The computation of bounds for the norm of the error in the conjugate gradient algorithm

Gérard Meurant

CEA/LV, 94195 Villeneuve St Georges cedex, France

In this paper we consider computing estimates of the norm of the error in the conjugate gradient (CG) algorithm. Formulas were given in an earlier paper [4]. Here, we first prove that these expressions are indeed upper and lower bounds for the A–norm of the error.

Moreover, starting from these formulas, we investigate the computation of the l_2–norm of the error. Finally, we define an adaptive algorithm where the approximations of the extreme eigenvalues that are needed to obtain upper bounds are computed when running CG leading to an improvement of the upper bounds for the norm of the error. Numerical experiments show the effectiveness of this algorithm.

Keywords: Errors bounds, Conjugate Gradient

AMS Subject classification: 65F50

1. Introduction

Let A be a large and sparse symmetric positive definite matrix of order n and suppose we have an approximate solution $\tilde{x}$ of the linear system

$$Ax = g, \tag{1}$$

where g is a given vector. The residual r is defined as,

$$r = g - A\tilde{x}. \tag{2}$$

The error e being $e = x - \tilde{x}$, we obviously have,

$$e = A^{-1}r. \tag{3}$$

Therefore, if we consider the A–norm of the error,

$$\|e\|_A^2 = e^T Ae = r^T A^{-1} A A^{-1} r = r^T A^{-1} r. \tag{4}$$

It is sometimes also of interest to use the l_2–norm, for which $\|e\|^2 = r^T A^{-2} r$.

Hence, if we want approximations of the A–norm of the error, we must consider the quadratic form $r^T A^{-1} r$. This has been done in several papers ([2], [3], [4], [6]). We will not repeat here the contents of these papers. Let us just say that the quadratic form is considered as a Stieltjes integral with an (unknown) positive measure. Then, quadrature rules are used to obtain approximations of the integral. It turns out that for the case we consider the Gauss rule gives a lower bound and the Gauss–Radau rule both a lower and an upper bound when the prescribed nodes are either the left or right ends of a segment containing the (positive) eigenvalues of A. Prescribing both ends of the segment, we obtain the Gauss–Lobatto rule and an upper bound. Nodes and weights of the quadrature formulas can be computed by using the orthogonal polynomials associated with the measure. In turn, the recurrence relations of these orthogonal polynomials are given by the Lanczos algorithm starting from the (normalized) residual. Finally, everything amounts to computing the $(1,1)$ element of the inverse of the tridiagonal matrix constructed from the Lanczos coefficients, see [3]. For the Gauss–Radau and the Gauss–Lobatto rules, the tridiagonal matrix has to be extended in order to have the prescribed nodes as eigenvalues.

The contents of the paper are as follows. In Section 2, we recall how to compute approximations of the A–norm and then, we show that these approximations are lower and upper bounds. Section 3 show how to compute approximations of the l_2–norm. Finally in Section 4, we introduce an adaptive algorithm that computes estimates of the smallest eigenvalue of A that is needed to obtain upper bounds of the norm. We also give some numerical experiments showing the effectiveness of this approach.

2. Bounds for the A–norm

If we have an approximate solution of equation 1, we can compute the residual and then, we run a few steps of the Lanczos algorithm to obtain bounds for the A–norm of the error. Numerical experiments in [4] show that this method is quite efficient. However, this algorithm does not seem to make too much sense when the approximate solution is computed by CG. As the Lanczos algorithm and CG are in some sense equivalent, it would be strange to use the Lanczos method starting from CG residuals. Instead, it was proposed in [4] to use the following formula,

$$\|e^k\|_A^2 = \|x - x^k\|_A^2 = {r^0}^T A^{-1} r^0 - \|r^0\|^2 (J_k^{-1})_{1,1} \tag{5}$$

$$= \|r^0\|^2 ((J_n^{-1})_{1,1} - (J_k^{-1})_{1,1}). \tag{6}$$

where

$$J_k = \begin{pmatrix} \omega_1 & \gamma_1 & & & \\ \gamma_1 & \omega_2 & \gamma_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \gamma_{k-2} & \omega_{k-1} & \gamma_{k-1} \\ & & & \gamma_{k-1} & \omega_k \end{pmatrix}, \tag{7}$$

