A PARALLEL LANCZOS METHOD FOR SOLVING SYMMETRIC POSITIVE DEFINITE LINEAR SYSTEMS

GÉRARD MEURANT*

Abstract. In this paper we propose a Lanczos–like parallel method for solving symmetric positive definite linear systems. We use the connections of the Lanczos algorithm with orthogonal polynomials to construct a (non-orthogonal) basis of the Krylov subspace. Although this algorithm needs more inner products than the CG algorithm it offers much more potential for parallelism.

1. Introduction. The conjugate gradient (CG) algorithm is the iterative method of choice for solving sparse symmetric positive definite linear systems. However solving large sparse linear systems requires using parallel computers and CG is not so well suited for such architectures. There are many synchronization points in CG and the computation of two inner products per iteration can be also a bottleneck; see [16].

Several attempts have been made to modify CG to make it "more parallel". Some people tried to remove synchronization points as much as possible; see Van Rosendale [18], Chronopoulos and Gear [2], Meurant [14], d'Azevedo, Eijkhout and Romine [4]. Unfortunately, most of these variants are less stable than the classical CG algorithm. Another way of research was started by Fischer and Freund [7]. Their goal was to get rid of the inner products by using the spectral information that can be obtained after a few steps of CG or of the Lanczos algorithm (which is equivalent to CG).

In this paper we propose a Lanczos-like algorithm to solve symmetric positive definite linear systems. Although this method uses more storage than the Lanczos algorithm, it offers more opportunities for parallelism. It also allows to group together several consecutive iterations. This could help improving the speed of the matrix-vector product by computing at the same time several matrix-vector products like $Av, A^2v, \ldots$; see [3]. Minimizing communications and increasing parallelism in linear algebra algorithms have been a major concern recently with the advent of multicore architectures; see [1] and [5].

Here we use the same starting point as in Fischer and Freund [6] doing first a few iterations of the Lanczos algorithm to obtain an estimate of the spectrum of A or, to be more precise, of the piecewise constant measure associated with A and the starting residual vector. From this spectral information we compute another approximate piecewise constant measure which is generally closer to the exact one. Then we use algorithms to compute the coefficients of the three-term recurrence satisfied by the orthogonal polynomials associated to this approximate measure. These coefficients are used to generate a (non-orthogonal) basis of the Krylov subspace associated with A and the initial residual vector. The basis is then used to compute an approximation of the solution of the linear system. This methodology requires more storage and the computation of more inner products than the CG or Lanczos algorithms. However, it is more parallel since, after the initial Lanczos iterations, almost everything can be done in parallel.

The contents of the paper are the following. Section 2 recalls the Lanczos algorithm and the links from the coefficients of the algorithm to the orthogonal polynomials associated with a piecewise constant measure that depends on the matrix A and the initial vector. Section 3 describes how to obtain an approximate piecewise constant measure from the spectral information that can be gathered after a few Lanczos

*(gerard.meurant@gmail.com) started in Saint Aubin sur mer, July 2009, revised May 2010

initial iterations. Computing the coefficients of the three-term recurrence satisfied by the orthogonal polynomials associated with this approximate measure is an inverse eigenvalue problem. Section 4 uses a basis computed with these coefficients to compute an approximation of the solution of the linear system. To obtain an efficient solver we need to introduce preconditioning. This is done in section 5. Numerical experiments on a sequential computer are provided in section 6. Results on a parallel computer will be given in a forthcoming paper. Finally, section 7 gives some conclusions.

2. The Lanczos algorithm. Let A be a symmetric sparse nonsingular square matrix of order n. In the Lanczos algorithm, the Lanczos basis vectors v^k can be written as $v^k = p_k(A)v$ where $v = v^1$ is the starting vector (of unit l_2-norm) and p_k is a polynomial of order $k-1$. The Lanczos vectors are computed as

$$\eta_1 v^2 = Av^1 - \alpha_k v^1, \quad \eta_k v^{k+1} = Av^k - \alpha_k v^k - \eta_{k-1} v^{k-1}, \ k = 2, \ldots$$

From this definition, it is clear that the polynomials p_k satisfy a three-term recurrence

$$\eta_k p_{k+1}(\lambda) = (\lambda - \alpha_k) p_k(\lambda) - \eta_{k-1} p_{k-1}(\lambda), \quad p_0 \equiv 0, \, p_1 \equiv 1.$$

