ZAMM · Z. Angew. Math. Mech. **83**, No. 10, 637 – 647 (2003) / **DOI** 10.1002/zamm.200310070

Sparsing in real time simulation

Anton Schiela[1,2,*] **and Folkmar Bornemann**[1]

[1] TU München, 85747 Garching, Germany
[2] DLR Oberpfaffenhofen, 82230 Weßling, Germany

Published online 10 September 2003

Key words stiffness, ODE, DAE, sparsing, partitioning
MSC (2000) 65L20

Modelling of mechatronical systems often leads to large DAEs with stiff components. In real time simulation neither implicit nor explicit methods can cope with such systems in an efficient way: explicit methods have to employ too small steps and implicit methods have to solve too large systems of equations. A solution of this general problem is to use a method that allows manipulations of the Jacobian by computing only those parts that are necessary for the stability of the method. Specifically, manipulation by sparsing aims at zeroing out certain elements of the Jacobian leading to a structure that can be exploited using sparse matrix techniques. The elements to be neglected are chosen by an a priori analysis phase that can be accomplished before the real-time simulation starts. In this article a sparsing criterion for the linearly implicit Euler method is derived that is based on block diagonalization and matrix perturbation theory.

1 Introduction

Real time simulation is a growing field of applications for simulation software. Especially the so called "Hardware in the Loop" (HIL) simulation is an application that becomes more and more important, e.g., in controller design and testing. In such a scenario a piece of hardware (e.g. a controller) is set into a virtual environment, created by a simulation that is running in real time. This controller communicates with the simulation software in short time cycles, so that the software is required to provide results once in a cycle.

Using new modelling techniques, such as object oriented modelling, it is possible to describe more and more complex technical models (see e.g. [4,9,10]). Many of those are multi-domain models, which means that they contain components from more than one physical domain. Mechanic, electric, hydraulic, or thermodynamic components are often coupled together in one model. This means that there are several different time scales present in the model, which again leads to stiff ordinary differential equations (ODEs) or differential algebraic equations (DAEs).

Efficient real time simulation in such a stiff case poses problems very different from off-line simulation. Classical methodology of efficient algorithms, such as step size control or the reuse of the Jacobian, is not feasible in the context of real time simulation.

This paper covers one possible form of adaptivity for real time simulation: sparsing, which means that certain elements of the Jacobian matrix are set to zero to permit more efficient LU-decompositions. In Sect. 2 the linearly implicit Euler method is discussed in the light of real time simulation and sparsing. In Sect. 3 a preprocessing routine analyzing a model and performing the sparsing is described. Results of numerical experiments are presented in Sect. 4.

2 Linearly implicit Euler and sparsing

2.1 Stiff real time simulation

We consider the numerical solution of a quasi-linear differential algebraic initial value problem of index 1:

$$L\dot{x} = f(x,t); \qquad x(t_0) = x_0. \tag{1}$$

This DAE is a model for a dynamic process that starts at model time t_0 and runs for a certain amount of time δt. For a given time grid $\Delta = \{t_0, t_1, \ldots, t_N = t_0 + \delta t\}$ on the interval $[t_0, t_0 + \delta t]$ and for consistent initial values x_0 we would like to obtain a numerical approximation $\{x_0, x_1, \ldots, x_N\}$ of the solution of (1) on Δ. For this purpose we use an algorithm that starts to run on a computer at real time T_0. The special feature of "real time simulation" (in an abstract sense) is that for each grid point t_n there is a specified "deadline" δT_n, a time span in "real time", and we require that the computation of x_n is complete at the time $T_0 + \delta T_n$.

"Hardware in the Loop simulation" (in an abstract sense) requires that for each grid point t_n there is a time span δS_n such that $f(x_n, t_n)$ cannot be evaluated by the algorithm before the time $T_0 + \delta S_n$. This means that for given x_n the computation

* Corresponding author, e-mail: Anton.Schiela@web.de

of the next solution point x_{n+1} cannot start before $T = T_0 + \delta S_n$ and is required to take no longer than the time span $D_n = \delta T_{n+1} - \delta S_n$. If this "real time" requirement is not met at each time instant the simulation fails.

In most applications we have an equidistant time grid Δ with a constant "sample time" $\tau = t_1 - t_0 = \ldots = t_N - t_{N-1}$ and a constant "deadline" $\delta = D_0 = \ldots = D_{N-1}$. Often τ is so small that it is also used as the step size for the numerical integration scheme.

In short we observe two characteristic features of real time simulation: the required time grid is part of the specification of the problem and the computing time is tied closely to the temporal behavior of the simulated model.

Appropriate numerical methods. To perform real time simulation in a reliable way we have to use numerical methods that perform predictable computational work for each step. This requirement rules out implicit methods (they iteratively solve a non-linear system of equations) and leads us to explicit methods in the non-stiff ODE case and to linearly implicit methods in the stiff or differential algebraic case. Taking into account that the grid interval τ is usually very small, we concentrate on low order methods like the explicit Euler method,

$$x_{n+1} = x_n + \tau f(x_n, t_n), \tag{2}$$

or the linearly implicit Euler method,

$$x_{n+1} = x_n + \tau (L - \tau Df|_{(x_n,t_n)})^{-1} f(x_n, t_n). \tag{3}$$

In the following, we will concentrate on the stiff and differential algebraic case, so we will deal with the linearly implicit Euler method.

Adaptivity. Classically, adaptivity adjusts the behavior of the algorithm while the simulation is running. Two examples are step size control and the reuse of the Jacobian. This is done to make the algorithm more reliable (in terms of error control) and more efficient (in terms of overall computation time).

In real time simulation we do not have the possibility to change the step size because it is given or severely restricted by the real time specifications of the problem. The best thing we can do for "controlling" the error is to estimate it and log the estimates so that the value of the results can be judged a-posteriori.

