Complex conjugate gradient method

P. Joly
C.N.R.S. Laboratoire d'Analyse Numérique
Université Pierre et Marie Curie

G. Meurant
C.E.A. Centre d'Etudes de Limeil-Valenton

1 Introduction

Complex linear systems arise from many various problems, such as the Helmholtz equation approximated by finite differences or finite elements method, or also by integral equations method; the resulting matrix is generally non-Hermitian, and consequently the classical conjugate gradient method is no more usefull. However the increasing number of unknowns (specially for 3D-problems) leads to give up the direct methods solvers, because of the giant memory core they need. So iterative methods are still attractive, even if their convergence is not always guaranteed. This paper is devoted to the study of complex conjugate gradient methods, and is a generalisation of a previous paper related to the real case [Joly 84]. (See also a recent publication [Ashby, Manteuffel, Saylor 90] for a complete review of those methods).

2 A general algorithm

To solve the linear system $Ax = b$, where $A \in \mathcal{C}^{n\times n}$ is a regular matrix and $b \in \mathcal{C}^n$, we introduce the function $J : \mathcal{C}^n \mapsto \mathbb{R}^+$, $\forall r \in \mathcal{C}^n \quad J(r) = (r, Hr)$, with $(\cdot,\cdot)$ the usual complex scalar product $\mathcal{C}^n$: $(x, y) = \displaystyle\sum_{i=1}^{n} x_i \overline{y}_i$ and $H \in \mathcal{C}^{n\times n}$ a definite matrix. In the following, r stands for the residual $b - Ax$. As J is a strictly convex function on $\mathcal{C}^n$, it admits an unique minimum in $r_m \in \mathcal{C}^n$. From the property $J(r) \geq 0 \quad \forall r \in \mathcal{C}^n$, it follows that $r_m = 0$; the corresponding vector $x_m = A^{-1}(b - r_m)$ is then solution of the problem (P).

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

A general minimization algorithm of the function J can be then generated as

- choose x^0 in $\mathcal{C}^n$, then generate vectors $\{d^0, d^1, \ldots, d^k\}$ in $\mathcal{C}^n$ orthogonal in the scalar product related to the Hermitian matrix $N = A^H \times H \times A$
- minimize J over each subspace $x^0 + < Kg^0, Kg^1, \ldots, Kg^k >$ with g^k gradient of J in x^k

The general algorithm is

Initialization

$$\begin{aligned}
\text{Choose } x^0 &\in \mathcal{C}^n \\
\text{set } r^0 &= b - Ax^0 \\
g^0 &= A^H H r^0 \\
d^0 &= K g^0
\end{aligned}$$

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

1) minimize J over $x^0 + < d^0, d^1, \ldots, d^k >$

$$\begin{aligned}
\alpha^k &= (g^k, d^k)/(d^k, N d^k) \\
x^{k+1} &= x^k + \alpha^k d^k \\
r^{k+1} &= r^k - \alpha^k A d^k \\
g^{k+1} &= g^k - \alpha^k N d^k
\end{aligned}$$

2) generate the new direction

$$\begin{aligned}
d^{k+1} &= K g^{k+1} + \sum_{l=0}^{k} \beta_l^{k+1} d^l \\
\beta_l^{k+1} &= -(K g^{k+1}, N d^l)/(d^l, N d^l) \quad 0 \le l \le k
\end{aligned}$$

Remark

This algorithm is formally equivalent to the classical conjugate gradient method, but the function J to minimize, and the directions generation are abstracted.

- **Some properties of the complex algorithm**

1) $$(d^k, N d^l) = 0 \qquad \forall k \neq l$$

2) $$(g^k, d^l) = 0 \qquad 0 \le l < k$$

3) $$(g^l, d^k) = (g^0, d^k) \qquad 0 \le l \le k$$

4) $$(g^k, d^k) = (g^k, K g^k)$$

5) $$(K g^k, N d^k) = (d^k, N d^k)$$

6) $$(g^k, K g^l) = 0 \qquad 0 \le l < k$$

7) $$\begin{aligned}
E^k &= < d^0, d^1, \ldots, d^k > \\
&= < K g^0, K g^1, \ldots, K g^k > \\
&= < d^0, K N d^0, \ldots, (KN)^k d^0 >
\end{aligned}$$

8) $$\text{dimension } E^k = k + 1 \text{ if } g^k \neq 0$$

9) x^{k+1} realize the minimum of J over $x^0 + E^k$

10) $g^n = 0$

11) if K is Hermitian $\beta_l^{k+1} = 0 \qquad 0 \le l < k$

and $\beta_k^{k+1} = (g^{k+1}, Kg^{k+1})/(g^k, Kg^k)$

Demonstration : 1 to 6) are obtained by induction:

1) $(d^1, Nd^0) = (Kg^1 + \beta^1 d^0, Nd^0) = 0$ by definition of β_0^1

2) $(g^1, d^0) = (g^0 - \alpha^0 Nd^0, d^0) = 0$ by definition of α^0

3) $(g^1, d^1) = (g^0 - \alpha^0 Nd^0, d^1) = (g^0, d^1)$ by 1)

