Journal of Computational and Applied Mathematics 18 (1987) 249–263
North-Holland

An iterative solution method for solving $f(A)x = b$, using Krylov subspace information obtained for the symmetric positive definite matrix A

H.A. VAN DER VORST
Faculteit Wiskunde en Informatica, TU Delft, 2600 AJ Delft, The Netherlands

Received 25 January 1986
Revised 10 April 1986

Abstract: The conjugate gradients method generates successive approximations x_i for the solution of the linear system $Ax = b$, where A is symmetric positive definite and usually sparse. It will be shown how intermediate information obtained by the conjugate gradients (cg) algorithm (or by the closely related Lanczos algorithm) can be used to solve $f(A)x = b$ iteratively in an efficient way, for suitable functions f. The special case $f(A) = A^2$ is discussed in particular. We also consider the problem of solving $Ax = b$ for different right-hand sides b. A variant on a well-known algorithm for that problem is proposed, which does not seem to suffer from the usual loss of orthogonality in the standard cg and Lanczos algorithms.

Keywords: Conjugate gradients method, Lanczos method, matrix equations, sparse matrices, Krylov subspace.

1. Introduction

Throughout this paper we will assume that the matrix A is symmetric and positive definite. Then the conjugate gradients (cg) method and the Lanczos method are equivalent methods that solve iteratively the linear system $Ax = b$. When A is symmetric indefinite the Lanczos method can still be used in order to solve $Ax = b$, see, e.g. [7].

These methods generate an orthogonal basis for the Krylov subspace $K^i(A;\ r_0) \equiv \mathrm{Span}\{r_0,\ Ar_0, \ldots, A^{i-1}r_0\}$, where $r_0 = b - Ax_0$. The ith iterand x_i is the element from $K^i(A;\ r_0)$ that minimizes $(y - x,\ A(y - x))$ over all $y \in K^i(A;\ r_0)$, and the iterated vectors $r_j = b - Ax_j$ constitute an orthogonal basis for the Krylov subspace (assuming exact arithmetic). For details see, e.g. [1,6]. We will give the cg formulation of this algorithm in Section 2.

It is well-known that eigenvectors which have a significant component in the direction of b are increasingly well approximated by vectors in the Krylov subspace for increasing i, see, e.g. [3,6,10]. This leads one to expect that for a different right-hand side, which is also rich in these eigenvectors directions, a satisfactory approximated solution can be obtained by solving the projected equation onto $K^i(A;\ r_0)$. Relevant situations where this may be of help include

discretised time-dependent p.d.e's, domain decomposition methods, iterative solution methods for 3D problems in which 2D subproblems are solved accurately in each step and in the solution of non-linear problems. Still assuming exact arithmetic, Saad [11] has given a quantitative analysis for the error in the approximated solution for a different right-hand side. Parlett [7] has taken into account the problems that arise when computing in finite precision. It is then well-known that the orthogonality among the iterated basisvectors r_j is going to be lost gradually for increasing i [4,5,6,8]. Parlett [7] suggests to improve the Lanczos algorithm by an orthogonalization technique. In [7] Parlett also proposes an algorithm for improving the accuracy of the approximated solutions, obtained for subsequent right-hand sides, by continuing the Lanczos algorithm in such a way that the new iterated Lanczos vectors are also orthogonal to the old subspace $K^i(A;\ r_0)$.

In Section 3 we give the standard algorithm for the computation of the approximated solution for a different right-hand side and we propose a slight modification which seems to make the algorithm insensitive to the loss of orthogonality among the basisvectors r_j. Our formulation of the algorithm is particularly useful when the new right-hand side does not depend on the solution of the first one, since then the storage of the basis vectors can be avoided.

In Section 4 we consider the problem of solving $A^2x = b$. This can be done in two steps via $Ay = b$ and $Ax = y$. Then if y_i is the ith cg iterand approximating y, we have by construction that $y_i \in K^i(A;\ r_0)$. Though it is obvious that in this case the second right-hand side depends on the solution of the first system it can be shown that excellent approximations $\hat{x}_i$ can be computed with only little additional work, without storing the basisvectors. It is also not necessary to compute the vectors y_i explicitly.

In Section 5 we are led quite naturally to a generalization, namely the approximate solution of $f(A)x = b$, for suitable functions f. The proposed algorithm requires the solution of an eigensystem of low dimension as compared with the dimension of A.

Numerical examples that provide evidence, supporting the proposed ideas, are presented in Section 6.

2. The conjugate gradients method for $Ax = b$

Among the many computational schemes for cg we have, quite arbitrarily, chosen for the following variant [1,2]:

x_0 is a start-vector, for convenience we will use $x_0 = 0$ (which is not essential since an arbitrary $x_0 \neq 0$ can be translated by $z \equiv x - x_0$ to the problem $Az = b - Ax_0 = \tilde{b}$ for which $z_0 = 0$); $r_0 = b - Ax_0$ and $p_0 = r_0$.

For $j = 0, 1, \ldots$ until some stopping criterion is fulfilled:

$$\alpha_j = (r_j,\ r_j)/(p_j,\ Ap_j), \tag{2.1}$$

$$x_{j+1} = x_j + \alpha_j p_j, \tag{2.2}$$

$$r_{j+1} = r_j - \alpha_j A p_j, \tag{2.3}$$

$$\beta_j = (r_{j+1},\ r_{j+1})/(r_j,\ r_j), \tag{2.4}$$

$$p_{j+1} = r_{j+1} + \beta_j p_j. \tag{2.5}$$

The above scheme is not essential for the proposed new algorithms. In fact any Lanczos type scheme might be used. We have not considered the question whether a specific scheme would be preferable for some reason, e.g. stability. Possibly a stable Lanczos scheme, as proposed by Paige [3,4], with some kind of reorthogonalisation [8] leads to better results.

