NORGES TEKNISK-NATURVITENSKAPELIGE
UNIVERSITET

A Stochastic Approach to Error Estimates for Iterative Linear Solvers, Part 2

by

Gene H. Golub and Hallgeir Melbø

PREPRINT
NUMERICS NO. 4/2001

NORWEGIAN UNIVERSITY OF SCIENCE AND
TECHNOLOGY
TRONDHEIM, NORWAY

This report has URL
`http://www.math.ntnu.no/preprint/numerics/2001/N4-2001.ps`
Address: Department of Mathematical Sciences, Norwegian University of Science and Technology, N-7491 Trondheim, Norway.

A STOCHASTIC APPROACH TO ERROR ESTIMATES FOR ITERATIVE LINEAR SOLVERS PART 2

G.H. GOLUB[1] AND H. MELBØ[2]

[1] *Computer Science Department, Stanford University, Stanford CA 94305, USA. email: golub@sccm.stanford.edu*

[2] *Department of Mathematics, Norwegian University of Science and Technology 7491 Trondheim, Norway. email: hallgeir@math.ntnu.no*

Abstract.

We consider the problem of estimating the magnitude of the error in the Conjugate Gradient method after k iterations. Assuming that the initial error can be described using a probability distribution we derive l_2-estimates for the magnitude of the error in the average case. In Part 1 of this article the main ideas were presented and applied to a simple splitting method, while this part extends the same ideas to the Conjugate Gradient method.

AMS subject classification: 65F20.

Key words: Stochastic error estimates, Conjugate Gradient method

1 Introduction.

Reliable estimates for the size of the error is relevant to devising a good stopping criterion for iterative linear solvers. The usual way of stopping is to calculate the residual and then stop when some function of the norm of the residual is small. Normally, this is a good stopping criterion if we are interested in a solution that approximately satisfies the equation. However, such a solution does not necessarily give a good approximation to the exact solution. Indeed, if the condition number of the matrix is large, then it is possible that the magnitude of the residual is not a good measure of the size of the error; the latter may be much larger.

One possibility for estimating the size of the error is to use classical upper bounds for the norm of the error vector, derived from the spectral radius of the matrix. Unfortunately, these upper bounds typically over-estimate the actual error.

Part 1 of this article described how to make a statistically based estimator for the error in a simple splitting method associated with a symmetric linear system. It also showed how to evaluate the expected value and the variance of the estimator. In this part we will show how the ideas can be extended to the Conjugate Gradient method. We will in other words show how statistical estimates can be used to estimate the l_2-norm of the error, in the *average case*, after k iterations with the Conjugate Gradient method (CG). We will also briefly look at how this can be done if a preconditioner is applied.

Our analytical observations will be accompanied by numerical examples.

2 CG as polynomials of a matrix

The most common way of writing up the Conjugate Gradient algorithm is to express the new approximate solution as the solution from the previous iteration plus a coefficient times a direction found in the previous iteration. It is however also possible to express it in quite a different form, as the three term recurrence relation shown in Concus et.al.[4], which is the formulation we will be using. Consider the equation

$$Ax = b \ ,$$

where A can be split into $M - N$, A and M are SPD and N is symmetric. We will think of M^{-1} as a preconditioner, meaning that

$$Mz = d$$

is easy to solve and that $M^{-1}A$ is in some sense close to the identity. In the unpreconditioned case we simply set $M = I$. Using the formulation in Golub and Van Loan [7] we then get the following version of Conjugate Gradients:

CG, 3-term recurrence

$x_{-1} = 0,\ x_0 =$ initial guess, $k = 0,\ r_0 = b - Ax_0$
while $r_k \neq 0$
    $k = k + 1$
    Solve $Mz_{k-1} = r_{k-1}$ for z_{k-1}.
    $\gamma_{k-1} = z_{k-1}^T M z_{k-1} / z_{k-1}^T A z_{k-1}$
    if $k = 1$
        $\omega_1 = 1$
    else
        $\omega_k = \left(1 - \frac{\gamma_{k-1}}{\gamma_{k-2}} \frac{z_{k-1}^T M z_{k-1}}{z_{k-2}^T M z_{k-2}} \frac{1}{\omega_{k-1}}\right)^{-1}$
    end
    $x_k = x_{k-2} + \omega_k(\gamma_{k-1} z_{k-1} + x_{k-1} - x_{k-2})$
    $r_k = b - Ax_k$
end
$x = x_k$

Using this formulation and the relation

$$Mz_k = r_k = A(x - x_k) = Ae_k \ ,$$

we may subtract the exact solution, x, from both sides of the x_k update in the algorithm and obtain

$$\begin{aligned} e_k &= e_{k-2} + \omega_k \Big(\gamma_{k-1}(-M^{-1}A)e_{k-1} + e_{k-1} - e_{k-2} \Big) \\ &= (1-\omega_k)e_{k-2} + \omega_k(I - \gamma_{k-1}M^{-1}A)e_{k-1} \ , \end{aligned} \tag{2.1}$$

with

$$e_1 = (I - \gamma_0 M^{-1} A)e_0 \ .$$

As we can see from the expression, e_k is a k'th degree polynomial in $M^{-1}A$ multiplied by e_0, or in the unpreconditioned case a polynomial in A multiplied by e_0. The unpreconditioned case will be considered first, which means we can write

$$e_k = p_k(A)\, e_0 \ ,$$

with

$$p_0(A) = I \quad \text{and} \quad p_1(A) = I - \gamma_0 A \ .$$

3 Estimates for unpreconditioned CG

To derive the error estimates for CG, we will proceed in exactly the same way as was done in Part 1 of this article. The main difference is that the error after iteration k is now expressed as a polynomial that is *not* known in advance in contrast to the polynomial λ^{2k} which was known in advance for the simple solver.