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

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

6) $(g^1, Kg^0) = (g^0 - \alpha^0 Nd^0, Kg^0) = (g^0, Kg^0) - \alpha^0(Nd^0, Kg^0)$
$= (g^0, d^0) - \alpha^0(Nd^0, d^0) = 0$ by definition of α^0

Suppose these properties satisfied untill $k-1$, then

1) $(d^k, Nd^l) = (Kg^{k-1} + \sum_{i=1}^{k-1} \beta_i^{k-1} d^i, Nd^l) = (Kg^{k-1}, Nd^l) + \beta_l^{k-1}(d^l, Nd^l) = 0$

for $0 \le l < k$ by definition of β_i^{k-1}; 1) follows because N is Hermitian.

2) $(g^k, d^l) = (g^l - \sum_{i=l}^{k-1} \alpha^i Nd^i, d^l) = (g^l, d^l) - \alpha^l(d^l, Nd^l) = 0$ for $0 \le l < k$

from 1) and the definition of α^l

3) $(g^l, d^k) = (g^0 - \sum_{i=0}^{l-1} \alpha^i Nd^i, d^k) = (g^0, d^k)$ for $0 \le l \le k$ from 1)

4) $(g^k, d^k) = (g^k, Kg^k + \sum_{i=1}^{k-1} \beta_i^{k-1} d^i) = (g^k, Kg^k)$ from 2)

5) $(Kg^k, Nd^k) = (d^k - \sum_{i=1}^{k-1} \beta_i^{k-1} d^i, Nd^k) = (d^k, Nd^k)$ from 1)

6) $(g^k, Kg^l) = (g^k, d^l - \sum_{i=1}^{l-1} \beta_i^{l-1} d^i) = 0$ for $0 \le l < k$ from 2)

So properties 1) to 6) are satisfied for all k.

7) et 8) This is obviously true for $k = 1$, suppose it true untill $k-1$ included. Then

by definition $Kg^k = Kg^{k-1} - \alpha^{k-1} KNd^{k-1}$

but $Kg^{k-1} \in < d^0, KNd^0, \ldots, (KN)^{k-1} d^0 >$

and $d^{k-1} \in < d^0, KNd^0, \ldots, (KN)^{k-1} d^0 >$

so $Kg^k \in < d^0, KNd^0, \ldots, (KN)^k d^0 >$

From another hand $\quad d^k = Kg^k + \sum_{l=0}^{k-1} \beta_l^k d^l$

but $\quad d^l \in < Kg^0, Kg^1, \ldots, Kg^{k-1} > \quad 0 \leq l < k$

so $\quad d^k \in < Kg^0, Kg^1, \ldots, Kg^k >$

Combining these results with the induction hypothesis, we obtain :

$$< d^0, d^1, \ldots, d^k > \subset < Kg^0, Kg^1, \ldots, Kg^k > \subset < d^0, KNd^0, \ldots, (KN)^k d^0 >$$

But from 1) dimension $< d^0, d^1, \ldots, d^k >= k+1$ and finally

$$< d^0, d^1, \ldots, d^k >=< Kg^0, Kg^1, \ldots, Kg^k >=< d^0, KNd^0, \ldots, (KN)^k d^0 >$$

The case $g^k = 0$ will be examined later
9)

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

Now using 1), the relation $g^l = A^H H r^l$ and 3) , we obtain

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

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

Define now $J_l(\alpha) = -(\alpha^l \overline{(g^l, d^l)} + \overline{\alpha}^l (g^l, d^l) + \alpha^l \overline{\alpha}^l (d^l, Nd^l) \quad (0 \leq l \leq k)$, where $\alpha \in \mathcal{C}$ The minimum of J_l is realized with $\alpha^l = (g^l, d^l)/(d^l, Nd^l)$. Finally, from the calculation of $\alpha^l \quad 0 \leq l \leq k$, at each step of the algorithm x^{k+1} realize the minimum of J over the subspace $x^0 + E^k$.

10) 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 the dimension n, and x^n realize the minimum of J, that is $J(r^n) = 0 = (r^n, Hr^n)$, hence $r^n = 0$!

Remark
This result comes directly from 6)

11) Suppose now that the matrix K is Hermitian, then

$$\begin{aligned}(Kg^{k+1}, Nd^l) &= (Kg^{k+1}, \frac{1}{\alpha^l}(g^l - g^{l+1})) \\ &= \frac{1}{\alpha^l}(g^{k+1}, K(g^l - g^{l+1})) \\ &= 0 \quad \text{pour } 0 \le l \le k\end{aligned}$$

and $\beta_l^{k+1} = 0$ pour $0 \le l < k$.

