Complex conjugate gradient methods

Pascal Joly

CNRS, Laboratoire d'Analyse Numérique, Université Pierre et Marie Curie, Paris, France

Gérard Meurant

CEA, Centre d'Etudes de Limeil–Valenton, 94195 Villeneuve-St. Georges, France

Communicated by C. Brezinski
Received 5 January 1993

Linear systems with complex coefficients arise from various physical problems. Examples are the Helmholtz equation and Maxwell equations approximated by finite difference or finite element methods, that lead to large sparse linear systems. When the continuous problem is reduced to integral equations, after discretization, one obtains a dense linear system. The resulting matrices are generally non-Hermitian but, most of the time, symmetric and consequently the classical conjugate gradient method cannot be directly applied. Usually, these linear systems have to be solved with a large number of unknowns because, for instance, in electromagnetic scattering problems the mesh size must be related to the wave length of the incoming wave. The higher the frequency of the incoming wave, the smaller the mesh size must be. When one wants to solve 3D-problems, it is no longer practical to use direct method solvers, because of the huge memory they need. So iterative methods are attractive for this kind of problems, even though their convergence cannot be always guaranteed with theoretical results. In this paper we derive several methods from a unified framework and we numerically compare these algorithms on some test problems.

Keywords: Conjugate gradient methods, linear systems.

Subject classification: AMS(MOS) 65F10.

1. Introduction

This paper is concerned with the study of complex conjugate gradient-like methods, and is a generalisation of a previous paper that addressed the real case [11]. We introduce a general framework from which most of the well known methods can be derived and generalized to solving complex non-Hermitian linear systems. Examples are the Helmholtz equation and Maxwell equations approximated by finite difference or finite element methods, that lead to large sparse linear systems. Usually, these linear systems have to be solved with a large number of unknowns because, for instance, in electromagnetic scattering problems the mesh size must be related to the wave length of the incoming wave. The higher the frequency of the incoming wave, the smaller the mesh size must be. When one wants to solve 3D-problems, it is no longer practical to use direct method solvers, because of the huge memory they need. So iterative methods are attractive for this kind of problems.

The paper is organized as follows: in section 2, we derive the general algorithm and prove some of its properties. The next few sections study some realizations of the general algorithm. In section 3, we consider the normal equation method. Section 4 is devoted to the generalization of Orthomin. Section 5 is concerned with Gmres. A variant of the Biconjugate gradient method is introduced in section 6. From this algorithm, a Biconjugate Gradient Squared method is derived in section 7. Finally, section 8 presents some numerical experiments on some model problems and some 2D-problems, comparing the different methods.

The interested reader may also look at a recent paper by Ashby et al. [1] for a complete review of those methods, although the goal of their paper, which is to introduce a general taxonomy of generalizations of the conjugate gradient method, is different from ours.

2. A general framework

To solve the linear system

$$\text{(P)} \qquad Ax = b\,,$$

where $A \in \mathbb{C}^{n\times n}$ is a non-singular matrix and $b \in \mathbb{C}^n$, we introduce the functional $J : \mathbb{C}^n \mapsto \mathbb{R}^+$, such that

$$\forall r \in \mathbb{C}^n, \quad J(r) = (r, Hr)\,,$$

with $(\cdot,\cdot)$ being the usual complex scalar product of $\mathbb{C}^n$ defined as: $(x, y) = \sum_{i=1}^n x_i\bar{y}_i$, where $\bar{y}_i$ stands for the complex conjugate of y_i and $H \in \mathbb{C}^{n\times n}$ is an Hermitian matrix.

In the following, r stands for the residual $b - Ax$.

As J is a strictly convex functional on $\mathbb{C}^n$, there exists a unique minimum in $r_m \in \mathbb{C}^n$. From the property that $J(r) \geqslant 0, \forall r \in \mathbb{C}^n$, it follows that $r_m = 0$ (see [12] for example).

By an appropriate choice of the matrix H, many different methods can be generated. Another matrix $K \in \mathbb{C}^{n\times n}$ is also introduced, which we suppose to be definite.

A general minimization algorithm of the functional J can be described as

- Choose x^0 in $\mathbb{C}^n$, set $d^0 = KA^{\mathrm{H}}Hr^0$, then generate vectors $\{d^0, d^1, \ldots, d^k\}$ in $\mathbb{C}^n$ orthogonal in the scalar product related to the Hermitian matrix $N = A^{\mathrm{H}}HA$, where A^{H} denotes the Hermitian transpose of A;
- minimize J over each subspace $x^0 + \langle Kg^0, Kg^1, \ldots, Kg^k\rangle$, with g^k being the gradient of J at x^k.

The general algorithm is therefore

```
Initialization
        Choose $x^0 \in \mathbb{C}^n$,
        set $r^0 = b - Ax^0$,
        $g^0 = A^{\mathrm{H}}Hr^0$,
        $d^0 = Kg^0$.
Iterations: for $k = 0, 1, \ldots$ until convergence do
    (1) Minimize $J$ over $x^0 + \langle d^0, d^1, \ldots, d^k \rangle$ by
            $\alpha_k = (g^k, d^k)/(d^k, Nd^k)$,
            $x^{k+1} = x^k + \alpha_k d^k$,
            $r^{k+1} = r^k - \alpha_k Ad^k$,
            $g^{k+1} = g^k - \alpha_k Nd^k$.
    (2) Generate the new direction by
            $d^{k+1} = Kg^{k+1} + \sum_{l=0}^{k} \beta_{k+1}^l d^l$,
            $\beta_{k+1}^l = -(Kg^{k+1}, Nd^l)/(d^l, Nd^l), \quad 0 \leqslant l \leqslant k$.
```

Remark

This algorithm is formally equivalent to the classical conjugate gradient method, but the functional J to be minimized and the way the directions are generated are more general. Below, we prove some properties of the general algorithm.

2.1. SOME PROPERTIES OF THE COMPLEX ALGORITHM

$$(d^k, Nd^l) = 0, \qquad \forall k \neq l, \tag{2.1}$$

$$(g^k, d^l) = 0, \qquad 0 \leqslant l < k, \tag{2.2}$$

$$(g^l, d^k) = (g^0, d^k), \qquad 0 \leqslant l \leqslant k, \tag{2.3}$$

$$(g^k, d^k) = (g^k, Kg^k), \tag{2.4}$$

$$(Kg^k, Nd^k) = (d^k, Nd^k), \tag{2.5}$$

$$(g^k, Kg^l) = 0, \qquad 0 \leqslant l < k, \tag{2.6}$$

$$E^k = \langle d^0, d^1, \ldots, d^k \rangle \tag{2.7}$$

$$= \langle Kg^0, Kg^1, \ldots, Kg^k \rangle$$

$$= \langle d^0, KNd^0, \ldots, (KN)^k d^0 \rangle, \tag{2.8}$$

$$\text{dimension } E^k = k+1 \qquad \text{if } g^k \neq 0, \tag{2.9}$$

$$x^{k+1} \text{ realizes the minimum of } J \text{ over } x^0 + E^k, \tag{2.10}$$

$$g^n = 0; \tag{2.11}$$

if we suppose that K is Hermitian then we have

$$\beta_{k+1}^l = 0, \quad 0 \leqslant l < k, \tag{2.12}$$

and

$$\beta_{k+1}^k = (g^{k+1}, Kg^{k+1})/(g^k, Kg^k). \tag{2.13}$$

Proof

Properties (2.1) to (2.6) are obtained by induction:

(2.1) $(d^1, Nd^0) = (Kg^1 + \beta_1^0 d^0, Nd^0) = 0$ by definition of β_1^0,

(2.2) $(g^1, d^0) = (g^0 - \alpha_0 Nd^0, d^0) = 0$ by definition of α_0,

(2.3) $(g^1, d^1) = (g^0 - \alpha_0 Nd^0, d^1) = (g^0, d^1)$ by (2.1),

(2.4) $(g^0, d^0) = (g^0, Kg^0)$ by definition of d^0,

(2.5) $(Kg^0, Nd^0) = (d^0, Nd^0)$ by definition of d^0,

(2.6) $(g^1, Kg^0) = (g^0 - \alpha_0 Nd^0, Kg^0) = (g^0, Kg^0) - \alpha_0 (Nd^0, Kg^0) = (g^0, d^0)$
$-\alpha(Nd^0, d^0) = 0$ by definition of α_0.