As we assume A to be SPD, which is the standard requirement for CG to converge, we can write

$$A = Q\Lambda Q^T$$

just as was done for the matrix B in Part 1. Further, the initial error e_0 can be spanned as

$$e_0 = Q\hat{\alpha} = \sum_{i=1}^{n} \alpha_i q_i \ .$$

The α_i's are now assumed to be identically distributed random variables arising from some probability distribution. When there can be no confusion we will therefore only write α instead of α_i. The l_2-norm of the error can then be expressed as

$$\begin{aligned} \|e_k\|^2 &= (e_k, e_k) \\ &= (p_k(A)e_0, p_k(A)e_0) \\ &= e_0^T Q p_k(\Lambda) Q^T Q p_k(\Lambda) Q e_0 \\ &= \hat{\alpha}^T p_k^2(\Lambda)\hat{\alpha} \\ &= \sum_{i=1}^{n} \alpha_i^2 p_k^2(\lambda_i) \ . \end{aligned}$$

For simplicity we have not distinguished between the p_k that takes a matrix as argument and the p_k that takes a scalar as argument. Similar to the simple case discussed in Part 1 this leaves us with

$$E(\|e_k\|^2) \;=\; E(\alpha^2)\sum_{i=1}^{n} p_k^2(\lambda_i) \tag{3.1}$$

$$Var(\|e_k\|^2) \;=\; Var(\alpha^2)\sum_{i=1}^{n} p_k^4(\lambda_i) \;+\; 2Cov(\alpha_i^2,\alpha_j^2)\sum_{i<j} p_k^2(\lambda_i)p_k^2(\lambda_j)\;. \tag{3.2}$$

As in Part 1 we will be looking at two different distributions for the α's, a normal distribution $N(0,\sigma^2)$, in which the covariance is zero, and a Dirichlet distribution in which the covariance is nonzero. In this case the covariance term is however small, and as was shown in Part 1 it can be neglected. The normal distribution is the one most suited for practical cases, but for tests where lots of different initial vectors are chosen the Dirichlet distribution will be very convenient. In the case of $\alpha \sim N(0,\sigma^2)$ we have

$$\begin{aligned} E(\alpha^2) &= \sigma^2 \\ Var(\alpha^2) &= 2\sigma^4 \\ Cov(\alpha_i^2,\alpha_j^2) &= 0 \qquad i \neq j. \end{aligned} \tag{3.3}$$

If the α's are distributed according to a Dirichlet distribution ($(\alpha_1^2,...,\alpha_n^2) \sim \mathcal{D}(1/2,...1/2)$) it means that the α's are found as

$$\alpha_i = \frac{\beta_i}{\sqrt{\sum_{j=1}^{n}\beta_j^2}} \qquad i = 1...n. \tag{3.4}$$

with $\beta_i \sim N(0,\sigma^2)$. This gives a way of normalising the initial error such that different initial errors are comparable. If the covariance term is neglected we then have

$$\begin{aligned} E\left(\alpha^2\right) &= \frac{1}{n} \\ Var\left(\alpha^2\right) &= \frac{2\,(n-1)}{n^2\,(n+2)} \\ Cov\left(\alpha_i^2\,,\alpha_j^2\right) &\approx 0 \qquad i \neq j. \end{aligned} \tag{3.5}$$

To estimate the sums we need to evaluate the polynomials. One possibility is to build the polynomial by storing the coefficients for each power of the matrix A in equation (2.1). However, this turns out to be a very unstable process. The following is a more stable procedure.

From equation (2.1) and the fact that $p_k(A)e_0 = Qp_k(\Lambda)\hat{\alpha}$ we get the relation

$$\begin{aligned} e_k = \; p_k(A)e_0 &= (1-\omega_k)p_{k-2}(A)e_0 \;+\; \omega_k(I-\gamma_{k-1}A)p_{k-1}(A)e_0 \\ &= Q\Big[(1-\omega_k)p_{k-2}(\Lambda) \;+\; \omega_k(I-\gamma_{k-1}\Lambda)p_{k-1}(\Lambda)\Big]\hat{\alpha}\;. \end{aligned}$$

The matrix Q cancels if we multiply the system by Q^T from the left, and since Λ is nonsingular and the equation holds for all $\hat{\alpha}$'s, the system decouples and we obtain the following expression for the polynomial,

$$p_k(x) = (1-\omega_k)p_{k-2}(x) + \omega_k(1-\gamma_{k-1}x)p_{k-1}(x) \ .$$

Initial values are

$$p_0 = 1 \quad \text{and} \quad p_1 = 1 - \gamma_0 x \ .$$

To evaluate the polynomial, start with p_0 and p_1 evaluated at x and recur until the desired order is obtained.

