Iterative Methods for Large Linear Systems

Henk A. van der Vorst
Mathematical Institute
Utrecht University
Utrecht, The Netherlands
vorst@math.uu.nl

June 24, 2002

Contents

Chapter 1

Introduction

In these notes we present an overview of a number of related iterative methods for the solution of linear systems of equations. These methods are so-called Krylov projection type methods and they include popular methods such as Conjugate Gradients, MINRES, SYMMLQ, Bi-Conjugate Gradients, QMR, Bi-CGSTAB, CGS, LSQR, and GMRES. We will show how these methods can be derived from simple basic iteration formulas and how they are related.

Iterative methods are often used in combination with so-called preconditioning operators (easily invertible approximations for the operator of the system to be solved). We will give a brief overview of the various preconditioners that exist.

The purpose of these notes is to make the reader familiar with the ideas and the usage of iterative methods. We expect that guided with sufficient knowledge about the background of iterative methods, one will be able to make a proper choice for a particular class of problems. It will also provide guidance on how to tune these methods, in particular for the selection or construction of effective preconditioners.

For the application of iterative schemes one usually has linear sparse systems in mind, for instance linear systems arising in the finite element or finite difference approximations of (systems of) partial differential equations. However, the structure of the operators plays no explicit role in any of these schemes, and these schemes may also successfully be used to solve certain large dense linear systems. Depending on the situation, that may be attractive in terms of numbers of floating point operations.

Finally, we pay some attention to implementation aspects of these methods, especially for parallel computers.

Before we start the actual discussion of iterative methods, we will first give a motivation for their usage. As we will see, iterative methods are not only great fun to play with and interesting objects for analysis, but they are really useful in many situations. For truly large problems they may offer the only way towards solution, as we will see.

Acknowledgements I am very indebted to Michael Saunders who helped me to correct part of the text. The remaining errors are mine.

1.1 On the origin of iterative methods

In scientific computing most computational time is spent with solving systems of linear equations. These systems can be quite large, for instance, as in computational fluid flow problems where each equation describes how the value of a local unknown parameter (for example the local velocity of the flow) depends on (unknown) values in the near neighbourhood.

The actual computation is restricted to values on a previously constructed grid and the number of gridpoints determines the dimensions of the linear system. In Figure 1.1 we see such a grid for the computation of two-dimensional ocean flows. Each gridpoint is associated with one or more unknowns and equations that describe how these unknowns are related to unknowns for neighbouring gridpoints. These relations are dictated by the physical model. Because many gridpoints are necessary in order to have a realistic computational model, we will have many equations as a consequence. A nice property of these linear systems is that each equation contains only a few unknowns. The matrix of the system contains mainly zeros. This property will be of great importance for the solution methods, as we will see later.

We see that the grid consists of four differently represented subgrids. The reason for this is that in the actual computations for this problem, we had to do the work in parallel: in this case on 4 parallel computers. This made it possible to do the work in very short time and that is convenient for model studies. We will see that most of the methods that we will describe lend themselves for parallel computation.

As we will see, the process of solving the unknowns from these large linear systems involves much computational work. The approach via direct Gaus-

Figure 1.1: The computational grid for an ocean flow

sian elimination is often not attractive. This was already recognized by the great Gauss himself, in 1823, be it for different reasons as in present circumstances [84]. In that year he proposed an iterative method for the solution of 4 equations with 4 unknowns, arising from triangular measurements.

In order to appreciate his way of computing, we start with the familiar (Gaussian) elimination process. As an example we consider the small linear system

$$\begin{bmatrix} 10 & 0 & 1 \\ \frac{1}{2} & 7 & 1 \\ 1 & 0 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 21 \\ 9 \\ 8 \end{bmatrix}.$$

The elimination process proceeds as follows. One subtracts $\frac{1}{20}$ times the first row from the second row and then $\frac{1}{10}$ times the first row from the third one. After this we have zeros in the first column below the diagonal and the system becomes

$$\begin{bmatrix} 10 & 0 & 1 \\ 0 & 7 & 1-\frac{1}{20} \\ 0 & 0 & 6-\frac{1}{10} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 21 \\ 9-\frac{21}{20} \\ 8-\frac{21}{10} \end{bmatrix}.$$

As a coincidence we also have a zero element in the second column below the diagonal, and now we can solve the system without much effort. It leads to the solution $x_3 = 1$, $x_2 = 1$, and $x_1 = 2$. Note that we have used exact computation. This is not a problem in this case, which has been designed to have a "nice" solution. However, in more realistic situations, we may have non-integer values and then exact computation may lead to significant computational effort for a human being. It is not so easy to avoid human errors and after checking that the computed erroneous solution does not satisfy the initial system, it is not easy to find the place where the error occurred. Gauss suffered from this in his computations. He had a good physical intuition and he knew that the solution of his system should have components of about the same order of magnitude. Because his matrices had strongly dominating elements on the diagonal, he knew that the main contribution in the right hand side came from the components in the solution that had been multiplied by a diagonal element: in our example $10x_1$, $7x_2$, and $6x_3$. This implies that if we neglect the off-diagonal elements in our system:

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 21 \\ 9 \\ 8 \end{bmatrix},$$

we still may expect, from this perturbed system, a fairly good approximation for the solution of the unperturbed system; in our example: $x_1 = 2.1$, $x_2 = \frac{9}{7}$, and $x_3 = \frac{8}{6}$. Indeed, this is a crude approximation for the wanted solution. This way of approximation is still popular; it is known as a Gauss-Jacobi approximation, because the mathematician-astronomer Jacobi used it for the computation of perturbations in the orbits of the planets in our solar system.

Gauss made another intelligent improvement. He observed that we can approximate the original system better if we only replace non-zero elements in the strict upper triangular part. This leads to

$$\begin{bmatrix} 10 & 0 & 0 \\ \frac{1}{2} & 7 & 0 \\ 1 & 0 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 21 \\ 9 \\ 8 \end{bmatrix}.$$

This system has the solution $x_1 = 2.1$, $x_2 = \frac{7.95}{7}$, and $x_3 = \frac{5.9}{6}$. Indeed, this leads to an improvement (it should be noted that this is not always the case; there are situations where this approach does not lead to an improvement). The approach is known as the Gauss-Seidel approximation.

All together, we have obtained a worse solution for our small system for only a small reduction in the computational costs. At this point it is good to discuss the computational complexity. For a system with n equations and n unknowns we need $2(n-1)^2$ operations for creating zeros in the first column (if we ignore possible already present zeros. Then for the second column we need $2(n-2)^2$ operations. From this we conclude that for the elimination of all elements in the lower triangular part, we need about $\frac{2}{3}n^3$ operations. The costs for solving the resulting upper triangular system again requires roughly. n^2 operations, which is a relatively a minor cost for larger values of n. We may conclude that the costs for obtaining the exact solution are proportional to n^3. It is easy to see that the costs for computing only the Gauss-Seidel approximation are proportional to n^2 and it may be clear that this promises great advantages for larger systems.

The question now arises how to improve the obtained approximated solution at relatively low costs. Of course, Gauss had considered this aspect as well. In order to explain his approach, we will use matrix notation. This was not yet invented in Gauss' time and the lack of it makes the reading of his original description not so easy. We will write the system as

$$Ax = b,$$

with

$$A = \begin{bmatrix} 10 & 0 & 1 \\ \frac{1}{2} & 7 & 1 \\ 1 & 0 & 6 \end{bmatrix},$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \text{en} \quad \begin{bmatrix} 21 \\ 9 \\ 8 \end{bmatrix}.$$

The lower triangular part of A is denoted by L:

$$L = \begin{bmatrix} 10 & 0 & 0 \\ \frac{1}{2} & 7 & 0 \\ 1 & 0 & 6 \end{bmatrix}.$$

The Gauss-Seidel approximation is then obtained from solving the system

$$L\widetilde{x} = b.$$

For a correction to this solution we look for the "missing" part Δx:

$$A(\widetilde{x} + \Delta x) = b,$$

and this missing part satisfies the equation

$$A\Delta x = b - A\widetilde{x} \equiv r.$$

It is now an obvious idea to compute an approximation for Δx again with a Gauss-Seidel approximation, that is we compute $\widetilde{\Delta x}$ from

$$L\widetilde{\Delta x} = r,$$

and we correct our first approximation with this approximated correction $\widetilde{x}$:

$$\widetilde{\widetilde{x}} = \widetilde{x} + \widetilde{\Delta x}.$$

Of course, we can repeat this trick and that leads to the following simple iteration procedure:

$$x^{(i+1)} = x^{(i)} + L^{-1}(b - Ax^{(i)}),$$

where the vector $y = L^{-1}(b - Ax^{(i)})$ is computed by solving

$$Ly = b - Ax^{(i)}.$$

We try this process for our little linear system. In absence of further information on the solution, we start with $x^{(0)} = 0$. In Table 1.1 we display the results for the first three iteration steps.