Suppose these properties are satisfied until $k-1$, then

(2.1) $(d^k, Nd^l) = (Kg^{k-1} + \sum_{i=1}^{k-1} \beta_{k-1}^i d^i, Nd^l) = (Kg^{k-1}, Nd^l) + \beta_{k-1}^l (d^l, Nd^l) = 0$
for $0 \leqslant l < k$ by definition of β_{k-1}^i; (1) follows because N is Hermitian.

(2.2) $(g^k, d^l) = (g^l - \sum_{i=1}^{k-l} \alpha_i Nd^i, d^l) = (g^l, d^l) - \alpha_l (d^l, Nd^l) = 0$ for $0 \leqslant l < k$ from (2.1) and the definition of α_l.

(2.3) $(g^l, d^k) = (g^0 - \sum_{i=0}^{l-1} \alpha_i Nd^i, d^k) = (g^0, d^k)$ for $0 \leqslant l \leqslant k$ from (2.1).

(2.4) $(g^k, d^k) = (g^k, Kg^k + \sum_{i=1}^{k-1} \beta_{k-1}^i d^i) = (g^k, Kg^k)$ from (2.2).

(2.5) $(Kg^k, Nd^k) = (d^k - \sum_{i=1}^{k-1} \beta_{k-1}^i d^i, Nd^k) = (d^k, Nd^k)$ from (2.1).

(2.6) $(g^k, Kg^l) = (g^k, d^l - \sum_{i=1}^{l-1} \beta_{l-1}^i d^i) = 0$ for $0 \leqslant l < k$ from (2.2).

So properties (2.1) to (2.6) are satisfied for all k.

Properties (2.7) and (2.9) are obviously true for $k = 1$. Suppose they are true until $k-1$ included. Then

$$Kg^k = Kg^{k-1} - \alpha_{k-1} KNd^{k-1} \text{ by definition},$$

but

$$Kg^{k-1} \in \langle d^0, KNd^0, \ldots, (KN)^{k-1} d^0 \rangle$$

and

$$d^{k-1} \in \langle d^0, KNd^0, \ldots, (KN)^{k-1} d^0 \rangle,$$

so

$$Kg^k \in \langle d^0, KNd^0, \ldots, (KN)^k d^0 \rangle .$$

On the other hand,

$$d^k = Kg^k + \sum_{l=0}^{k-1} \beta_k^l d^l ,$$

but

$$d^l \in \langle Kg^0, Kg^1, \ldots, Kg^{k-1} \rangle, \quad 0 \leqslant l < k ,$$

so

$$d^k \in \langle Kg^0, Kg^1, \ldots, Kg^k \rangle .$$

Combining these results with the induction hypothesis, we obtain:

$$\langle d^0, d^1, \ldots, d^k \rangle \subset \langle Kg^0, Kg^1, \ldots, Kg^k \rangle \subset \langle d^0, KNd^0, \ldots, (KN)^k d^0 \rangle .$$

But from (2.1) dimension $\langle d^0, d^1, \ldots, d^k \rangle = k+1$ and finally

$$\langle d^0, d^1, \ldots, d^k \rangle = \langle Kg^0, Kg^1, \ldots, Kg^k \rangle = \langle d^0, KNd^0, \ldots, (KN)^k d^0 \rangle .$$

The case $g^k = 0$ will be handled later on.

(2.10)

$$\begin{aligned} J(r^k) &= J\left(r^0 - \sum_{l=0}^{k} \alpha_l A d^l\right) \\ &= J(r^0) - \sum_{l=0}^{k} \alpha_l (Ad^l, Hr^0) - \sum_{l'=0}^{k} \bar{\alpha}_{l'} (r^0, HAd^{l'}) + \sum_{l,l'=0}^{k} \alpha_l \bar{\alpha}_{l'} (d^l, Nd^l) . \end{aligned}$$

Now, using (2.1), the relation $g^l = A^H H r^l$ and (2.3), we obtain

$$J(r^k) = J(r^0) - \sum_{l=0}^{k} \alpha_l \overline{(g^l, d^l)} + \bar{\alpha}_l (g^l, d^l) + \sum_{l=0}^{k} \alpha_l \bar{\alpha}_l (d^l, Nd^l) .$$

For given x^0, k and l, $J(r^k)$ is a quadratic function in α_l.

Define $J_l(\alpha) = -(\alpha\overline{(g^l, d^l)} + \bar{\alpha}(g^l, d^l) + \alpha\bar{\alpha}(d^l, Nd^l), 0 \leqslant l \leqslant k$, where $\alpha \in \mathbb{C}$. The minimum of J_l is obtained with $\alpha_l = (g^l, d^l)/(d^l, Nd^l)$. Finally, from the calculation of $\alpha_l, 0 \leqslant l \leqslant k$, at each step of the algorithm, x^{k+1} realizes the minimum of J over the subspace $x^0 + E^k$.

(2.11) If at step $k < n-1, g^k = 0$, then the algorithm has converged, because $g^k = A^H H r^k$. Otherwise, for $k = n-1, E^{n-1}$ has dimension n, and x^n realizes the minimum of J, that is $J(r^n) = 0 = (r^n, Hr^n)$, hence $r^n = 0$! Note that this result can be directly derived from (2.6).

(2.12) Suppose now that the matrix K is Hermitian, then

$$\begin{aligned}(Kg^{k+1}, Nd^l) &= \left(Kg^{k+1}, \frac{1}{\alpha_l}(g^l - g^{l+1})\right)\\ &= \frac{1}{\alpha_l}(g^{k+1}, K(g^l - g^{l+1}))\\ &= 0, \quad \text{for } 0 \leqslant l \leqslant k,\end{aligned}$$

and $\beta^l_{k+1} = 0$ for $0 \leqslant l < k$.

(2.13) We have $\alpha_k = (g^k, d^k)/(d^k, Nd^k) = (g^k, Kg^k)/(d^k, Nd^k)$, so

$$\begin{aligned}\beta^k_{k+1} &= -(Kg^{k+1}, Nd^k)/(d^k, Nd^k)\\ &= -(Kg^{k+1}, \frac{1}{\alpha_k}(g^k - g^{k+1}))/(d^k, Nd^k)\\ &= \frac{1}{\alpha_k}(Kg^{k+1}, g^{k+1})/(d^k, Nd^k)\\ &= (g^{k+1}, Kg^{k+1})/(g^k, Kg^k).\end{aligned}$$

It follows that β^k_{k+1} is real when the matrix K is Hermitian.

Remark

In this algorithm, no breakdown can occur as $\alpha_k = 0$ leads to $(g^k, Kg^k) = 0$, that is, $g^k = 0$, because the matrix K is definite.

Similarly $(d^k, Nd^k) = 0$ leads to $d^k = 0$, that is, $g^k = 0$ from (2.4).

2.2. CONVERGENCE RESULTS

From the definitions of $J(r) = (r, Hr)$ and α_k, it follows that

$$J(r^{k+1}) = J(r^k) - |(g^k, Kg^k)|^2/(d^k, Nd^k),$$

so the convergence of the algorithm is monotone, in case of equality $J(r^{k+1}) = J(r^k)$ then $(g^k, Kg^k) = 0$, and $g^k = 0$. This means that the residual is monotonically decreasing when measured in the norm defined by matrix H.