4 Estimating the sums

As shown in Part 1, the sums in equations (3.1) and (3.2) can be written as integrals,

$$\sum_{i=1}^{n} p_k^m(\lambda_i) \quad = \quad \int_{\lambda_{min}}^{\lambda_{max}} \delta(\lambda - \lambda_j)\, p_k^m(\lambda)\, d\lambda = \int_{\lambda_{min}}^{\lambda_{max}} \omega\,(\lambda)\, g_m(\lambda)\; d\lambda \ . \tag{4.1}$$

Here $\omega\,(\lambda)$ is a weight function, $m = 1$ or 2, $g_m(\lambda)$ is the function to be integrated and $\delta\,(\lambda - \lambda_j)$ is Dirac's delta function which is nonzero each time the variable λ hits one of the eigenvalues of A. The integrals can then be approximated by a quadrature rule, but due to the unknown polynomials the error terms get very complex and it is now difficult to find upper and lower bounds for the integral. Such bounds were easily obtained for the simple method discussed in Part 1.

Another and much more serious problem is that for matrices that are not well conditioned the approximation of the integrals may be very poor. What happens when CG converges is that the polynomials become zero when evaluated at the eigenvalues. How these high order polynomials behave *away* from the eigenvalues, however, may change a great deal from one matrix to the other. This is clearly seen in Figure 4.1 and 4.2 where the polynomials and the eigenvalues from two different cases are plotted. The problem solved is

$$Ax = 0,$$

where a fixed initial guess is used. In both examples the matrix is

$$A = K\bar{A}K^T \ , \qquad A \in \mathbf{R}^{100\times 100} \ ,$$

where $\bar{A}$ denotes the 5 point discretisation of the two dimensional Laplacian on a square, using finite differences, and K is a diagonal matrix. In the first case K is the identity and in the second case

$$K = diagonal(1, \cdots, 1, 4, \cdots, 4)$$

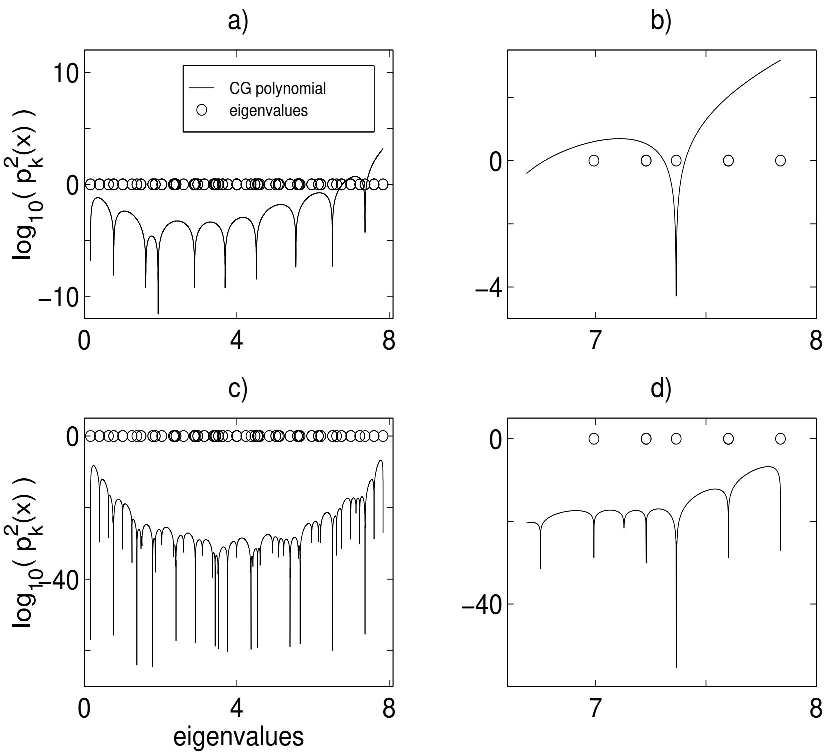

Figure 4.1: The CG polynomial from the first example plotted after 10 and 70 iterations. Plot *a*) and *b*) show the CG polynomial after 10 iterations, where *b*) is just a zoom-in on *a*). Similarly plot *c*) and *d*) show the polynomial after 70 iterations. Note the logarithmic axes.

with half of the entries in K being unity. From the plots we can clearly see that in the second case evaluation of the polynomials outside the eigenvalues would give totally wrong estimates even if it may work well in the first case. Different integration methods have been tested out, but since it is probably too expensive to integrate the polynomials exactly we have obtained the best results by simply averaging the polynomials evaluated in the eigenvalues that we assume are known.

As the extreme eigenvalues can be well estimated at no extra cost we assume that n_0 eigenvalues are known. Letting n denote the total number of eigenvalues, the averaging formula for estimating the integrals, or rather the sums, reads

$$I_{Avg} = \frac{n}{n_0} \sum_{i=1}^{n_0} p_k^m(\lambda_{(i)}) , \quad m = 2 \text{ or } 4 .$$

