Numerical experiments in computing bounds for the norm of the error in the preconditioned conjugate gradient algorithm

Gérard Meurant

CEA/DIF, BP 12, 91680 Bruyères le Châtel, France

In this paper we consider algorithms to compute bounds of the A–norm of the error in the preconditioned conjugate gradient (PCG) algorithm. We extend to PCG formulas that were given in an earlier paper [8]. We give numerical experiments which show that good upper and lower bounds can be obtained provided estimates of the lowest and largest eigenvalues of the preconditioned matrix are given or adaptively computed.

Keywords: Errors bounds, Preconditioned Conjugate Gradient

AMS Subject classification: 65F50

1. Introduction

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

$$Ax = g, \tag{1.1}$$

where the right hand side g is a given vector. The residual r is defined as

$$r = g - A\tilde{x}. \tag{1.2}$$

The error e being $e = x - \tilde{x}$, we have the following relationship between the error and the residual

$$e = A^{-1}r. \tag{1.3}$$

Therefore, the A–norm of the error $\|e\|_A$ is given by

$$\|e\|_A^2 = e^T Ae = r^T A^{-1} A A^{-1} r = r^T A^{-1} r. \tag{1.4}$$

In a series of papers ([1], [3], [4], [5], [7]) it was shown how to compute lower and upper bounds for the quadratic form $r^T A^{-1} r$. Basically the quadratic form is considered as a Stieltjes integral with an (unknown) positive measure. Quadrature rules are then used to obtain approximations of the integral. 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.

In [5] and [8] it was shown how to use these ideas to compute bounds for the A–norm of the error in the conjugate gradient algorithm. In this paper we extend these results to the preconditioned conjugate gradient algorithm and we provide numerical experiments showing that we can reliably compute good estimates of the A–norm of the error. The contents of the paper are as follows. In section 2, we recall the results of [5] and [8] for CG without preconditioning. Section 3 shows how to incorporate a preconditioner and Section 4 gives numerical experiments.

2. CG without preconditioning

Let x^k be the CG approximation at iteration k and r^k be the corresponding residual. For computing the A–norm of the error it was proposed in [5] to use the following formula (see [7] and section 3 below for a proof),

$$\|e^k\|_A^2 = \|x - x^k\|_A^2 = {r^0}^T A^{-1} r^0 - \|r^0\|^2 (J_k^{-1})_{1,1} \tag{2.1}$$

$$= \|r^0\|^2 ((J_n^{-1})_{1,1} - (J_k^{-1})_{1,1}). \tag{2.2}$$

where

$$J_k = \begin{pmatrix} \omega_1 & \gamma_1 & & & \\ \gamma_1 & \omega_2 & \gamma_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \gamma_{k-2} & \omega_{k-1} & \gamma_{k-1} \\ & & & \gamma_{k-1} & \omega_k \end{pmatrix}, \tag{2.3}$$

is the tridiagonal matrix of the coefficients in the Lanczos algorithm. These coefficients are related to those in CG by

$$\omega_k = \frac{1}{\alpha_{k-1}} + \frac{\beta_{k-1}}{\alpha_{k-2}}, \quad \beta_0 = 0, \quad \alpha_{-1} = 1 \tag{2.4}$$

$$\gamma_k = \frac{\sqrt{\beta_k}}{\alpha_{k-1}}, \tag{2.5}$$

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

$$\alpha_{k-1} = \frac{{r^{k-1}}^T r^{k-1}}{{p^{k-1}}^T A p^{k-1}}, \tag{2.6}$$

$$x^k = x^{k-1} + \alpha_{k-1} p^{k-1}, \tag{2.7}$$

$$r^k = r^{k-1} - \alpha_{k-1} A p^{k-1}, \tag{2.8}$$

$$\beta_k = \frac{{r^k}^T r^k}{{r^{k-1}}^T r^{k-1}}, \tag{2.9}$$

$$p^k = r^k + \beta_k p^{k-1}. \tag{2.10}$$

In [8] it is shown how formula (2.2) can be used to estimate the error. At CG iteration k, we do not know $(J_n^{-1})_{1,1}$. But it is known that $(J_k^{-1})_{1,1} \rightarrow (J_n^{-1})_{1,1}$. Therefore, we use the current value of $(J_k^{-1})_{1,1}$ to approximate the final value. Let b_k be the computed value of $(J_k^{-1})_{1,1}$. It is obtained (see [8]) in an additive way by using the Sherman–Morrison formula (see [6]). Let j_k be the last column of the inverse of J_k, then

