Further experiences with GMRESR

Report 92-12

C. Vuik

TU Delft
Technische Universiteit Delft
Delft University of Technology

Faculteit der Technische Wiskunde en Informatica
Faculty of Technical Mathematics and Informatics

ISSN 0922-5641

Copies of these reports may be obtained from the bureau of the Faculty of Technical Mathematics and Informatics, Julianalaan 132, 2628 BL Delft, phone +3115784568.
A selection of these reports is available in PostScript form at the Faculty's anonymous ftp-site. They are located in the directory /pub/publications/tech-reports at ftp.twi.tudelft.nl

Further experiences with GMRESR

C. Vuik
Faculty of Technical Mathematics and Informatics
Delft University of Technology
Mekelweg 4, Delft
The Netherlands

Abstract

The GMRES method proposed in [8] is a popular method for the iterative solution of sparse linear systems with an unsymmetric nonsingular matrix. We propose in [13] a variant of the GMRES algorithm, GMRESR, in which it is allowed to take a different preconditioner in each iteration step. Here some properties of this approach are discussed and illustrated with numerical experiments. We compare GMRESR with GMRES, Bi-CGSTAB [11] and FGMRES [7].

Key words. GMRES, nonsymmetric linear system, iterative solver.

AMS(MOS) subject classifications. 65F10

1 Definitions and properties of the GMRESR method. We specify the GMRESR method given in [13] and summarize some of its properties.

1.1 Definitions. In this subsection we give the GMRESR method [13]. The new idea behind the GMRESR method is to use a preconditioning which may be different for each iteration step. We obtain the preconditioning by carrying out a number of GMRES steps in what may be regarded as an inner loop. However, any other process, which approximates $A^{-1}y$ for a given vector y, can be chosen (e.g. LSQR [6], Bi-CGSTAB [11], QMR [3] or GMRESR). Furthermore, the choice of the process used in the innerloop may be different in each iteration. Note that one can use GMRESR in a recursive way, which motivates the name of the method GMRESR(ecursive).

We denote the approximate solution of $A^{-1}r$ by $P_m(A)r$, where P_m represents the GMRES polynomial that is implicitly constructed in m iteration steps of GMRES. Note that this polynomial depends on the residual r, so that we have effectively different polynomials in different steps of the outer iteration. We will make this dependence explicit by adding the number of the current outer iteration as an index to $P : P_{m,k}(A)r_k$. The resulting process, GMRESR, is represented by the following iteration scheme for the solution of $Ax = b$:

GMRESR algorithm ([13])

1. Start: Select x_0, m, eps;
$r_0 = b - Ax_0$, $k = -1$;

2. Iterate: while $\|r_{k+1}\|_2 > eps$ do
$k := k+1,\ \ u_k^{(0)} = P_{m,k}(A)r_k,\ \ c_k^{(0)} = Au_k^{(0)};$
for $i = 0, ..., k-1$ do
$\alpha_i = c_i^T c_k^{(i)},\ \ c_k^{(i+1)} = c_k^{(i)} - \alpha_i c_i,\ \ u_k^{(i+1)} = u_k^{(i)} - \alpha_i u_i;$
endfor
$c_k = c_k^{(k)}/\|c_k^{(k)}\|_2;\ \ u_k = u_k^{(k)}/\|c_k^{(k)}\|_2;$
$x_{k+1} = x_k + u_k c_k^T r_k;$
$r_{k+1} = r_k - c_k c_k^T r_k;$
endwhile.

Note: In the remainder of this paper, the process to calculate $u_k^{(0)}$ is called the innerloop of the GMRESR method. If GMRES in the innerloop stagnates we obtain $u_k^{(0)} = 0$ and the method breaks down. In such a case we avoid break down by using one step of LSQR [6] in the innerloop: $u_k^{(0)} = A^T r_k$. In Section 6 we give a motivation for this choice and specify some examples where the LSQR switch gives a much better convergence behaviour.

A more practical scheme, in our opinion, arises when the outer iteration is restarted after ls iterations, in order to limit memory requirements, or to include only updates from the last lt outer iterations (a truncated GMRESR version). The resulting scheme is denoted by GMRESR (ls, lt, m). In Section 3 we give other truncation strategies and compare restarting and truncation for some testproblems.

1.2 Properties. In this subsection we summarize some properties of GMRESR without restarting or truncation. This method is denoted by GMRESR(m). For the proofs of these properties we refer to [13]. We assume that the inner iteration is always started with initial guess $u_{k,0}^{(0)} = 0$ (note that in this notation $u_k^{(0)} = u_{k,m}^{(0)} = P_{m,k}(A)r_k$). In the GMRESR algorithm c_k is undefined if $c_k^{(k)} = 0$. If this happens, while $r_k \neq 0$, then we will speak of a breakdown of GMRESR.

The next theorem says that GMRESR(m) is a robust method, and that it is a minimum residual method.

Theorem 1.1

(a) GMRESR(m) does not break down,

(b) In GMRESR(m) the residual r_k is minimized over the space

$$r_0 + span\{c_0, c_1, ..., c_{k-1}\}.$$

It is easily seen that Theorem 1.1 (b) implies that $r_k = 0$, for some $k \leq n$, so GMRESR is a finite method.

