# Weierstraß–Institut
## für Angewandte Analysis und Stochastik

im Forschungsverbund Berlin e.V.

Preprint ISSN 0946 – 8633

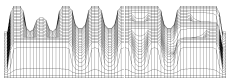

## Parallel modular dynamic process simulation


Jürgen Borchardt, Klaus Ehrhardt, Friedrich Grund
and Dietmar Horn



Preprint No. 439

Berlin 1998

*1991 Mathematics Subject Classification.* 65Y05, 65L05, 65H10, 65F50, 80A30.

*Keywords and Phrases.* Systems of differential–algebraic equations; Block partitioned systems; Newton–type methods; Sparse–matrix techniques; Parallelization; Chemical process simulation; Dynamic simulation of distillation plants.

This work was supported by the Federal Ministry of Education, Science, Research and Technology, Germany under grant GR7FV1.

# Parallel Modular Dynamic Process Simulation *

Jürgen Borchardt, Klaus Ehrhardt, Friedrich Grund, and Dietmar Horn

Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstrasse 39, 10117 Berlin, Germany, e-mail borchardt@wias-berlin.de

**Abstract.** To meet the needs of plant wide dynamic process simulation of today's complex, highly interconnected chemical production plants, parallelizable numerical methods using divide and conquer strategies are considered. The large systems of differential algebraic equations (DAE's) arising from an unit oriented modular modeling of chemical and physical processes in a chemical plant are partitioned into blocks. Using backward differentiation formulas (BDF), a partitioned system of nonlinear equations has to be solved at each discretization point of time. By formally extending these systems, block–structured Newton–type methods are applied for their solution. These methods enable a coarse grain parallelization and imply an adaptive relaxation decoupling between blocks. The resulting linear subsystems with sparse and unsymmetric coefficient matrices are solved with a Gaussian elimination method using pseudo code techniques for an efficient multiple refactorization and solution. Results from dynamic simulation runs for industrial distillation plants on parallel computers are given.

## 1 Introduction

The hierarchical modular structure of large chemical production plants can be exploited for a plant wide process simulation. Here a plant is considered as a network of connected process units like reactors, head exchangers or trays of distillation columns. By a so called flowsheeting the units are connected by mass and energy streams and a parameter dependent mathematical model is linked to each unit type. For the dynamic simulation, this leads to an initial value problem for a large system of DAE's which is structured corresponding to the units into $m$ coupled subsystems

$$F_i(t, y(t), \dot{y}(t), u(t)) = 0, \quad i = 1(1)m, \tag{1}$$

$$F_i : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^l \to \mathbb{R}^{n_i}, \quad \sum_{i=1}^{m} n_i = n, \quad t \in [t_0, t_{end}]$$

with given piecewise continuous parameter function $u(t)$ and the unknown function $y(t)$. For the considerations in this paper it is assumed that the DAE system (1) is index one [5]. In real life applications, it can involve tens of thousands of equations or more. For solving such large scale problems, a

* This work was supported by the Federal Ministry of Education, Science, Research and Technology, Germany under grant GR7FV1.

two level hierarchical structure of the system is considered. The first level of the structure is built by the subsystems of the DAE system, while the second level of the hierarchy is obtained by merging subsystems to blocks

$$\tilde{\mathcal{F}}_j = (F_{j_1}, F_{j_2}, \ldots, F_{j_{m_j}})^T, \qquad j = 1(1)p, \quad \sum_{j=1}^{p} m_j = m. \tag{2}$$

Such a so called block partitioning (2) can be predefined by a macro model description covering functional blocks or can otherwise be generated automatically by different partitioning algorithms [8]. It can be changed during the numerical simulation if needed.

In Section 2, it is described how the hierarchical structure of the DAE system can be used to construct effectively parallelizable block–structured Newton–type methods. These methods, based on block Schur–complement techniques, require a repeated solution of linear systems with the same pattern structure of the sparse and unsymmetric coefficient matrices but with different right hand sides. For this, a direct solver [2] is described in Section 3. It uses a pseudo code technique for an efficient multiple LU-refactorization and solution. Finally, in Section 4, results for large scale real life applications of the Bayer AG Leverkusen are given.