It is well-known that in exact arithmetic the residual vectors r_j form an orthogonal basis $\{r_0, \ldots, r_i\}$ for the Krylov space $K^{i+1}(A; r_0) = \mathrm{Span}(r_0, Ar_0, \ldots, A^i r_0\}$. The projected matrix A has, with respect to the basis $\{r_0, \ldots, r_i\}$, tridiagonal form. Paige [5] has shown that in finite precision floating point arithmetic the stable Lanczos scheme leads to:

$$AR_i = R_i T_i - t_{i+1,i} r_{i+1} e_{i+1}^{\mathrm{T}} + E_i, \tag{2.6}$$

in which: $A \in \mathbb{R}^{n \times n}$, $R_i \in \mathbb{R}^{n \times (i+1)}$ is the matrix with columns $r_0, \ldots, r_i$, $T_i \in \mathbb{R}^{(i+1)\times(i+1)}$ is tridiagonal, $t_{i+1,i}$ is the last subdiagonal element of T_{i+1}, e_{i+1} is the $(i+1)$st unit vector in $\mathbb{R}^{i+1}$ and $E_i \in \mathbb{R}^{n \times n}$ is the erromatrix with $\| E_i \|_2 \leqslant ug$. The constant u is the relative working precision and g is a very moderate factor. For details, see [5]. In exact arithmetic we have that $E_i = 0$. In the derivation of our formulas we will always assume exact arithmetic. For the given cg scheme (2.1)–(2.5), the tridiagonal matrix T_i, after i steps of the algorithm, can be written as:

$$T_i = \begin{pmatrix} \frac{1}{\alpha_0} & -\frac{\beta_0}{\alpha_0} & & & \emptyset \\ -\frac{1}{\alpha_0} & \frac{1}{\alpha_1} + \frac{\beta_0}{\alpha_0} & -\frac{\beta_1}{\alpha_1} & & \\ & -\frac{1}{\alpha_1} & \ddots & & \\ & & \ddots & \ddots & -\frac{\beta_{i-1}}{\alpha_{i-1}} \\ \emptyset & & & -\frac{1}{\alpha_{i-1}} & \frac{1}{\alpha_i} + \frac{\beta_{i-1}}{\alpha_{i-1}} \end{pmatrix}. \tag{2.7}$$

The cg approximation x_{i+1} is the solution of the projected equation onto $K^{i+1}(A; r_0)$:

$$R_i^{\mathrm{T}} A x_{i+1} = R_i^{\mathrm{T}} b. \tag{2.8}$$

Since $x_{i+1} \in K^{i+1}(A; r_0)$, it follows that $x_{i+1} = R_i y_{i+1}$, for $y_{i+1} \in \mathbb{R}^{i+1}$, hence

$$R_i^{\mathrm{T}} A R_i y_{i+1} = R_i^{\mathrm{T}} R_i T_i y_{i+1} = R_i^{\mathrm{T}} b \tag{2.9}$$

or

$$T_i y_{i+1} = \left(R_i^{\mathrm{T}} R_i\right)^{-1} R_i^{\mathrm{T}} b \tag{2.10}$$

and

$$x_{i+1} = R_i T_i^{-1} \left(R_i^{\mathrm{T}} R_i\right)^{-1} R_i^{\mathrm{T}} b. \tag{2.11}$$

We have $r_0 = b$ since $x_0 = 0$ has been assumed.Hence x_{i+1} can be expressed as

$$x_{i+1} = R_i T_i^{-1} e_1, \tag{2.12}$$

where e_1 is the first unit vector in $\mathbb{R}^{i+1}$. Note that if we use a Lanczos scheme with $\| r_j \|_2 = 1$, $j = 0, 1, 2, \ldots, i$, then (2.12) should be replaced by

$$x_{i+1} = \| b \|_2 R_i T_i^{-1} e_1. \tag{2.13}$$

3. An approximate solution for $A\tilde{x} = \tilde{b}$

Once $Ax = b$ has been solved approximately by the cg method or Lanczos method, the generated basis for $K^{i+1}(A; b)$ and T_i can be used to obtain a cheap approximation for $A^{-1}\tilde{b}$, for $\tilde{b} \neq b$. To this end we project the equation $A\tilde{x} = \tilde{b}$ onto the Krylov space $K^{i+1}(A; b)$ and solve the projected equation. Analogously to the derivation of (2.11) it follows that the solution $\hat{x}_{i+1}$ of the projected equation can be written as

$$\hat{x}_{i+1} = R_i T_i^{-1} \left(R_i^{\mathrm{T}} R_i\right)^{-1} R_i^{\mathrm{T}} \tilde{b}. \tag{3.1}$$

We expect $\hat{x}_{i+1}$ to be close to $\tilde{x}$ when $\tilde{b}$ is close to $K^{i+1}(A; b)$, for i so large that x_{i+1} is close to x. For a quantitative analysis see [11]. Note that we use the superscript $\hat{}$ in order to stress the fact that $\hat{x}_{i+1}$ differs from the $\tilde{x}_{i+1}$ that one would obtain when using the cg scheme for $A\tilde{x} = \tilde{b}$.

Let the vector b be defined by

$$\hat{b} = \left(\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_i\right)^{\mathrm{T}}, \tag{3.2}$$

with

$$\hat{b}_j = \left(r_j, \tilde{b}\right)/(r_j, r_j), \qquad j = 0, 1, \ldots, i. \tag{3.3}$$

Equation (3.1) can be rewritten in the form:

$$\hat{x}_{i+1} = R_i T_i^{-1} \hat{b}. \tag{3.4}$$

It appears that the computation of the $\hat{b}_j$ from (3.3) leads to numerically very unstable results for $\hat{x}_{i+1}$, since in practice the set of basis vectors r_j may be far from orthogonal for increasing i. Parlett [7] suggests a selective orthogonalization technique for the vectors r_j in order to overcome this severe problem. We propose a slight modification to the formula (3.3) for the computation of the $\hat{b}_j$.