is the tridiagonal matrix of the Lanczos coefficients and

$$\omega_k = \frac{1}{\alpha_{k-1}} + \frac{\beta_{k-1}}{\alpha_{k-2}}, \quad \beta_0 = 0, \quad \alpha_{-1} = 1 \tag{8}$$

$$\gamma_k = \frac{\sqrt{\beta_k}}{\alpha_{k-1}}, \tag{9}$$

α_k and β_k being the CG coefficients: let x^0 be given, $r^0 = g - Ax^0$, $p^0 = r^0$, for $k = 1, \ldots$ until convergence

$$\alpha_{k-1} = \frac{{r^{k-1}}^T r^{k-1}}{{p^{k-1}}^T A p^{k-1}}, \tag{10}$$

$$x^k = x^{k-1} + \alpha_{k-1} p^{k-1}, \tag{11}$$

$$r^k = r^{k-1} - \alpha_{k-1} A p^{k-1}, \tag{12}$$

$$\beta_k = \frac{{r^k}^T r^k}{{r^{k-1}}^T r^{k-1}}, \tag{13}$$

$$p^k = r^k + \beta_k p^{k-1}. \tag{14}$$

Formula 6 has already been used in [6] for reconstructing the A–norm of the error when CG has converged but the $(1,1)$ element of the inverse of the tridiagonal matrix was computed by means of continued fractions. A round off error analysis given in [6] shows that below a certain value of $\|e^k\|_A^2$, no more useful information can be obtained from this algorithm.

Formula 6 has also been used in [2] but there, the computations of $\|e^k\|_A$ were not done below 10^{-5} and the same difficulties arise as in [6]. It has been shown in [4] that these difficulties can be overcome and that reliable estimates of $\|e^k\|_A$ can be computed whatever the value of the norm is.

Of course, equation 6 cannot be used directly as, at CG iteration k, we do not know $(J_n^{-1})_{1,1}$. But it is known that $(J_k^{-1})_{1,1} \to (J_n^{-1})_{1,1}$. So, we will use the current value of $(J_k^{-1})_{1,1}$ to approximate the final value. Let b_k be the computed value of $(J_k^{-1})_{1,1}$. This is obtained in an additive way by using the Sherman–Morrison formula. Let $j_k = J_k^{-1}e_k$ be the last column of the inverse of J_k, e_k being the k–th column of the identity matrix, then

$$(J_{k+1}^{-1})_{1,1} = (J_k^{-1})_{1,1} + \frac{\gamma_k^2 (j_k j_k^T)_{1,1}}{\omega_{k+1} - \gamma_k^2 (j_k)_k}. \tag{15}$$

The first and last elements of the last column of the inverse of J_k that we need can be computed using the Cholesky decomposition of J_k. Let $d_1 = \omega_1$ and

$$d_i = \omega_i - \frac{\gamma_{i-1}^2}{d_{i-1}}, \quad i = 2, \dots, k \tag{16}$$

then, let

$$(j_k)_1 = (-1)^{k-1}\frac{\gamma_1 \cdots \gamma_{k-1}}{d_1 \cdots d_k}, \quad (j_k)_k = \frac{1}{d_k}. \tag{17}$$

Using these results, we have

$$f_k = \frac{\gamma_{k-1}^2 c_{k-1}^2}{d_{k-1}(\omega_k d_{k-1} - \gamma_{k-1}^2)}, \quad b_k = b_{k-1} + f_k. \tag{18}$$

Notice that by using the definition of d_k, f_k can be computed as c_k^2/d_k. As J_k is positive definite, this shows that $f_k > 0$. Let s_k be the estimate of $\|e^k\|_A^2$ we are looking for and d be a positive integer (to be named the delay), at CG iteration number k, we set

$$s_{k-d} = \|r^0\|^2 (b_k - b_{k-d}). \tag{19}$$

This will give us an estimate of the error d iterations before the current one. However, it was shown in [4] that if we compute b_k and use straightforwardly (2.2), there exists a k_{max} such that if $k > k_{max}$ then, $s_k = 0$. This happens because, when k is large enough, $\gamma_k/d_k < 1$ and $c_k \to 0$ and consequently $f_k \to 0$. Therefore, when $k > k_{max}, b_k = b_{kmax}$.