At least $\alpha^k = (g^k, d^k)/(d^k, Nd^k) = (g^k, Kg^k)/(d^k, Nd^k)$, where

$$\begin{aligned}\beta_k^{k+1} &= -(Kg^{k+1}, Nd^k)/(d^k, Nd^k) \\ &= -(Kg^{k+1}, \frac{1}{\alpha^k}(g^k - g^{k+1}))/(d^k, Nd^k) \\ &= \frac{1}{\alpha^k}(Kg^{k+1}, g^{k+1})/(d^k, Nd^k) \\ &= (g^{k+1}, Kg^{k+1})/(g^k, Kg^k)\end{aligned}$$

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

Remark
No breakdown can occur : $\alpha^k = 0$ leads to $(g^k, Kg^k) = 0$
that is $g^k = 0$, because matrix K is definite.
Similarly $(d^k, Nd^k) = 0$ leads to $d^k = 0$, that is $g^k = 0$ from 4).

- **Convergence**

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 monotonous, in case of equality $J(r^{k+1}) = J(r^k)$ then $(g^k, Kg^k) = 0$, and $g^k = 0$. Furthermore as $J(r^k) = (r^k, Hr^k) = (g^k, N^{-1}g^k)$:

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

Suppose now that the matrix K is Hermitian, we obtain

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

and then

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

we introduce $L:\ K = L^H \times L$, $M = L^H \times N \times L$ and $h^k = Lg^k$, we obtain

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

now using the Kantorovitch's inequality

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

Remark
Following [Golub, Meurant 81], and with the Tchebyshev's polynomials properties

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

Many algorithms can be generated by an appropriate choice of matrices H and K. For example the choice $H = A^{-1}$ and $K = I$ leads to the classical complex conjugate gradient method; with $H = A^{-1}$ and $K = l^H \times l$ (incomplete complex Cholesky factorization), we obtain the complex conjugate gradient method preconditioned by the matrix K. The next table resume the most popular methods. The convergence rate is related to the condition number of the matrix M similar to the matrix $K \times N$

Algorithm	H	K	M	Convergence
Conjugate Gradient	A^{-1}	I	A	A herm.
Prec. Conjugate Gradient	A^{-1}	$l^{-H} \times l^{-1}$	$l^{-H} \times A \times l^{-1}$	A herm.
Gen. Conjugate Residual	I	A^{-H}	$(AL)^H \times (AL)^{(1)}$	A def.
Orthomin	I	A^{-H}	$(AL)^H \times (AL)^{(1)}$	A def.
Gmres	I	A^{-H}	$(AL)^H \times (AL)^{(1)}$	A def.
Normal Equation	I	I	$A^H \times A$	A reg.
Minimal Error	$(A \times A^H)^{-1}$	$A^H \times A$	$A^H \times A$	A reg.

(1) with $L \times L^H = (K + K^H)/2$

It is not the aim of this paper to study all the possibilities of the general algorithm, so we limit ourselves to the most interesting ones : the normal equation method (see for example [EISENSTAT, ELMAN, SCHULTZ 83], the complex Orthomin method (see [VINSOME 76]), the complex Gmres algorithm (see [SAAD, SCHULTZ 86]), the Biconjugate Gradient method ([FLETCHER 76]) and the accelerated variant Conjugate Gradient Squared ([SONNEVELD 89]).

3 The normal equation method

The normal equation method is very popular to solve non-Hermitian linear systems . It follows from the choice $H = I$, $K = I$ and minimize the function $J(r) = (r, r)$. The algorithm can be written as

Initialization

$$\begin{aligned} Choose \quad & x^0 \in \mathcal{C}^n \\ set \quad & r^0 = b - Ax^0 \\ & d^0 = r^0 \end{aligned}$$

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

1) minimize J over $x^0 + < d^0, d^1, \ldots, d^k >$

$$\begin{aligned} \alpha^k &= (A^H r^k, A^H r^k)/(Ad^k, Ad^k) \\ x^{k+1} &= x^k + \alpha^k d^k \\ r^{k+1} &= r^k - \alpha^k A d^k \end{aligned}$$

2) generate the new direction

$$\begin{aligned} d^{k+1} &= A^H r^{k+1} + \beta^{k+1} d^k \\ \beta^{k+1} &= (A^H r^{k+1}, A^H r^{k+1})/(A^H r^k, A^H r^k) \end{aligned}$$

If we suppose A regular, this algorithm converges in at most n iterations, but two difficulties have to be overcome :

- it needs two products matrix $\times$ vector by iteration, that is time consuming.
- the convergence is related to the condition number of A to the square.

Remark

As the convergence rate of the normal equation method is governed by the singular values of the matrix A, rather than its eigenvalues, so it remains competitive towards Gmres or the Biconjugate Gradient method for a class of linear systems (see [NACHTIGAL, REDDY, TREFETHEN 90]).

