1992

J.G. Blom, J.G. Verwer, R.A. Trompert

A comparison between direct and iterative methods to solve the linear systems arising from a time-dependent 2D groundwater flow model

CWI is het Centrum voor Wiskunde en Informatica van de Stichting Mathematisch Centrum

CWI is the Centre for Mathematics and Computer Science of the Mathematical Centre Foundation

CWI is the research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a non-profit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands organization for scientific research (NWO).

A Comparison between Direct and Iterative Methods to Solve the Linear Systems arising from a Time-dependent 2D Groundwater Flow Model†

J.G. Blom, J.G. Verwer, R.A. Trompert‡
CWI
P.O.Box 4079, 1009 AB Amsterdam, The Netherlands

In this paper various methods are compared for solving nonsymmetric linear systems. Both direct and ILU preconditioned iterative solvers are considered. The linear systems arise when the implicit BDF time-stepping method in combination with modified Newton is applied to a system of nonlinear time-dependent two-dimensional PDEs. The PDE system under study is a groundwater flow model simulating transport of brine.

1991 Mathematics Subject Classification: Primary: 65N22. Secondary: 65F05, 65F10, 65M20.
1987 CR Categories: G1.3, G1.8.
Keywords & Phrases: method of lines, finite differences, nonsymmetric sparse linear systems, direct solvers, bandsolvers, sparse matrix solvers, iterative solvers, GMRES, CGS, BI-CGSTAB, ILU preconditioning
Note: This paper will be submitted elsewhere.

1. INTRODUCTION

In [17, 15, 16, 19, 18] an adaptive-grid finite-difference method is studied to solve time-dependent two-dimensional systems of PDEs. Among others, a general research code is developed which uses the implicit two-step BDF method for time-stepping. This poses the task of solving at each time step large systems of nonlinear equations. Initially modified Newton in combination with a sparse matrix solver was used to solve these systems, but when the program was applied to nontrivial problems, as in [18], it readily appeared that it was no longer possible to get a solution in a reasonable time, even on a powerful vector computer. Since the computer time needed to solve the linear systems is the bulk of the total time needed to solve the systems of PDEs, it became obvious that significantly faster linear solvers should be implemented in our research code.

The objective of this paper is to report on a performance study of a number of linear solvers. This performance study involves both direct and iterative solvers and focuses on application to a 2D groundwater flow model for coupled flow and transport of brine in porous media (cf. [18]). As time integrator we have used the (implicit) BDF family as implemented in the DASPK package. DASPK is a modular version by Petzold, Brown, Hindmarsh and Ulrich of the public-domain code DASSL[1] that is available from Netlib[4]. Our choice of DASPK for the performance study, rather than the adaptive-grid code, is not essential, since at the linear system solution level both codes perform a similar task. Hence, conclusions drawn here can be carried over to the adaptive-grid code. The advantage of using DASPK is that we do not need to explain the (intricacies of the) adaptive-grid method. We note that DASPK provides among others the possibility to solve the nonlinear systems

†This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF) for the use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO).
‡Part of contract research by order of the Laboratory for Soil and Groundwater Research of RIVM - the Dutch National Institute of Public Health and Environmental Protection - in connection with Project 'Locatiespecifieke modelvalidatie' 725205 with financial support from the Dutch Ministry of Economic Affairs.

Report NM-R9205
ISSN 0169-0388
CWI
P.O.Box 4079, 1009 AB Amsterdam, The Netherlands

with a combination of matrix-free Newton and the iterative method GMRES. However, we did not use this option and restricted ourselves to modified Newton, since our aim was to compare only various *linear* system solvers.

In Section 2 we describe the brine transport model we have used and we give an outline of the method to solve the system of PDEs. There we also discuss the relevant differences between the straightforward way we used here to solve the system of PDEs and the above-mentioned adaptive-grid method. In Section 3 we briefly survey the linear solvers. One of the options in DASPK is to solve the arising linear systems with the direct bandsolver SGBFA/SGBSL from LINPACK[5]. We have added the direct sparse matrix solver Y12M of Zlatev[22] and, from the Sparse Linear Algebra Package (SLAP) of Greenbaum and Seager (with contributions of several other authors), the iterative solvers GMRES[12] and CGS[14] + ILU preconditioner. Both Y12M and SLAP are public domain codes that are available from Netlib[4]. We also added a variant of CGS, the iterative solver BI-CGSTAB (van der Vorst[20]). In Section 4 we give some implementation details on the stopping criteria used. In Section 5 the actual comparison of the linear solvers follows, both with respect to their memory efficiency and with respect to the computational complexity. This comparison involves running tests in vector mode on 1 CPU of a Cray Y-MP and in scalar mode on an IBM ES/9000. The conclusions can be found in Section 6.

2. Preliminaries

In this section we give a short description of the brine transport problem and the Method-of-Lines (MOL) approach we used to solve this problem. Since this test was set up to determine a more efficient way to solve the linear systems in a code that employs local uniform grid refinement (LUGR), we will also comment on the differences with respect to the resulting Jacobians in the Newton process between the method used in this paper and the LUGR method.

2.1. The 2D fluid-flow salt-transport problem

In [18] we consider a model for a non-isothermal, single-phase, two-component saturated flow problem which consists of 3 PDEs basic to groundwater flow: the continuity equation, the transport equation and the temperature equation. Here, we restrict ourselves to the isothermal case, i.e., to a system of 2 PDEs constituted by the continuity equation and (a slightly modified version of) the transport equation. For the specific details of these equations we refer to [18].