The coefficient α_k is determined to enforce the orthogonality condition $(v^k)^T v^{k+1} = 0$ and is given by an inner product

$$\alpha_k = (Av^k, v^k).$$

The coefficient η_k is computed to have $\|v^{k+1}\| = 1$. Theoretically it is also given by the orthogonality condition $(v^{k-1})^T v^{k+1} = 0$ which leads to

$$\eta_{k-1} = (Av^k, v^{k-1}).$$

The Lanczos recurrence computes an orthonormal basis of the Krylov subspace $\mathcal{K}(A, v) = \{v, Av, A^2v, \ldots\}$. If we collect the basis vectors v^j, $j = 1, \ldots, k$ as the columns of a matrix V_k we have the matrix relation

$$AV_k = V_k T_k + \eta_k v^{k+1} (e^k)^T,$$

where T_k is a symmetric tridiagonal matrix whose diagonal coefficients are the α_j's and the subdiagonal coefficients are the η_j's. The vector e^k is the k-th column of the identity matrix of order k.

Using the spectral decomposition of the symmetric matrix A, $A = Q\Lambda Q^T$ with Q the orthogonal matrix of the eigenvectors and Λ a diagonal matrix whose diagonal elements λ_j are the eigenvalues of A, the formula defining α_k is

$$\alpha_k = (Av^k, v^k) = (Ap_k(A)v, p_k(A)v) = (\Lambda p_k(\Lambda)\bar{v}, p_k(\Lambda)\bar{v}) = \sum_{i=1}^{n} \lambda_i [p_k(\lambda_i)]^2 (\bar{v}_i)^2,$$

with $\bar{v} = Q^T v$. The last sum can be written as a Riemann–Stieltjes integral

$$\alpha_k = \int_{\lambda_1}^{\lambda_n} \lambda [p_k(\lambda)]^2 \, d\sigma(\lambda).$$

The measure σ is given by

$$\sigma(\lambda) = \begin{cases} 0 & \text{if } \lambda < \lambda_1 \\ \sum_{j=1}^{i} [\bar{v}_j^1]^2 & \text{if } \lambda_i \le \lambda < \lambda_{i+1} \\ \sum_{j=1}^{n} [\bar{v}_j^1]^2 & \text{if } \lambda_n \le \lambda \end{cases}$$

where $\bar{v}_j$ is the jth component of $Q^T v$ that is (q^j, v). Similarly, we have

$$\eta_{k-1} = \int_{\lambda_1}^{\lambda_n} \lambda p_k(\lambda) p_{k-1}(\lambda)\, d\sigma(\lambda).$$

The polynomials p_k are orthogonal with respect to the scalar product defined by the Riemann–Stieltjes integral for the measure σ; see [11]. This measure is piecewise constant with jumps at the eigenvalues of A, see the blue curve in figure 2.1. Of course, the measure σ is unknown since we do not know the eigenvalues and eigenvectors of A. Hence we cannot compute the coefficients α_j and η_j directly from the integrals. Therefore the idea is to construct an approximation of the measure σ. It was remarked by several people (see for instance Fischer and Freund [6]) that the Jacobi matrix T_k computed by the Lanczos algorithm provides an approximation to σ after a few iterations. This is shown in figure 2.1. This exemple uses a matrix arising from the finite difference discretization of the Poisson equation in the unit square using a 10×10 regular mesh. This gives a linear system of order $n = 100$. Note that the matrix A has only 51 distinct eigenvalues. The blue curve is the measure associated with A and a starting vector with random entries between 1 and -1 and the red curve is the measure obtained from the eigenvalues and eigenvectors of T_{10} and the vector e^1; see [11].