But, as it was noticed in [4], we can compute s_{k-d} in another way as we just need to sum up the last d values of f_j. The algorithm computing the iterates of CG and estimates from the Gauss (s_{k-d}), Gauss–Radau ($\underline{s}_{k-d}$ and $\bar{s}_{k-d}$) and Gauss–Lobatto ($\breve{s}_{k-d}$) rules is the following (with slight simplifications from [4]):

Algorithm 1 CGQL.

```
    let $x^0$ be given, $r^0 = g - Ax^0$, $p^0 = r^0$, $\beta_0 = 0$, $\alpha_{-1} = 1$, $c_1 = 1$,
    for $k = 1, \dots$ until convergence

        $\alpha_{k-1} = \frac{{r^{k-1}}^T r^{k-1}}{{p^{k-1}}^T A p^{k-1}},$    (20)

        $\omega_k = \frac{1}{\alpha_{k-1}} + \frac{\beta_{k-1}}{\alpha_{k-2}},$    (21)

if $k = 1$ ————————————————
        $f_1 = \frac{1}{\omega_1},$    (22)

        $d_1 = \omega_1,$    (23)

        $\bar{d}_1 = \omega_1 - a,$    (24)

        $\underline{d}_1 = \omega_1 - b,$    (25)

else ————————————————
        $c_k = c_{k-1} \frac{\gamma_{k-1}}{d_{k-1}},$    (26)

        $d_k = \omega_k - \frac{\gamma_{k-1}^2}{d_{k-1}},$    (27)

        $f_k = \frac{\gamma_{k-1}^2 c_{k-1}^2}{d_{k-1}(\omega_k d_{k-1} - \gamma_{k-1}^2)} = \frac{c_k^2}{d_k},$    (28)

        $\bar{d}_k = \omega_k - a - \frac{\gamma_{k-1}^2}{\bar{d}_{k-1}} = \omega_k - \bar{\omega}_{k-1},$    (29)

        $\underline{d}_k = \omega_k - b - \frac{\gamma_{k-1}^2}{\underline{d}_{k-1}} = \omega_k - \underline{\omega}_{k-1}$    (30)

end ————————————————
        $x^k = x^{k-1} + \alpha_{k-1} p^{k-1},$    (31)

        $r^k = r^{k-1} - \alpha_{k-1} A p^{k-1},$    (32)
```

$$\beta_k = \frac{{r^k}^T r^k}{{r^{k-1}}^T r^{k-1}}, \tag{33}$$

$$p^k = r^k + \beta_k p^{k-1}, \tag{34}$$

$$\bar{\omega}_k = a + \frac{\gamma_k^2}{\bar{d}_k}, \tag{35}$$

$$\underline{\omega}_k = b + \frac{\gamma_k^2}{\underline{d}_k}, \tag{36}$$

$$\breve{\omega}_k = \frac{\bar{d}_k \underline{d}_k}{\underline{d}_k - \bar{d}_k} \left(\frac{b}{\bar{d}_k} - \frac{a}{\underline{d}_k} \right), \tag{37}$$

$$\breve{\gamma}_k^2 = \frac{\bar{d}_k \underline{d}_k}{\underline{d}_k - \bar{d}_k} (b - a), \tag{38}$$

$$\bar{f}_k = \frac{\gamma_k^2 c_k^2}{d_k(\bar{\omega}_k d_k - \gamma_k^2)}, \tag{39}$$

$$\underline{f}_k = \frac{\gamma_k^2 c_k^2}{d_k(\underline{\omega}_k d_k - \gamma_k^2)}, \tag{40}$$

$$\breve{f}_k = \frac{\breve{\gamma}_k^2 c_k^2}{d_k(\breve{\omega}_k d_k - {\breve{\gamma}_k}^2)}, \tag{41}$$