In exact computation $(R_i^{\mathrm{T}} R_i)^{-1} R_i^{\mathrm{T}} \tilde{b}$ represents the projection of $\tilde{b}$ onto $K^i(A; b)$, with respect to the basisvectors r_j. Therefore we want to have $P\tilde{b}$, i.e., the projection of $\tilde{b}$, in the form

$$P\tilde{b} = R_i \hat{b} = \hat{b}_0 r_0 + \hat{b}_1 r_1 + \cdots + \hat{b}_i r_i.$$

The idea is now to determine, for each newly generated lanczos vector r_j, the component of the part of $\tilde{b}$ which has not yet been spanned by the previous vectors $r_0, r_1, \ldots, r_{j-1}$. This leads to the following algorithm for the computation of the components of $\hat{b}$:

$$\begin{aligned} &\bar{b}_0 \equiv \tilde{b}, \\ &\hat{b}_j = \left(r_j, \bar{b}_j\right)/(r_j, r_j), \quad \bar{b}_{j+1} = \bar{b}_j - \hat{b}_j r_j, \qquad j = 0, \ldots, i. \end{aligned} \tag{3.3a}$$

Algorithm (3.3a) is known as the Modified Gram–Schmidt Algorithm (MGSA), for orthogonalizing $\tilde{b}$ against the set $\{r_j\}$. Ruhe [9] discusses this algorithm for the situation in which the r_j are "less than perfectly orthogonal", which is typically our case (it may even occur that the r_j are linearly dependent).

Now it is obvious that a component of $\tilde{b}$ in a direction of a previous vector r_l does not reenter the process, whether the new r_j is orthogonal to r_l or not. From numerical experiments it appears that the computation of $\hat{x}_{i+1}$ by (3.3a) and (3.4) is very stable, also in situations where a severe loss of orthogonality among the Lanczos vectors is observed. See Section 6 for an example.

At first glance, (3.4) suggests that it is always necessary to store the vectors r_j, which build R_i, in order to compute $\hat{x}_{i+1}$. We will derive a version of the algorithm for the computation of $\hat{x}_{i+1}$ for which the storage of the complete set of basisvectors is not required when $\tilde{b}$ and b are both known at the same time (i.e., $\tilde{b}$ does not depend on x).

The matrix T (we will drop the index i when it is obvious) can be written in factored form as:

$$T = LDU \tag{3.5}$$

with

$$L = \begin{pmatrix} 1 & & & & \emptyset \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ \emptyset & & & -1 & 1 \end{pmatrix}, \quad D = \begin{pmatrix} \frac{1}{\alpha_0} & & & \emptyset \\ & \frac{1}{\alpha_1} & & \\ & & \ddots & \\ \emptyset & & & \frac{1}{\alpha_i} \end{pmatrix},$$

$$U = \begin{pmatrix} 1 & -\beta_0 & & & \emptyset \\ & 1 & -\beta_1 & & \\ & & 1 & \ddots & \\ & & & \ddots & -\beta_{i-1} \\ \emptyset & & & & 1 \end{pmatrix}.$$

Hence $T^{-1} = U^{-1}D^{-1}L^{-1}$, with

$$U^{-1} = \begin{pmatrix} 1 & \beta_0 & \beta_0\beta_1 & \beta_0\beta_1\beta_2 & \cdots \\ & 1 & \beta_1 & \beta_1\beta_2 & \cdots \\ & & 1 & \beta_2 & \cdots \\ & & & 1 & \cdots \\ \emptyset & & & & \ddots \end{pmatrix}, \quad L^{-1} = \begin{pmatrix} 1 & & & & \emptyset \\ 1 & 1 & & & \\ 1 & 1 & 1 & & \\ \vdots & & & \ddots & \\ 1 & 1 & & & 1 \end{pmatrix}. \tag{3.6}$$

For the jth component of the vector $D^{-1}L^{-1}\hat{b} \in \mathbb{R}^{i+1}$, it follows that

$$(D^{-1}L^{-1}\hat{b})_j = \alpha_j \sum_{k=0}^{j} \hat{b}_k, \quad j = 0, \ldots, i. \tag{3.7}$$

The vector $\hat{x}_{i+1}$, given by (3.4), can be rewritten as

$$\hat{x}_{i+1} = (RU^{-1})(D^{-1}L^{-1}\hat{b}). \tag{3.8}$$

From straight-forward computation it follows that the jth column of (RU^{-1}), denoted by $(RU^{-1})_{.j}$, can be computed recursively from the previous column:

$$(RU^{-1})_{.j} = \beta_{j-1}(RU^{-1})_{.j-1} + r_j, \quad j = 1, \ldots, i \tag{3.9}$$

with

$$(RU^{-1})_{.0} = r_0.$$

The combination of (3.7) and (3.9) in formula (3.8) shows that the vectors $\hat{x}_{i+1}$ can be computed recursively and that, when $\tilde{b}$ is known before $Ax = b$ has been solved, it is not necessary to store all the r_j-vectors:

$$\hat{x}_{i+1} = \sum_{j=0}^{i} (D^{-1}L^{-1}\hat{b})_j (RU^{-1})_{.j}. \tag{3.10}$$

In the practical situation that x_{i+1} is already sufficiently accurate for some value of i, but $\hat{x}_{i+1}$ is not accurate enough, we suggest to use $\hat{x}_{i+1}$ as a starting vector for the cg scheme applied to $A\tilde{x} = \tilde{b}$, or to continue the Lanczos algorithm for $A\tilde{x} = \tilde{b}$, with starting vector $\hat{x}_{i+1}$, in a way as described by Parlett [7].

4. An efficient cg-like algorithm for the iterative solution of $A^2x = b$

4.1. Introduction

A straight-forward approach for solving $A^2x = b$ iteratively by the cg method is to apply the scheme (2.1)–(2.5), with A therein replaced by A^2. This implies that two matrix vector multiplications have to be carried out for each cg iteration step, since in general one may want to avoid the explicit computation of the matrix A^2. The cg iterands of this process will be denoted by x_{i+1}, $i = 0, 1, 2, \ldots, x_0 = 0$.

Another approach is to solve $A^2x = b$ in two steps. First y is solved from $Ay = b$ and then x is solved from $Ax = y$. If $Ay = b$ is solved by the cg method, leading to iterands y_{i+1} ($y_0 = 0$), then in view of Section 3 one might hope that $Ax = y_{i+1}$ can be solved efficiently so that it leads to a sufficient accurate approximation $\hat{x}_{i+1}$ for the solution of $A^2x = b$. We will first present an algorithm for the computation of $\hat{x}_{i+1}$ in section 4.2. Accuracy aspects will be considered in section 4.3.

