Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational and Applied Mathematics 213 (2008) 387–399

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

A new implementation of the CMRH method for solving dense linear systems

M. Heyouni, H. Sadok*

L.M.P.A, Université du Littoral, 50 rue F. Buisson BP699, F-62228 Calais Cedex, France

Received 30 June 2006; received in revised form 9 January 2007

Abstract

The CMRH method [H. Sadok, Méthodes de projections pour les systèmes linéaires et non linéaires, Habilitation thesis, University of Lille1, Lille, France, 1994; H. Sadok, CMRH: A new method for solving nonsymmetric linear systems based on the Hessenberg reduction algorithm, Numer. Algorithms 20 (1999) 303–321] is an algorithm for solving nonsymmetric linear systems in which the Arnoldi component of GMRES is replaced by the Hessenberg process, which generates Krylov basis vectors which are orthogonal to standard unit basis vectors rather than mutually orthogonal. The iterate is formed from these vectors by solving a small least squares problem involving a Hessenberg matrix. Like GMRES, this method requires one matrix–vector product per iteration. However, it can be implemented to require half as much arithmetic work and less storage. Moreover, numerical experiments show that this method performs accurately and reduces the residual about as fast as GMRES. With this new implementation, we show that the CMRH method is the only method with long-term recurrence which requires not storing at the same time the entire Krylov vectors basis and the original matrix as in the GMRES algorithm. A comparison with Gaussian elimination is provided.

Keywords: Linear system; Krylov method; Hessenberg process; Dense matrices

1. Introduction

In this paper we are concerned with the solution of dense, non-Hermitian linear systems by the CMRH iterative method [30,31]. The systems that we consider arise in the solution of many scientific applications such as boundary element methods [4,8,10], integral [17,2], elastohydrodynamic lubrication problems [12,13], quantum mechanical problems [27,32], economic models [11], large least squares problems [5].

In [30,31] the CMRH algorithm is described as an alternative method to the generalized minimal residual (GMRES) [29] and quasi-minimal residual (QMR) algorithms [14,24]. These three methods are Krylov subspace methods for solving linear systems and can be derived as particular cases of the Generalized Hessenberg method [21]. The main difference between these methods is in the generation of the basis vectors for the Krylov subspace. The GMRES algorithm uses the Arnoldi process [3] which constructs an orthonormal basis and whose work and storage requirements grow linearly with iterations. The QMR algorithm uses the Lanczos process [23], which has low storage and constant

* Corresponding author. Tel.: +33 3 21 46 55 76; fax: +33 3 21 46 55 77.
E-mail addresses: heyouni@lmpa.univ-littoral.fr (M. Heyouni), sadok@lmpa.univ-littoral.fr (H. Sadok).

0377-0427/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2007.01.004

work per iteration. But this process can suffer from a possible breakdown or a near breakdown and one must use look ahead strategies for avoiding such problems [7,15,25].

The CMRH method is based on the Hessenberg process [22], which requires less arithmetic work and storage than Arnoldi's process since it builds, at iteration k, a lower trapezoidal basis $\{l_1, \ldots, l_k\}$ such that l_i has $i-1$ components equal to zero and one component equal to one. Moreover, we have to perform, as in the Arnoldi process, a matrix–vector product Al_i which has a lower cost. Notice also that, when the matrix is large and sparse, the computational and storage requirements in the CMRH and GMRES algorithms grow with the iteration. So to address such problems, these methods are used iteratively, i.e., the CMRH and GMRES algorithms are restarted every m steps, where m is some fixed integer parameter [29,31]. Unfortunately, this strategy slows the convergence of the methods and can make them stagnate in some situations [19,31].

For dense matrices, to overcome storage constraints and to have nice monotonic convergence properties, we propose a new implementation of the CMRH algorithm without a restarting strategy. The new algorithm is based on the Hessenberg process with over-storage (i.e., we overwrite progressively the columns of the matrix A of the linear system by the nonzero entries of the Hessenberg matrix and the nonzero components of the Krylov basis $\{l_1, \ldots, l_k\}$).

The outline of the paper is as follows. In next section, we describe the Hessenberg process and give some fundamental properties. Then an implementation with over-storage of the Hessenberg process is described in Section 3. In Section 4, we review the CMRH method and give the new implementation which is used to solve dense linear systems. Finally we present in Section 5, some numerical experiments in order to compare the CMRH method with over-storage with the Gaussian elimination method and we conclude with some remarks and open problems.

Throughout the paper, we adopt the following notations: uppercase, respectively lowercase, letters denotes matrices, respectively vectors, except for special illustration. A superscript on a matrix or vector denotes an iteration number. Columns of a matrix are indexed by subscript; elements of a matrix have row and column indices as subscripts. For a vector v, $\|v\|$ denotes to Euclidean norm $\|v\| = \sqrt{v^{\mathrm{T}}v}$ and $\|v\|_\infty$ is the maximum norm $\|v\|_\infty = \max_{j=1,\ldots,n}|(v)_j|$, where v^{T} is the transpose of v and $(v)_j$ is the jth component of the vector v. I_k is the $k \times k$ identity matrix or simply I whenever its dimension is clear from the context; e_j is its jth column. We also use MATLAB-like notations $A_{i:j,k:l}$, respectively, $(v)_{i:j}$, to denote the submatrix of A consisting of the intersections of rows i to j and columns k to l, respectively, the vector of components $(v)_i, \ldots, (v)_j$, and when $i : j$ is replaced by :, it means all rows, similarly for columns. The "$\leftrightarrow$" symbol means "swap contents": $x \leftrightarrow y \Leftrightarrow t = x;\ x = y;\ y = t$.

2. The Hessenberg process

In [18], the Hessenberg process is described as an algorithm for computing the characteristic polynomial of a given matrix A. This process can also be used for the reduction to the Hessenberg form of A and is presented as an oblique projection in [33].

For ease of notation we will assume that the matrix and the vectors involved in the solution algorithms are real, but the results given here and in other sections are easily modified for a complex matrix and complex vectors.

Let v be a column vector of $\mathbb{R}^n$ and A an $n \times n$ real matrix. The Hessenberg reduction process (without pivoting strategy) computes a unit trapezoidal matrix $L_m = [l_1, \ldots, l_m]$ whose columns form a basis of the Krylov subspace $K_m(A, v) \equiv \mathrm{span}\{v, Av, \ldots, A^{k-1}v\}$ by using the following formulas:

$$\begin{cases} \beta = (v)_1, & l_1 = v/\beta, \\ h_{k+1,k} l_{k+1} = Al_k - \sum\limits_{j=1}^{k} h_{j,k} l_j & \text{for } k = 1, \ldots, m. \end{cases}$$

The parameters $h_{j,k}$ are determined such that

$$l_{k+1} \perp e_1, \ldots, e_k \quad \text{and} \quad (l_{k+1})_{k+1} = 1. \tag{1}$$

Suppose that $\{l_i\}_{i=1,\ldots,k}$ have been computed such that the $i-1$ first components of l_i equal zero and the ith component equals one. To obtain l_{k+1}, we first compute $u = Al_k$ and then we subtract multiples of $l_1, \ldots, l_k$ in order to annihilate the components $1, \ldots, k$ of the u to obtain the $w = Al_k - \sum_{i=1}^{k} h_{i,k} l_i$. Finally we choose $h_{k+1,k} = (w)_{k+1}$ and took $l_{k+1} = w/h_{k+1,k}$.

