



ELSEVIER

Applied Numerical Mathematics 19 (1995) 287–317



APPLIED  
NUMERICAL  
MATHEMATICS

# A new family of preconditioned iterative solvers for nonsymmetric linear systems<sup>\*</sup>

Ulrike Meier Yang<sup>\*</sup>, Kyle A. Gallivan<sup>†</sup>

*Coordinated Science Laboratory, University of Illinois, 1308 W. Main St., Urbana, IL 61801, USA*

---

## Abstract

A new family of iterative methods, the family of EN-like methods, is introduced, and its relationship to other methods is investigated. The complexity and convergence behavior of the new methods as well as their restarted and truncated versions are examined. The methods are also shown to be suitable in the context of inner/outer iteration schemes. Their adaptive versions are included into a robust software package PARASPAR, and numerical experiments are presented, which demonstrate the efficiency of several members of this new family in comparison with other known methods.

---

## 1. Introduction

There still is a great need to find a robust parallel iterative solver and preconditioner for a general sparse linear system. A large number of iterative methods have been developed, which, when convergent, are efficient. Such methods, however, fail often, and the more robust methods available tend to converge too slowly. Many preconditioning techniques have been proposed with various restrictions on their applicability. The more general and robust ones tend to be costly in sequential terms and can have difficulty exploiting more than a moderate number of processors when implemented in parallel.

In this paper, we investigate preconditioned iterative solvers based on rank-one updates for the nonsymmetric linear system  $Ax = b$  where  $A$  is a general sparse matrix. Our goal is to design and implement an efficient robust iterative solver for such systems.

Specifically, two families of algorithms are considered:

- (i) the family of Broyden algorithms for nonsymmetric linear systems,
- (ii) the family of EN-like methods, a new family of methods, which includes a method proposed by Eirola and Nevanlinna [12].

---

<sup>\*</sup> This research was supported in part by the National Science Foundation under Grant No. CCR-9120105.

<sup>\*</sup> Corresponding author. E-mail: meier@csrd.uiuc.edu.

<sup>†</sup> E-mail: gallivan@csrd.uiuc.edu.

In the past, methods of family (i) had a bad reputation for solving linear systems, but recent efforts [11] have shown that different line search principles lead to versions that are competitive with GMRES [24]. Under certain assumptions, members of both families will terminate after a finite number of steps and have local superlinear convergence. As with other iterative methods such as GMRES, the full methods are too expensive, and restarted, truncated and adaptive versions must be considered. The computational complexity per iteration step of methods of family (ii) is almost twice as high as the corresponding methods of family (i) and GMRES. Whereas the full EN-like methods in many cases only converge about twice as fast, their restarted versions often converge significantly more than twice as fast as the corresponding Broyden methods and GMRES and are therefore more efficient. We will also see that they often require less memory than the corresponding Broyden methods and GMRES.

Iterative methods in both families require, like most other methods, a good preconditioner in order to be robust. There are different ways to precondition iterative methods. We will consider here two different types of preconditioners, the use of an inner iterative method as a preconditioner similar to GMRESR [29] or FGMRES [22] and an incomplete LU factorization with numerical dropping. For the former type of preconditioning the new algorithms are considered as an inner as well as an outer method. The latter preconditioner is taken from PARASPAR, a robust parallel software package based on Y12M [15], which has many other interesting features. The new family of methods appears to be very suitable for the strategy used in PARASPAR that gives it its robustness.

A more detailed discussion of the results of this paper as well as the proofs for the theorems and lemmas can be found in [19].

## 2. Two families of iterative linear solvers

### 2.1. The family of Broyden methods

An important class of methods based on rank- $k$  updates are the quasi-Newton methods [7]. The purpose of quasi-Newton methods is to determine the zero of a function  $F$  or minimize a function  $G$ . They approximate the Jacobian of  $F$  or the Hessian of  $G$ , which is symmetric and often positive definite, or their inverses. There are a variety of effective quasi-Newton methods, such as the Fletcher–Powell–Davidon method and the BFGS method [7]. These methods, however, assume symmetric (and often positive definite) matrices, and we will not consider them here, since our goal is to solve nonsymmetric linear systems. Instead, we will focus on some variants of Broyden's method [3], a quasi-Newton method, which is suitable for solving nonsymmetric linear systems.  $F$  is defined here by  $F(x) = Ax - b$ , and its Jacobian equals  $A$ .

In its most general form, Broyden's method is given by

#### **Algorithm 1** (*Broyden's method*).

Initialization:  $x_0$ ,  $H_0$  arbitrary,  $r_0 = b - Ax_0$ .

For  $k = 0, 1, \dots$ :

$$p_k = H_k r_k \tag{1}$$

$$q_k = Ap_k \tag{2}$$

$$x_{k+1} = x_k + \alpha_k p_k \quad (3)$$

$$r_{k+1} = r_k - \alpha_k q_k \quad (4)$$

$$H_{k+1} = H_k + \frac{(p_k - H_k q_k) f_k^H}{f_k^H q_k} \quad (5)$$

end

where  $f_k$  needs to be chosen in such a way, that  $f_k^H q_k \neq 0$ .

There are two undefined variables,  $f_k$  and  $\alpha_k$ , whose choice must be considered.

For the original Broyden's method, which is also often called Broyden's "good" method (GBM), Broyden used  $f_k = H_k^H p_k$  [3]. It can be proved that, with this choice,  $H_{k+1}$  is the solution to the minimization problem  $\min \|H_k^{-1} - B\|_F$  on the set of all matrices  $B$  that fulfill the secant condition

$$B p_k = q_k, \quad (6)$$

see [8]. ( $\|\cdot\|_F$  denotes the Frobenius norm.)

By a similar argument, minimizing  $\|H_k - H\|_F$  where  $H$  is an element of the set of all matrices that fulfill the following form of the secant condition

$$H q_k = p_k, \quad (7)$$

one obtains the choice  $f_k = q_k$ , which is also called Broyden's "bad" method (BBM). Obviously from the name, this variant often does not perform as well as GBM.

These are the best known choices for  $f_k$ . There are, however, a few other interesting choices. For the special case of a Hermitian matrix  $A$ , a Hermitian update for  $H_k$  is needed, which yields the choice  $f_k = p_k - H_k q_k$ . The interesting aspect of this method is that it finds an approximation  $H_k$  of the inverse of  $A$ , which is corrected during each iteration by a rank-one update in such a way that  $H_{k+1} q_i = p_i$ ,  $i \leq k$  for  $k+1$  points  $q_i = A p_i$ ,  $i = 0, \dots, k$ . Then, Broyden's method will terminate within at most  $n$  steps, since the algorithm constructs a better approximation  $H_k$  to  $A^{-1}$  on each iteration, until finally  $H_n = A^{-1}$ , if  $f_i^H q_i \neq 0$ ,  $i = 0, \dots, n-1$  (see also [18]). Unfortunately, this is not the case if  $A$  is nonsymmetric, and finite termination within  $n$  steps is no longer guaranteed for this choice of  $f_k$ .

In order to get the same effect as in Broyden's method with Hermitian updates for the nonsymmetric case and preserve termination within  $n$  steps, one needs to choose  $f_k$  to be orthogonal to  $q_i$ ,  $i = 0, \dots, k$ . One obvious choice for  $f_k$  that fixes the above problem but also increases the amount of work per iteration significantly is

$$f_k = z_k - \sum_{i=0}^{k-1} \tilde{q}_i^H z_k \tilde{q}_i, \quad (8)$$

where  $z_k \neq 0$  is some arbitrary vector in  $\mathbb{R}^n$ ,  $\tilde{q}_0, \dots, \tilde{q}_{k-1}$  are an orthonormal basis of the space spanned by  $q_i$ ,  $i = 0, \dots, k-1$ . It turns out that the best choice for  $z_k$  is  $q_k$ , since for this choice the error matrix  $E_{k+1} = I - A H_{k+1}$  can be determined through the product of  $E_k$  and a projection matrix, and  $\|E_k\|$  does not increase with increasing  $k$ .

Note, that the choice

$$f_k = H_k^H \left( p_k - \sum_{i=0}^{k-1} \tilde{p}_i^H p_k \tilde{p}_i \right), \quad (9)$$

where  $\tilde{p}_0, \dots, \tilde{p}_{k-1}$  form an orthonormal basis of the space spanned by  $p_i$ ,  $i = 0, \dots, k-1$ , will also lead to termination within  $n$  steps.

This method was developed independently by Gay and Schnabel [17] who call it Broyden's method with projected updates.

There is another choice of  $f_k$  that leads to a version that is equivalent to the general conjugate residual method (GCR) [13] or GMRES with the initial vector  $\tilde{x}_0 = x_0 + H_0 r_0$  and consequently terminates within  $n$  steps [31]. For this method we choose  $f_k = (I - AH_k)^H (I - AH_k) q_k$ . Its convergence behavior is similar to that of GCR or GMRES.

We will focus in our experiments on GBM and BBM, since the Hermitian update is of no interest for general nonsymmetric systems and Broyden's method with projected updates is related to GCR and GMRES (see Section 4), which have a lower computational complexity.

Let us now turn our attention to the second undetermined parameter,  $\alpha_k$ . The most obvious choice for  $\alpha_k$  is 1. One can show that for this case Broyden's method terminates within at most  $2n$  steps [16] (see also Section 5). Nevertheless, this is not always a desirable choice. For example, Fig. 1 shows the convergence behavior of GBM (dashed curves) with  $H_0 = cI$  for different choices of  $c$ , where  $c = 1$  for (1),  $c = 1/\lambda_{\max}$  for (2) and  $c = 2/(\lambda_{\max} + \lambda_{\min})$  for (3). The test problem is taken from [28]. It is a nonsymmetric matrix of the form  $SDS^{-1}$ , where  $D$  is a diagonal matrix with the diagonal vector  $(1, 2, \dots, 50)^T$ , and

$$S = \begin{pmatrix} 1 & \beta & & \\ & \ddots & & \\ & & \ddots & \beta \\ & & & 1 \end{pmatrix}.$$

We chose  $\beta = 0.9$  and the order of the matrix to be 50.

Broyden suggested in [3] to choose  $\alpha_k$ , so that  $\|r_{k+1}\| < \|r_k\|$ . He also states that this choice of  $\alpha_k$  can lead to worse results than choosing an  $\alpha_k$  that does not necessarily fulfill  $\|r_{k+1}\| < \|r_k\|$ , e.g.,  $\alpha_k = 1$ .

Deuffhard et al. [11] propose

$$\alpha_k = \frac{f_k^H r_k}{f_k^H q_k} \quad (10)$$

and they show that the best  $\alpha_k$  for a method depends on the choice of  $f_k$ . Their experiments show that this choice produces Broyden's methods that are competitive with GMRES. In our experiments, this choice of  $\alpha_k$  will be used.