The model describes groundwater flow in a region surrounding a salt dome (cf. HYDROCOIN, Case 5, [13]). We present the model in scaled, non-conservative form. As independent variables we have the pressure p and the salt mass fraction ω. The continuity equation for the fluid, respectively, the salt transport equation are

$$n\rho\gamma\frac{\partial\omega}{\partial t} + \nabla\cdot(\rho\mathbf{q}) = 0, \tag{2.1a}$$

$$n\rho\frac{\partial\omega}{\partial t} + \rho\mathbf{q}\cdot\nabla\omega + \nabla\cdot(\rho\mathbf{J}) = 0, \tag{2.1b}$$

where $n = 0.2$ is the porosity parameter of the porous medium, $\gamma = \ln(1.2)$ is a salt coefficient, and $\rho = \exp(\gamma\omega)$ is the density. Darcy's law gives the equation for the fluid velocity $\mathbf{q} = (q_1, q_2)^T$

$$\mathbf{q} = -\frac{k}{\mu}(\nabla p - \rho\mathbf{g}), \tag{2.2}$$

with $\mathbf{g} = (0, -9.81)^T$ the acceleration-of-gravity vector. The permeability k and viscosity μ are taken constant. Since we have scaled the problem we have $k/\mu = 1$. The equation for the salt-dispersion flux vector is given by Fick's law

$$\mathbf{J} = -n\mathbf{D}\nabla\omega, \tag{2.3}$$

with the dispersion tensor $\mathbf{D}$ for the solute salt defined as

$$n\mathbf{D} = (nD_{mol} + \alpha_T \mathbf{q}^T \mathbf{q})I + (\alpha_L - \alpha_T)\mathbf{q}\mathbf{q}^T. \tag{2.4}$$

The coefficients $D_{mol} = 3 \cdot 10^{-6}$, $\alpha_T = 0.2$ and $\alpha_L = 2$ correspond, respectively, with the molecular diffusion and the transversal and longitudinal dispersion. Note that the definition of the dispersion tensor $\mathbf{D}$ is different from [18]. This formulation was advised by [8] to avoid the difficulties that arise when $\mathbf{q} \approx 0$ (cf. [11]).

Equations (2.1) constitute a system of coupled, nonlinear PDEs. The second equation is parabolic in nature and of the convection-diffusion type. Because ω_t appears twice, the first one can be replaced by a stationary equation which is elliptic in nature (second order in p), showing that (2.1) constitutes an elliptic-parabolic system. We emphasize, however, that we have discretized the problem in the present form (2.1). The problem is defined on the space domain $\Omega = \{(x, y) : 0 < x < 9 \wedge -3 < y < 0\}$. This domain represents a vertical cross section with the y-axis as vertical direction. The initial values at $t = 0$ at Ω are the equations for hydrostatic pressure and fresh water

$$p(x, y) = (1 - \frac{x}{9}) - 9.81y \quad \text{and} \quad \omega(x, y) = 0. \tag{2.5}$$

For $0 < t \leq t_{end} = 160$ (steady state) the following boundary conditions are imposed

$$\begin{array}{lll}
x = 0, 9, & -3 \leq y \leq 0: & q_1 = 0, \\
 & & \omega_x = 0, \\
y = -3, & 0 < x < 9: & q_2 = 0, \\
 & 3 \leq x \leq 6: & \omega = \min(t, 1) \\
 & 0 < x < 3 \wedge 6 < x < 9: & \omega_y = 0, \\
y = 0, & 0 < x < 9: & p = 1 - \frac{x}{9}, \\
 & & \text{inflow } q_2 \leq 0: \quad \omega = 0, \\
 & & \text{outflow } q_2 > 0: \quad \omega_y = 0.
\end{array} \tag{2.6}$$

The interval $3 \leq x \leq 6$ at the lower boundary $y = -3$ represents the location of the salt dome. This salt dome is a source of saturated brine that is released to the overlying groundwater system. In this groundwater system fresh water flows into the domain from approximately the left half of the top boundary and leaves the aquifer from the right half of the top boundary. In the final steady state the salt forms a plume flowing upwards and to the right. The transition brine - fresh water is rather sharp here, especially at the left side of the plume. Furthermore, the flow field shows vortices at the left and at the right of the salt dome.

2.2. Space and time integration

We have discretized system (2.1) in space in a straightforward way on a uniform grid. As space discretization we used second order central differences and for the boundary equations (2.6) second order one-sided differences. The obtained DAE system is integrated in time with the code DASPK. Like DASSL[1], DASPK uses as time integrator the (implicit) BDF family in variable stepsize - variable order mode. The nonlinear systems are solved with modified Newton and the resulting linear systems with the various solvers that will be described in the next section. Required Jacobian matrices are computed in DASPK by numerical differencing. Note that DASPK does not compute a new Jacobian for the Newton process every time step, but only if the size of two subsequent time steps differs considerably or when required for convergence reasons. Also note that the linear systems are band-structured sparse matrices. Because we employ a 9-point stencil a row holds 18 nonzero entries.

2.3. LUGR method

If the solution of a PDE problem possesses sharp moving transitions in space and time, grid uniformity requires that a very fine grid is needed on the entire domain. To restrict the refinement to those regions

where it is truly required, a variety of methods has been developed. One approach is based on static regridding, i.e., a space grid is held fixed for (at least) one time step, combined with local uniform grid refinement. The idea of LUGR is, while starting from a uniform coarse base grid, to generate nested local uniform spatial subgrids which possess internal, nonphysical, boundaries. The level of refinement will be determined by the fine scale structure of the solution. On each of the subgrids a temporal integration step is then performed. If an implicit time integrator is used and if the nonlinear systems are solved with a Newton process, the resulting linear systems then consist of matrices that can be derived from the regular band-structured matrices arising from the standard uniform approach by deleting the rows and columns for those grid points that are not part of the respective subgrid. Consequently, at the linear equation solution level there is, apart from the structure of the Jacobian, no essential difference between using fully or locally uniform space grids. The adaptive grid methods studied in [17, 15, 16, 19, 18] are of the static LUGR type, so that conclusions derived here for DASPK carry over to the research code mentioned in the introduction. This code uses the BDF2 method as time-integrator, also in variable stepsize mode.

3. LINEAR SYSTEM SOLVERS

Roughly speaking, there are two approaches to solve linear systems. The first, the so-called *direct method*, is to factorize the coefficient matrix and invert the factors. The second approach is formed by the *iterative methods*. In these methods a sequence of approximate solutions is generated until a solution satisfies a required precision. For a general overview of both approaches we refer to [3, 10]. This performance study compares the direct solvers SGBFA/SGBSL (from LINPACK[5]) and Y12M[22], the iterative solvers GMRES[12] and CGS[14] (both from the SLAP library) and BI-CGSTAB[20].