4.2. Derivation of the computational scheme

In this subsection we will present a computational scheme which computes $\hat{x}_{i+1}$, $i = 0, 1, 2, \ldots$, simultaneously with the process for y_{i+1}, without storing the r_j's (and even without computing y_{i+1} explicitly). For $y_0 = 0$ we can, similarly to (2.12), write y_{i+1} in the form

$$y_{i+1} = RT^{-1}e_1 \quad \text{(indices for } R \text{ and } T \text{ have been dropped)}. \tag{4.1}$$

The vector $\hat{x}_{i+1}$ is now defined by

$$\hat{x}_{i+1} \equiv RT^{-1}(R^{\mathrm{T}}R)^{-1}R^{\mathrm{T}}y_{i+1} \tag{4.2}$$

As has been shown in Section 3, $\hat{x}_{i+1}$ is obviously the solution of the projected equation $Ax = y_{i+1}$ onto $K^i(A; b)$.

Inserting (4.1) in (4.2) leads to

$$\hat{x}_{i+1} = RT^{-1}(R^{\mathrm{T}}R)^{-1}R^{\mathrm{T}}RT^{-1}e_1 = RT^{-2}e_1. \tag{4.3}$$

From (3.5) it follows that

$$\hat{x}_{i+1} = RU^{-1}D^{-1}(UL)^{-1}D^{-1}L^{-1}e_1. \tag{4.4}$$

The trick is now to rewrite UL in the form $\tilde{L}\tilde{U}$ and it will be shown that $\tilde{L}$ and $\tilde{U}$ can be constructed recursively while carrying out the iteration process for $Ay = b$.

From straight-forward calculations it follows that

$$UL = \begin{pmatrix} 1+\beta_0 & -\beta_0 & & \emptyset \\ -1 & 1+\beta_1 & -\beta_1 & \\ & -1 & \ddots & \ddots \\ \emptyset & & \ddots & \end{pmatrix} = \tilde{L}\tilde{U}, \tag{4.5}$$

with

$$\tilde{L} = \begin{pmatrix} a_0 & & & \emptyset \\ -1 & a_1 & & \\ & -1 & \ddots & \\ & & \ddots & \\ \emptyset & & -1 & a_i \end{pmatrix}, \qquad \tilde{U} = \begin{pmatrix} 1 & b_0 & & & \emptyset \\ & 1 & b_1 & & \\ & & \ddots & \ddots & \\ & & & & b_{i-1} \\ \emptyset & & & & 1 \end{pmatrix}$$

and

$$\left.\begin{aligned} &a_0 = 1+\beta_0, \qquad b_0 = -\beta_0/a_0, \\ &a_j = 1+\beta_j - \beta_{j-1}/a_{j-1}, \quad j = 1, 2, \ldots, i, \\ &b_j = -\beta_j/a_j, \qquad\qquad\quad j = 1, 2, \ldots, i-1. \end{aligned}\right\} \tag{4.6}$$

Equation (4.4) can be rewritten as

$$\hat{x}_{i+1} = (RU^{-1}D^{-1}\tilde{U}^{-1})(\tilde{L}^{-1}D^{-1}L^{-1}e_1). \tag{4.7}$$

From (3.5) and (3.6) we have that

$$D^{-1}L^{-1}e_1 = (\alpha_0, \ldots, \alpha_i)^{\mathrm{T}} \tag{4.8}$$

With (4.6) it follows that the second part in parentheses in (4.7) can be written as

$$\tilde{L}^{-1}D^{-1}L^{-1}e_1 = (\theta_0, \ldots, \theta_i)^{\mathrm{T}}, \tag{4.9}$$

with

$$\theta_0 = \alpha_0/a_0, \qquad \theta_j = (\alpha_j + \theta_{j-1})/a_j, \quad j = 1, 2, \ldots, i.$$

As we have seen in Section 3, relation (3.9), the columns of RU^{-1} can be computed recursively from each other. Denoting the jth column of $RU^{-1}D^{-1}$ by $\tilde{r}_j$, it follows that

$$\tilde{r}_j = \alpha_j (RU^{-1})_{.j}. \tag{4.10}$$

Finally, we have for the jth column of $RU^{-1}D^{-1}\tilde{U}^{-1}$, denoted by $\hat{r}_j$, that

$$\left.\begin{aligned} &\hat{r}_0 = \tilde{r}_0, \\ &\hat{r}_j = (\beta_{j-1}/a_{j-1})\hat{r}_{j-1} + \tilde{r}_j, \quad j = 1, 2, \ldots, i. \end{aligned}\right\} \tag{4.11}$$

Of course the vectors $\tilde{r}_j$ need not be formed explicitly, and the $\hat{r}_j$ can be computed directly from the vectors $(RU^{-1})_{.j}$. The vector $\hat{x}_{i+1}$ can now be computed recursively from

$$\left.\begin{aligned} &\hat{x}_0 = 0, \\ &\hat{x}_j = \hat{x}_{j-1} + \theta_{j-1}\hat{r}_{j-1}, \quad j = 1, 2, \ldots, i+1. \end{aligned}\right\} \tag{4.12}$$

It is easily verified that it is not necessary to store all r_j's and that by (4.12) we obtain an approximation $\hat{x}_{i+1}$ to the solution x of $A^2x = b$, by only about 4 additional flops per iterationstep per unknown extra, as compared with the computational work that is required for $i+1$ iterationsteps of cg for the equation $Ay = b$. Note also that the vectors y_j need not be computed.

4.3. Convergence analysis

It is obvious that the cg iterand y_{i+1} is an element of $K^{i+1}(A;\ b)$. Saad [11] has pointed out that the accuracy of the solution $\hat{x}_{i+1}$, of the projected equation $Ax = y_{i+1}$ onto $K^{i+1}(A;\ b)$, has the same order of accuracy as the iterand $\tilde{x}_{i+1}$, obtained by the cg method for $Ax = y_{i+1}$ with $\tilde{x}_0 = 0$.

We will show that the residuals $\| A\hat{x}_{i+1} - y_{i+1} \|$ and $\| A\tilde{x}_{i+1} - y_{i+1} \|$ can be bounded by approximately the same sharp upperbound. By $\| x \|$ we denote the standard euclidean norm. Also upperbounds for $\| A^2x_{i+1} - b \|$ and $\| A^2\hat{x}_{i+1} - b \|$ are presented. These upperbounds indicate that in many relevant situations it is much cheaper to compute $\hat{x}_{i+1}$ than to compute x_{i+1}.