## 2.2. The family of EN-like methods

The EN method was first proposed by Eirola and Nevanlinna in [12]. The main idea is to improve an approximation  $H_k$  to  $A^{-1}$  via a rank-one update  $\tilde{u}_k \tilde{v}_k^H$  on each iteration of the method

while simultaneously improving an approximation  $x_k$  to the solution of the linear system. The rank-one update is chosen in such a way that the matrix  $E_k = I - AH_k$ , which is an indicator of the quality of  $H_k$ , is obtained by premultiplying  $E_{k-1}$  by a projector  $I - cc^H$ , in order to guarantee that the new approximation will not be worse than the old one. This can be achieved by choosing  $v_k = E_k^H A \tilde{u}_k / \|A \tilde{u}_k\|^2$ . The best choice for  $\tilde{u}_k$  would be  $\tilde{u}_k = A^{-1} E_k r_k$  (where the residual  $r_k$  is defined by  $r_k = b - Ax_k$ ), which would lead to  $r_{k+1} = 0$ . Such a choice clearly begs the question of solving the system of linear equations, so the best available approximation of  $A^{-1}$  is used to yield  $\tilde{u}_k = H_k E_k r_k$ .

The resulting algorithm is:

**Algorithm 2** (EN method (original version)).

Initialization:  $x_0, H_0$  arbitrary,  $r_0 = b - Ax_0, E_0 = I - AH_0$ .

For  $k = 0, 1, \dots$ :

$$\tilde{u}_k = H_k E_k r_k$$

$$v_k = \frac{E_k^H A \tilde{u}_k}{\|A \tilde{u}_k\|^2}$$

$$H_{k+1} = H_k + \tilde{u}_k v_k^H$$

$$E_{k+1} = I - AH_{k+1}$$

$$x_{k+1} = x_k + H_{k+1} r_k$$

$$r_{k+1} = E_{k+1} r_k$$

end

The EN method and the family of Broyden's methods are related. As a matter of fact, using this relationship, it is possible to define a new family of EN-like methods.

Recalling the definitions of  $p_k$  and  $q_k$  in Algorithm 1, we can rewrite the evaluation of  $H_{k+1}$  as

$$H_{k+1} = H_k + \frac{H_k E_k r_k f_k^H}{f_k^H A H_k r_k}. \quad (11)$$

Setting  $f_k = E_k^H A H_k E_k r_k$ , we obtain

$$H_{k+1} = H_k + \frac{H_k E_k r_k r_k^H E_k^H H_k^H A^H E_k}{\|A H_k E_k r_k\|^2}, \quad (12)$$

which looks just like  $H_{k+1}$  as evaluated in the EN method. Consequently, it is possible to derive a family of methods with a general  $f_k$ , just as can be done for Broyden's methods. The EN method is a special case of this family. The new general form of the EN method is given by

**Algorithm 3** (EN-like method).

Initialization:  $x_0, H_0$  arbitrary,  $r_0 = b - Ax_0, E_0 = I - AH_0$ .

For  $k = 0, 1, \dots$ :

$$H_{k+1} = H_k + \frac{H_k E_k r_k f_k^H}{f_k^H A H_k r_k} \quad (13)$$

$$E_{k+1} = I - AH_{k+1} \quad (14)$$

$$x_{k+1} = x_k + H_{k+1}r_k \quad (15)$$

$$r_{k+1} = E_{k+1}r_k \quad (16)$$

end

where  $f_k$  needs to be chosen in such a way, that  $f_k^H AH_k r_k \neq 0$ .

Algorithm 3 can also be written in a form that more closely resembles Broyden's methods as follows:

**Algorithm 4** (*EN-like method*).

Initialization:  $x_0, H_0$  arbitrary,  $r_0 = b - Ax_0$ .

For  $k = 0, 1, \dots$ :

$$p_k = H_k r_k \quad (17)$$

$$q_k = Ap_k \quad (18)$$

$$H_{k+1} = H_k + \frac{(p_k - H_k q_k) f_k^H}{f_k^H q_k} \quad (19)$$

$$\tilde{p}_k = H_{k+1} r_k \quad (20)$$

$$\tilde{q}_k = A \tilde{p}_k \quad (21)$$

$$x_{k+1} = x_k + \tilde{p}_k \quad (22)$$

$$r_{k+1} = r_k - \tilde{q}_k \quad (23)$$

end

where  $f_k$  needs to be chosen in such a way, that  $f_k^H q_k \neq 0$ .

So, in some way, Broyden's method is to the EN-like method what the Jacobi method is to the Gauss–Seidel method. Whereas the direction vector for Broyden's method is evaluated using  $H_k$ , for the EN-like method the new approximation  $H_{k+1}$  to  $A^{-1}$  is used, which can lead to a faster convergence.

Fig. 1 shows that the EN-like method (solid lines) with  $f_k = H_k^H p_k$ , which we will call GEN method converges about twice as fast as GBM (dashed lines) for our test problem. This leads to a comparable sparse matrix–vector multiplication count for both methods, since the EN-like method has a higher computational complexity per iteration step. We also see that the disturbing increase of the residual in Broyden's method that occurs for case (1) is significantly increased for the EN-like method.

The similarity of two steps of Broyden and one step of the corresponding EN-like method are examined in the following lemma.

**Lemma 1.** *One iteration step of an EN-like method can be decomposed in the following way:*

$$\tilde{x}_{k+1} = x_k + H_k r_k, \quad (24)$$

$$\tilde{r}_{k+1} = r_k - AH_k r_k, \quad (25)$$

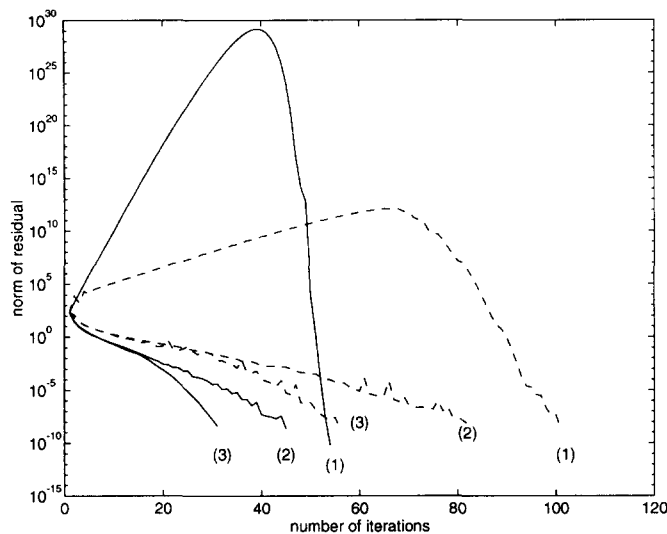


Fig. 1. Broyden's "good" method (GBM) versus corresponding EN-like method (GEN).

$$x_{k+1} = \tilde{x}_{k+1} + \alpha_k H_k \tilde{r}_{k+1}, \quad (26)$$

where

$$\alpha_k = \frac{f_k^H r_k}{f_k^H q_k}.$$

The proof is straightforward and can be found in [19]. This lemma shows that one step of an EN-like method can be considered as an iteration step of the corresponding Broyden's method with  $\alpha_k = 1$  followed by an iteration step of Broyden's method using the optimal line search principle of [11] without updating the approximation to  $A^{-1}$ .

In our experiments in Section 7, we will focus our attention specifically on EN, GEN and the EN-like method with  $f_k = q_k$ , which we will call BEN.

### 2.3. Scaling invariance

We will consider now the influence of scaling the linear system on the iterative solver.

**Definition 2.** An iterative method is called scaling invariant, if  $x_k = \tilde{x}_k$ , where  $x_k$  is the  $k$ th iterate generated by applying the iterative solver to  $Ax = b$ , and  $\tilde{x}_k$  is the  $k$ th iterate generated by applying the iterative solver to  $\rho Ax = \rho b$  for any  $\rho \neq 0$ .

In general, neither Broyden's method nor the EN-like method is scaling invariant. This is also indicated by the experiments in Fig. 1, where we chose  $H_0 = cI$ . They are equivalent to applying the solvers to the linear system  $(1/c)Ax = (1/c)b$ .

One way of fixing this problem is to premultiply  $H_0$  by a scaling parameter in the initialization phase. We will denote the methods that are generated this way through a prefix “s” (e.g. sEN for the scaling invariant version of the EN method).

**Theorem 3.** Define for a Broyden or EN-like method  $H_0 = \gamma M$ , where

$$\gamma = \frac{(Az)^H z}{\|Az\|^2} \quad \text{or} \quad \gamma = \frac{(AMz)^H z}{\|AMz\|^2}, \quad (27)$$

$z$  an arbitrary vector with  $Az \neq 0$  and  $M$  a nonsingular matrix. If  $\alpha_k = \tilde{\alpha}_k$  for Broyden's method and  $\tilde{f}_k = g(\rho) f_k$  for a function  $g: \mathcal{C} \rightarrow \mathcal{C}$  (where  $\tilde{\alpha}_k$  and  $\tilde{f}_k$  are generated by applying the method to the scaled system  $\rho Ax = \rho b$ ), then the method is scaling invariant.

An induction-based proof can be found in [19]. Even though an arbitrary  $z$  theoretically guarantees scaling invariance, its choice is important practical matter. An unfortunate choice of  $z$  might make no difference or even degrade the convergence of the method, whereas a well-chosen  $z$  might lead to an improvement in the number of iterations. An example of this can be found in Fig. 1, where for case (1)  $z$  is the eigenvector belonging to the eigenvalue 1 and for case (2) the one belonging to the largest eigenvalue  $\lambda_{\max}$ .

Vuik and van der Vorst [31] suggest another scaling invariant version of the EN method, which we will call the SEN method. While examining the error matrix

$$E_{k+1} = (I - c_k c_k^H) \cdots (I - c_0 c_0^H) E_0, \quad (28)$$

they realized that the scaling invariance was caused by the factor  $I - AH_0$ , and they suggested to introduce a scaling parameter  $\gamma_k$ , leading to the product  $(I - c_k c_k^H) \cdots (I - c_0 c_0^H) (I - \gamma_k AH_0)$ .  $\gamma_k$  is determined by minimizing the vector  $(I - \gamma_k AH_0) r_k$ , which leads to

$$\gamma_k = \frac{(AH_0 r_k)^H r_k}{\|AH_0 r_k\|^2}. \quad (29)$$

The SEN method is slightly more expensive and requires two more dotproducts per iteration. Both dotproducts can be performed simultaneously.