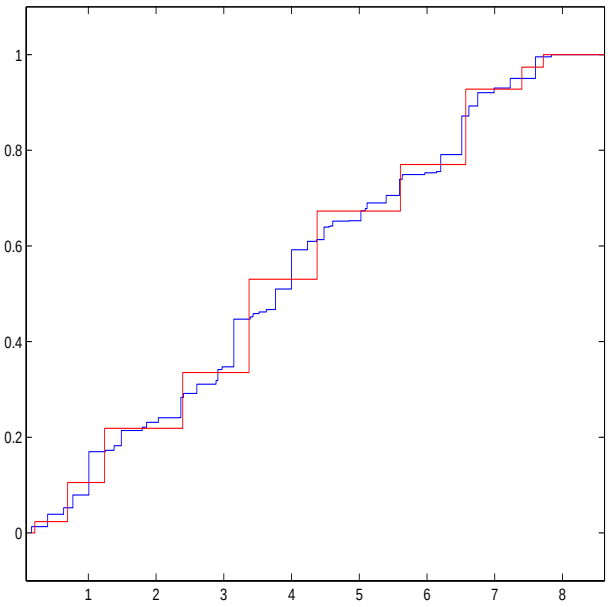

FIG. 2.1. *Measures for A (blue) and T_{10}(red)*

3. Computation of an approximate measure. We would like to obtain an approximation of the measure σ by using only the information obtained from the matrix T_k with k small. We use the same remark as in Fischer and Freund [6] which is that the middles of the vertical segments of the measure for T_k give good pointwise approximations to the measure for A. The idea is then to construct a monotone piecewise cubic interpolation of σ by using these middle points as interpolation points. This is done using the Matlab function `pchip`; see [8]. We obtain the green curve in

figure 2.1 using 200 points to display the continuous approximation. This gives what is a good approximation for the measure σ.

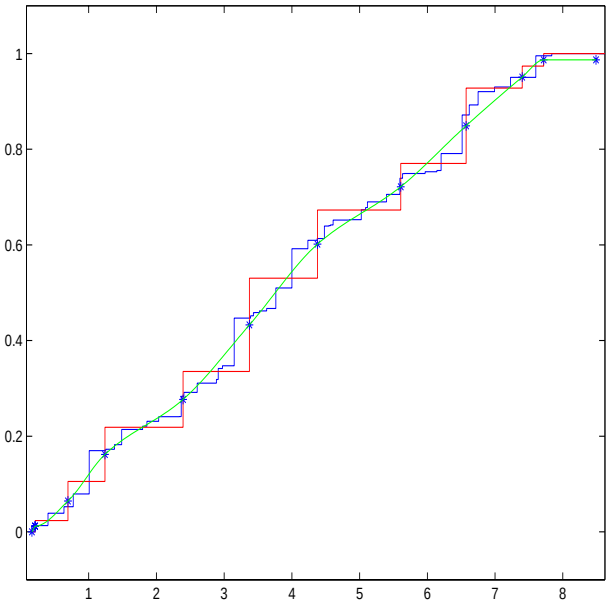

FIG. 3.1. *Measures for A (blue), T_{10} (red), the interpolation points (blue stars) and the interpolant (green)*

Then we use this interpolant to compute approximate values of α_k and η_k. We first discretize the continuous monotone interpolant. The interpolation points ξ_j are chosen as the Fejer points (see Gautschi [9]) plus λ_{min} and λ_{max} which are lower and upper bounds for the smallest and largest eigenvalues of A. The values f_j are obtained from the continuous interpolant at the mid-point of each interval.

The problem of computing the coefficients from the discretization $\tilde{\sigma}$ of the continuous approximation of the measure σ can be considered as an inverse eigenvalue problem. We have the (interpolation) points ξ_j and the weights which are the differences of the values $f_{j+1} - f_j$. We then use the Gragg and Harrod algorithm (see [12]) as implemented in the package OPQ of Gautschi [10] which computes the Jacobi matrix (the coefficients $\tilde{\alpha}_k$ and $\tilde{\eta}_k$) from the spectral data. These are the coefficients of the three-term recurrence of the orthogonal polynomials associated to $\tilde{\sigma}$. The Gragg and Harrod algorithm is known as being more stable and reliable than the Stieltjes or Lanczos algorithms; see [11]. We could have also use the pftoqd algorithm from Laurie [13].

4. Solving the linear system. Since the coefficients are sensitive to variations of the measure (see [17]) we do not use them directly to solve the linear system. We construct a Krylov basis using the three-term recurrence generated by the coefficients $\tilde{\alpha}_k$ and $\tilde{\eta}_k$. This gives us a polynomial $\tilde{p}_k(A)$ and we can obtain new basis vectors $\tilde{v}^k$ by $\tilde{v}^1 = v^1$ and

$$\tilde{v}^k = \tilde{p}_k(A)v^1, \ k = 2, \ldots$$

The vectors $\tilde{v}^k$ are defined as

$$\tilde{\eta}_k \tilde{v}^{k+1} = Av^k - \tilde{\alpha}_k v^k - \tilde{\eta}_{k-1} v^{k-1}, \quad \tilde{v}^1 = v^1.$$