4 Complex Orthomin

With $H = I$ and $K = A^{-H}$, we obtain the General Residual algorithm, which minimize $J(r) = (r, r)$. This method is summarized by

Initialisation

$$\begin{aligned} \text{Choose } x^0 &\in \mathcal{C}^n \\ \text{set } r^0 &= b - Ax^0 \\ d^0 &= r^0 \end{aligned}$$

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

1) minimize J over $x^0 + < d^0, d^1, \ldots, d^k >$

$$\begin{aligned} \alpha^k &= (r^k, Ad^k)/(Ad^k, Ad^k) \\ x^{k+1} &= x^k + \alpha^k d^k \\ r^{k+1} &= r^k - \alpha^k Ad^k \end{aligned}$$

2) generate the new direction

$$\begin{aligned} d^{k+1} &= r^{k+1} + \sum_{l=0}^{k} \beta_l^{k+1} d^l \\ \beta_l^{k+1} &= -(Ar^{k+1}, Ad^l)/(Ad^l, Ad^l) \end{aligned}$$

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

- all the generated directions d^l and products Ad^l, have to be stored $(1 \leq l \leq k)$, that makes this algorithm core consuming when the iterations number is too large
- two products matrix $\times$ vector have to be computed each step.

Vinsome first proposed to limit the directions number to a fixed value m set in advance, and depending on the memory core available. The new direction generation is then modified in :

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

Then a new set of vectors z^l such that $z^l = Ad^l$ is introduced.

Combining these two modifications leads to the well-known Orthomin formula

Initialization

$$\begin{aligned}
\text{Choose } x^0 &\in \mathcal{C}^n \\
\text{set } r^0 &= b - Ax^0 \\
d^0 &= r^0 \\
z^0 &= Ad^0
\end{aligned}$$

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

1) minimize J over $x^0 + < d^0, d^1, \ldots, d^k >$

$$\begin{aligned}
\alpha^k &= (r^k, z^k)/(z^k, z^k) \\
x^{k+1} &= x^k + \alpha^k d^k \\
r^{k+1} &= r^k - \alpha^k z^k
\end{aligned}$$

2) generate the new direction

$$\begin{aligned}
d^{k+1} &= r^{k+1} + \sum_{l=k-m+1}^{k} \beta_l^{k+1} d^l \\
z^{k+1} &= Ar^{k+1} + \sum_{l=k-m+1}^{k} \beta_l^{k+1} z^l \\
\beta_l^{k+1} &= -(Ar^{k+1}, z^l)/(z^l, z^l)
\end{aligned}$$

Now subsists just one product matrix $\times$ vector by iteration : Ar^{k+1}, and the number of stored vectors is $2m$.

- **Property of the Orthomin method**

1) $(Ad^k, Ad^l) = 0 \qquad k - m \leq l < k$

2) $(r^k, Ad^l) = 0 \qquad k - m \leq l < k$

3) $(r^l, Ad^k) = (r^{k-m}, Ad^k) \qquad k - m \leq l \leq k$

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

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

6) $(r^k, Ar^l) = 0 \qquad k - m \leq l < k$

7) $E^k =< d^{k-m}, d^{k-m+1}, \ldots, d^{k-1} >$

8) dimension $E^k = m$ if $r^k \neq 0$

9) x^{k+1} realize the minimum of J over $x^{k-m} + E^k$

10) $\|r^k\| \mapsto 0$ quand $k \mapsto +\infty$ if A is definite

Demonstration : properties 1) to 9) are obtained by induction as in paragraph 2. To demonstrate 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 finite number k such as $J(r^{k+1}) = J(r^k)$, then $g^k = 0$, and $r^k = 0$;
- otherwise, the serial $\left(J(r^k)\right)_{k\in N}$ is strictly decreasing, and lower bounded by 0. So it converges towards a finite limit J^∞,

Then we notice that

$$\begin{aligned}\|r^{k+1} - r^k\| &= \|r^{k+1}\| + \|r^k\| - (r^{k+1}, r^k) - (r^k, r^{k+1}) \\ &= \|r^{k+1}\| - \|r^k\| + 2\,|(r^k, Ar^k)|^2/(Ad^k, Ad^k) \\ &= \|r^k\| - \|r^{k+1}\| \\ &= J(r^k) - J(r^{k+1})\end{aligned}$$

Then the serial $\left(r^k\right)_{k\in N}$ converges towards a finite limit r^∞. Furthermore from the relation

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

we deduce the following inequality $\|Ad^k\| \leq \|Ar^k\|$.

All these results show that $|(r^k, Ar^k)| \mapsto 0$ quand $k \mapsto \infty$, and finally $(r^\infty, Ar^\infty) = 0$, that is $r^\infty = 0$. So the restrained variant of the general residual algorithm is an iterative method! This drawback of the method is not really important because n iterations are not needed in a realistic computation, for which an approximation of the solution is requiered, according to the criterion $\|r^k\| < \varepsilon\,\|r^0\|$. This method is used successfully for a large class of rather well-conditionned problems.

Remark

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 referenced as GCR(m), Generalized Conjugate Residual algorithm with m directions, and is not equivalent to Orthomin(m).

5 Complex Gmres

To the vector storage of the previous algorithm, Saad and Schultz ([SAAD, SCHULTZ 86]) have proposed another modification, orthogonalizing the directions $d^0, d^1, \ldots, d^k$ in the natural scalar product of C^n, so the storage of vectors $Ad^0, Ad^1, \ldots, Ad^k$ is avoided. But then the coefficients α^l have to be computed as the solution of a least-squared problem.