### 3. Efficiency considerations

#### 3.1. Formulations of higher efficiency

One of the disadvantages of the methods we have considered so far is their computational complexity. Whereas  $A$  is in general a sparse matrix,  $H_k$  is in general dense and therefore would require  $O(n^2)$  number of operations, compared to  $O(n)$  for  $A$ . One can develop more efficient versions for all these methods by avoiding the actual computation of  $H_{k+1}$ .

If we replace  $H_{k+1}$  by its definition, we get

$$\begin{aligned}
 H_{k+1}x &= H_kx + \frac{f_k^H x}{f_k^H q_k} (p_k - H_k q_k) \\
 &= H_0x + \sum_{i=0}^k \frac{f_i^H x}{f_i^H q_i} (p_i - H_i q_i).
 \end{aligned}$$

Using this, we can rewrite Broyden's method in the following form (see also [11]).

**Algorithm 5** (*Broyden's method*).

Initialization:  $x_0, H_0$  arbitrary,  $r_0 = b - Ax_0$ ,  $p_0 = H_0 r_0$ ,  $q_0 = Ap_0$ ,

For  $k = 0, 1, \dots$ :

$$\zeta_k = f_k^H q_k$$

$$t_k = H_0 q_k + \sum_{i=0}^{k-1} \frac{f_i^H q_k}{\zeta_i} z_i$$

$$z_k = p_k - t_k$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k q_k$$

$$p_{k+1} = (1 - \alpha_k) p_k + \frac{f_k^H r_k}{\zeta_k} z_k$$

$$q_{k+1} = Ap_{k+1}$$

end

The general EN-like method can be rewritten as follows:

**Algorithm 6** (*EN-like method*).

Initialization:  $x_0, H_0$  arbitrary,  $r_0 = b - Ax_0$ ,

For  $k = 0, 1, \dots$ :

$$p_k = H_0 r_k + \sum_{i=0}^{k-1} \frac{f_i^H r_k}{\zeta_i} z_i$$

$$q_k = Ap_k$$

$$\zeta_k = f_k^H q_k$$

$$t_k = r_k - q_k$$

$$z_k = H_0 t_k + \sum_{i=0}^{k-1} \frac{f_i^H t_k}{\zeta_i} z_i$$

$$s_k = p_k + \frac{f_k^H r_k}{\zeta_k} z_k$$

$$x_{k+1} = x_k + s_k$$

$$r_{k+1} = r_k - As_k$$

end

For Broyden's "bad" method (BBM) and for the corresponding EN-like method (BEN), one only needs to replace  $f_k$  with  $q_k$ . Similarly, for Broyden's projected method, set  $f_k = q_k - \sum_{i=0}^{k-1} \tilde{q}_i^H q_k \tilde{q}_i$ . Broyden's "good" method (GBM) and GEN, however, or any other member of the families for which the evaluation of  $f_k$  involves  $H_k^H$ , such as the original EN method, requires further modification. Fortunately, for GBM and GEN, those can be performed without significantly increasing the number of operations or the storage needed, if one chooses the order of computations carefully. Unfortunately, the evaluation is highly recursive and leads to a decrease in parallelism. The complete algorithms are given in [19]. For the EN and the SEN method, where  $f_k = E_k^H E_k q_k$ , it is possible to make use of the orthogonality of some vectors and so avoid the decrease in parallelism we encountered for GBM and GEN. Since the evaluation of  $f_k$  is more complicated than for GEN or BEN, the number of operations is however increased. For detail see [12,19,31].

If we do not require the evaluation of  $x_{k+1}$  in each iteration step, it is possible to achieve further savings in EN and SEN by avoiding the evaluation of the updates for  $x_{k+1}$  in each iterations step and accumulating the coefficients instead. Such an approach has been used for GCR in [9]. These new even more efficient versions, which we will call eEN, eSEN and eGCR can be found in [19].

Even the efficient versions are computationally expensive, since the gradual increase of the underlying subspace leads to an increase of both operation count and memory requirement with each iteration step. We will therefore also consider their restarted and truncated versions. An overview of restarted and truncated algorithms can be found in [23].

The methods can be restarted after  $m+1$  iterations by using  $x_{m+1}$  as the new starting guess  $x_0$ . We truncate these methods by including only updates of the last  $m$  iterations. We will denote the restarted methods with Method( $m$ ) and the truncated versions with tMethod( $m$ ). One would expect the latter approach to lead to better convergence, since more information is being kept. We will see in Section 7 that this is not always true. In fact, Deuffhard, Freund and Walter [11] saw in their experiments that for Broyden's methods this approach in general is worse than restarting. Our experiments will however show that this result does not necessarily transfer to the EN-like methods. It is also possible to use more sophisticated truncated schemes, which can lead to better convergence (see [10,30]). The application of these to the family of EN-like methods is left as future work.

Most of the efficient versions we have mentioned here truncate easily. The truncation of eEN, eSEN and eGCR is far more complicated. Since the actual computational complexity of truncated eEN, eSEN and eGCR is not superior to truncated EN, SEN and GCR (which is also called ORTHOMIN), we will not consider them here.

One of the disadvantages of restarted and truncated methods is the fact that a new parameter  $m$  is introduced. It is unclear how to choose it, since a small  $m$  might lead to extremely slow convergence or possibly divergence, and a large  $m$  leads to a high number of operations per iteration step. We have developed adaptive versions, which gradually increase the subspace depending on the convergence rate observed. These versions are considered in some detail in Section 6.

### 3.2. Complexity

In order to compare the methods with each other as well as with existing methods, it is necessary to develop a model. There are different issues that need to be considered.

- the total operation count,
- the type of operations,

- and the memory requirements.

Since we are dealing with iterative methods, the total operation count consists of the number of operations per iteration step, which we consider in this section and the number of iterations required to achieve a certain accuracy. The second aspect is considered in more detail in Section 5.

It is important to consider the type of operations involved, since the performance of computational primitives can differ significantly from one another and depends strongly on the type of machine. This is particularly true for parallel computers, our target architectures, which perform operations such as dense matrix–vector multiplications more efficiently than e.g. general sparse matrix–vector multiplications.

In Table 1 the types and number of operations are given for a variety of methods. We have included GCR and the equivalent GMRES [24] here since they are related to the considered methods (see next section), and CGS [26] and BiCGSTAB [27] since they are very popular solvers. For Broyden's methods, only the operation counts for optimal line search according to [11] are given. Those for GBM or BBM with  $\alpha_k = 1$  would be slightly lower. “dmv” stands here for dense matrix–vector multiplication. There are two types of dense matrix–vector operations included, first any computations of dotproducts of the form  $c_i^H x$ ,  $i = 0, \dots, k-1$  can also be performed as the multiplication of the vector  $x$  with the matrix  $(c_0, \dots, c_{k-1})$  of order  $n \times k$ , and second, the operation  $\sum_{i=0}^{k-1} \alpha_i c_i$  can be performed as a multiplication of the matrix  $(c_0, \dots, c_{k-1})$  with the vector  $(\alpha_0, \dots, \alpha_{k-1})^H$ . Both matrix–vector multiplications take about  $2kn$  flops. We also consider the following vector operations: the inner product of two vectors (“dp”), the vector operation  $x = x + \alpha y$  (“daxpy”), and plain vector additions, subtractions or multiplications of a vector with a scalar (+, −, \*). Another important part of each algorithm are multiplications of the sparse matrix  $A$  with a vector (“smv”). The flop counts depend strongly on the linear system we are considering. They can be small for a very sparse matrix and require little time if the matrix is also well structured, such as banded. They can be large, if the matrix is fairly dense or even given in implicit form. Another unknown factor is the preconditioning step (“prec”), which also strongly depends on the preconditioner used. For a diagonal preconditioner, e.g., the cost is almost negligible, for an incomplete LU preconditioner, however, it can be quite expensive due to its potentially low degree of parallelism.

Note that for GMRES and the efficient versions eGCR, eEN and eSEN a postprocessing step that consists of one “dmv” ( $2mn$  flops after  $m$  iterations) is required in order to get the actual solution and an additional “dmv” for GMRES if one needs to get the residual vector as is necessary for restarted GMRES.

Note that the “dmv”s here are actually  $k$  dotproducts or  $k$  daxpys that can be performed simultaneously. Consequently, methods with “dmvs” such as BBM, GCR, BEN, EN, etc. have a higher degree of parallelism than methods such as GMRES in its usual implementation with modified Gram–Schmidt, GBM and GEN, which have to perform these dotproducts and daxpys recursively. It is possible to use the classical Gram–Schmidt algorithm for GMRES to increase its parallelism. In this case its operation count as given in Table 1 is identical to that of eGCR. Since it also can lead to instability, it has been suggested to use the classical Gram–Schmidt algorithm twice [6].

In order to get an idea of the actual computational complexity in terms of flops, in Tables 3.2 and 3.2 the number of flops per iteration step for the truncated and restarted versions are given. We use the notation “tMethod( $m$ )” for the truncated versions and “Method( $m$ )” for the restarted versions. Note that tGCR( $m$ ) is better known as ORTHOMIN( $m$ ). Those numbers are straightforward for the truncated methods assuming that one uses a window of  $m$  vectors and neglects the startup. For the

Table 1  
Number and types of operations for various methods in  $k$ th iteration step

Method	dmv	dp	daxpy	+, −, *	smv	prec
Operation count	$2kn$	$2n$	$2n$	$n$		
BBM	2	2	3	1	1	1
GBM	-	$k+2$	$k+3$	1	1	1
GCR	3	2	2	-	1	1
eGCR	2	2	1	-	1	1
GMRES	-	$k+2$	$k$	-	1	1
BEN	4	2	1	3	2	2
GEN	-	$2k+2$	$2k+1$	2	2	2
EN	6	2	2	2	2	2
eEN	4	2	1	2	2	2
SEN	6	4	3	2	2	2
eSEN	4	4	3	2	2	2
CGS	-	2	5	1	2	2
BiCGSTAB	-	4	6	-	2	2

Table 2  
Computational complexities for truncated versions

Method	Number of operations per iteration
tBBM( $m$ )	$(4m+11)n + 1$ smv + 1 prec
tGBM( $m$ )	$(4m+11)n + 1$ smv + 1 prec
ORTHOMIN( $m$ )	$(6m+8)n + 1$ smv + 1 prec
tBEN( $m$ )	$(8m+9)n + 2$ smv + 2 prec
tGEN( $m$ )	$(8m+9)n + 2$ smv + 2 prec
tEN( $m$ )	$(12m+10)n + 2$ smv + 2 prec
tSEN( $m$ )	$(12m+16)n + 2$ smv + 2 prec
CGS	$15n + 2$ smv + 2 prec
BiCGSTAB	$20n + 2$ smv + 2 prec

restarted methods, one can only give the average number of iterations per iteration step, since the actual number is continually changing due to the changing size of the window. Note that it is possible here to use the more efficient versions eGCR, eEN and eSEN. The averaged operation count also takes into account the previously mentioned postprocessing step for the efficient methods. Clearly, using restarted methods leads to lower operation counts per iteration step, consequently the use of truncated methods will only pay off if the number of iterations is significantly lower. We will consider this aspect in more detail in Section 7.