We observe that in this case we improve the solution by about two decimals per iteration. Of course, this is not always the case. It depends on how strongly the diagonal elements dominate. For instance, for the ocean flow problems we have almost no diagonal dominance and Gauss-Seidel iteration

Table 1.1: Results for three Gauss-Seidel iterations

iteration	1	2	3
x_1	2.1000	2.0017	2.000028
x_2	1.1357	1.0023	1.000038
x_3	0.9833	0.9997	0.999995

is so slow that it is not practical in this bare form.

The computational costs per iteration step amount to roughly $2n^2$ operations (additions, subtractions, multiplications) for the computation of $Ax^{(i)}$, plus n^2 operations for the solution of the lower triangular system with L: in total $\approx 3n^2$ operations per iteration step. Solution via the direct Gaussian elimination process takes $\approx \frac{2}{3}n^3$ operations. This implies a gain in efficiency if we are satisfied with the approximations and if these are obtained after less than

$$(\frac{2}{3}n^3)/(3n^2) = \frac{2}{9}n$$

iterations.

Computation with this iteration method was very attractive for Gauss, not because of efficiency, but mainly because he did not have to compute the approximated solutions accurately. A few decimal places were sufficient. Unintentional errors and rounding errors "correct themselves" in the later iterations. Another advantage is that the residual $b - Ax^{(i)}$ has to be computed in each step, so that one can see in a glance how well the computed approximation satisfies the system. In his letter to colleague Gerling, Gauss was elated over this process and he mentioned that one could carry through the computations even half asleep, or one could think of more important things while computing.

The linear systems that Gauss had to solve, were strongly diagonally dominant and for that reason he could observe fast convergence. The Gauss-Seidel iteration process is much too slow for the very large linear systems that we see in many applications. For this reason there has been much research for faster methods and we will see the results of this later on in this text.

1.2 Arguments for iterative methods

For the solution of a linear system $Ax = b$, with A a non-singular n by n matrix, we have the choice between direct and iterative methods.

The usual pro arguments for iterative methods are based on economy of computer storage and (sometimes) CPU time. On the con side, it should be noted that the usage of iterative methods requires some expertise. If CPU-time and computer storage are not really at stake, then it would be unwise to consider iterative methods for the solution of a given linear system. The question remains whether there are situations where iterative solution methods are really preferable. In this section we will try to substantiate the pro arguments; the con arguments will appear in our more detailed presentation of iterative methods. We hope that the reader will feel sufficiently familiar, after reading these notes, with some of the more popular iterative methods in order to make a proper choice for classes of linear systems to be solved.

Dense linear systems, and sparse systems with a suitable non-zero structure, are most often solved by a so-called direct method, like for instance Gaussian elimination. A direct method leads, in the absence of rounding errors, in a finite and fixed amount of work to the exact solution of the given linear system. Rounding errors can be handled fairly well by pivoting strategies. Problems arise when the direct solution scheme becomes too expensive for the task.

For instance, the elimination steps in Gaussian elimination may cause quite some zero entries of a sparse matrix to become non-zero entries, and non-zero entries require storage as well as CPU time. This is what may make Gaussian elimination, even with strategies for the reduction of the so-called fill-in, expensive.

In order to get a more quantitative impression of this, we consider a sparse system related to discretization of a second order PDE over a (not necessarily regular) grid, with about m unknowns per dimension. Think, for instance, of a finite element discretization over an irregular grid [143]. In a 3D situation this leads typically to a bandwidth $\sim n^{\frac{2}{3}}$ ($\approx m^2$ and $m^3 \approx n$, $1/m$ the (average) gridsize).

Gaussian elimination is carried out in two steps: first the matrix A is factored into a lower triangular matrix L, and an upper triangular matrix U (after suitable permutations of rows and columns):

$$A = LU.$$

When taking proper account of the band structure, the number of flops is then usually $\mathcal{O}(nm^4) \sim n^{2\frac{1}{3}}$ [89, 62]. We make the caveat "usually", because it may happen that fill-in is very limited when the sparsity pattern of the matrix is special.
For 2D problems the bandwidth is $\sim n^{\frac{1}{2}}$, so that the number of flops for a direct method then varies like n^2.

Then, in the second step, one has to solve x from $LUx = b$, which, again, is done in two steps:
a. first solve y from $Ly = b$,
b. then solve x from $Ux = y$.

The LU factorization is the expensive part of the computational process; the solution of the two triangular systems is usually a minor cost item. If one has to solve many systems with different right-hand sides, then one has to factor the matrix only once, after which the cost for solving each system will vary like $n^{\frac{5}{3}}$ for 3D problems, and like $n^{\frac{3}{2}}$ for 2D problems.

In order to be able to quantify the amount of work for iterative methods, we have to be a little more specific. Let us assume that the given matrix is symmetric positive definite, in which case we may use the Conjugate Gradient (CG) method. The error reduction per iteration step of CG is $\sim \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$, with $\kappa = \|A\|_2 \|A^{-1}\|_2$ [40, 8, 88].
For discretized second order PDEs over grids with gridsize $\frac{1}{m}$, it can be shown that $\kappa \sim m^2$ (see, for instance, [143]). Hence, for 3D problems we have that $\kappa \sim n^{\frac{2}{3}}$, and for 2D problems: $\kappa \sim n$.
In order to have an error reduction by a factor of ϵ, the number j of iteration steps must satisfy

$$\left(\frac{1 - \frac{1}{\sqrt{\kappa}}}{1 + \frac{1}{\sqrt{\kappa}}} \right)^j \approx (1 - \frac{2}{\sqrt{\kappa}})^j \approx e^{-\frac{2j}{\sqrt{\kappa}}} < \epsilon.$$

For 3D problems, it follows that

$$j \sim -\frac{\log \epsilon}{2} \sqrt{\kappa} \approx -\frac{\log \epsilon}{2} n^{\frac{1}{3}},$$

whereas for 2D problems,

$$j \approx -\frac{\log \epsilon}{2} n^{\frac{1}{2}}.$$

If we assume the number of flops per iteration to be $\sim fn$ (f stands for the average number of nonzeros per row of the matrix and the overhead per

unknown introduced by the iterative scheme), then the required number of flops for a reduction of the initial error with ϵ is
- for 3D problems: $\sim -fn^{\frac{4}{3}} \log \epsilon$, and
- for 2D problems: $\sim -fn^{\frac{3}{2}} \log \epsilon$.

f is typically a modest number, say of order $10 - 15$.

From comparing the flopcounts for the direct scheme with those for the iterative CG method we conclude that the CG method may be preferable if we have to solve one system at a time, and if n is large, or f is small, or ϵ is modest.

If we have to solve many systems $Ax = b_k$ with different right-hand sides b_k, and if we assume their number to be so large that the costs for constructing the LU factorization of A is relatively small per system, then it seems likely that direct methods will be more efficient for 2D problems. For 3D problems this is unlikely, because the flops count for the two triangular solves associated with a direct solution method is proportional to $n^{\frac{5}{3}}$, whereas the number of flops for the iterative solver (for the model situation) varies like $n^{\frac{4}{3}}$.

1.3 An Example

The above given arguments are quite nicely illustrated by observations made by Horst Simon [154]. He predicted that by now we will have to solve routinely linear problems with some 5×10^9 unknowns. From extrapolation of the CPU times observed for a characteristic model problem, he estimated the CPU time for the most efficient direct method as $520,040$ years, provided that the computation can be carried out at a speed of 1 TFLOPs.

On the other hand, the extrapolated guess for the CPU time with preconditioned conjugate gradients, assuming still a processing speed of 1 TFLOPs, as 575 seconds. As we will see, the processing speed for iterative methods may be a factor lower than for direct methods, but, nevertheless, it is obvious that the differences in CPU time requirements are gigantic. The ratio of the two times is of order n, just as we might have expected from our previous arguments.
Also the requirements for memory space for the iterative methods are typically smaller by orders of magnitude. This is often the argument for the usage of iterative methods in 2D situations, when flop counts for both classes of methods are more or less comparable.