Furthermore, as $J(r^k) = (r^k, Hr^k) = (g^k, N^{-1}g^k)$:

$$J(r^{k+1})/J(r^k) = 1 - \frac{|(g^k, Kg^k)|}{(g^k, N^{-1}g^k)} \frac{|(g^k, Kg^k)|}{(d^k, Nd^k)}.$$

Suppose now that the matrix K is Hermitian, then we obtain

$$(d^k, Nd^k) = (Kg^k, NKg^k) - \sum_{l=0}^{k-1} |(Kg^k, Nd^l)|^2/(d^l, Nd^l) \leqslant (Kg^k, NKg^k)$$

and then

$$J(r^{k+1})/J(r^k) \leqslant 1 - \frac{|(g^k, Kg^k)|}{(g^k, N^{-1}g^k)} \frac{|(g^k, Kg^k)|}{(Kg^k, NKg^k)};$$

we introduce L such that: $K = L^{\mathrm{H}} \cdot L$, $M = L^{\mathrm{H}} \cdot N \cdot L$ and $h^k = Lg^k$, we obtain

$$J(r^{k+1})/J(r^k) \leqslant 1 - \frac{(h^k, h^k)}{(h^k, M^{-1}h^k)} \frac{(h^k, h^k)}{(h^k, Mh^k)},$$

now using the Kantorovitch inequality [12]

$$J(r^k) \leqslant J(r^0) \left(\frac{cond(M) - 1}{cond(M) + 1} \right)^{2k}.$$

Remark

Following Golub and Meurant [7], and using the Chebyshev polynomials properties it can be proved (in case K is Hermitian) that

$$J(r^k) \leqslant J(r^0) \left(\frac{cond(M)^{1/2} - 1}{cond(M)^{1/2} + 1} \right)^{2k}.$$

Many algorithms can be generated by an appropriate choice of matrices H and K. For example, when A is Hermitian, the choice $H = A^{-1}$ and $K = I$ leads to the classical complex conjugate gradient method. In that case, $M = A$. With the same hypothesis, if we choose $H = A^{-1}$ and $K = (l \cdot l^{\mathrm{H}})^{-1}$ being the inverse of an incomplete complex Cholesky factorization of A, we obtain the complex conjugate gradient method preconditioned by the matrix K. Table 1 summarizes the definition of the most popular methods. The convergence rate is related to the condition number of the matrix $M = L^{\mathrm{H}} \cdot A^{\mathrm{H}} \cdot H \cdot A \cdot L$ with $K = L^{\mathrm{H}} \cdot L$.

Notice that GCR and Orthomin are generated by the same choices. Mathematically they are all equivalent. However, we will see that to be used in practice, these methods have to be slightly modified. The modified versions are no longer equivalent.

Table 1
Classical algorithms according to H and K.

Algorithm	H	K	M	Convergence
Conjugate Gradient	A^{-1}	I	A	A Hermitian
Prec. Conjugate Gradient	A^{-1}	$l^{-\mathrm{H}}l^{-1}$	$l^{-\mathrm{H}}Al^{-1}$	A Hermitian
Gen. Conjugate Residual	I	$A^{-\mathrm{H}}$	$(AL)^{\mathrm{H}}AL$	A definite
Orthomin	I	$A^{-\mathrm{H}}$	$(AL)^{\mathrm{H}}AL$	A definite
Gmres	I	$A^{-\mathrm{H}}$	$(AL)^{\mathrm{H}}AL$	A regular
Normal Equation	I	I	$A^{\mathrm{H}}A$	A regular
Minimal error	$(AA^{\mathrm{H}})^{-1}$	$A^{\mathrm{H}}A$	$A^{\mathrm{H}}A$	A regular

It is not the aim of this paper to study all the many possibilities of the general algorithm, so we limit ourselves to the most interesting ones: the normal equation method (see for example Eisenstat et al. [4]), the complex Orthomin method (see Vinsome [16]), the complex Gmres algorithm (see Saad and Schultz [14]), the Biconjugate method (Fletcher [5]) and the accelerated variant Conjugate Gradient Squared (Sonneveld [15]). For these methods, we study how they are generated and their properties (in exact arithmetic) without preconditioning.

3. The normal equation method

The normal equation method is very popular to solve non-Hermitian linear systems. It is derived from the choice $H = I, K = I$ and minimizes the functional $J(r) = (r, r)$, that is, the l_2 norm of the residual. The algorithm can be written as

```
*Initialization*
    Choose $x^0 \in \mathbb{C}^n$,
    set $r^0 = b - Ax^0$,
    $d^0 = r^0$
*Iterations*: for $k = 0, 1, \ldots$ until convergence do
  (1) Minimize $J$ over $x^0 + \langle d^0, d^1, \ldots, d^k \rangle$ by
          $\alpha_k = (A^H r^k, A^H r^k)/(Ad^k, Ad^k)$,
          $x^{k+1} = x^k + \alpha_k d^k$,
          $r^{k+1} = r^k - \alpha_k A d^k$.
  (2) Generate the new direction by
          $d^{k+1} = A^H r^{k+1} + \beta_{k+1} d^k$,
          $\beta_{k+1} = (A^H r^{k+1}, A^H r^{k+1})/(A^H r^k, A^H r^k)$.
```

Notice that there are two matrix vector products per iteration and that the access to A^H is requested. This last point is not too much of a problem if A is symmetric, in that case only A has to be stored and the conjugates of the coefficients can be computed every time they are needed. Another problem is that $M = A^H \times A$, so the convergence rate is related to the condition number of A squared. The main advantage is that this algorithm is very robust.

If we suppose A non-singular, this algorithm converges in at most n iterations.

Remark

As the convergence rate of the normal equation method is, more precisely, governed by the singular values of the matrix A, rather than its eigenvalues, it may remain competitive towards Gmres or the Biconjugate Gradient method for certain classes of linear systems (see Nachtigal et al. [13] for examples).

4. Complex Orthomin

If we choose $H = I$ and $K = A^{-\mathrm{H}}$, we obtain the General Conjugate Residual algorithm, which minimizes $J(r) = (r, r)$. This method is summarized as

```
Initialization
    Choose $x^0 \in \mathbb{C}^n$,
    set $r^0 = b - Ax^0$,
    $d^0 = r^0$.
Iterations: for $k = 0, 1, \ldots$ until convergence do
  (1) Minimize $J$ over $x^0 + \langle d^0, d^1, \ldots, d^k \rangle$ by
        $\alpha_k = (r^k, Ad^k)/(Ad^k, Ad^k)$,
        $x^{k+1} = x^k + \alpha_k d^k$,
        $r^{k+1} = r^k - \alpha_k Ad^k$.
  (2) Generate the new direction by
        $d^{k+1} = r^{k+1} + \sum_{l=0}^{k} \beta_{k+1}^l d^l$,
        $\beta_{k+1}^l = -(Ar^{k+1}, Ad^l)/(Ad^l, Ad^l)$.
```

If the matrix A is definite (corresponding to the case K definite), this algorithm converges in at most n iterations, but there are two drawbacks:

- all the generated directions d^l and products Ad^l have to be stored $(1 \leqslant l \leqslant k)$, this makes this algorithm memory bound when the number of iterations is large;
- two matrix vector products have to be computed at each step.

Vinsome [16] first proposed to limit the number of directions to a fixed value m set in advance and depending on the available memory. The definition of the new direction is then modified as:

$$d^{k+1} = r^{k+1} + \sum_{l=k-m+1}^{k} \beta_{k+1}^l d^l,$$

keeping only the last m directions.

