A Bluffers' Guide to Iterative Methods for Systems of Linear Equations

Victor Eijkhout

March 2003

1 Introduction

Iterative methods are a popular way of solving linear systems of equation

$$Ax = b$$

where A is square and non-singular. There are many iterative methods, and then there is the mysterious topic of 'preconditioning'. This article goes unabashedly for the breadth rather than the depth in describing this universe. There are no proofs of any sort.

1.1 Where do linear systems come from

The statement that linear systems can be solved with iterative methods is a bit deceptive: not every linear system can successfully be solve by an iterative method. See the next section for a few remarks about direct methods, which have a more catholic taste in linear systems.

The linear systems that are most amenable to solution by iterative methods are those that come from discretisation of partial differential equations, in particular those from mathematical physics. Here are a few of the possible sources of linear systems:

- Elliptic equations. These always give a linear system to solve. One characteristic[1] of elliptic systems is that all points of the solution are 'globally coupled': any point depends on the solution in any other point. This has the immediate consequence that the more parallel a solution method is, the slower it will converge because of the decreased global coupling.
- Parabolic equations. These give an elliptic linear system if an implicit method, such as backward Euler, is used.
- Nonlinear problems. The Jacobian system in a Newton-Raphson method can be solved iteratively. Iterative methods are particularly attractive here because full precision is often not needed, and one can stop iterating after the desired precision has been reached.
- Eigenvalue calculations. In order to find interior eigenvalues of a matrix' spectrum, one often applies a Lanczos method to a shifted and inverted form of A: in effect one computes eigenvalues of $(A - \sigma)^{-1}$. Since a Lanczos method requires multiplication with the coefficient matrix, we need solve a system with $A - \sigma$ in each iteration. This can be done iteratively, although these systems are by their very nature indefinite, which makes applying iterative methods harder.

1. So to speak

Of the above, elliptic equations are by far the most natural environment for iterative methods to thrive in. Hence, some further remarks. The discretisation of an elliptic PDE on a regular domain has a mesh size parameter h associated with it. One can then show that the 'spectral condition number' $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ of a matrix is of $O(h^{-2})$ for second order problems, and $O(h^{-4})$ for fourth order problems, independent of the number of space dimensions.

The PDE origin of linear systems has implications for their sparsity structure. Discretisation of PDEs by finite differences, finite elements, or finite volumes establishes connection of each discretisation point with just a few neighbours, the number of neighbours being independent of the total discretisation. This means that the number of nonzeros per row is rather small, and not related to the matrix size.

1.2 Iterative versus direct methods

The alternative to iterative methods is Gaussian elimination, which is equivalent to finding a LU factorization of the matrix and using that to solve the system. The use of a factorization is called a direct method since it yields its results in a fixed number of operations. Direct methods have a wider applicability than iterative methods; on the other hand, they can take large amounts of time and space, as elements that are zero in the sparse coefficient matrix will fill in during the factorisation.

Fill-in can be somewhat aleviated by finding an appropriate ordering of the equations. For instance, in two space dimensions, the recursive bisection (or nested dissection) method [25] reduces fill-in to about $N \log N$. However, it can be proved that this does not hold in higher dimensions [35, 36].

Iterative methods, unlike direct methods, depend on numerical properties of the linear system for their time to solution. This makes them both potentially faster than direct methods, and potentially unsuitable for certain linear systems. It also means that applying them is considerably less straightforward than using direct methods.

2 Iterative methods

2.1 Stationary iterative methods

The easiest iterative methods to explain are the so-called 'stationary iterative methods'. Here's the idea.

Suppose that solving a system $Ax = b$ is not computationally feasible, but that solving $Mx = b$ *is* possible, and that presumably in some sense $M \approx A$. Let x_1 (the first iterate) be the solution of $Mx_1 = b$. With this we get the error vector $e_1 = x_1 - x$ and residual vector $r_1 = Ax_1 - b$. The two are related by $r_1 = Ae_1$.

Since $x = A^{-1}b = x_1 - A^{-1}r_1$, we can define $x_2 = x_1 - M^{-1}r_1$, and $x_3 = x_2 - M^{-1}r_2$, et cetera. This procedure is also called 'iterative refinement'. The basic scheme is then

$$x_{i+1} = x_i - M^{-1}r_i, \tag{1}$$

and this is called stationary because every iteration follows the same recipe, without iteration dependencies.

The analysis of this procedure is fairly simple. The relation between the first two residuals is

$$r_2 = Ax_2 - b = A(x_1 - M^{-1}r_1) - b = (I - AM^{-1})r_1$$

which inductively gives

$$r_n = (I - AM^{-1})^{n-1} r_1 \tag{2}$$

so the method will converge if all

$$|\lambda(I - AM^{-1})| < 1 \tag{3}$$

For future reference we note that equation (2) can be written more abstractly as

$$r_n = P^{(n-1)}(AM^{-1}) r_i \tag{4}$$

where $P^{(n-1)}(x)$ is a polynomial of degree $n-1$ and $P^{(n)}(0) = 1$ for all n. Because of this form, these iterative methods are called ‘polynomial iterative methods’.