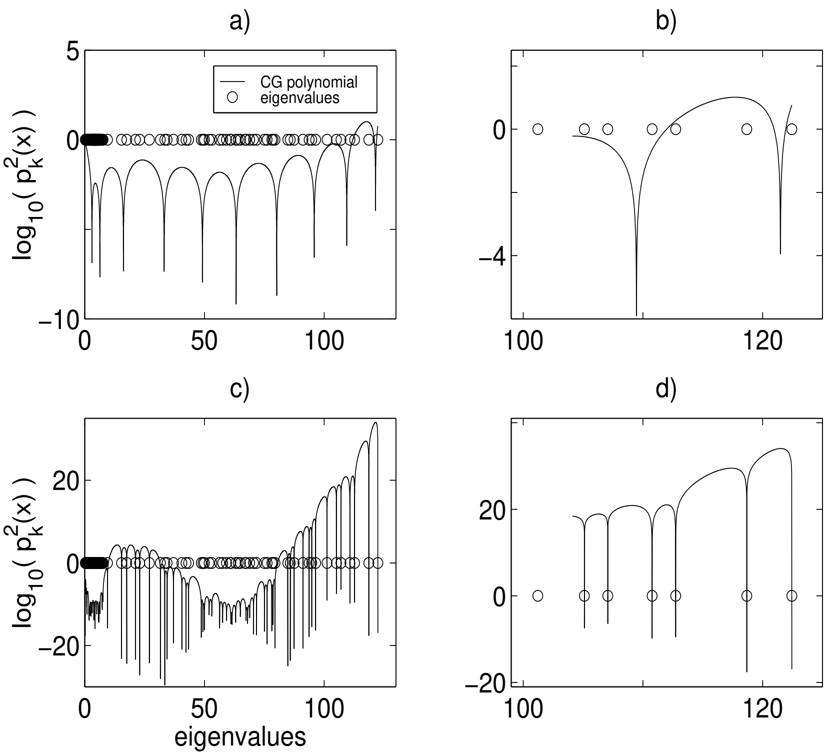

Figure 4.2: The CG polynomial from the second test case plotted after 10 and 70 iterations. Plot *a*) and *b*) show the CG polynomial after 10 iterations, where *b*) is just a zoom-in on *a*). Similarly plot *c*) and *d*) show the polynomial after 70 iterations. Note the logarithmic axes.

Compared to other formulas, averaging is not a very accurate integration method for low order polynomials, but the loss of accuracy during the first few iterations is very small compared to what seems to be gained for high order polynomials. As we are only interested in the order of magnitude of the error, averaging seems to be the best choice. What we are suggesting here will obviously be more accurate when more eigenvalues are estimated. However, as has already been seen, it is important that the estimated eigenvalues used are very close to exact.

In Figure 4.3 we can see the convergence plots for the two examples when the averaging rule is applied to estimate the two sums. Only the smallest and largest eigenvalue of A are assumed to be known. The plots show the average results over 40 runs with different initial vectors. To make all initial vectors have length one they have been constructed by drawing α's from a Dirichlet distribution and explicitly multiplying them with the eigenvectors of A. In the first case we got slightly better results with a Gauß-Radau rule than with averaging, but in the

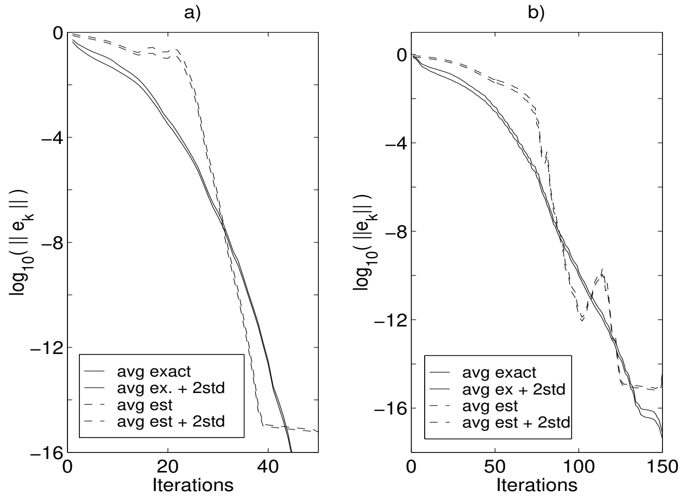

Figure 4.3: The two test cases for CG with averaging used as integration rule. Plot a) shows the results from the first test example and plot b) shows the results for the second example. The plots show the average error and the average error plus 2 standard deviations over the 40 runs for both the exact error and the estimated error. For the estimated error the average of the estimated expected value is used. The standard deviation is computed as the average of the square root of the estimated variance.

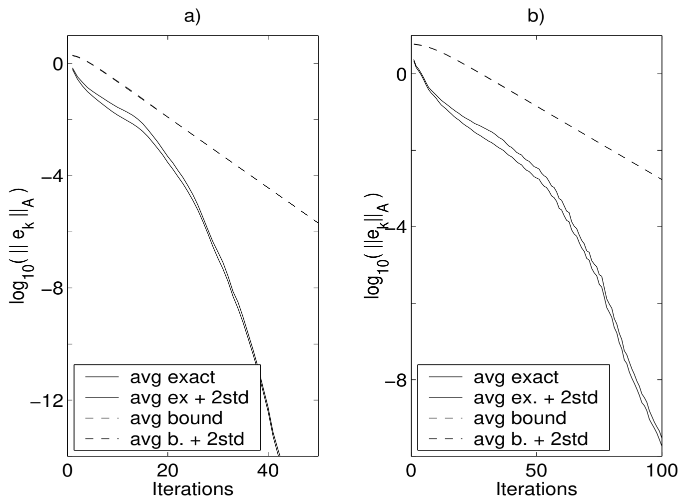

Figure 4.4: Plot a) shows the upper bound measured in A-norm for the first test example. Plot b) shows the same bound for the second example.

second case a Gauß-Radau rule of order 2 will not show convergence. This is in accordance with Figure 4.2.