In [13] we show that the norm of the k-th GMRESR(m) residual is larger than or equal to the norm of the full GMRES residual after $k \cdot m$ iterations. This implies that GMRESR takes at least as many matrix vector products as full GMRES in order to obtain comparable accuracy.

The following lemma states that it is never necessary to compute the inner approximation more accurately than the outer one.

Lemma 1.2

If the norm of the j-th innerloop residual $\|r_k - Au_{k,j}^{(0)}\|_2$ is less than eps, where $j \leq m$, then it is proved in [13] that using $u_k^{(0)} = u_{k,j}^{(0)}$, the outer loop residual satisfies $\|r_{k+1}\|_2 < eps$.

From this lemma it follows that it is not necessary to apply m iterations in the final inner loop. If the innerloop residual satisfies the stopping criterion for $j < m$ one can stop the inner iteration process and use the approximation $u_{k,j}^{(0)}$ of $A^{-1}r_k$ as the final search direction. In general this saves some CPU time.

1.3 The choice of m. In order to compare the efficiency of GMRES and GMRESR(m), estimates for the amount of work and the required memory of both methods are listed in Table 1.

method	GMRES	GMRESR(m)
steps	m_g	m_{gr}
matvec	m_g	$m_{gr} \cdot m$
vector updates	$\frac{1}{2}m_g^2$	$m_{gr} \cdot (\frac{m^2}{2} + m_{gr})$
inner products	$\frac{1}{2}m_g^2$	$m_{gr} \cdot (\frac{m^2}{2} + \frac{m_{gr}}{2})$
memory vectors	m_g	$2m_{gr} + m$

Table 1: Amount of work and memory for GMRES and GMRESR(m)

It appears from our numerical experiments that in many cases $m_{gr} \cdot m$ is approximately equal to m_g. This observation is used to derive optimal choices for m with respect to work and required memory. In the following, we assume that a vector update costs as much as an inner product. Using $m_{gr} = m_g/m$ it appears that the minimal amount of work is attained for $m = \sqrt[3]{3m_g}$, and it is less than $2.5\ m_g^{4/3}$. Note that the amount of work of GMRES is equal to m_g^2. With respect to memory requirements the optimal value is equal to $m = \sqrt{2m_g}$, so the amount of memory for GMRESR(m) is equal to $2\sqrt{2m_g}$. This combined with Table 1 implies that for large value of m_g, GMRESR(m) needs much less memory than GMRES. Both optimal values of m are slowly varying with m_g. Thus a given m is near-optimal for a wide range of values of m_g. Note that the optimal m with respect to work is in general less than the optimal m with respect to memory. It depends on the problem and the available computer, which value is preferred. In our experiments we observe that for both choices the amount of work and required memory is much less than for GMRES.

2 Numerical experiments. In this section we illustrate the theoretical properties of GMRESR with experiments.

In our numerical experiments we use a linear system obtained from a discretization of the following *pde*:

$$-(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + \beta(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}) = f \text{ on } \Omega,$$

$$u|_{\partial\Omega} = 0,$$

where Ω is the unit square. The exact solution u is given by $u(x,y) = \sin(\pi x)\sin(\pi y)$. In the discretization we use the standard five point central finite difference approximation. The stepsizes in x- and y-direction are equal. We use the following iterative methods: GMRES(m), Bi-CGSTAB, and GMRESR(m). We use a more or less optimal choice of m to obtain results using GMRES(m). We start with $x_0 = 0$ and stop if $\|r_k\|_2/\|r_0\|_2 \leq 10^{-12}$.

In Table 2 we present results for GMRESR(m). CPU time is measured in seconds using 1 processor of a Convex $C-3820$. Using full GMRES we observe that $m_g = 183$, which means, that one needs 183 vectors in memory. Furthermore full GMRES costs

m	4	8	12	16	20
m_{gr}	47	25	19	16	14
$m \cdot m_{gr}$	188	200	228	256	280
CPU time	0.82	0.57	0.68	0.83	1.01
memory vectors	98	58	50	48	48

Table 2: The results for GMRESR(m) with $\beta = 1$ and $h = 1/50$.

$4.4s$ CPU time. Note that for small values of $m, m_{gr} \cdot m$ is approximately equal to m_g. Suppose m_g is unknown and we use $m_{gr} \cdot m$ as an approximation of m_g. Then using the formulas given in Section 1.3, we obtain the following bounds for the optimal values of m:

work: $\quad \sqrt[3]{3.188} = 8.3 < m < 9.4 = \sqrt[3]{3.280}$,

memory: $\sqrt{2.188} = 19.4 < m < 23.7 = \sqrt{2.280}$.

Comparing this with Table 2 we note that there is a good correspondence between theory and experiments in this example. As expected, the optimal value of m with respect to memory is larger than with respect to work. However, for both choices of m we observe a considerable gain in computing time and memory requirements.

Results comparing the three iterative methods are shown in Table 3. It appears that

method	iterations	matvec	CPU time
GMRES(32)	1355	1355	26
Bi-CGSTAB	237	474	1.7
GMRESR(10)	36	360	4.3

Table 3: Results for $\beta = 1$ and $h = 1/100$.

GMRESR(10) is better than GMRES(32). Although Bi-CGSTAB uses less CPU time, it uses more matrix vector products than GMRESR(10).