3.1. Direct solvers

The easiest although not always the most optimal way to solve a linear system is by using a direct method. The most widely used direct methods are based on Gaussian elimination with some sort of pivoting to preserve the stability of the process. DASPK uses the well-known bandsolver SGBFA/SGBSL from LINPACK[5]. On most vector computers there exists an efficient implementation of the LINPACK library, so it is expected that on such a computer the execution time will be acceptable for a wide range of applications, even though a lot of superfluous computing will be done. In this connection we emphasize that due to the second order discretization of the boundary terms, the bandwidth of the Jacobian incurred by the 9-point stencil is doubled. Therefore, this way to solve the linear system will take up a large amount of memory. Moreover, certainly so on a scalar computer, this method readily becomes slow on a fine grid.

If the matrix has not a regular band structure, as is the case in the LUGR approach, it is natural, especially on a scalar computer, to look at the performance of a sparse matrix solver. With a sparse matrix solver only the nonzero elements of the matrix need to be stored and when performing the LU decomposition the solver will try to maintain the sparsity. The computational overhead, however, is large so that a sparse matrix solver will only be of advantage if the percentage of nonzero elements in the matrix will be small, say 5%–10%. In [17, 15, 16, 19] the HARWELL sparse matrix solver MA28[6, 7] (also available from Netlib) has been used. Later experiments turned out that Y12M[22] performed better, so we decided to use the latter in this competitive examination. A sparse matrix solver often gives the user the choice how to balance the preservation of sparsity versus stability. We have chosen for an 'exact' LU decomposition, i.e., no drop-tolerance (elements in the LU decomposition smaller than the drop-tolerance are omitted) was used. In our experience partial pivoting was sufficient and the possibility to incorporate more rows in the search for a pivotal element made the solver slower.

3.2. Iterative solvers

An iterative method to solve the linear system $Ax = b$, with $A \in \mathbb{R}^{N\times N}$ and $x, b \in \mathbb{R}^N$, is based on the identity

$$A^{-1} = I + (I - A) + (I - A)^2 + \cdots. \tag{3.1}$$

If this series converges, the iteration scheme

$$x_{i+1} = x_i + (b - Ax_i), \tag{3.2}$$

that corresponds to taking partial sums of (3.1), will produce eventually an accurate enough approximation. This recursive relation can be rewritten as

$$x_{i+1} = x_0 + c_0 r_0 + c_1 A r_0 + \cdots + c_i A^i r_0, \tag{3.3}$$

where $r_0 = b - Ax_0$. So x_{i+1} can be expressed as x_0 plus a vector from the so-called $(i+1)$-dimensional *Krylov subspace*

$$K_{i+1}(A; r_0) = span\{r_0, Ar_0, \ldots, A^i r_0\}. \tag{3.4}$$

Many iterative methods belong to the group of so-called Krylov subspace methods in which a basis for the Krylov subspace is build up iteratively and the residual is minimized with respect to this subspace. The *conjugate gradient* method to solve a linear system with a symmetric positive definite matrix is the most familiar example. One can look at CG as a Galerkin method to solve $Ax = b$ onto the Krylov subspace K_i. The solution vector $x \in K_i$ is chosen such that the residual vector is l_2-orthogonal to K_i. The Galerkin method leads to solving a linear system with a matrix which is tridiagonal if A is symmetric and thus can be implemented as a three-terms recurrency. For a non-symmetric matrix A the Galerkin approach leads to an upper Hessenberg matrix in stead of a tridiagonal matrix. This approach gives rise to the *generalized minimal residual* algorithm (GMRES[12]).

One can also interpret CG as a method to minimize the functional

$$\phi(x) = \frac{1}{2} x^T A x - x^T b. \tag{3.5}$$

A widely-used strategy for this is the *steepest descent* algorithm, where a positive α is chosen such that $\phi(x_k + \alpha r_k)$ is minimized. In CG however the search direction, p_k, is not along the residual vector, r_k, but is chosen linearly independent from (more precisely *A-conjugate to*) the previous search directions and such that not only $\phi(x_k + \alpha p_k)$ is minimized but that also

$$x_{k+1} = \arg \min_{x \in span\{p_1, \ldots, p_k\}} \phi(x). \tag{3.6}$$

In this way sequences of residual and direction vectors are chosen where the vectors $\{r_0, \ldots, r_k\}$ are mutually orthogonal and $\{p_0, \ldots, p_k\}$ mutually A-orthogonal. For non-symmetric matrices A one can preserve the efficient three-term recursion but then the functional $\phi(x)$ and thus the residuals are no longer minimized in each step. In practice, however, this method is still of value and especially the more efficient version of it, CGS[14], is widely used.

Note that in the case of iterative methods it is possible to get a taylor-made accuracy that ultimately results in the desired accuracy with which the nonlinear system has to be solved. With direct methods the approximation of the solution of the linear system is only dependent on the condition number of the Jacobian.

3.2.1. GMRES

As stated before GMRES iteratively builds up an l_2-orthononormal basis V_k of the Krylov subspace K_i. This is done using a modified Gram-Schmidt process. Then an approximate solution is determined of the form

$$x_{k+1} = x_0 + z_k, \tag{3.7}$$

where

$$z_k = \arg\min_{z \in K_i} \|b - A(x_0 + z)\|. \tag{3.8}$$

Since one has to keep the complete basis in memory *and* orthogonalize in each step against all previous basisvectors this algorithm is obviously not very practical if many iterations are needed. Therefore an iterative version of the algorithm is mostly used, GMRES(m), where after m steps the approximate solution is determined and the process is restarted with this solution as initial value. Since the subsequent basisvectors are then no longer orthogonal to the first m basisvectors it is clear that convergence is no longer guaranteed. Numerical experiments have shown that it is important to choose m large enough, because otherwise the convergence of GMRES(m) is very slow *if* it converges at all.