This defines a matrix $\tilde{V}_k$ whose columns are the basis vectors $\tilde{v}^j$. The coefficients $\tilde{\alpha}_j$ and $\tilde{\eta}_j$ are all known in advance; hence, to introduce more parallelism we can remark that for computing $\tilde{v}^{k+1}$ we have $A\tilde{v}^k = (A^2\tilde{v}^{k-1} - \tilde{\alpha}_{k-1}A\tilde{v}^{k-1} - \tilde{\eta}_{k-2}A\tilde{v}^{k-2})/\tilde{\eta}_{k-1}$. This allows to be able to compute simultaneously $A\tilde{v}^{k-1}$ and $A^2\tilde{v}^{k-1}$. On modern processors this can help improving the computational speed; see [3]. We can compute an approximate solution $x^k = x^0 + \tilde{V}_k z$ by solving

$$\tilde{V}_k^T A \tilde{V}_k z = \tilde{V}_k^T r^0.$$

Note that the matrix-vector products $A\tilde{v}^j$ have already been computed by the three-term recurrence. When going from step $k-1$ to step k we have to compute the inner products $(\tilde{v}^k, \tilde{v}^j)$, $j = 1, \ldots, k$ and $(\tilde{v}^k, r^0)$. All these inner products can be computed in parallel. Is the basis given by $\tilde{V}_k$ better than the natural Krylov basis given by the vectors $A^j v^1$? The numerical rank of the natural basis is 26 for our small example and 51 for the basis obtained from $\tilde{\alpha}_k$ and $\tilde{\eta}_k$.

Let us summarize the algorithm:

- Do `init` iterations of the Lanczos algorithm to obtain $T_{\texttt{init}}$
- Compute the eigenvalues of $T_{\texttt{init}}$
- Compute the interpolant using `pchip`
- Discretize the interpolant
- Compute the coefficients $\tilde{\alpha}_k$ and $\tilde{\eta}_k$ using the Gragg and Harrod algorithm
- Compute the basis vectors $\tilde{v}^k$
- Compute the Cholesky factorization of $\tilde{V}_k^T A \tilde{V}_k$
- Compute the approximation x^k

It may seem that this way of computing an approximate solution has several drawbacks. The first one is that we have to compute the full matrix $V_k^T A \tilde{V}_k$ since it is not tridiagonal because the vectors $\tilde{v}^k$ are orthogonal but not for the usual inner product. This means that, contrary to our first goal, we have to compute more inner products. As we said, they can be computed in parallel and there is no synchronization points between the inner products. Anyway we have to solve a dense system even though the Cholesky factorization can be done incrementally in parallel with the computation of some of the inner products.

The second problem is that the vectors $\tilde{v}^k$ may loose their linear independence which means that the matrix $\tilde{V}_k^T A \tilde{V}_k$ may become singular or quasi-singular. Figure 4.1 is the same example as before with `init`=10. Around iteration 40 the matrix becomes quasi-singular and the norm of the residual (red curve) stagnates. Then, we restart the algorithm with the current solution at iteration 30 (green), 40 (magenta) and 50 (cyan). We see that we regain the same speed of convergence as before. For CG we show the norm of the iterated residual; for the other computations we display the norm of $b - Ax^k$. In these computations we did not use the last Lanczos iterate as the starting vector, we just use the given x^0 (which is 0 in our case).

Choosing the moment of restart can be automatized by testing if the matrix is quasi-singular when computing the Cholesky factorization. This is done by testing the pivots that are computed in the Cholesky factorization. Let $C_k = \tilde{V}_k^T A \tilde{V}_k = L_k L_k^T$, the matrix L_k being lower triangular. The factorization of C_{k+1} is computed by first solving $L_k y^k = C_{k+1}(1 : k, k+1)$ where this last vector is made of the k first

components of the last column of C_{k+1}. Then

$$(L_{k+1})_{k+1,k+1} = \sqrt{(C_{k+1})_{k+1,k+1} - (y^k)^T y^k}$$

and

$$L_{k+1} = \begin{pmatrix} L_k & 0 \\ (y^k)^T & (L_{k+1})_{k+1,k+1} \end{pmatrix}.$$

We restart the algorithm if $(L_{k+1})_{k+1,k+1} < \varepsilon_L$ where ε_L is a given threshold.