In Table 4 we take the stepsize $h = 1/100$ and β a function of x and y as follows:

$$\beta(x,y) = \begin{cases} 1 & \text{for } x,y \in [\frac{1}{2},\frac{3}{5}]^2 \\ 1000 & \text{for } x,y \in [0,1]^2 \setminus [\frac{1}{2},\frac{3}{5}]^2. \end{cases} \tag{1}$$

method	iterations	matvec	CPU time
GMRES(32)	1536	1536	30
Bi-CGSTAB	n.c.		
GMRESR(10)	56	560	7.8

Table 4: Results, with β given by (1) and $h = 1/100$.

Note that in this problem GMRESR(10) is the best method.

The following example comes from a discretization of the incompressible Navier-Stokes equations. This discretization leads to two different linear systems, the momentum equations and the pressure equation (for a further description we refer to [13], [14]). We consider a specific testproblem, which describes the flow through a curved channel. The problem is discretized with 32×128 finite volumes. The pressure equation is solved using an average of an ILU and MILU preconditioner with $\alpha = 0.975$ [1], [14]. We start with $x_0 = 0$ and stop when $\|r_k\|_2/\|r_0\|_2 \leq 10^{-6}$. The results are shown in Table 5. Note that for this problem GMRESR(4) is the fastest method with respect to CPU time.

method	iterations	matvec	CPU time
full GMRES	47	47	0.77
CGS	38	76	0.49
Bi-CGSTAB	34	68	0.44
GMRESR(4)	12	48	0.43

Table 5: Iterative methods applied to the pressure equation.

3 Restarting and truncation strategies. We present some truncation strategies and compare the results with restarted and full GMRESR.

There are many different ways to truncate GMRESR. The first one, which is already given in Section 1.3, is to use the lt last search directions (denoted by trunclast). To obtain another truncation strategy we note that in many cases GMRESR has a superlinear convergence behaviour. This means that after some iterations GMRESR converges as if some eigenvalues of the matrix are absent (compare [12]). Restarting or truncation can destroy this behaviour ([4]; pp. 1334,1335). If superlinear convergence occurs, it appears a good idea to use the $lt - 1$ first and 1 last search directions (denoted by truncfirst). Both strategies are used in the following experiments. It follows from Table 6 that truncation with $lt - 1$ first and 1 last search directions is the best strategy for this example. If there are only 18 memory vectors available, the gain in CPU time for $ls = 50, lt = 5$ with respect to $ls = lt = 5$ is equal to 40%. Furthermore, comparing full GMRESR(8) with GMRESR(50,10,8) (truncfirst variant) we see that the amounts of CPU time are approximately the same, whereas the amount of memory is halved.

We conclude this section with some other truncation strategies. First we note that it seems an awkward choice to use one last search direction in the truncfirst variant.

$lt = ls$	restart		$ls = 50$	trunclast		truncfirst		memory
	iterations	CPU	lt	iterations	CPU	iterations	CPU	vectors
5	57	1.15	5	41	0.87	37	0.79	18
10	45	0.97	10	32	0.73	29	0.68	28
15	33	0.74	15	29	0.69	26	0.62	38
20	29	0.67	20	25	0.60	25	0.60	48
25	25	0.60	25	25	0.60	25	0.60	58

Table 6: Results with GMRESR$(ls, lt, 8)$, $\beta = 1$ and $h = 1/50$.

This choice is motivated by the fact that if one applies GCR to a symmetric problem then it is necessary and sufficient to use one last search direction in order to obtain the same convergence behaviour as full GCR. We have done experiments without this final direction (truncfirst1). These results are given in Table 7. Note that the truncfirst1 variant is the worst truncation strategy, so it is indeed a good idea to include one last search direction, which is done in the original truncfirst variant.

lt	5	10	15	20	25
truncfirst1	55	49	34	26	25
minalfa	36	28	25	25	25

Table 7: Number of iterations for GMRESR$(50, lt, 8)$, $\beta = 1$ and $h = 1/50$.

Finally in ([4]; p. 1335) another truncation strategy is proposed for a GCR-like method. For the GMRESR algorithm this strategy leads to the folowing variant (minalfa): if $k \geq lt$ then the search direction with the smallest absolute value of α_i in the for loop is thrown away. The motivation is that the search direction with the smallest $|\alpha_i|$ has only a limited influence on the GMRESR convergence. From Table 7 it appears that this leads to the best truncation strategy for this example. Another important advantage of the minalfa variant is that it is a black box strategy. For instance if the bad eigenvector components (with respect to the convergence behaviour) appear after some iterations, then the truncfirst variant is much worse than the minalfa variant.

4 **Some ideas for choosing an iterative solution method.** There are a large number of known iterative solution methods for non-symmetric problems. In this section we present some ideas to motivate a choice of a feasible iterative method. These ideas are based on our experiments. Probably they should be adapted for other classes of problems. The insights in this section can be used to guess a priori if it has sense to change from one iterative method to another. Furthermore it is shown that two parameters: the ratio of the CPU time for a matrix vector product and a vector update, and the expected number of full GMRES iterations, are important to choose an iterative method. Finally, the ideas given in this section show a good agreement with our experiments given in Section 2.

In the remainder of this section we assume that the amount of required memory is available. Otherwise restarted or truncated versions of GMRES (GMRESR) can be used, however it is not clear if the results in this section holds in such a case.