Remarks:

- With suitable preconditioning we may have $\sqrt{\kappa} \sim n^{\frac{1}{6}}$ and the flops count is then roughly proportional to

$$-fn^{\frac{7}{6}} \log \epsilon,$$

  see, e.g., [96, 10].

- Special methods may be even (much) faster for special problems: Fast Poisson Solvers [31, 163], multigrid [102, 192]. For more general problems, we see combinations of these methods with iterative schemes. For instance, iterative schemes can be used as smoothers for multigrid, or a multigrid cycle for an approximating regular problem may be used as a preconditioner for an iterative method for an irregular problem. Also, preconditioners have been designed with multigrid-like properties.

- For matrices that are not positive definite symmetric the situation can be more problematic: it is often difficult to find the proper iterative method or a suitable preconditioner. However, for methods related in some sense to CG, like GMRES, QMR, TFQMR, Bi-CG, CGS, and Bi-CGSTAB, we often see that the flops counts are similar to that for CG.

- Iterative methods can be attractive even when the matrix is dense. Again, in the positive definite symmetric case, if the condition number is $n^{2-2\varepsilon}$ then, since the amount of work per iteration step is $\sim n^2$, and the number of iteration steps $\sim n^{1-\varepsilon}$, the total work estimate is roughly proportional to $n^{3-\varepsilon}$, and this is asymptotically less than the amount of work for Choleski's method (the symmetric positive definite variant of Gaussian elimination) , which varies like $\sim n^3$. This says that the condition number has to be less than n^2 in order to make iterative methods potentially competitive for dense matrices.

- In many situations the condition number tells only part of the story. Methods like CG can be a good deal faster than the condition number predicts. This happens, for instance, when the eigenvalues of A have relatively big gaps at the lower end of the spectrum (see section 4.2).

Table 1.2: Speed in Megaflops for 50 Iterations of ICCG and CG

Machine	optimized ICCG	Scaled CG	Peak Performance
NEC SX-3/22 (2.9 ns)	607	1124	2750
CRAY Y-MP C90 (4.2 ns)	444	737	952
CRAY 2 (4.1 ns)	96	149	500
IBM 9000 Model 820	40	75	444
IBM 9121 (15 ns)	11	25	133
DEC Vax/9000 (16 ns)	10	17	125
IBM RS/6000-550 (24 ns)	18	21	81
CONVEX C3210	16	19	50
Alliant FX2800	2	3	40

1.4 Performance aspects

The question remains at the moment how well iterative methods can take advantage of modern computer architectures. From Dongarra's LINPACK benchmark [55] it may be concluded that the solution of a dense linear system can (in principle) be computed with computational speeds close to peak speeds on most computers. This is already the case for systems of, say, order 50000 on parallel machines with as many as 1024 processors.

In sharp contrast to the dense case are computational speeds reported in [59] for the preconditioned as well as the unpreconditioned conjugate gradient method (ICCG and CG, respectively). We list some of these results for classical vector computers, for regular sparse systems from a 7-point stencil discretisations of 3D elliptic PDEs, of order $n = 10^6$, in Table 1.2. We see that, especially for ICCG, the performance stays an order of magnitude below the theoretical peak of most machines.

A benchmark for representative components of iterative methods is proposed in [58]. In this benchmark the asymptotic speeds, that is the Mflops rates for large values of the order of the linear system, are computed for the standard unpreconditioned Conjugate Gradients method and the unpreconditioned GMRES(20) method. The performance of preconditioners, necessary in order to speed up the convergence, is measured separately. When a preconditioner is used, the overall performance of the iterative process is typically

Table 1.3: Performances in Mflops for modern processors

Processor	peak	cg	gmres	ilu
EV6	759	285	216	163
Athlon 600 MHz	154	43	44	34
SGI Origin	106	70	71	57
ALPHA 533 MHz	81	45	40	33
IBM 375 MHz	606	254	209	120
SUN 296 MHz	154	57	37	34
R1200 270 MHz	155	52	78	62
PPC G4 450 MHz	198	45	38	31
Pentium III 550 MHz	96	37	39	27

closer to the performance of the preconditioning part. In Table 1.3 we list some asymptotic performances for Conjugate Gradients (cg), GMRES(20) (gmres), and the popular preconditioner ILU, for more modern processors. The situation for these processors is certainly somewhat better than for the vector processors. However, note that high performances are now sought by coupling large numbers of these processors (massively parallel computing). Parallelism in preconditioned iterative methods is not a trivial matter. We will come back to this in our discussion on preconditioning techniques.

Chapter 2

Basic Iteration Methods

2.1 Introduction

The idea behind iterative methods is to replace the given system by some nearby system that can be more easily solved. That is, instead of $Ax = b$ we solve the simpler system $Kx_0 = b$ and take x_0 as an approximation for x. The iteration comes from the systematic way in which the approximation can be improved. Obviously, we want the correction z that satisfies

$$A(x_0 + z) = b.$$

This leads to a new linear system

$$Az = b - Ax_0.$$

Again, we solve this system by a nearby system, and most often one takes K again:

$$Kz_0 = b - Ax_0.$$

This leads to the new approximation $x_1 = x_0 + z_0$. The correction procedure can now be repeated for x_1, and so on, which gives us an iterative method. In some iteration methods one selects a cycle of different approximations K, as, for instance, in ADI. In such cases one can regard the approximation for x after one cycle, as being obtained from the approximation prior to the cycle with an implicitly constructed K that represents the full cycle. This observation is of importance for the construction of preconditioners.

For the basic iteration, introduced above, it follows that

$$x_{i+1} = x_i + z_i$$

$$
\begin{aligned}
&= x_i + K^{-1}(b - Ax_i) \\
&= x_i + \widetilde{b} - \widetilde{A}x_i, \qquad (2.1)
\end{aligned}
$$

with $\widetilde{b} = K^{-1}b$ and $\widetilde{A} = K^{-1}A$. We use K^{-1} only for notational purposes; we (almost) never compute inverses of matrices explicitly. When we speak of $K^{-1}b$, we mean the vector $\widetilde{b}$ that is solved from $K\widetilde{b} = b$, and likewise for $K^{-1}Ax_i$.

The formulation in (2.1) can be interpreted as the basic iteration for the preconditioned linear system

$$\widetilde{A}x = \widetilde{b}, \qquad (2.2)$$

with approximation $K = I$ for $\widetilde{A} = K^{-1}A$.

In order to simplify our formulas, we will from now on assume that if we have some preconditioner K, we apply our iterative schemes to the (preconditioned) system (2.2), and we will skip the superscipt $\widetilde{\cdot}$. This means that we iterate for $Ax = b$ with approximation $K = I$ for A. In some cases it will turn out to be more convenient to incorporate the preconditioner explicitly in the iteration scheme, but that will be clear from the context.

We have thus arrived at the well-known Richardson iteration:

$$x_{i+1} = b + (I - A)x_i = x_i + r_i, \qquad (2.3)$$

with the residual $r_i = b - Ax_i$.

Because relation (2.3) contains x_i as well as r_i, it cannot easily be analysed. Multiplication by $-A$ and adding b gives

$$b - Ax_{i+1} = b - Ax_i - Ar_i$$

or

$$
\begin{aligned}
r_{i+1} &= (I - A)r_i \qquad (2.4) \\
&= (I - A)^{i+1}r_0 \\
&= P_{i+1}(A)r_0. \qquad (2.5)
\end{aligned}
$$

In terms of the error $x - x_i$, we get

$$A(x - x_{i+1}) = P_{i+1}(A)A(x - x_0),$$

so that, for nonsingular A:

$$x - x_{i+1} = P_{i+1}(A)(x - x_0).$$

In these expressions P_{i+1} is a (special) polynomial of degree $i+1$. Note that $P_{i+1}(0) = 1$.
The expressions (2.4) and (2.5) lead to interesting observations. From (2.4) we conclude that

$$\|r_{i+1}\| \leq \|I - A\|\|r_i\|,$$

which shows that we have guaranteed convergence for all initial r_0 if $\|I - A\| < 1$. This puts restrictions on the preconditioner (remember that A represents the preconditioned matrix). We will see later that for the convergence of more advanced iterative schemes, we may drop the restriction $\|I - A\| < 1$.