## 2 Parallel Newton–type methods

Because (1) is usually a stiff problem, BDF methods [5] are used for its solution. For these methods a system of nonlinear equations has to be solved at each discretization point of time. Based on a block partitioning (2), this system is formally extended to use block–structured Newton–type methods for its solution on parallel computers. The extension is done by determining the internal variables $x = (x_1, \ldots, x_p)^T$ of the blocks, duplicating of external couple variables $z = (z_1, \ldots, z_p)^T$, and appending identification equations $\mathcal{G}(z) = 0$, yielding the extended block partitioned system

$$\mathcal{F}_j(x_j, z_j) = 0, \qquad j = 1(1)p, \tag{3a}$$

$$\mathcal{G}(z) = 0, \tag{3b}$$

where the nonlinear functions $\mathcal{F}_j : \mathbb{R}^{r_j} \times \mathbb{R}^{s_j} \longrightarrow \mathbb{R}^{q_j}$, $r_j + s_j \geq q_j$, corresponding to the blocks $\tilde{\mathcal{F}}_j$ have disjunctive arguments and the function $\mathcal{G} : \mathbb{R}^s \longrightarrow \mathbb{R}^{r+s-n}$ with $\sum_{j=1}^{p} r_j = r$, $\sum_{j=1}^{p} s_j = s$, and $\sum_{j=1}^{p} q_j = n$ is linear.

Using the abbreviations $\Delta x_j := x_j^{k+1} - x_j^k$, $\mathcal{F}_j := \mathcal{F}_j(x_j^k, z_j^k)$, $\mathcal{G} := \mathcal{G}(z^k)$, $\partial_{x_j}\mathcal{F}_j := \frac{\partial \mathcal{F}_j}{\partial x_j}(x_j^k, z_j^k)$ and corresponding terms for $\Delta z_j$, $\partial_{z_j}\mathcal{F}_j$ and $\partial_{z_j}\mathcal{G}$, one formally gets for the $k$th iteration step of a Newton–type approach

$$0 = \mathcal{F}_j + \partial_{x_j}\mathcal{F}_j \Delta x_j + \partial_{z_j}\mathcal{F}_j \Delta z_j, \qquad j = 1(1)p, \tag{4a}$$

$$0 = \mathcal{G} + \sum_{j=1}^{p} \partial_{z_j} \mathcal{G} \Delta z_j. \tag{4b}$$

In [2,3] we have proposed block–structured Newton–type methods based on a splitting of the block functions $\mathcal{F}_j$ into $\mathcal{F}_j = (\mathcal{F}_j^1, \mathcal{F}_j^2)^T$. The splitting is obtained by determining $r_j$ pivot elements in the $q_j \times r_j$ dimensional matrix $\partial_{x_j} \mathcal{F}_j$, so that the $r_j$ pivot rows determine $\mathcal{F}_j^1$, and the regularity of $\partial_{x_j} \mathcal{F}_j^1$ is ensured. In this case one gets

$$\Delta x_j = -\Delta \hat{x}_j - B_j \Delta z_j, \qquad j = 1(1)p, \tag{5a}$$

$$0 = \hat{\mathcal{F}}_j + C_j \Delta z_j, \qquad j = 1(1)p, \tag{5b}$$

$$0 = \mathcal{G} + \partial_z \mathcal{G} \Delta z \tag{5c}$$

with $\Delta \hat{x}_j := (\partial_{x_j} \mathcal{F}_j^1)^{-1} \mathcal{F}_j^1$, $\hat{\mathcal{F}}_j := \mathcal{F}_j^2 - \partial_{x_j} \mathcal{F}_j^2 \Delta \hat{x}_j$, $B_j := (\partial_{x_j} \mathcal{F}_j^1)^{-1} \partial_{z_j} \mathcal{F}_j^1$, and $C_j := \partial_{z_j} \mathcal{F}_j^2 - \partial_{x_j} \mathcal{F}_j^2 B_j$.

Thus, after computing the approximations $\Delta \hat{x}_j$ and the right hand sides $\hat{\mathcal{F}}_j$ for each block system, the correction of the external variables $\Delta z$ can be computed from the so called main system or coupling equations (5b),(5c) and the correction of the internal variables $\Delta x$ can then be computed from the block system equations (5a).