Algorithm 1 summarizes the Hessenberg process in its standard form, i.e., without pivoting.

Algorithm 1. *Hessenberg process*

```
(1)  $\beta = (v)_1$; $l_1 = v/\beta$;
(2)  for $k = 1, \ldots, m$
         $u = Al_k$;
         for $j = 1, \ldots, k$
            $h_{j,k} = (u)_j$; $u = u - h_{j,k} l_j$;
         end
         $h_{k+1,k} = (u)_{k+1}$; $l_{k+1} = u/h_{k+1,k}$;
     end
```

Now it is important that the entry $h_{k+1,k}$ never becomes zero. If this occurs —such situation is called a breakdown— the Hessenberg process cannot proceed. In addition, small values of $h_{k+1,k}$ can cause severe loss of accuracy. To avoid such a breakdown and also ensure numerical stability, the process can be modified to include a pivoting strategy such as in Gaussian elimination method. This is done by replacing the orthogonality condition (1) by the following one:

$$l_{k+1} \perp e_{p_1}, \ldots, e_{p_k} \quad \text{and} \quad (l_{k+1})_{p_{k+1}} = 1, \tag{2}$$

where $p_j \in \{1, 2, \ldots, n\}$.

To compute p_{k+1}, we follow the practical procedure described in [33]. We suppose that $p_1, \ldots, p_k$ have already been obtained, then we compute $u = Al_k$ and subtract multiples of $l_1, \ldots, l_k$ in order to annihilate the k components $p_1, \ldots, p_k$ of the vector u to obtain a vector $w = Al_k - \sum_{i=1}^{k} h_{i,k} l_i$. Then we set $p_{k+1} = i_0$, where i_0 satisfies $\|w\|_\infty = |(w)_{i_0}|$. Finally we normalize the vector l_{k+1} by taking $h_{k+1,k} = (w)_{i_0}$ and $l_{k+1} = w/(w)_{i_0}$.

Notice that If $\|w\|_\infty = 0$ at step k, then, in exact arithmetic, the minimal polynomial with respect to the vector v has the degree k which means that we have constructed an invariant subspace and the process must be stopped.

Using the pivoting strategy described above, the Hessenberg process is reproduced in Algorithm 2.

Algorithm 2. *Hessenberg process with pivoting strategy*

```
(1)  $p = [1, 2, \ldots, n]^T$;
     Determine $i_0$ such that $|(v)_{i_0}| = \|v\|_\infty$;
     $\beta = (v)_{i_0}$; $l_1 = v/\beta$; $p_1 \leftrightarrow p_{i_0}$;
(2)  for $k = 1, \ldots, m$
        $u = Al_k$;
        for $j = 1, \ldots, k$
           $h_{j,k} = (u)_{p_j}$; $u = u - h_{j,k} l_j$;
        end
        If ($k < n$ and $u \neq 0$) then
           Determine $i_0 \in \{k+1, \ldots, n\}$ such that $|(u)_{p_{i_0}}| = \|(u)_{p_{k+1}:p_n}\|_\infty$;
           $h_{k+1,k} = (u)_{p_{i_0}}$; $l_{k+1} = u/h_{k+1,k}$; $p_{k+1} \leftrightarrow p_{i_0}$;
        else
           $h_{k+1,k} = 0$; Stop.
        end
     end
```

Letting L_k be the $n \times k$ matrix with column vectors $l_1, \ldots, l_k$, $\overline{H}_k$ be the $(k+1) \times k$ upper Hessenberg matrix whose nonzero entries are the $h_{j,k}$ and by H_k the matrix obtained from $\overline{H}_k$ by deleting its last row. Then it is easy to show that these matrices given either by Algorithms 1 or 2 satisfy the well-known formulas

$$AL_k = L_{k+1} \overline{H}_k, \tag{3}$$
$$= L_k H_k + h_{k+1,k} l_{k+1} e_k^T \tag{4}$$

and $\mathscr{P}_k L_k$ is lower trapezoidal where $\mathscr{P}_k^T = [e_{p_1}, e_{p_2}, \ldots, e_{p_n}]$ and the p_i's (for $i = 1, \ldots, n$) are defined in Algorithm 2. Below, we apply the two previous algorithms on two simple examples.

Example 1. Consider the matrix A and the vector v given by

$$A=\begin{bmatrix}1&2&0&-1\\0&1&-1&2\\-2&0&2&1\\-1&1&0&2\end{bmatrix},\quad v=\begin{bmatrix}1\\7\\8\\9\end{bmatrix}.$$

- Algorithm 1 applied to A and v breaks down at step $k=2$ and gives the iterates

$$L_2=\begin{bmatrix}1&0\\7&1\\8&1\\1&\frac{6}{5}\end{bmatrix}\quad\text{and}\quad \overline{H}_2=\begin{bmatrix}6&\frac{4}{5}\\-25&-\frac{16}{5}\\0&0\end{bmatrix}.$$

- Algorithm 2 terminates at step 3 and gives the iterates

$$L_3=\begin{bmatrix}\frac{1}{9}&1&0\\\frac{7}{9}&-\frac{1}{2}&1\\\frac{8}{9}&\frac{1}{2}&1\\1&0&0\end{bmatrix},\quad \overline{H}_3=\begin{bmatrix}\frac{8}{3}&-\frac{3}{2}&1\\\frac{10}{27}&\frac{1}{6}&\frac{17}{9}\\0&\frac{1}{4}&\frac{1}{6}\\0&0&0\end{bmatrix}\quad\text{and}\quad p=\begin{bmatrix}4\\1\\3\\2\end{bmatrix}.$$

This result indicates that the degree of the minimal polynomial with respect to A and v is equal to 3. Moreover, the permutation vector p defines the following permutation matrix:

$$\mathscr{P}_3=\begin{bmatrix}0&0&0&1\\1&0&0&0\\0&0&1&0\\0&1&0&0\end{bmatrix},$$

which gives that $\mathscr{P}_3L_3$ is a unit lower trapezoidal matrix since

$$\mathscr{P}_3L_3=\begin{bmatrix}1&0&0\\\frac{1}{9}&1&0\\\frac{8}{9}&\frac{1}{2}&1\\\frac{7}{9}&-\frac{1}{2}&1\end{bmatrix}.$$