Equation (2.5) is also of interest, because it shows that all residuals can be expressed in terms of powers of A times the initial residual. This observation will be crucial for the derivation of methods like the Conjugate Gradients method. It shows something more. Let us assume that A has n eigenvectors w_j, with corresponding eigenvectors λ_j. Then we can express r_0 in terms of the eigenvector basis as

$$r_0 = \sum_{j=1}^{n} \gamma_j w_j,$$

and we see that

$$r_i = P_i(A)r_0 = \sum_{j=1}^{n} \gamma_j P_i(\lambda_j) w_j.$$

This formula shows that the error reduction depends on how well the polynomial P_i damps the initial error components. It would be nice if we could construct iterative methods for which the corresponding error reduction polynomial P_i has better damping properties than for the standard iteration (2.3).

From now on we will also assume that $x_0 = 0$ to simplify future formulas. This does not mean a loss of generality, because the situation $x_0 \neq 0$ can be transformed with a simple shift to the system

$$Ay = b - Ax_0 = \bar{b}$$

for which obviously $y_0 = 0$.

With the simple Richardson iteration, we can proceed in different ways. One way is to include iteration parameters, for instance, by computing x_{i+1} as

$$x_{i+1} = x_i + \alpha_i r_i. \tag{2.6}$$

This leads to the error reduction formula

$$r_{i+1} = (I - \alpha_i A) r_i.$$

It follows that the error reduction polynomial P_i in this case can be expressed as

$$P_i = \prod_{j=1}^{i} (I - \alpha_j A).$$

Apparently, we can now construct methods for which the corresponding error polynomials have better damping properties. For instance, if the eigenvalues λ_j are all in the real interval $[a, b]$, with $a > 0$, then we can select the α_j as the zeros of a Chebyshev polynomial, shifted from $[-1, 1]$ to $[a, b]$ and scaled so that $P_i(0) = 1$, which leads to the *Chebyshev iteration method*. An obvious criticism could be that we have to select different sets of iteration parameters for different values of i (because all Chebyshev polynomials have different zeros), but it turns out that the recurrence relations for Chebyshev polynomials can be expoited to derive recurrence relations for the corresponding x_i. For further details on this, see [185, Chapter 5] or [90, Chapter 10.1.5].

An important consequence of this polynomial damping interpretation is that it is not longer necessary that $I - A$ has all its eigenvalues inside the unit ball. The eigenvalues may, in principle, be anywhere as long as we have a chance to construct, implicitly, iteration polynomials that damp the unwanted error components.

A seemingly different approach is to save all approximations x_i and to try to recombine them into something better. This may seem awkward, but after some reflection we see that all approximations can be expressed in polynomial form, and hence any possible combination as well. This shows that the Chebychev iteration is a member of this class, but we may hope that the approach leads to optimal methods; optimal in a sense to be defined later.

First we have to identify the subspace in which the successive approximate solutions are located. By repeating the simple Richardson iteration, we observe that

$$\begin{aligned}
x_{i+1} &= r_0 + r_1 + r_2 + \cdots + r_i && (2.7)\\
&= \sum_{j=0}^{i} (I - A)^j r_0 && (2.8)\\
&\in \operatorname{span}\{r_0, Ar_0, \ldots, A^i r_0\} && (2.9)
\end{aligned}$$

$$\equiv \quad K^{i+1}(A; r_0). \tag{2.10}$$

The m-dimensional space spanned by a given vector v, and increasing powers of A applied to v, up to the $(m-1)$-th power, is called the m dimensional Krylov subspace, generated with A and v, denoted by $K^m(A; v)$.

Apparently, the Richardson iteration, as it proceeds, delivers elements of Krylov subspaces of increasing dimension. This is also the case for the Richardson iteration (2.6) with parameters. Including local iteration parameters in the iteration would lead to other elements of the same Krylov subspaces. Let us write such an element still as x_{i+1}. Since $x_{i+1} \in K^{i+1}(A; r_0)$, we have that

$$x_{i+1} = Q_{i+1}(A) r_0,$$

with Q_{i+1} an arbitrary polynomial of degree $i+1$. It follows that

$$r_{i+1} = b - Ax_{i+1} = (I - AQ_{i+1}(A)) r_0 = \widetilde{P}_{i+1}(A) r_0, \tag{2.11}$$

with, just as in the standard Richardson iteration, $\widetilde{P}_{i+1}(0) = 1$. The standard Richardson iteration is characterized by the polynomial $P_{i+1}(A) = (I - A)^{i+1}$.

The consequence is that if we want to make better combinations of the generated approximations, then we have to explore the Krylov subspace.

2.2 The Krylov subspace approach

Methods that attempt to generate better approximations from the Krylov subspace are often referred to as Krylov subspace methods. Because optimality usually refers to some sort of projection, they are also called Krylov projection methods. The Krylov subspace methods, for identifying suitable $x \in \mathcal{K}^k(A; r_0)$, can be distinguished in four different classes (we will still assume that $x_0 = 0$):

1. The *Ritz-Galerkin approach*: Construct the x_k for which the residual is orthogonal to the current subspace: $b - Ax_k \perp \mathcal{K}^k(A; r_0)$.

2. The *minimum residual approach*: Identify the x_k for which the Euclidean norm $\|b - Ax_k\|_2$ is minimal over $\mathcal{K}^k(A; r_0)$.

3. The *Petrov-Galerkin approach*: Find an x_k so that the residual $b - Ax_k$ is orthogonal to some other suitable k-dimensional subspace.

4. The *minimum error approach*: Determine x_k in $A^T\mathcal{K}^k(A^T; r_0)$ for which the Euclidean norm $||x_k - x||_2$ is minimal.

The Ritz-Galerkin approach leads to well-known methods such as Conjugate Gradients, the Lanczos method, FOM, and GENCG. The minimum residual approach leads to methods like GMRES, MINRES, and ORTHODIR. The main disadvantage of these two approaches is that, for most unsymmetric systems, they lead to long and therefore expensive recurrence relations for the approximate solutions. This can be relieved by selecting other subspaces for the orthogonality condition (the Galerkin condition). If we select the k-dimensional subspace in the third approach as $\mathcal{K}^k(A^T; s_0)$, then we obtain the Bi-CG and QMR methods, and these methods indeed work with short recurrences. The fourth approach is not so obvious, but for $A = A^T$ it already becomes more natural. In this case it leads to the SYMMLQ method [137]. For the unsymmetric case it leads to the less well-known GMERR methods [190, 191]. Hybrids of these approaches have been proposed, like CGS, Bi-CGSTAB, BiCGSTAB(ℓ), TFQMR, FGMRES, and GMRESR.

The choice for a method is a delicate problem. If the matrix A is symmetric positive definite, then the choice is easy: Conjugate Gradients. For other types of matrices the situation is very diffuse. GMRES, proposed in 1986 by Saad and Schultz [151], is the most robust method, but in terms of work per iteration step it is also relatively expensive. Bi-CG, which was suggested by Fletcher in 1977 [76], is a relatively inexpensive alternative, but it has problems with respect to convergence: the so-called breakdown situations. This aspect has received much attention in the literature. Parlett et al [140] introduced the notion of look-ahead in order to overcome breakdowns and this was further perfected by Freund and Nachtigal [79], and by Brezinski and Redivio-Zaglia [28]. The theory for this look-ahead technique was linked to the theory of Padé approximations by Gutknecht [99]. Other contributions to overcome specific breakdown situations were made by Bank and Chan [15], and Fischer [74]. We discuss these approaches in the sections on Bi-CG and QMR.

The development of hybrid methods started with CGS, published in 1989 by Sonneveld [160], and was followed by Bi-CGSTAB, by van der Vorst in 1992 [177], and others. The hybrid variants of GMRES: Flexible GMRES and GMRESR, in which GMRES is combined with some other iteration scheme, were proposed in the mid-1990s.

A nice overview of Krylov subspace methods, with focus on Lanczos-based methods, is given in [78]. Simple algorithms and unsophisticated software for some of these methods are provided in the "Templates" book [18]. This book is complemented, with respect to theory, by the very elegant textbook [93], authored by Greenbaum. Iterative methods with much attention to various forms of preconditioning have been described in [10]. Another book on iterative methods was published by Saad [150]; it is very algorithm oriented, with, of course, a focus on GMRES and preconditioning techniques, like threshold ILU, ILU with pivoting, and incomplete LQ factorizations. A nice introduction to Krylov subspace methods, viewed from the standpoint of polynomial methods, can be found in [75].
An annotated entrance to the vast literature on preconditioned iterative methods is given in [29].