Initialization

$$\begin{aligned} \text{Choose } x^0 &\in \mathcal{C}^n \\ \text{set } r^0 &= b - Ax^0 \\ d^0 &= r^0 / \|r^0\| \end{aligned}$$

Iterations :

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

$$\begin{aligned} \tilde{d}^{k+1} &= Ad^k + \sum_{l=0}^{k} \gamma_l^{k+1} d^l \\ \gamma_l^{k+1} &= -(Ad^k, d^l) \qquad 0 \le l < k \\ \gamma_k^{k+1} &= \|\tilde{d}^{k+1}\| \\ d^{k+1} &= \tilde{d}^{k+1} / \gamma_k^{k+1} \end{aligned}$$

2) minimize J over $x^0 + < d^0, d^1, \ldots, d^m >$

$$\begin{aligned} D_{m-1} &= (d^0, d^1, \ldots, d^{m-1}) \in \mathrm{C}^{n \times m} \\ \tilde{\Gamma}_m &= D_{m-1}^H \times A \times D_{m-1} \in \mathrm{C}^{m \times m} \\ \Gamma_m &= \begin{pmatrix} \tilde{\Gamma}_m \\ 0 \ldots 0 \ \gamma_{m-1}^m \end{pmatrix} \in \mathrm{C}^{m+1 \times m} \\ e^{m+1} &= (1, 0, \ldots, 0)^T \in \mathrm{C}^{m+1} \\ z^m \in \mathrm{C}^m &\text{ solution of } \min_{z \in \mathrm{C}^m} \|e^{m+1} - \Gamma_m z\| \\ x^m &= x^0 + D_m z^m \end{aligned}$$

6 The Biconjugate gradient method

In the previous methods, two drawbacks have been encountered:

- the use of an Hermitian matrix K may slow down the convergence, as in the normal equation method.
- on the other hand, when the matrix K is not Hermitian the vector storage leads to use approximated variants of the complete algorithm.

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

$$H = \begin{pmatrix} 0 & A^H \\ A & 0 \end{pmatrix}^{-1} \qquad \text{et} \qquad K = \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix}$$

This is equivalent to solve the $2n$ rank linear system:

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

From paragraphs 2 and 3, we obtain

Initialization

$$\begin{aligned} \text{Choose } x_1^0, x_2^0 &\in \mathcal{C}^n \\ \text{set } r_1^0 &= b_1 - Ax_1^0 \\ r_2^0 &= b_2 - A^H x_2^0 \\ d_1^0 &= r_1^0 \\ d_2^0 &= r_2^0 \end{aligned}$$

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

1) compute the extremum of J

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

2) generate the new direction

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

The CPU cost is approximatively twice than in the Hermitian conjugate gradient method : two vectors serials x^k, r^k and d^k, two products matrix $\times$ vector by step, but the storage all of the previous directions is avoided. It is not necessary to compute x_2^k ; for example the choice $r_2^0 = \overline{r}_1^0$, which corresponds to an arbitrary choice of b_2 and x_2^0, allows calculations.

Remark

This algorithm is different form the Jacobs' method ([JACOBS 80]), recalled in Appendix.

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

$$r_2^k = \overline{r}_1^k \qquad , \qquad d_2^k = \overline{d}_1^k \quad \forall k$$

This corresponding algorithm can be written as

```
Initialization

        Choose $x^0 \in \mathcal{C}^n$
        set $r^0 = b - Ax^0$
        $d^0 = r^0$

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

1) compute the extremum of $J$

        $\alpha^k = Re\{(r^k, \overline{r}^k)\}/Re\{(Ad^k, \overline{d}^k)\}$
        $x^{k+1} = x^k + \alpha^k d^k$
        $r^{k+1} = r^k - \alpha^k Ad^k$

2) generate the new direction

        $\beta^{k+1} = Re\{(r^{k+1}, \overline{r}^{k+1})\}/Re\{(r^k, \overline{r}^k)\}$
        $d^{k+1} = r^{k+1} + \beta^{k+1} d^k$
```

7 Accelerated Biconjugate gradient method

The Conjugate gradient squared method was introduced in [SONNEVELD 89] . In the Bcg method, the different vectors r_1^k, r_2^k, d_1^k et d_2^k satisfy

$$\begin{array}{rlrl} r_1^k &= \phi_k(A) r_1^0 & \qquad d_1^k &= \theta_k(A) r_1^0 \\ r_2^k &= \phi_k(A^H) r_1^0 & \qquad d_2^k &= \theta_k(A^H) r_1^0 \end{array}$$

where ϕ_k and θ_k are polynomials of degree less or equal to k, and satisfy

$$\begin{aligned} \phi_{k+1}(A) &= \phi_k(A) - \alpha^k A \theta_k(A) \\ \theta_{k+1}(A) &= \phi_{k+1}(A) + \beta^{k+1} \theta_k(A) \end{aligned}$$