Concerning efficiency we observe that in real time simulations performance is measured in a special way. For a given step size τ the performance of the method on a computer is indicated by the minimal deadline $\delta_{\min}$ in which each integration step can be accomplished. This means that the most expensive step or in other words the "worst case" determines $\delta_{\min}$. As a consequence, techniques like reusing the Jacobian over several steps are not useful in the real time case because the most expensive steps – the ones where the Jacobian is updated – determine the performance.

So, if we want to design an adaptive algorithm, we have to gather information before real time simulation starts and then use a form of adaptivity that relies on time independent properties of the problem. This requires an analysis of the problem before real time simulation starts. The goal of this paper is to describe one way to perform such an analysis automatically during a preprocessing phase.

As the overall computation time is secondary in real time simulations, such a strategy makes sense even if the preprocessing phase is much more expensive than the simulation itself.

Stiffness. Real time simulation is mostly performed in industrial applications. There, the systems that have to be simulated are often modelled by large differential algebraic equation systems with stiff components. In most cases, stiffness is caused by subsystems with fast dynamics (such as controllers or electric circuits) compared to the time scales of interest (e.g. the movement of mechanical parts). Systems with such a structure are sometimes called "separably stiff" (see e.g. [6].). There are various ways to exploit this structure in off line simulation (see e.g. [6], [13]). However, these methods are not suitable for real time simulations.

On the other hand, we observe that this structure inherent to the model is often time invariant. Therefore we can also exploit this structure in the real time case. Moreover, issues as stability of the numerical discretization are easy to predict because the step size is kept constant.

Sparsing. The linearly implicit Euler method for stiff simulations consumes a large part of computation time by the decomposition of the matrix $(L - \tau Df)$. If the Jacobian is large and sparse this effort can be reduced by the use of direct sparse solvers (see e.g. Duff and Nowak [2]): the more sparse the matrix the faster the factorization of the matrix.

This leads to the idea of zeroing selected elements of the Jacobian, such that – while the integration method is still stable – the matrix factorization is performed more efficiently.

An example where sparsing was performed successfully in off line simulation is described by Nowak [8]. Here the matrix elements were zeroed out dynamically before each time step using a criterion based on the magnitude of the matrix elements.

In our approach for the real time case the Jacobian $Df|_{(x_n,t_n)}$ is analyzed for several (x_n, t_n) and a fixed sparsity pattern is selected during a preprocessing phase. During simulation only the remaining elements will be taken into account by the factorization routine. The factorization routine itself also has to meet real time requirements. We will therefore use only the time independent structural features of the linear system that do not affect the stability of the factorization.

For a successful application of sparsing we have to deal with two questions: How does the approximation of the Jacobian affect the stability of the integration method? How is the accuracy of the method affected by sparsing? In the rest of this section these two questions are answered for the linearly implicit Euler method.

2.2 Linearly implicit Euler with inexact Jacobian

As the simplest linearly implicit method the linearly implicit Euler method is a good candidate for sparsing. We are going to analyze this method for the case of an inexact Jacobian, i.e., a matrix $J \approx Df$ that is used in the discretization:

$$x_{n+1} = x_n + \tau(L - \tau J)^{-1} f(x_n, t_n). \tag{4}$$

The difference between the exact and the approximate Jacobian is denoted by $\Delta J := Df - J$.

Accuracy. In general the use of an inexact Jacobian for linearly implicit methods leads to a loss of order in the method. However one can construct methods without loss of order, called W-methods. They were first studied by Steihaug and Wolfbrandt [11]. The linearly implicit Euler method is just the simplest example of a W-method. We can verify this by computing the error expansion. However, order is only an asymptotic concept. If we consider the linearly implicit Euler method at a given step size τ, modifying the Jacobian of course changes the dynamics of the numerical method.

The conclusion concerning accuracy is therefore: sparsing has only a small effect if sufficiently small step sizes are used.

Stability. Linearly implicit methods solve linear systems of equations for the sake of stability. Stability is the property of the method most severely affected by sparsing.

It is well known that stability of a linear difference equation

$$x_{n+1} = Ax_n \tag{5}$$

is determined by the magnitude of the eigenvalues λ of A. Stability requires that all eigenvalues lie on the unit disc: $|\lambda| \leq 1$ and that all eigenvalues on the border of the disc are algebraically simple (see e.g. [1]). This result can be generalized to

$$Bx_{n+1} = Ax_n. \tag{6}$$

If B is invertible then the same requirement does apply for the generalized eigenvalues λ of the "matrix pair" (A, B):

$$Av - \lambda Bv = 0. \tag{7}$$

In the case of the linearly implicit Euler method with inexact Jacobian applied to a linearized DAE

$$L\dot{x} = Df|_{x^*} x \tag{8}$$

we obtain

$$(L - \tau J)x_{n+1} = (L - \tau J)x_n + \tau Df x_n = (L + \tau \Delta J)x_n. \tag{9}$$

So instead of the eigenvalue problem for an exact Jacobian

$$Lv - \lambda(L - \tau Df)v = 0, \tag{10}$$

we have to deal with

$$(L + \tau\Delta J)v - \lambda(L - \tau Df + \tau\Delta J)v = 0. \tag{11}$$

It is a standard result of numerical analysis that for stable linear DAEs of index 1 the linearly implicit Euler method with an exact Jacobian is also stable regardless of the step size τ (A-stability). For an inexact Jacobian this property has to be checked. If ΔJ is introduced deliberately, this has to be done in such a way that stability is preserved.

3 A sparsing criterion

For appropriate sparsity structure of the Jacobian matrix we have to analyze within a preprocessing routine the effects of sparsing on the dynamical behavior of the numerical method. The core of such a routine is a cheap sparsing criterion that estimates those effects for each non-zero element of the Jacobian.

In the off-line case it did pay off to use a very simple criterion based on the magnitude of the entries, applying it dynamically during the simulation run [8]. However, in the real time case a more accurate, more expensive criterion seems to be more appropriate. This criterion can then be applied during a preprocessing phase before the simulation starts. This preprocessing phase yields a sparsity structure that remains fixed during the entire simulation.