Another, equivalent, way of looking at polynomial iterative methods is to consider the ‘Krylov sequence’

$$k_{i+1} = AM^{-1}k_i \qquad \text{where } k_1 = r_1.$$

The residuals are then combinations of this sequence, and the method generating the residuals is called a ‘Krylov method’[2]. The span of the Krylov vectors, and the initial parts of it, are called a Krylov space, or Krylov subspace, respectively.

2.1.1 Examples

Here are some of the more popular choices of finding an $M \approx A$:

- Jacobi method:

  $$M = D_A = \mathrm{diag}(A)$$

- Gauss-Seidel method:

  $$M = D_A + L_A$$

- SOR method:

  $$M = D_A + \omega L_A$$

  In the past, much attention was given to finding an optimal value for ω [29, 56, 57]. Because of the limited applicability of this method (and the fact that it is not very suited as a preconditioner in the kind of methods we will discuss below), this interest has largely evaporated. However, a variant of SOR is still popular:
- The SSOR method

  $$M = (D_A + \omega L_A) D_A^{-1} (D_A + \omega U_A)$$

  can be viewed as a symmetric version of SOR: a forward triangular solve followed by a backward solve. The popularity of this method derives from its use as a preconditioner in the iterative methods discussed later.

2. We note that stationary iterative methods are Krylov methods; the popular usage of the term is restricted to methods based on some form of orthogonality as will be discussed later.

2.1.2 Implementation

For methods such as the Jacobi and Gauss-Seidel methods, which use the actual matrix elements, implementation as in equation (1) is wasteful. A slight rewrite gives for instance for the Gauss-Seidel method

$$(D_A + L_A)x_{i+1} = b - U_A x_i.$$

(Some people motivate this formulation by saying 'Split $A = M + N$, then $Ax = b$ becomes $Mx = b - Nx$, so you iterate $Mx_{i+1} = b - Nx_i$.')

2.1.3 Regular splittings and M-matrices

The convergence condition equation (3) is satisfied for these methods if A is a so-called M matrix:

- A positive definite
- $a_{ij} \leq 0$ for $i \neq j$.

(There are many other equivalent definitions of M-matrices [6].) Defining a matrix N from $A = M - N$ gives a 'splitting' of A; the above choices for M give so-called 'regular splittings'. The matrix M is also called the 'preconditioner' and N is called the 'rest matrix'.

2.1.4 Speed of convergence

For matrices that have a mesh parameter h associated with it, we can sometimes give an order estimate for the convergence of a splitting.

- For the Jacobi method,

$$\rho(I - AM^{-1}) = \max_{\lambda} |\lambda(I - AM^{-1})| = 1 - O(h^2).$$

- The Gauss-Seidel method can be shown to converge faster than the Jacobi method, but only by a constant: we still have

$$\rho(I - AM^{-1}) = 1 - O(h^2).$$

- Finding the optimal ω in the SOR method can lead to a more substantial acceleration:

$$\rho(I - AM^{-1}) = 1 - O(h).$$

2.2 Not quite stationary methods

If we slightly generalise the stationary iteration scheme by introducing a parameter:

$$x_{i+1} = x_i + \alpha_i M^{-1} r_i$$

we get a 'line search': the next iterate is found on a line through the previous iterate composed of multiples of a 'search direction', which is in this case the residual, but we will generalise this later.

In the case where A is symmetric and positive definite, the residual is also the gradient direction, or the direction of 'steepest descent'. The value of α_i is then easily computed as $r_i^t r_i / r_i^t A r_i$.

Another nonstationary method is the ‘Chebyshev method’. This introduces a new idea: the iterates and residuals in this method satisfy a ‘three-term recurrence’:

$$\begin{aligned}\alpha_i x_{i+1} + \beta_i x_i - (\alpha_i + \beta_i) x_{i-1} &= M^{-1} r_i \\ \alpha_i r_{i+1} + \beta_i r_i - (\alpha_i + \beta_i) r_{i-1} &= AM^{-1} r_i\end{aligned}$$

The coefficients are chosen such that the polynomials in equation (4) are subsequent shifted and normalized Chebyshev polynomials, where the shift is chosen to maximise the residual reduction on an interval that contains the spectrum of $M^{-1}A$.

2.3 Let’s take a step back

In the Chebyshev method above we see the two main ways of differentiating iterative methods:

- Choice of the ‘preconditioner’ M, and
- choice of the iteration coefficients.

These two are really separate issues, and we will have a few more things about them to say here.

Above, we stated that M somehow had to approximate A. For the moment we are not quantifying that statement; here we will just remark that the above methods can also be considered as methods with $M = I$ for a transformed system $M^{-1}Ax = M^{-1}f$. This is called a ‘left-preconditioned system’ (but note that $r_n = P^{(n-1)}(AM^{-1})r_1$). There are also ‘right-preconditioned systems’, but they don’t compute the solution to the original system: they require a back transformation.