if $k > d$ ————————————————

$$t_k = \sum_{j=k-d+1}^{k} f_j, \tag{42}$$

$$s_{k-d} = \|r^0\|^2 t_k, \tag{43}$$

$$\bar{s}_{k-d} = \|r^0\|^2 (t_k + \bar{f}_k), \tag{44}$$

$$\underline{s}_{k-d} = \|r^0\|^2 (t_k + \underline{f}_k), \tag{45}$$

$$\breve{s}_{k-d} = \|r^0\|^2 (t_k + \breve{f}_k) \tag{46}$$

end ————————————————

In this algorithm a and b are lower and upper bounds of the smallest and largest eigenvalues of A. Notice that the value of s_k is independent of a and b, $\bar{s}_k$ depends only on a and $\underline{s}_k$ only on b. Let us now prove that this algorithm do give lower and upper bounds for the A–norm of the error.

Lemma 2. Let J_k, $\underline{J}_k$, $\bar{J}_k$ and $\breve{J}_k$ be the tridiagonal matrices of the Gauss, Gauss–Radau (with b and a as prescribed nodes) and the Gauss–Lobatto rules. Then, if $0 < a \leq \lambda_{min}(A)$ and $b \geq \lambda_{max}(A)$, $\|r^0\|(J_k^{-1})_{1,1}$, $\|r^0\|(\underline{J}_k^{-1})_{1,1}$ are lower bounds of $\|e^0\|^2 = r^0 A^{-1} r^0$, $\|r^0\|(\bar{J}_k^{-1})_{1,1}$ and $\|r^0\|(\breve{J}_k^{-1})_{1,1}$ are upper bounds of $r^0 A^{-1} r^0$.

Proof. see [4]. The proof is obtained easily as we know the sign of the remainder in the quadrature rules. Notice that $\underline{J}_k$ and $\bar{J}_k$ are of order $k+1$ and $\breve{J}_k$ is of order $k+2$. We have that $\bar{f}_k > \underline{f}_k$ and therefore, $\bar{\omega}_k < \underline{\omega}_k$. □

Theorem 3. At iteration number k of CGQL, s_{k-d} and $\underline{s}_{k-d}$ are lower bounds of $\|e^{k-d}\|_A^2$, $\bar{s}_{k-d}$ and $\breve{s}_{k-d}$ are upper bounds of $\|e^{k-d}\|_A^2$.

Proof. We have

$$\|e^{k-d}\|_A^2 = \|r^0\|^2((J_n^{-1})_{1,1} - (J_{k-d}^{-1})_{1,1}) \tag{47}$$

and

$$s_{k-d} = \|r^0\|^2((J_k^{-1})_{1,1} - (J_{k-d}^{-1})_{1,1}). \tag{48}$$

Therefore,

$$\|e^{k-d}\|_A^2 - s_{k-d} = \|r^0\|^2((J_n^{-1})_{1,1} - (J_k^{-1})_{1,1}) > 0, \tag{49}$$

showing that s_{k-d} is a lower bound of $\|e^{k-d}\|_A^2$. The same kind of proof applies for the other cases as, for instance,

$$\bar{s}_{k-d} = \|r^0\|^2((\bar{J}_k^{-1})_{1,1} - (J_{k-d}^{-1})_{1,1}). \tag{50}$$

□

Therefore, the quantities that we are computing in CGQL are indeed upper and lower bounds of the A–norm of the error. It turns out that the best bounds are the ones computed by the Gauss–Radau rule.

3. Estimates for the l_2–norm

The computation of the l_2–norm is more complicated as there is no such formula as 6 for the l_2–norm. We are going to use a technique introduced by Bai and Golub [1] for computing estimates of the trace of the inverse of a matrix. Consider a given iteration k of CG and for simplicity let $r = r^k$ and

$$\mu_p = \int_a^b \lambda^p \, d\alpha(\lambda) = (r, A^p r). \tag{51}$$

Notice that we know μ_1, μ_0. The moment μ_1 can be computed during CG iterations by computing Ar^k. Then, Ap^k is computed recursively by $Ap^k = Ar^k + \beta_k Ap^{k-1}$ to save a matrix multiply at the expense of storing one more vector. Moreover, we know upper and lower bounds of μ_{-1}. We are interested in computing estimates of μ_{-2}. As in [1], we use a one point Gauss–Radau formula to compute an estimate $\tilde{\mu}_{-2}$. We have