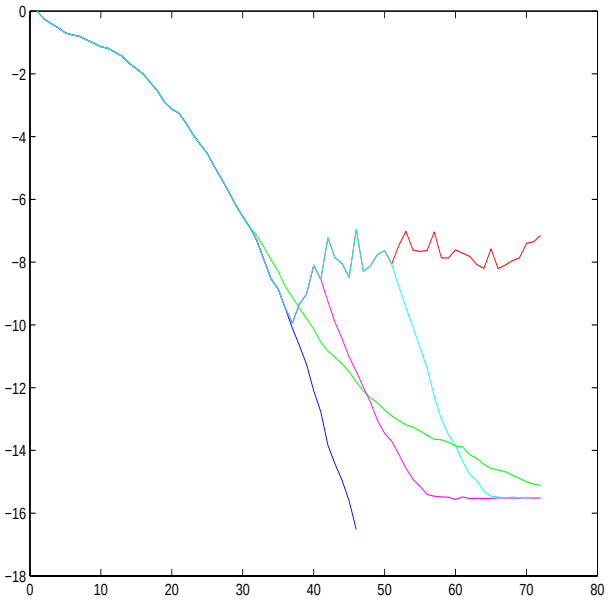

FIG. 4.1. $\log_{10}$ *of the norm of the residual, CG (blue),* `init`=10, *no restart (red),* `restart`=*30 (green), 40 (magenta), 50 (cyan),* $n = 100$

5. Preconditioning. To obtain a practical algorithm we have to introduce preconditioning. Let us assume that we have a preconditioner M which is a symmetic positive definite matrix. First we have to use the preconditioned Lanczos algorithm. We solve the linear system

$$M^{-\frac{1}{2}} A M^{-\frac{1}{2}} y = M^{-\frac{1}{2}} b.$$

The solution is recovered by $x = M^{-\frac{1}{2}} y$. The orthogonal basis is computed using

$$M^{-\frac{1}{2}} A M^{-\frac{1}{2}} \bar{V}_k = \bar{V}_k \bar{T}_k + \bar{\eta}_k \bar{v}^{k+1} (e^k)^T.$$

Of course we cannot compute $M^{-\frac{1}{2}}$. We can get rid if it by a change of variable. Let $V_k = M^{-\frac{1}{2}} \bar{V}_k$. Then

$$M^{-1} A V_k = V_k \bar{T}_k + \bar{\eta}_k v^{k+1} (e^k)^T.$$

The first vectors are defined as

$$\bar{v}^1 = \frac{\bar{r}^0}{\|\bar{r}^0\|}, \quad v^1 = M^{-\frac{1}{2}} \bar{v}^1 = \frac{M^{-\frac{1}{2}} \bar{r}^0}{\|\bar{r}^0\|}.$$

But

$$M^{-\frac{1}{2}}\bar{r}^0 = M^{-1}r^0 = M^{-1}(b - Ax^0)$$

and $\|\bar{r}^0\| = (M^{-1}r^0, r^0)$. Therefore

$$v^1 = \frac{M^{-1}r^0}{(M^{-1}r^0, r^0)}.$$

It remains to see how to compute the coefficients $\bar{\alpha}_k$ and $\bar{\eta}_k$. We have

$$\bar{\alpha}_k = (M^{-\frac{1}{2}}AM^{-\frac{1}{2}}\bar{v}^k, \bar{v}^k) = (Av^k, v^k).$$

The other coefficient is given by

$$\bar{\eta}_k = \|M^{-\frac{1}{2}}AM^{-\frac{1}{2}}\bar{v}^k - \bar{\alpha}_k\bar{v}^k - \bar{\eta}_{k-1}\bar{v}^{k-1}\|.$$

It is not difficult to see that

$$\bar{\eta}_k^2 = (M^{-1}Av^k - \bar{\alpha}_k v^k - \bar{\eta}_{k-1}v^{k-1}, Av^k).$$

The Lanczos iterates are computed as $y^k = y^0 + \bar{V}_k\bar{z}$ with

$$\bar{T}_k\bar{z} = \|\bar{r}^0\|e^1.$$

It gives $x^k = x^0 + V_k\bar{z}$.

For the parallel Lanczos algorithm we first do `init` iterations of the preconditioned Lanczos algorithm and then we compute the approximate measure from the eigenvalues of $\bar{T}_k$. The Gragg and Harrod algorithm gives the coefficients $\tilde{\alpha}_j$ and $\tilde{\eta}_j$ from which we obtain the matrix $\tilde{V}_k$. Even with preconditioning the linear system to solve is

$$\tilde{V}_k^T A\tilde{V}_k\tilde{z} = \tilde{V}_k^T(b - Ax^0)$$

and the approximate solution is obtained by $x^k = x^0 + \tilde{V}_k\tilde{z}$.