So far we have seen two-term and three-term equations for the iterates and residuals. The general form of polynomial iterative methods is

$$r_{n+1}h_{n+1,n} = AM^{-1}r_n - \sum_{i\leq n} r_i h_{in} \tag{5}$$

which we can also write compactly as

$$AM^{-1}R_{n+1} = R_n H_n$$

where H_n is an $(n+1)\times n$ ‘Hessenberg matrix’, satisfying $\sum_i h_{in} = 0$. This last condition can be shown to be equivalent to the normalisation condition $P(0) = 1$ (section 2.1).

Another way of looking at this general form of iterative methods is to go back to the stationary method of equation (1), and let this generate iterates $\tilde{x}_i$ and residuals $\tilde{r}_i$. The residuals of equation (5) (and corresponding iterates) can then be considered as ‘affine combinations’

$$r_i = \sum_{j\leq i} \gamma_{ij}\tilde{r}_j, \qquad \sum_j \gamma_{ij} = 1$$

of the stationary residuals (respectively iterates). Since the affine coefficients are presumably chosen to let the sequence r_i converge faster than $\tilde{r}_i$, the methods with general coefficients are often called ‘accelerated iterative methods’[3].

Accelerated methods satisfy an equation for the iterates

$$x_{n+1} - x_n \in M^{-1}[\![r_1, AM^{-1}r_1, \ldots, (AM^{-1})^n r_1]\!]$$

This equation is often given as the basic definition of polynomial iterative methods.

3. At the time of this writing, such terms as ‘conjugate gradient acceleration’ are quite unfashionable, however.

2.4 Orthogonality based methods

Above we have already seen two choices of the polynomials in Krylov methods:

- Stationary iterative methods were based on the polymial $P^{(n)}(x) = (1-x)^n$, which gives a geometric decrease of the residual norm[4].
- Chebyshev methods recursively construct polynomials that maximally reduce the residual norm, given an estimate for the matrix spectrum.

From now on we will look at a choice of polynomials that has proved very succesful in practice: the polynomials are chosen such that the residuals satisfy some orthogonality condition to a certain subspace. In an extremely informal sense this sounds like a plausible strategy: orthogonalising implies projecting onto a subspace, which implies minimising over a subspace.

We will be considerably more precise than this in our upcoming presentation of methods such as the Conjugate Gradients method. For now we note that orthogonalisation involves inner products, and all the computational aspects that it brings with it; we also note that the above hand-waving argument presumes that the matrix is definite, and it implies that the further a problem is from being elliptic, the less appropriate iterative methods are.

We also note the – in practice rather useless fact – that residuals being orthogonal implies that the method will converge to an exact solution in N iterations. Round-off error, anyone?

2.5 The method of Conjugate Gradients; theory

Historically the first orthogonality-based Krylov method is the ‘Method of Conjugate Gradients’ [30], which orthogonalises the residuals. Several comments:

- The conjugate gradients method is derived for symmetric positive matrices.
- From symmetry of the matrix it follows that equation (5) reduces to just a three-term recurrence.
- Adding in definiteness of the matrix to its symmetry, one can show that the residual in the n-th iteration is minimal over the Krylov subspace generated so far. However, minimality is in the A^{-1} norm (the ‘energy norm’), not in the L_2 norm.
- While predicting the exact number of iterations of CG is impossible, at least we can give an order bound: the expected number of iterations is proportional to $\sqrt{\kappa(AM^{-1})}$.
- Even though the CG method is equivalent to a three-term version of equation (5), it uses a computational form of coupled two-term recurrences:

  $$\begin{aligned} x_{i+1} &= x_i + \alpha_i p_i \\ r_{i+1} &= r_i + \alpha_i A p_i \\ p_{i+1} &= M^{-1} r_{i+1} + \beta_i p_i \end{aligned}$$

  where the p_i vectors are the search directions.

The above remarks more or less outline the issues concerning iterative methods we will be covering in the sequel.

2.6 Minimisation and orthogonality

Minimisation in the energy norm, which was proved for the Conjugate Gradients method, is a special case that only holds for symmetric positive definite systems. Minimisation in

4. Kinda sorta.

the L_2 norm, which is closer to what we are interested in, can be proved in a more general setting: r_{n+1} is minimal if it is orthogonal to $AM^{-1}R_n$.

This minimisation, incidentally, makes the search directions orthogonal, which is reflected in the name of the OrthoDir method.

Such orthogonality and minimisation, can be achieved in several ways.

- One can explicitly use a form of Gramm-Schmid to impose the proper orthogonality of residuals or search directions. This is done in GCR [20] and GMRESr [50].
- One could generate orthogonal residuals (or a normalized form of them) with the attendant Hessenberg matrix, and form the minimum residuals by taking combinations determined by a QR factorization of the Hessenberg matrix. This is the idea behind MinRes [41] and GMRES [45].
- Finally, one can observe that the minimum residuals are related through a two-term recurrence to the orthogonal residuals. Deriving the residuals this way is called 'residual smoothing' [55, 58]. Only the TFQMR method [23] is traditionally derived with residual smoothing, but other minimising methods such as GMRES and QMR could as well have been presented this way.