Then a new set of vectors z^l such that $z^l = Ad^l$ is introduced. Combining these two modifications of GCR leads to the well-known Orthomin formulation:

Initialization

```
    Choose $x^0 \in \mathbb{C}^n$,
    set $r^0 = b - Ax^0$,
    $d^0 = r^0$.
    $z^0 = Ad^0$.
*Iterations*: for $k = 0, 1, \ldots$ until convergence do
  (1) Minimize $J$ over $x^0 + \langle d^0, d^1, \ldots, d^k \rangle$ by
          $\alpha_k = (r^k, z^k)/(z^k, z^k)$,
          $x^{k+1} = x^k + \alpha_k d^k$,
          $r^{k+1} = r^k - \alpha_k z^k$.
  (2) Generate the new direction by
          $d^{k+1} = r^{k+1} + \sum_{l=k-m+1}^{k} \beta_{k+1}^l d^l$,
          $z^{k+1} = Ar^{k+1} + \sum_{l=k-m+1} \beta_{k+1} z^l$,
          $\beta_{k+1}^l = -(Ar^{k+1}, z^l)/(z^l, z^l)$.
```

Now just one matrix vector product remains per iteration: Ar^{k+1}, and the number of stored vectors is only $2m$. Of course, the properties of the algorithm are no longer the same.

$$(Ad^k, Ad^l) = 0, \qquad k - m \leqslant l < k, \tag{4.1}$$

$$(r^k, Ad^l) = 0, \qquad k - m \leqslant l < k, \tag{4.2}$$

$$(r^l, Ad^k) = (r^{k-m}, Ad^k), \qquad k - m \leqslant l \leqslant k, \tag{4.3}$$

$$(r^k, Ad^k) = (r^k, Ar^k), \tag{4.4}$$

$$(Ar^k, Ad^k) = (Ad^k, Ad^k), \tag{4.5}$$

$$(r^k, Ar^l) = 0, \qquad k - m \leqslant l < k, \tag{4.6}$$

$$E^k = \langle d^{k-m}, d^{k-m+1}, \ldots, d^{k-1} \rangle, \tag{4.7}$$

$$\text{dimension } E^k = m, \qquad \text{if } r^k \neq 0, \tag{4.8}$$

$$x^{k+1} \text{ realizes the minimum of } J \text{ over } x^{k-m} + E^k, \tag{4.9}$$

$$\|r^k\| \mapsto 0 \qquad \text{when } k \mapsto +\infty \text{ if } A \text{ is definite}. \tag{4.10}$$

Proof

Properties (4.1) to (4.9) are obtained by induction as in section 2. To prove (4.10), we use the equality

$$J(r^{k+1}) = J(r^k) - |(g^k, Kg^k)|^2/(d^k, Nd^k),$$

that is, in this particular case

$$\|r^{k+1}\|^2 = \|r^k\|^2 - |(r^k, Ar^k)|^2/(Ad^k, Ad^k)\,.$$

So the alternative is:

- there exists one number $k, k < \infty$ such that $J(r^{k+1}) = J(r^k)$, then $g^k = 0$, and $r^k = 0$;
- otherwise, the sequence $(J(r^k))_{k \in N}$ is strictly decreasing, and bounded below by 0. So it converges towards a finite limit J^∞.

Then, we notice that

$$\begin{aligned}\|r^{k+1} - r^k\| &= \|r^{k+1}\| + \|r^k\| - (r^{k+1}, r^k) - (r^k, r^{k+1}) \\ &= \|r^k\| - \|r^{k+1}\| = J(r^k) - J(r^{k+1})\,.\end{aligned}$$

This implies that the sequence $(r^k)_{k \in N}$ converges towards a finite limit r^∞. Furthermore, from the relation

$$(Ad^k, Ad^k) = (Ar^k, Ad^k)$$

we see that $\|Ad^k\| \leqslant \|Ar^k\|$.

All these results show that $|(r^k, Ar^k)| \mapsto 0$ when $k \mapsto \infty$, and finally $(r^\infty, Ar^\infty) = 0$, that is, $r^\infty = 0$. So, this variant of the general residual algorithm can only be used as an iterative method. This is not really important as n iterations are usually not needed in a realistic computation, for which only an approximation of the solution is required, according to the convergence criterion $\|r^k\| < \epsilon \|r^0\|$. This method has been used successfully for a large class of rather well-conditioned problems (see [2] for example).

Remarks

It is also possible to restart the algorithm after m iterations (once x^m has been computed, the method is restarted with $x^0 = x^m$). This method is known as GCR(m), Generalized Conjugate Residual algorithm with m directions, and is not equivalent to Orthomin (m).

According to property (4.4), no breakdown can occur when A is definite.

As we know exactly the decrease of J on one iteration:

$$\|r^{k+1}\|^2 = \|r^k\|^2 - |(r^k, Ar^k)|^2/(Ad^k, Ad^k)\,,$$

it is also possible to restart the algorithm when the change remains too small. This modification of the algorithm is very cheap, but not always successful.

5. Complex Gmres

To reduce the storage needed by the previous algorithm, Saad and Schultz [14] have proposed another modification, constructing a basis of the Krylov space as in the classical Lanczos method:

$$d^{k+1} = Ad^k - \sum_{l=0}^{k} h_{k+1,l} d^l .$$

The coefficients $h_{k+1,l}$ are chosen in order to obtain an orthonormal basis, so the storage of vectors $Ad^0, Ad^1, \ldots, Ad^k$ is avoided. We are now looking for a solution $x^k = x^0 + d^k z^k$, where $z^k \in \mathbb{C}^k$, and $D^k = [d^0, d^1, \ldots, d^k]$. The residual r^k can be written as

$$r^k = b - Ax^k = r^0 - AD^k z^k = r^0 - D^{k+1} H_e^k z^k ,$$

where

$$H_e^k = \begin{bmatrix} H^k \\ 0 \ldots 1 \end{bmatrix}$$

and H^k is an upper Hessenberg $k \times k$ matrix.

The vector z^k may be obtained either by an orthogonalization argument (that is, force r^k to be orthogonal to $E^k = \langle d^0, d^1, \ldots, d^k \rangle$ as in the General Conjugate Residual algorithm) or by a minimization argument (that is, choose r^k such that $\|r^k\|$ is minimal).

Saad and Schultz used the second option, but this leads to solving a least-squares problem. The complete algorithm is then written as follows:

Initialization

Choose $x^0 \in \mathbb{C}^n$,
set $r^0 = b - Ax^0$,
$d^0 = r^0 / \|r^0\|$.

Iterations

(1) Generate the m directions: for $k = 0, 1, \ldots, m-1$: compute

$$\tilde{d}^{k+1} = Ad^k - \sum_{l=0}^{k} H_{k+1,l}^m d^l ,$$

$$H_{k,l}^m = (Ad^k, d^l) \quad 0 \leqslant l < k \leqslant m-1 ,$$

$$H_{k+1,k}^m = \|\tilde{d}^{k+1}\| ,$$

$$d^{k+1} = \tilde{d}^{k+1} / H_{k+1,k}^m .$$

(2) Minimize J over $x^0 + \langle d^0, d^1, \ldots, d^m \rangle$ by

$$D^{m-1} = [d^0, d^1, \ldots, d^{m-1}] \in \mathbb{C}^{n \times m} ,$$

$$\tilde{H}^m = (D^{m-1})^{\mathrm{H}} \times A \times D^{m-1} \in \mathbb{C}^{m+1 \times m} ,$$

$$\tilde{H}^m = \begin{bmatrix} \tilde{H}^m \\ 0 \ldots 0 H^m_{m+1,m} \end{bmatrix} \in \mathbb{C}^{m+1 \times m},$$

$$e^{m+1} = (1, 0, \ldots, 0)^{\mathrm{T}} \in \mathbb{C}^{m+1},$$

$$z^m \in \mathbb{C}^m \text{ solution of } \min_{z \in \mathbb{C}^m} \|e^{m+1} - H^m z\|,$$

$$x^m = x^0 + D^m z^m.$$

Because of the least-squares calculation of z^k, this algorithm is no longer equivalent to the Generalized Conjugate Residual algorithm, and in particular it can deal with skew-symmetric matrices.

6. The Biconjugate Gradient method

In the previous methods, two drawbacks have been found:

- the use of an Hermitian matrix K may lead to slow convergence, as in the normal equation method;
- on the other hand, when the matrix K is not Hermitian the storage problem leads to use only approximate variants of the complete algorithm.