$$(J_{k+1}^{-1})_{1,1} = (J_k^{-1})_{1,1} + \frac{\gamma_k^2 (j_k j_k^T)_{1,1}}{\omega_{k+1} - \gamma_k^2 (j_k)_k}. \tag{2.11}$$

The first and last elements of the last column of the inverse of J_k that we need are easily computed using the Cholesky decomposition of J_k. Let s_k be the estimate of $\|e^k\|_A^2$ and d be a positive integer (the delay). At CG iteration number k, we set

$$s_{k-d} = \|r^0\|^2 (b_k - b_{k-d}). \tag{2.12}$$

This will give us an estimate of the error d iterations before the current one. It is shown in [5] how to efficiently compute the difference $b_k - b_{k-d}$. Numerical results in [5] and [8] show the effectiveness of this approach. What is needed to obtain lower and upper bounds of the error are estimates of the lowest and largest eigenvalues of A. This can be computed when running the CG iterations. In [8] an adaptive algorithm is described which starts from rough approximations of the eigenvalues. It improves these approximations during the first CG iterations by computing the extreme eigenvalues of the Lanczos matrix which is implicitly constructed by CG and switch to the "converged" eigenvalues when certain criteria are reached.

3. CG with preconditioning

Let M be a symmetric positive definite matrix which is going to be the preconditioner. It is well known that PCG for solving (1.1) is obtained by applying CG to the transformed system

$$M^{-1/2}AM^{-1/2}(M^{1/2}x) = M^{-1/2}g, \tag{3.1}$$

for which the matrix is still symmetric positive definite. Then we obtain recurrences for the approximations to x by going back to the original variables. Let $r^k = g - Ax^k$ and $y^k = M^{1/2}x^k$. For (3.1) the residual is

$$\hat{r}^k = M^{-1/2}g - M^{-1/2}AM^{-1/2}y^k = M^{-1/2}(g - Ax^k) = M^{-1/2}r^k. \tag{3.2}$$

Let z^k be given by solving $Mz^k = r^k$. Then, the scalar product we need in PCG is

$$(\hat{r}^k)^T\hat{r}^k = (\hat{r}^k, \hat{r}^k) = (M^{-1}r^k, r^k) = (z^k, r^k). \tag{3.3}$$

Moreover, let $\hat{p}^k = M^{1/2}p^k$. Then

$$(\hat{p}^k, M^{-1/2}AM^{-1/2}\hat{p}^k) = (p^k, Ap^k). \tag{3.4}$$

By using this change of variable, the PCG algorithm is the following: let x^0 be given, $r^0 = g - Ax^0$, $Mz^0 = r^0$, $p^0 = z^0$, for $k = 1, \ldots$ until convergence

$$\alpha_{k-1} = \frac{{z^{k-1}}^T r^{k-1}}{{p^{k-1}}^T Ap^{k-1}}, \tag{3.5}$$

$$x^k = x^{k-1} + \alpha_{k-1} p^{k-1}, \tag{3.6}$$

$$r^k = r^{k-1} - \alpha_{k-1} A p^{k-1}, \tag{3.7}$$

$$M z^k = r^k, \tag{3.8}$$

$$\beta_k = \frac{{z^k}^T r^k}{{z^{k-1}}^T r^{k-1}}, \tag{3.9}$$

$$p^k = z^k + \beta_k p^{k-1}. \tag{3.10}$$

Let $\hat{e}^k = y^k - y$ where $y = M^{1/2} x$ and $e^k = x^k - x$. Then,

$$\|\hat{e}^k\|^2_{M^{-1/2}AM^{-1/2}} = (M^{-1/2}AM^{-1/2}(y^k - y), y^k - y) = (A(x^k - x), x^k - x) = \|e^k\|^2_A. \tag{3.11}$$