2.7 Three-term recurrences

We have noted that, in order to obtain orthogonality of residuals, symmetric systems give three-term recurrences, and nonsymmetric systems use long recurrences where all previous residuals (or search directions) are saved. The question whether orthogonality was possible in the nonsymmetric case with less than full recurrences was solved essentially in the negative [22]: short recurrences are only possible if the spectrum of the matrix lies on a straight line in the complex plane.

In practice, the unbounded storage needs of GMRES are a problem. The most common solution is to use restarting: after a number of iterations the whole history is thrown away, and the method is resumed using the current iterate as initial guess.

2.8 Biconjugacy and Quasi-minimisation

The conclusion of the above sections is that orthogonality-based iterative methods for nonsymmetric systems inevitably needs large amounts of storage. One way out of this problem is to relax the orthogonality condition.

Recall that Krylov methods generate (implicitly) a sequence of polynomials such that $r_{n+1} = P^{(n)}(AM^{-1})r_1$. The Biconjugate Gradient method generates a second sequence $\{s_i\}$ satisfying $s_{n+1} = P^{(n)}(A^tM^{-t})s_1$, and only enforces orthogonality of r_{n+1} against S_n, and vice versa. The s_1 vector can be chosen arbitrarily. Although this choice influences the convergence, no algorithm exists for finding an appropriate value, so in practice $s_1 = r_1$ is taken.

The computationally attractive part of this method is that both the $\{r_i\}$ and $\{s_i\}$ sequences are generated using coupled two-term recurrences, so the disadvantages of GMRES have been obviated. One disadvantage of BiCG (and QMR below) is that it needs products with A^t evaluated. If the matrix is given in operator form, this may be impossible.

Methods that generate true orthogonal residual are called 'Arnoldi methods' [1], which methods based on biconjugacy are called 'Lanczos methods' [34]. Both methods were originally derived for eigenvalue problems.

Applying one of the above methods for deriving minimised residuals to the residuals of the BiCG method gives the 'QMR' [24] method. (The method was originally presented using a QR factorisation, but an implementation based on residual smoothing is considerably simpler.) Since BiCG does not use true orthogonalisation, we also do not get true minimisation in QMR: the term 'quasi minimisation' is applied here, hence the name Quasi Minimum Residual method.

2.9 Squared methods

The Biconjugate Gradient method has given rise to two methods that work on a somewhat different principle from other Krylov methods. The first of these is the Conjugate Gradients Squared method [46]. It is based on two observations:

- BiCG computes quantities such as

  $$s_n^t r_n = (P^{(n)}(A^t)s_1)^t (P^{(n)}(A)r_1) = s_1^t {P^{(n)}}^2(A)r_1 = s_1\tilde{r}_n$$

  if we define $\tilde{r}_n = {P^{(n)}}^2(A)r_1$. Apparently these inner products can be evaluated using an expression that no longer involves the $\{s_1\}$ sequence.
- The vectors $\tilde{r}_n$ can actually be computed.

Eliminating the s_n sequence has the advantage that products with A^t are no longer needed.

This method is unlike Krylov methods discussed earlier, because the subsequent $\tilde{r}_n$ vectors are not in Krylov subspaces of dimensions increasing by 1. Also, they no longer satisfy orthogonality or minimization properties.

Intuitively the CGS method is attractive since, if the BiCG residuals are decreasing, the residuals of this method might be decreasing twice as fast. However, the flip side of this coin is that in cases of irregular convergence behaviour CGS will be even more irregular. This method is not much used in practice. TFQMR, which smooths out CGS, does not essentially improve the convergence.

The irregularity of CGS is greatly alleviated in BiCGstab [53], which is based on the observation that not only can ${P^{(n)}}^2(A)r_1$ be computed efficiently, we can also compute $Q^{(n)}(A)P^{(n)}(A)r_1$. The choice taken in practice is to let Q be a product of steepest descent polynomials (or equivalently GMRES(1)). Choices such as GMRES(2) have been suggested but not found wide acceptance.

This method is currently among the most popular iterative methods in use.

Both CGS and BiCGstab use only a small number of temporary vectors; their work per iteration is comparable to a BiCG iteration, except that the transpose matrix-vector product has been replaced by a regular one.

3 Preconditioning

Above, in section 2.3, we explained how a preconditioner functions as a transformation of a linear system, presumably such that it will improve the convergence of iterative methods.

Regarding this improvement we make a few points.

- First of all, the added cost of applying the preconditioner in each iteration – and this can easily be more than the rest of the operations combined – has to be less than the improvement gained from the reduction in the number of iterations.