2.3 The Krylov subspace

In order to identify the approximations corresponding to the four different approaches, we need a suitable basis for the Krylov subspace; one that can be extended in a meaningful way for subspaces of increasing dimension. The obvious basis r_0, Ar_0, $\ldots$, $A^{i-1}r_0$ for $\mathcal{K}^i(A;r_0)$, is not very attractive from a numerical point of view, since the vectors $A^j r_0$ point more and more in the direction of the dominant eigenvector for increasing j (the power method!), and hence the basis vectors become dependent in finite precision arithmetic. It does not help to compute this nonorthogonal generic basis first and to orthogonalize it afterwards. The result would be that we have orthogonalized a very ill-conditioned set of basis vectors, which is numerically still not an attractive situation.

We now derive an orthogonal basis that, in exact arithmetic, spans the Krylov subspace. For this we follow ideas from [161, Ch. 4.3]. We start with the generic basis for $\mathcal{K}^{i+1}(A;r_0)$ and we denote the basis vectors by u_j:

$$u_j = A^{j-1}r_0.$$

We define the n by j matrix U_j as the matrix with columns u_1, $\ldots$, u_j. We leave the connection between A and U_i as an exercise.

Exercise 2.1 *Show that*

$$AU_i = U_i B_i + u_{i+1}e_i^T, \tag{2.12}$$

with e_i the i-th canonical basis vector in $\mathbb{R}^i$, and B_i an i by i matrix with $b_{j+1,j} = 1$ and all other elements zero.

The next step is to decompose U_i, still in exact arithmetic, as

$$U_i = Q_i R_i,$$

with $Q_i^T Q_i = I$ and R_i upper triangular. Then, with (2.12) it follows that

$$AQ_i R_i = Q_i R_i B_i + u_{i+1} e_i^T,$$

or

$$\begin{aligned} AQ_i &= Q_i R_i B_i R_i^{-1} + u_{i+1} e_i^T R_i^{-1} \\ &= Q_i \widetilde{H}_i + u_{i+1} e_i^T R_i^{-1}. \end{aligned} \tag{2.13}$$

Exercise 2.2 *Show that $\widetilde{H}_i$ is an upper Hessenberg matrix.*

We can also decompose U_{i+1} as $U_{i+1} = Q_{i+1} R_{i+1}$, and if we write the last column of R_{i+1} as $(\widetilde{r}, r_{i+1,i+1})^T$, that is

$$R_{i+1} = \begin{pmatrix} R_i & \widetilde{r} \\ 0 & r_{i+1,i+1} \end{pmatrix},$$

then it follows that

$$u_{i+1} = Q_i \widetilde{r} + r_{i+1,i+1} q_{i+1}.$$

In combination with (2.13), this gives

$$\begin{aligned} AQ_i &= Q_i(\widetilde{H}_i + \frac{1}{r_{i,i}}\widetilde{r}) + \frac{r_{i+1,i+1}}{r_{i,i}} q_{i+1} e_i^T \\ &= Q_i H_i + \alpha q_{i+1} e_i^T. \end{aligned} \tag{2.14}$$

From this expression we learn at least two things: first

$$Q_i^T A Q_i = H_i, \tag{2.15}$$

with H_i upper Hessenberg, and second

$$q_{i+1}^T A q_i = \alpha,$$

which, with $Q_{i+1}^T A Q_{i+1} = H_{i+1}$, leads to $\alpha = h_{i+1,i}$.

The implicit Q theorem [90, Th. 7.4.2] states that the orthogonal Q that reduces A to upper Hessenberg form is uniquely determined by $q_1 = \frac{1}{\|r_0\|} r_0$, except for signs (that is, q_j may be multiplied by -1). The orthogonality of the q_j basis gives us excellent opportunities to compute this basis in finite precision arithmetic.

Arnoldi [6] proposed to compute the orthogonal basis as follows. In fact, with Arnoldi's procedure we compute in a straight forward manner the columns of Q_i and the elements of H_i. Start with $v_1 \equiv r_0/\|r_0\|_2$. Then compute Av_1, make it orthogonal to v_1 and normalize the result, which gives v_2. The general procedure is as follows. Assuming we already have an orthonormal basis $v_1, \ldots, v_j$ for $K^j(A; r_0)$, this basis is expanded by computing $t = Av_j$ and by orthonormalizing this vector t with respect to $v_1, \ldots, v_j$. In principle the orthonormalization process can be carried out in different ways, but the most commonly used approach is then modified Gram-Schmidt procedure [90].

```
v_1 = r_0/||r_0||_2;
for j = 1,..,m-1
    t = A v_j;
    for i = 1,...,j
        h_{i,j} = v_i^T t;
        t = t - h_{i,j} v_i;
    end;
    h_{j+1,j} = ||t||_2;
    v_{j+1} = t/h_{j+1,j};
end
```

Figure 2.1: Arnoldi's method with modified Gram–Schmidt orthogonalization

This leads to an algorithm for the creation of an orthonormal basis for $\mathcal{K}^m(A; r_0)$, as in Fig 2.1. It is easily verified that $v_1, \ldots, v_m$ form an orthonormal basis for $\mathcal{K}^m(A; r_0)$ (that is, if the construction does not terminate at a vector $t = 0$). The orthogonalization leads, in exact arithmetic, to the relation that we have seen before (cf. (2.14), but now expressed in terms of the v_j. Let V_j denote the matrix with columns v_1 up to v_j then it follows that

$$AV_{m-1} = V_m H_{m,m-1}. \tag{2.16}$$

The m by $m-1$ matrix $H_{m,m-1}$ is upper Hessenberg, and its elements $h_{i,j}$ are defined by the Arnoldi algorithm.

From a computational point of view, this construction is composed of three basic elements: a matrix vector product with A, inner products, and vector updates. We see that this orthogonalization becomes increasingly expensive for increasing dimension of the subspace, since the computation of each $h_{i,j}$ requires an inner product and a vector update.

Note that if A is symmetric, then so is $H_{m-1,m-1} = V_{m-1}^T A V_{m-1}$, so that in this situation $H_{m-1,m-1}$ is tridiagonal. This means that in the orthogonalization process, each new vector has to be orthogonalized with respect to the previous two vectors only, since all other inner products vanish. The resulting three-term recurrence relation for the basis vectors of $K_m(A; r_0)$ is known as the *Lanczos method* [118] and some very elegant methods are derived from it. In this symmetric case the orthogonalization process involves constant arithmetical costs per iteration step: one matrix vector product, two inner products, and two vector updates.

2.3.1 A more accurate basis for the Krylov subspace

A more accurate implementation for the construction of an orthonormal basis, useful for ill-conditioned matrices A, was suggested by Walker [188]. He suggested using Householder reflections rather than modified Gram-Schmidt. An alternative is to do two iterations with (modified) Gram-Schmidt if necessary. This works as follows. If one wants to have a set of orthogonal vectors to almost working precision then one has to check, after the orthogonalization of a new vector with respect to the existing set, whether the resulting unnormalized vector is significantly smaller in norm than the new vector at the start of the orthogonalization step, say more than $\kappa < 1$ smaller. In that case we may have had cancellation effects, and we apply modified Gram-Schmidt once again. This is the basis for the refinement technique suggested in [46]. It leads to a set of vectors for which the mutual loss of orthogonality is limited to $1/\kappa$, in a relative sense. In the template in Fig. 2.2, we incorporate this technique into the Arnoldi algorithm.
Note that, in exact arithmetic, the constants ρ in Fig. 2.2 are equal to zero. It is easily verified that, in exact arithmetic, the $v_1, \ldots, v_m$ form an orthonormal basis for $\mathcal{K}^m(A; v)$ (that is, if the construction does not terminate at a vector $t = 0$).

```
v is a convenient starting vector
Select a value for κ, e.g., κ = .25
v_1 = v/||v||_2
for j = 1,..,m − 1
    t = Av_j
    τ_in = ||t||_2
    for i = 1,...,j
        h_{i,j} = v_i^* t
        t = t − h_{i,j} v_i
    end
    if ||t||_2/τ_in ≤ κ
        for i = 1,...,j
            ρ = v_i^* t
            t = t − ρ v_i
            h_{i,j} = h_{i,j} + ρ
        end
    endif
    h_{j+1,j} = ||t||_2
    v_{j+1} = t/h_{j+1,j}
end
```