This shows that we can use the formula

$$\|e^k\|^2_A = (z^0, r^0)((J_n^{-1})_{1,1} - (J_k^{-1})_{1,1}), \tag{3.12}$$

where the Lanczos matrix J_k is constructed from the PCG coefficients. This result can also be proved directly in the following way (we give this proof for the convenience of the reader. It is essentially similar to the proof in the report from where [7] is issued). Let $K = M^{-1}A$. It is well known (see for instance Lemma 6.9 of [9]) that PCG generates a polynomial P_k of degree k which satisfies a three–term recurrence and such that

$$z^{k+1} = [I - K P_k(K)] z^0. \tag{3.13}$$

This implies that we have

$$r^{k+1} = r^0 - A P_k(K) M^{-1} r^0. \tag{3.14}$$

Because of the orthogonality relations which hold for PCG (namely orthogonality of r^{k+1} and p^k), we have

$$\|e^{k+1}\|^2_A = (r^{k+1}, A^{-1} r^0) = (r^0, A^{-1} r^0) - (P_k(K) M^{-1} r^0, r^0). \tag{3.15}$$

Let $Z_k = [z^0 \; z^1 \ldots z^k]$ be the matrix whose columns are the vectors z^j. Then we have

$$Z_k^T K Z_k = Z_k^T Z_k T_k, \tag{3.16}$$

where T_k is a tridiagonal matrix of order $k+1$. Let D_k be a diagonal matrix such that $Z_k^T M Z_k = D_k^2$ and $\tilde{Z}_k = M^{1/2} Z_k D_k^{-1}$. We have (see [9])

$$(\tilde{Z}_k)^T M^{1/2} K M^{-1/2} \tilde{Z}_k D_k = D_k T_k \tag{3.17}$$

and

$$(\tilde{Z}_k)^T \tilde{Z}_k = D_k^{-1} Z_k^T M Z_k D_k^{-1} = I. \tag{3.18}$$

Therefore $\tilde{Z}_k$ is an orthonormal matrix. The matrix T_k is similar to $(\tilde{Z}_k)^T \hat{K} \tilde{Z}_k$ where $\hat{K} = M^{-1/2} A M^{-1/2}$ which is similar to $M^{-1}A$. Then,

$$(\tilde{Z}_k)^T M^{1/2} P_k(K) M^{-1/2} \tilde{Z}_k = D_k P_k(T_k) D_k^{-1}. \tag{3.19}$$

From this we can compute $(P_k(K)M^{-1}r^0, r^0)$ by noticing that

$$\tilde{z}^k = \frac{M^{1/2} z^k}{\sqrt{(z^k, r^k)}} = \frac{M^{1/2} z^k}{\delta_k}. \tag{3.20}$$

Therefore, since $\tilde{z}^0 = \tilde{Z}_k e_1$ where e_1 is the first column of the identity matrix of order $k+1$,

$$\begin{aligned}
(P_k(K)M^{-1}r^0, r^0) &= (P_k(K)z^0, Mz^0), && (3.21)\\
&= \delta_0^2 (M^{1/2} P_k(K) M^{-1/2} \tilde{z}^0, \tilde{z}^0), && (3.22)\\
&= \delta_0^2 (\tilde{Z}_k^T M^{1/2} P_k(K) M^{-1/2} \tilde{Z}_k e_1, e_1), && (3.23)\\
&= (z^0, r^0)(D_k P_k(T_k) D_k^{-1} e_1, e_1). && (3.24)
\end{aligned}$$