A promising way is to modify the general algorithm using symmetric non-Hermitian matrices H and K:

$$H = \begin{bmatrix} 0 & A^{\mathrm{H}} \\ A & 0 \end{bmatrix}^{-1} \quad \text{and} \quad K = \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}.$$

Note that this does not fit in the general framework of section 2, as the matrices H and K are not definite anymore. Therefore, we will not get all the properties of the general algorithm, particularly there can be breakdowns of the algorithm (see Gutknecht [8]).

This is equivalent to solving the indefinite linear system of order $2n$:

$$\begin{bmatrix} 0 & A^{\mathrm{H}} \\ A & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_2 \\ b_1 \end{bmatrix}.$$

From sections 2 and 3, we obtain the following algorithm

```
Initialization
        Choose $x_1^0, x_2^0 \in \mathbb{C}^n$,
        set $r_1^0 = b_1 - A x_1^0$,
        $r_2^0 = b_2 - A^H x_2^0$,
        $d_1^0 = r_1^0$,
        $d_2^0 = r_2^0$.
Iterations: for $k = 0, 1, \ldots$ until convergence do
```

(1) Compute the extremum of J by

$$\begin{aligned}\alpha_k &= \operatorname{Re}\{(r_1^k, r_2^k)\}/\operatorname{Re}\{(Ad_1^k, d_2^k)\},\\ x_1^{k+1} &= x_1^k + \alpha_k d_1^k,\\ x_2^{k+1} &= x_2^k + \alpha_k d_2^k,\\ r_1^{k+1} &= r_1^k - \alpha_k A d_1^k,\\ r_2^{k+1} &= r_2^k - \alpha_k A^H d_2^k.\end{aligned}$$

(2) Generate the new directions by

$$\begin{aligned}\beta_{k+1} &= \operatorname{Re}\{(r_1^{k+1}, r_2^{k+1})\}/\operatorname{Re}\{(r_1^k, r_2^k)\},\\ d_1^{k+1} &= r_1^{k+1} + \beta_{k+1} d_1^k,\\ d_2^{k+1} &= r_2^{k+1} + \beta_{k+1} d_2^k.\end{aligned}$$

The CPU cost is approximately twice the one in the Hermitian conjugate gradient method: two vector updates for x^k, r^k and d^k, two matrix vector products by step, but the storage of all of the previous directions is avoided. It is not necessary to compute x_2^k; for example, the choice $r_2^0 = \bar{r}_1^0$, which corresponds to an arbitrary choice of b_2 and x_2^0, allows to simplify the computations. This only modifies the initialization step of the previous algorithm:

$$\begin{aligned}&\text{Choose } x^0 \in \mathbb{C}^n,\\ &\text{set } r_1^0 = b - Ax^0, \quad r_2^0 = \bar{r}_1^0,\\ &\qquad d_1^0 = r_1^0, \qquad\quad d_2^0 = r_2^0.\end{aligned}$$

The following relations are deduced from the general formulation:

$$(r_1^k, r_2^l) = 0, \qquad \forall k \neq l, \tag{6.1}$$

$$(Ad_1^l, d_2^k) = (d_1^k, A^H d_2^l) = 0, \quad \forall k \neq l, \tag{6.2}$$

$$(r_1^k, d_2^l) = (r_2^k, d_1^l) = 0, \qquad \forall k \neq l. \tag{6.3}$$

Remark

This algorithm is different from Jacobs' method [10], recalled in the appendix.

In the important case where A is complex symmetric ($A = A^T$), the relation $r_2^0 = \bar{r}_1^0$ leads to

$$r_2^k = \bar{r}_1^k, d_2^k = \bar{d}_1^k, \quad \forall k.$$

The corresponding algorithm can be written as

```
Initialization
        Choose $x^0 \in \mathbb{C}^n$,
        set $r^0 = b - Ax^0$,
        $d^0 = r^0$.
Iterations: for $k = 0, 1, \ldots$ until convergence do
    (1) Compute the extremum of $J$ by
            $\alpha_k = \mathrm{Re}\{(r^k, \bar{r}^k)\} / \mathrm{Re}\{(Ad^k, \bar{d}^k)\}$,
            $x^{k+1} = x^k + \alpha_k d^k$,
            $r^{k+1} = r^k - \alpha_k A d^k$.
    (2) Generate the new direction by
            $\beta_{k+1} = \mathrm{Re}\{(r^{k+1}, \bar{r}^{k+1})\} / \mathrm{Re}\{(r^k, \bar{r}^k)\}$,
            $d^{k+1} = r^{k+1} + \beta_{k+1} d^k$.
```

Note that the formulas of this algorithm are very close to the ones for the usual Conjugate Gradient algorithm.

7. Biconjugate Gradient Squared method

The Conjugate Gradient Squared method was introduced by Sonneveld [15]. In the BCG method, the vectors r_1^k, r_2^k, d_1^k and d_2^k satisfy

$$\begin{aligned} r_1^k &= \phi_k(A) r_1^0, & d_1^k &= \theta_k(A) r_1^0, \\ r_2^k &= \phi_k(A^{\mathrm{H}}) r_1^0, & d_2^k &= \theta_k(A^{\mathrm{H}}) r_1^0, \end{aligned}$$

where ϕ_k and θ_k are polynomials of degree less than or equal to k, defined by the following recurrences

$$\begin{aligned} \phi_{k+1}(A) &= \phi_k(A) - \alpha_k A \theta_k(A), \\ \theta_{k+1}(A) &= \phi_{k+1}(A) + \beta_{k+1} \theta_k(A). \end{aligned}$$

To speed up the convergence rate of the BCG method, Sonneveld introduced a new algorithm, where the residual after k iterations is $\phi_k^2(A) r^0$ instead of $\phi_k(A) r^0$. When the Biconjugate Gradient method converges, $\phi_k(A)$ is a contraction for large values of k, and so $\phi_k^2(A)$ is a contraction of smaller norm.

With the help of the induction relations between ϕ_k and θ_k, the desired residual $\phi_k^2(A) r^0$ is obtained after a few lines of algebra (given in the appendix). The resulting algorithm, called BiCGS (BiConjugate Gradient Squared method) is then

```
Initialization
        Choose $x^0 \in \mathbb{C}^n$,
        set $r^0 = b - Ax^0$,
        $q^0 = p^0 = r^0$.
Iterations: for $k = 0, 1, \ldots$ until convergence do
```

$$\begin{aligned}
\alpha_k &= \mathrm{Re}\{(r^k, r^0)\}/\mathrm{Re}\{(Aq^k, r^0)\}, \\
u^k &= p^k - \alpha_k A q^k, \\
x^{k+1} &= x^k + \alpha_k (p^k + u^k), \\
r^{k+1} &= r^k - \alpha_k A (p^k + u^k), \\
\beta_{k+1} &= \mathrm{Re}\{(r^{k+1}, r^0)\}/\mathrm{Re}\}(r^k, r^0)\}, \\
p^{k+1} &= r^{k+1} + \beta_{k+1} u^k, \\
q^{k+1} &= p^{k+1} + \beta_{k+1}(u^k + \beta_{k+1} q^k).
\end{aligned}$$

The CPU cost for one iteration is almost the same as in the Biconjugate Gradient method, but multiplications by A^{H} are avoided, so BiCGS is easy to vectorize when vectorization of the matrix vector product is available. The potential problem with this method is that, if there are some oscillations in BiCG, they will be amplified in BiCGS. In the case of a symmetric complex matrix, this method has to converge twice as fast to remain efficient!

8. Numerical results

Some of the previous algorithms as well as some other variants have been tested on problems introduced in Freund [6] and Gutknecht [9].