$$\tilde{\mu}_p = w_0 t_0^p + w_1 t_1^p, \tag{52}$$

where $t_0 = a$ or b is the prescribed node. Moreover,

$$c\tilde{\mu}_p + d\tilde{\mu}_{p-1} - \tilde{\mu}_{p-2} = 0, \tag{53}$$

t_0 being a solution of $c\xi^2 + d\xi - 1 = 0$. In fact, we know the exact values of $\tilde{\mu}_1 = \mu_1 = r^T Ar$ and $\tilde{\mu}_0 = \mu_0 = r^T r$. Then,

$$\begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} \mu_1 & \mu_0 \\ t_0^2 & t_0 \end{pmatrix}^{-1} \begin{pmatrix} \tilde{\mu}_{-1} \\ 1 \end{pmatrix}. \tag{54}$$

This gives

$$c = \frac{1}{\mu_1 t_0 - \mu_0 t_0^2}(t_0 \tilde{\mu}_{-1} - \mu_0), \tag{55}$$

$$d = \frac{1}{\mu_1 t_0 - \mu_0 t_0^2}(\mu_1 - t_0^2 \tilde{\mu}_{-1}). \tag{56}$$

Finally,

$$\tilde{\mu}_{-2} = \frac{1}{\mu_1 t_0 - \mu_0 t_0^2}[(t_0 \tilde{\mu}_{-1} - \mu_0)\mu_0 + (\mu_1 - t_0^2 \tilde{\mu}_{-1})\tilde{\mu}_{-1}]. \tag{57}$$

If we would know the exact value μ_{-1}, setting $t_0 = a$ would give an upper bound and $t_0 = b$ a lower bound. We compute two estimates of μ_{-2} by taking $t_0 = a$ and an upper bound of μ_{-1} from the Gauss–Radau rule for $r^T A^{-1} r$ as well as $t_0 = b$ and a lower bound of μ_{-1}. Unfortunately, so far, we have not been able to prove that this gives an upper and a lower bound for μ_{-2}. All we know is that $\mu_1 t_0 - \mu_0 t_0^2 > 0$. It turns out, in the numerical examples we have ran, that the quadratic function $\tilde{\mu}_{-2}$ is an increasing function of $\tilde{\mu}_{-1}$ giving numerically upper and lower bounds.

4. An adaptive algorithm

The lower bound s_k is independent of the extreme eigenvalues a and b. However, to compute the upper bound $\bar{s}_k$ we need an estimate of the smallest eigenvalue of A. We can see that $\bar{s}_k \to \infty$ if $a \to 0$. Therefore, if we take a value of a that is too small, we could obtain a large overestimate of the A–norm. Of course, there are cases where we can obtain good analytic estimates of the smallest eigenvalue, for example if the matrix A arises from finite element methods. However, as we are using CG we can obtain numerical estimates of the smallest eigenvalue when running the algorithm.

It is well known that the extreme eigenvalues of J_k are approximations of the extreme eigenvalues of A that are getting better and better as k increases. Therefore, we propose the following algorithm. We start the CGQL iterations with $a = a_0$ an underestimate of $\lambda_{min}(A)$. An estimate of the smallest eigenvalue can be easily obtained by inverse iteration (see [5]) as, for computing the bounds of the norm, we already compute incrementally the Cholesky factorization of J_k. The smallest eigenvalue of J_k is obtained by repeatedly solving tridiagonal systems. We use a fixed number n_a of (inner) iterations of inverse iteration at every CG iteration, giving a value δ_k. When δ_k is such that

$$\frac{|\delta_k - \delta_{k-1}|}{\delta_k} \le \varepsilon_a, \tag{58}$$

with a prescribed ε_a, then we switch by setting $a = \delta_k$, we stop computing the eigenvalue estimate and we go on with CGQL.