Now it is useful to look at the recurrences which are verified by the vectors $\tilde{z}^k$. We know (see chapter 6 of [9]) that the vectors z^k satisfy a three–term recurrence relation:

$$z^{k+1} = z^{k-1} - \theta_{k+1}(\zeta_k M^{-1} A z^k - z^k + z^{k-1}). \tag{3.25}$$

This gives

$$\delta_{k+1} \tilde{z}^{k+1} = \delta_{k-1} \tilde{z}^{k-1} - \theta_{k+1}(\delta_k \zeta_k \hat{K} \tilde{z}^k - \delta_k \tilde{z}^k + \delta_{k-1} \tilde{z}^{k-1}), \tag{3.26}$$

or

$$\hat{K} \tilde{z}^k = -\frac{\delta_{k+1}}{\theta_{k+1} \zeta_k \delta_k} \tilde{z}^{k+1} + \frac{1}{\zeta_k} + \frac{\delta_{k-1}(1-\theta_{k+1})}{\theta_{k+1} \zeta_k \delta_k} \tilde{z}^{k-1}. \tag{3.27}$$

In the Lanczos algorithm applied to $\hat{K}$ we have

$$\hat{K} q^k = \eta_{k+1} q^{k+1} + \epsilon_k q^k + \eta_k q^{k-1}, \tag{3.28}$$

where $\epsilon_k = (q^k, \hat{K}q^k)$ and $\eta_k = (q^{k-1}, \hat{K}q^k)$. In PCG we have (see [9])

$$\zeta_k = \frac{(z^k, Mz^k)}{(z^k, Az^k)}. \tag{3.29}$$

This implies that

$$\frac{1}{\zeta_k} = (\tilde{z}^k, \hat{K}\tilde{z}^k). \tag{3.30}$$

Moreover (see [9])

$$\theta_{k+1} = \frac{(z^{k-1}, r^{k-1})}{\zeta_k(z^{k-1}, Az^k) + (z^{k-1}, r^{k-1})}. \tag{3.31}$$

From this we can show that

$$\rho_k = \frac{\delta_{k-1}(1-\theta_{k+1})}{\theta_{k+1}\zeta_k\delta_k} = (\tilde{z}^{k-1}, \hat{K}\tilde{z}^k). \tag{3.32}$$

It is also easy to show that

$$-\frac{\delta_{k+1}}{\theta_{k+1}\zeta_k\delta_k} = \rho_{k+1}. \tag{3.33}$$

By looking at the first two iterates, this shows that the vectors $\tilde{z}^k$ are the Lanczos iterates q^k. From (3.3) we have

$$\chi_{k+1}\tilde{z}^{k+1} = [I - \hat{K}P_k(\hat{K})]\tilde{z}^0, \tag{3.34}$$

where

$$\chi_{k+1} = \frac{\delta_{k+1}}{\delta_0} = \frac{\sqrt{(r^{k+1}, z^{k+1})}}{\sqrt{(r^0, z^0)}}. \tag{3.35}$$

Therefore,

$$\frac{1}{\chi_{k+1}}[I - \hat{K}P_k(\hat{K})] \tag{3.36}$$

is the Lanczos polynomial of degree $k+1$ which we denote by $q_{k+1}(\hat{K})$. Let us write

$$q_{k+1}(\lambda) = q_{k+1}(0) - \lambda\bar{q}_k(\lambda). \tag{3.37}$$

Obviously, we have

$$q_{k+1}(0) = \frac{1}{\chi_{k+1}} \tag{3.38}$$

and

$$\bar{q}_k(\lambda) = \frac{1}{\chi_{k+1}} P_k(\lambda). \tag{3.39}$$

Being the Lanczos polynomial for $\hat{K}$, the polynomial q_{k+1} is proportional to the characteristic polynomial of J_{k+1} the Lanczos matrix. From the Cayley–Hamilton theorem we have $q_{k+1}(J_{k+1}) = 0$. Then,

$$q_{k+1}(0)I = J_{k+1}\bar{q}_k(J_{k+1}). \tag{3.40}$$

Multiplying by J_{k+1}^{-1} we obtain

$$J_{k+1}^{-1} = \frac{\bar{q}_k(J_{k+1})}{q_{k+1}(0)} = P_k(J_{k+1}). \tag{3.41}$$