- Secondly, there is usually a preprocessing cost associated with constructing the preconditioner. This cost can be a substantial portion of the total iteration cost, but if more than one system is solved with the same coefficient matrix, for instance in time-stepping methods or inexact Newton methods, this cost is amortized.
- In the case of linear systems that have a discretisation parameter associated with it (see section 1.1) the convergence improvement can take the form of a reduction by a constant, or by an order. For instance, for elliptic systems the condition number of the original coefficient matrix is $\kappa(A) = O(h^{-2})$, but some preconditioners can reduce this to $\kappa(M^{-1}A) = O(h^{-1})$, or even to $O(1)$. In this last case we call A and M 'spectrally equivalent'; the number of iterations will then be bounded by a constant in the mesh size parameter.

We will now discuss a number of preconditioners in detail.

3.1 Preconditioners using the matrix elements

The splittings discussed in section 2.1.1 can also be used as preconditioners. (Alternatively, we can view these preconditioned methods as accelerations of the stationary methods; see section 2.3.) Thus we have the Jacobi preconditioner and the SSOR.

Gauss-Seidel and SOR are almost never used as preconditioners. One reason for this is that they are nonsymmetric, even if the original matrix is symmetric. Preserving symmetry is usually a good idea. For SOR there is an additional reason: with the optimal ω, a CG method would reduce to stationary iteration.

One remark about the Jacobi preconditioner: its use is equivalent to applying an unpreconditioned method to a system that has been scaled to unit diagonal. Since other scalings are possible, this raises the question if the Jacobi scaling is optimal. The answer is yes: out of all possible scalings, the one using the matrix diagonal gives the highest reduction of the matrix condition number.

3.2 Incomplete factorisation preconditioners

While a full factorisation of the coefficient matrix A may be prohibitive in time and space demands, we can derive useful preconditioners from the general idea of factorisation. So-called 'incomplete LU factorisations' work by ignoring certain of the fill elements; zero elements in the original matrix that would turn nonzero during a factorisation. That is, where a normal factorisation would have a statement

$$a_{ij} \leftarrow a_{ij} - a_{jk}a_{kk}^{-1}a_{kj}$$

in the inner loop, an incomplete factorisation would have

$$(i,j) \in S \quad \Rightarrow \quad a_{ij} \leftarrow a_{ij} - a_{jk}a_{kk}^{-1}a_{kj}$$

where S is some set of matrix element locations.

In a way, the SSOR preconditioner

$$M = (D_A + L_A)D_A^{-1}(D_A + U_A)$$

is an incomplete factorisation[5]. The algorithm deriving it simply never alters any matrix elements at all.

5. There are several mathematically, though not computationally, equivalent ways of writing a factorisation. The form used here brings out the symmetry of the preconditioner; a computationally more suitable form would be $M = (D + L)(I + D^{-1}U)$.

Next up in complexity is ILU-D [42], the factorisation where only alteration of the diagonal is allowed. This leads to a form

$$M = (D + L_A)D^{-1}(D + U_A)$$

where $D \neq D_A$. This implies that the extra storage required for the factorisation is only one vector's worth of pivots. The 'Eisenstat algorithm' [19] is an efficient implementation of CG with this preconditioner, effectively eliminating the duplicated operations with original matrix elements in the preconditioner.

The ILU(0) preconditioner is obtained by letting S be the set of the original nonzero locations. The term ILU(0) is often mistakingly applied to the above ILU-D method; on the other hand, for matrices such as from linear finite elements[6] the two methods are identical.

All ILU preconditioners discussed so far reduce the condition number of the system, but only by a constant: asymptotically the condition stays $O(h^{-2})$ for elliptic systems.

3.3 Fill-in strategies: ILU(k), ILU(r)

So far we have seen only fairly simple ways of dealing with fill-in. There are two basic strategies for allowing fill-in in an incomplete factorisation:

- Fill levels. We define the original matrix nonzeros to be of level 0. A fill-in element caused by a_{ik} and a_{kj} being of level 0 is said to be of level 1. If either element is of level 1, the fill element is of level 2, et cetera. The ILU(k) factorisation allows fill-in of level k at most.
- Drop tolerances. We can also allow fill-in only if it is above a certain threshold. This method may be more accurate than one based on levels; however, this ILU(r) method is harder to program since the amount of storage depends dynamically on the factorisation, rather than statically on the matrix structure.

3.4 Existence problem for ILU methods

There is a potential problem with the use of ILU preconditioners. While the exact factorisation of a positive definite matrix is guaranteed to give positive pivots, this guarantee does not exist in general for incomplete factorisations of definite matrices. A zero pivot would mean the breakdown of the algorithm, while a negative pivot implies an indefinite preconditioner, which would destroy the theory of iterative methods such as CG.

Fortunately, for the important class of M-matrices one can show that, no matter the fill strategy, pivots will always be positive [2, 38].

For the more general case, several remedies have been suggested [17, 28, 32, 37]. Of these, probably the best is the idea to base the factorisation on $A + \alpha I$, where α is chosen large enough to get a well-defined factorisation, but not too large, since that negates the appropriateness of the factorisation.