As we have seen in Sect. 2, stability of the numerical integration scheme applied to a linear problem $L\dot{x} = Dfx$ depends on the eigenvalues of the discrete evolution of the linearly implicit Euler method, i.e. the matrix pair

$$(L + \tau\Delta J, L - \tau Df + \tau\Delta J). \tag{12}$$

If $|\lambda| < 1$ for all eigenvalues λ of (12) then the discrete evolution is asymptotically stable.

For small perturbation matrices ΔJ it is feasible to estimate the changes of the eigenvalues with the help of linear perturbation theory. For this purpose the system is first transformed to block diagonal form and then the change of eigenvalues is estimated for each block.

3.1 Simultaneous block diagonalization of matrix pairs

In the following section a short overview is given about the numerical tools that are used to calculate the sparsing criterion. Our approach leads to the generalized eigenvalue problem (7). A detailed discussion of the theoretical and numerical issues of this generalized eigenvalue problem can be found in [12] or [5]. In our application we can assume that B is invertible, simplifying the discussion. This is because we are dealing with index 1 problems only.

The goal of this section is to simultaneously transform A and B to block diagonal form with blocks that are upper triangular.

Triangularization. The most important numerical tool used for the solution of the generalized eigenvalue problem is the QZ-algorithm, an adaptation of the QR-Algorithm to obtain the generalized Schur form.

For a given regular matrix pair (A, B) the QZ-Algorithm performs orthogonal transformations Q and Z such that

$$QAZ = A_T, \qquad QBZ = B_T, \tag{13}$$

where A_T and B_T are both upper triangular matrices. Information about the eigenvalues is given by the diagonal elements α_{ii} and β_{ii} of A_T and B_T. If $\beta_{ii} \neq 0$ (true if B is non-singular) then $\lambda_i = \alpha_{ii}/\beta_{ii}$.

Block diagonalization. After the eigenvalues were computed, the next step would be to calculate the eigenvectors, as they contain information about the behavior of the eigenvalues under perturbations. Unfortunately, for non-symmetric matrix pairs the eigenvector problem is in general badly conditioned. Especially if eigenvalues are close together, eigenvector sensitivity becomes large (see again [12] or [5]).

In technical applications multiple eigenvalues and Jordan blocks are very common. Moreover, algebraic equations lead to a multiple eigenvalue 0 with sometimes very high multiplicity. Hence at least some eigenvectors cannot be computed reliably in real life problems. To construct a robust algorithm we have to be satisfied with less: To given disjunct subsets of eigenvalues (with well conditioned corresponding eigenspaces) compute right and left orthogonal bases. The unions of these bases yield two transformation matrices X and Y, such that

$$Y^H AX = A_B, \qquad Y^H BX = B_B, \tag{14}$$

where A_B and B_B have the same block diagonal structure and the column subsets of X and Y corresponding to the blocks are sets of orthonormal vectors. This kind of factorization is sometimes called "spectral resolution" (see e.g. [12]).

If we assume without loss of generality that there are only two blocks, (14) has the following structure:

$$\begin{aligned} \begin{pmatrix} Y_1^H \\ Y_2^H \end{pmatrix} A \begin{pmatrix} X_1 & X_2 \end{pmatrix} &= \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}, \\ \begin{pmatrix} Y_1^H \\ Y_2^H \end{pmatrix} B \begin{pmatrix} X_1 & X_2 \end{pmatrix} &= \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}. \end{aligned} \tag{15}$$

The standard approach to obtain a block diagonal form is via solving a Sylvester equation (see [12]). This however leads to non-orthogonal bases X_k and Y_k, which may distort the sparsing criterion.

An algorithm for block diagonalization, that preserves orthogonality inside the subsets, is based on the observation that the first i columns of Z are orthogonal bases for the eigenspaces corresponding to the first i eigenvalues, if they are ordered by appearance on the diagonal of the triangular matrices A_T and B_T. The same holds correspondingly for the last i rows of Q. To construct orthogonal bases for a given set of eigenvalues we only have to sort them to the top of the diagonal or to the bottom, respectively. This can be done by a sequence of Givens rotations in a way similar for the standard Schur form (see [5]). If this is done for all given subsets, we get the desired transfomation matrices X and Y^H. This algorithm is slower in performance than block diagonalization by solving the Sylvester equation, but still $O(n^3)$ in complexity.

3.2 Results from perturbation theory

In this section the theoretical basis of the sparsing criterion is derived. We study the behavior of the eigenvalues of a matrix pair (A, B), if a small perturbation (E, F) is applied. For the theoretical foundation of the following we mainly refer to the book of Stewart and Sun [12]. In contrast to the results presented there, our main goal is to achieve first order approximations of the eigenvalue perturbations rather than bounds. This is because a sparsing criterion compares elements with each other rather than estimates a worst case.

As a starting point of our considerations we use a very general result of [12], that contains all the information we will need for our purpose. As we only need a weaker result, we state a simplified formulation.

Theorem 1. (Perturbations of block diagonal matrix pairs) *Let the regular matrix pair (A, B) have a spectral resolution* (15). *Given a sufficiently small $O(\varepsilon)$ perturbation (E, F), let*

$$\begin{aligned} \begin{pmatrix} Y_1^H \\ Y_2^H \end{pmatrix} E \begin{pmatrix} X_1 & X_2 \end{pmatrix} &= \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix}, \\ \begin{pmatrix} Y_1^H \\ Y_2^H \end{pmatrix} F \begin{pmatrix} X_1 & X_2 \end{pmatrix} &= \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix}. \end{aligned} \tag{16}$$

Then there is also a spectral resolution of $(\tilde{A}, \tilde{B}) = (A + E, B + F)$ with pairs of diagonal blocks

$$(\tilde{A}_1, \tilde{B}_1) = (A_1 + E_{11} + O(\varepsilon^2), B_1 + F_{11} + O(\varepsilon^2)),$$
$$(\tilde{A}_2, \tilde{B}_2) = (A_2 + E_{22} + O(\varepsilon^2), B_2 + F_{22} + O(\varepsilon^2)).$$

Proof. Consider [12] Theorem VI.2.15 that gives explicit perturbation bounds and check that the terms that are not listed here are $O(\varepsilon^2)$. □

Remark 1. The theorem states that the non-diagonal blocks of the transformed perturbation matrix can be neglected up to first order. Dealing with finite perturbations, the question about the quality of this approximation arises.