3.2.2. CGS and BI-CGSTAB

To preserve the implementational simplicity of CG for non-symmetric matrices A one can use two pairs of three-term recursions, one to generate the residual vectors r and the direction vectors p and the other for $\tilde{r}$ and $\tilde{p}$, with $\tilde{r}_i^T r_j = 0$ and $\tilde{p}_i^T A p_j = 0$ (*the biconjugate gradients method*[9]). It is easy to see that in BI-CG (as in CG) $r_i = P_i(A)r_0$, with P_i an i-th degree polynomial in A, and $\tilde{r}_i = P_i(A^T)r_0$. The constant α and the direction vectors are then chosen so as to force the biorthogonality conditions above and *not* to minimize the functional $\phi(x)$. Therefore, the residuals are (theoretically) not minimized in any sense and although convergence is guaranteed in N steps, the convergence rate will be strongly dependent on the choice of $\tilde{r}_0$. The amount of work per iteration is for BI-CG about twice as much as for CG. Moreover, it also involves the product $A^T v$. In a more efficient version of it, CGS[14], only one row of residuals is constructed with $\hat{r}_i = P_i^2(A)r_0$ (the biorthogonality between r and $\tilde{r}$ is obtained implicitly) and the matrix vector product $A^T v$ is no longer needed.

So CGS can be seen as a polynomial variant of CG. In CG the relation $r_i = P_i(A)r_0$ is satisfied and in CGS this polynomial is squared, i.e., $r_i = P_i^2(A)r_0$. However, as van der Vorst points out in [20], $P_i(A)$ is very dependent on r_0 and the fact that $P_i(A)$ is a reduction operator for r_0 does not at all mean that it is also reduction operator for $P_i(A)r_0$. Van der Vorst therefore has developed a variant of CGS, BI-CGSTAB[20], in which a polynomial of the form

$$Q_i(x) = (1 - \omega_1 x)(1 - \omega_2 x) \cdots (1 - \omega_i x) \tag{3.9}$$

is used, where the constant ω_i is chosen to minimize r_i for all residuals that can be expressed as $r_i = Q_i(A)P_i(A)r_0$.

3.2.3. Preconditioning

When applying iterative methods it is often necessary to use (explicit or implicit) preconditioning of the system, i.e., to find a suitable, easily invertible matrix K, and to premultiply A and b with K^{-1} with the aim to accelerate the convergence of the iterative method. As for the convergence rate it is known that the CG method when applied to symmetric positive definite systems converges (at least) linearly with convergence factor $1 - 2/\sqrt{\kappa}$, where $\kappa = |\lambda_{max}/\lambda_{min}|$ is the spectral condition number of A.

With respect to nonsymmetric matrices some results are known if A has positive real eigenvalues and a complete system of eigenvectors. For such a system the convergence rate of GMRES is [12]

$$\|r_i\| \leq \|r_0\| \left(1 - \frac{\lambda_{min}^2(M)}{\lambda_{max}(A^T A)}\right)^{\frac{i}{2}}, \tag{3.10}$$

where M is the symmetric part of A. For BI-CG and CGS it can be proved that there exists an initial $\tilde{r}_0$ such that these methods have a comparable convergence behavior as CG [14].

We may therefore expect that preconditioning of the system with the aim to reduce the spectral radius of A will be with GMRES and CGS of equal importance as with CG. The simplest method is *diagonal scaling* of A so that the diagonal of the resulting matrix equals 1. It is obvious that for diagonally dominant matrices this will be often a good enough preconditioner. However, it failed in our numerical experiments. Trivially, the best preconditioner to reduce the spectral radius is A itself. A widely used class of preconditioners is therefore based on *incomplete factorizations* of A. We have used here the standard incomplete LU factorization, ILU, that is incorporated in the SLAP-package. In ILU, the preconditioner $K = LU = A + E$, where E is the deviation matrix and L and U have a prescribed sparsity pattern, the nonzero pattern of the lower, respectively, the upper triangle of A itself. K is obtained by applying standard Gaussian elimination but only the nonzero entries of A are modified.

4. Implementation

4.1. Solving the nonlinear system

To solve the arising nonlinear systems, here denoted by $F(y^*) = 0$, we have used the modified Newton process as implemented in DASPK. The Jacobian is computed only when necessary, i.e., if the stepsize has changed much, or if the Newton process does not converge using an 'old' Jacobian. DASPK offers the possibility to compute an approximation of a banded Jacobian using numerical differencing. In this implementation the number of right hand-side evaluations is equal to the bandwidth. Since for our problem we need only 18 right hand-side evaluations to compute the Jacobian, we replaced the DASPK subroutine to avoid the unnecessary right hand-side evaluations and incorporated our own code to compute the Jacobian with numerical differencing, using the same algorithm as DASPK, and storing it either in band or in sparse storage mode. Details of the implementation of DASPK are described in [1]. Here we discuss only those that influence the stopping criterion for the linear solver.

The Newton process is continued until the iteration error $\|y^* - y^{(k)}\|_w$ is sufficiently small, where the user-set time-tolerances are buried in the norm DASPK uses, a weighted root mean square norm,

$$\|v\|_w = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{v_i}{w_i}\right)^2}, \tag{4.1}$$

with

$$w_i = ATOL_i + |y_i^{(0)}|RTOL_i. \tag{4.2}$$

Assuming convergence of the Newton process the inequality

$$\|y^* - y^{(k)}\|_w \leq \frac{\rho}{1-\rho}\|y^{(k)} - y^{(k-1)}\|_w, \tag{4.3}$$

holds, where ρ is the convergence rate which is in actual computation approximated by

$$\rho \approx \rho^{(k)} = \left(\frac{\|y^{(k)} - y^{(k-1)}\|_w}{\|y^{(1)} - y^{(0)}\|_w}\right)^{\frac{1}{k-1}}, \tag{4.4}$$

This leads to the stopping criterion for the Newton iteration as used in DASPK

$$\frac{\rho^{(k)}}{1-\rho^{(k)}}\|y^{(k)} - y^{(k-1)}\|_w < 0.33. \tag{4.5}$$

4.2. Scaling problems

There are two kinds of scaling problems that influence the iterative linear system solution, viz.,

1. different scales of the *solution* components in a PDE,
2. different scales of the *residual equations*, e.g., between the boundary equations and the PDEs defined on the internal domain.