3.5 Modified and relaxed ILU preconditioners

It is possible to reduce the condition number of the system to $O(h^{-1})$. For this, fill-in can not just be discarded: it has to be added to the diagonal. That is, the algorithm now has in

6. Technically: matrices for which the connectivity graph is a set of triangles.

its inner loop

$$\begin{array}{lll}(i,j) \in S & \Rightarrow & a_{ij} \leftarrow a_{ij} - a_{jk}a_{kk}^{-1}a_{kj} \\ (i,j) \notin S & \Rightarrow & a_{ii} \leftarrow a_{ii} - a_{jk}a_{kk}^{-1}a_{kj}\end{array}$$

Moving fill to the diagonal is one way of enforcing the condition that $Mv = Av$ where v is a positive vector, in this case the vector identically 1.

While in practice moving fill to the diagonal is often enough for a condition number reduction, for a rigorous proof small perturbations have to be added to the diagonal [4, 15, 27, 47]. This phenomenon has led to 'relaxed ILU' methods, where only a fraction of the fill is moved to the diagonal [3, 9, 52].

3.6 Computational efficiency of ILU methods

Solving a system with an ILU factorisation is inefficient in two ways:

- the basic scalar code is inefficient, and
- because of the large sequential component of the method, parallel execution is not trivial.

3.6.1 Sequential efficiency

The inefficiency of ILU on a single processor is hard to remedy. One thing one could do is replace the solution $Lx = y$ by another algorithm, which hopefully is close enough in accuracy. One such suggestion is to expand a normalized factor as

$$(I - L)^{-1} = I + L + L^2 + \cdots$$

and use only a few terms of this 'Neumann expansion' [51].

Use of recursive doubling and such would be another way of increasing the scalar efficiency.

3.6.2 Parallel efficiency

An efficient parallel execution can be arrived at in two ways:

- by finding parallelism in the original implementation, or
- by suitably permuting the system to a form that has different numerics, but displays more parallelism.

As an example of the first strategy, one could observe that the solution of a triangular system proceeds by 'wavefonts'; the independent variables on such a wavefront could them be partitioned over multiple processors [40], or executed in vector fashion on vector computers [49].

The second strategy is more likely to bear fruit. A 're-ordering' of the variables will affect the numerics, and most likely adversely, but the gains from large scale parallelism will sufficiently offset that. The ordering best suited for parallelism is the 'multi-colour ordering', where uncoupled variables are grouped together in colours. The variables in each colour are independent, and so can be processed in parallel [31].

3.7 Subdomain methods

There are a number of methods that proceed by cutting up the physical domain of definition in a number of (simply-connected) subdomains. (Colour sets, as described above, are not connected.) One motivation for this is that the restriction of an elliptic problem to a subdomain is again an elliptic problem. This means that, partitioning a problem like this over a parallel computer, each processor gets to solve an elliptic (sub)problem, so we get to reuse all our traditional knowledge of preconditioners.

There are three basic ways of connecting subdomains.

3.7.1 Block Jacobi preconditioning

The simplest domain partitioning method is arrived at by letting the subdomains be disjoint sets, and the local solutions are computed on them independently. The resulting method is called the ‘Block Jacobi preconditioner’, since it is a block form of the Jacobi preconditioner which solves all points independently of each other.

This method asymptotically improves the condition number only by a constant factor, which decreases with an increasing number of subdomains.

3.7.2 Domain decompositioning

The subdomains can be separated by a ‘separator or interface’, which is such that the subdomains connect to the interface variables, but not to another subdomain. The resulting method is called ‘Domain Decomposition’ or a ‘Schur Complement Method’.

The prime theoretical result [14, 7, 8] of domain decompositioning is that the Schur complement operator is equivalent to the square root of the Laplacian operator, with condition number $\kappa(S) = O(h^{-1})$. Thus, iterating with this operator gives a faster converging method then with the original system. This does, however, presume that the subdomains systems are solved exactly. In practice, one approximates both the Schur complement and the subdomain solves.

3.7.3 Schwarz methods

The third possibility for domain partitioning methods is to have the subdomains overlap. The ‘alternating’, or ‘multiplicative’, Schwarz method solves the subdomains alternately, with each stage using the solution of the other stage as boundary conditions.

A higher degree of parallelism, though a lower speed of convergence, is achieved in the ‘additive’ Schwarz method, where all subdomains solve a local problem in parallel, simply adding their solutions together. In the context of preconditioning, this will still give a converging method.

The condition number of a Schwarz-preconditioned operator is close to optimal; for true optimality, a ‘coarse grid operator’ may be needed to ensure sufficient global coupling.

3.8 Fast solvers as preconditioners

So-called ‘fast solvers’, such as the Fast Fourier Transform, are able to solve a linear system in an almost optimal number of operations of $O(N \log N)$ [48], provided the system comes from a ‘separable’ PDE

$$Au = -(a(x)u_x)_x - (b(y)u_y)_y.$$

If the system is not separable, but elliptic, a preconditioner based on a separable approximation of the coefficient matrix can still be used as a preconditioner, and in fact will give a number of iterations that is bounded by a constant in the problem size [16, 21].