Figure 2.2: The Arnoldi Method with refined Modified Gram-Schmidt

Chapter 3

Construction of approximate solutions

3.1 The Ritz-Galerkin approach

The Ritz-Galerkin conditions imply that $r_k \perp \mathcal{K}^k(A; r_0)$, and this is equivalent to

$$V_k^T(b - Ax_k) = 0.$$

Since $b = r_0 = \|r_0\|_2 v_1$, it follows that $V_k^T b = \|r_0\|_2 e_1$ with e_1 the first canonical unit vector in $\mathbb{R}^k$. With $x_k = V_k y$ we obtain

$$V_k^T A V_k y = \|r_0\|_2 e_1.$$

This system can be interpreted as the system $Ax = b$ projected onto the subspace $\mathcal{K}^k(A; r_0)$.
Obviously we have to construct the $k \times k$ matrix $V_k^T A V_k$, but this is, as we have seen, readily available from the orthogonalization process:

$$V_k^T A V_k = H_{k,k},$$

so that the x_k for which $r_k \perp \mathcal{K}^k(A; r_0)$ can be easily computed by first solving $H_{k,k} y = \|r_0\|_2 e_1$, and then forming $x_k = V_k y$. This algorithm is known as FOM or GENCG [151].

When A is symmetric, then $H_{k,k}$ reduces to a tridiagonal matrix $T_{k,k}$, and the resulting method is known as the *Lanczos* method [119]. When A is in addition positive definite then we obtain, at least formally, the *Conjugate Gradient* method. In commonly used implementations of this method, one implicitly forms an LU factorization for $T_{k,k}$, without generating $T_{k,k}$ itself, and this leads to very elegant short recurrences for the x_j and the corresponding r_j see Section 4.

The positive definiteness is necessary to guarantee the existence of the LU factorization, but it allows also for another useful interpretation. From the fact that $r_i \perp \mathcal{K}^i(A; r_0)$, it follows that $A(x_i - x) \perp \mathcal{K}^i(A; r_0)$, or $x_i - x \perp_A \mathcal{K}^i(A; r_0)$. The latter observation expresses the fact that the error is A-orthogonal to the Krylov subspace and this is equivalent to the important observation that $\|x_i - x\|_A$ is minimal[1]. For an overview of the history of CG and main contributions on this subject, see [87].

3.2 The minimum residual approach

The creation of an orthogonal basis for the Krylov subspace, with basis vectors $v_1, \ldots, v_{i+1}$, leads to

$$AV_i = V_{i+1}H_{i+1,i}, \tag{3.1}$$

where V_i is the matrix with columns v_1 to v_i. We look for an $x_i \in K^i(A; r_0)$, that is $x_i = V_i y$, for which $\|b - Ax_i\|_2$ is minimal. This norm can be rewritten, with $\rho \equiv \|r_0\|_2$, as

$$\|b - Ax_i\|_2 = \|b - AV_i y\|_2 = \| \rho V_{i+1} e_1 - V_{i+1} H_{i+1,i} y\|_2.$$

Now we exploit the fact that V_{i+1} is an orthonormal transformation with respect to the Krylov subspace $K^{i+1}(A; r_0)$:

$$\|b - Ax_i\|_2 = \|\rho e_1 - H_{i+1,i} y\|_2,$$

and this final norm can simply be minimized by solving the minimum norm least squares problem for the $i + 1$ by i matrix $H_{i+1,i}$ and right-hand side $||r_0||_2 e_1$. The least squares problem is solved by constructing a QR factorization of $H_{i+1,i}$, and because of the upper Hessenberg structure that can conveniently be done with Givens transformations [90].
The GMRES method is based upon this approach; see Section 5.

[1]The A-norm is defined by $\|y\|_A^2 = (y, y)_A \equiv (y, Ay)$, and we need the positive definiteness of A in order to get a proper inner product $(\cdot, \cdot)_A$.

3.3 The Petrov-Galerkin approach

For unsymmetric systems we cannot, in general, reduce the matrix A to a symmetric system in a lower-dimensional subspace, by orthogonal projections. The reason is that we cannot create an orthogonal basis for the Krylov subspace by a three-term recurrence relation [72]. We can, however, obtain a suitable non-orthogonal basis with a three-term recurrence, by requiring that this basis be orthogonal with respect to some other basis.
We start by constructing an arbitrary basis for the Krylov subspace:

$$h_{i+1,i}v_{i+1} = Av_i - \sum_{j=1}^{i} h_{j,i}v_j, \tag{3.2}$$

which can be rewritten in matrix notation as $AV_i = V_{i+1}H_{i+1,i}$. The coefficients $h_{i+1,i}$ define the norm of v_{i+1}, and a natural choice would be to select them such that $\|v_{i+1}\|_2 = 1$. In Bi-CG implementations, a popular choice is to select $h_{i+1,i}$ such that $\|v_{i+1}\|_2 = \|r_{i+1}\|_2$.

Clearly, we can not use V_i for the projection, but suppose we have a W_i for which $W_i^T V_i = D_i$ (an i by i diagonal matrix with diagonal entries d_i), and for which $W_i^T v_{i+1} = 0$. Then

$$W_i^T AV_i = D_i H_{i,i}, \tag{3.3}$$

and now our goal is to find a W_i for which $H_{i,i}$ is tridiagonal. This means that $V_i^T A^T W_i$ should be tridiagonal too. This last expression has a similar structure to the right-hand side in (3.3), with only W_i and V_i reversed. This suggests generating the w_i with A^T.

We start with an arbitrary $w_1 \neq 0$, such that $w_1^T v_1 \neq 0$. Then we generate v_2 with (3.2), and orthogonalize it with respect to w_1, which means that $h_{1,1} = w_1^T Av_1/(w_1^T v_1)$. Since $w_1^T Av_1 = (A^T w_1)^T v_1$, this implies that w_2, generated with

$$h_{2,1}w_2 = A^T w_1 - h_{1,1}w_1,$$

is also orthogonal to v_1.

This can be continued, and we see that we can create bi-orthogonal basis sets $\{v_j\}$ and $\{w_j\}$ by making the new v_i orthogonal to w_1 up to w_{i-1}, and then by generating w_i with the same recurrence coefficients, but with A^T instead of A.

Now we have that $W_i^T AV_i = D_i H_{i,i}$, and also that $V_i^T A^T W_i = D_i H_{i,i}$. This implies that $D_iH_{i,i}$ is symmetric, and hence $H_{i,i}$ is a tridiagonal matrix, which gives us the desired 3-term recurrence relation for the v_j's, and the w_j's. Note that $v_1,\ldots,v_i$ form a basis for $\mathcal{K}^i(A;v_1)$, and w_1, ..., w_i form a basis for $\mathcal{K}^i(A^T;w_1)$.

3.4 The minimum error approach

In SYMMLQ [137], with $A = A^T$ we minimize the norm of the error $x - x_k$, for $x_k = x_0 + AV_ky_k$, which means that y_k is the solution of the normal equations

$$V_k^TAAV_ky_k = V_k^TA(x - x_0) = V_k^Tr_0 = ||r_0||_2e_1. \qquad (3.4)$$

This system can be further simplified by exploiting the Lanczos relations (2.16), with $T_{k+1,k} \equiv H_{k+1,k}$:

$$V_k^TAAV_k = T_{k+1,k}^TV_{k+1}^TV_{k+1}T_{k+1,k} = T_{k+1,k}^TT_{k+1,k}. \qquad (3.5)$$

A stable way of solving this set of normal equations is based on an $L\widetilde{Q}$ decomposition of $T_{k+1,k}^T$, but note that this is equivalent to the transpose of the $Q_{k+1,k}R_k$ decomposition of $T_{k+1,k}$, constructed for MINRES (by Givens rotations), and where R_k is an upper tridiagonal matrix (only the diagonal and the first two codiagonals in the upper triangular part contain non-zero elements):

$$T_{k+1,k}^T = R_k^TQ_{k+1,k}^T.$$

This leads to

$$T_{k+1,k}^TT_{k+1,k}y_k = R_k^TR_ky_k = ||r_0||_2e_1,$$

from which the basic generating formula for SYMMLQ is obtained:

$$\begin{aligned} x_k &= x_0 + AV_kR_k^{-1}R_k^{-T}||r_0||_2e_1 \\ &= x_0 + V_{k+1}Q_{k+1,k}R_kR_k^{-1}R_k^{-T}||r_0||_2e_1 \\ &= x_0 + (V_{k+1}Q_{k+1,k})\ (L_k^{-1}||r_0||_2e_1), \end{aligned} \qquad (3.6)$$

with $L_k \equiv R_k^T$. The actual implementation of SYMMLQ [137] is based on an update procedure for $W_k \equiv V_{k+1}Q_{k+1,k}$, and on a three-term recurrence relation for $||r_0||_2L_k^{-1}e_1$.