To speed-up the Bcg Method, Sonneveld defined 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 (displayed in Appendix). The resulting algorithm, called Cgs (Conjugate Gradient Squared method) is then developped

Initialization

$$\begin{aligned} \text{Choose } x^0 &\in \mathcal{C}^n \\ \text{set } r^0 &= b - A x^0 \\ q^0 &= p^0 = r^0 \end{aligned}$$

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

$$\begin{aligned} \alpha^k &= Re\{(r^k, r^0)\} / Re\{(A q^k, r^0)\} \\ u^k &= p^k - \alpha^k A q^k \\ x^{k+1} &= x^k + \alpha^k (p^k + u^k) \\ r^{k+1} &= r^k - \alpha^k A (p^k + u^k) \\ \beta^{k+1} &= Re\{(r^{k+1}, r^0)\} / Re\{(r^k, r^0)\} \\ p^{k+1} &= r^{k+1} + \beta^{k+1} u^k \\ q^{k+1} &= p^{k+1} + \beta^{k+1} (u^k + \beta^{k+1} q^k) \end{aligned}$$

The CPU cost is almost the same than in the Biconjugate Gradient method, but multiplications by A^H are avoided, so Cgs is easy to vectorize when a vectorization of the product matrix $\times$ vector is available. In the case of a symmetric complex matrix, this method has to converge twice faster to remain efficient!

8 Preconditioning the iterations

The use of a preconditioning matrix to speed-up convergence is very usefull, specially in the case of non-Hermitian matrices. When the matrix has no particular structure, it seems more efficient to use the incomplete factorization developped by [BEHIE, FORSYTH 83], [WALLIS 83] and [WATTS 81], which works in two steps. The first step is a logical factorization of the matrix, in order to obtain the matrix $L+U$ skeleton, that is the locations of non-zeros elements in L and U, by the fill-in level notion. During the second step, the corresponding elements of matrices L and U are computed.

To summarize the logical factorization (first step), the fill-in level notion is introduced according to :

- all the non-zeros elements of A have the level value zero.

- each fill-in element created by k-level elements is set to the level value $k+1$.

It is obvious that the exact factorization of A is obtained for a finite level value, say $l(A)$. Any value of level l, $0 \leq l < l(A)$ leads to an incomplete factorization.

9 Numerical results

Some of the test-problems introduced by [FREUND 90] are used to compare the previous algorithms. The linear system matrix satisfies $A = A_0 - \sigma_1 h^2 I + ih^2 D$, where A_0 is the Laplacian operator matrix for a squared shape domain, $\sigma_1 \in \mathbb{R}$, $i^2 = -1$, and D is a diagonal matrix. The components of the right-hand side x are choosen randomly in $[-10., 10.]$. In a first time, an incomplete factorization of level value 2 is used to define the preconditioning matrix. All algorithms are initialized to $x^0 = 0$, and the convergence criterion is $\|r^k\| < 10^{-6}\|r^0\|$. We use an Apollo DN1000 work-station.

- **example 1**

For the first example $h = 1/64$, $\sigma_1 = 200.$, and D is a diagonal matrix arising from the boundary conditions $\dfrac{\partial u}{\partial n} = i\sigma_2 u$ (on a part of the boundary) and $\sigma_2 = 10.$. There are 3969 unknowns, and non-zeros elements of the matrices are respectively , 15624 for the system matrix, and 30876 for the preconditioning matrix (the factorization time is .55 seconds).

Algorithm	Iterations Number	Time (in seconds)
Normal Equation	121	33.07
Orthomin(10)	NC 3969	$Q = 1.10^{-3}$ *
Orthomin(20)	NC 3969	$Q = 1.10^{-3}$
Gmres(10)	24	48.39
Gmres(20)	15	74.44
Bcg (Jacobs)	58	16.80
Bcg (JM)	60	17.58
Cgs (Jacobs)	NC 3969	$Q = 1.10^{4}$
Cgs (JM)	165	44.07
Gauss (Direct Method)	1	18.14

Table 1

* For non convergent methods, we give the value of $Q = \|r^n\|/\|r^0\|$ at the end of iterations

- **example 2**

For the second example, $h = 1/32$, $\sigma_1 = 100.$, and the D_i's are random numbers in [0.,10.]. There are 961 unknowns, and non-zeros elements of the matrices are respectively , 3720 for the system matrix, and 7260 for the preconditioning matrix (the factorization time is .11 seconds).

Algorithm	Iterations Number	Time (in seconds)
Normal Equation	42	2.6

Orthomin(10)
Orthomin(20)
	279
	279
		9.70
		9.75

Gmres(10)
Gmres(20)
	10
	2
		4.55
		2.22

Bcg (Jacobs) Bcg (JM)	22 54	1.44 3.50
Cgs (Jacobs) Cgs (JM)	NC 961 NC 961	$Q = 7.10^9$ $Q = 2.10^{26}$
Gauss (Direct Method)	1	.99

Table 2

- **example 3**

For the third example, $h = 1/32$, $\sigma_1 = 100.$, and the matrix D arises from the boundary conditions $\frac{\partial u}{\partial n} = i\sigma_2 u$, $\sigma_2 = 10.$. There are 961 unknowns, and non-zeros elements of the matrices are respectively , 3720 for the system matrix, and 7260 for the preconditioning matrix (the factorization time is .11 seconds).