The first scaling problem can be solved by a careful choice of the tolerances in (4.2). In [1, p.131] the following is recommended: Set $RTOL_i = 10^{-(m+1)}$ where m is the number of significant digits required for solution component y_i and set $ATOL_i$ to the value at which $|y_i|$ is essentially insignificant (e.g., $RTOL_i.\max|y_i|$). In this way differently scaled components of a PDE can be solved in an equally accurate way without the need to scale the problem.

A notion that often turns up in connection with iterative linear system solvers is the *scaling invariance* of the method or a stopping criterion. This means that the result when solving $\gamma Ax = \gamma b$ should be independent of γ ($\gamma \neq 0$). However, it is also possible that the equations are scaled differently per component, i.e., $\Gamma Ax = \Gamma b$ with Γ a diagonal matrix with different diagonal elements. This occurs for instance in the discretization of a time-dependent PDE where the, time-independent, boundary equations can be scaled in different ways.

In Section 4.4 we will discuss how to cope with these two scaling problems in the iterative solution of the linear systems.

4.3. The influence of the linear system solution error on the modified Newton process

When solving the nonlinear system $F(y^*) = 0$ with modified Newton we get the iteration process

$$G.c^{(k)} \stackrel{\text{def}}{=} G.(y^{(k)} - y^{(k-1)}) = -F(y^{(k-1)}), \tag{4.6}$$

with G the Jacobian in some old solution value y_{old}

$$G = F'(y_{old}) = \frac{\partial F}{\partial y}(y_{old}). \tag{4.7}$$

To determine the stopping criterion for the linear solver when solving (4.6) we wish to develop a relation between the error made by the linear system solver and the accuracy desired of the solution of the modified Newton process. It is clear that it is not needed to solve each linear system exactly, but on the other hand we do not want to disturb the Newton process by returning an approximation that is not accurate enough. We therefore will examine the influence of the error made in the linear system solver on the modified Newton process.

Supposing that the linear systems are solved exactly, the error in the k-th iterate satisfies

$$\begin{aligned} \delta^{(k)} &\stackrel{\text{def}}{=} y^* - y^{(k)} = \delta^{(k-1)} - G^{-1}(F(y^*) - F(y^{(k-1)}) \\ &= \left(I - G^{-1}F'(y^*)\right)\delta^{(k-1)} + O((\delta^{(k-1)})^2). \end{aligned} \tag{4.8}$$

Neglecting the $O((\delta^{(k-1)})^2)$ term and using the notation

$$M \stackrel{\text{def}}{=} \left(I - G^{-1}F'(y^*)\right), \tag{4.9}$$

the k-th iteration error can be expressed in the initial error $\delta^{(0)}$

$$\delta^{(k)} = M^k\delta^{(0)}. \tag{4.10}$$

If the linear system $G.c^{(k)} = -F(y^{(k-1)})$ is not exactly solved but up to a certain precision

$$\epsilon^{(k)} \stackrel{\text{def}}{=} c^{(k)} - c^{(k)^{(l)}} \gg O((\delta^{(k-1)})^2) \tag{4.11}$$

the error will look like

$$\begin{array}{lcl} \tilde{\delta}^{(0)} & = & \delta^{(0)} \\ \tilde{\delta}^{(1)} & = & M\tilde{\delta}^{(0)} + \epsilon^{(1)} = M\delta^{(0)} + \epsilon^{(1)} \\ \tilde{\delta}^{(2)} & = & M\tilde{\delta}^{(1)} + \epsilon^{(2)} = M^2\delta^{(0)} + M\epsilon^{(1)} + \epsilon^{(2)} \\ \vdots & & \vdots \\ \tilde{\delta}^{(k)} & = & M^k\delta^{(0)} + \sum_{j=1}^{k} M^{k-j}\epsilon^{(j)} \\ & = & \delta^{(k)} + \sum_{j=1}^{k} M^{k-j}\epsilon^{(j)}. \end{array} \tag{4.12}$$

In order that the Newton process be convergent M has to be a contracting operator

$$\|Mv\| < \|v\|. \tag{4.13}$$

Now, if we solve the linear system up to

$$\|\epsilon^{(j)}\|_w < TOL_{LSS}, \tag{4.14}$$

we get

$$\begin{array}{lcl} \|\tilde{\delta}^{(k)}\|_w & \leq & \|\delta^{(k)}\|_w + \sum_{j=1}^{k} \|M^{k-j}\epsilon^{(j)}\|_w \\ & \leq & \|\delta^{(k)}\|_w + k\, TOL_{LSS}. \end{array} \tag{4.15}$$

Since DASPK continues the Newton iteration until $\|\tilde{\delta}^{(k)}\|_w < 0.33$, a choice of, say,

$$TOL_{LSS} = \frac{0.03}{max.\_\#\_Newton\_it.}, \tag{4.16}$$

implies that the error we make in solving the linear system will have only a negligible influence on the Newton process. Because of the inequality

$$\|\delta^{(k)}\|_w \leq \|\tilde{\delta}^{(k)}\|_w + k\, TOL_{LSS} \tag{4.17}$$

the ultimate accuracy of the Newton process too will not be violated much.