The CPU time of many iterative methods consists of two main parts:

- the total CPU time used for matrix vector products, which is denoted by t_m (if a preconditioner is used, t_m includes the time for preconditioning),
- the total CPU time used for vector updates and inner products, which is denoted by t_v.

We prefer GMRES (GMRESR) if t_v is less than γt_m for a given constant γ. Note that for every other Krylov subspace method the gain in CPU time is always less than $\frac{\gamma}{1+\gamma} \cdot 100\%$. In our experiments it appears that for the choice $\gamma = 0.5, (1+\gamma)t_m$ of GMRES (GMRESR) is approximately equal to $t_m + t_v$ of Bi-CGSTAB. So in our example at the end of this section we take $\gamma = 0.5$ (for this choice $\frac{\gamma}{1+\gamma} \cdot 100\% = 33\%$).

The CPU time used for one matrix (+ preconditioner) vector product is denoted by t_{m1}, and the CPU time of one vectorupdate (or inner product) is denoted by t_{v1}. The factor f is defined by $f = t_{m1}/t_{v1}$. Using the assumption that $m_{gr} \cdot m \cong m_g$ we obtain the following expressions:

GMRES: $\quad t_m = m_g \cdot t_{m1} \quad , \; t_v = m_g^2 t_{v1}$;

GMRESR: $\quad t_m = m_g \cdot t_{m1} \quad , \; t_v = 2.5 m_g^{4/3} t_{v1}$.

As said before we prefer GMRES (GMRESR) if $t_v \leq \gamma t_m$. For GMRES this means $m_g \leq \gamma f$, whereas for GMRESR this means $m_g \leq (\gamma f/2.5)^3$. For GMRESR the bound of this inequality is given by $m_g = (\gamma f/2.5)^3$. This equation defines γ as a slow varying function of m_g, so the total CPU time of GMRESR is a slowly varying function of m_g.

Figure 1 illustrates the given bounds for the choice $\gamma = 0.5$. We emphasize that this figure only gives qualitative information. It illustrates the dependence of the choice on f and m_g. Below we specify some applications using this information.

- For a given system and computer, f can be measured. This together with an estimate of m_g and Figure 1 gives an impression of which iterative method is feasible.
- Suppose Bi-CGSTAB is the best method for a certain problem, without preconditioning. Including preconditioning, Figure 1 suggests that GMRESR can be better for this preconditioned system, because m_g is (much) lower and f is (much) higher (in general a preconditioner is harder to vectorize than a matrix vector product).

Note that the applicability of GMRESR for large values of f is much wider than GMRES.

For the first example given in Section 3, $f = 10$, so Figure 1 agrees with our observation that Bi-CGSTAB costs more matrix vector products but less CPU time than GMRESR. In the practical examples given in Section 3 and [13], $f = 20$ and $m_g \leq 50$. In these examples the CPU time of GMRESR is less than the CPU time of Bi-CGSTAB, which is also in accordance with Figure 1.

Finally we compare GMRESR with QMR. It is easily seen from [3], equations (2.7), (2.8) and (3.1) that the QMR method uses k multiplications with A and A^T to construct a solution, which is an element of a Krylov subspace with dimension k. So we

choose $\gamma = 1$ in order to compare GMRESR and QMR (Figure 2). From Figure 2 we note that GMRESR has a large region of feasibility with respect to QMR.

5 Comparison of GMRESR with FGMRES. Another GMRES-like iteration scheme with a variable preconditioner is proposed in [7]. In Saad's scheme (FGMRES) a Krylov subspace is generated that is different from ours (GMRESR). We specify an example for which FGMRES breaks down. Comparison shows that in our examples the convergence behaviour of GMRESR and FGMRES are approximately the same. An advantage of GMRESR is that it can be truncated and/or restarted, whereas FGMRES can only be restarted. In Section 3 we have seen that in some problems truncated GMRESR converges faster than restarted GMRESR. Using such an example we show that restarted FGMRES (FGMRES can not be truncated) costs more CPU time than truncated GMRESR.

A well-known property of GMRES is, that it has no serious breakdown. From the following example we conclude that it is possible that FGMRES breaks down. For the algorithm we refer to ([7]; p.4) and note that FGMRES is equal to GMRES if $M_j = M$ for all j. So breakdown of FGMRES is only possible if one chooses different M_j.

Example

Take $A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $x = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ and $x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$. In Algorithm 2.2 of [7] we choose $M_1 = I$ and $M_2 = A^2$. These choices lead to $z_1 = z_2 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, and $h_{3,2} = 0$, which implies that v_3 does not exist. Since $x_2 = x_0 + \alpha z_1 + \beta z_2$, it follows that $x_2 \neq x$, so this is a serious breakdown. Note that $Az_2 = v_2$ but $x_2 \neq x$, which contradicts Property 1 of ([7]; p.5).

In the inner loop of GMRESR, we calculate an approximate solution of $Au_k^{(0)} = r_k$. In FGMRES an approximation of $Az_k = v_k$ is calculated. If the preconditioner is the same for every k, then u_i and z_i, $i = 0, ..., k$ span the same Krylov subspace. However, if the preconditioner varies, the Krylov subspaces can be different. To illustrate this we calculate the solution of the problem given in Section 3, with $\beta = 1$ and $h = 1/50$. As inner loop we take one step of GMRES(10) in both methods. The results are given in Figure 3. As expected $\|r_o\|_2$ and $\|r_1\|_2$ are the same for both methods. In this example the differences of the norms of the residuals are small. We have also done experiments with the same search directions in both methods. In these experiments the results of GMRESR and FGMRES are the same to machine precision.