Figure 4.4 shows the average upper bound for the Conjugate Gradient method in these two examples. The upper bound is computed as (see e.g. [7])

$$\|e_k\|_A \leq 2 \left[\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right]^k \|e_0\|_A \ , \tag{4.2}$$

where $\kappa = \lambda_{max}/\lambda_{min}$. Since this bound is given as the A-norm of the error, everything in the plot is measured in that norm and it is therefore not directly comparable to Figure 4.3. However, one can clearly see that the upper bound is a poor estimator for the error. Even if the l_2-norm of the initial vector is unity, its A-norm may change from one run to the other. That explains why we take averages also in this case.

5 Estimates for preconditioned CG

In practice the method of conjugate gradients is often used with a preconditioner and we would like to make the estimator work also in the preconditioned case. The problem with using preconditioned CG as it is usually derived, is that it leads to polynomials in the matrix $M^{-1}A$, which may not be symmetric. In general that means $M^{-1}A \neq Q\Lambda Q^T$, and the previous arguments do not work. It is however possible to get around this problem.

Consider the equation

$$Ax = b \ ,$$

where the preconditioner $M = LL^T$ is assumed to be known and A can be split into

$$A = M - N.$$

If we apply a split preconditioner, the system can be written as

$$\tilde{A}u = \tilde{b} \ ,$$

where $u = L^T x,\ \tilde{b} = L^{-1}b$ and

$$\tilde{A} = L^{-1}AL^{-T} = I - \tilde{N},$$

with $\tilde{N} := L^{-1}NL^{-T}$. We may now use the same conjugate gradient algorithm as before on this system, and since $\tilde{A}$ is symmetric we may write

$$\tilde{A} = \tilde{Q}\tilde{\Lambda}\tilde{Q}^T \ ,$$

and thereby the same estimates apply with the $\tilde{\lambda}$'s in the resulting formulas being $\tilde{\lambda}(\tilde{A})$.

Although we need split preconditioning to get through the argument, it is well known that left preconditioning and split preconditioning are equivalent, so in practice we can use the standard CG algorithm. This is easily seen from the following:

Using a similarity transform immediately shows that $M^{-1}A$ and $\tilde{A}$ have the same eigenvalues,

$$\lambda(M^{-1}A) = \lambda(L^{-T}L^{-1}A) = \lambda(L^{-1}AL^{-T}) = \tilde{\lambda}(\tilde{A}) \; .$$

To see that the CG polynomials are also the same, first note that $\tilde{r}_k = L^{-1}b - L^{-1}AL^{-T}u = L^{-1}r_k$, and that $Mz_k = LL^Tz_k = r_k$. From the algorithm "CG, 3-term recurrence" in Section 2 we then get

$$\begin{aligned}
\gamma_k &= \frac{(z_{k-1}, Mz_{k-1})}{(z_{k-1}, Az_{k-1})} \\
&= \frac{(M^{-1}r_{k-1}, r_{k-1})}{(M^{-1}r_{k-1}, AM^{-1}r_{k-1})} \\
&= \frac{(L^{-T}\tilde{r}_{k-1}, L\tilde{r}_{k-1})}{(L^{-T}\tilde{r}_{k-1}, AL^{-T}\tilde{r}_{k-1})} \\
&= \frac{(\tilde{r}_{k-1}, \tilde{r}_{k-1})}{(\tilde{r}_{k-1}, \tilde{A}\tilde{r}_{k-1})} \\
&= \tilde{\gamma}_k \; .
\end{aligned}$$

Once again doing the same manipulation as was done in the numerator in the above equations, we immediately get $\omega_k = \tilde{\omega}_k$, which shows that the polynomials are identical, and therefore split preconditioning and left preconditioning give exactly the same method.

The only difference from the unpreconditioned case turns out to be that we are no longer estimating $\|e_k\|_2^2$ but rather $\|e_k\|_M^2$. Since we are solving $\tilde{A}u = \tilde{b}$ with $\tilde{A} = \tilde{Q}\tilde{\Lambda}\tilde{Q}^T$ and $\tilde{e}_0 = \tilde{Q}\tilde{\alpha}$, we get (as in Section 2),

$$\tilde{e}_k = (1-\omega_k)\tilde{e}_{k-2} + \omega_k(I - \gamma_{k-1}\tilde{A})\tilde{e}_{k-1} \; ,$$

and

$$\begin{aligned}
\|\tilde{e}_k\|^2 &= \sum_{i=1}^{n} \tilde{\alpha}_i^2 \tilde{p}_k^2(\tilde{\lambda}_i) \, , \\
\tilde{e}_k &= u - u_k = L^T(x - x_k) = L^Te_k \, , \\
\|\tilde{e}_k\|^2 &= (\tilde{e}_k, \tilde{e}_k) \\
&= (L^Te_k, L^Te_k) \\
&= (e_k, LL^Te_k) \\
&= \|e_k\|_M^2 \; .
\end{aligned}$$

The expected value and the variance of $||\tilde{e}_k||$ can then be computed as previously described.