Example 2. Consider the matrix $\hat{A}=\mathscr{P}_3A\mathscr{P}_3$ and the vector $\hat{v}=\mathscr{P}_3v$ then Algorithms 1 and 2 applied to the pair $(\hat{A},\hat{v})$ both terminate at step 3, produce the same iterates which are

$$L_3=\begin{bmatrix}1&0&0\\\frac{1}{9}&1&0\\\frac{8}{9}&\frac{1}{2}&1\\\frac{7}{9}&-\frac{1}{2}&1\end{bmatrix},\quad \overline{H}_3=\begin{bmatrix}\frac{8}{3}&-\frac{3}{2}&1\\\frac{10}{27}&\frac{1}{6}&\frac{17}{9}\\0&\frac{1}{4}&\frac{1}{6}\\0&0&0\end{bmatrix}\quad\text{and}\quad p=\begin{bmatrix}1\\2\\3\\4\end{bmatrix}.$$

3. The Hessenberg process with over-storage

In this section, we will show that the three arrays needed to store the matrices A, L_k and $\overline{H}_k$ are not really needed. To minimize memory use on the computer, both L_k and $\overline{H}_k$ can be written into the same array as A. In fact, we will modify Algorithm 2 so that the entries of the k first columns of A could be overwritten by those of L_k and H_k.

Before describing the new implementation of the Hessenberg process that will allow us to overwrite A by L_k and $\overline{H}_k$, let us give some important remarks.

We note that if k steps of Algorithm 1 are performed, then L_k is unit lower trapezoidal matrix. Moreover, when computing $u = Al_k$ the $k-1$ first columns of A are not used. Hence, to minimize memory requirements, the upper triangular part of $\overline{H}_k$, respectively the lower part of L_k, can be stored in the upper triangular part, respectively in the lower triangular part of A. And we have to use an additional vector h of length k to store the sub-diagonal of $\overline{H}_k$.

Unfortunately, in practice and as explained earlier, we cannot use the previous remarks with Algorithm 1 since it can suffer from a possible breakdown and from a loss of accuracy. Moreover, these remarks does not hold for Algorithm 2 since it not gives a lower trapezoidal matrix L_k. In fact, after k steps of Algorithm 2, each ith column l_i is normalized ($\|l_i\|_\infty = 1$) and has $i-1$ zero components and one component equal to one. Note also that if $\mathscr{P}$ is the $n \times n$ permutation matrix defined by the permutation vector p given at the end of step k, then we can easily check that $\mathscr{P}L_k$ is a unit lower trapezoidal matrix.

More precisely, let $p^{(k+1)}$, $L_{k+1} = [L_k, l_{k+1}]$ and $\overline{H}_k$ be, respectively, the permutation vector, the Hessenberg basis, the upper Hessenberg matrix obtained after k steps of the Hessenberg process with pivoting strategy. Clearly, if $\mathscr{P}_{k+1}^{\mathrm{T}} = [e_{p_1}, \ldots, e_{p_n}]$ is the permutation matrix defined by $p^{(k+1)}$ then by noticing that $\mathscr{P}_{k+1}^{\mathrm{T}}\mathscr{P}_{k+1} = I_n$ and premultiplying (4) by $\mathscr{P}_{k+1}$ we have

$$A^{(k+1)}\hat{L}_k^{(k+1)} = \hat{L}_k^{(k+1)}H_k + h_{k+1,k}\hat{l}_{k+1}^{(k+1)}e_k^{\mathrm{T}}, \tag{5}$$

where $A^{(k+1)} = \mathscr{P}_{k+1}A\mathscr{P}_{k+1}^{\mathrm{T}}$, $\hat{L}_k^{(k+1)} = \mathscr{P}_{k+1}L_k$ and $\hat{l}_{k+1}^{(k+1)} = \mathscr{P}_{k+1}l_{k+1}$.

Notice that now $\hat{L}_k^{(k+1)}$ is a unit lower trapezoidal matrix and let $\hat{v}^{(k+1)} = \mathscr{P}_{k+1}v$, then equality (5) allows us to give the following result

Theroem 1. *Suppose that k steps of Algorithm 2 are applied to the pair (A, v) to obtain the Hessenberg basis L_{k+1}, the Hessenberg matrix $\overline{H}_k$ and the permutation vector $p^{(k+1)}$, then k steps of Algorithm 1 can be applied without encountering any breakdown to the pair $(A^{(k+1)}, v^{(k+1)})$. Moreover, the obtained Hessenberg basis and Hessenberg matrix are, respectively, $\hat{L}_{k+1}^{(k+1)} = \mathscr{P}_{k+1}L_{k+1}$ and $\overline{H}_k$, where $\mathscr{P}_{k+1}$ is the permutation matrix defined by $p^{(k+1)}$.*

Now, we are able to describe a new implementation of the Hessenberg process which progressively constructs the matrices $A^{(k+1)}$, $\hat{L}_k^{(k+1)}$ and $\overline{H}_k$. This new implementation called the Hessenberg process with over-storage will help us to overwrite the k first columns of $A^{(k+1)}$ by those of $\hat{L}_k^{(k+1)}$ and $\overline{H}_k$.

Let $A^{(0)} = A$, $v^{(0)} = v$ and $p^{(0)} = p = [1, 2, \ldots, n]^{\mathrm{T}}$ stored, respectively, in $\hat{A}$, $\hat{v}$ and $\hat{p}$. Then the step $k = 0$ of the Hessenberg process with over-storage looks like this:

- scan the vector $\hat{v}$ to identify $\beta = |(\hat{v})_{i_0}| = \|\hat{v}\|_\infty$ the largest element in magnitude and its index i_0 and store the scalar β in $(h)_1$. Define $\hat{l}_1^{(0)} = \hat{v}/(\hat{v})_{i_0}$ which is stored in $\hat{v}$ and interchange the components 1 and i_0 of the permutation vector p to obtain $p^{(1)}$ which is stored in $\hat{p}$.
  Let P_1, respectively $\mathscr{P}_1$, be the permutation matrix obtained by interchanging the rows 1 and i_0 of the identity matrix, respectively, defined by the vector $p^{(1)}$;
- form $\hat{l}_1^{(1)} = P_1\hat{l}_1^{(0)}$, respectively $A^{(1)} = P_1\hat{A}P_1$, by interchanging the components 1 and i_0 of the vector $\hat{v}$, respectively, by interchanging the rows and columns 1 and i_0 of $\hat{A}$. The arrays $\hat{l}_1^{(1)}$ and $A^{(1)}$ are respectively stored in $\hat{v}$ and $\hat{A}$.