Along the lines of the proof of Saad's proposition 1 in [11] we find that

$$\| A\hat{x}_{i+1} - y_{i+1} \|_{A^{-1}}^2 \leqslant \alpha^2 4\rho_A^{2i+2} \| b \|_{A^{-1}}^2, \tag{4.13}$$

with

$$\rho_A = \frac{\sqrt{c} - 1}{\sqrt{c} + 1}, \quad c \text{ is the conditionnumber of } A,\ c = \frac{\lambda_{\max}}{\lambda_{\min}}, \text{ where } \lambda_{\min}$$

and $\lambda_{\max}$ are the smallest and largest eigenvalue respectively for which the corresponding eigenvectors has a component in b. For α we obtain

$$\alpha = \frac{(y_{i+1},\ b)}{(b,\ b)} = \frac{(r_0,\ RT^{-1}e_1)}{(r_0,\ r_0)} = \alpha_0 = \frac{(r_0,\ r_0)}{(r_0,\ Ar_0)} \leqslant \frac{1}{\lambda_{\min}}.$$

Hence

$$\| A\hat{x}_{i+1} - y_{i+1} \|^2 \leqslant 4\frac{\lambda_{\max}}{\lambda_{\min}^3}\rho_A^{2i+2} \| b \|^2. \tag{4.14}$$

For the residual for $\tilde{x}_{i+1}$, obtained with the regular cg process for $Ax = y_{i+1}$, with starting value $\tilde{x}_0 = 0$, it follows that

$$\begin{aligned} \| A\tilde{x}_{i+1} - y_{i+1} \|^2 &\leqslant \lambda_{\max} \| A\tilde{x}_{i+1} - y_{i+1} \|_{A^{-1}}^2 \\ &\leqslant \lambda_{\max} 4\rho_A^{2i+2} \| y_{i+1} \|_{A^{-1}}^2 \lesssim 4\frac{\lambda_{\max}}{\lambda_{\min}^3}\rho_A^{2i+2} \| b \|^2. \end{aligned} \tag{4.15}$$

(The latter inequality follows, provided i so large that $\| y_{i+1} \| \approx \| A^{-1}b \|$) Hence the residuals for $\tilde{x}_{i+1}$ and $\hat{x}_{i+1}$ have approximately the same upperbound. These upperbounds are rather sharp as we will argue now. A factor $\lambda_{\min}^2$ in the denominator is realistic when the eigenvectors corresponding to the smallest eigenvalues dominate in b, and the factor $\lambda_{\max}$ is realistic when these eigenvector components have been eliminated from the residual by the cg-process. The

latter is usually the case for i not too small (see, e.g., [12]). The factor $4\rho_A^{2i+2}$ is a well-known reduction factor for the cg-residual (see, e.g. [1]). This factor is realistic when the extremal eigenvalues of A are not very well separated (for an analysis of the cg-convergence see [12]).

Finally we want to consider the question whether it might be preferable to apply the cg-algorithm directly to $A^2x = b$. Let x_{i+1} denote the $(i+1)$st iterand for this process (with $x_0 = 0$), then it follows that

$$\begin{aligned}\| A^2x_{i+1} - b \|^2 &\leqslant \lambda_{\max}^2 \| A^2x_{i+1} - b \|_{A^{-2}}^2 \\ &\leqslant 4\lambda_{\max}^2 \rho_{A^2}^{2i+2} \| b \|_{A^{-2}}^2 \leqslant 4c^2\rho_{A^2}^{2i+2} \| b \|^2, \end{aligned} \tag{4.16}$$

with

$$\rho_{A^2} = (c-1)/(c+1), \quad c \text{ defined as in (4.13).}$$

For $\hat{x}_{i+1}$ we have that

$$A^2\hat{x}_{i+1} - b = A(A\hat{x}_{i+1} - y_{i+1}) + Ay_{i+1} - b, \tag{4.17}$$

and hence

$$\| A^2\hat{x}_{i+1} - b \|^2 \leqslant \| A \|^2 \| A\hat{x}_{i+1} - y_{i+1} \|^2 + \| Ay_{i+1} - b \|^2. \tag{4.18}$$

The second term on the right-hand side of (4.18) can be bounded by

$$\| Ay_{i+1} - b \|^2 \leqslant \lambda_{\max} \| Ay_{i+1} - b \|_{A^{-1}}^2 \leqslant 4c\rho_A^{2i+2} \| b \|^2. \tag{4.19}$$

From (4.14), (4.19) and (4.18) it then follows that

$$\| A^2\hat{x}_{i+1} - b \|^2 \leqslant 4\rho_A^{2i+2} \| b \|^2 (c^3 + c) \approx 4c^3\rho_A^{2i+2} \| b \|^2. \tag{4.20}$$

Since each iterationstep for x_{i+1} is approximately twice as expensive as for $\hat{x}_{i+1}$, it follows that $\hat{x}_{2i+2}$ is as expensive to compute as x_{i+1}. From (4.16) and (4.20) it follows that the upperbound for the residual of $\hat{x}_{2i+2}$ is smaller than the upperbound for the residual of x_{i+1}, if

$$c^3\rho_A^{4i+4} < c^2\rho_{A^2}^{2i+2}. \tag{4.21}$$

In practical situations c will be large enough as to allow for some simplifications:

$$1 > \frac{c^3 + c}{c^2} \frac{\rho_A^{4i+4}}{\rho_{A^2}^{2i+2}} \approx c \frac{(1 - 2/\sqrt{c})^{4i+4}}{(1 - 2/c)^{2i+2}} \approx c\left(1 - \frac{2}{\sqrt{c}}\right)^{4i+4} \tag{4.22}$$

which leads to the following condition

$$i \gtrsim \tfrac{1}{8}\sqrt{c} \ln c.$$

This indicates that for i not too small it may be much more efficient to determine $\hat{x}_{2i+2}$ as it is to compute an x_j, which has comparable accuracy as $\hat{x}_{2i+2}$, for practical values of c.

For numerical evidence that the computation of $\hat{x}_i$ has to be preferred over the computation of x_i, from an efficiency standpoint, see Example 4 in Section 6.