Algorithm	Iterations number	Time (in seconds)
Normal Equation	47	2.91

Orthomin(10)
Orthomin(20)
	NC 961
	NC 961
		$Q = 2.10^{-2}$
		$Q = 2.10^{-2}$

Gmres(10)
Gmres(20)
	13
	3
		5.91
		3.31

Bcg (Jacobs) Bcg (JM)	24 32	1.57 2.08
Cgs (Jacobs) Cgs (JM)	NC 961 NC 961	$Q = 2.10^{6}$ $Q = 8.10^{3}$
Gauss (Direct Method)	1	.99

Table 3

- **example 4**

At least $h = 1/32$, $\sigma_1 = 1000.$, and $D_i = 100.$. There are 961 unknowns, and non-zeros elements of the matrices are respectively , 3720 for the system matrix, and 7260 for the preconditioning matrix (the factorization time is .11 seconds).

Algorithm	Iteration number	Time (in seconds)
Normal Equation	22	1.38

Orthomin(10)
Orthomin(20)
	NC 961
	18
		$Q = 1.10^{-1}$
		1.18

Gmres(10)
Gmres(20)
	NC 961
	NC 961
		$Q = 1.$
		$Q = 1.$

Bcg (Jacobs) Bcg (JM)	53 609	3.44 39.22
Cgs (Jacobs) Cgs (JM)	NC 961 NC 961	$Q = 1.10^{5}$ Overflow ...
Gauss (Direct Method)	1	.99

Table : Solution of Problem 4

Conclusion

Bibliography

[S.F. ASHBY, T.A. MANTEUFFEL, P.E. SAYLOR 90] *A taxonomy for conjugate gradient methods.*
SIAM J. Num. Anal. vol 27

[A. BEHIE, P. A. FORSYTH Jr 83] *Practical Considerations for Incomplete Factorization Methods in Reservoir Simulation. Proceedings of the Seventh Symposium on Reservoir Simulation of the Society of Petroleum Engineers. San Francisco*

[S.C. EISENSTAT, H.C. ELMAN, M.H. SCHULTZ 83] *Variational iteration methods for non symmetric systems of linear equations.*
SIAM J. Num. Anal. vol 20

[R. FLETCHER 76] *Conjugate gradient methods for indefinite systems.*
Proceedings of the Dundee Conference in Numerical Analysis 1975
Springer Verlag

[R. FREUND 90] *On Conjugate Gradient Type Methods and Polynomial Preconditioners for a Class of Complex Non-Hermitian Matrices. Numer.Math. vol 57*

[G. GOLUB, G. MEURANT 81] *Résolution numérique des grands systèmes linéaires. Eyrolles*

[D.A.H. JACOBS 86] *A Generalization of the Conjugate-Gradient Method to Solve Complex Systems. I.M.A. Journal of Numerical Analysis. vol 6*

[P. JOLY 84] *Méthodes de gradient conjugué.*
Publications du Laboratoire d'Analyse Numérique
Université Pierre et Marie Curie. Paris

[N.M. NACHTIGAL, S.C. REDDY, L.N. TREFETHEN 90] *How Fast are Nonsymmetric Matrix Iterations.*
Numerical Analysis Report 90-2. Massachusetts Institute of Technology

[J. A. MEIJERINK, H.A. VAN DER VORST 77] *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix.*
Math. of Comp. vol 31

[Y. SAAD, M.H. SCHULTZ 86] *Gmres a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. vol 7*

[P. SONNEVELD 89] *CGS a Fast Lanczos-type Solver for Nonsymmetric Linear Systems. SIAM J. Sci. Stat. Comput. vol 10*

[P. K. W. VINSOME 76] *Orthomin : an iterative method for solving sparse sets of simultaneous linear equations. Proceedings of* 4^{th} *symposium on reservoir simulation*

[J.R. WALLIS 83] *Incomplete Gaussian Elimination as a Preconditioning for Generalized Conjugate Gradient Acceleration.*
Proceedings of the Seventh Symposium on Numerical Simulation of Reservoir Performance of the Society of Petroleum Engineers. Dallas