It follows from Algorithm 2.2 [7] that only the vectors v_k are updated in the orthogonalization process. Assuming that GMRESR and FGMRES use both m_{gr} iterations for convergence, GMRESR needs $\frac{1}{2}m_{gr}^2$ vectorupdates more than FGMRES. Note that GMRESR and FGMRES are feasible for relatively large values of f (see Section 4 for the definition of f). In this case the CPU time of $\frac{1}{2}m_{gr}^2$ extra vectorupdates is negli-

gible.

FGMRES			GMRESR (truncfirst)			memory
ls	iterations	CPU	lt	iterations	CPU	vectors
5	128	12.3	5	64	6.5	20
10	83	8.2	10	46	4.9	30
15	68	6.9	15	41	4.7	40
20	59	6.1	20	41	4.8	50
25	50	5.3	25	39	4.7	60

Table 8: Results with FGMRES($ls, 10$) and GMRESR(50,$lt, 10$),$\beta = 1$ and $h = 1/100$.

Finally we compare restarted FGMRES and truncated GMRESR. As we already note it is impossible to truncate FGMRES. In Table 8 we give results for both methods, for $\beta = 1$ and $h = 1/100$. As inner loop we use one step of GMRES(10) for both methods. For this example FGMRES is more expensive than GMRESR. If there is only a small number of memory vectors available (≤ 20), then FGMRES uses 2 times as many iterations and 2 times as much CPU-time.

6 Recent results. In this section we give some recent results, which are subject to further study. First, we give some experiences with the LSQR switch for an indefinite system of equations. Thereafter we report some experiments with GMRESR, in which we use a single precision innerloop.

6.1 The LSQR switch. First we give a motivation of the LSQR switch. Thereafter we give a problem where the convergence of GMRESR is much faster using the LSQR switch.

We use the LSQR switch in the innerloop in the case that GMRES (nearly) stagnates. Due to the optimality property of GMRES it is easily seen that every other Krylov subspace method based on $K_k(A, r_0) := span\{r_0, Ar_0, ..., A^{k-1}r_0\}$ stagnates or breaks down. However it is possible that LSQR, which is based on $K_k(A^TA, A^Tr_0)$, converges reasonably fast. Examples of such problems are given in [2] and [5]. The idea is that GMRESR with LSQR switch not only works if GMRES has a good convergence behaviour but also if GMRES stagnates and LSQR converges reasonably fast. Furthermore if GMRES stagnates after some iterations it is not necessary to restart with another iterative method, it is sufficient to change the iterative method in the innerloop (for instance LSQR).

In [13] we propose to relax the LSQR switch condition, instead of switching when $\|Au^{(0)}_{k,m} - r_k\|_2 = \|r_k\|_2$, we switch when $\|Au^{(0)}_{k,m} - r_k\|_2 \geq s\|r_k\|_2$, $s \leq 1$ for some suitable s close to 1. Below we describe some other GMRESR variants using a relaxed switch condition. We compare them for a discretization of the following *pde*:

$$-(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + \beta(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}) - 100u = f \text{ on } \Omega,$$

$$u|_{\partial\Omega} = 0,$$

where Ω is the unit square. The discretization is the same as the one used for the similar testproblem given in Section 2. The linear system can be indefinite due to the

final term on the left hand side of the *pde*. We take $\beta = 10$, $h = 1/10$ and use the trunclast variant with $lt = 10$.

We consider the following GMRESR variants:

GMRESR(5):	innerloop consists of GMRES(5), combined with the strict LSQR switch ($s = 1$),
GMRESR1(5):	innerloop consists of GMRES(5) followed by one LSQR iteration,
GMRESR2(5):	innerloop consists of one LSQR iteration followed by GMRES(5),
GMRESR3(5):	innerloop consists of GMRES(5), combined with a relaxed LSQR switch ($s = 0.99$),
GMRESR4(5):	innerloop consists of GMRES(5), if $\|AU_{k,5}^{(0)} - r_k\| \geq 0.99\|r_k\|_2$ then GMRES(5) is followed by one LSQR iteration.

GMRESR(5) with $s = 1$ does not converge within 1000 iterations. The results using the other variants are given in Figure 4. Note that GMRESR1 and GMRESR3 have a reasonable good convergence behaviour. This motivates us to combine the ideas between both, which leads to the GMRESR4 variant. The advantages of GMRESR4 are: it only uses an LSQR iteration if it is necessary, and the GMRES(5) results are not thrown away, which is done in the GMRESR3 variant. Furthermore it appears from Figure 4 that GMRESR4 has the best convergence behaviour. All variants use approximately the same CPU-time per iteration.

6.2 The single precision innerloop. In the GMRESR method, the innerloop calculates an approximate solution of $Au_{k,m}^{(0)} = r_k$. This approximation is used as search direction, so its accuracy only influences the convergence but not the accuracy of the final solution vector. Since some computers calculate faster in single precision than in double precision, we have done experiments with a single precision innerloop. The vector $c_k^{(0)} = Au_k^{(0)}$ should be calculated in double precision. In such a case m_{opt} with respect to work can be chosen sligthly larger because the innerloop is cheaper. A comparable approach is given in [15] and [10]. In [10] they use as inner loop GMRES(m) in single precision and as outer loop an iterative refinement algorithm in double precision.