3.9 Preconditioning by symmetric part

The 'symmetric part' of a matrix is defined as $(A + A^t)/2$. Basing a preconditioner on this is an attractive idea, since symmetric preconditioners suffer less from breakdown phenomena. Furthermore, in certain cases, preconditioning by the symmetric part gives spectral equivalence [54]. A particularly useful idea is to use a separable approximation to the symmetric part; this still gives spectral equivalence, and the separable preconditioner can be solved very efficiently.

3.10 Sparse approximate inverses

All preconditioners discussed so far were based on the idea of finding an operator M which approximated A, and with which linear systems could be solved relatively easily. However, one could also look at approximations of the inverse of A. One immediate practical advantage of this approach is that application of such an operator is likely to be easily parallelised: if $M \approx A^{-1}$ is explicitly given, its application is a matrix-vector product.

There are two basic approaches in use for finding approximate inverses.

- One could decide a sparsity pattern for the approximate inverse M and minimize $\|I - AM\|$ over such matrices [26, 33]. This algorithm is attractive in that the columns of M follow from independent systems, making parallel computation of M very simple.
- With a bordering algorithm that is in the spirit of LU factorisation one can compute M more gradually [5, 18, 43]. This algorithm has the theoretical advantage that it can preserve symmetry and definiteness of the original matrix.

3.11 Multigrid and multilevel methods

Ahem.

4 Miscellaneous topics

4.1 Numerical precision issues

4.2 Inner products

Iterative methods based on conjugacy necessarily involve inner products, which, in a parallel context, imply synchronisation points. It is easy enough to see that the two inner products of the CG method are inter-dependent, so there is no easy way to reduce this overhead. Several people have come up with mathematically equivalent formulations of CG (and other methods) that make the inner products independent, so that their communications can be combined. One can either eliminate the $r^t r$ inner product [44, 39], or the $p^t Ap$ one [10, 12, 11]. It is also possible to hide the communication by overlapping it with the preconditioner computation [13]; however, this presupposes a preconditioner like Block Jacobi.

4.3 Further reading

- The ‘Templates’ book.
  A short, cheap, practical introduction to iterative methods, preconditioners, data structures. `http://www.netlib/org/templates`
- Youcef Saad’s book.
  Covers almost all iterative methods and preconditioners. Practical sections on derivation of the linear systems, data storage, and parallelism. `http://cs.umn.edu/~saad/`

References

[1] W.E. Arnoldi. The principle of minimzed iterations in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.

[2] O. Axelsson. A general incomplete block-matrix factorization method. *Lin. Alg. Appl.*, 74:179–190, 1986.

[3] Owe Axelsson and Gunhild Lindskog. On the eigenvalue distribution of a class of preconditioning matrices. *Numer. Math.*, 48:479–498, 1986.

[4] R. Beauwens. On Axelsson’s perturbations. *Lin. Alg. Appl.*, 68:221–242, 1985.

[5] Michele Benzi and Miroslav TøUma. A sparse aproximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19:968–994, 1998.

[6] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences.* SIAM, 1994. originally published by Academic Press, 1979, New York.

[7] P. Bjørstad and O. Widlund. Iterative methods for the solution of elliptic problems on regions partitioned in to substructures. *SIAM J. Numer. Anal.*, 23:1097–1120, 1986.

[8] J.H. Bramble, J.E. Pasciak, and A.H. Schatz. An iterative method for elliptic problems on regions partitioned into substructures. *Math. Comp.*, 46:361–369, 1986.

[9] Tony F. Chan. Fourier analysis of relaxed incomplete factorization preconditioners. *SIAM J. Sci. Stat. Comput.*, 12:668–680, 1991.

[10] A. Chronopoulos and C.W. Gear. s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25:153–168, 1989.

[11] E.F. D’Azevedo, V.L. Eijkhout, and C.H. Romine. Lapack working note 56: Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessor. Technical Report CS-93-185, Computer Science Department, University of Tennessee, Knoxville, 1993.

[12] E.F. D’Azevedo and C.H. Romine. Reducing communcation costs in the conjugate gradient algorithm on distributed memory multiprocessors. Technical Report ORNL/TM-12192, Oak Ridge National Lab, 1992.

[13] J. Demmel, M. Heath, and H. Van der Vorst. Parallel numerical linear algebra. In *Acta Numerica 1993*. Cambridge University Press, Cambridge, 1993.

[14] M. Dryja. A capacitance method for elliptic problems on regions partitioned into substructures. *Numer. Math.*, 39:51–64, 1982.

[15] T. Dupont, R. Kendall, and H. Rachford. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM J. Numer. Anal.*, 5:559–573, 1968.

[16] E.G. D’Yakonov. The method of variable directions in solving systems of finite difference equations. *Soviet Mathematics / Doklady*, 2:577–580, 1961. TOM 138, 271–274.

[17] Victor Eijkhout. The ‘weighted modification’ incomplete factorisation method. Technical Report Lapack working note 145, UT-CS-99-436, Computer Science Department, University of Tennessee, 1999.