One obvious issue is the norm of the transformation matrices Y_i^H, X_j. The lower it is, the smaller are the transformed perturbations E_{ij}. This should have an impact on the way of computing these transformations.

Another issue is the relation between the two blocks, an exhaustive analysis of which is beyond the scope of this article. What can be said here is that the closer the eigenvalues of the blocks are, the larger the errors tend to be. This should have an impact on the choice of the blocks.

If we consider only 1×1 blocks we get a result about perturbations of eigenvalues:

Corollary 2. *Consider the diagonalizable matrix pair (A, B) with invertible B and no multiple eigenvalues. Let Y^H and X be the transformation matrices containing the left and right eigenvectors such that $Y^H AX$ and $Y^H BX$ are diagonal matrices. Furthermore consider a sufficiently small $O(\varepsilon)$ perturbation (E, F). Then the diagonal matrix $\tilde{\Lambda}$ of eigenvalues of the perturbed matrix pair $(A + E, B + F)$ can be approximated as follows:*

$$\tilde{\Lambda} = \operatorname{diag}((Y^H(B + F)X)^{-1} Y^H(A + E)X) + O(\varepsilon^2). \tag{17}$$

Proof. From Theorem 1 and the non-singularity of B follows that

$$\tilde{\lambda}_i = \frac{a_i + e_{ii} + O(\varepsilon^2)}{b_i + f_{ii} + O(\varepsilon^2)} = \frac{a_i + e_{ii}}{b_i + f_{ii}} + O(\varepsilon^2),$$

or written in matrix form

$$\tilde{\Lambda} = \operatorname{diag}((Y^H(B + F)X))^{-1} \operatorname{diag}(Y^H(A + E)X)) + O(\varepsilon^2).$$

Lemma 3 below shows that

$$\operatorname{diag}((Y^H(B + F)X))^{-1} \operatorname{diag}(Y^H(A + E)X)) = \operatorname{diag}((Y^H(B + F)X)^{-1} Y^H(A + E)X) + O(\varepsilon^2). \qquad \square$$

Lemma 3. *Let D_1, D_2 be diagonal matrices and E_1, E_2 sufficiently small $O(\varepsilon)$ perturbations. Then we have*

$$\operatorname{diag}((D_1+E_1)^{-1}(D_2+E_2)) = (\operatorname{diag}(D_1+E_1))^{-1}\operatorname{diag}(D_2+E_2) + O(\varepsilon^2). \tag{18}$$

Proof. Without loss of generality assume that E_1, E_2 have zero diagonal. Otherwise the diagonal elements of E_i can be moved to the D_i. Then it remains to show

$$\operatorname{diag}((D_1+E_1)^{-1}(D_2+E_2)) = D_1^{-1}D_2 + O(\varepsilon^2).$$

Neumann expansion of the leftmost factor yields

$$\begin{aligned}\operatorname{diag}((D_1+E_1)^{-1}(D_2+E_2)) &= \operatorname{diag}(D_1^{-1}(I - E_1D_1^{-1} + O(\varepsilon^2))(D_2+E_2)) \\ &= \operatorname{diag}(D_1^{-1}(D_2 - E_1D_1^{-1}D_2 + E_2)) + O(\varepsilon^2).\end{aligned}$$

Due to the fact that E_i multiplied with diagonal matrices still has zero diagonal all the terms containing E_i drop out and the proof is complete. □

With the help of Corollary 2 we can now proof a theorem about symmetric perturbations. It will be the theoretical basis for the sparsing criterion in the next section.

Theorem 4. *Let (A, B) be a diagonalizable matrix pair with eigenvalues λ_i and with left and right eigenvectors y_i and x_i. Assume all eigenvalues to be algebraically simple and B to be invertible. Let E be a sufficiently small $O(\varepsilon)$ perturbation matrix. Let $\tilde{\lambda}_i$ be the eigenvalues of the perturbed matrix pair $(A+E, B+E)$. Then a first order approximation of the sum of the differences between the original and the perturbed eigenvalues is given by the trace of $B^{-1}E(I - B^{-1}A)$:*

$$\sum_i (\tilde{\lambda}_i - \lambda_i) = \operatorname{tr}(B^{-1}E(I - B^{-1}A)) + O(\varepsilon^2). \tag{19}$$

Proof. Corollary 2 and the invariance of the trace under coordinate transformation yield

$$\begin{aligned}\sum(\tilde{\lambda}_i - \lambda_i) &= \operatorname{tr}((Y^H(B+E)X)^{-1}Y^H(A+E)X - (Y^HBX)^{-1}Y^HAX) + O(\varepsilon^2) \\ &= \operatorname{tr}((X^{-1}(B+E)^{-1}(A+E)X - X^{-1}B^{-1}AX) + O(\varepsilon^2) \\ &= \operatorname{tr}((B+E)^{-1}(A+E) - B^{-1}A) + O(\varepsilon^2).\end{aligned}$$

If we linearize this formula using the first order Neumann expansion and sort out higher order terms we obtain

$$\begin{aligned}\sum(\tilde{\lambda}_i - \lambda_i) &= \operatorname{tr}(B^{-1}(I - EB^{-1} + O(\varepsilon^2))(A+E) - B^{-1}A) + O(\varepsilon^2) \\ &= \operatorname{tr}(B^{-1}(A - EB^{-1}A + E + O(\varepsilon^2) - A) + O(\varepsilon^2) \\ &= \operatorname{tr}(B^{-1}E(I - B^{-1}A)) + O(\varepsilon^2).\end{aligned}$$

□

Remark 2. With this theorem we get information about the behavior of the eigenvalues *without* using information about the eigenvectors. Moreover the estimate is invariant under similarity transformations.