In the following we write

- Normal Equation for the Normal Equation method.
- Orthomin(m) for the Orthomin method with m directions [16].
- Gmres(m) for the General Minimal Residual method with m directions [14].
- Bicg(Jacobs) for Jacobs' version of the Biconjugate Gradient algorithm [10].
- Bicg(JM) for our version of the Biconjugate Gradient algorithm.
- TfBicg for the Transpose free Biconjugate Gradient algorithm proposed by Chan et al. [3].
- BiCgs(Jacobs) for Sonneveld's version of the Biconjugate Gradient Squared algorithm [15].
- BiCgs(JM) for our version of the Biconjugate Gradient Squared algorithm.

- BicgStab1 and BicgStab2 for the two Stabilized Biconjugate Gradient algorithms of Gutknecht [9] (the first one is equivalent to Van der Vorst's algorithm if applied to real data).
- And finally Gauss stands for the classical direct solution method, that we use to compare the computational cost of direct and iterative methods (table 2 summarizes the memory requirements for each numerical example).

All algorithms are initialized with $x^0 = 0$, and the convergence criterion is $\|r^k\| < 10^{-8}\|r^0\|$. A Hewlett-Packard/Apollo DN10000 workstation is used for the computations in complex double precision. For each problem, both real and imaginary parts of the right-hand side components are chosen randomly in $[-1, 1]$ (the same for all methods). All the algorithms are used without preconditioning.

The first three problems correspond to banded Toeplitz matrices, and are borrowed from Gutknecht [9].

EXAMPLE 1

Let us first consider the tridiagonal matrix

$$\begin{bmatrix} 4 & -2 & & \\ 1 & 4 & -2 & \\ & 1 & 4 & \ddots \\ & & \ddots & \ddots \end{bmatrix}.$$

We set the number of unknowns to the value $N = 10{,}000$, because of the rather fast convergence of all the methods for solving this problem. Results are displayed in table 3. For each algorithm we give the number of iterations, the number of matrix vector products, and the total computing time.

The convergence is rather fast regarding the number of unknowns, this may be related to the good condition number of this matrix, and the fact that the matrix is real. The algorithms gather by family, and our methods converge within almost the same number of iterations as the classical corresponding algorithms.

EXAMPLE 2

Let us consider now another real banded matrix, again of dimension 10,000:

Table 2
Memory size for the four problems.

Problem	N	Nz	Lm
1	10,000	19,998	19,998
2	10,000	19,997	39,994
3	10,000	29,997	59,994
4	1,089	3,968	65,534

Table 3
Convergence results for problem 1.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	14	28	3.03
Orthomin(5)	23	23	7.67
Orthomin(10)	23	23	10.99
Orthomin(20)	23	23	14.52
Orthomin(50)	23	23	14.21
Gmres(5)	5	30	5.12
Gmres(10)	3	33	7.94
Gmres(20)	2	42	15.89
Gmres(50)	2	102	82.54
Bicg (Jacobs)	24	48	6.79
Bicg (JM)	24	48	5.94
TfBicg	25	75	7.54
BiCgs (Jacobs)	13	26	2.82
BiCgs (JM)	13	26	2.87
BiCgStab1	13	26	3.07
BiCgStab2	13	26	4.42

$$\begin{bmatrix} 2 & 1 & & & \\ 0 & 2 & 1 & & \\ 1 & 0 & 2 & 1 & \\ & 1 & 0 & 2 & \ddots \\ & & \ddots & \ddots & \ddots \end{bmatrix}.$$

Results are displayed in table 4. Again convergence is easily obtained.

The algorithms also gather by family, and our methods converge within the same number of iterations as the classical corresponding algorithms.

EXAMPLE 3

Let us consider now a complex banded matrix, of rank 10,000:

$$\begin{bmatrix} 4 & 0 & 1 & 0.7 & & \\ 2i & 4 & 0 & 1 & 0.7 & \\ & 2i & 4 & 0 & 1 & \ddots \\ & & 2i & 4 & 0 & \ddots \\ & & & 2i & 4 & \ddots \\ & & & & \ddots & \ddots \end{bmatrix}.$$

Table 4
Convergence results for problem 2.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	26	52	5.60
Orthomin(5)	47	47	20.57
Orthomin(10)	43	43	23.83
Orthomin(20)	41	41	38.38
Orthomin(50)	41	41	187.46
Gmres(5)	9	54	9.17
Gmres(10)	5	55	13.21
Gmres(20)	3	63	23.71
Gmres(50)	2	102	90.20
Bicg (Jacobs)	44	88	11.18
Bicg (JM)	45	90	10.93
TfBicg	47	141	14.80
BiCgs (Jacobs)	23	46	4.95
BiCgs (JM)	23	46	5.02
BiCgStab1	25	50	5.83
BiCgStab2	23	46	7.90

Results are displayed in table 5. The results are the same as in the previous examples, despite the complex character of the problem.

Now we can observe a little difference due to the complex coefficients of the matrix, but our methods remain in the fastest group.

EXAMPLE 4

The last examples arises from PDE approximation, and is devoted to solving the linear system using the five point finite difference approximation of Helmholtz's problem in a squared domain Ω, with first order radiation boundary condition:

$$\Delta u + k^2 u = f \quad \text{in } \Omega,$$
$$\frac{\partial u}{\partial n} - \mathrm{i}ku = g \quad \text{on } \partial\Omega,$$

where n is the normal to the boundary $\partial\Omega, \mathrm{i}^2 = -1$, and $k = 2\pi/\lambda, \lambda = \alpha h$ is the wave length, α is a real constant related to the physical data, and $h = 1/32$ is the mesh size. There are 1089 unknowns, and 3968 non-zero elements in the symmetric complex matrix A.

We choose different values of the parameter α, in order to modify the condition number of the matrix A. Results are summarized in table 6 for the value $\alpha = 10$. Note that the property of complex symmetry of the matrix A is explicitly used in the two Bicg algorithms, so there is only one matrix vector product per iteration.

Table 5
Convergence results for problem 3.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	67	134	16.82
Orthomin(5)	43	43	20.12
Orthomin(10)	41	41	25.81
Orthomin(20)	40	40	42.24
Orthomin(50)	40	40	206.16
Gmres(5)	9	54	9.95
Gmres(10)	5	55	13.97
Gmres(20)	2	42	16.48
Gmres(50)	2	102	90.94
Bicg (Jacobs)	45	90	13.23
Bicg (JM)	61	122	17.22
TfBicg	43	126	16.01
BiCgs (Jacobs)	22	44	5.36
BiCgs (JM)	35	70	8.50
BiCgStab1	25	50	6.56
BiCgStab2	22	44	8.26

Table 6
Convergence results for problem 4 with $k = 2\pi/2h$.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	52	104	2.90
Orthomin(5)	NC	–	–
Orthomin(10)	64	64	9.12
Orthomin(20)	54	54	11.88
Orthomin(50)	45	45	12.93
Gmres(5)	168	1008	48.56
Gmres(10)	13	143	9.47
Gmres(20)	4	84	8.55
Gmres(50)	2	102	21.22
Bicg (Jacobs)	51	51	1.76
Bicg (JM)	53	53	1.84
TfBicg	51	153	4.47
SymBicg	30	60	1.76
BiCgs (Jacobs)	30	60	1.90
BiCgs (JM)	30	60	1.88
BiCgStab1	108	216	6.98
BiCgStab2	45	90	4.02
Gauss	1	–	1.17

Then Jacobs' version of the Biconjugate Gradient algorithm is equivalent to the Symmetric Biconjugate algorithm of Freund [6].

We can observe that this problem is rather stiff. Both Orthomin and Gmres fail to converge within N iterations ($N = 1089$), even the stabilized form of the Cgs algorithm converges rather poorly.

But this is no longer true when the value of the parameter k decreases, as shown by tables 7 to 11, corresponding respectively to $k = 2\pi/5$, $k = 2\pi/10h$, $k = 2\pi/50h$, $k = 2\pi/100h$ and $k = 0$.

Note that for $k = 0$, the matrix A is singular. Our Bicg method converges then within the same number of iterations as TfBicg as k vanishes, but runs faster, because of the operational cost.