6. Numerical experiments. We first solve a linear system corresponding to the finite difference discretization of the Poisson equation on a regular 30×30 mesh of the unit square. We use a random right hand side with a unit norm. Figure 6.1 displays the result without preconditioning. With $\varepsilon_L = 10^{-5}$ (red curve) there are restarts at iterations 86 and 177. With $\varepsilon_L = 10^{-7}$ the restarts happen at iterations 103 and 188. Figure 6.2 shows the results with an incomplete Cholesky IC(0) preconditioner. Using a good preconditioner allows to reduce the number of initial Lanczos iterations. The results of figure 6.3 use only `init`=3. With $\varepsilon_L = 10^{-5}$ there are two restarts at iterations 28 and 55. With $\varepsilon_L = 10^{-7}$ the algorithm restarts at iterations 35 and 70.

Then we consider a problem that is more difficult to solve. We look for an approximate solution to the following PDE

$$-\mathrm{div}(\lambda(x, y)\nabla u) = f,$$

in $\Omega =]0, 1[^2$ with homogeneous Dirichlet boundary conditions.

The diffusion coefficient is defined as

$$\lambda(x, y) = \frac{1}{(2 + p\sin\frac{x}{\eta})(2 + p\sin\frac{y}{\eta})}.$$

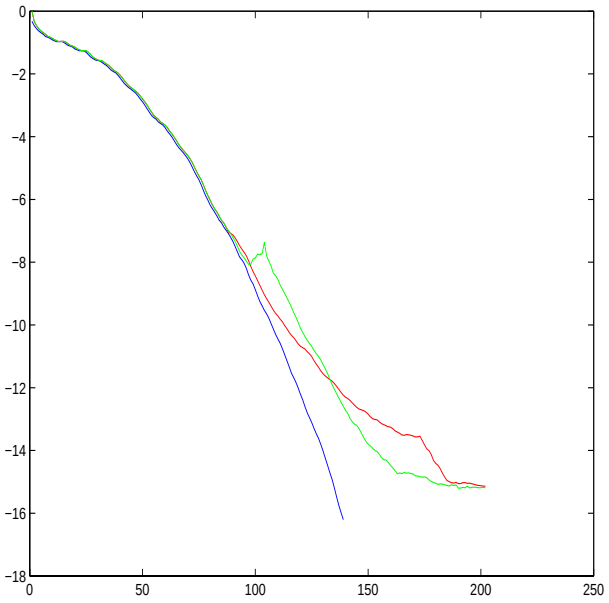

FIG. 6.1. $\log_{10}$ *of the norm of the residual, CG (blue),* `init`=10, $\varepsilon_L = 10^{-5}$ *(red),* $\varepsilon_L = 10^{-7}$ *(green), Poisson problem* $n = 900$

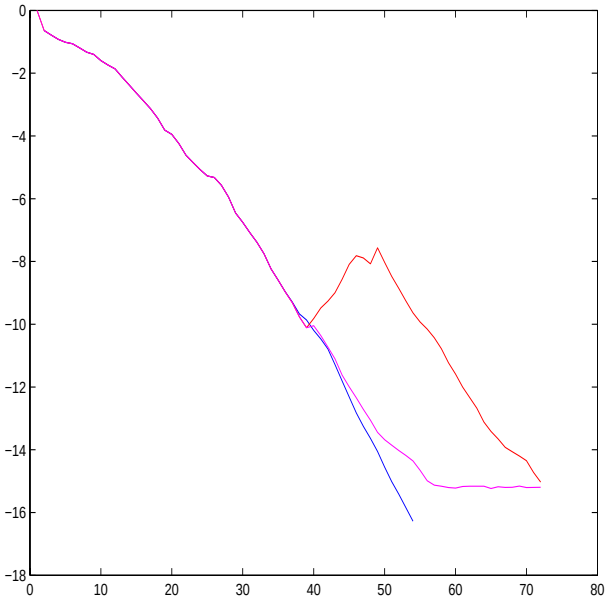

FIG. 6.2. $\log_{10}$ *of the norm of the residual, IC(0) preconditioning, CG (blue),* `init`=10, $\varepsilon_L = 10^{-2}$ *(magenta),* $\varepsilon_L = 10^{-5}$ *(red), Poisson problem* $n = 900$

The solution is given as $u = \sin(\alpha\pi x)\sin(\beta\pi y)$ with α and β being positive integers. Given this solution, the right hand side is $f = N/D$ with.