4.4. Stopping criteria for the iterative linear solvers

When solving the linear system (4.6) with the iterative solvers in the SLAP-package the breakdown and stopping criteria need some adaptation because they are obviously not intended for solving subsequent iterations in a Newton process arising from a system of time-dependent PDEs. Omitting the iteration index (k) in (4.6) the linear system we want to solve is

$$G.c^* = -F. \tag{4.18}$$

The stopping criteria used in the SLAP-package are all relative to the right hand-side vector of the linear system, e.g.,

$$\frac{\|r^{(l)}\|_2}{\|F\|_2} < TOL, \tag{4.19}$$

where $r^{(l)} = -F - G.c^{(l)}$ and $c^{(l)}$ is the l-th iterate generated by the solver. According to Dembo e.a.[2] this is a natural stopping rule and they show that with an appropriate choice of TOL (dependent on the iteration index k of the Newton process) the convergence rate of the total process will not be influenced. Moreover this stopping criterion satisfies the second one of the scaling problems above, viz., it is indeed scaling invariant. However, the maximum number of Newton iterations allowed is in ODE codes very small, say 3 or 4, and it is not so much the convergence rate we are interested in as well as the accuracy with which the linear system is solved. Upon Newton convergence $F(y^{(k-1)}) \to 0$ and therefore a stopping rule like (4.19) will result in an unnecessarily stringent bound on the residual and thus in a lot of iterations of the iterative solver. Of course one could reformulate the linear system that is to be solved in the k-th Newton iterate from

$$G.c^{(k)} = -F(y^{(k-1)}) \tag{4.20}$$

into

$$G.y^{(k)} = G.y^{(k-1)} - F(y^{(k-1)}). \tag{4.21}$$

But then the iterative solver will have difficulties if the solution of the PDE is zero. Moreover it is not possible to resolve the first scaling problem mentioned in the previous subsection. We therefore will develop an alternative stopping criterion.

The precision with which we want to solve the linear system $G.c^* = -F$ is

$$\|\epsilon^{(l)}\|_w = \|c^* - c^{(l)}\|_w < TOL_{LSS}. \tag{4.22}$$

The following relation holds for the weighted root mean square norm

$$\|Ax\|_w \stackrel{\text{def}}{=} \frac{1}{\sqrt{N}}\|W^{-1}Ax\|_2 \leq \frac{1}{\sqrt{N}}\|W^{-1}AW\|_2\|W^{-1}x\|_2 = \|W^{-1}AW\|_2\|x\|_w, \tag{4.23}$$

where W is a diagonal matrix with (nonzero) elements w_i defined in (4.2).

Now, let K be the ILU decomposition of G and $r^{(l)}$ the residual of the linear system after l iterations of the iterative solver. Then

$$\begin{aligned} \|c^* - c^{(l)}\|_w &= \|(K^{-1}G)^{-1}\,(K^{-1}r^{(l)})\|_w \\ &\leq \|W^{-1}(K^{-1}G)^{-1}W\|_2\,\|K^{-1}r^{(l)}\|_w \\ &\leq \kappa_2(W)\,\sigma_{max}((K^{-1}G)^{-1})\,\|K^{-1}r^{(l)}\|_w, \end{aligned} \tag{4.24}$$

where σ_{max} is the largest singular value and $\kappa_2(W)$ is the condition number, i.e., the quotient of the largest and the smallest entry of W. Therefore if we continue the iteration until

$$\|K^{-1}r^{(l)}\|_w < TOL_{LSS}\frac{\sigma_{min}(K^{-1}G)}{\kappa_2(W)} \tag{4.25}$$

the linear system is accurately enough solved. Note, that the scaling problems are satisfied using this stopping criterion; the first one because we use the weighted norm and the second because the scaling of the residual equations is cancelled since the factors are both in K and in r.

For unscaled problems the condition number of W can be very large and (4.25) could result in a possibly too pessimistic bound. Another possibility is to solve the system

$$(W^{-1}\,K^{-1}G\,W)\,(W^{-1}c^*) = -(W^{-1}K^{-1}F), \tag{4.26}$$

with as stopping criterion

$$\|K^{-1}r^{(l)}\|_w < TOL_{LSS}.\sigma_{min}(W^{-1}\,K^{-1}G\,W). \tag{4.27}$$

The implementation of GMRES in SLAP allows these extra diagonal scaling matrices and the CGS-type algorithms could be easily adapted to this aim. However, if $K^{-1}G$ is not a symmetric matrix the condition of the linear system will be probably worse than without diagonal scaling. Therefore, it is in our opinion better to scale the independent variables of the PDE such that $\kappa_2(W) \approx 1$ and use (4.25) as stopping criterion.

Since the ILU preconditioning should hopefully result in a value of $\sigma_{min}(K^{-1}G)$ close to 1, we have chosen to actually implement the simpler stopping criterion

$$\|K^{-1}r^{(l)}\|_w < TOL = \frac{TOL_{LSS}}{\kappa_2(W)}. \tag{4.28}$$

	Data storage	
	integer	floating-point
SGBFA/SL	N	$6.Nx.NPDE.N$
Y12M	$11.N + 2.LUF + 4.NZ$	$LUF + N + 2.NZ$
GMRES(m)	$3.NZ + 4.N$	$2.NZ + (m+8).N$
CGS	$3.NZ + 4.N$	$2.NZ + 8.N$
BI-CGSTAB	$3.NZ + 4.N$	$2.NZ + 8.N$

TABLE 1. Storage requirements for the linear solvers.
$LUF = ((2.Nx + 5).NPDE + 2).N + 2.NZ.$

5. NUMERICAL RESULTS

5.1. Data storage estimates

To have a good real-time performance when solving a real-life problem, memory is nowadays often a more restrictive factor than CPU power. Most workstations have very powerful CPUs sometimes even vector processors. However if the job to be executed does not fit into memory the computational data will be swapped from memory to disk and vice versa. In particular, if the disk is allocatable only via a network, swapping will be by far the dominating factor in the turn-around time. Therefore not only computational complexity of an algorithm is of importance but also the amount of memory needed.

Below we will give the amount of data storage needed for each linear solver. For a large number of grid points this will be the bulk of the memory requirements. Suppose the spatial domain Ω is a rectangle covered with a uniform grid with Nx points in the X-direction and Ny points in the Y-direction. The order of the linear system is then $N = Nx.Ny.NPDE$, where $NPDE$ is the number of PDEs (in our brine flow model $NPDE = 2$).