5. The solution of more general systems $f(A)x = b$

Analogously to $A^2x = b$, the linear system $A^mx = b$ can be solved using the iteration results from $Ax = b$. The successive approximations x_i for the solution x of $A^mx = b$ can then be

computed each at the cost of *only one matrix vector product with the matrix A*, instead of A^m, which implies a considerable saving in general per iteration step. Of course, for increasing values of m, the recurrence relations for the x_i will be gradually more complicated and more expensive too. In this section we suggest an alternative algorithm that avoids the construction of these recurrence relations.

We will now assume that an orthonormal basis for the Krylov subspace $K^{i+1}(A;\ b)$ is obtained (e.g., by a Lanczos type method). It should be noted that in practice the iterated basis vectors r_i may be far from orthogonal for increasing i due to rounding errors. The algorithm that we will describe does not.seem to suffer seriously from this loss of orthogonality, except that it possibly needs a few iteration steps more than would have been necessary in exact computation. The orthonormality implies that T_i is symmetric.

It is well-known, e.g., see [6], that the Krylov subspace is invariant with respect to translations, i.e. $K^{i+1}(A;\ b) = K^{i+1}(A + \alpha I;\ b)$, for α a complex scalar. In fact, let R_i, T_i, $t_{i+1,i}$, r_{i+1} be quantities as in (2.6), *obtained for* $Ax = b$, with $x_0 = 0$, then for $A + \alpha I$ it follows that

$$(A + \alpha I)R_i = R_i(T_i + \alpha I) - t_{i+1,i}r_{i+1}e_{i+1}^{\mathrm{T}}. \tag{5.1}$$

Provided $T_i + \alpha I$ is non-singular, we have, similarly as in Section 2, for $\tilde{x}_{i+1}$, which is the projection of the solution of $(A + \alpha I)\tilde{x} = b$ onto $K^{i+1}(A + \alpha I;\ b)$ that

$$\tilde{x}_{i+1} = \| b \|_2 R_i(T_i + \alpha I)^{-1}e_1. \tag{5.2}$$

For the case when $T_i + \alpha I$ is singular, or near singular, one has to replace an expression like $(T_i + \alpha I)^{-1}e_1$ by a suitable solution of $(T_i + \alpha I)z = e_1$, which leads to obvious changes in our algorithms.

From Section 4 it is now obvious that an approximation x_{i+1} to the solution x of $(A + \alpha_1 I)(A + \alpha_2 I)x = b$ can be obtained by projecting the equation onto $K^{i+1}(A;\ b)$, which gives

$$\hat{x}_{i+1} = \| b \|_2 R_i(T_i + \alpha_2 I)^{-1}(T_i + \alpha_1 I)^{-1}e_1, \tag{5.3}$$

provided the factors are non-singular.

This approach can be generalized to kth degree matrix polynomials $P_k(A)$ in A. An approximate solution x_{i+1} for $P_k(A)x = b$, with respect to $K^{i+1}(A;\ b)$, can be written as

$$\hat{x}_{i+1} = \| b \|_2 R_i(P_k(T_i))^{-1}e_1, \tag{5.4}$$

provided $P_k(A)$ and $P_k(T_i)$ are non-singular.

For such more general polynomials it will be very unattractive to factor explicitly a given matrix polynomial (e.g., for stability reasons), and also the derivation and computation of the subsequent recurrence relations for the projected factors will be increasingly complex. This can, however, be avoided by forming $P_k(T_i)$ explicitly and to compute $(P_k(T_i))^{-1}$ via, e.g., Choleski factorization.

We will now show how it can be avoided to compute the matrix $P_k(T_i)$ explicitly. Since T_i is symmetric, it can be written as $T_i = Q_i D_i Q_i^{\mathrm{T}}$, where D_i is a diagonal matrix and Q_i is orthogonal. Hence it follows for $P_k(T_i)$ that

$$P_k(T_i) = Q_i P_k(D_i)Q_i^{\mathrm{T}}, \tag{5.5}$$

and

$$(P_k(T_i))^{-1} = Q_i(P_k(D_i))^{-1}Q_i^{\mathrm{T}}. \tag{5.6}$$

Note that, at the cost of the reduction of T_i (and *not* of $P_k(T_i)$) to diagonal form, we have avoided the factorization of P_k, the construction of the recurrence relations for x_{i+1} and the explicit construction of $P_k(T_i)$. Since i will be in general small as compared with the order of A, the reduction of T_i to diagonal form is considered to be a small problem with regard to, e.g., computer time and memory space.

In combination with (5.4), (5.6) leads to

$$\hat{x}_{i+1} = \| b \|_2 R_i Q_i (P_k(D_i))^{-1} Q_i^T e_1. \tag{5.7}$$

It seems wise not to evaluate $\hat{x}_j$ for each value of j, but to do so after a couple of iteration steps (i.e., after a couple of the r_i's have been generated). Strategies for selecting the appropriate value(s) for j have not been investigated yet.

We have used the above described approach also for general functions $f(A)$, which are not finite degree polynomials in A. In that case (5.7) changes in

$$\hat{x}_{i+1} = \| b \|_2 R_i Q_i (f(D_i))^{-1} Q_i^T e_1, \tag{5.8}$$

which serves as an approximation to the solution of $f(A)x = b$. In our experiments this led to nice results when $f(A)$ 'behaved' like a low degree polynomial. We believe that this approach might turn out to be useful in connection with, e.g., higher-order single or multistep implicit schemes for certain non-stationary p.d.e.'s.

Note again, that the approach of generating a basis for the Krylov subspace $K^{i+1}(A; b)$, represented by R_i, and the projection of A onto $K^{i+1}(A; b)$, given by T_i, has the big advantage that, when solving $f(A)x = b$, one does *not* need to form $f(A)$, nor matrix vector products like $f(A)p_j$. These quantities are in general much more expensive to compute than Ap_j.