6 Numerical test for preconditioned CG

To test the preconditioned version we will solve

$$Ax = 0 \ ,$$

where

$$A = K\bar{A}K^T \ , \qquad A \in \mathbf{R}^{n\times n} \ , \quad n = 225 \, ,$$

and $\bar{A}$ is again the finite difference discretisation of the two dimensional Laplacian on a square. The matrix K is chosen as

$$K = \mathit{diagonal}\,(0.1, \ldots, 0.1 \, , \, 1, \ldots, 1 \, , \, 10, \ldots, 10).$$

The length of the argument vector for the diagonal matrix is n, and the two first parts of the vector have $n/4$ elements each while the last part has $n/2$ elements. As preconditioner we use incomplete Cholesky factorisation with zero fill-in, reducing the condition number from $\kappa = 3.3 \cdot 10^5$ to $\kappa = 9.95$, which

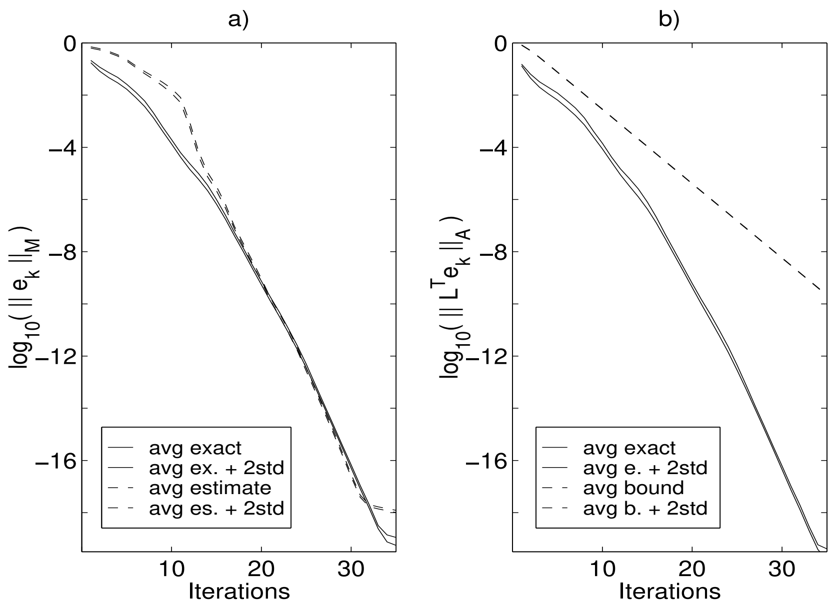

Figure 6.1: Experiment with preconditioned CG. Plot a) shows the average estimated error together with the average exact error over the 40 runs. "std" means "standard deviation". In plot b) the upper bound is compared to the actual error, both measured in A-norm.

makes CG a lot more effective. We do 40 runs where the α's are drawn from a standard Dirichlet distribution. The results are shown in Figure 6.1, and the estimator clearly works very well in this case.

7 Numerical experiments, single run

Finally we will test the estimator on two realistic examples. The first test is an unpreconditioned case while the second test includes preconditioning.

In the unpreconditioned case the matrix is again the two dimensional Laplacian, which has already been described, but this time with a nonzero right-hand side. Two of the entries in the solution vector x are set to 1, while the shadow cells outside the boundaries are zero. This means the problem is a two dimensional heat equation with two sources in the interior of the domain and with homogeneous Dirichlet boundary conditions. The mesh is 30×30 nodes, making the size of the matrix 900×900 and its condition number $\kappa = 328$. As initial vector we use a vector of all zeros. There are no sources in the domain, except for the two fixed points, so the solution takes on a maximum value of 1 in the interior and the constant value 0 along the boundary. The exact solution will therefore be in the interval $[0, 1]$. We assume the α's to be normally distributed, and since our initial vector is all zeros, it seems reasonable to guess that $\alpha \sim N(0,1)$. In this example we assume that the two smallest and the two largest eigenvalues are known.

With these assumptions and guesses we get the result shown in Figure 7.1. Plot *a*) shows the computed solution, and as previously plots *b*) and *c*) show how good the estimate is and how tight the upper bound is, respectively. There is now only one run, so there are no averages taken. We can see that even if the estimated value is not as close to the true error as in the tests with average values, we still get quite a good estimate.

Turning to the preconditioned case, we use a larger version of the same matrix, adding interior interfaces. Now the mesh is 50×50, so $A \in \mathbf{R}^{2500 \times 2500}$, and the domain is partitioned into 3 strips of different conductivity. The first strip has conductivity 10, the second has conductivity 1 and the third strip has conductivity 100. The two fixed points in the interior now fall inside the two domains with the highest conductivity, and this time we are discretising the true heat equation instead of multiplying by a diagonal matrix from each side to form the interfaces.

This problem produces a matrix with condition number $\kappa = 2.4 \cdot 10^4$. Incomplete Cholesky factorisation with zero fill-in is used as a preconditioner, and the preconditioned matrix has condition number $\kappa = 50$. The boundary conditions are the same as in the unpreconditioned case and the initial guess, x_0, is the vector of all zeros. Now the α's actually describe the "preconditioned" initial error $L^T e_0$, or equivalently the initial error measured in M-norm, which makes it