Also, clearly, methods like GBM, BBM, GCR and GMRES have the advantage of using only one “smv” and one “prec” per iteration step, which in case of an expensive “smv” and “prec” will make the other methods only attractive if they converge at least twice as fast.

Our final complexity criterion is the memory required. Table 3.2 lists the amount of memory in terms of vector elements required in addition to the matrix and the right-hand side. We ignore memory requirements of order  $k$  or  $k^2$ , since we assume that  $k$  is in general small compared to  $n$ .

It is possible to save a substantial amount of storage for GBM (see [11,19]). This change requires however an additional  $k$  daxpys per iteration step if  $\alpha_i \neq 1$ . Unfortunately, it is not possible to use a similar trick for GEN.

Table 3  
Computational complexities for restarted versions

Method	Average number of operations per iteration
BBM( $m$ )	$(2m+11)n + 1 \text{ smv} + 1 \text{ prec}$
GBM( $m$ )	$(2m+11)n + 1 \text{ smv} + 1 \text{ prec}$
eGCR( $m$ )	$(2m+8)n + 1 \text{ smv} + 1 \text{ prec}$
GMRES( $m+1$ )	$(2m+8)n + 1 \text{ smv} + 1 \text{ prec}$
BEN( $m$ )	$(4m+9)n + 2 \text{ smv} + 2 \text{ prec}$
GEN( $m$ )	$(4m+9)n + 2 \text{ smv} + 2 \text{ prec}$
eEN( $m$ )	$(4m+10)n + 2 \text{ smv} + 2 \text{ prec}$
eSEN( $m$ )	$(4m+16)n + 2 \text{ smv} + 2 \text{ prec}$

Table 4  
Additional work vectors required

Method	Memory required
(t)BBM( $m$ )	$2m+5$
(t)GBM( $m$ )	$2m+5$ or $m+6$
(t)GCR( $m$ ), eGCR( $m$ )	$2m+4$
(t)BEN( $m$ )	$2m+6$
(t)GEN( $m$ )	$2m+6$
(t)EN( $m$ ), eEN( $m$ )	$2m+4$
(t)SEN( $m$ ), eSEN( $m$ )	$2m+4$
GMRES( $m+1$ )	$m+4$
CGS	8
BiCGSTAB	8

We see here that unless  $m$  is small, CGS and BiCGSTAB require less memory than the other methods. All the other methods require approximately the same amount of memory, except for GMRES and GBM, which use about half the amount of memory.

#### 4. Relationships between methods

In the previous sections, we have indicated that Broyden's methods and the EN-like methods are related to various other known methods, particularly GCR. In this section, we summarize these relationships.

As mentioned in [12] and proved in [31], one can derive GCR from the EN method by replacing  $\tilde{u}_k = H_k E_k r_k$  through

$$\tilde{u}_k = H_k r_k. \quad (30)$$

A more thorough investigation shows that GCR and ORTHOMIN are related to SEN and tSEN.

**Lemma 4.** *If the scaling parameter  $\gamma_k$  in (29) equals 0, the SEN iteration step is reduced to a GCR iteration step.*

Since  $\gamma_k = (AH_0 r_k)^H r_k / \|AH_0 r_k\|^2$ , this situation can occur only when  $AH_0$  is not positive definite. Moreover, when  $\gamma_k = 0$ , the algorithm stagnates.

Comparing  $\text{GCR}(k)$  (or  $\text{ORTHOMIN}(k)$ ) and  $\text{SEN}(k)$  (or  $\text{tSEN}(k)$ ), one finds the following equivalence for the special case  $k = 0$  for which restarted and truncated versions are equivalent:

**Lemma 5.** *Two steps of  $\text{GCR}(0)$  is equivalent to one step of  $\text{SEN}(0)$ , i.e., given the same initial vector  $x_0$ ,*

$$x_{2k}^{\text{GCR}(0)} = x_k^{\text{SEN}(0)}. \quad (31)$$

The proofs are straightforward.

We have also mentioned that Broyden's method with  $f_k = E_k^H E_k q_k$  is equivalent to GCR using the initial vector  $x_0 + H_0 r_0$ . This is equivalent to applying one step of Richardson's method to the linear system  $H_0 A x = H_0 b$  with the initial vector  $x_0$  and applying GCR to the so obtained iterate. Note that even though GCR is scaling invariant, Broyden's method with  $f_k = E_k^H E_k q_k$  is not, since Richardson's method is not.

Since the projected Broyden's method also terminates within  $n$  steps and generates orthogonal vectors, it is reasonable to suspect that it is also related to GCR. The following lemma relates the  $f_k$  generated in the projected Broyden's method to the Krylov subspace generated by GCR.

**Lemma 6.** *For the projected Broyden's method,  $f_k = q_k - \sum_{i=0}^{k-1} \tilde{q}_i^H q_k \tilde{q}_i$  is orthogonal to the  $k$ th Krylov space  $[AH_0 r_0, \dots, (AH_0)^k r_0] = K^k(AH_0, AH_0 r_0)$ .*

The vectors  $\tilde{q}_0, \dots, \tilde{q}_{k-1}$  form an orthonormal basis for  $K^k(AH_0, AH_0 r_0)$ . Moreover, defining

$$\tilde{r}_k = r_k - \sum_{i=0}^{k-1} \tilde{q}_i^H r_k \tilde{q}_i, \quad (32)$$

where  $\tilde{r}_{k+1} = \tilde{r}_k - \tilde{q}_k$ , one obtains a sequence of "residuals" with the property

$$\tilde{r}_k \perp K^k(AH_0, AH_0 r_0). \quad (33)$$

Implicitly, residuals for GCR applied to  $AH_0(H_0^{-1}x) = b$  are generated.

Since the EN-like method generates an approximation  $H_k$  to  $A^{-1}$ , a relationship to matrix iterations that compute the inverse of a matrix is also likely. Such a method can be found in [25].

Choose arbitrary  $X_0$ ,

$$X_{k+1} = X_k(2I - AX_k).$$

This method is based on Newton's method and possesses quadratic convergence in the sense that

$$I - AX_{k+1} = (I - AX_k)^2.$$

Investigating one step of the EN-like method, one can show

**Theorem 7.** *For the EN-like method, the direction vector  $H_{k+1}r_k$  can be presented in the following way:*

$$H_{k+1}r_k = (1 - \omega_k)H_k r_k + \omega_k H_{k+1}^N r_k \quad (34)$$

where

$$H_{k+1}^N = H_k(2I - AH_k) \quad (35)$$

and

$$\omega_k = \frac{f_k^H r_k}{f_k^H q_k}. \quad (36)$$

The interesting part of this presentation is that for  $\omega_k = 0$ , we have Broyden's method, for  $\omega_k = 1$  we have a much faster converging but far more costly method based on Newton's method for approximating the inverse. Of course, in general  $\omega_k$  would be neither 0 nor 1, since it strongly depends on the vectors  $f_k$ ,  $q_k$  and  $r_k$ . However, local convergence considerations show that for some choices of  $f_k$ ,  $\omega_k$  converges towards 1, when  $H_0$  is a good approximation for  $A^{-1}$ , and in this case  $H_{k+1}$  acts on  $r_k$  in a manner similar to the Newton iterate [19].

## 5. Convergence theory

As mentioned in Section 3.2, the EN-like methods need to converge at least twice as fast as the Broyden methods, in order to be competitive. We will show here that theoretically this is often the case. In Section 7, we will show that in practice they often perform significantly better.

One of the amazing, unexpected properties of Broyden's method is its finite termination property, which occurs for  $\alpha_k = 1$ . Gay showed that Broyden's method terminates within at most  $2n$  steps [16]. Recently, O'Leary [20] characterized the vectors that cause the finite termination.

Now, due to the relationship between Broyden's method and the EN-like method, it is also possible to prove finite termination for the EN-like method (see [19]).

**Theorem 8.** *If  $f_k^H q_k \neq 0$ ,  $k = 0, 1, \dots$ , the EN-like method converges within at most  $n$  steps.*

The finite termination property is more of theoretical than of practical interest. Therefore, it is important to examine the convergence behavior of the methods.

Due to space limitation, we present the following theorems in condensed form. Most of the results for GBM and BBM can also be found in [11]. The proofs for the results for the EN-like methods and the additional results for the Broyden methods can be found in [19].

The following theorem characterizes convergence for BBM, BEN, EN and the projected Broyden's method.

**Theorem 9.** *Assume that  $f_k = q_k$ ,  $f_k = E_k^H E_k q_k$ , or  $f_k = q_k - \sum_{i=0}^{k-1} \tilde{q}_i^H q_k \tilde{q}_i$ , and for Broyden's method  $\alpha_k = 1$  or  $\alpha_k = q_k^H r_k / q_k^H q_k$ . Assume additionally that  $\|E_0\| \leq \delta < 1$ .*

*The following inequalities then hold:*

$$\|r_{k+1}\| \leq \delta \|r_k\| \quad (37)$$

for Broyden's method and

$$\|r_{k+1}\| \leq \delta^2 \|r_k\| \quad (38)$$

for the EN-like method, and the methods converge.

Assume for Broyden's method that  $f_k = E_k^H E_k q_k$ , and  $\alpha_k = f_k^H r_k / f_k^H q_k$ . Assume additionally that  $\|E_0\| \leq \delta < \sqrt{2} - 1$ .

Then,

$$\|r_{k+1}\| \leq \delta(\delta + 2)\|r_k\|, \quad (39)$$

and the method converges.

Additionally, for all methods considered above, the convergence is  $q$ -superlinear, i.e., there exists a sequence  $c_k$  with  $0 \leq c_k < 1$  and  $\lim_{k \rightarrow \infty} c_k = 0$ , so that

$$\|r_{k+1}\| \leq c_k \|r_k\|. \quad (40)$$

There is a similar theorem for GBM, GEN, and a few other methods with  $f_k$  as defined below. For the following theorem, we define the matrices

$$\tilde{E}_k := I - A^{-1} H_k^{-1}. \quad (41)$$

**Theorem 10.** Assume that  $f_k = H_k^H p_k$ , or  $f_k = H_k^H \tilde{E}_k^H \tilde{E}_k p_k$ , or for Broyden's method  $f_k = H_k^H (p_k - \sum_{i=0}^{k-1} \tilde{p}_i^H p_k \tilde{p}_i)$ , and one of the following choices of  $\alpha_k$  for Broyden's method:

- (i)  $\alpha_k = 1$ ,
- (ii)  $\alpha_k = p_k^H p_k / p_k^H H_k q_k$ ,
- (iii)  $\alpha_k = p_k^H H_k q_k / q_k^H H_k^H H_k q_k$ ,
- (iv)  $\alpha_k = \|\tilde{E}_k p_k\|^2 / p_k^H \tilde{E}_k^H \tilde{E}_k H_k q_k$ , or
- (v)  $\alpha_k = p_k^H \tilde{E}_k^H \tilde{E}_k H_k q_k / \|\tilde{E}_k H_k q_k\|^2$ .