It is also easily seen that

$$D_k T_k D_k^{-1} = J_{k+1}. \tag{3.42}$$

This shows that

$$(P_k(K)M^{-1}r^0, r^0) = (r^0, z^0)(P_k(J_{k+1})e_1, e_1) = (r^0, z^0)(J_{k+1}^{-1}e_1, e_1). \tag{3.43}$$

Using this result the modified version of the preconditioned conjugate gradient algorithm is:

Algorithm PCGQL

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

$$\alpha_{k-1} = \frac{{z^{k-1}}^T r^{k-1}}{{p^{k-1}}^T A p^{k-1}}, \tag{3.44}$$

$$\omega_k = \frac{1}{\alpha_{k-1}} + \frac{\beta_{k-1}}{\alpha_{k-2}}, \tag{3.45}$$

if $k = 1$ ——————————————————

$$f_1 = \frac{1}{\omega_1},$$

$$d_1 = \omega_1,$$

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

$$\underline{d}_1 = \omega_1 - b, \tag{3.46}$$

else

$$c_k = c_{k-1}\frac{\gamma_{k-1}}{d_{k-1}},$$

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

$$f_k = \frac{\gamma_{k-1}^2 c_{k-1}^2}{d_{k-1}(\omega_k d_{k-1} - \gamma_{k-1}^2)} = \frac{c_k^2}{d_k}, \tag{3.47}$$

$$\bar{d}_k = \omega_k - a - \frac{\gamma_{k-1}^2}{\bar{d}_{k-1}} = \omega_k - \bar{\omega}_{k-1}, \tag{3.48}$$

$$\underline{d}_k = \omega_k - b - \frac{\gamma_{k-1}^2}{\underline{d}_{k-1}} = \omega_k - \underline{\omega}_{k-1} \tag{3.49}$$

end

$$x^k = x^{k-1} + \alpha_{k-1} p^{k-1}, \tag{3.50}$$

$$r^k = r^{k-1} - \alpha_{k-1} A p^{k-1}, \tag{3.51}$$

$$M z^k = r^k, \tag{3.52}$$

$$\beta_k = \frac{{z^k}^T r^k}{{z^{k-1}}^T r^{k-1}}, \tag{3.53}$$

$$\gamma_k = \frac{\sqrt{\beta_k}}{\alpha_{k-1}}, \tag{3.54}$$

$$p^k = z^k + \beta_k p^{k-1}, \tag{3.55}$$

$$\bar{\omega}_k = a + \frac{\gamma_k^2}{\bar{d}_k}, \tag{3.56}$$

$$\underline{\omega}_k = b + \frac{\gamma_k^2}{\underline{d}_k}, \tag{3.57}$$

$$\breve{\omega}_k = \frac{\bar{d}_k \underline{d}_k}{\underline{d}_k - \bar{d}_k}\left(\frac{b}{\bar{d}_k} - \frac{a}{\underline{d}_k}\right), \tag{3.58}$$

$$\breve{\gamma}_k^2 = \frac{\bar{d}_k \underline{d}_k}{\underline{d}_k - \bar{d}_k}(b-a), \tag{3.59}$$

$$\bar{f}_k = \frac{\gamma_k^2 c_k^2}{d_k(\bar{\omega}_k d_k - \gamma_k^2)}, \tag{3.60}$$