The bandsolver SGBFA/SL requires the largest amount of memory. The integer storage is neglectable (N), but the floating-point (FP) storage is $3.lbw.N$ where lbw is the half-bandwidth of the Jacobian. If we sort the grid points along X-lines $lbw = 2.Nx.NPDE$. Note that the factor 2 arises from the second order discretization of the boundary equations. All other solvers need for the storage of the Jacobian $2.NZ$ integer words and NZ FP words, where $NZ = 9.NPDE.N$ is the number of nonzeros of the Jacobian since we use a 9-point stencil. The amount of memory needed for the sparse solver is not so clear. The documentation of Y12M suggests for the integer amount $11.N + 8.NZ$ and for the FP data $5.NZ + N$ inclusive the storage needed for the Jacobian. It is obvious however that this will not be enough to store the LU-decomposition alone, since for our uniform grid the inner band, with a bandwidth of $2.(Nx + 1).NPDE + 1$, will be completely filled. We therefore estimated the integer amount to be $11.N + 2.LUF$ and the FP amount to be $LUF + N$ where LUF is the fill-in plus some elbow room, i.e., $LUF = ((2.Nx+5).NPDE+2).N+2.NZ$. The iterative solvers will take much less storage space, all $O(N)$ plus the storage needed for the Jacobian and the ILU decomposition $O(NZ)$. For a complete overview we refer to Table 1.

5.2. CPU timings

We discretized our problem (2.1) on an 81x81 grid and used a time-stepping tolerance of 1E-3, i.e., in DASPK we have $ATOL_i = RTOL_i = 10^{-3}$ for all components. In our first test we executed the automatically vectorized code on one processor of a Cray-YMP. We did not add any vector directives to the code but note that the SLAP library is supplied by the authors with vector directives for a Cray and the bandsolver SGBFA/SL is part of the LINPACK library which is in optimal vectorized form available on the Cray-YMP. For the results of this test we refer to Table 2, which contains integration history information accumulated over the complete integration interval. Because we are only interested here in the performance of the various linear solvers, we refrain from discussing the specific details with respect to the time integration. Note that the average number of iterations used by the iterative linear solvers is LI/NI.

	Steps	ETF	CFN	NI	LI	Jacs	CPUsec	Data storage (Mw) integer	FP
SGBFA/SL	85	3	6	174	-	51	1433	.01	13
Y12M	-	-	-	-	-	-	-	10	5
GMRES(10)	86	4	8	177	2542	55	488	.8	.7
GMRES(30)	80	3	7	178	2341	52	461	.8	1.0
GMRES(50)	81	3	8	185	2360	55	480	.8	1.2
CGS	78	4	4	151	2188	40	498	.8	.6
BI-CGSTAB	81	4	4	158	1196	41	385	.8	.6

TABLE 2. Results for the automatically vectorized code on 1 processor of a Cray-YMP.
Steps: # successful steps NI: # Newton iterations
ETF: # time step failures LI: # linear iterations
CFN: # Newton failures Jacs: # Jacobian evaluations

Results for the sparse matrix solver are absent. The reason for this is that a test run on a 31x31 grid took already more CPU-time than the band code on an 81x81 grid. This is very likely since the sparse matrix code does not vectorize very well, in contrast to the band solver. It is obvious that the iterative solvers and especially BI-CGSTAB perform well. However, the difference in efficiency between the bandsolver and the iterative solvers stems largely from the fact that in the latter case we solve the linear systems only approximately and use an improved stopping criterion. If we require an 'exact' solution of the linear systems and if we use the stopping criteria recommended by the authors, the CPU-time for the CGS-type solvers and GMRES(50) is approximately 1200 CPU-seconds and for GMRES(30) 2700. GMRES(10) does not converge in the very first time step. In this case the difference between CGS and BI-CGSTAB is less striking. Finally, inspection of the residual errors yields that for our groundwater flow problem the stabilizing influence of polynomial (3.9) is very noticeable in the first iterations, where the CGS residuals had 'peaks' of a factor 10^5 in comparison with the surrounding residual values. Such peaks did not occur with BI-CGSTAB.

We planned to execute these tests also in scalar mode on a workstation. But due to the limited amount of memory (16 Mbytes) on our workstations this would involve swapping and hence take a too large turn-around time for the direct solvers. Therefore our timings with the scalar code were obtained on an IBM ES/9000 with a central memory of 256 Mbytes. In this case again Y12M was so much slower than the bandsolver that we did not want to include the timings for a full run. For the precise results we refer to Table 3. We attribute the slight difference in time-stepping behavior to the different FP representation on both machines.

Note that in the scalar case the bandsolver is, as expected, much slower than the iterative solvers since the bandsolver vectorizes notably better than the ILU preconditioner. For uniform grids it is of course possible (but complicated because of the $9.NPDE$ diagonals) to vectorize the ILU preconditioner. For the linear systems arizing from a locally-uniform-grid-refinement method however, this is no longer feasible because of the irregular band structure. With respect to the performance on a workstation one should bear in mind that the real time for the bandsolver to finish this job will presumably be in the range of one week, due to the permanent need of memory-swap.

When the problem is 'small', as for instance a 2D scalar convection-diffusion equation with a relative small number of nodes, the sparse matrix solver appeared to be rather efficient, but for system (2.1) and a large number of nodes our results indicated that it is very inefficient to use Y12M. Of course, there may exist more efficient sparse matrix solvers, but one can not expect too much of these because ultimately half the bandwidth will be filled with nonzeros.