Now, we suppose that $k-1$ steps of the new process were performed and that we obtained the basis $\hat{L}_k^{(k)} = [\hat{l}_1^{(k)}, \hat{l}_2^{(k)}, \ldots, \hat{l}_k^{(k)}]$, the Hessenberg matrix $\overline{H}_{k-1}$ and the permutation vector $p^{(k)}$ satisfying $\hat{l}_i^{(k)} = [0, \ldots, 0, 1, (l_i)_{p_{i+1}^{(k)}}, \ldots, (l_i)_{p_n^{(k)}}]^{\mathrm{T}} = \mathscr{P}_k l_i$ where l_i is the ith column vector of the basis L_k constructed by performing $k-1$ steps of Algorithm 2 and $\mathscr{P}_k$ is the permutation matrix defined by $p^{(k)}$.

We also assume that $\hat{l}_k^{(k)}$ is stored in $\hat{v}$ and that the $n-k+1$ last columns of the matrix $A^{(k)} = \mathscr{P}_k A \mathscr{P}_k^{\mathrm{T}}$, the nonzeros entries of $\hat{L}_{k-1}^{(k)} = \mathscr{P}_k L_{k-1}$ and $\overline{H}_{k-1}$, are stored in the $k-1$ first columns of the array $\hat{A}$ as given below

$$\hat{A} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,k-2} & h_{1,k-1} & a_{p_1^{(k)},p_k^{(k)}} & \cdots & a_{p_1^{(k)},p_n^{(k)}} \\ (l_1)_{p_2^{(k)}} & \ddots & h_{2,k-2} & h_{2,k-1} & a_{p_2^{(k)},p_k^{(k)}} & \cdots & a_{p_2^{(k)},p_n^{(k)}} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (l_1)_{p_{k-1}^{(k)}} & \cdots & (l_{k-2})_{p_{k-1}^{(k)}} & h_{k-1,k-1} & a_{p_{k-1}^{(k)},p_k^{(k)}} & \cdots & a_{p_{k-1}^{(k)},p_n^{(k)}} \\ (l_1)_{p_k^{(k)}} & \cdots & (l_{k-2})_{p_k^{(k)}} & (l_{k-1})_{p_k^{(k)}} & a_{p_k^{(k)},p_k^{(k)}} & \cdots & a_{p_k^{(k)},p_n^{(k)}} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (l_1)_{p_n^{(k)}} & \cdots & (l_{k-2})_{p_n^{(k)}} & (l_{k-1})_{p_n^{(k)}} & a_{p_n^{(k)},p_k^{(k)}} & \cdots & a_{p_n^{(k)},p_n^{(k)}} \end{bmatrix}.$$

Using Theorem 2, the kth step of the new process consists in applying the kth step of Algorithm 2 to the pair $(A^{(k)}, \hat{l}_k^{(k)})$. Hence, the kth step is described as follows:

- Compute the vector $\hat{u} = A^{(k)}\hat{l}_k^{(k)}$ by noticing that $(\hat{l}_k^{(k)})_{1:k} = [0_{1:k-1}, 1]^{\mathrm{T}}$ and the $n-k+1$ last columns of the matrix $A^{(k)}$ are stored, respectively, in $\hat{v}$ and in the $n-k+1$ last columns of the array $\hat{A}$ and so

  $$\hat{u} = \hat{A}_{:,k} + \hat{A}_{:,k+1:n}(\hat{v})_{k+1:n}.$$

  Since the kth column of $\hat{A}$ will not be used in the next steps, we can save $(\hat{l}_k^{(k)})_{k+1:n}$ in $\hat{A}_{k+1:n,k}$, i.e., we can save $(\hat{v})_{k+1:n}$ in $\hat{A}_{k+1:n,k}$.
- Subtract successively multiples of $\hat{l}_1^{(k)}, \ldots, \hat{l}_k^{(k)}$—which are stored in $\hat{A}_1, \ldots, \hat{A}_k$—in order to annihilate the k first components of the vector $\hat{u}$ to obtain the new vector $\hat{u} = A^{(k)}\hat{l}_k^{(k)} - \sum_{j=1}^{k} h_{j,k}\hat{l}_j^{(k)}$. Save $[h_{1,k}, \ldots, h_{k,k}]^{\mathrm{T}}$ and $\hat{w}$, respectively, in $\hat{A}_{1:k,k}$ and $\hat{v}$.

  $$\hat{A}_{j,k} = (\hat{u})_j;\ (\hat{u})_j = 0;\ (\hat{u})_{j+1:n} = (\hat{u})_{j+1:n} - \hat{A}_{j,k}\hat{A}_{j+1:n,j};\ \text{ for } j = 1, \ldots, k.$$

- Scan $\hat{u}$ to identify $h_{k+1,k}$ the largest element in magnitude and its index i_0 and store $h_{k+1,k}$ in $(h)_{k+1}$. Then normalize $\hat{u}$ by $\hat{v} = \hat{u}/(h_{k+1,k} = (\hat{u})_{i_0})$ to obtain $\hat{l}_{k+1}^{(k)}$ and interchange the components $k+1$ and i_0 of the permutation vector p to obtain $p^{(k+1)}$.
  Let P_{k+1} be the permutation matrix obtained by interchanging the rows $k+1$ and i_0 of the identity matrix.
- Form $\hat{l}_{k+1}^{(k+1)} = P_{k+1}\hat{v}$, respectively, $\hat{A} = P_{k+1}\hat{A}P_{k+1}$ by interchanging the components $k+1$ and i_0 of the vector $\hat{v}$, respectively, by interchanging the rows and columns $k+1$ and i_0 of $\hat{A}$. Notice that as $i_0 \in \{k+1, \ldots, n\}$ then the entries of the upper triangular part of H_k are not affected by permutation matrices. Hence, at the end of the kth step of the new process we obtain the new array $\hat{A}$ given below

$$\hat{A} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,k-1} & h_{1,k} & a_{p_1^{(k+1)},k+1} & \cdots & a_{p_1^{(k+1)},n} \\ (l_1)_{p_2}^{(k+1)} & \ddots & h_{2,k-1} & h_{2,k} & a_{p_2^{(k+1)},k+1} & \cdots & a_{p_2^{(k+1)},n} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (l_1)_{p_k^{(k+1)}} & \cdots & (l_{k-1})_{p_k^{(k+1)}} & h_{k,k} & a_{p_k^{(k+1)},k+1} & \cdots & a_{p_k^{(k+1)},n} \\ (l_1)_{p_{k+1}^{(k+1)}} & \cdots & (l_{k-1})_{p_{k+1}^{(k+1)}} & (l_k)_{p_{k+1}^{(k+1)}} & a_{p_{k+1}^{(k+1)},k+1} & \cdots & a_{p_{k+1}^{(k+1)},n} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (l_1)_{p_n^{(k+1)}} & \cdots & (l_{k-1})_{p_n^{(k+1)}} & (l_k)_{p_n^{(k+1)}} & a_{p_n^{(k+1)},k+1} & \cdots & a_{p_n^{(k+1)},n} \end{bmatrix}.$$

Finally, the new Hessenberg process is summarized as follows.

