methods*. ACM Trans. Math. Soft. **8**, p. 93–113 (1982)

[26] G. Sottas: *Dynamic Adaptive Selection between Explicit and Implicit Methods when Solving ODE's*. Report, Sect. de Math., Univ. Geneve (1984)

[27] F.W. Schneider, M. Heinrichs, T. Poll: *Private communication*. (1980)

[28] B.Ch. Schmidt, J. Weyer, P. Deuflhard, U. Nowak, U. Pöhle: *Die Ausbreitung von HIV/AIDS in Ballungsgebieten.* Technical Report TR 91–9, Konrad-Zuse-Zentrum Berlin (1991)

[29] T. Steihaug, A. Wolfbrandt: *An Attempt to Avoid Exact Jacobian and Nonlinear Equations in the Numerical Solution of Stiff Differential Equations.* Math. Comp. **33**, p. 521–534 (1979)

[30] K. Strehmel, R. Weiner: *Behandlung steifer Anfangswertprobleme gewöhnlicher Differentialgleichungen mit adaptiven Runge–Kutta Methoden.* Computing **29**, p. 153–165 (1982)

[31] D.S. Watkins, R.W. HansonSmith: *The Numerical Solution of Separably Stiff Systems by Precise Partitioning.* ACM Trans. Math. Soft. **9**, p. 293–301 (1983)