$$N = (2 + p\sin\frac{x}{\eta})(2 + p\sin\frac{y}{\eta})\sin(\alpha\pi x)\sin(\beta\pi y)[\alpha^2 + \beta^2]\pi^2$$

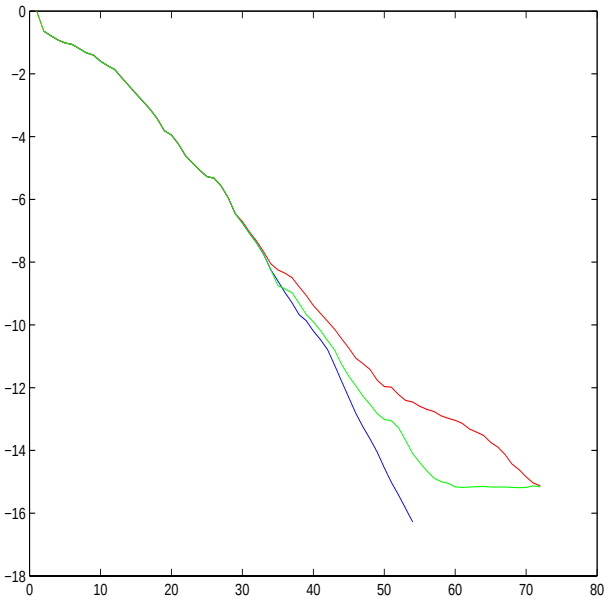

FIG. 6.3. $\log_{10}$ *of the norm of the residual, IC(0) preconditioning, CG (blue),* `init`=3, $\varepsilon_L = 10^{-5}$ *(red),* $\varepsilon_L = 10^{-7}$ *(green), Poisson problem* $n = 900$

$$+\frac{\pi p}{\eta}[\alpha\cos(\alpha\pi x)\sin(\beta\pi y)\cos\frac{x}{\eta}(2+p\sin\frac{y}{\eta})+\beta\sin(\alpha\pi x)\cos(\beta\pi y)\cos\frac{y}{\eta}(2+p\sin\frac{x}{\eta})],$$

and

$$D=(2+p\sin\frac{x}{\eta})^2(2+p\sin\frac{y}{\eta})^2.$$

The λ function may have peaks. The parameter η allows to choose the number of peaks and the value of the parameter p determines the heights of the peaks. We are interested in the values $p = 1.8$, $\eta = 0.1$ for which the diffusion coefficient has a single peak and $\alpha = 1, \beta = 1$. We use simple five-point finite differences on a regular cartesian mesh. The mesh has $m \times m$ interior points for which we want to compute the solution which is 0 on the square boundary. Figures 6.4 and 6.5 show the $\log_{10}$ of the norm of the residual for two different values of the number of initial Lanczos iterations.

7. Conclusions. In this paper we have proposed a Lanczos–like method for solving symmetric positive definite linear systems. We use the connections of the Lanczos algorithm with orthogonal polynomials to construct a (non-orthogonal) basis of the Krylov subspace. Although this algorithm needs more inner products than the CG algorithm it offers more potential for parallelism.

REFERENCES

[1] G. BALLARD, J. DEMMEL, O. HOLTZ, AND O. SCHWARTZ, *Minimizing communication in linear algebra*, Technical Report, no. UCB/EECS-2, Berkeley, University of California Berkeley, 05/2009.

[2] A.T. CHRONOPOULOS AND C.W. GEAR, *On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy*, Parallel Comp., v 11, (1989), pp 37–53.

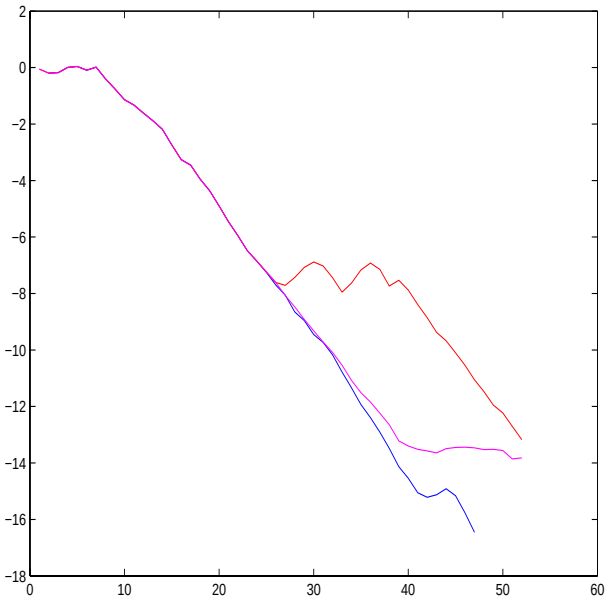

FIG. 6.4. $\log_{10}$ *of the norm of the residual, IC(0) preconditioning, CG (blue),* `init`=10, $\varepsilon_L = 10^{-2}$ *(magenta),* $\varepsilon_L = 10^{-5}$ *(red), second problem* $n = 900$