Since the objective of this paper was to compare the various linear system solvers in an adaptive grid environment, we will here comment briefly on the relevant differences between the uniform grid approach and the LUGR approach. In the LUGR approach the bandsolver will perform more or less

	Steps	ETF	CFN	NI	LI	Jacs	CPUsec
SGBFA/SL	84	3	5	165	-	44	48772
GMRES(10)	78	3	5	157	2321	44	956
GMRES(30)	87	3	7	177	2198	53	1072
GMRES(50)	87	3	7	177	2158	53	1068
CGS	87	4	7	173	2315	52	1352
BI-CGSTAB	89	5	7	184	1329	54	959

TABLE 3. Results for the scalar code on an IBM ES/9000.
Steps: # successful steps NI: # Newton iterations
ETF: # time step failures LI: # linear iterations
CFN: # Newton failures Jacs: # Jacobian evaluations

as demonstrated here since the bandwidth stays the same and only the dimension of the system is decreased. A sparse matrix solver can benefit from the fact that computations take place only on a small part of the grid, since an irregular bandstructure is generated and so the fill-in would possibly be restricted. Iterative solvers will perform notably better in an adaptive grid environment since, owing to their decreasing dimension, the condition numbers of the LUGR linear systems are readily much less than the condition number of the linear system arising from a global uniform approach (cf. [18]).

6. CONCLUSIONS AND FUTURE PLANS

In this paper we have tested various methods for solving the linear systems that arise when a 2D fluid-flow salt transport problem like (2.1) is solved in time with an implicit integrator. We used so called *direct* methods, viz., a bandsolver and a sparse solver, and *iterative* methods, viz., GMRES, CGS and BI-CGSTAB. A clear disadvantage of direct methods is the large amount of memory needed to store the factors of the matrix. However, on a vector computer bandsolvers are, even though they perform a lot of void operations, still relatively fast because the code vectorizes well. For systems like (2.1) a sparse solver appeared to be very slow, on the one hand because the number of nonzero entries per row is much too large to surpass the 'sparsity' overhead and on the other hand because even a sparse solver requires more memory to hold the data alone than a moderate workstation contains. Even if we take into account that the here used sparse solver Y12M is not state-of-the-art (e.g., it does not switch to dense matrix techniques nor uses a frontal approach on a vector machine; see, e.g., [22]), one cannot have high expectations of sparse matrix techniques for a system like (2.1) with relatively dense factors. Hence, although the LUGR method genèrates less regular bandmatrices and therefore presumably less fill-in occurs, we do not advocate a sparse solver to solve the arising linear systems.

The iterative solvers appear to be by far preferable, both with respect to CPU-time and with respect to the amount of memory. In this paper we used as preconditioner the standard ILU decomposition which resulted in very few linear iterations, but which is presumably not the best possible choice on a vector computer, because for irregular matrices the ILU preconditioner does not vectorize well. In the future we plan to look after better vectorizable preconditioners, e.g., polynomial preconditioners. We also plan to test the recently developed GMRESR method (van der Vorst and Vuik[21]), that has so to say its own build-in preconditioner. If it would be possible to use a preconditioner that is independent of the Jacobian of the Newton system, we plan to look also at the *matrix-free* Newton process, to spare both the numerical computation of the Jacobian and the storage needed.

ACKNOWLEDGEMENTS

We thank Linda Petzold for placing DASPK at our disposal.

References

[1] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* North-Holland, New-York, 1989.

[2] R.S. Dembo, S.C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[3] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, Philadelphia, 1991.

[4] J.J. Dongarra and E. Grosse. Distribution of mathematical software via electronic mail. *Commun. ACM*, 30:403–407, 1987. (netlib@research.att.com).

[5] J.J. Dongarra, C.B. Moler, and G.W. Stewart J.R. Bunch. *LINPACK Users' Guide.* SIAM, Philadelphia, 1979.

[6] I.S. Duff. MA28: A set of Fortran subroutines for sparse unsymmetric linear equations. AERE Rep. R.8730, HMSO, London, 1977.

[7] I.S. Duff and J.K. Reid. Some design features of a sparse matrix code. *ACM Trans. Math. Softw.*, 5:18–35, 1979.

[8] J.C.H. van Eijkeren and S.M. Hassanizadeh, 1991. Private Communication.

[9] R. Fletcher. Conjugate gradient methods for indefinite systems. In *Lecture Notes in Mathematics 506*, pages 73–89. Springer-Verlag, Berlin, Heidelberg, New York, 1976.

[10] G.H. Golub and C.F. van Loan. *Matrix Computations (second edition).* The John Hopkins University Press, Baltimore and London, 1989.

[11] S.M. Hassanizadeh and R.A. Trompert, 1992. Private Communication.

[12] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[13] Swedish Nuclear Power Inspectorate (SKI), editor. *The International HYDROCOIN Project. Level 1: Code Verification.* NEA, OECD, Paris, 1988.

[14] P. Sonneveld. CGS: A fast Lanczos-type solver for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 10:36–52, 1989.

[15] R.A. Trompert and J.G. Verwer. Analysis of the implicit Euler local uniform grid refinement method. Report NM-R9011, CWI, Amsterdam, 1990. (to appear in SIAM J. Sci. Stat. Comput.).

[16] R.A. Trompert and J.G. Verwer. Runge-Kutta methods and local uniform grid refinement. Report NM-R9022, CWI, Amsterdam, 1990. (submitted to Math. Comp.).

[17] R.A. Trompert and J.G. Verwer. A static-regridding method for two-dimensional parabolic partial differential equations. *Appl. Numer. Math.*, 8:65–90, 1991.

[18] R.A. Trompert, J.G. Verwer, and J.G. Blom. Computing brine transport in porous media with an adaptive-grid method. Report NM-R9201, CWI, Amsterdam, 1992.

[19] J.G. Verwer and R.A. Trompert. Local uniform grid refinement for time-dependent partial differential equations. Report NM-R9105, CWI, Amsterdam, 1991. (Paper presented at the 14th Biennial Conference on Numerical Analysis, 25th–28th June 1991, Dundee, Scotland).

[20] H.A. van der Vorst. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2), 1992.

[21] H.A. van der Vorst and C. Vuik. GMRESR: A family of nested GMRES methods. Report 91-80, Faculty of Technical Mathematics and Informatics, TU Delft, the Netherlands, 1991.

[22] Z. Zlatev. *Computational Methods for General Sparse Matrices*, volume 65 of *Mathematics and Its Applications*. Kluwer Academic Publishers, Dordrecht-Boston-London, 1991.