The drawback is that if the terms $\tilde{\lambda}_i - \lambda_i$ of the sum cancel out, the sum might be small but the changes in the eigenvalues might be large anyhow.

3.3 Computation of a first order sparsing criterion

For the linearly implicit Euler method the matrix pair under consideration is $(L, L - \tau Df)$. Recall that the matrix $B = L - \tau Df$ is invertible for reasonable step sizes τ. To this matrix pair special perturbations ΔJ_{ij} (zeroing out the matrix element Df_{ij}) are applied symmetrically. So the perturbed matrix pair is $(L + \Delta J_{ij}, L - \tau Df + \Delta J_{ij})$.

We will now use the numerical methods and theoretical results of the last two sections to derive a sparsing criterion that estimates the impact of zeroing out a matrix element on a predefined cluster of eigenvalues.

The computation of the criterion is split into two phases: first a block diagonalization as described in Sect. 3.1 is performed:

$$Y^HLX = A_B, \qquad Y^H(L - \tau Df)X = B_B \tag{20}$$

and we obtain the matrices Y^H, X, A_B, and B_B. Recall that A_B and B_B are block diagonal matrices with upper triangular blocks A_k and B_k. The resulting blocks can now be treated separately which is justified by Theorem 1.

To evaluate eq. (19) for each block we will now compute the matrices on the left and the right of the perturbation matrix. Included are the transformation matrices that lead to the block diagonal form,

$$U^{(k)} = B_k^{-1} Y_k^H, \tag{21}$$

$$V^{(k)} = X_k (I - B_k^{-1} A_k). \tag{22}$$

As B_k is upper triangular this can be done by solving upper triangular matrix equations. Up to now all computations could be performed without knowledge about the perturbation. This means that this $O(n^3)$ computation can be done once for all elements.

The second phase, the computation of a sparsing criterion for each element and each block is a simple matter now and also – as we will show – computationally cheap. To apply Theorem 4 we have to compute

$$c_{ij}^{(k)} = |\operatorname{tr}(U^{(k)} \tau \Delta J_{ij} V^{(k)})| \tag{23}$$

for the k^{th} cluster of, say, m eigenvalues. Because ΔJ_{ij} contains only one non-zero element the matrix whose trace is calculated is a rank 1 matrix:

$$U^{(k)} \Delta J_{ij} V^{(k)} = (U_{j1}^{(k)}, \ldots, U_{jm}^{(k)})^T (-\tau D f_{ij}) (V_{1i}^{(k)}, \ldots, V_{mi}^{(k)}). \tag{24}$$

The trace of such a rank 1 matrix can be computed very cheaply in $O(m)$ operations:

$$\left|\operatorname{tr}(U^{(k)} \tau \Delta J_{ij} V^{(k)})\right| = \left|\tau D f_{ij} \cdot (U_{j1}^{(k)}, \ldots, U_{jm}^{(k)})(V_{1i}^{(k)}, \ldots, V_{mi}^{(k)})^T\right| = \left|\tau D f_{ij} \cdot \sum_{l=1}^{m} U_{jl}^{(k)} V_{li}^{(k)}\right|. \tag{25}$$

3.4 Discussion of the criterion

In Sect. 3.2 some issues concerning approximation errors and the sparsing criterion itself were mentioned briefly and shall be discussed more thoroughly in this section. Some conclusions will be drawn on how to use the degrees of freedom that are still left in the design of the algorithm.

Errors introduced by the block diagonalization. As we have already mentioned our criterion does not take into account higher order terms. Therefore algorithmic decisions should be made in order to keep those terms small. This affects the choice of the blocks and also the algorithm used to transform the problem to block diagonal form.

Concerning the choice of the blocks – or equivalently the clustering of the eigenvalues – there is a tendency that close eigenvalues in different blocks lead to large higher order terms. So blocks should be chosen in such a way that eigenvalues that are grouped together stay together in one block. The existence of such clusters is a property of the model and models with distinct clustering are especially well suited for sparsing.

As we see from Theorem 1 the perturbation matrix E is transformed by Y^H and X and the non-diagonal blocks that are responsible for the higher order terms are neglected. Thus the size of the higher order terms is largely dependent on the transformations Y^H and X. Therefore, we compute Y and X as column wise orthogonal matrices to control the size of those terms.

Cancellation. An important issue that needs to be addressed is the fact that due to cancellation it is possible that $\left|\sum(\tilde{\lambda}_i - \lambda_i)\right| \ll \sum |(\tilde{\lambda}_i - \lambda_i)|$.

This means that it may happen that an element with a strong impact on the eigenvalues receives a small criterion and is classified in the wrong way. This also suggests to use rather small blocks when performing the block diagonalization so that this effect happens less probable.

One conclusion is that, as we deal with real perturbations of real matrices, only the real part of the perturbations is considered by the criterion if an eigenvalue and its conjugate complex counterpart are located inside one block.

However, at least if we consider the stiff eigenvalues, there is an heuristic argument that suggests that the magnitude of the sum is a good estimation for the sum of magnitudes. Sparsing was described as a way of blending implicit and explicit methods such that the result is a numerically stable algorithm. So, the more elements of the Jacobian are deleted, the more will the method behave like an explicit one. Therefore, the stiff eigenvalues will move to the left of the complex half-plane. Therefore, they will all move into the same direction, and so the effects of cancellation might not be so grave. This effect was also observed in numerical experiments. Concerning oscillatory modes, this tendency to move into one direction is not that strong and it might well happen that cancellation plays a role in the choice of an element.

These observations give a hint how to choose the clusters of eigenvalues. For the stiff eigenvalues we can use few clusters but the oscillatory eigenvalues should be divided into as many clusters as possible to avoid cancellation.