Algorithm 3. *Hessenberg process with over-storage*

```
(1) $p = [1, \cdots, n]^T$;
    Determine $i_0$ such that $|(v)_{i_0}| = \|v\|_\infty$; $\beta = (v)_{i_0}$; $v = v/\beta$;
    $p_{i_0} \longleftrightarrow p_1$; $(v)_{i_0} \longleftrightarrow (v)_1$;
    $A_{i_0,:} \longleftrightarrow A_{1,:}$; $A_{:,i_0} \longleftrightarrow A_{:,1}$;
(2) for $k = 1, \ldots, m$
      $u = A_{:,k} + A_{:,k+1:n}(v)_{k+1:n}$; $A_{k,k+1:n} = (v)_{k+1:n}$;
      for $j = 1, \ldots, k$
        $A_{j,k} = (u)_j$; $(u)_j = 0$;
        $(u)_{j+1:n} = (u)_{j+1:n} - A_{j,k}A_{j+1:n,j}$;
      end
      Determine $i_0$ such that $|(u)_{i_0}| = \|u\|_\infty$;
      $(h)_{k+1} = (u)_{i_0}$;$v = u/(h)_{k+1}$;
      $p_{i_0} \longleftrightarrow p_{k+1}$; $(v)_{i_0} \longleftrightarrow (v)_{k+1}$;
      $A_{i_0,:} \longleftrightarrow A_{k+1,:}$; $A_{:,i_0} \longleftrightarrow A_{:,k+1}$;
    end
```

4. The CMRH method

Let A be an n by n nonsingular matrix, b a given n-vector and consider the following system of linear equations:

$$Ax = b. \tag{6}$$

Given an initial guess x_0 for the exact solution $x^* = A^{-1}b$, the CMRH method [31,21] constructs approximate solutions $\{x_k\}_{k=1,\ldots,m}$ of the form

$$x_k = x_0 + w_k, \quad w_k \in K_k(A, r_0), \tag{7}$$

where $r_0 \equiv b - Ax_0$ is the initial residual and $K_k(A, r_0) = \mathrm{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$. The CMRH method, first described in [30,31], is a general projection method which is based on the Hessenberg process with pivoting strategy described in Section 2.

Letting L_k be the $n \times k$ matrix computed by the Hessenberg process applied to the pair (A, r_0) and as the columns of L_k form a basis of the Krylov subspace $K_k(A, r_0)$, we give the correction w_k under the form $w_k = L_k d_k$ where $d_k \in \mathbb{R}^k$.

To compute d_k, the following minimizing seminorm condition is imposed on the kth CMRH residual vector

$$|r_k|_{Z_k} = \min_{x \in x_0 + K_k(A,r_0)} |b - Ax|_{Z_k}, \tag{8}$$

where $|u|_{Z_k} = \sqrt{u^T Z_k u}$, $Z_k = (L_{k+1}^l)^T L_{k+1}^l$ and L_{k+1}^l is a left inverse of L_{k+1}^l. For more details on the seminorm $|.|_{Z_k}$ we refer to [20,21].

If $\overline{H}_k$ The $(k+1) \times k$ upper Hessenberg matrix given by the Hessenberg process is full rank, then the CMRH iterate x_k is given by

$$x_k = x_0 + L_k d_k, \tag{9}$$

where d_k is the solution of the least squares problem

$$\min_{d \in \mathbb{R}^{k+1}} \|\beta e_1 - \overline{H}_k d\|. \tag{10}$$

In order to solve the above least squares problem, the upper Hessenberg matrix $\overline{H}_k$ is transformed into upper triangular form by a QR-factorization with Givens rotations [21,29]. Hence, if we are dealing with the CMRH approximation from $K_k(A, r_0)$ then letting $c_i, s_i \in \mathbb{R}$ such that $c_i^2 + s_i^2 = 1$ and Ω_i be the $(k+1) \times (k+1)$ rotation matrix given by

$$\Omega_i = \begin{pmatrix} I_{i-1} & & & \\ & c_i & s_i & \\ & -s_i & c_i & \\ & & & I_{k-i} \end{pmatrix},$$

we have to multiply the Hessenberg matrix $\overline{H}_k$ and the corresponding right-hand side $\bar{g}_0 \equiv \beta e_1$ by a sequence of such matrices from the left, at each time choosing the scalars c_i, s_i in order to eliminate $h_{i+1,i}$. For example, after applying the rotations Ω_i, $i = 1, \ldots, k$, the matrix $\overline{H}_k$ and the right-hand side βe_1 are transformed into

$$\begin{bmatrix} h_{1,1}^{(k)} & h_{1,2}^{(k)} & \cdots & h_{1,k}^{(k)} \\ & h_{2,2}^{(k)} & \cdots & h_{2,4}^{(k)} \\ & & \ddots & \vdots \\ & & & h_{k,k}^{(k)} \\ & & & 0 \end{bmatrix}, \quad \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_k \\ \mu_{k+1} \end{bmatrix}.$$

The scalars c_i and s_i of the rotation Ω_i are defined by

$$s_i = \frac{h_{i+1,i}}{\sqrt{(h_{i,i}^{(i-1)})^2 + (h_{i+1,i})^2}}, \quad c_i = \frac{h_{i,i}^{(i-1)}}{\sqrt{(h_{i,i}^{(i-1)})^2 + (h_{i+1,i})^2}}. \tag{11}$$

Letting $Q_k = \Omega_k \Omega_{k-1} \ldots \Omega_1$, $\overline{R}_k = Q_k \overline{H}_k$ and $\bar{g}_k = Q_k(\beta e_1) = [\mu_1, \ldots, \mu_{k+1}]^{\mathrm{T}}$, we have

$$\min_{d \in \mathbb{R}^{k+1}} \|\beta e_1 - \overline{H}_k d\| = \min_{d \in \mathbb{R}^{k+1}} \|\bar{g}_k - \overline{R}_k d\| \quad (\beta = \|r_0\|_\infty).$$

Notice that the last row of $\overline{R}_k$ is zero, hence the solution to the above least squares problem is given by simply solving the upper triangular system resulting from ignoring the last row of $\overline{R}_k$ and right-hand side $\bar{g}_k$.