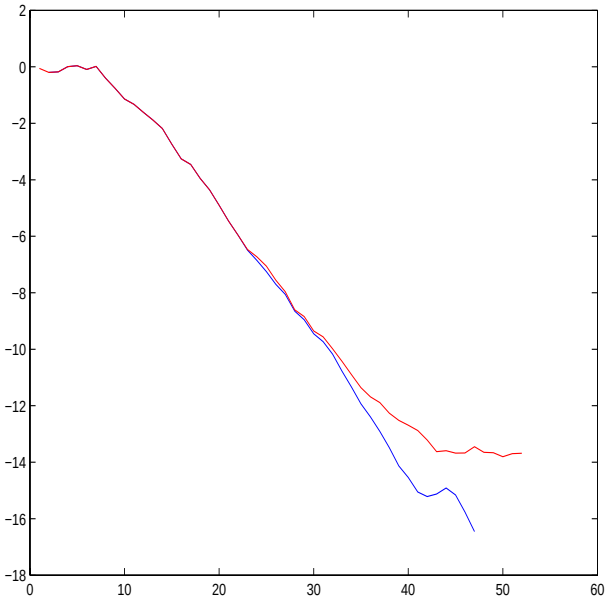

FIG. 6.5. $\log_{10}$ *of the norm of the residual, IC(0) preconditioning, CG (blue),* `init`=3, $\varepsilon_L = 10^{-5}$ *(red), second problem* $n = 900$

[3] K. DATTA, S. KAMIL, S. WILLIAMS, L. OLIKER, J. SHALF AND K. YELICK, *Optimization and performance modeling of stencil computations on modern microprocessors*, SIAM Review, December 2008.

[4] E.F. D'AZEVEDO, V. EIJKHOUT AND C. ROMINE, *Reducing communication costs in the con-*

jugate gradient algorithm on distributed memory multiprocessors, Report Rep. Working Note 56, LAPACK, (1992).

[5] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Avoiding communication in sparse matrix computations*, IEEE International Parallel and Distributed Processing Symposium, April 2008.

[6] B. Fischer and R.W. Freund, *On adaptive weighted polynomial preconditioning for hermitian positive definite matrices*, SIAM J. Sci. Comput., v 15 n 2, (1994), pp 408–426.

[7] B. Fischer and R.W. Freund, *An inner product-free conjugate gradient-like algorithm for hermitian positive definite systems*, Proceedings of the Lanczos centenary conference, (1994).

[8] F.N. Fritsch and R.E. Carlson, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal.,v 17, (1980), pp 238–246.

[9] W. Gautschi, *Orthogonal polynomials: computation and approximation*, Oxford University Press, (2004).

[10] W. Gautschi, *Orthogonal polynomials (in Matlab)*, J. of Comp. and Appl. Math., v 178 n 1–2, (2005), pp 215–234.

[11] G.H. Golub and G. Meurant, *Matrix, moments and quadrature with applications*, Princeton University Press, (2009).

[12] W.B. Gragg and W.J. Harrod, *The numerically stable reconstruction of Jacobi matrices from spectral data*, Numer. Math., v 44, (1984), pp 317–335.

[13] D.P. Laurie, *Accurate recovery of recursion coefficients from Gaussian quadrature formulas*, J. Comp. Appl. Math., v 112, (1999), pp 165–180.

[14] G. Meurant, *The conjugate gradient method on supercomputers*, Supercomputer, v 13, (1986), pp 9–17.

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

[16] G. Meurant, *The Lanczos and Conjugate Gradient algorithms, from theory to finite precision computations*, SIAM, (2006).

[17] D.P. O'Leary, Z. Strakoš and P. Tichý, *On sensitivity of Gauss–Christoffel quadrature*, Numer. Math., v 107 n 1, (2007), pp 147–174.

[18] J. Van Rosendale, *Minimizing inner product data dependencies in conjugate gradient iteration*, Report 172178, ICASE, NASA Langley, (1983).