In Section 6 we give numerical evidence for the nice convergence behaviour of the x_i's for $e^A x = b$. In the specific example A has been chosen, for reasons of simplicity and so that the experiments could be easily checked on small computers, as a diagonal matrix. Hence, for this special A, $f(A)x = b$ could have been solved, much more efficiently, directly. However, the displayed convergence behaviour, observed when using our algorithm, does not depend on the special structure of A and thus we get some feeling for what we might expect for matrices with a similar spectrum but with a more general non-zero structure. This example suggests that an approach via (5.8) deserves serious consideration for the computation of $y(t) = \exp(-At)y_0$, e.g. arising in initial value problems (provided A is symmetric).

6. Numerical examples

General introduction to the examples

In this section we present a few examples that serve as illustrations to the ideas presented in previous sections. All the computations have been carried out in 48-bit floating point precision (14 ~ 15 decimal places) on a CRAY X-MP 24 computer.

Both the cg and the Lanczos algorithm do not require the matrix A in explicit form as an array containing the elements of A. They only require a rule (subroutine) which produces the vector $y = Ax$ for any given input vector x and hence they do not take any advantage from a special form of A. Therefore it is quite usual to select matrices in diagonal form for numerical experiments, since then the complete eigensystem of A is known (as well as the solution, so that an observed convergence behaviour can be better understood (e.g., see [12]).

Of course, this implies that in all our examples A, A^2 and $f(A)$ are diagonal matrices which could have been solved efficiently (even without any cg or Lanczos scheme, but directly). However, as said before, the Lanczos type algorithms do not take advantage from this with respect to their convergence behaviour. A Krylov subspace for A and b is generated in which the structure of A does not play any role at all. With the obtained Krylov subspace information for A, we have solved systems like $A^2x = b$ or $f(A)x = b$ along the lines presented in Sections 4 and 5. This means that neither A^2 nor $f(A)$ had to be computed explicitly, though this should have been rather trivial for our sample matrices. It should be mentioned that these methods have also been tested in more realistic and practical situation that A is a non-diagonal matrix, giving then essentially the same kind of results, with regard to the convergence behaviour. In fact the model matrices have been chosen so that their diagonal elements reflect the spectra of the more complex problems that we have solved.

Our examples are restricted to two different diagonal matrices A_{I} and A_{II}.

$$A_{\mathrm{I}} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{900}),$$
$$\lambda_1 = 0.034, \quad \lambda_2 = 0.082, \quad \lambda_3 = 0.127, \quad \lambda_4 = 0.155, \quad \lambda_5 = 0.19,$$
$$\lambda_j = 0.2 + (j-5)/895, \quad j = 6, 7, 8, \ldots, 900.$$

This spectrum is, with regard to its extremal smallest eigenvalues, modelled roughly on the type of spectra that is typical for the preconditioned matrix in ICCG. It has been observed that orthogonality among the vectors r_j is maintained quite well for the number of iteration steps that is necessary to solve $A_{\mathrm{I}}x = b$ with acceptable accuracy.

$$A_{\mathrm{II}} = \mathrm{diag}(\mu_1, \mu_2, \ldots, \mu_{900}),$$
$$\mu_1 = 214.827, \quad \mu_2 = 57.4368, \quad \mu_3 = 48.5554, \quad \mu_4 = 35.0624,$$
$$\mu_5 = 27.3633, \quad \mu_6 = 21.8722, \quad \mu_7 = 17.7489,$$
$$\mu_j = 1.0 + 15.6624 * (j-8)/892, \quad j = 8, 9, \ldots, 900.$$

This spectrum has been modelled roughly on a spectrum that has been encountered in an application of the MICCG method. For this spectrum the orthogonality among the r_j vectors is lost very soon (after 15 iteration steps, say), long before $A_{\mathrm{II}}x = b$ has been solved with a reasonable accuracy.

Example 1. $A_{\mathrm{I}}x = b$ and $A_{\mathrm{I}}x = \tilde{b}$ are solved simultaneously, using the relations (3.1), (3.3) and (3.4). $b = (1, 1, 1, \ldots, 1)^{\mathrm{T}}$ and in order to demonstrate the instability of (3.3), $\tilde{b}$ has been chosen equal to b. The results are given in Table 1.

Example 2. Now $A_{\mathrm{II}}x = b$ and $A_{\mathrm{II}}x = \tilde{b}$ are solved, using (3.3a) instead of (3.3). Again we have chosen $b = (1, 1, 1, \ldots, 1)^{\mathrm{T}}$ and $\tilde{b} = b$. Though the type of spectrum of A_{II} leads to an early loss of orthogonality among the r_j vectors, we do not observe the instability in the solution of the second system as observed in Example 1. The results in Table 2 illustrate the superiority of (3.3a) over (3.3).

Example 3. Now we give an example of two systems with right-hand sides that are not very similar. $A_{\mathrm{I}}x = b$, $b = (1, 1, \ldots, 1)^{\mathrm{T}}$ and $A_{\mathrm{I}}x = \tilde{b}$, $(b)_k = (1/k)$. The relations (3.1), (3.3a) and (3.4) are used. For the results see Table 3.

Table 1
Example 1

i	$\|A_I x_i - b\|_2$	$\|A_I \hat{x}_i - \tilde{b}\|$
0	30.0	30.0
5	1.326	1.326
10	0.3988	0.3988
20	$0.1636\ 10^{-2}$	$0.1636\ 10^{-2}$
30	$0.7286\ 10^{-6}$	$0.7286\ 10^{-6}$
40	$0.1464\ 10^{-9}$	$0.1588\ 10^{-6}$
47	$0.3371\ 10^{-12}$	$0.4080\ 10^{-4}$

Note: the orthogonality among the r_j's is significantly lost around $i = 35$.

Table 2
Example 2

i	$\|A_{II} x_i - b\|_2$	$\|A_{II} \hat{x}_i - \tilde{b}\|_2$
0	30.0	30.0
10	1.759	1.759
20	$0.1922\ 10^{-1}$	$0.1922\ 10^{-1}$
30	$0.4110\ 10^{-3}$	$0.4110\ 10^{-3}$
40	$0.8440\ 10^{-5}$	$0.8440\ 10^{-5}$
50	$0.1617\ 10^{-6}$	$0.1617\ 10^{-6}$
60	$0.2765\ 10^{-8}$	$0.2765\ 10^{-8}$
70	$0.6080\ 10^{-10}$	$0.6079\ 10^{-10}$

Table 3
Example 3

i	$\|A_I x_i - b\|_2$	$\|A_I \hat{x}_i - \tilde{b}\|_2$
0	30.0	1.28
5	1.33	1.59
10	0.399	0.576
15	$0.421\ 10^{-1}$	0.201
20	$0.164\ 10^{-2}$	0.120
30	$0.729\ 10^{-6}$	$0.555\ 10^{-1}$

Note: Obviously the vector $\tilde{b}$ is not very well contained in the Krylov space $K^i(A;\ b)$, for $i \leqslant 30$.