Assume additionally that  $H_k A$  is nonsingular, and  $\|\tilde{E}_0\| \leq \delta$ , where  $\delta < \frac{1}{2}$  for Broyden's method with  $\alpha_k = 1$ ,  $\delta < \frac{1}{3}$  for Broyden's method with the other choices for  $\alpha_k$  and  $\delta < \sqrt{2} - 1$  for the EN-like method.

The following inequalities then hold:

$$\|e_{k+1}\| \leq \frac{\delta}{1 - \delta} \|e_k\| \quad (42)$$

for Broyden's method with  $\alpha_k = 1$ ,

$$\|e_{k+1}\| \leq \frac{2\delta}{1 - \delta} \|e_k\| \quad (43)$$

for Broyden's method with choices (ii)–(v) for  $\alpha_k$ , and

$$\|e_{k+1}\| \leq \frac{2\delta^2}{(1 - \delta)^2} \|e_k\| \quad (44)$$

for the EN-like method (where  $e_k = x_k - x$  denotes the actual error) and the methods converge.

Additionally, the convergence is  $q$ -superlinear, i.e., there exists a sequence  $c_k$  with  $0 \leq c_k < 1$  and  $\lim_{k \rightarrow \infty} c_k = 0$ , so that

$$\|e_{k+1}\| \leq c_k \|e_k\| \quad (45)$$

for Broyden's method with choices (i)–(iii) for  $\alpha_k$  and the EN-like method.

These theorems show that, under the above assumptions, the EN-like methods converge twice as fast as the corresponding Broyden methods, and the upper limits of the convergence rate for the EN-like methods are approximately squared compared to those of the corresponding Broyden methods, a fact that is useful for the development of the adaptive methods (see Section 6).

Now let us examine restarted and truncated versions of these methods. Certainly, those versions do not possess the finite termination property. The local convergence behavior of the restarted methods is, however, characterized by Theorems 9 and 10, except that one can no longer prove q-superlinear but only q-linear convergence.

For truncated methods, it is far more difficult to derive any convergence results. It is however possible for one of the methods considered to prove convergence using an argument similar to that used for ORTHOMIN [13,14]. Let us investigate the tSEN method for a real linear system  $Ax = b$ . Since one can prove convergence for the full SEN method and the restarted method SEN( $m$ ) in the same way, we summarize the result in the following theorem.

**Theorem 11.** Assume, that the symmetric part  $M$  of  $AH_0$  is symmetric positive definite,  $R$  is the skew-symmetric part of  $AH_0$ ,  $\eta_i$  the angle between  $c_i$  and  $AH_0r_k$ , and  $\{r_k\}$  the sequence of residuals generated by SEN, SEN( $m$ ), or tSEN( $m$ ), then

$$\|r_{k+1}\| \leq \left(1 - \sum_{i=\phi(m)}^{k-1} \cos^2 \eta_i\right)^{1/2} \left(1 - \frac{\lambda_{\min}^2(M)}{\lambda_{\max}(H_0^T A^T A H_0)}\right)^{1/2} \|r_k\|, \quad (46)$$

and

$$\|r_{k+1}\| \leq \left(1 - \sum_{i=\phi(m)}^{k-1} \cos^2 \eta_i\right)^{1/2} \left(1 - \frac{\lambda_{\min}^2(M)}{\lambda_{\min}(M)\lambda_{\max}(M) + |\lambda_{\max}(R)|^2}\right)^{1/2} \|r_k\|, \quad (47)$$

where  $\phi(m) = 0$  for SEN and SEN( $m$ ), and  $\phi(m) = k - m$  for tSEN( $m$ ), and the method converges.

If we recall the equivalent estimate for ORTHOMIN [13,14]

$$\|r_{k+1}\| \leq \left(1 - \frac{\lambda_{\min}^2(AH_0)}{\lambda_{\max}(H_0^T A^T A H_0)}\right) \|r_k\|, \quad (48)$$

we see that the main difference between the two estimates is the factor

$$1 - \sum_{i=k-m+1}^{k-1} \cos^2 \eta_i,$$

which equals 1 only when  $AH_0r_k$  is orthogonal to all  $c_i$ ,  $i = k - m + 1, \dots, k - 1$ . In all other cases, the estimate is better. It is optimal, when  $AH_0r_k$  is parallel to one of the  $c_i$ .

## 6. Preconditioning

The convergence theory has shown that, in general, it is desirable to start these methods with an  $H_0$  that is already a fairly good approximation of the inverse of the matrix of the linear system

to be solved. A possible choice would be a good preconditioner. In this section, we consider two different ways of preconditioning: preconditioning with an inner iterative method, and the use of an ILU factorization with numerical dropping.

### 6.1. Preconditioning with iterative methods

The idea of using an inner iterative method as a preconditioner can be found in the CGT method introduced by Rutishauser [21] who used the Chebyshev method as inner method to precondition the conjugate gradient method, an approach equivalent to Chebyshev polynomial preconditioning.

For the nonsymmetric case, there are various other methods, such as FGMRES [22] with GMRES as the outer method, or GMRESR [29] with GCR as the outer method and GMRES as the inner method, and others we consider further in this and the following section. If the inner method is a polynomial method, i.e., the residuals can be expressed in terms of a polynomial in  $A$  applied to the original residual  $r_0$  (which is the case for all the methods considered here), this approach can also be considered as a type of polynomial preconditioning.

Now, this type of preconditioning is defined by evaluating any occurrences of the form  $z = H_0 y$  by applying  $m + 1$  iterations of an inner iterative method to the linear system  $Az = y$ . If the inner method is a polynomial method, this can also be expressed as

$$z = P_{m,k} y. \quad (49)$$

Consequently, since the coefficients of the polynomials in general depend on the original residual used in the iterative process, which in turn depends on the vector  $y$ , we encounter a different preconditioner in each iteration step of the outer method.

Since this approach is possible for any of the methods considered in the previous sections, one can come up with a large variety of methods. We consider the performance of some of these methods, which specifically involve GCR, GMRES and EN in further detail in Section 7.

A potential drawback of the inner/outer iteration schemes is that the generated Krylov space of the outer iteration is ignored when applying the inner iterative method to  $Az = y$ . This drawback was observed by de Sturler and Fokkema [10] for the case of GMRESR. The outer method generates a minimal residual polynomial, which is ignored by the inner iteration, which searches for a new minimal residual polynomial. They therefore suggest for the inner method to solve the following equation

$$(I - C_{k-1} C_{k-1}^H) Az = y, \quad (50)$$

where  $C_{k-1}$  is the matrix consisting of the outer orthogonal vectors  $c_0, \dots, c_{k-1}$ . This approach keeps the inner residual orthogonal to the outer Krylov subspace. It turns out that in many cases it leads to far better convergence than GMRESR and in spite of a higher number of operations per iteration step it often also decreases the solution time. The new method is called GCRO.

Since EN and SEN also generate orthogonal vectors  $c_0, \dots, c_{k-1}$ , it is possible to use the same approach with EN or SEN as outer method, i.e., for every occurrence of  $w = H_0 v$ , one can apply an inner iterative method to  $(I - C_{k-1} C_{k-1}^H) Aw = v$ . Due to space limitations, we will not pursue this approach in this paper, but it is considered in [19].

```

Initial drop tolerance  $\tau$  and desired accuracy  $\varepsilon$  given
Do until ( $x_k$  is accepted):
    if (LU( $\tau$ ) exists) then
         $M \leftarrow \text{LU}(\tau)$ 
        if (not converged or too slow) then
             $\tau \leftarrow f_1(\tau)$ 
        endif
    else
         $\tau \leftarrow f_2(\tau)$ 
    endif
end

```

Fig. 2. PARASPAR.

## 6.2. PARASPAR

A hybrid software package called PARASPAR, which is based on Y12M [15,32] and combines both iterative and direct methods, has been used as a framework for some of our experiments. Direct methods, e.g., sparse Gauss elimination schemes, while achieving in general high accuracy, are often too time consuming and have only a low level of parallelism. Iterative methods have far more parallelism and, when converging, require a significantly lower computing time, but they lack the robustness of direct methods. PARASPAR takes advantage of the desirable qualities of both groups, while attempting to minimize their disadvantages.

One of PARASPAR's strengths is its robustness, which makes it a very promising code for those linear systems that have shown to be problematic for many of the existing iterative solvers and preconditioners. The elements of the package that lead to its robustness are an ILU preconditioner, which uses numerical dropping and various pivoting strategies, a sophisticated stopping criterion used for the iterative methods and an effective heuristic strategy to determine a good preconditioner. During the evaluation of the preconditioner, elements are dropped when they are below the given drop tolerance. After the user chooses an initial drop tolerance, the package will automatically decrease the drop tolerance and reevaluate the preconditioner, if the chosen preconditioner or iterative method fails, and try again, until eventually the solution has been obtained with the desired accuracy. In the worst case, the drop tolerance will be zero, and a complete LU factorization with a few steps of the iterative method is performed. The complete algorithm is given in Fig. 2.

One of the pivoting strategies is specifically designed for parallel computing. It performs a search for parallel pivots and is consequently very effective on a parallel computer.

For further efficiency, PARASPAR also has a switch to dense matrix techniques, which is used when the occurring sparse matrices become small enough or dense enough due to fill in so that dense matrix techniques are more efficient than sparse matrix techniques on the target architecture.

A more detailed description of PARASPAR, including the stopping criterion and the choice of the drop tolerance can be found in [15,32].

### 6.3. Adaptive versions

All the methods under investigation have been added to PARASPAR. In the process of doing so, we had to consider the following issues:

- the choice of stopping criteria,
- an adaptive procedure, which automatically alters the size of the underlying subspace of the considered methods,
- criteria to detect and deal with failures of the methods (i.e., breakdown, divergence or stagnation), and
- criteria to evaluate the quality of the preconditioner.

All the above points are closely related. Therefore, we treat them together in this section.

Our strategy varies somewhat for the different methods, however the overall concept is very similar.