In SYMMLQ it is possible to generate the Galerkin approximations as a side product. This means that for positive definite symmetric matrices the CG results can be reconstructed, at relatively low costs, from the SYMMLQ results [137]. This gives no advantage for positive definite matrices, but it can be used for indefinite symmetric matrices. The advantage over CG in that situation is that SYMMLQ avoids to construct the LU decomposition of $T_{k,k}$ and the latter may not exist (without pivoting), or be singular, for indefinite systems.

Chapter 4

The Conjugate Gradients method

As explained in section 3.1, the conjugate gradients method can be viewed as a variant of the Lanczos method. The method is based on relation (2.16), which for symmetric A reduces to $AV_i = V_{i+1}H_{i+1,i}$ with tridiagonal $H_{i+1,i}$. For the k-th column of V_k, we have that

$$Av_k = h_{k+1,k}v_{k+1} + h_{k,k}v_k + h_{k-1,k}v_{k-1}. \tag{4.1}$$

In the Galerkin approach, the new residual $b - Ax_{k+1}$ is orthogonal to the subspace spanned by $v_1, \ldots, v_k$, so that r_{k+1} is in the direction of v_{k+1}. Therefore, we can also select the scaling factor $h_{k+1,k}$ so that v_{k+1} coincides with r_{k+1}. This would be convenient, since the residual gives useful information on our solution, and we do not want to work with two sequences of auxiliary vectors.
From the consistency relation (2.11) we have that r_k can be written as

$$r_k = (I - AQ_{k-1}(A))r_0.$$

By inserting the polynomial expressions for the residuals in (4.1), and comparing the coefficient for r_0 in the new relation, we obtain

$$h_{k+1,k} + h_{k,k} + h_{k-1,k} = 0,$$

which defines $h_{k+1,k}$.
At the end of this section we will consider the situation where the recurrence relation terminates.

With R_i we denote the matrix with columns r_j:

$$R_i = (r_0, \ldots, r_{i-1}),$$

then we have

$$AR_i = R_{i+1} T_{i+1,i}, \tag{4.2}$$

where $T_{i+1,i}$ is a tridiagonal matrix (with $i+1$ rows and i columns); its non-zero elements are defined by the $h_{i,j}$.
Since we are looking for a solution x_i in $K^i(A; r_0)$, that vector can be written as a combination of the basis vectors of the Krylov subspace, and hence

$$x_i = R_i y.$$

(Note that y has i components).
Furthermore, the Ritz-Galerkin condition says that the residual for x_i is orthogonal with respect to r_0, …, r_{i-1}:

$$R_i^T (Ax_i - b) = 0,$$

and hence

$$R_i^T A R_i y - R_i^T b = 0.$$

Using equation (4.2), we obtain

$$R_i^T R_i T_{i,i} y = \|r_0\|_2^2 e_1$$

Since $R_i^T R_i$ is a diagonal matrix with diagonal elements $\|r_0\|_2^2$ up to $\|r_{i-1}\|_2^2$ we find the desired solution by solving y from

$$T_{i,i} y = e_1 \quad \Rightarrow \quad y \quad \Rightarrow \quad x_i = R_i y.$$

So far we have only used the fact that A is symmetric and we have assumed that the matrix T_i is not singular. The Krylov subspace method that has been derived here is known as the Lanczos method for symmetric systems [119]. We will exploit the relation between the Lanczos method and the conjugate gradients method for the analysis of the convergence behaviour of the latter method.

Note that for some $j \leq n-1$ the construction of the orthogonal basis must terminate. In that case we have that $AR_{j+1} = R_{j+1} T_{j+1,j+1}$. Let y be the solution of the reduced system $T_{j+1,j+1} y = e_1$, and $x_{j+1} = R_{j+1} y$. Then it follows that $x_{j+1} = x$, i.e., we have arrived at the exact solution, since $Ax_{j+1} - b = AR_{j+1} y - b = R_{j+1} T_{j+1,j+1} y - b = R_{j+1} e_1 - b = 0$ (we have assumed that $x_0 = 0$).

Exercise 4.1 *Show that exact termination for a $j \leq n-1$ also holds if $x_0 \neq 0$.*

The Conjugate Gradients method [106], CG for short, is a clever variant on the above approach, that saves storage and computational effort. If we follow naively the above sketched approach when solving the projected equations, then we see that we have to save all columns of R_i throughout the process in order to recover the current iteration vectors x_i. This can be done in a more memory friendly way. If we assume that the matrix A is in addition positive definite then, because of the relation

$$R_i^T A R_i = R_i^T R_i T_{i,i},$$

we conclude that $T_{i,i}$ can be transformed by a rowscaling matrix $R_i^T R_i$ into a positive definite symmetric tridiagonal matrix (note that $R_i^T A R_i$ is positive definite for $y \in \mathbb{R}^i$). This implies that $T_{i,i}$ can be LU decomposed without any pivoting:

$$T_{i,i} = L_i U_i,$$

with L_i lower bidiagonal, and U_i upper bidiagonal with unit diagonal. Hence

$$x_i = R_i y = R_i T_i^{-1} e_1 = (R_i U_i^{-1})(L_i^{-1} e_1). \tag{4.3}$$

We concentrate on the factors, placed between parentheses, separately.

1.

$$L_i = \begin{pmatrix} \delta_0 & & & & \\ \phi_0 & \delta_1 & & & \\ & \phi_1 & \ddots & & \\ & & \ddots & \ddots & \\ & & & \phi_{i-2} & \delta_{i-1} \end{pmatrix}$$

With $q \equiv L_i^{-1} e_1$ we have that q can be solved from $L_i q = e_1$. Hence $q_0 = 1/\delta_0$, and $q_{i-1} = -\phi_{i-2} q_{i-2}/\delta_{i-1}$, so that the elements of q can be computed in a recursive manner.

2. Defining $P_i \equiv R_i U_i^{-1}$, we have that

$$R_i = P_i U_i = \begin{pmatrix} & & & \\ p_0 & \cdots & p_{i-2} & p_{i-1} \\ & & & \end{pmatrix} \begin{pmatrix} 1 & \epsilon_0 & & \\ & 1 & \epsilon_2 & \ddots \\ & & \ddots & \epsilon_{i-2} \\ & & & 1 \end{pmatrix}$$

$$\Rightarrow \quad r_{i-1} = \epsilon_{i-2} p_{i-2} + p_{i-1},$$

so that the vector p_{i-1} can be recursively computed as

$$p_{i-1} = r_{i-1} - \epsilon_{i-2} p_{i-2}.$$

Exercise 4.2 *Show that in step i from this process we can simply expand L_i to L_{i+1}, and U_i to U_{i+1}. Hence, we can simply compute q_i and p_i, using the above obtained recursion relations with the new values ϕ_{i-1}, δ_i, and ϵ_{i-1}.*

Pasting the two recurrences together we obtain

$$x_i = \begin{pmatrix} p_0 & \cdots & p_{i-1} \end{pmatrix} \begin{pmatrix} \vdots \\ \vdots \\ q_{i-1} \end{pmatrix}$$

$$= x_{i-1} + q_{i-1} p_{i-1}.$$

In principle the method is not too complicated: one generates the tridiagonal matrix with a simple three-term recurrence, one factorizes this matrix and solves for both factors. However, as we will see, *it is not be necessary to generate $T_{i,i}$ explicitly*, we can obtain the required information in an easier way.
To see this, we simplify the notation for our recurrence relations and then we exploit the orthogonality properties of the underlying Lanczos method. First we write $\alpha_i \equiv q_i$, and $\beta_i \equiv e_i$.
Then our two-term recurrence relations can be recast as

$$\begin{aligned} p_{i-1} &= r_{i-1} + \beta_{i-2} p_{i-2} && (4.4)\\ x_i &= x_{i-1} + \alpha_{i-1} p_{i-1} && (4.5)\\ r_i &= r_{i-1} - \alpha_{i-1} A p_{i-1}. && (4.6) \end{aligned}$$

Exercise 4.3 *Show that $p_i^T A p_j = 0$ for $i \neq j$.*

The vector $\alpha_{i-1} p_{i-1}$ is the correction vector that leads to the new minimum of $\|x - x_i\|_A$. It is thus tangential to the surface $\|x - z\|_A = \|x - x_i\|_A$, for $z \in \mathcal{K}^{i+1}(A; r_0)$. The vectors p_j are A-orthogonal, and can be interpreted as *conjugate* (= A-orthogonal) *gradients* for $\|x - z\|_A$, as a function of z. This gave the method its name.