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

$$\breve{f}_k = \frac{\breve{\gamma}_k^2 c_k^2}{d_k(\breve{\omega}_k d_k - \breve{\gamma}_k{}^2)}, \tag{3.61}$$

if $k > d$ ——————————————————

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

$$s_{k-d} = (r^0, z^0) t_k, \tag{3.62}$$

$$\bar{s}_{k-d} = (r^0, z^0)(t_k + \bar{f}_k), \tag{3.63}$$

$$\underline{s}_{k-d} = (r^0, z^0)(t_k + \underline{f}_k), \tag{3.64}$$

$$\breve{s}_{k-d} = (r^0, z^0)(t_k + \breve{f}_k) \tag{3.65}$$

end ——————————————————

In this algorithm s_{k-d}, $\underline{s}_{k-d}$ are lower bounds of $\|e^{k-d}\|_A^2$ and $\bar{s}_{k-d}$, $\breve{s}_{k-d}$ are upper bounds, a and b are lower and upper bounds of the smallest and largest eigenvalues of $M^{-1}A$. 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.

4. Numerical experiments

As test problems, we use two of the examples that were used in [5]. Example 1 and 2 arises from the 5–point finite difference approximation of a diffusion equation in a unit square,

$$-\mathrm{div}(a \nabla u)) = f, \tag{4.1}$$

with Dirichlet boundary conditions. Example 1 is the Poisson model problem, that is to say $a \equiv 1$. In Example 2 $a(x, y)$ is a diagonal matrix with equal

Table 1
Example 1, IC(1,1)

d	Gauss	Gauss–Radau	Gauss–Lobatto
1	2.24253	2.39271, 4.62705	6.32828
2	2.50579	2.5370, 3.15325	3.93472
3	2.56254	2.57011, 2.73920	2.93778
5	2.57958	2.57978, 2.58555	2.59949
10	2.58011	2.58011, 2.58011	2.58011

diagonal elements. This element is equal to 1000 in a square $]1/4, 3/4[\times]1/4, 3/4[$, 1 otherwise. For these two problems, we choose $n = 900$ (this corresponds to a 30×30 mesh), a right hand side such that the exact solution x_{ex} is $x_{ex} = (1, \ldots, 1)^T$ and a random initial guess x^0.

As preconditioners, we use the Incomplete Cholesky decomposition IC(1,1) and its modified version MIC(1,1) where the structure of the triangular factors is the same as the corresponding part of A (see [9]) as well as a sparse inverse suggested by Benzi, Meyer and Tuma (see [2]).

Let us consider first Example 1 and IC(1,1). The smallest eigenvalue $\lambda_{min}(M^{-1}A)$ is 0.0342, the largest one $\lambda_{max}(M^{-1}A) = 1.2045$. We ran PCGQL with different values of the delay d and choosing $a = 0.03$ and $b = 1.3$. Table 1 gives the computed bounds (multiplied by 10^9) at a given iteration where the exact A–norm of the error is $2.58011\ 10^{-9}$. One can see that the best bounds are given by the Gauss–Radau rule. A delay $d = 1$ already gives the order of magnitude of the error and excellent bounds are obtained with $d = 5$. We do not give plots of the error and the estimates as in [5] since the curves are indistinguishable on a logarithmic scale. In this example we need 46 iterations to have the A–norm of the error less than 10^{-12}.

Then, we use MIC(1,1). As it is well known theoretically, the smallest eigenvalue $\lambda_{min}(M^{-1}A)$ is 1. The largest eigenvalue $\lambda_{max}(M^{-1}A)$ is 9.0068. We choose $a = 1$ and $b = 9.5$. Table 2 gives the computed bounds (multiplied by 10^9) at a given iteration where the exact A–norm of the error is $6.87286\ 10^{-9}$. We have the same conclusions as for IC(1,1). The number of iterations is 36.