So, using the notations $C :=\mathrm{diag}(C_j)$ and $\hat{\mathcal{F}} := (\hat{\mathcal{F}}_1, \hat{\mathcal{F}}_2, \ldots, \hat{\mathcal{F}}_p)^T$, one gets from (5a)–(5c) that the evaluation of the corrections $\Delta x$ and $\Delta z$ in the $k$th iteration step of a modified Newton method with scalar constant $c$ can be efficiently realized in the following basic steps:

$$\begin{bmatrix} \partial_{x_j} \mathcal{F}_j^1 & \emptyset \\ \partial_{x_j} \mathcal{F}_j^2 & I \end{bmatrix} \begin{pmatrix} \Delta \hat{x}_j \\ \hat{\mathcal{F}}_j \end{pmatrix} = c \begin{pmatrix} \mathcal{F}_j^1 \\ \mathcal{F}_j^2 \end{pmatrix}, \qquad j = 1(1)p, \tag{6a}$$

$$\begin{bmatrix} \partial_{x_j} \mathcal{F}_j^1 & \emptyset \\ \partial_{x_j} \mathcal{F}_j^2 & I \end{bmatrix} \begin{bmatrix} B_j \\ C_j \end{bmatrix} = \begin{bmatrix} \partial_{z_j} \mathcal{F}_j^1 \\ \partial_{z_j} \mathcal{F}_j^2 \end{bmatrix}, \qquad j = 1(1)p, \tag{6b}$$

$$\begin{bmatrix} C \\ \partial_z \mathcal{G} \end{bmatrix} \Delta z = - \begin{pmatrix} \hat{\mathcal{F}} \\ c\, \mathcal{G} \end{pmatrix}, \tag{6c}$$

$$\Delta x_j = -\Delta \hat{x}_j - B_j \Delta z_j, \qquad j = 1(1)p. \tag{6d}$$

In this paper Newton-type methods based on (6a)–(6d) are called *type 1 methods*. For these methods, the steps (6a),(6b), and (6d) can be done concurrently for all block systems. Implementing them on parallel computers with shared memory, both main parts of the computational amount, namely all the calculation of functions and Jacobians as well as most of the amount for the solution of the linear systems, can be covered together in one parallel loop built up from (6a) and (6b). This results into a coarse grain parallelism. The bottleneck is the sequential part (6c), which is dominated by the LU–factorization of the main system matrix. To reduce this sequential amount of the algorithm and to increase the efficiency of the implementation on parallel computers, various modifications of the method as e.g. *multilevel Newton iteration* techniques can be considered [4].

Another possibility to reduce the computational amount for the solution of the coupling system is based on identifying input and output streams of the units in the flowsheet. Due to this, the external variables $z_j$ of a block system can be divided into input and output variables $u_j$ and $v_j$ with

$$z_j = (u_j, v_j)^T, \quad u_j \in \mathbb{R}^{r_j+s_j-q_j}, \; v_j \in \mathbb{R}^{q_j-r_j}, \tag{7}$$

and $\mathcal{G}$ can be chosen so that $\partial_v \mathcal{G} = -I$ and $\Delta v = \partial_u \mathcal{G} \Delta u$. Here, $s = 2(q-r)$ is assumed for notational simplicity, that means, that $\partial_u \mathcal{G}$ is a permutation matrix and there is only one input per output. But the following can be extended to the multiple input case as well.

If the output variables can be computed from the block system equations together with the internal variables, then an inverse $\left[\partial_{x_j}\mathcal{F}_j \; \partial_{v_j}\mathcal{F}_j\right]^{-1}$ exists and it is not necessary to split the block system equations. Using the abbreviation $B_v = diag(B_{v_i})$, this results into a *type 2 method*, if (6a)–6d) is replaced by

$$\left[\partial_{x_j}\mathcal{F}_j \; \partial_{v_j}\mathcal{F}_j\right] \begin{pmatrix} \Delta\hat{x}_j \\ \Delta\hat{v}_j \end{pmatrix} = c\,\mathcal{F}_j\,, \qquad j = 1(1)p, \tag{8a}$$

$$\left[\partial_{x_j}\mathcal{F}_j \; \partial_{v_j}\mathcal{F}_j\right] \begin{bmatrix} B_{x_j} \\ B_{v_j} \end{bmatrix} = \partial_{u_j}\mathcal{F}_j\,, \qquad j = 1(1)p, \tag{8b}$$

$$[\partial_u \mathcal{G} + B_v]\,\Delta u = -\Delta\hat{v}\,, \tag{8c}$$

$$\Delta v = \partial_u \mathcal{G} \Delta u\,, \tag{8d}$$

$$\Delta x_j = -\Delta\hat{x}_j - B_{x_j}\Delta u_j, \qquad j = 1(1)p. \tag{8e}$$

Compared to *type 1 methods* the size of the main system (8c) of a *type 2 method* is reduced to the half. Apart from the fact that *type 1 methods* can be applied to more general problems, *type 2 methods* enable a better parallelization and applicability of relaxation decoupling between blocks.