One important aspect of cancellation is that it smooths out the large first order estimates of eigenvalues that are very close together because the matrix is a numerically perturbed Jordan matrix. In this case, first order estimates are very large but meaningless, because the higher order terms are large, too. Clearly a sufficiently small perturbation will not cause multiple eigenvalues to jump to infinity as suggested by the first order criterion. To sum up, cancellation makes the criterion more robust dealing with multiple eigenvalues. In fact the presence of multiple eigenvalues was the reason to retreat to block diagonalization.

3.5 Implementation of sparsing

Equipped with a first order estimate of eigenvalue perturbations we turn to the implementation of sparsing. As we are going to treat non-linear problems (with some time independent structure) the first point is to obtain several linearizations to grasp as good as possible the dynamical structure of the model. This can e.g. be performed by running the simulation off-line and keeping linearizations that differ sufficiently from previous ones. Then sparsing will be applied to all linearizations and a pattern will be chosen that guarantees stability for all of them. As we can regard a new Jacobian as a perturbation of an older one we can use our first order sparsing criterion also to decide when a new Jacobian has to be analyzed. In this case we have $\Delta J = Df_{\mathrm{old}} - Df_{\mathrm{new}}$.

Secondly, as we want to *guarantee* stability for a certain step size, the application of the criterion should be embedded into an iterative process of sparsing and a-posteriori check of stability. In this process a reasonable trade-off has to be achieved between sparsity and the stability properties of the method. In some applications it might be acceptable that the stiff eigenvalues move towards the stability boundary for the sake of sparsity in others not. The sparsing criterion gives us a measure for the expected deviation. The magnitude of the accepted deviation has to be provided by a higher instance, e.g., by fixing a threshold value that can be chosen separately for each cluster of eigenvalues if necessary.

4 Numerical experiments

To test our sparsing criterion on a practical problem we consider the model of an industrial robot with six degrees of freedom (see [9]). In this test example many aspects of simulation with sparsing can be observed. For each joint the mechanical part,

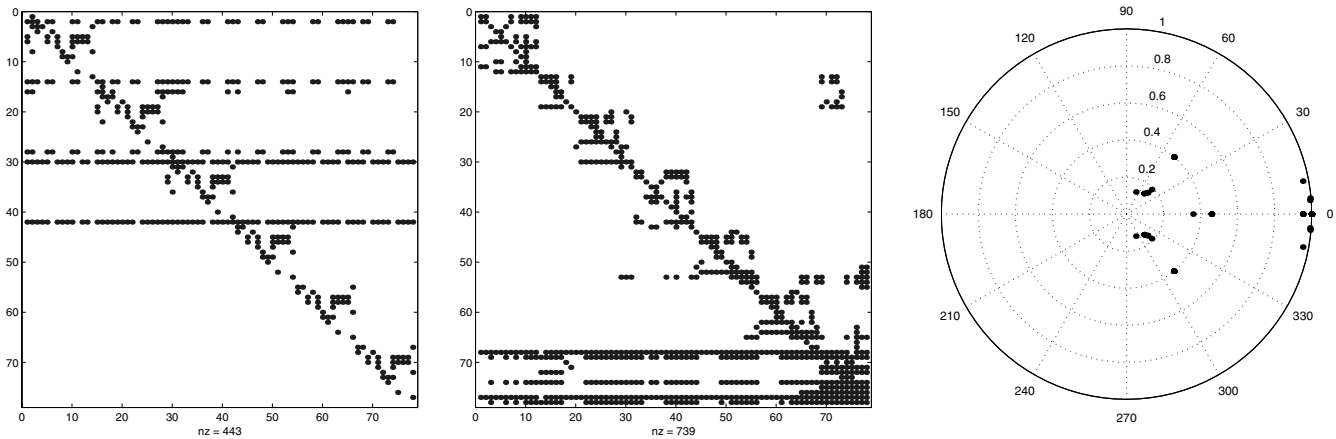

Fig. 1 The Jacobian Df of the robot model before sparsing, the LU-factorization of $I - \tau Df$, and the eigenvalues of the discrete evolution.

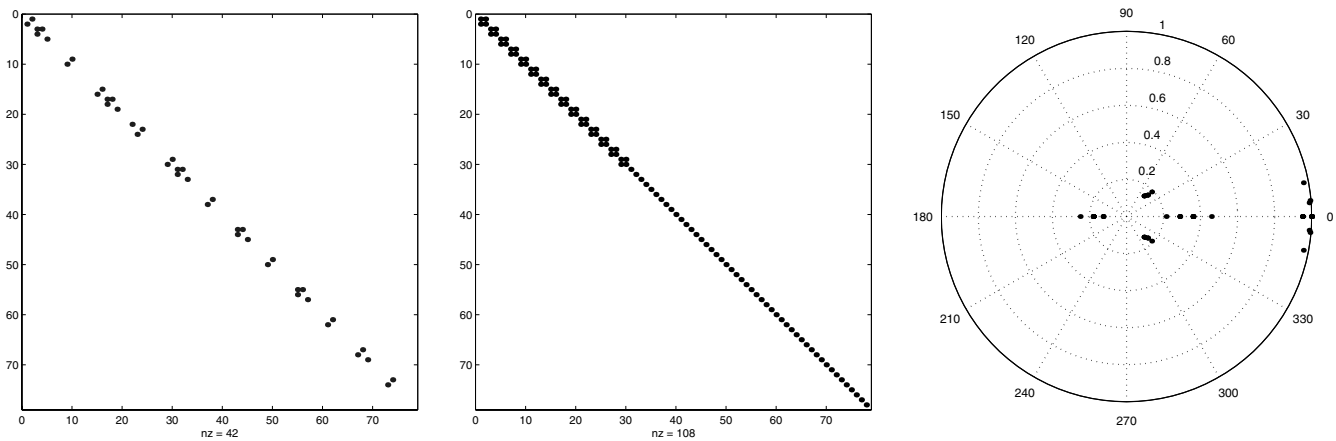

Fig. 2 The Jacobian J of the robot model after sparsing, the LU-factorization of $I - \tau J$, and the eigenvalues of the discrete evolution.

the electric circuit of the motor, and a controller are modelled. This leads to a system of 78 ODEs. This industrial robot is a very characteristic example for a modular system as described in Sect. 2.1. The dynamic behavior of the electric circuits and the controller takes place on faster time scales than the movement of the robot. An interesting feature of the robot is also that the six drives are only coupled by the mechanical part. We will see that this structure is visible in the sparsity structure of the Jacobian after sparsing.