The results for the testproblem of Section 2 are given in Table 9. The CPU time of GMRESR(15) with single precision innerloop is indeed less than the CPU time of

method	iterations	CPU time
GMRESR(10)	36	4.4
GMRESR(15) single precision	27	2.9

Table 9: Results for $\beta = 1$, $h = 1/100$ on the Convex C3820.

GMRESR(10), whereas the final solution vectors have the same accuracy. With respect to memory one needs an extra single precision copy of the matrix, however this increase is in general less than the decrease in memory caused by the fact that m can be chosen larger (and thus closer to m_{opt} with respect to memory) and the auxiliary vectors used in the innerloop are single precision.

We have also done experiments on one processor of a CRAY Y-MP4/464. On this machine single precision arithmetic (64 bits) is much faster than double precision arithmetic (128 bits). Note that there are practical problems, where the system of equations is very ill conditioned. These problems can only be solved using a high accuracy iterative method. In these experiments we choose as termination criterion: $\|r_k\|_2/\|r_0\|_2 \leq 10^{-20}$. It follows from Table 10 that GMRESR with a single precision innerloop is much faster than with a double precision innerloop. The solution vectors have the same accuracy. This experiment shows that it is possible to calculate on the Cray a double precision result in approximately the same CPU time as to calculate a single precision result.

method	iterations	CPU time
GMRESR(20)	30	108
GMRESR(20) single precision	30	10.8

Table 10: Results for $\beta = 1$, $h = 1/100$ on the CRAY Y-MP4/464 (high accuracy).

On the Convex C3820 we have also applied GMRESR with single precision innerloop on the discretized Navier-Stokes equations (see Section 2 and [14]). In these experiments GMRESR with single precision innerloop is 25% faster than GMRESR, and the Bi-CGSTAB method.

7 Conclusions. We consider the GMRESR(m) method [13], which can be used for the iterative solution of a linear system $Ax = b$ with an unsymmetric and nonsingular matrix A.

Optimal choices for the parameter m are easily obtained and do not change very much for different problems. In most experiments we observe for GMRESR(m) a considerable improvement, in computing and memory requirements, in comparison with more familiar GMRES variants. Furthermore, it appears that in many experiments GMRESR(m) is a robust method even without activating the relaxed LSQR switch.

With respect to CPU time full GMRESR(m) seems to be the best variant. However, memory requirements can be so large that restarted and/or truncated GMRESR(m) should be used. From our experiments it appears that the "minalfa" truncation variant is the best strategy, which leads to a large decrease of memory requirements and only a small increase of CPU time.

We compare GMRESR(m) with GMRES, Bi-CGSTAB and QMR. It appears that two easy to measure parameters, which depend on the system of equations and the used computer, can be used to facilitate the choice of an iterative method.

In [7] a new GMRES-like method is proposed: FGMRES. It appears that full GMRESR is compatible with full FGMRES, however FGMRES can break down, and can only be restarted. From examples it follows that truncated GMRESR can be much better than restarted FGMRES.

We give some new results with respect to the relaxed LSQR switch. The best innerloop strategy seems to be: always apply GMRES(m), and if necessary do one LSQR iteration.

Finally, if one uses a computer on which single precision arithmetic is faster than double precision arithmetic, and the condition number of A is not too large, then a single precision innerloop saves CPU time.

Acknowledgement. I would like to thank H.A. van der Vorst for stimulating discussions and the suggestions for the GMRESR1 and GMRESR2 variants given in Section 6.1. 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).

References

[1] O. Axelsson and G. Lindskog,
On the eigenvalue distribution of a class of preconditioning methods,
Numer. Math., 48, (1986), pp. 479-498.

[2] P.N. Brown,
A theoretical comparison of the Arnoldi and GMRES algorithms,
SIAM J. Sci. Statist. Comput., 13, (1991), pp. 58–78.

[3] R.W. Freund and N.M. Nachtigal,
QMR: a quasi-minimal residual method for non-Hermitian linear systems,
Num. Math., 60, (1991), pp.315–339.

[4] C.P. Jackson and P.C. Robinson,
A numerical study of various algorithms related to the preconditioned conjugate gradient method,
Int. J. Num. Meth. Engng., 21, (1985), pp. 1315–1338.

[5] N.M. Nachtigal, S.C. Reddy and L.N. Trefethen,
How fast are non symmetric matrix iterations,
SIAM. J. Sci. Statist. Comput., 13, (1992), pp. 778–795.

[6] C.C. Paige and M.A. Saunders,
LSQR: an algorithm for sparse linear equations and sparse least squares,
ACM Trans. Math. Soft., 8 (1982), pp. 43–71.

[7] Y. Saad,
A flexible Inner-Outer preconditioned GMRES algorithm,
SIAM J. Sci. Statist. Comput., 14, (1993), pp.461–469.

[8] Y. Saad and M.H. Schultz,
GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems,
SIAM J. Sci. Statist. Comput., 7, (1986), pp. 856–869.

[9] P. Sonneveld,
CGS: a fast Lanczos-type solver for nonsymmetric linear systems,
SIAM J. Sci. Statist. Comput., 10, (1989), pp. 36–52.