5. Numerical experiments

As test problems, we use three of the examples that were used in [4]. Example 2 arises from the 5–point finite difference approximation of a diffusion equation in a unit square,

$$-\mathrm{div}(a \nabla u)) = f,$$

Figure 1. CG, Example 3, $d = 20$, solid line: log_{10} of l_2–norm, dashed and dotted lines: estimates

Figure 2. CG, Example 4, $d = 10$, solid line: log_{10} of l_2–norm, dashed and dotted lines: estimates

Figure 3. CG, Example 4, $d = 10$, relative differences between the l_2–norm and the estimates

with Dirichlet boundary conditions. $a(x, y)$ is a diagonal matrix with equal diagonal elements. This element is equal to 1000 in a square $]1/4, 3/4[\times]1/4, 3/4[$, 1 otherwise. Example 3 is the same with different diffusion coefficients. The coefficient in the x direction is 100 if $x \in [1/4, 3/4]$, 1 otherwise. The coefficient in the y direction is constant and equal to 1. For this two problems, we choose $n = 900$, the right hand side of equation 1 such that the exact solution x_{ex} is $x_{ex} = (1, \dots, 1)^T$ and a random initial guess x^0.

Example 4 is taken from [6]. The matrix A is diagonal. The diagonal elements are defined as

$$\mu_i = a + \frac{i-1}{n-1}(b-a)\rho^{n-i}, \quad i = 2, \dots, n-1 \quad \mu_1 = a, \ \mu_n = b$$

As in [6], we take $n = 48$, $a = 0.1$, $b = 100$ and $\rho = 0.875$.

We start by computing estimates of the l_2–norm of the error for Example 3 and a delay $d = 20$. The log_{10} of the l_2–norm of the error and of the estimates are given on Figure 1. We see that we can compute good bounds of the order of magnitude of the l_2–norm.

Figure 2 gives the results for the l_2–norm on Example 4 with $d = 10$. The results are quite good. This is confirmed by Figure3 that shows the relative differences between the l_2–norm and the two estimates.

Let us now look at the results of the adaptive algorithm for computing bounds of the A–norm of the error. We consider Example 2 with $d = 20$. The exact smallest eigenvalue is $1.022 \ 10^{-5}$. We start CGQL with $a_0 = 10^{-10}$ and we use $n_a = 2$, $\varepsilon = 10^{-4}$. We can see on Figure 4 that, at the beginning, we have a large overestimate of the norm. It takes 80 CG iterations for the smallest eigenvalue to converge within the prescribed accuracy. Then, we switch to the new value of a and the upper bound becomes very close to the actual value of the A–norm. Numerical experiments show that the results are not very sensitive to the value of n_a. The value of ε_a has to be taken small enough to prevent an early convergence of the smallest eigenvalue because in that case we can obtain a value that is much larger than the exact value, in which case the estimate is no

Figure 4. CG, Example 2, $d = 20$, solid line: log_{10} of the A–norm of the error, dashed line: adaptive algorithm

longer an upper bound. However, we do not loose too much work by taking ε_a small as, for instance, we need only 92 iterations to converge if $\varepsilon_a = 10^{-7}$.

References

[1] Z. Bai and G.H. Golub, Bounds for the trace of the inverse and the determinant of symmetric positive definite matrices, to appear, (1996).

[2] B. Fischer and G.H. Golub, On the error computation for polynomial based iteration methods, Report NA 92–21, Stanford University (1992).

[3] G.H. Golub and G. Meurant, Matrices, moments and quadrature, in: *Numerical Analysis 1993*, eds. D.F. Griffiths and G.A. Watson, Pitman Research Notes in Mathematics, v 303, 1994, pp. 105–156.

[4] G.H. Golub and G. Meurant, Matrices, moments and quadrature II or how to compute the norm of the error in iterative methods, to appear in BIT, v 37, (1997).

[5] G.H. Golub and C. Van Loan, *Matrix computations,* Johns Hopkins University Press, 1989.

[6] G.H. Golub and Z. Sstrakoš, Estimates in quadratic formulas, Numerical Algorithms, v 8, no II–IV, (1994).