Sparsing was performed for a simulation of the robot with the linearly implicit Euler method and a step size $\tau = 1\,\mathrm{ms}$. In the right part of Fig. 1 we see the eigenvalues of the discretized system (with exact Jacobian) at time $t = 0$, i.e. the eigenvalues of the matrix pair $(I, I - \tau Df|_{0,x_0})$. We can observe the clustering of the eigenvalues very well: the stiff eigenvalues are all inside a disc with radius 0.5, a couple of non-stiff eigenvalues are very close to 1, and near the stability boundary there are three pairs of oscillatory eigenvalues. Sparsing of this system leads to a tridiagonal matrix. Except for the stiff eigenvalues all eigenvalues remain virtually unchanged.

Both matrices are factorized by the Matlab 6.0 sparse LU-solver with a pre-ordering applied to rows and columns (command: `colamd(I-tau*Df)`). Figs. 1 and 2 show a comparison of the structure of the Jacobians, the structure of the LU-factors (factorization in place) and of the eigenvalues of the corresponding discrete system. Note that both factorizations are performed with threshold pivoting. Therefore, especially the factors of the unsparsed Jacobian could be much denser if the choice of the pivot elements had to be changed for numerical reasons. The tridiagonal sparsed Jacobian allows for far better worst case estimates.

If the robot was a linear system, then real-time simulation could be performed using the sparsed Jacobian described above. However, during the movement of the robot the Jacobian changes significantly and some of the eigenvalues that were close to zero at start are now moving into the unit sphere and spread out. This makes sparsing more difficult and allows for less elements to be sparsed. Moreover, we want to balance between sparsity and good approximation of these intermediate eigenvalues because they have a stronger influence on the long term behavior of the difference equation than the stiff eigenvalues . We can see in Figs. 3 and 4 how sparsing affects the structure of the Jacobian and the location of the eigenvalues in this case.

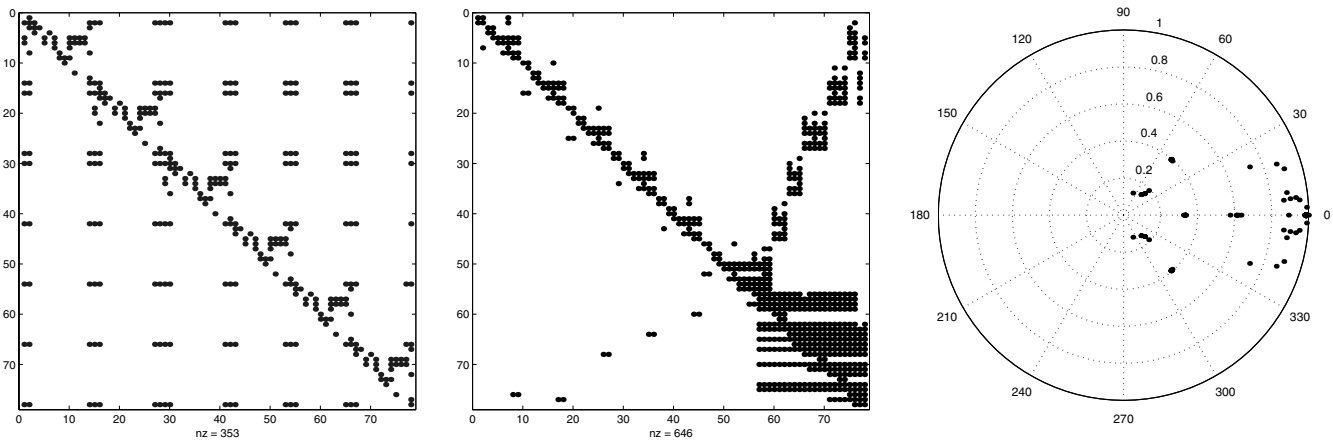

Fig. 3 The Jacobian Df of the moving robot model before sparsing, the LU-factorization of $I - \tau Df$, and the eigenvalues of the discrete evolution.

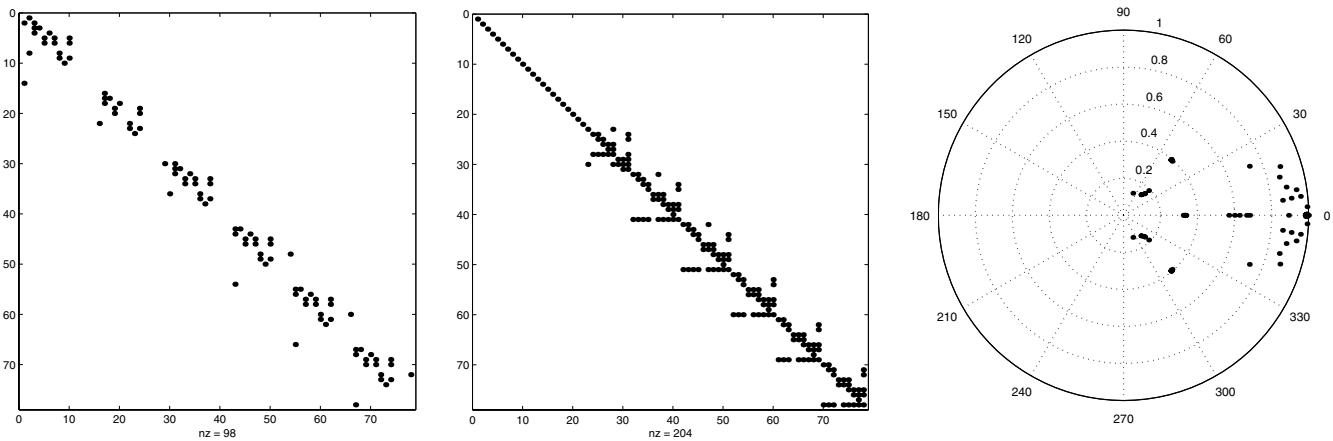

Fig. 4 The Jacobian J of the moving robot model after sparsing, the LU-factorization of $I - \tau J$, and the eigenvalues of the discrete evolution.