[J.W. WATTS 81] *Society of Petroleum Engineers Journal 21*

Appendix 1 : Biconjugate Gradient method acceleration

The residuals and the directions are binded by the relations

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

where $\phi_k(z)$ and $\theta_k(z)$ are complex polynomial 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

$$\tilde{r}^{k+1} = \tilde{r}^k - 2\alpha^k Ap^k + (\alpha^k)^2A^2q^k$$

$$q^{k+1} = \tilde{r}^{k+1} + 2\beta^{k+1}(p^k - \alpha^k Aq^k) + (\beta^{k+1})^2q^k$$

$$p^{k+1} = q^{k+1} + \beta^{k+1}(p^k - \alpha^k Aq^k)$$

Define now $\tilde{x}^k$, and α^k, β^{k+1} From

$$\tilde{r}^{k+1} = \tilde{r}^k - \alpha^k A\big(p^k + (p^k - \alpha^k Aq^k)\big)$$

we get

$$\tilde{x}^{k+1} = \tilde{x}^k - \alpha^k\big(p^k + (p^k - \alpha^k Aq^k)\big)$$

and

$$\alpha^k = Re\{(r_1^k, r_2^k)\}/Re\{(d_2^k, Ad_1^k)\}$$

$$\beta^{k+1} = Re\{(r_1^{k+1}, r_2^{k+1})\}/Re\{(r_1^k, r_2^k)\}$$

so

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

$$(d_2^k, Ad_1^k) = (\theta_k(A^H)r^0, A\theta_k(A)r^0) = (r^0, A\theta_k^2(A)r^0) = (r^0, Aq^k)$$

$$\alpha^k = Re\{(\tilde{r}^k, \tilde{r}^0)\}/Re\{(\tilde{r}^0, Aq^k)\}$$

$$\beta^{k+1} = Re\{(\tilde{r}^{k+1}, \tilde{r}^0)\}/Re\{(\tilde{r}^k, \tilde{r}^0)\}$$

Appendice 2 : Jacobs' Biconjugate Gradient method

In [JACOBS 80] is proposed an algorithm using the residuals r_1^k and r_2^l orthogonality. This algorithm is written as :

Initialization

$$\begin{aligned} \text{Choose } x_1^0, x_2^0 &\in \mathcal{C}^n \\ \text{set } r_1^0 &= b - Ax_1^0 \\ r_2^0 &= \overline{r}_1^0 \\ d_1^0 &= r_1^0 \\ d_2^0 &= r_2^0 \end{aligned}$$

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

1) compute α^k

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

2) generate the new direction

$$\begin{aligned} \beta^{k+1} &= (r_1^{k+1}, r_2^{k+1})/(r_1^k, r_2^k) \\ d_1^{k+1} &= r_1^{k+1} + \beta^{k+1} d_1^k \\ d_2^{k+1} &= r_2^{k+1} + \overline{\beta}^{k+1} d_2^k \end{aligned}$$

The CPU cost of this algorithm is exactly the same as 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 et K are defined as

$$H = \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}^{-1} \qquad K = \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix}$$

The following relations are deduced by induction:

$$
\begin{array}{lll}
1) & (r_1^k, r_2^l) = 0 & \forall k \neq l \\
2) & (Ad_1^l, d_2^k) = (d_1^k, A^H d_2^l) = 0 & \forall k \neq l \\
3) & (r_1^k, d_2^l) = (r_2^k, d_1^l) = 0 & \forall k \neq l
\end{array}
$$

Furthermore :

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

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

Initialization

$$
\begin{aligned}
\text{Choose } x^0 &\in \mathcal{C}^n \\
\text{set } r^0 &= b - Ax^0 \\
q^0 &= p^0 = r^0 \\
\tilde{r}^0 &= r^0 \text{ or any other choice } \neq 0
\end{aligned}
$$

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

$$
\begin{aligned}
\alpha^k &= (\tilde{r}^0, r^k)/(\tilde{r}^0, Aq^k) \\
u^k &= p^k - \alpha^k A q^k \\
x^{k+1} &= x^k + \alpha^k (p^k + u^k) \\
r^{k+1} &= r^k - \alpha^k A(p^k + u^k) \\
\beta^{k+1} &= (\tilde{r}^0, r^{k+1})/(\tilde{r}^0, r^k) \\
p^{k+1} &= r^{k+1} + \beta^{k+1} u^k \\
q^{k+1} &= p^{k+1} + \beta^{k+1}(u^k + \beta^{k+1} q^k)
\end{aligned}
$$

If A is symmetric complex, then

$$
r_2^k = \overline{r}_1^k \qquad , \qquad d_2^k = \overline{d}_1^k \quad \forall k
$$

and the algorithm can be rewritten in

Initialization

$$\begin{aligned} \text{Choose } x^0 &\in \mathcal{C}^n \\ \text{set } r^0 &= b - Ax^0 \\ d^0 &= r^0 \end{aligned}$$

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

1) compute the extremum of J

$$\begin{aligned} \alpha^k &= (r^k, \overline{r}^k)/(Ad^k, \overline{d}^k) \\ x^{k+1} &= x^k + \alpha^k d^k \\ r^{k+1} &= r^k - \alpha^k Ad^k \end{aligned}$$

2) generate the new direction

$$\begin{aligned} \beta^{k+1} &= (r^{k+1}, \overline{r}^{k+1})/(r^k, \overline{r}^k) \\ d^{k+1} &= r^{k+1} + \beta^{k+1} d^k \end{aligned}$$