Furthermore, we see that both our versions of Bicg and BiCgs are competitive in many situations, especially for small values of k, where they run very near the Gauss method (direct method), which needs 65,534 terms in a skyline storage of the matrix A, instead of the 3968 required by a condensed storage scheme (see table 2).

Table 2 summarizes the calculation conditions for all the previous problems.

- N is the number of unknowns.
- Nz is the number of non-zero coefficients in the matrix A.
- Lm is the profile length, that is, the memory required to solve the linear system by the Gauss method.

Table 7
Convergence results for problem 4 with $k = 2\pi/5h$.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	NC	–	–
Orthomin(5)	NC	–	–
Orthomin(10)	NC	–	–
Orthomin(20)	NC	–	–
Orthomin(50)	NC	–	–
Gmres(5)	NC	–	–
Gmres(10)	NC	–	–
Gmres(20)	NC	–	–
Gmres(50)	NC	–	–
Bicg (Jacobs)	831	831	28.58
Bicg (JM)	1582	1582	54.39
TfBicg	1211	3633	105.62
SymBicg	831	831	28.58
BiCgs (Jacobs)	912	1824	56.98
BiCgs (JM)	NC	–	–
BiCgStab1	NC	–	–
BiCgStab2	NC	–	–
Gauss	1	–	1.17

Table 8
Convergence results for problem 4 with $k = 2\pi/10h$.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	NC	–	–
Orthomin(5)	NC	–	–
Orthomin(10)	NC	–	–
Orthomin(20)	NC	–	–
Orthomin(50)	NC	–	–
Gmres(5)	NC	–	–
Gmres(10)	NC	–	–
Gmres(20)	NC	–	–
Gmres(50)	NC	–	–
Bicg (Jacobs)	356	356	5.26
Bicg (JM)	628	628	10.00
TfBicg	498	1494	19.62
SymBicg	356	356	5.50
BiCgs (Jacobs)	359	718	9.82
BiCgs (JM)	456	912	12.71
BiCgStab1	NC	–	–
BiCgStab2	1219	2438	50.66
Gauss	1	–	1.17

Table 9
Convergence results for problem 4 with $k = 2\pi/50h$.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	1632	3264	137.92
Orthomin(5)	NC	–	–
Orthomin(10)	NC	–	–
Orthomin(20)	NC	–	–
Orthomin(50)	519	519	300.26
Gmres(5)	NC	–	–
Gmres(10)	NC	–	–
Gmres(20)	792	16,632	2036.18
Gmres(50)	134	6834	1833.24
Bicg (Jacobs)	184	184	9.81
Bicg (JM)	187	187	9.68
TfBicg	181	543	24.34
SymBicg	184	184	9.52
BiCgs (Jacobs)	146	292	14.26
BiCgs (JM)	147	294	13.70
BiCgStab1	172	374	18.46
BiCgStab2	159	318	21.90
Gauss	1	–	1.17

Table 10
Convergence results for problem 4 with $k = 2\pi/100h$.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	1569	3138	44.63
Orthomin(5)	NC	–	–
Orthomin(10)	NC	–	–
Orthomin(20)	NC	–	–
Orthomin(50)	382	382	84.48
Gmres(5)	NC	–	–
Gmres(10)	NC	–	–
Gmres(20)	36	756	28.94
Gmres(50)	21	1071	82.49
Bicg (Jacobs)	168	168	2.49
Bicg (JM)	171	171	2.61
TfBicg	166	498	6.48
SymBicg	168	168	2.58
BiCgs (Jacobs)	135	270	3.70
BiCgs (JM)	136	272	3.85
BiCgStab1	134	268	3.95
BiCgStab2	123	246	4.99
Gauss	1	–	1.17

Table 11
Convergence results for problem 4 with $k = 0$.

Algorithm	Number of iterations	Number of products matrix × vector	Time (seconds)
Normal Equation	826	1652	23.50
Orthomin(5)	129	129	4.92
Orthomin(10)	129	129	7.48
Orthomin(20)	129	129	12.35
Orthomin(50)	129	129	24.77
Gmres(5)	161	966	18.38
Gmres(10)	44	484	12.36
Gmres(20)	14	294	11.26
Gmres(50)	4	204	15.74
Bicg (Jacobs)	140	140	2.07
Bicg (JM)	142	142	2.12
TfBicg	132	396	5.06
SymBicg	140	140	2.08
BiCgs (Jacobs)	91	182	2.49
BiCgs (JM)	91	182	2.50
BiCgStab1	108	216	3.10
BiCgStab2	105	210	4.21
Gauss	1	–	1.17

9. Conclusion

Our methods compare favorably with the other ones used here, and are, in fact, some of the best methods according to the CPU time cost in both versions: basic and accelerated Bicg.

In more realistic electromagnetic scattering problems, we will have to solve complex linear systems with a large number of unknowns as the mesh size must be related to the wave length of the incoming wave, particularly for 3D-problems. In these situations, direct method solvers are no longer practical, because of the huge memory they need. Iterative methods are the only way to solve these problems, and the algorithms considered in this paper seem good candidates for doing so, particularly on parallel computers.

Appendix

DERIVATION OF THE BICONJUGATE GRADIENT METHOD ACCELERATION

The residuals and the directions are linked by the relations

$$r_1^k = \phi_k(A)r^0, \quad d_1^k = \theta_k(A)r^0,$$
$$r_2^k = \phi_k(A^{\mathrm{H}})r^0, \quad d_2^k = \theta_k(A^{\mathrm{H}})r^0,$$

where $\phi_k(z)$ and $\theta_k(z)$ are complex polynomials of degree at most k in the variable z, and satisfying

$$\phi_{k+1}(z) = \phi_k(z) - \alpha_k z\theta_k(z),$$
$$\theta_{k+1}(z) = \phi_{k+1}(z) + \beta_{k+1}\theta_k(z).$$

From the previous equations, it follows that

$$\phi_{k+1}^2(z) = \phi_k^2(z) - 2\alpha_k z\phi_k(z)\theta_k(z) + (\alpha_k)^2 z^2\phi_k^2(z),$$
$$\theta_{k+1}^2(z) = \phi_{k+1}^2(z) + 2\beta_{k+1}\phi_{k+1}(z)\theta_k(z) + (\beta_{k+1})^2\theta_k^2(z).$$

Define now

$$\tilde{r}^k = \phi_k^2(A)r^0,$$
$$p^k = \phi_k(A)\theta_k(A)r^0,$$
$$q^k = \theta_k^2(A)r^0;$$

these vectors satisfy

$$\tilde{r}^{k+1} = \tilde{r}^k - 2\alpha_k A p^k + (\alpha_k)^2 A^2 q^k ,$$
$$q^{k+1} = \tilde{r}^{k+1} + 2\beta_{k+1}(p^k - \alpha_k A q^k) + (\beta_{k+1})^2 q^k ,$$
$$p^{k+1} = q^{k+1} + \beta_{k+1}(p^k - \alpha_k A q^k) .$$

Define now $\tilde{x}^k$, and α_k, β_{k+1}. From

$$\tilde{r}^{k+1} = \tilde{r}^k - \alpha_k A(p^k + (p^k - \alpha_k A q^k))$$

we get

$$\tilde{x}^{k+1} = \tilde{x}^k - \alpha_k (p^k + (p^k - \alpha_k A q^k))$$

and

$$\alpha_k = \mathrm{Re}\{(r_1^k, r_2^k)\}/\mathrm{Re}\{(d_2^k, A d_1^k)\} ,$$
$$\beta_{k+1} = \mathrm{Re}\{(r_1^{k+1}, r_2^{k+1})\}/\mathrm{Re}\{(r_1^k, r_2^k)\} ,$$

so

$$(r_1^k, r_2^k) = (\phi_k(A) r^0, \phi_k(A^{\mathrm{H}}) r^0)$$
$$= (\phi_k^2(A) r^0, r^0) = (\tilde{r}^k, r^0) ,$$

$$(d_2^k, A d_1^k) = (\theta_k(A^{\mathrm{H}}) r^0, A\theta_k(A) r^0)$$
$$= (r^0, A\theta_k^2(A) r^0) = (r^0, A q^k) ,$$

$$\alpha_k = \mathrm{Re}\{(\tilde{r}^k, \tilde{r}^0)\}/\mathrm{Re}\{(\tilde{r}^0, A q^k)\} ,$$
$$\beta_{k+1} = \mathrm{Re}\{(\tilde{r}^{k+1}, \tilde{r}^0)\}/\mathrm{Re}\{(\tilde{r}^k, \tilde{r}^0)\} .$$