To introduce a relaxation decoupling in the case of weakly coupled blocks, previous values of $\Delta u$ are used to approximate $B_v \Delta u$ in (8c). This formally gives

$$\left[\partial_{x_j}\mathcal{F}_j \; \partial_{v_j}\mathcal{F}_j\right] \begin{pmatrix} \Delta\hat{x}_j \\ \Delta\hat{v}_j \end{pmatrix} = c\,\mathcal{F}_j\,, \qquad j = 1(1)p, \tag{9a}$$

$$\left[\partial_{x_j}\mathcal{F}_j \; \partial_{v_j}\mathcal{F}_j\right] \begin{pmatrix} B_{x_j}\Delta u_j \\ B_{v_j}\Delta u_j \end{pmatrix} = \partial_{u_j}\mathcal{F}_j \Delta u_j\,, \qquad j = 1(1)p, \tag{9b}$$

$$\Delta v_j = -\Delta\hat{v}_j - B_{v_i}\Delta u_j, \qquad j = 1(1)p, \tag{9c}$$

$$\Delta u = \left[\partial_u \mathcal{G}\right]^T \Delta v, \tag{9d}$$

$$\Delta x_j = -\Delta\hat{x}_j - B_{x_j}\Delta u_j, \qquad j = 1(1)p. \tag{9e}$$

Here the explicit evaluation of the sensitivity matrices $B_{x_j}$ and $B_{v_j}$ of internal and of output variables with respect to input variables can be avoided by

computing only the vectors $(B_{x_j} \Delta u_j)$ and $(B_{v_j} \Delta u_j)$. Usually the steps (9b)–(9d) are iterated starting with a first approximate $\Delta u \approx - \left[\partial_u \mathcal{G}\right]^T \Delta \hat{v}$. Except of the permutation (9d) all other steps can be done in parallel.

## 3 Repeated solution of sparse linear systems

The block–structured Newton–type methods described in the previous section imply a repeated solution of linear systems. For solving these systems

$$Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad x, b \in \mathbb{R}^n \tag{10}$$

with unsymmetric and sparse matrices $A$ the Gaussian elimination method

$$PAQ = LU, \tag{11}$$

$$Ly = Pb, \quad UQ^{-1}x = y \tag{12}$$

is used. The nonzero elements of the matrix $A$ are stored in compressed sparse row format. $L$ is a lower triangular matrix and $U$ an upper triangular matrix. The row permutation matrix $P$ is used to provide numerical stability and the column permutation matrix $Q$ is used to control sparsity.

The determination of the pivots is essential for solving linear systems using the Gaussian elimination method. The numerical stability can be saved for linear systems with dense matrices using partial or complete pivoting. The numerical complexity is $O(n^3)$ in these case. In contrast to linear systems with dense matrices the numerical complexity can be reduced dramatically for linear systems with sparse matrices. The fundamental problem is the identification of the pivot columns, what corresponds to the determination of the permutation matrix $Q$. In [10] two cases for the determination of the permutation matrix $Q$ are considered.

In the first case the pivot column is determined in each elimination step and the columns are dynamically reordered. The pivot columns are found with a heuristic criterion. For this, the first column with a minimal number of nonzero elements is searched in the matrix to be factorized. In the previous version the numerical complexity of this method was $O(n^2)$. Now a new method has been developed having only a complexity of $O(n)$. For keeping the method numerically stable partial pivoting is applied in the pivot column.

In the second case the permutation matrix $Q$ is determined by a minimum degree ordering of $A^T A$ or of $A^T + A$. The columns are statically reordered in this case. Partial pivoting is used in the pivot columns as well.