In practice, and like in the GMRES algorithm, the above procedure is implemented progressively, i.e., at each step of the CMRH method the QR factorization is performed on the new column of $\overline{H}_k$ [21,29]. This allows us to consider $|\mu_{k+1}|$ as an estimate of the residual norm $\|b - Ax_k\|$ without having to compute x_k. In fact, in [30,31], it is shown that

$$|r_k| = |b - Ax_k|_{Z_k} = |\mu_{k+1}|, \tag{12}$$

and that we can use

$$|\mu_{k+1}| \leqslant \varepsilon, \tag{13}$$

as a stopping criterion for the CMRH algorithm where ε is a choosen tolerance.

Notice that in exact arithmetic, the CMRH algorithm cannot break down and gives the solution of the system at the same iteration as GMRES. The following result is given in [30,31].

Theroem 2. *Let r_k^G, r_k^C be, respectively, the kth residual vector of the GMRES and CMRH algorithms and m be the degree of the minimal polynomial of the matrix A for the vector r_0^C. Then the iterates x_k in the CMRH method are well defined for $k = 1, \ldots, m$ and x_m is the exact solution. Moreover, if $r_0^C = r_0^G$, then*

$$\|r_k^C\| \leqslant \chi(L_{k+1}) \|r_k^G\|,$$

where $\chi(L_{k+1}) = \|L_{k+1}^+\| \|L_{k+1}\|$ and $L^+ = (L^{\mathrm{T}} L)^{-1} L^{\mathrm{T}}$ is the pseudoinverse of L.

We also recall that as in the GMRES method computational and storage constraints usually force the CMRH method to be restarted after a fixed number of iterations with subsequent loss of monotonic convergence properties. In particular, stagnation is often encountered if the size m of a restart is too small [19]. So, in order to solve dense linear systems, we propose to use, in the CMRH method, the Hessenberg process with over-storage instead of the Hessenberg process with pivoting strategy. The new method is called CMRH algorithm with over-storage and is described below. Notice that, in this algorithm, the initial residual r_0 and the approximated solution x_k are stored in b.

Algorithm 4. *CMRH method with over-storage*

```
(1) Start:
    Choose an initial guess $x_0$ and a tolerance $\varepsilon$;
    Let $p = [1, \cdots, n]^T$; compute $b = b - Ax_0$;
    Determine $i_0$ such that $|(b)_{i_0}| = \|b\|_\infty$; $\beta = (b)_{i_0}$; $b = b/\beta$;
    $p_{i_0} \longleftrightarrow p_1$; $(b)_{i_0} \longleftrightarrow (b)_1$;
    $A_{i_0,:} \longleftrightarrow A_{1,:}$; $A_{:,i_0} \longleftrightarrow A_{:,1}$;
(2) Loop:
    For $k = 1, \ldots,$ until convergence do,
      $u = A_{:,k} + A_{:,k+1:n}(b)_{k+1:n}$; $A_{k,k+1:n} = (b)_{k+1:n}$;
      for $j = 1, \ldots, k$
        $A_{j,k} = (u)_j$; $(u)_j = 0$;
        $(u)_{j+1:n} = (u)_{j+1:n} - A_{j,k}A_{j+1:n,j}$;
      end
      Determine $i_0 \in \{k+1, \ldots, n\}$ such that $|(u)_{p_{i_0}}| = \|(u)_{p_{k+1}:p_n}\|_\infty$;
      $h = (u)_{p_{i_0}}$; $v = u/h$;
      $p_{i_0} \longleftrightarrow p_{k+1}$; $(v)_{i_0} \longleftrightarrow (v)_{k+1}$;
      $A_{i_0,:} \longleftrightarrow A_{k+1,:}$; $A_{:,i_0} \longleftrightarrow A_{:,k+1}$;
      Update the QR factorization of $\overline{H}_k$, i.e.
        • Apply the rotations $\Omega_i$ to the kth column of $\overline{H}_k$,
        i.e. apply $\Omega_i$, $i = 1, \ldots, k-1$ to $[A_{1:k,k}, h]$;
        • Compute the rotation coefficients $c_k$, $s_k$ by (11),
        i.e. $s_i = \frac{h}{\sqrt{(A_{i,i})^2+h^2}}$, $c_i = \frac{A_{i,i}}{\sqrt{(A_{i,i})^2+h^2}}$;
      Apply previous rotations to $\overline{H}_k$ and $\overline{g}_k$, i.e. compute
        • $\mu_{k+1} = -s_k\mu_k$,
        • $\mu_k = c_k\mu_k$,
        • $A_{k,k} = c_kA_{k,k} + s_kh$,
      If $|\mu_{k+1}| \leqslant$ goto (3); end
    end
(3) Update:
    Solve $H_kd_k = \beta e_1$, $(H_k = triu(A_{1:k,1:k}))$;
    Update $x_k = x_0 + L_kd_k$, $(L_k = diag(ones(k,1)) + tril(A_{:,1:k}, -1))$;
    Reorder the components of $x_k$
      for $i = 1, \ldots, n$
        $(b)_{p_i} = (x_k)_i$;
      end
```

We end this section by giving the operation count for the new algorithm and comparing it with those of the GMRES method. If we neglect the cost of updating the QR factorization and computing d_k for both methods, then iteration k of Algorithm 4 involves

- $u = A_{:,k} + A_{:,k+1:n}(b)_{k+1:n}$ which requires $n(n-k)$ multiplications.
- $(u)_{j+1:n} = (u)_{j+1:n} - A_{j,k}A_{j+1:n,j}$ for $j = 1, \ldots, k$ which requires $\sum_{j=1}^{k}(n-j) = nk - k(k+1)/2$ multiplications.

Notice that the corresponding operations in the GMRES algorithm are

- $u = Av_k$ which requires n^2 multiplications.
- $u = u - H_{j,k}v_j$ for $j = 1, \ldots, k$ which requires nk multiplications.

In conclusion, for dense matrices, if Algorithm 4 converges in m steps, then it requires $mn^2 - m^3/6$ multiplications or additions, while full GMRES requires $mn^2 + m^2n$ multiplications or additions.

5. Numerical experiments

In this section, we provide experimental results of using the CMRH algorithm with over-storage to solve dense linear systems and compare its performances with the Gaussian elimination method. All the experiments were performed on a computer of Intel Pentium-4 processor at 3.4 GHz and 2048 MBytes of RAM. In all the examples, the starting guess for the CMRH method was taken to be zero and the right-hand side b is set in a such a way that the exact solution x^* is known. This allows us to compare not only $\|\mathrm{res}^G = b - Ax^G\|$ and $\|\mathrm{res}^C = b - Ax^C\|$ but also $\|\mathrm{err}^G = x^* - x^G\|$ and $\|\mathrm{err}^C = x^* - x^C\|$ which are, respectively, the norm of the residual and error vector given by the Gaussian elimination method and the CMRH method.