JACOBS' BICONJUGATE GRADIENT METHOD

In [10] an algorithm is proposed using the orthogonality of the residuals r_1^k and r_2^l. This algorithm is written as:

Initialization

```
    Choose $x_1^0, x_2^0 \in \mathbb{C}^n$,
    set $r_1^0 = b - A x_1^0$,
    $r_2^0 = \bar{r}_1^0$,
    $d_1^0 = r_1^0$,
    $d_2^0 = r_2^0$.
```

Iterations: for $k = 0, 1, \ldots$ until convergence do

(1) Compute α_k by

$$\begin{aligned}
\alpha_k &= (r_1^k, r_2^k)/(Ad_1^k, d_2^k)\,,\\
x^{k+1} &= x^k + \alpha_k d_1^k\,,\\
r_1^{k+1} &= r_1^k - \alpha_k A d_1^k\,,\\
r_2^{k+1} &= r_2^k - \bar{\alpha}_k A^{\mathrm{H}} d_2^k\,.
\end{aligned}$$

(2) Generate the new direction by

$$\begin{aligned}
\beta_{k+1} &= (r_1^{k+1}, r_2^{k+1})/(r_1^k, r_2^k)\,,\\
d_1^{k+1} &= r_1^{k+1} + \beta_{k+1} d_1^k\,,\\
d_2^{k+1} &= r_2^{k+1} + \bar{\alpha}_{k+1} d_2^k\,.
\end{aligned}$$

The CPU cost of this algorithm is exactly the same as that of the one in section 5.

Remark

It is possible to obtain this algorithm from the general formulation, but $(\cdot,\cdot)$ must be changed to the real scalar product, and the matrices H and K are then defined as

$$H = \begin{bmatrix} 0 & A^{\mathrm{T}} \\ A & 0 \end{bmatrix}^{-1}, \quad K = \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}.$$

The following relations are obtained by induction:

$$(r_1^k, r_2^l) = 0, \qquad \forall k \neq l\,, \tag{A.1}$$

$$(Ad_1^l, d_2^k) = (d_1^k, A^{\mathrm{H}} d_2^l) = 0, \qquad \forall k \neq l\,, \tag{A.2}$$

$$(r_1^k, d_2^l) = (r_2^k, d_1^l) = 0, \qquad \forall k \neq l\,. \tag{A.3}$$

Furthermore

$$\begin{aligned}
\alpha_k &= (r_1^k, r_2^k)/(Ad_1^k, d_2^k)\\
&= (d_1^k, r_2^k)/(Ad_1^k, d_2^k)\\
&= (r_1^k, d_2^k)/(Ad_1^k, d_2^k)\,,\\
\beta_{k+1} &= -(r_1^{k+1}, A^{\mathrm{H}} d_2^k)/(Ad_1^k, d_2^k)\\
&= -(Ad_1^k, r_2^{k+1})/(Ad_1^k, d_2^k)\\
&= (r_1^{k+1}, r_2^{k+1})/(r_1^k, r_2^k)\,.
\end{aligned}$$

This algorithm may also be accelerated, as in section 5:

```
Initialization
        Choose $x^0 \in \mathbb{C}^n$,
        set $r^0 = b - Ax^0$,
        $q^0 = p^0 = r^0$,
        $\tilde{r}^0 = r^0$ or some other choice $\neq 0$.
Iterations: for $k = 0, 1, \ldots$ until convergence do
        $\alpha_k = (\tilde{r}^0, r^k)/(\tilde{r}^0, Aq^k)$,
        $u^k = p^k - \alpha_k A q^k$,
        $x^{k+1} = x^k + \alpha_k (p^k + u^k)$,
        $r^{k+1} = r^k - \alpha_k A(p^k + u^k)$,
        $\beta_{k+1} = (\tilde{r}^0, r^{k+1})/(\tilde{r}^0, r^k)$,
        $p^{k+1} = r^{k+1} + \beta_{k+1} u^k$,
        $q^{k+1} = p^{k+1} + \beta_{k+1}(u^k + \beta_{k+1} q^k)$.
```

If A is symmetric complex, then

$$r_2^k = \bar{r}_1^k, d_2^k - \bar{d}_1^k, \quad \forall k,$$

and the algorithm can be rewritten in

```
Initialization
        Choose $x^0 \in \mathbb{C}^n$,
        set $r^0 = b - Ax^0$,
        $d^0 = r^0$.
Iterations: for $k = 0, 1, \ldots$ until convergence do
        $\alpha_k = (r^k, \bar{r}^k)/(Ad^k, \bar{d}^k)$,
        $x^{k+1} = x^k + \alpha_k d^k$,
        $r^{k+1} = r^k - \alpha_k A d^k$,
        $\beta_{k+1} = (r^{k+1}, \bar{r}^{k+1})/(r^k, \bar{r}^k)$,
        $d^{k+1} = r^{k+1} + \beta_{k+1} d^k$.
```

References

[1] S.F. Ashby, T.A. Manteufel and P.E. Saylor, A taxonomy for conjugate gradient methods, SIAM J. Numer. Anal. 27 (1990).

[2] G.A. Behie and P.A. Forsyth Jr., Incomplete factorization methods for fully implicit simulation of enhanced oil recovery, SIAM J. Sci. Stat. Comput. 5 (1984).

[3] T.F. Chan, L. de Pillis and H. van der Vorst, A transpose-free squared Lanczos algorithm and application to solving nonsymmetric linear systems, UCLA Research report (1991).

[4] S.C. Eisenstat, H.C. Elman and M.H. Schultz, Variational iteration methods for nonsymmetric systems of linear equations, SIAM J. Numer. Anal. 20 (1983).
[5] R. Fletcher, Conjugate gradient methods for indefinite systems, *Proc. Dundee Conf. on Numerical Analysis* (Springer, 1976).
[6] R. Freund, On Conjugate Gradient type methods and polynomial preconditioners for a class of complex non-Hermitian matrices, Numer. Math. 57 (1990).
[7] G. Golub and G. Meurant, *Résolution Numérique des Grands Systèmes Linéaires* (Eyrolles, Paris, 1981).
[8] M.H. Gutknecht, Changing the norm in conjugate gradient type algorithms, ETH IPS Research report.
[9] M.H. Gutknecht, Variants of BICGSTAB for matrices with complex spectrum, ETH IPS Research report 91–14.
[10] D.A.H. Jacobs, A generalization of the Conjugate-Gradient method to solve complex systems, IMA J. Numer. Anal. 6 (1986).
[11] P. Joly, Méthodes de gradient conjugué, Publications du Laboratoire d'Analyse Numérique Université Pierre et Marie Curie, Paris (1984).
[12] D.G. Luenberger, *Introduction to Linear and Nonlinear Programming* (Addison–Wesley, 1973).
[13] N.M. Nachtigal, S.C. Reddy and L.N. Trefethen, How fast are nonsymmetric matrix iterations, Numerical Analysis Report 90-2, Massachusetts Institute of Technology (1990).
[14] Y. Saad and M.H. Schultz, Gmres, a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986).
[15] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 10 (1989).
[16] P.K.W. Vinsome, Orthomin: an iterative method for solving sparse sets of simultaneous linear equations, *Proc. 4th Symp. on Reservoir Simulation* (1976).