To perform several factorizations for matrices with the same pattern structure using the same pivot sequence as well as to solve the linear system for several right hand sides, a pseudo code is generated [10]. This code describes the operations that are necessary for factorization (11) and solution (12) of the linear system. Using the pseudo code enables a fast refactorization and multiple solution as well.

## 4 Applications

The methods described in Section 2 and 3 are used in the block oriented process simulation package BOP. This dynamic simulation package uses a hierarchically structured data interface [11], which is currently generated from the data supplied by the commercial process simulator SPEEDUP [1]. The interface describes the system of DAE's structured into subsystems corresponding to the units of the plant and is usable for an independent evaluation of subsystem functions and Jacobian matrices. If no block decomposition is predefined a topological block partitioning algorithm is used. To apply numerical integration with BDF methods, the DASSL code [5] has been modified with respect to the nonlinear and linear solver, consistent initialization [13] and handling of discontinuities. The simulation package BOP is currently implemented on moderate parallel computers Cray J90, SGI Origin 2000 and DEC AlphaServer using multiprocessing compiler directives for parallelization.

Used for the dynamic process simulation of various large distillation plants of the Bayer AG Leverkusen, BOP has shown a good parallel performance. All times given in Table 1 and 2 are measured for whole simulation runs on non dedicated machines Cray J90 and include the times for sequential pre– and post–processing.

**Table 1.** Dynamic simulation of plant *bayer12* (19 558 equations) with BOP

| Processors | 1 | 1 | 7 | 21 |
|---|---|---|---|---|
| Blocks | 1 | 21 | 21 | 21 |
| CPU time (sec.) | 1 250 | 1 124 | 1 161 | 1 142 |
| Wall clock time (sec.) | 1 285 | 1 148 | 245 | 142 |
| Speedup factor | 1 | 1.12 | 5.24 | 9.05 |

**Table 2.** Dynamic simulation of plant *bayer01* (57 735 equations) with BOP

| Processors | 1 | 1 | 8 |
|---|---|---|---|
| Blocks | 1 | 16 | 16 |
| CPU time (sec.) | 1 833 | 1 071 | 1 538 |
| Wall clock time (sec.) | 1 866 | 1 084 | 372 |
| Speedup factor | 1 | 1.72 | 5.02 |

In Table 3 the performance of BOP using different implemented block structured Newton-type methods is compared to that of SPEEDUP [1] at a Cray J90. The example is a reactor model built up modularly by a multi phase cell model which might be associated to a simplified reactive separation volume element.

**Table 3.** Dynamic simulation of *reactor600* (45 600 equations)

| Simulation with | Processors | Blocks | CPU time (sec.) | Wall clock time (sec.) |
|---|---|---|---|---|
| SPEEDUP | 1 | 1 | 7 008 | 7 516 |
| BOP | 1 | 1 | 5 089 | 5 120 |
| BOP with type 1 | 1 | 18 | 5 814 | 5 870 |
| BOP with type 2 | 1 | 18 | 4 932 | 4 967 |
| BOP with type 1 | 6 | 18 | 6 208 | 1 904 |
| BOP with type 2 | 6 | 18 | 5 140 | 1 371 |

The linear solver described in Section 3 is realized in the package GSPAR, which is integrated in the simulation package BOP. In Table 4 the performance of GSPAR is compared to that of SuperLU [7] regarding to the first factorization (pivoting and factorization) of coefficient matrices of linear systems resulting from real life dynamic process simulation of chemical plants.

**Table 4.** CPU times for first factorization with GSPAR and SuperLU

| | | | GSPAR | | | SuperLU |
|---|---|---|---|---|---|---|
| name | n | $\|A\|$ | previous | new | mmd | mmd |
| bayer01 | 57 735 | 277 774 | 34.92 | 2.35 | 7.53 | 4.48 |
| bayer02 | 13 935 | 63 679 | 2.20 | 0.55 | 1.41 | 0.85 |
| bayer03 | 6 747 | 56 196 | 0.67 | 0.30 | 0.63 | 0.66 |
| bayer04 | 20 545 | 159 082 | 5.18 | 1.82 | 3.03 | 2.17 |
| bayer05 | 3 268 | 27 836 | 0.13 | 0.07 | 0.45 | 0.60 |
| bayer06 | 3 008 | 27 576 | 0.82 | 0.83 | 1.62 | 0.65 |
| bayer09 | 3 083 | 21 216 | 0.20 | 0.10 | 0.25 | 0.23 |
| bayer10 | 13 436 | 94 926 | 3.07 | 1.27 | 2.17 | 1.35 |