[18] Victor Eijkhout and Panayot Vassilevski. Positive definitess aspects of vectorizable preconditioners. *Parallel Computing*, 10:93–100, 1989.

[19] Stanley C. Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM J. Sci. Stat. Comput.*, 2:1–4, 1981.

[20] Howard C. Elman. *Iterative Methods for Large Sparse Nonsymmetric Systems of Equations*. PhD thesis, 1982.

[21] Howard C. Elman and Martin H. Schultz. Preconditioning by fast direct methods for non self-adjoint nonseparable elliptic equations. *SIAM J. Numer. Anal.*, 23:44–57, 1986.

[22] V. Faber and T. Manteuffel. Orthogonal error methods. *SIAM J. Numer. Anal.*, 24:170–187, 1987.

[23] Roland W. Freund. A tranpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14(2):470–482, 1993.

[24] Roland W. Freund and Noël M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

[25] Alan George and Joseph H-W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1981.

[26] Marcus J. Grote and Thomas Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18, 1997.

[27] I. Gustafsson. A class of first-order factorization methods. *BIT*, 18:142–156, 1978.

[28] Ivar Gustafsson. An incomplete factorization preconditioning method based on modification of element matrices. *BIT*, 36:86–100, 1996.

[29] Louis A. Hageman and David M. Young. *Applied Iterative Methods*. Academic Press, New York, 1981.

[30] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Nat. Bur. Stand. J. Res.*, 49:409–436, 1952.

[31] M.T. Jones and P.E. Plassmann. A parallel graph coloring heuristic. *SIAM J. Sci. Stat. Comput.*, 14, 1993.

[32] D.S. Kershaw. The incomplete cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *J. Comp. Phys.*, 26:43–65, 1978.

[33] L. Yu. Kolotilina and A. Yu. Yeremin. On a family of two-level preconditionings of the incomlete block factorization type. *Sov. J. Numer. Anal. Math. Modelling*, pages 293–320, 1986.

[34] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research, Nat. Bu. Stand.*, 45:255–282, 1950.

[35] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16:346–358, 1979.

[36] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.

[37] T.A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comp.*, 34:473–497, 1980.

[38] J.A. Meijerink and H.A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Math Comp*, 31:148–162, 1977.

[39] Gerard Meurant. Multitasking the conjugate gradient method on the CRAY X-MP/48. *Parallel Computing*, 5:267–280, 1987.

[40] Thomas Oppe and Wayne D. Joubert. Improved ssor and incomplete cholesky solution of linear equations on shared memory and distributed memory parallel computers. *Numerical Linear Algebra with Applications*, 1:287–311, 1994.

[41] C.C. Paige and M.A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[42] Claude Pommerell. *Solution of Large Unsymmetric Systems of Linear Equations*, volume 17 of *Series in Micro-electronics, volume 17*. Hartung-Gorre Verlag, Konstanz, 1992.

[43] Youcef Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston, 1996. available for download from `http://www.cs.umn.edu/~saad`.

[44] Yousef Saad. Practical use of some krylov subspace methods for solving indefinite and nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 5:203–228, 1984.

[45] Yousef Saad and Martin H. Schultz. GMRes: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[46] Peter Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 10:36–52, 1989.

[47] H. Stone. Iterative solution of implicit approximations of multidimensional partial differetntial equations. *SIAM J. Numer. Anal.*, 5:530–558, 1968.

[48] P.N. Swarztrauber. The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. *SIAM Review*, 19:490–501, 1977.

[49] H.A. van der Vorst. (M)ICCG for 2D problems on vectorcomputers. In A.Lichnewsky and C.Saguez, editors, *Supercomputing*. North-Holland, 1988. also as Report No.A-17, Data Processing Center, Kyoto University, Kyoto, Japan, December 17, 1986.

[50] H.A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Num. Lin. Alg. with Appls.*, 1:1–7, 1993.

[51] Henk van der Vorst. A vectorizable variant of some ICCG methods. *SIAM J. Sci. Stat. Comput.*, 3:350–356, 1982.

[52] Henk van der Vorst. High performance preconditioning. *SIAM J. Sci. Stat. Comput.*, 10:1174–1185, 1989.

[53] Henk van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[54] P.S. Vassilevski. Preconditioning nonsymmetric and indefinite finite element matrices. *J. Numer. Alg. Appl.*, 1:59–76, 1992.

[55] R. Weiss. *Convergence Behavior of Generalized Conjugate Gradient Methods*. PhD thesis, University of Karlsruhe, 1990.

[56] David M. Young. *Iterative method for solving partial differential equations of elliptic type*. PhD thesis, Harvard Univ., Cambridge, MA, 1950.

[57] David M. Young. *Iterative solution of large linear systems*. Academic Press, 1971.

[58] Lu Zhou and Homer F. Walker. Residual smoothing techniques for iterative methods. *SIAM J. Sci. Stat. Comput.*, 15:297–312, 1992.

Index