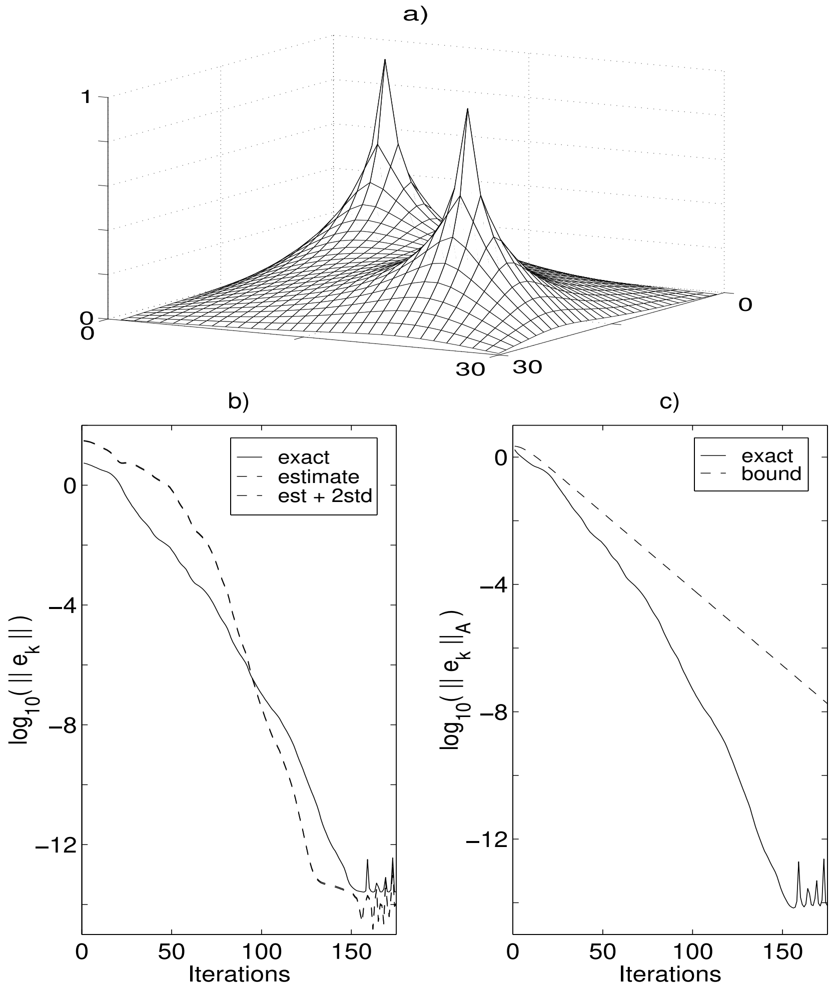

Figure 7.1: Experiment with unpreconditioned CG, single run. Plot *a*) shows the computed solution to the problem. In plot *b*) the estimated error is shown together with the exact error. "std" means "standard deviation". Note that the estimated variance is negligible. In plot *c*) the upper bound is compared to the actual error, both measured in A-norm.

harder to make an intuitive guess for the probability distribution. We first make a rough guess of a smaller variance than last time, choosing $\alpha \sim N(0, 0.25^2)$.

Figure 7.2 shows the resulting solution and estimates. In plot *a*) is shown the computed solution, and the interfaces can be seen clearly. What is more interesting, however, is that the estimate in plot *b*) lies very close to the actual error.

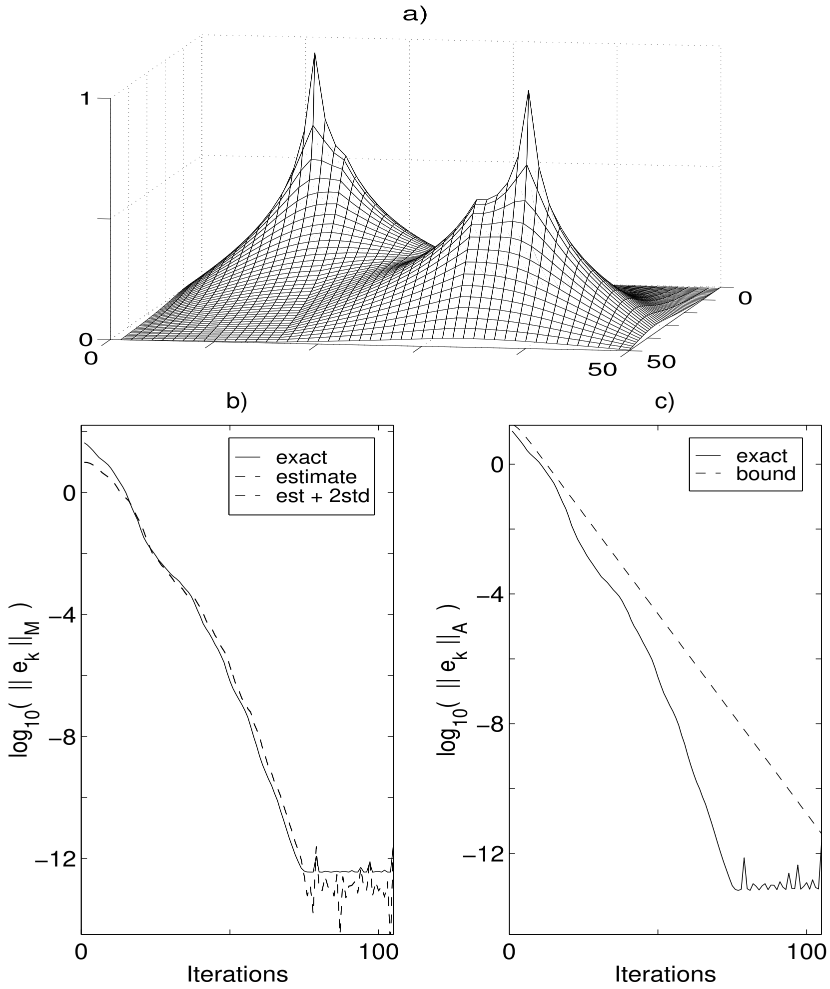

Figure 7.2: Experiment with preconditioned CG, single run. Plot a) shows the computed solution to the problem. In plot b) the estimated error is shown together with the exact error. "std" means "standard deviation". In plot c) the upper bound is compared to the actual error, both measured in A-norm. Note the label on the vertical axis in plot b), $\|\tilde{e}_k\|_{\tilde{A}} = \|e_k\|_A$.