Experiment 1. The numerical results in this first set of experiments were obtained using Matlab7.

Example 1.1: Consider the solution of the Fredholm integral equation of the first kind

$$\int_6^6 \kappa(s,t)x(t)\,\mathrm{d}t = y(s), \quad -6 \leqslant s \leqslant 6, \tag{14}$$

discussed in [26]. Its solution, kernel and right-hand side are given by

$$x(t) = \begin{cases} 1 + \cos\left(\frac{\pi}{3}t\right) & \text{if } |t| < 3, \\ 0 & \text{otherwise,} \end{cases}$$
$$\kappa(s,t) = x(s-t),$$
$$y(s) = (6 - |s|)\left(1 + \frac{1}{2}\cos\left(\frac{\pi}{3}t\right)\right) + \frac{9}{2\pi}\sin\left(\frac{\pi}{3}|s|\right).$$

We use the code philips.m from [16] to discretize (14) by a Galerkin method with orthonormal box functions as test and trial functions to obtain the matrix $A_1 \in \mathbb{R}^{n\times n}$ and a scaled approximate solution $x^* \in \mathbb{R}^n$.

Example 1.2: We consider the matrix

$$A_2 = \begin{bmatrix} K & M \\ M^{\mathrm{T}} & 0_{3\times 3} \end{bmatrix}, \tag{15}$$

where

$$M = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_{n-3} & y_{n-3} & 1 \end{bmatrix}, \quad K_{i,j} = \begin{cases} 0 & \text{if } i = j, \\ \dfrac{-1}{8\pi} d_{i,j}^2 \log(d_{i,j}) & \text{if } i \neq j, \end{cases}$$

and $d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Notice that the matrix A comes from thin plate splines problems [6].

Example 1.3: Consider the solution of the Fredholm integral equation of the second kind

$$(I + \kappa^2 K_m)x = x^i, \tag{16}$$

where

$$K_m u = \int_\Omega g(z-y)m(y)u(y)\,\mathrm{d}y.$$

discussed in [9]. This equation is related to the integral equation formulation of the Helmholtz partial differential equation for modeling scattered waves for which the convergence is dictated by the wave number κ. Setting $\Omega = [0,1] \times [0,1]$ and discretizing the integral equation (16) by using n mesh nodes and applying a composite trapezoid rule results in the linear system

$$A_3 x = b, \tag{17}$$

Table 1
Results obtained for the matrices A_1, A_2 and A_3

A		Iter.	Res. norm	Err. norm
A_1	CMRH	70	2.14×10^{-9}	3.84×10^{-5}
$n = 8000$ $\varepsilon = 10^{-10}$	GAUSS		9.40×10^{-14}	3.66×10^{-1}
A_2	CMRH	1512	2.50×10^{-10}	1.17×10^{-4}
$n = 6000$ $\varepsilon = 10^{-13}$	GAUSS		1.35×10^{-10}	5.55×10^{-5}
A_3, $\kappa = 30$	CMRH	217	1.50×10^{-11}	5.48×10^{-11}
$n = 3600$ $\varepsilon = 10^{-12}$	GAUSS		1.68×10^{-12}	1.29×10^{-12}
A_3, $\kappa = 120$	CMRH	692	3.80×10^{-11}	1.70×10^{-11}
$n = 3600$ $\varepsilon = 10^{-12}$	GAUSS		3.81×10^{-12}	9.48×10^{-13}

Table 2
Results obtained for the matrices A_4, A_5, A_6 and A_7

A		Iter.	Time	Res. norm	Err. norm
A_4	CMRH	668	397	3.81×10^{-9}	6.46×10^{-5}
	GAUSS		3124	3.13×10^{-9}	5.04×10^{-5}
A_5	CMRH	1107	663	2.82×10^{-5}	2.84×10^{-5}
	GAUSS		3185	3.03×10^{-6}	2.13×10^{-6}
A_6	CMRH	2235	2131	2.41×10^{-6}	1.02×10^{-9}
	GAUSS		3604	1.37×10^{-6}	3.56×10^{-10}
A_7	CMRH	791	744	3.53×10^{-10}	6.76×10^{-13}
	GAUSS		3595	4.31×10^{-10}	7.79×10^{-13}

where $A_3 = (I + \kappa^2 KM) \in \mathbb{C}^{n\times n}$, $K = (K_{i,j}) = (h^2 g(z_i - z_j)$, M is the diagonal matrix defined by $M_{j,j} = m(z_j)$, $z_j \in \Omega$ and h^2 is the area of each partition. For more details on the integral equation (16) and for the function g and the right-hand side b, we refer to [9,28]. As the convergence of Krylov methods applied to (17) depends on κ [28], we consider values of κ which are $\kappa = 30$ and 120. The matrix A_3 is a dense non-Hermitian matrix. Note that the residuals norm shown in Table 1 exhibit the behavior of the CMRH method compared to the Gaussian elimination method.

The CMRH method converges in 70 iterations for the first example (matrix A_1) and gives better approximate than Gaussian elimination method. For the approximation problem (matrix A_2) CMRH methods converges in a large number of iteration. For the Helmoltz problem (matrix A_3) the number of iteration increases with κ. The CMRH method gives a good approximation of the solution of the linear system even if κ is large.

The obtained results for the above matrices are summarized in Table 2 where ε is the tolerance used in (13) to stop the iteration in the CMRH algorithm, n is the size of each matrix.

Experiment 2. In this second set of experiments, the algorithms were coded in Fortran-77 and we used a tuned BLAS library for Intel Pentium processors and the LAPACK library [1]. Notice that for the CMRH algorithm, the tolerance used in (13) as stopping test for iteration index k is $\varepsilon = \min(10^{-13}, 10^{-3}\,\mathrm{err}^G)$.

Example 2.1: Three real matrices of size $n = 15\,000$ are considered which are

$$A_4 = (a_{j,k}) = \frac{2\min(j,k) - 1}{n - j + k},$$

$$A_5 = (a_{j,k}) = \begin{cases} 0 & \text{if } j = k, \\ |j-k| + \dfrac{1}{j-k} & \text{if } j \neq k. \end{cases}$$

Example 2.2: Two complex matrices of size $n = 11\,000$ are considered which are

$$A_6 = (a_{j,k}) = \begin{cases} 1 + \dfrac{k}{10} + \imath \dfrac{j}{10} & \text{if } j > k, \\ 1 + k\imath & \text{if } j = k, \\ 1 + \imath & \text{if } j < k, \end{cases}$$

$$A_7 = (a_{j,k}) = \begin{cases} \dfrac{1}{2k-1} + \imath \dfrac{k}{10} & \text{if } j = k, \\ \dfrac{1}{j+k-1} & \text{if } j \neq k. \end{cases}$$

We see from this table that Gaussian elimination requires more CPU times as compared to CMRH method.