	$\mathrm{nnz}_{I-\tau J}$	$\mathrm{nnz}_{\mathrm{LU}}$	t_{fac}
sparsing	158	204	0.22 ms
no sparsing	390	646	0.67 ms
dense lin. alg.	78×78	78×78	2.0 ms

Table 1 Performance of sparse LU factorization with sparsing, without sparsing and dense LU factorization.

Table 1 lists some results concerning the performance of sparse factorizations of the matrices shown in Figs. 3 and 4. As we assume that a sequence of systems of equations has to be solved we measure only the time for the factorization itself. We observe that while the number of non-zeros of $I - \tau J$ is approximately halved, the performance gains in LU factorization is at about a factor of 3 compared to a sparse solver and about a factor of 11 compared to a dense solver. We also observe that the sparsity patterns of Figs. 1 and 3 are completely different. In real time simulation this would force us to use a dense solver. In contrast to this our preprocessing routine found a sparsity pattern that can be held constant and therefore enables us to use a sparse solver in real time simulation.

Turning to the simulation, we first have to say that meaningful time measurements are not available yet. The evaluation of the model equations in Matlab introduces a large overhead due to the interface between Matlab and the modelling tool. However, the impact of sparsing on the accuracy of the solution can be studied. As we already mentioned, the analysis of the Jacobian at $t = 0$ is not sufficient to obtain a stable simulation. The instabilities do not originate from the stiff part but from a pair of eigenvalues close to the stability boundary. If we use a linearization of the robot model at an instant when the robot is in motion, we obtain a stable simulation. Comparing the simulated trajectory to one with higher accuracy, we observe that the errors introduced by sparsing are in our case not negligible and somewhat larger than the errors obtained by integration with the exact Jacobian. However the relative errors are still in a range of 1% to 3%. If we apply more aggressive sparsing we obtain spurious oscillations in the solution. This indicates that the chosen step size of $1\ \mathrm{ms}$ is slightly too large and sparsing therefore also influences the behavior of the discrete evolution on large time scales.

5 Conclusions and outlook

We have shown that sparsing of the Jacobian is a practicable way of improving efficiency in real time simulation using one of the few possibilities of adaptivity in this field. We have derived a first order sparsing criterion based on perturbation theory. Details of its application were given. We have also tested our sparsing routine on an industrial application and we have achieved considerable performance gains in the factorization of the Jacobian.

To judge the overall performance of the method it will be necessary to implement a real time simulator in a low level programming language. Then on the one hand the effects of the function evaluations can be included, on the other hand the computational overhead can be reduced, e.g., by using static data structures together with worst case estimates for the sparse factorization.

A typical case of application in object oriented modelling is a separated DAE with a large sparse algebraic part. If such a system is very large it will be necessary to exploit this structure for the computation of the criterion. On the other hand, if the algebraic part has got moderate size only, it might be possible to treat this part more carefully.

There are some related issues that could be interesting to explore. For example, how can the sparsing criterion be utilized for off-line simulation using, e.g., the linearly implicit Euler method with extrapolation. Or the other way round, how to combine sparsing and extrapolation to improve accuracy in real time simulations.

Acknowledgements The work of the first author was in parts supported by the European Commission under contract IST-199-11979 with DLR under the Information Societies Technology project entitled "Real-time simulation for design of multi-physics systems". The test model was generated using the modelling language Modelica [7] implemented in the modelling and simulation package Dymola [3].

The authors would like to thank the people at DLR, especially Dr. Martin Otter, and the developers of Dymola for their support and encouragement.

References

[1] P. Deuflhard and F. Bornemann, Numerische Mathematik II (De Gruyter, 1994).
[2] I.S. Duff and U. Nowak, On sparse solvers in a stiff integrator of extrapolation type, IMA J. Numer. Anal. **7**, 391–405 (1987).
[3] Dymola: Dynasim AB, Lund, Sweden. Homepage: http://www.dynasim.se.
[4] H. Elmqvist, S. E. Mattsson, and M. Otter, Object-oriented and hybrid modeling in Modelica, APII-JESA J. Eur. Syst. Autom. **35**(1), I–X (2001).
[5] G. H. Golub and C.F. Van Loan, Matrix Computations (Johns Hopkins University Press, Baltimore, Maryland, 1990).
[6] E. Hairer and G. Wanner, Solving Ordinary Differential Equations II. Stiff and Differential Algebraic Problems, Springer Series in Computational Mathematics, 2nd Edition (Springer, 1996).
[7] Modelica, A unified object-oriented language for physical systems modelling, Modelica homepage: http://www.modelica.org.
[8] Nowak, Dynamic sparsing in stiff extrapolation methods, Impact Comput. Sci. Eng. **5**, 53–74 (1993).

[9] M. Otter, Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter, Dissertation, Fortschrittberichte VDI, Reihe 20, Nr. 147 (1995).
[10] M. Otter et al., Objektorientierte Modellierung physikalischer Systeme 1–17, Automatisierungstechnik **47**(1)–**48**(12), (1999/2000).
[11] T. Steihaug and A. Wolfbrandt, An attempt to avoid exact Jacobian and nonlinear equations in the numerical solution of stiff differential equations, Math. Comput. **33**, 521–534 (1979).
[12] G. W. Stewart and J. Sun, Matrix Perturbation Theory, Computer Science and Scientific Computing (Academic Press 1990).
[13] M. Weiner et al., Partitioning strategies in Runge-Kutta type methods, IMA J. Numer. Anal. **13**, 303–319 (1993).