The EN-like methods and Broyden methods naturally use a right preconditioner, as opposed to the original implementation of PARASPAR, which uses a left preconditioner for all of its iterative methods. Consequently, the vector  $r_k$  that is evaluated in each method is the actual residual and not the preconditioned residual. Therefore, it appeared to be reasonable to replace the sophisticated stopping criterion that PARASPAR uses with the simpler one

$$\frac{\|r_k\|}{\|r_0\|} \leq \varepsilon. \quad (51)$$

In order to avoid that loss of accuracy during the evaluation of the  $r_k$  of each algorithm leads to false convergence, we explicitly evaluate  $b - Ax_k$  when the above criterion indicates convergence and restart the algorithm with  $x_0 = x_k$  when  $b - Ax_k$  does not fulfill the stopping criterion.

In order to determine when to increase the subspace, we monitored the convergence rate

$$\rho_k := \frac{\|r_{k+1}\|}{\|r_k\|} \quad (52)$$

assuming that this is also a good estimate for the convergence rate

$$\eta_k := \frac{\|e_{k+1}\|}{\|e_k\|} \quad (53)$$

for GBM and GEN, since  $Ae_k = r_k$ .

We increased the subspace by one if either

$$\rho_{k+1} > \rho_k \quad (54)$$

or

$$\rho_k \geq c \quad (55)$$

where  $c$  was determined empirically. The value  $c = 0.8$  seems to be a good choice for the EN-like methods and  $c = 0.9$  for the Broyden methods and GCR. Note that the optimal  $c$  for the Broyden methods is approximately the square of the optimal  $c$  for the EN-like methods, which is in accordance with the convergence theory in Section 5.

There are several possibilities for failure of the methods such as breakdowns, caused by division by a near zero value, stagnation or divergence. Due to space limitation we do not present these

here. There are however several characteristics of the considered methods that are very useful in the context of PARASPAR. The first is the drastic divergence of some of the methods, when facing an ill-conditioned problem. This can be detected easily and save useless iterations (see Section 7.5). Another characteristic is the availability, at no extra cost, of parameters, that are indicators for the quality of the preconditioner. The latter characteristic can lead to a significant improvement in efficiency, when the iterative method fails due to a low quality preconditioner, but does not diverge drastically. A detailed discussion can be found in [19].

## 7. Numerical experiments

In this section, we illustrate some of the properties of the algorithms described in the previous sections with numerical experiments. For our experiments, we use several matrices derived from partial differential equations as well as a few test matrices from the Harwell–Boeing collection. All runs were performed on an Alliant FX/80, a parallel computer with 8 vector processors, using vectorization and parallelization. For the stopping criterion, we chose  $\varepsilon = 10^{-8}$ . Note that the implementation of GMRES here uses classical Gram–Schmidt and is therefore more efficient than the usual implementation with modified Gram–Schmidt. For the problems considered here the use of classical Gram–Schmidt did not effect the stability. The first three matrices were obtained through a standard five point finite difference discretization of the following two-dimensional partial differential equation, which was taken from [29],

$$-u_{xx} - u_{yy} + \beta(u_x + u_y) = f \quad \text{on } \Omega \quad (56)$$

with Dirichlet boundary conditions

$$u = c \quad \text{on } \partial\Omega, \quad (57)$$

where  $\Omega = [0, 1] \times [0, 1]$ .

We chose step size  $h = 0.01$ , which leads to a matrix of order 9801.

### 7.1. Problem 1

For our first problem we chose  $\beta = 1$ . This is an example for which CGS and BiCGSTAB converge fairly quickly. Since their computational complexity per iteration step and memory requirements are low, they are hard to beat in such a case. On the other hand, GMRES( $k+1$ ) converges very slowly for this case even for an optimal (with regard to time)  $k$ . We were interested to see the performance of the Broyden and EN-like methods for this example. The timings in Table 5, which were optimal for limited storage (i.e.,  $k$  smaller than 17, which corresponds for GMRES to  $k$  smaller than 34) show that whereas GEN, GBM and BBM are not competitive here, restarted EN, SEN and BEN are about 50 percent faster than GMRES. This is due to a significant reduction in additional flops, i.e., dmvs, daxpys and dotproducts, which also offsets the increase in sparse matrix–vector multiplications for restarted EN and SEN. If we include truncated versions, we see that tGEN, tGBM, tBBM as well as ORTHOMIN fail. However, tEN, tSEN and tBEN work very well and are only about 50 percent slower than CGS and BiCGSTAB. The decrease in additional flops for tBEN is caused here by the smaller computational complexity per iteration step of tBEN compared to that of tEN and tSEN.

Table 5  
Times for Problem 1

Method	Time	Number of iterations	Number of smvs	Additional flops/ $n$
CGS	13.5	246	492	3690
BiCGSTAB	13.7	236	472	4720
GMRES(33)	45.7	838	838	60336
eEN(8)	31.9	460	920	19320
eSEN(4)	29.0	466	932	14912
BEN(12)	28.7	346	692	19722
GEN(16)	60.7	603	1206	44019
GBM(8)	61.0	1375	1375	37125
BBM(12)	78.4	1673	1673	58555
tEN(3)	21.2	271	542	12466
tSEN(3)	20.3	256	512	13312
tBEN(3)	18.8	263	526	8679
tGEN, tGBM	diverge			
ORTHOMIN, tBBM	stagnate			

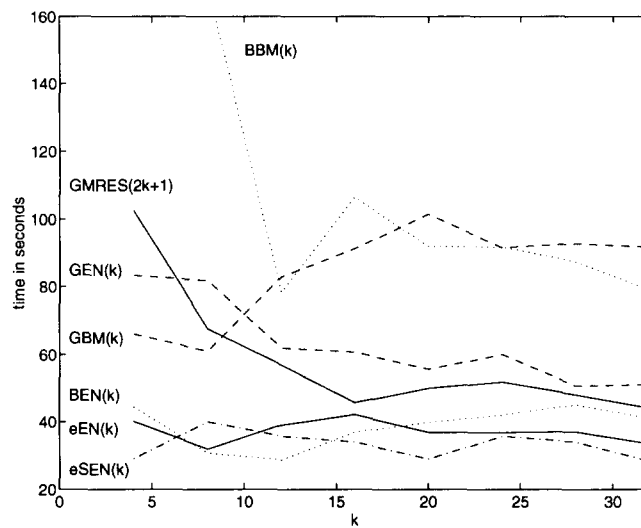
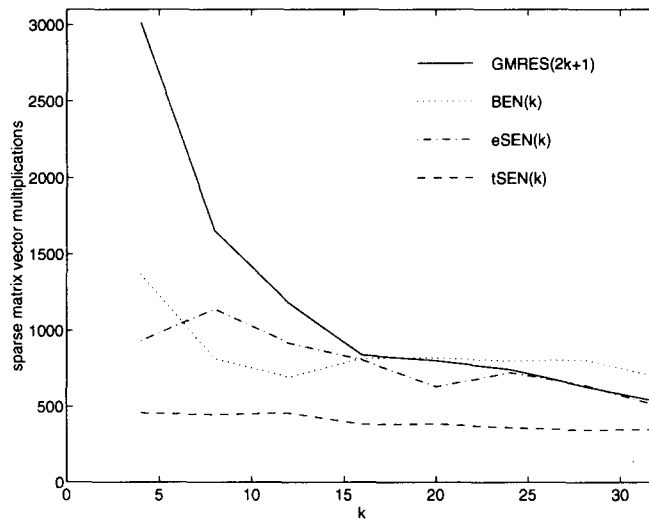
Since memory requirement can be crucial for GMRES, the Broyden and EN-like methods, times are plotted versus  $k$  in Fig. 3. These results are interesting, since now we see that if we have only a limited amount of memory available, the EN-like methods become even more attractive in comparison to GMRES. Consider for example the case  $k = 4$ , i.e., about 12 memory vectors, SEN(4) is more than three times faster than GMRES(9). Since these times are of course machine dependent and those results can vary significantly on different architectures, we also show the number of sparse matrix–vector multiplications versus  $k$  in Fig. 4 and the number of additional flops/ $n$  versus  $k$  in Fig. 5 for GMRES, BEN, eSEN and tSEN. We used tSEN as an example for the truncated method since it is here overall (i.e., for almost all  $k$ ) the best performing of the truncated methods. In Fig. 4, we see that, for  $k = 4$ , GMRES(9) needs about three times as many sparse matrix–vector multiplications as eSEN(4) and six times as many as tSEN(4). With increasing  $k$  the difference decreases. Whereas overall with increasing  $k$  the number of sparse matrix–vector multiplications decreases, this is not the case for the additional operations. For GMRES, the number of flops is high overall, for the other methods, it increases steadily with the strongest increase for the truncated method tSEN.

## 7.2. Problem 2

Even though CGS and BiCGSTAB are the fastest for the previous problem and require little storage, their robustness can be a problem. This is evident in Problem 2, where we chose  $\beta = 500$ .

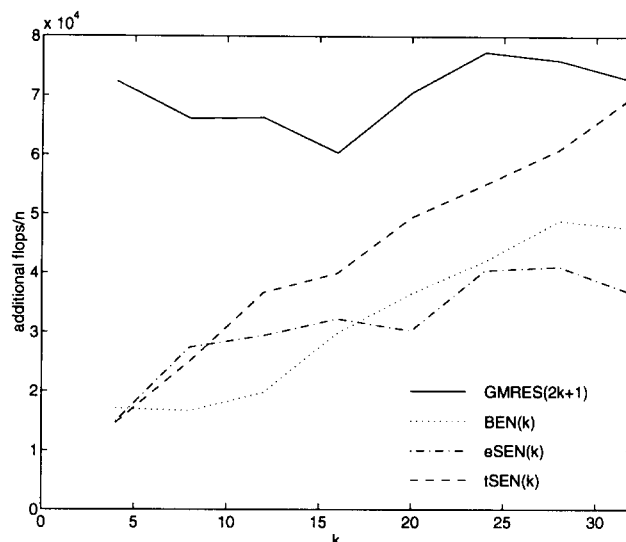
For this example, BEN and GEN fail. They both run into the situation we have also observed in Fig. 1, which leads to extremely large residuals and failure. However, use of the scaling invariant versions described in Section 2.3 with  $z = (1, \dots, 1)^T$  turns out to be successful, see Table 6. Truncation leads to slower convergence and is consequently not competitive with the restarted methods.

For this example, GMRES converges very quickly. As a matter of fact, it converges best for small  $k$ . Under these circumstances, GMRES is difficult to beat. Nevertheless, the times for eSEN with comparable memory requirement, and sBEN and eEN with a slightly larger memory requirement are competitive. Among the truncated methods, tEN is only about 30 percent slower.

Fig. 3. Times for restarted methods with varying  $k$ .Fig. 4. Number of matrix–vector operations for several methods with varying  $k$ .