We now turn to using a sparse inverse AINV (see [2]) for which the dropping threshold is chosen to have almost the same number of non zeros as in A. The smallest eigenvalue $\lambda_{min}(M^{-1}A) = 0.0168$ and $\lambda_{max}(M^{-1}A) = 1.7058$. Here we choose $a = 0.015$ and $b = 1.8$. Table 3 gives the computed bounds (multiplied by

Table 2
Example 1, MIC(1,1)

d	Gauss	Gauss–Radau	Gauss–Lobatto
1	6.5243	6.60422, 6.99839	7.65775
2	6.82197	6.83503, 6.89233	6.96741
3	6.86437	6.86666, 6.87598	6.8871
5	6.87269	6.87273, 6.87292	6.87316
10	6.87286	6.87286, 6.87286	6.87286

Table 3
Example 1, AINV

d	Gauss	Gauss–Radau	Gauss–Lobatto
1	2.78502	2.9482, 8.52972	12.6287
2	3.06797	3.09879, 4.84314	6.75785
3	3.12475	3.13187, 3.60187	4.20977
5	3.14160	3.1420, 3.1731	3.2158
10	3.14193	3.14294, 3.14321	3.14341

10^{10}) at a given iteration where the exact A–norm of the error is $3.14294\ 10^{-10}$.

The results are a little bit worse than for the two previous preconditioners but they are still quite good. The reason for this is that the preconditioner was probably a little too sparse. In fact the number of iterations is twice the one for MIC(1,1) and greater than for IC. Keeping more non zero terms will improve upon that.

Figure 1 illustrates the use of the adaptive algorithm described in [8]. During the first PCG iterations we compute the smallest eigenvalue of J_k by inverse iteration (in fact we only do 10 iterations of inverse iteration at each PCG step). When we have decided that the smallest eigenvalue has converged (in practice when the relative change from the previous PCG iteration is less than 10^{-4}) we switch and take this estimate as the new value of a. Figure 1 shows the result for the Gauss–Radau upper bound with IC(1,1) when we start with $a = 10^{-10}$ and with $d = 2$. The switch occurred at iteration 14. To compare with the results of Table 1, the upper bound for Gauss–Radau is $3.07928\ 10^{-9}$ and the Gauss–Lobatto bound is $3.77352\ 10^{-9}$. Note they are slightly better than using the value $a = 0.03$ for the whole computation.

The chosen value for b does not matter too much. For instance, if we take $a = 0.03$ and $b = 10$ with $d = 2$, the lower bound for Gauss–Radau is $2.51269\ 10^{-9}$

Figure 1. PCG, Example 1, $d = 2$, log_{10} of A–norm of the error, dashed line: adaptive Gauss–Radau upper bound

Figure 2. PCG, Example 2, upper bounds, log_{10} of A–norm of the error, dashed line: $d = 2$, dash–dotted line: $d = 5$

Figure 3. PCG, Example 2, lower bounds, log_{10} of A–norm of the error, dashed line: $d = 2$, dash–dotted line: $d = 5$

and the upper bound for Gauss–Lobatto is $4.56235\ 10^{-9}$.

Example 2 is a little bit harder to solve, specially for low precisions as the error stagnates for a while. Using IC(1,1) we have to do 53 iterations to have the A–norm of the error less than 10^{-12}. We have $\lambda_{min}(M^{-1}A) = 7.11\ 10^{-5}$ and $\lambda_{max}(M^{-1}A) = 1.238$. Figure 2 shows the exact error and the results of the Gauss–Radau adaptive algorithm with $d = 2$ (dashed line) and $d = 5$ (dash–dotted line). We see that $d = 5$ gives excellent results. The switch for the smallest eigenvalue occurs at iteration 28. Figure 3 shows the Gauss–Radau lower bounds. At a given iteration where the exact error is $0.298541\ 10^{-3}$ we obtain (multiplied by 10^3) with $d = 5$, Gauss: 0.298521, Gauss–Radau: $0.298526, 0.400587$, Gauss–Lobatto: 0.679173.

We now use MIC(1,1) for Example 2. Then $\lambda_{min}(M^{-1}A) = 1$ and $\lambda_{max}(M^{-1}A) = 23.223$. We use $a = 1$ and $b = 30$. With $d = 2$, at a given iteration where the exact error is $1.63947\ 10^{-7}$ we obtain (multiplied by 10^7) Gauss: 1.61508, Gauss–Radau: $1.61821, 1.65132$ and Gauss–Lobatto: 1.67212.