For the $n \times n$ matrices[1] with $|A|$ nonzero elements the CPU times in seconds on a DEC AlphaServer (processor 21164A with 400 MHz) are given for GSPAR using the previous dynamic ordering, the new dynamic ordering, and a minimum degree ordering (mmd) of $A^TA$ respectively as well as for SuperLU using a minimum degree ordering of $A^TA$. With the new dynamic ordering GSPAR now achieves a fast first factorization. The corresponding CPU times needed for refactorization (factorization with given pivot sequence) are listed in [3,10].

[1] The matrices can be found in Tim Davis, University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/~davis/sparse/

**Acknowledgements.** We wish to thank the Bayer AG Leverkusen for the valuable support and Aspen Technology, Inc. for providing us a free SPEEDUP licence for academic use.

## References

1. Aspen Technology Inc. (1995) SPEEDUP, User Manual, Library Manual. Cambridge, Massachusetts
2. Borchardt, J., Grund, F., Horn, D., Michael, T. (1997) Parallelized Numerical Methods for Large–Scale Dynamic Process Simulation. In: Sydow, A. (ed.): Proceedings of the 15th IMACS World Congress on Scientific Computation, Wissenschaft & Technik Verlag, Berlin, vol. I, 547–552
3. Borchardt, J., Grund, F., Horn, D. (to appear) Parallelized Methods for Large Nonlinear and Linear Systems in the Dynamic Simulation of Industrial Applications. Surveys on Mathematics for Industry
4. Borchardt, J., (to appear) Parallelized Block–Structured Newton–Type Methods in Dynamic Process Simulation. In: Proceedings of PARA98, Springer, Berlin Heidelberg
5. Brenan, K.E., Campbell, S.L., Petzold, L.R. (1989) Numerical Solution of Initial–Value Problems in Differential–Algebraic Equations. North–Holland, New York
6. Gräb, R., Günther, M., Wever, U., Zheng, Q. (1996) Optimization of Parallel Multilevel–Newton Algorithms on Workstation Clusters. In: Bouge, L. et al. (Eds.): Euro–Par96 Parallel Processing, Berlin, Lecture Notes in Computer Science 1124, Springer, Berlin Heidelberg, 91–96
7. Demmel, J. W., Gilbert, J. R., Li, X. S. (1997) SuperLU Users' Guide, University of California, Berkeley; Xerox Palo Alto Research Center; NERSC, Lawrence Berkeley National Lab
8. Grund, F., Borchardt, J., Horn, D., Michael, T., Sandmann, H. (1996) Differential–algebraic systems in the chemical process simulation. In: Keil, F., Mackens, W., Voss, H., Werther, J. (eds.): Scientific Computing in Chemical Engineering, Springer–Verlag, Berlin, 68–74
9. Grund, F., Michael, T., Brüll, L., Hubbuch, F., Zeller, R., Borchardt, J., Horn, D., Sandmann, H. (1997) Numerische Lösung großer strukturierter DAE–Systeme in der chemischen Prozeßsimulation. In: Hoffmann, K.H., Jäger, W., Lohmann, Th., Schunk, H. (Hrsg.): Mathematik–Schlüsseltechnologie für die Zukunft, Springer–Verlag, Berlin Heidelberg, 91–103
10. Grund, F. (1998) Direct linear solvers for vector and parallel computers. Preprint No. 415, WIAS Berlin
11. Horn, D. (1996) Entwicklung einer Schnittstelle für einen DAE–Solver in der chemischen Verfahrenstechnik. In: Mackens, W., Rump, S.M. (Hrsg.),: Software Engineering im Scientific Computing, Vieweg & Sohn, Braunschweig, 249–255
12. Hoyer, W., Schmidt, J.W. (1984) Newton–Type Decomposition Methods for Equations Arising in Network Analysis. ZAMM 64, 397–405
13. Kröner, A., Marquardt, W., Gilles, E.D. (1992) Computing Consistend Initial Conditions for Differential Algebraic Equations. Computers & Chemical Engineering 16, 131–138