### 7.3. Problem 3

The first two problems illustrated cases where BiCGSTAB and CGS do very well and GMRES very poorly and vice versa. In the first case members of the family of EN-like methods were far better than GMRES and only 50 percent slower than BiCGSTAB and CGS, in the second case some EN-like methods were competitive with the better method, GMRES. For Problem 3, we chose

Fig. 5. Number of additional flops/ $n$  for several methods with varying  $k$ .Table 6  
Times for Problem 2

Method	Time	Number of iterations	Number of smvs	Additional flops/ $n$
CGS	diverges			
BiCGSTAB	false convergence			
GMRES(3)	9.6	338	338	4056
eEN(2)	9.6	171	342	3078
eSEN(1)	10.0	177	354	3540
sBEN(2)	10.5	170	340	3234
sGEN(8)	14.9	191	382	8217
GBM(10)	14.1	295	295	9145
sGBM(2)	12.1	331	331	5631
BBM(1)	28.3	825	825	10725
tEN(2)	12.7	181	362	6154
tSEN(1)	13.7	205	410	5740
tsBEN(2)	16.8	236	472	6476
tsGEN(2)	21.8	326	652	8806
ORTHOMIN, tBBM	stagnate			
tGBM	diverges			

$$\beta(x, y) = \begin{cases} 1, & \text{if } (x, y) \in [0.4, 0.6] \times [0.4, 0.6], \\ 500, & \text{elsewhere.} \end{cases} \quad (58)$$

This is an example where eEN wins. Both CGS and BiCGSTAB fail as in the previous example, see Table 7, and GMRES converges only slowly. The truncated methods are not competitive here. However, sBEN is competitive with GMRES with a larger memory requirement. Both eSEN and eEN have superior convergence with comparable memory requirement. GMRES(17) is about 50 percent slower than eEN(8).

Table 7  
Times for Problem 3

Method	Time	Number of iterations	Number of smvs	Additional flops/ $n$
CGS	diverges			
BiCGSTAB	false convergence			
GMRES(17)	47.9	1166	1166	46640
eEN(8)	31.5	459	918	19278
eSEN(8)	41.0	578	1156	27744
sBEN(4)	48.8	730	1460	19714
GBM(8)	76.9	1722	1722	46494
tEN(8)	60.4	554	1108	58724
ORTHOMIN, tBBM	stagnate			
tGBM	diverges			

#### 7.4. Testing the adaptive schemes

The following example has been taken from the Harwell–Boeing collection. SHERMAN3 is a matrix of order 5005 with 20033 nonzeros. We preconditioned it with the incomplete LU factorization of PARASPAR using a drop tolerance of 0.0625 (pILU(0.0625)). For this example, we tested the previously mentioned adaptive versions of the methods. Fig. 6 shows the number of sparse matrix–vector multiplications versus the norm of the residuals. We see here that the best method is the adaptive ORTHOMIN, closely followed by the adaptive tEN. BiCGSTAB and GMRES(53) need about twice as many sparse matrix–vector multiplications. If we consider the solution times achieved on the Alliant FX/80, which are given in Table 8 (note that the time it took to generate the preconditioner (5.6 seconds) is not included) we see that the adaptive tEN is actually the fastest. It is followed by the adaptive tBEN, which here, in spite of a very similar memory requirement and iteration count as tSEN, is more than 10 percent faster than tSEN. We see here the influence of the larger computational complexity of tSEN. ORTHOMIN, in spite of its small smv count, is about 20 percent slower than the adaptive tEN. This is due to the fact that ORTHOMIN requires almost twice as many additional work vectors, and consequently its operation count for dense matrix–vector operations is significantly higher than for the EN-like methods.

For this example, ORTHOMIN and tSEN have a tendency to stagnate. The smallest  $k$  for which ORTHOMIN( $k$ ) does not stagnate is  $k = 48$ , for tSEN it is  $k = 30$ . Note that in both cases the adaptive version finds a  $k$  close to these values.

To compare these methods further, we restricted the memory requirement for the adaptive ORTHOMIN to that of the optimal adaptive tEN, which is a maximum of  $k = 26$ . The results for these experiments can be seen in Fig. 7. As mentioned above, ORTHOMIN( $k$ ) fails for  $k \leq 26$ . The adaptive scheme, however, converges, even though it is four times as slow as the optimal adaptive ORTHOMIN, which requires far more memory. The peak in the curve for the adaptive ORTHOMIN(26) is caused by false convergence. The residual seems to indicate convergence where the actual error stagnates, see Fig. 7. In order to overcome this problem, we reevaluated the residual by  $r_k = b - Ax_k$  and restarted at this point with the new residual, which finally leads to convergence. We also included eGCR(26) in this experiment, since the adaptive ORTHOMIN is a hybrid of a truncated and a restarted method with the emphasis on truncation. eGCR(26) converges slower than the adaptive ORTHOMIN(26), see Fig. 7, but is actually somewhat faster with regard to time here, since its

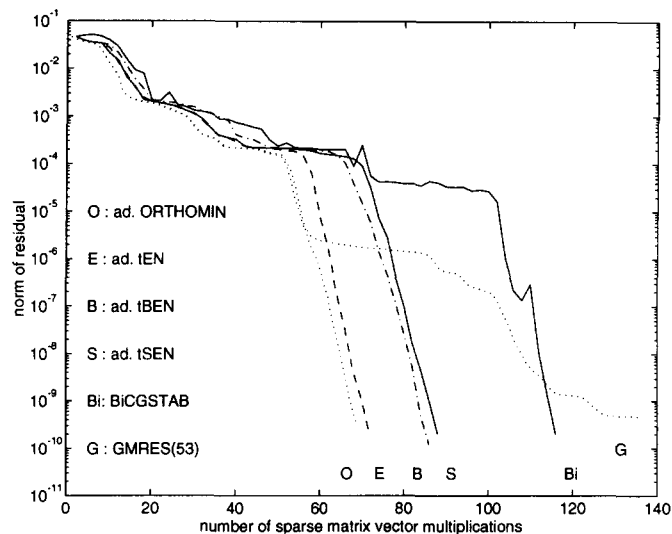


Fig. 6. Residuals for SHERMAN3 with pILU(0.0625).

Table 8

Solution times for various methods for SHERMAN3 with pILU(0.0625)

Method	Time	Number of iterations	Number of smvs	Maximal $k$
ad. ORTHOMIN	5.9	68	68	46
ad. tEN	4.9	35	70	26
ad. tBEN	5.5	42	84	30
ad. tSEN	6.4	43	86	30
BiCGSTAB	5.7	57	114	-
GMRES(53)	9.4	135	135	52
eEN(26)	9.5	81	162	26
tEN(26)	5.8	39	78	26
ORTHOMIN(48)	9.5	94	94	48
eGCR(26)	18.8	319	319	26
ORTHOMIN(26)	stagnates			26
ad. ORTHOMIN(26)	20.5	269	269	26

number of additional flops is lower. This is not true anymore, if we choose  $k = 25$  or  $k = 27$ , for which the adaptive ORTHOMIN( $k$ ) is faster than eGCR( $k$ ) (adaptive ORTHOMIN(25): 15.2 secs., eGCR(25): 19.3 secs., adaptive ORTHOMIN(27): 15.3 secs., eGCR(27): 27.4 secs.), or choose a lower accuracy for the stopping criterion (e.g.,  $\varepsilon = 10^{-6}$ , for which the adaptive ORTHOMIN(26) takes 12.7 and eGCR(26) 14.2 seconds).

The results above show that here the adaptive methods are very efficient and superior to the restarted and truncated methods. The overall fastest method is here the adaptive tEN.

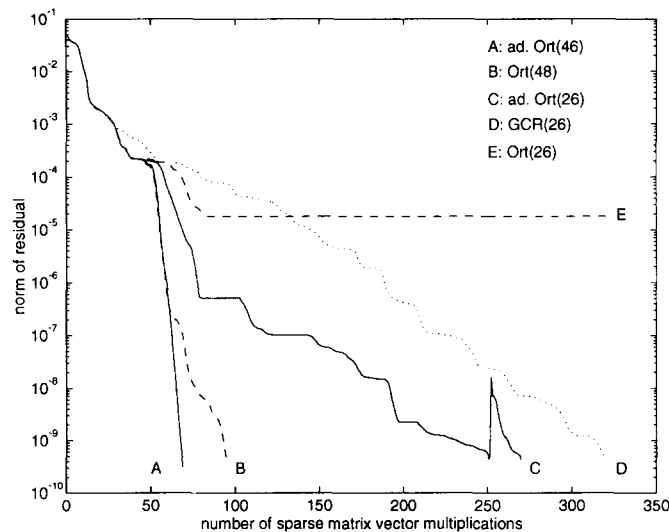


Fig. 7. Adaptive versus truncated ORTHOMIN for SHERMAN3 with pILU(0.0625).

### 7.5. Testing the methods inside PARASPAR

The following example is LNS\_3937 from the Harwell–Boeing collection, a matrix of order 3937 with 25407 nonzeros. Many standard preconditioners fail for this matrix. We use this example to demonstrate the influence of the characteristics of the various methods on the solution time in the context of PARASPAR. We use as initial drop tolerance 0.05. The resulting preconditioner is not good enough to solve the problem. Table 9 shows the reaction of the various methods to this situation. Here, the time to evaluate the preconditioners is included. Both CGS and BiCGSTAB iterate until the maximum number of iterations is reached. CGS shows a very erratic convergence behavior, which is typical for CGS with extreme peaks and is still far from the solution after 300 iterations. BiCGSTAB converges more smoothly with less serious peaks and reaches  $\|r_k\| \approx 10^{-5}$  after 300 iterations. The adaptive tEN, tBEN, tGEN and tGBM encounter a significant increase in  $\|r_k\|$  for increasing  $k$  from the very beginning, which clearly indicates divergence, and consequently the need for a new preconditioner can be detected quickly. Both the adaptive ORTHOMIN, tSEN and tBBM take far longer to determine their failure, since for these methods  $\|r_k\|$  cannot increase theoretically, a bad preconditioner very likely will lead to stagnation or extreme slow convergence, which takes far longer to detect, since we do not want to terminate the iteration process if there is still hope for convergence. We see that tGEN and tBBM do not even converge with the second preconditioner pILU( $1.56e-3$ ) and a third preconditioner evaluation is necessary, which leads to pILU( $4.88e-5$ ).

This example shows that in the context of PARASPAR the disastrous behavior of many of the EN-like methods when facing an ill-conditioned problem is of advantage, since one wastes no time in useless iterations.

In practice, PARASPAR is often used to determine a preconditioner by this adaptive procedure, which is then used for related matrices. In this case, the rapid adaptation to determine the preconditioner is not of interest, while the time to solve a system using the final preconditioner is of great interest.