Table 4
Example 4

i	$\|A_I^2 \hat{x}_i - b\|_2$	$\|A_I^2 x_i - b\|_2$	i	$\|A_I^2 \hat{x}_i - b\|_2$	$\|A_I^2 x_i - b\|_2$
0	$0.21\ 10^{2}$	$0.21\ 10^{2}$	40	$0.16\ 10^{-8}$	$0.28\ 10^{-2}$
5	0.34	0.75	45	$0.22\ 10^{-10}$	$0.29\ 10^{-2}$
10	0.18	0.15	50		$0.36\ 10^{-2}$
15	$0.49\ 10^{-2}$	$0.34\ 10^{-1}$	60		$0.10\ 10^{-2}$
20	$0.27\ 10^{-2}$	$0.16\ 10^{-1}$	70		$0.49\ 10^{-4}$
25	$0.20\ 10^{-3}$	$0.97\ 10^{-2}$	80		$0.18\ 10^{-5}$
30	$0.53\ 10^{-5}$	$0.63\ 10^{-2}$	100		$0.21\ 10^{-8}$
35	$0.99\ 10^{-7}$	$0.45\ 10^{-2}$	115		$0.13\ 10^{-10}$

Note: For this situation we have that $c \approx 35.3$ (cf. (4.13)).

Table 5
Example 5

i	$\|A_{II}^2\hat{x}_i - b\|_2$	i	$\|A_{II}^2\hat{x}_i - b\|_2$
0	$4.65\ 10^4$	35	$2.02\ 10^{-2}$
5	$1.16\ 10^3$	40	$4.89\ 10^{-3}$
10	$8.98\ 10^1$	45	$6.22\ 10^{-4}$
15	$1.51\ 10^1$	50	$8.85\ 10^{-5}$
20	5.78	55	$2.12\ 10^{-5}$
25	$5.59\ 10^{-1}$	60	$7.33\ 10^{-6}$
30	$1.03\ 10^{-1}$	65	$2.94\ 10^{-7}$

Note: Also about 60 iteration steps are required to reduce $\|Ay_0 - b\|_2$ by a factor 10^{-11}.

Table 6
Example 6

i	$\|f(A_I)\hat{x}_i - b\|_2$
30	$1.13\ 10^{-6}$
40	$2.21\ 10^{-9}$
50	$1.44\ 10^{-11}$

Note: We could not obtain a smaller residual norm by carrying out more Lanczos steps.

Example 4. For the equation $A_I^2x = b$, $(b)_k = \lambda_k^2$ (hence $(x)_k = 1$), the residual norms for $\hat{x}_i$ and x_i (see Section 4) are listed in Table 4.

Example 5. The equation $A_{II}^2x = b$, $(b)_k = \lambda_k^2$ (hence $(x)_k = 1$), is solved with the algorithm given in Section 4. The results are given in Table 5.

Example 6. $f(A_I)x = b$, b such that $(x)_k = 1$, $k = 1,\ldots,900$, and $f(A) = (A - 0.5I)^2 + 0.1I$. The iterands x_i have been computed via (5.8). We have used Paige's stable Lanczos algorithm [4], with starting vector b, in order to generate the Krylov space. For the results see Table 6.

Example 7. Finally we solve $f(A_I)x = b$, b such that $(x)_k = 1$, $k = 1,\ldots,900$, and $f(A) = e^A$. Since A_I's eigenvalues are all relatively small, (they are in the range [0.034, 1.2), the right-hand side b is rather close to the solution $x = (1,\ldots,1)^T$. This may explain that after 20 Lanczos steps we found that $\|f(A_I)\hat{x}_{20} - b\|_2 = 8.66\ 10^{-12}$, and this was as accurate as we could get using this algorithm in our working precision of 48 bits.

Acknowledgements

I would like to thank the Royal Dutch Meteorological Institute, KNMI, for their kind permission to carry out the numerical experiments with their computational facilities. Specially I am grateful for the help offered by mr. H. Hendriks and mr. T. van Dijk, both at KNMI.

References

[1] G.H. Golub and C.F. Van Loan, *Matrix Computations* (North Oxford Academic Press, Oxford, 1983).
[2] M.R. Hesteness and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Stand.* **49** (1952) 409–436.
[3] C.C. Paige, The computation of eigenvalues and eigenvectors of very large sparse matrices, Ph.D. Thesis, University of London, 1971.
[4] C.C. Paige, Computational variants of the Lanczos method for the eigenproblem, *J. Inst. Math. Applics.* **10** (1972) 373–381.
[5] C.C. Paige, Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix, *J. Inst. Math. Applics.* **18** (1976) 341–349.
[6] B.N. Parlett, *The Symmetric Eigenvalue Problem* (Prentice Hall, Englewood Cliffs, 1980).
[7] B.N. Parlett, A new look at the Lanczos algorithm for solving symmetric systems of linear equations, *Lin. Alg. Appl.* **29** (1980) 323–346.
[8] B.N. Parlett and D.S. Scott, The Lanczos algorithm with selective orthogonalization, *Math. Comp.* **33** (1979) 217–238.
[9] A. Ruhe, Numerical aspects of the Gram–Schmidt orthogonalization of vectors, *Lin. Alg. Appl.* **52/53** (1983) 591–601.
[10] Y. Saad, On the rate of convergence of the Lanczos and block Lanczos methods, *SIAM J. Num. Anal.* **17** (1980) 687–706.
[11] Y. Saad, On the Lanczos method for solving symmetric linear systems with several right-hand sides, Techn. Report YALEU/DCS/RR-396, Yale University, New Haven, June 1985.
[12] A. van der Sluis and H.A. van der Vorst, The rate of convergence of conjugate gradients, *Numer. Math.* **48** (1986) 543–560.