[10] K. Turner and H.F. Walker,
Efficient high accuracy solutions with GMRES(m),
SIAM J. Sci. Statist. Comput., 13, (1992), pp.815–825.

[11] H.A. van der Vorst,
Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of non symmetric linear systems,
SIAM J. Sci. Statist. Comput., 13, (1992), pp. 631–644.

[12] H.A. van der Vorst and C. Vuik,
The rate of convergence of the GMRES method,
Preprint 654, University of Utrecht, Department of Mathematics,
(1991).
J. Comp. Appl. Math., to appear.

[13] H.A. van der Vorst and C. Vuik,
GMRESR: a family of nested GMRES methods,
Report 91-80, Faculty of Technical Mathematics and Informatics,
Delft University of Technology, (1991).
J. Num. Lin. Alg. Appl., to appear.

[14] C. Vuik,
Solution of the discretized incompressible Navier-Stokes equations with the GMRES method,
Int. J. Num. Meth. Fluids, 16, (1993), pp. 507–523.

[15] M. Zubair, S.N. Gupta and C.E. Grosch,
A variable precision approach to speedup iterative schemes on fine grained parallel machines,
Parallel Comp., 18, (1992), pp. 1223–1232.

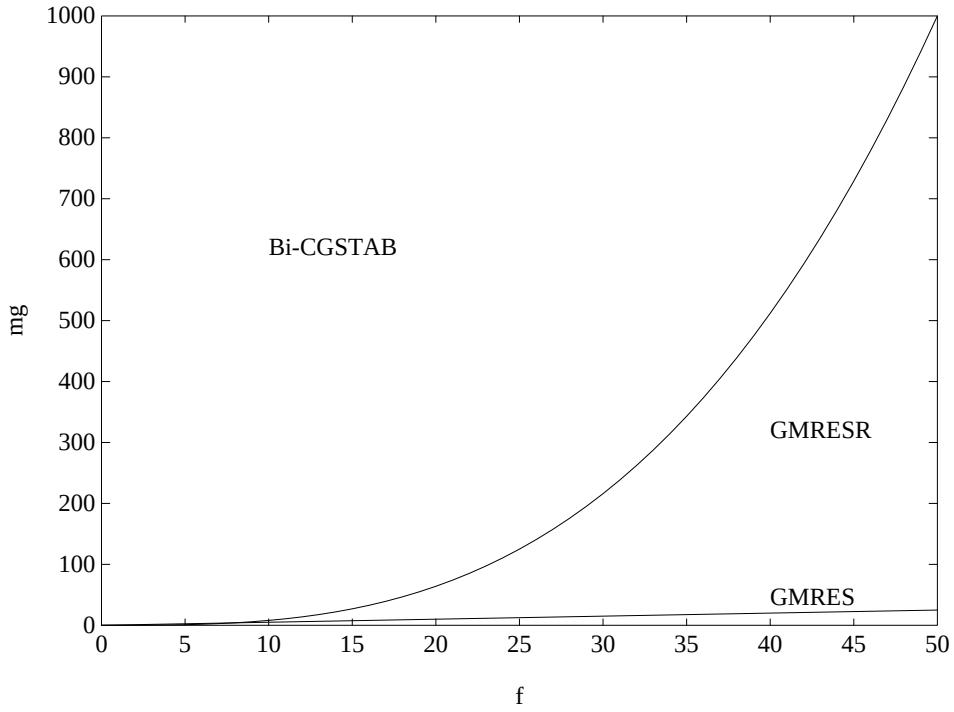

Figure 1: Regions of feasibility of Bi-CGSTAB, GMRES, and GMRESR for $\gamma = 0.5$.

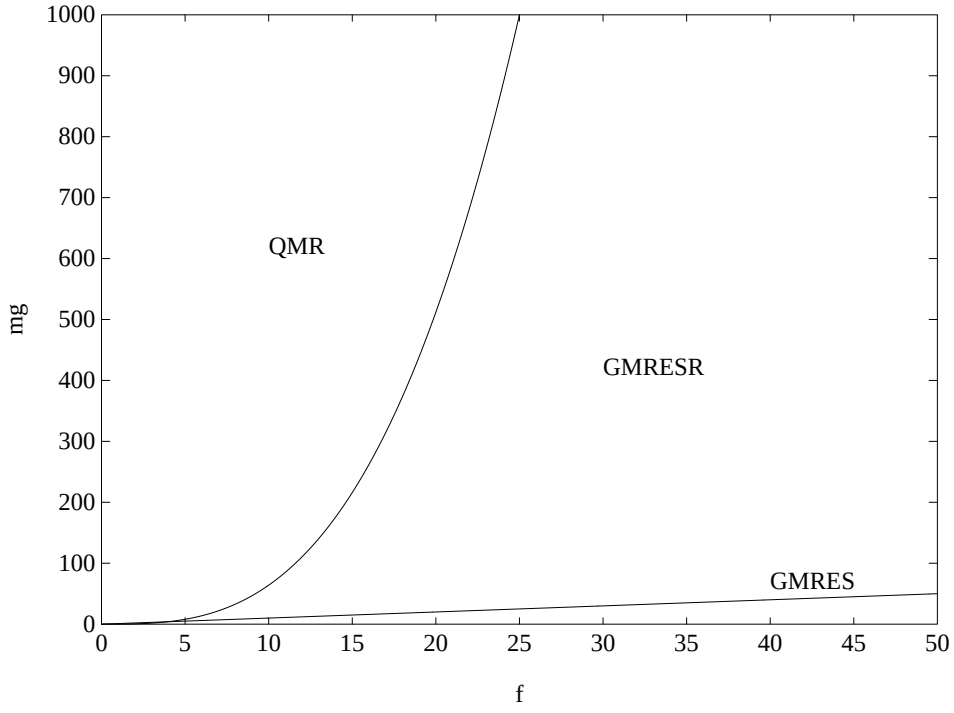

Figure 2: Regions of feasibility of QMR, GMRES, and GMRESR for $\gamma = 1$.