Table 9

Times for LNS\_3937 with pILU(0.05) in the first trial

Method	Total time	Number of trials	Number of iterations			Last run	
			Trial 1	Trial 2	Trial 3	Solution time	$k$
CGS	42.4	2	300	22		3.1	-
BiCGSTAB	43.5	2	300	19		2.6	-
ad. tEN	17.5	2	7	15		2.3	9
ad. tBEN	18.6	2	10	21		3.2	12
ad. tSEN	35.5	2	122	19		3.0	13
ad. tGEN	37.7	3	7	37	3	0.6	2
ad. ORTHOMIN	25.5	2	115	30		2.4	15
ad. tGBM	24.6	2	13	97		9.3	50
ad. tBBM	40.6	3	59	64	15	1.8	12

We have therefore included the times of the last run and largest  $k$  used. Once again, a member of the EN-like family is preferred. With pILU(1.56e – 3), the adaptive tEN is the fastest method.

### 7.6. Preconditioning with iterative methods

Finally, we present a few experiments with regard to the use of iterative methods as inner methods. We use Problem 1, and consider two types of experiments. For the first we run some combinations of EN and GCR or GMRES, respectively, with a fixed number of inner iterations  $m$ , but the full method outside. Whereas those combinations are fairly competitive with regard to time, there is a significant difference with regard to memory requirement clearly favoring those involving EN, see Table 10. Note also that these times are below those achieved when we apply unpreconditioned CGS and BiCGSTAB to this problem, see Table 5.

The second type of experiment chooses a fixed memory requirement of 20 additional memory vectors and runs different combinations (both restarted and truncated) of the above methods, see Table 10. For comparison we have added GMRES(17) and eEN(8), which have the same memory requirement. For combinations with restarted outer methods, those involving eEN are faster than eGCR(5)/GMRES(5), which is a form of GMRESR, or the unpreconditioned GMRES(17) and, with one exception, eEN(8). For combinations with truncated outer methods, those with ORTHOMIN (tGCR) as outer method (which includes tGMRESR) fail, whereas those with tEN as outer method achieve the best times and lowest sparse matrix–vector operation counts.

Overall, the results show that the use of EN in the context of inner/outer iteration schemes is competitive with GMRESR and superior when memory is limited. See [19] for a more detailed discussion, including the orthogonality preserving methods, which were mentioned in Section 6.1.

## 8. Conclusions and future work

We introduced a new family of methods, the EN-like methods. Its complexity, relationships to other methods and convergence behavior were examined. We also considered restarted, truncated and adaptive versions. Additionally, their use in the context of inner/outer iterative methods as well as PARASPAR, a robust software package, was investigated.

Table 10  
Times for inner/outer methods

Method Inner/Outer	Time	Iterations	smv	Additional flops/ $n$	Memory vectors
GMRESR(6)	12.5	48	331	8640	107
eEN/GMRES(6)	12.6	27	368	7290	65
eGCR/eEN(5)	10.9	27	351	4860	70
eEN/eEN(5)	11.8	15	390	4590	46
eGCR(5)/GMRES(5)	37.2	203	1215	15834	20
eEN(5)/GMRES(5)	24.2	66	792	9900	20
eGCR(2)/eEN(5)	29.0	77	1001	10164	20
eGCR(5)/eEN(2)	35.2	187	1307	11220	20
eEN(2)/eEN(5)	22.1	30	770	7740	20
eEN(5)/eEN(2)	18.8	50	700	5700	20
tGMRESR, tGCR/eEN	stagnate				20
tEN(5)/GMRES(5)	15.5	40	476	7600	20
tEN(2)/eEN(5)	16.8	23	580	6302	20
tEN(5)/eEN(2)	17.3	43	602	6622	20
GMRES(17)	67.7	1652	1652	66080	20
eEN(8)	31.9	460	920	19320	20

The work shows that several members (particularly EN, SEN and to some degree BEN) of the family of EN-like methods are certainly competitive and in many cases better than other existing methods. Even though methods like CGS and BiCGSTAB may converge faster for many problems, EN-like methods are in general more robust, since, like GMRES, they have the option of increasing the Krylov subspaces. Additionally, they are often more efficient with regard to memory usage than GMRES or ORTHOMIN. Also, the experiments indicate that truncated EN-like methods seem to be less prone to stagnation or divergence than ORTHOMIN or truncated Broyden methods. Nevertheless, we also encountered problems with the truncated versions. We have dealt with these through developing an adaptive scheme, which automatically adjusts the dimension of the underlying subspace. This version will also restart the method when it encounters certain potentially fixable problems. Our experiments show that these versions work well in many cases.

In the context of inner/outer methods, we found the new methods to be competitive with GMRESR in terms of computational complexity and even superior when memory is limited.

The new methods appear to be very suitable for inclusion in a hybrid package such as PARASPAR, since they evaluate the quality of the preconditioner and can respond quickly when the preconditioner is not acceptable.

In summary, we found the new methods a viable option for large linear systems that need a fairly robust solver when memory is restricted.

There are several possibilities for future work. We have seen the importance of using restarted and truncated methods (or hybrids of the two approaches) to reduce computations and memory requirements. However, the simple truncation used thus far can have difficulties. Therefore, we plan to investigate further more sophisticated truncation schemes—possibly exploiting application-specific information—in the context of an adaptive hybrid of restarted and truncated methods.

Additionally, due to the connection of EN-like methods to Broyden methods, they can also be used

as nonlinear solvers [19], which may also be significant for some applications. This is also under investigation.

## References

- [1] M. Arioli, I. Duff and D. Ruiz, Stopping criteria for iterative solvers, Tech. Rept., CERFACS, Toulouse (1992).
- [2] S. Ashby, M. Holst, T. Manteuffel and P. Saylor, The role of the inner product in stopping criteria for conjugate gradient iterations, Tech. Rept., Lawrence Livermore National Laboratory (1992).
- [3] C. Broyden, A new method of solving nonlinear simultaneous equations, *Comput. J.* 12 (1969) 94–99.
- [4] C. Broyden, The convergence of single-rank quasi-Newton methods, *Math. Comput.* 24 (1970) 365–382.
- [5] C. Broyden, J. Dennis and J. More, On the local and superlinear convergence of quasi-Newton methods, *J. Inst. Math. Appl.* 12 (1973) 223–245.
- [6] J. Demmel, M. Heath and H. van der Vorst, Parallel numerical linear algebra, *Acta Numer.* (1993) 111–197.
- [7] J. Dennis Jr and J. More, Quasi-Newton methods, motivation and theory, *SIAM Rev.* 1 (1977) 46–89.
- [8] J. Dennis Jr and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* (Prentice-Hall, Englewood Cliffs, NJ, 1983).
- [9] E. de Sturler, Nested Krylov methods based on GCR, Tech. Rept. 93-50, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft (1993); also: *J. Comput. Appl. Math.* (to appear).
- [10] E. de Sturler and D. Fokkema, Nested Krylov methods and preserving the orthogonality, in: T. Manteuffel and S. McCormick, eds., *Proceedings of the Sixth Copper Mountain Multigrid Conference on Multigrid Methods*, VA (1993).
- [11] P. Deufhard, R. Freund and A. Walter, Fast secant methods for the iterative solution of large nonsymmetric linear systems, *Impact Comput. Sci. Engrg.* 2 (1990) 244–276.
- [12] T. Eirola and O. Nevanlinna, Accelerating with rank-one updates, *Linear Algebra Appl.* 121 (1989) 511–520.
- [13] S. Eisenstat, H. Elman and M. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, *SIAM J. Numer. Anal.* (1983) 345–357.
- [14] H. Elman, Iterative methods for large, sparse, nonsymmetric systems of linear equations, Res. Rept. 229, Yale University, New Haven, CT (1982).
- [15] K. Gallivan, A. Sameh and Z. Zlatev, A parallel hybrid sparse linear system solver, *Comput. Systems Engrg.* 1 (1990) 183–195.
- [16] D. Gay, Some convergence properties of Broyden's method, *SIAM J. Numer. Anal.* 16 (1979) 623–630.
- [17] D. Gay and R. Schnabel, Solving systems of nonlinear equations by Broyden's method with projected updates, in: O. L. Mangasarian, R. Meyer and S. Robinson, eds., *Nonlinear Programming Vol. 3* (Academic Press, New York, 1978) 245–281.
- [18] D. Luenberger, *Linear and Nonlinear Programming* (Addison-Wesley, Reading, MA, 1984).
- [19] U. Meier Yang, A family of preconditioned iterative solvers for sparse linear systems, Ph.D. Thesis, Report UIUCDCS-R-95-1904, Department of Computer Science, University of Illinois, Urbana, IL (1995) (also available as CSRD-Report 1408 through anonymous ftp to sp2.csrd.uiuc.edu in directory CSRD-Info/reports).
- [20] D. O'Leary, Why Broyden's nonsymmetric method terminates on linear equations, Tech. Rept. CS-TR-3045, University of Maryland, College Park, MD (1993); also: *SIAM J. Optim.* (to appear).
- [21] H. Rutishauser, Theory of gradient methods, *Mitt. Inst. Angew. Math.* 8 (1959) 24–29.
- [22] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, Tech. Rept., Computer Science Department and MSI, University of Minnesota, Minneapolis, MN (1991).
- [23] Y. Saad and M. Schultz, Conjugate gradient-like algorithms for solving nonsymmetric linear systems, *Math. Comput.* 44 (1985) 417–424.
- [24] Y. Saad and M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear system, *SIAM J. Sci. Statist. Comput.* 7 (1986) 856–869.
- [25] G. Schultz, Iterative Berechnung der reziproken Matrix, *Z. Angew. Math. Mech.* 13 (1933) 57–59.
- [26] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 10 (1989) 36–52.
- [27] H. van der Vorst, BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 13 (1992) 631–644.

- [28] H. van der Vorst, Conjugate gradient type methods for nonsymmetric linear systems, in: R. Beauwens and P. de Groen, eds., *Iterative Methods in Linear Algebra* (North-Holland, Amsterdam, 1992) 67–76.
- [29] H. van der Vorst and C. Vuik, GMRESR: A family of nested GMRES methods, *Numer. Linear Algebra Appl.* (to appear).
- [30] C. Vuik, Further experiences with GMRESR, Tech. Rept. 92-12, Delft University of Technology, Delft (1992).
- [31] C. Vuik and H. van der Vorst, A comparison of some GMRES-like methods, *Linear Algebra Appl.* 160 (1992) 131–162.
- [32] Z. Zlatev, *Computational Methods for General Sparse Matrices* (Kluwer Academic Publishers, Dordrecht, 1991).