When using AINV, it was recommended to symmetrically scale the matrix. So now the next results were obtained for the scaled system. In AINV by changing the threshold parameter we can obtain different approximate inverses which are more or less sparse. The number of non zeros in the lower triangular part of A is 2640. We ran a few experiments using different thresholds. The number of non zeros and the number of iterations are given in Table 4.

The computations were done using Matlab, so the computing time does not mean too much; but it was decreasing when going from case 1 to case 7. We use the adaptive algorithm starting with $a = 10^{-5}$ and $b = 2$ on cases 4 and 7 with $d = 5$. For case 4 the switch occurred at iteration 30. At iteration 45 the A–norm of the error is $4.81028\ 10^{-10}$. We obtained (multiplied by 10^{10}), Gauss: 4.81016, Gauss–Radau: $4.81020, 5.40958$, Gauss–Lobatto: 7.09893.

For case 7 the switch occurred at iteration 10. At iteration 13 the A–norm

Table 4
Example 2, AINV

case	threshold	nnz	λ_{min}	λ_{max}	nb. iter
1	0.25	2652	$3.78\ 10^{-5}$	1.771	79
2	0.2	3271	$4.61\ 10^{-5}$	1.783	74
3	0.15	3613	$5.14\ 10^{-5}$	1.824	70
4	0.1	6883	$1.14\ 10^{-4}$	1.699	52
5	0.07	10017	$1.59\ 10^{-4}$	1.713	41
6	0.05	14485	$3.04\ 10^{-4}$	1.699	36
7	0.01	51881	$2.28\ 10^{-3}$	1.648	18

of the error is $8.79542\ 10^{-8}$. We obtained (multiplied by 10^8), Gauss: 8.79542, Gauss–Radau: $8.79542, 8.79542$, Gauss–Lobatto: 8.79543.

It should be noted that due to the eigenvalue distributions using good preconditioners the values of the delay d needed to obtain good bounds are much less than for the non–preconditioned case, see the results in [8]. Moreover, the better the preconditioner, the better are the bounds and the smaller could be the value of the delay d. In fact the Gauss lower bound (which is not dependent on the eigenvalue estimates) already gives with $d = 1$ a very reliable stopping criterion for PCG.

References

[1] Z. BAI and G.H. GOLUB, Bounds for the trace of the inverse and the determinant of symmetric positive definite matrices, Ann. Numer. Math. 4, (1997), pp 29–38

[2] M. BENZI, C.D. MEYER and M. TUMA, A sparse approximate inverse preconditioner for the conjugate gradient method, SIAM J. Sci. Comput., vol 17, (1996), pp 1135–1149

[3] B. FISCHER & G.H. GOLUB, On the error computation for polynomial based iteration methods, Report NA 92–21, Stanford University (1992)

[4] G.H. GOLUB & G. MEURANT, Matrices, moments and quadrature, in Numerical Analysis 1993, D.F. Griffiths & G.A. Watson, Eds. Pitman Research Notes in Mathematics, v 303, (1994), pp 105–156

[5] G.H. GOLUB & G. MEURANT, Matrices, moments and quadrature II or how to compute the norm of the error in iterative methods, BIT, v 37 no 3, (1997), pp 687–705

[6] G.H. GOLUB and C. VAN LOAN, Matrix computations, Johns Hopkins University Press, (1989)

[7] G.H. GOLUB & Z. STRAKOŠ, Estimates in quadratic formulas, Numerical Algorithms v 8 no II–IV, (1994)

[8] G. MEURANT, The computation of bounds for the norm of the error in the conjugate gradient algorithm, Numerical Algorithms v 16, (1997), pp 77–87

[9] G. MEURANT, *Computer solution of large linear systems,* North–Holland, (1999)