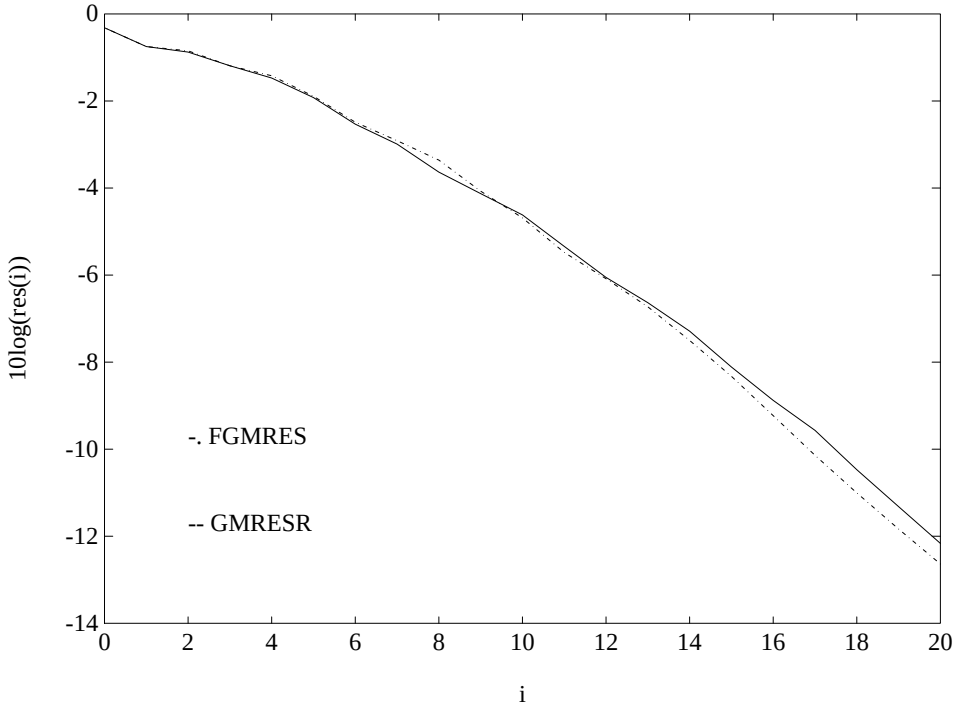

Figure 3: The norm of the residuals for $\beta = 1$ and $h = 1/50$.

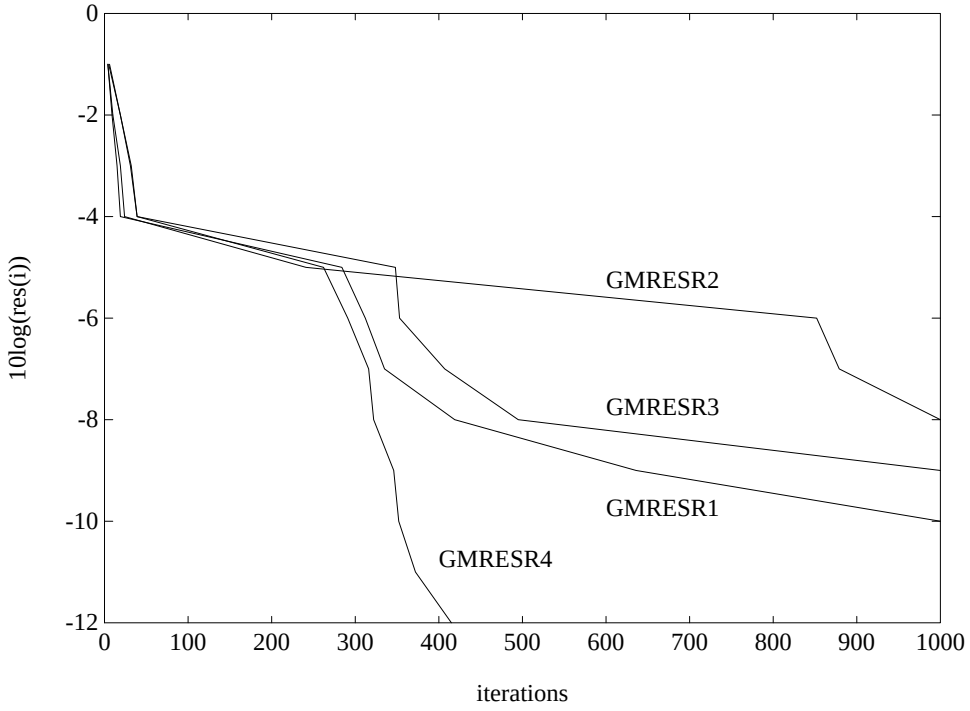

Figure 4: The convergence behaviour of GMRESR variants.