5.1. Conclusion

For solving linear systems of equations of large sparse matrices, Krylov subspace iterations like GMRES and QMR are among the most widely used. In our experience the convergence of GMRES and CMRH is very similar. They converge in general in almost the same number of iterations. For large dense matrices, we cannot use the full GMRES (we need to store the Arnoldi vectors and the Hessenberg matrix). We have proposed in this paper a new implementation of the CMRH method. Hence, the CMRH method is the only subspace Krylov method, which can be applied to dense linear systems, without having to store the entire matrix and all the Krylov vectors. A comparison with Gaussian elimination method is also considered.

Acknowledgment

We would like to thank A. Bouhamidi and J. Sifuentes for providing us with their MATLAB code for generating the matrices A_2 and A_3 in Experiment 1, respectively, and P. Marion for discussions and for the Fortran implementation of the CMRH method. We would also wish to thank the referee for his valuable comments and suggestions.

References

[1] E. Anderson, et al., LAPACK Users' Guide, second ed., SIAM. Philadelphia, PA, 1995.
[2] S. Amini, P.J. Harris, D.T. Wilton, Coupled boundary and finite element methods for the solution of the dynamic fluid-structure interaction problem, Springer, Berlin, 1992.
[3] W. Arnoldi, The principle of minimized iterations in the solution of the matrix eigenvalue problem, Quart. Appl. Math. 9 (1951) 17–29.
[4] A. Bendali, Numerical Analysis of the exterior boundary value problem for the time-harmonic maxwell equations by a boundary finite element method, Math. Comput. 43 (1984) 29–68.
[5] S. Bettadpur, V. Parr, B. Shutz, C. Hempel, Large least squares problems and geopotential estimation from satellite gravity gradiometry. Supercomputing '92. IEEE Computer Society Press, Los Alamitos, CA.
[6] A. Bouhamidi, A. Le Méhauté, Multivariate interpolating (m, l, s)-splines, Adv. Comput. Math. 11 (1999) 287–314.
[7] C. Brezinski, M.R. Zaglia, H. Sadok, Avoiding breakdown and near breakdown in Lanczos type algorithms, Numer. Algorithms 1 (1991) 199–206.
[8] K. Chen, On a class of preconditioning methods for dense linear systems from boundary elements, SIAM J. Sci. Comput. 20 (1998) 684–698.
[9] D. Colton, R. Kress, Inverse Acoustic and Electromagnetic Scattering Theory, Springer, Berlin, 1998.
[10] A. Edelman, The first annual large dense linear system survey, SIGNUM Newsletter 26 (1991) 6–12.
[11] A. Edelman, Large dense numerical linear algebra in 1993: the parallel computing influence, J. Supercomput. Appl. 7 (1993) 113–128.
[12] J. Ford, K. Chen, L.E. Scales, A new wavelet transform preconditioner for iterative solution of elastohydrodynamic lubrication problems, Internat. J. Comput. Math. 75 (2000) 497–513.
[13] J. Ford, Wavelet-based preconditioning of dense linear systems, Ph.D. Thesis, University of Liverpool, 2001.
[14] R.W. Freund, N.M. Nachtigal, QMR: a quasi minimal residual method for non-Hermitian linear systems, Numer. Math. 60 (1991) 315–339.
[15] R.W. Freund, N.M. Nachtigal, QMR: a implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, SIAM J. Sci. Comput. 14 (1993) 137–158.
[16] P.C. Hansen, Regularization tools: a Matlab package for analysis and solution of discrete ill-posed problems, Numer. Algorithms 6 (1994) 1–35 Software is available in Netlib at ⟨www.netlib.org⟩.
[17] S.C. Hawkins, K. Chen, New wavelet preconditioner for solving boundary integral equations over nonsmooth boundaries, Internat. J. Comput. Math. 81 (2) (2004) 353–360.

[18] K. Hessenberg, Behandlung der linearen Eigenwert-Aufgaben mit Hilfe der Hamilton-Cayleychen Gleichung, Darmstadt Dissertation, 1940.
[19] W.D. Joubert, On the convergence behavior of the restarted GMRES algorithm for solving non symmetric linear systems, J. Numer. Linear Algebra Appl. 1 (1994) 427–448.
[20] M. Heyouni, H. Sadok, On a variable smoothing procedure for Krylov subspace methods, Linear Algebra Appl. 268 (1998) 131–149.
[21] M. Heyouni, Méthode de Hessenberg Généralisée et Applications, Ph.D. Thesis, Université des Sciences et Technologies de Lille, France, 1996.
[22] A.S. Householder, F.L. Bauer, On certain methods for expanding the characteristic polynomial, Numer. Math. 1 (1959) 29–37.
[23] C. Lanczos, Solution of systems of linear equations by minimized iterations, J. Res. Nat. Bur. Stand. 49 (1952) 33–53.
[24] N.M. Nachtigal, A look-ahead variant of the Lanczos algorithm and its application to the quasi minimal residual method for non-Hermitian linear systems, Ph.D. Thesis, Massachusetts Institute of Technology, 1991.
[25] B.N. Parlett, D.R. Taylor, Z.A. Liu, A look-ahead Lanczos algorithm for unsymmetric matrices, Math. Comput. 44 (1985) 105–124.
[26] D.L. Philips, A technique for the numerical solution of certain integral equation of the first kind, J. ACM 9 (1962) 84–97.
[27] J.K. Prentice, A quantum mechanical theory for the scattering of low energy atoms from incommensurate crystal surface layers, Ph.D. Thesis, Department of physics, The University of New Mexico, 1992.
[28] J. Sifuentes, Preconditioning the integral formulation of the Helmoltz equation via deflation, Master Thesis, Rice university, Houston, April 2006.
[29] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 7 (1986) 856–869.
[30] H. Sadok, Méthodes de projections pour les systèmes linéaires et non linéaires, Habilitation thesis, University of Lille1, Lille, France, 1994.
[31] H. Sadok, CMRH: A new method for solving nonsymmetric linear systems based on the Hessenberg reduction algorithm, Numer. Algorithms 20 (1999) 303–321.
[32] D.W. Schwenk, D.G. Truhlar, Localized basis function and other computational improvements invariational nonorthogonal basis function methods for quantum mechanical scattering problems involving chemical reactions, in: R. Glowinski, A. Lichenewski (Eds.), Computing Methods in Applied Sciences and Engineering, SIAM, Philadelphia, PA, 1990, pp. 291–307.
[33] J.H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, England, 1965.