For this last example we have also performed a robustness test with regard to the guessed standard deviation of the α's. The exact same experiment was run, first increasing the guess for σ by a factor 10 compared to the initial guess, and later decreasing the guess by a factor 10. That means we are guessing $\alpha \sim N(0, 2.5^2)$ in the first case and $\alpha \sim N(0, 0.025^2)$ in the second case. The result is shown in Figure 7.3. The result is as expected since σ only takes part in the constant in front of the estimated sum. The estimate is therefore not very vulnerable to

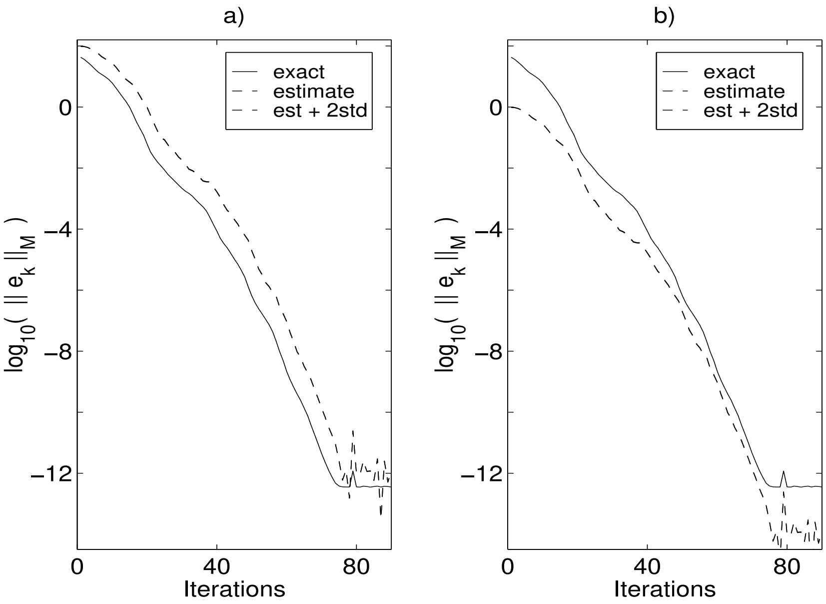

Figure 7.3: Over- and under estimating the variance relative to the initial guess. In plot *a*) the guess is $\alpha \sim N(0\,,\,2.5^2)$, while in plot *b*) the guess is $\alpha \sim N(0\,,\,0.025^2)$.

the guessed variance.

Note that in all our test cases except the "single run for unpreconditioned CG", we have only assumed λ_{min} and λ_{max} to be known, while in practice we usually have very good estimates for more eigenvalues, which makes the estimation of the integral a lot more accurate and robust. Another point worth noticing is that every time the upper bound has been plotted we have used the exact value of $\|e_0\|_A$ in equation (4.2). In practice we do not know this value, and it has to be guessed. On the other hand, guessing it within an order of magnitude is usually not too hard.

It is easy to construct test matrices where the upper bound does much worse than in this last example, while the estimate does about as well as here, but as we have to guess the distribution of the α's, we have chosen test cases where the equation has a physical interpretation which makes it easy to come up with a reasonable guess for the standard deviation.

REFERENCES

1. Z.Bai and G.H.Golub. *Bounds for the Trace of the Inverse and the Determinant of Symmetric Positive Definite Matrices.* SCCM-96-06, Stanford University. http://www-sccm.stanford.edu/reports.html#start-1996
2. C.Cheney. *Introduction to Approximation Theory.* Mc Graw Hill, NY, 1966.
3. P.Concus and G.H.Golub. *A Generalized Conjugate Gradient Method for Nonsymmetric Systems of Linear Equations.* In Lecture Notes in Economics and Mathematical Systems, number 134, ed. M.Beckmann and H.P.Künzi, Springer-Verlag, NY, 1976.
4. P.Concus, G.H.Golub and D.P.O'Leary. *A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations.* In Sparse Matrix Computations, ed. J.R.Bunch and D.J.Rose, Academic Press, NY, 1976, pp 309-332.
5. P.J.Davis and P.Rabinowitz. *Methods of numerical integration.* Academic Press, 1984, pp 14-19, 109-112.
6. G.H.Golub and G.Meurant. *Matrices, moments and quadrature.* SCCM-93-07, Stanford University http://www-sccm.stanford.edu/reports.html#start-1993.
7. G.H.Golub and C.F.Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 3rd edition, 1996.
8. N.L.Johnson and S.Kotz. *Encyclopedia of statistical sciences, Volume 2.* John Wiley & Sons, 1982, pp 386-387.
9. Y.Saad. *Iterative methods for sparse linear systems.* PWS Publishing Company, Boston, MA, 1996.