From $r_i^T r_{i-1} = 0$, we derive α_{i-1}:

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{r_{i-1}^T A p_{i-1}},$$

and using relation (4.4), we arrive at the elegant expression

$$\alpha_{i-1} = \frac{r_{i-1}^T r_{i-1}}{p_{i-1}^T A p_{i-1}}.$$

Exercise 4.4 *Show that* $r_{i-1}^T A p_{i-1} = p_{i-1}^T A p_{i-1}$.

For β_{i-1}, we can derive a similar elegant expression. First we multiply the recursion (4.4) for p_{i-1} by A:

$$Ap_{i-1} = Ar_{i-1} + \beta_{i-2} A p_{i-2},$$

and we eliminate Ap_{i-2} with the recurrence relation (4.6, which leads to

$$r_i = r_{i-1} - \alpha_{i-1} A r_{i-1} - \frac{\alpha_{i-1}\beta_{i-2}}{\alpha_{i-2}}(r_{i-2} - r_{i-1}). \tag{4.7}$$

Since $r_i^T r_{i-2} = 0$, we obtain

$$\begin{aligned} \beta_{i-2} = -\alpha_{i-2}\frac{r_{i-2}^T A r_{i-1}}{r_{i-2}^T r_{i-2}} &= -\alpha_{i-2}\frac{r_{i-1}^T A r_{i-2}}{r_{i-2}^T r_{i-2}} \\ &= \frac{r_{i-1}^T r_{i-1}}{r_{i-2}^T r_{i-2}} \quad \text{(cf. (4.7))}. \end{aligned}$$

Note that we need only two new inner products in iteration step i for the computation of the two iteration coefficients (precisely as many as for the Lanczos process).

Thus we have arrived at the well-known conjugate gradients method. The name stems from the property that the update vectors p_i, are A-orthogonal. Note that the positive definiteness of A is exploited only to guarantee safe decomposition of the implicitly generated tridiagonal matrix $T_{i,i}$. This suggests that the conjugate gradients method may also work for certain non-positive definite systems, but then at our own risk [136]. We will later see how other ways of solving the projected system lead to other well-known methods.

4.1 Computational notes

The standard (unpreconditioned) Conjugate Gradient algorithm for the solution of $Ax = b$ can be represented by the scheme in Figure 4.1.

x_0 is initial guess, $r_0 = b - Ax_0$
for $i = 1, 2, ...$
    $\rho_{i-1} = r_{i-1}^T r_{i-1}$
    if $i = 1$
        $p_i = r_{i-1}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $p_i = r_{i-1} + \beta_{i-1} p_{i-1}$
    endif
    $q_i = Ap_i$
    $\alpha_i = \rho_{i-1}/p_i^T q_i$
    $x_i = x_{i-1} + \alpha_i p_i$
    $r_i = r_{i-1} - \alpha_i q_i$
    if x_i accurate enough **then quit**
end

Figure 4.1: Conjugate Gradients without preconditioning

Exercise 4.5 *Consider the Conjugate Gradients scheme in Fig. 4.1 for a given linear system $Ax = b$, with starting vector x_0. Now consider the application of this CG scheme for the system*

$$\widetilde{A}\widetilde{x} = \widetilde{b}, \tag{4.8}$$

with $\widetilde{A} = Q^T AQ$, $\widetilde{x} = Q^T x$, and $\widetilde{b} = Q^T b$, and Q an orthonormal matrix: $Q^T Q = I$. Denote the computed variables for CG applied to (4.8) with a superscript $\widetilde{}$, and start the iteration with $\widetilde{x}_0 = Q^T x_0$.
Show that the scheme applied for (4.8) generates the same iteration constants as in the iteration process for $Ax = b$. Show also that $\|\widetilde{r}_i\|_2 = \|r_i\|_2$. Obviously, an orthonormal transformed system leads to the same CG iteration process.

Exercise 4.6 *Show that for studying the convergence behaviour of CG it is no restriction if we restrict ourselves to diagonal matrices A (except for rounding errors). Verify this by numerical experiments. Hint: use the results of the previous exercise.*

Exercise 4.7 *Do experiments with CG for suitably chosen $Ax = b$, and for the system that is orthogonally transformed with Q so that Q^TAQ is a diagonal matrix. Any difference between the convergence histories must be attributed to rounding errors. Is there much difference? Enough difference to prevent us from drawing conclusions on the convergence behaviour of CG if we do experiments with diagonal matrices?*

Exercise 4.8 *Do experiments with CG for diagonal matrices with only k different eigenvalues. What is the maximum number of iterations to obtain a reduction in the norm of the residual by, say, 10^{-10}? Explain this result. Hint: use the polynomial interpretation for the residuals in CG.*

CG is most often used in combination with a suitable approximation K for A and then K is called the preconditioner. We will assume that K is also positive definite. However, we cannot apply CG straight away for the explicitly preconditioned system $K^{-1}Ax = K^{-1}b$, as we suggested in the introduction, because $K^{-1}A$ is most likely not symmetric. One way out is to apply the preconditioner differently. Assume that K is given in factored form:

$$K = LL^T,$$

as is the case for ILU preconditioners.
We then apply CG for the symmetrically preconditioned system

$$L^{-1}AL^{-T}y = L^{-1}b,$$

with $x = L^{-T}y$.

This approach has the disadvantage that K must be available in factored form and that we have to backtransform the approximate solution afterwards. There is a more elegant alternative. Note first that the CG method can be derived for any choice of the inner product. In our derivation we have used the standard inner product $(x, y) = \sum x_i y_i$, but we have not used any specific property of that inner product. Now we make a different choice:

$$[x, y] \equiv (x, Ky).$$

Exercise 4.9 *Show that $[x, y]$ defines a proper inner product.*

It is easy to verify that $K^{-1}A$ is symmetric positive definite with respect to [,]:

$$\begin{aligned} [K^{-1}Ax, y] &= (K^{-1}Ax, Ky) = (Ax, y) \\ &= (x, Ay) = [x, K^{-1}Ay]. \end{aligned} \tag{4.9}$$

Hence, we can follow our CG procedure for solving the preconditioned system $K^{-1}Ax = K^{-1}b$, using the new [,]-inner product.
Apparently, we now are minimizing

$$[x_i - x, K^{-1}A(x_i - x)] = (x_i - x, A(x_i - x)),$$

which leads to the remarkable (and known) result that for this preconditioned system we still minimize the error in A-norm, but now over a Krylov subspace generated by $K^{-1}r_0$ and $K^{-1}A$.

In the computational scheme for preconditioned CG, displayed in Figure 4.2, for the solution of $Ax = b$ with preconditioner K, we have replaced the [,]-inner product again by the familiar standard inner product. E.g., note that with $\widetilde{r}_{i+1} = K^{-1}Ax_{i+1} - K^{-1}b$ we have that

$$\rho_{i+1} = [\widetilde{r}_{i+1}, \widetilde{r}_{i+1}]$$

$$= [K^{-1}r_{i+1}, K^{-1}r_{i+1}] = [r_{i+1}, K^{-2}r_{i+1}]$$

$$= (r_{i+1}, K^{-1}r_{i+1}),$$

and $K^{-1}r_{i+1}$ is the residual corresponding to the preconditioned system $K^{-1}Ax = K^{-1}b$.

Exercise 4.10 *In view of the fact that diagonal matrices can be used for studying the convergence behaviour of CG: is it really necessary to construct preconditioners in order to study the effect of clustering of eigenvalues?*
Do experiments with matrices that have clustered eigenvalues and compare the convergence histories with systems that have more uniformly distributed eigenvalue distributions. Does clustering make a difference?

Exercise 4.11 *For some classes of linear systems, particular preconditioners have the effect that they reduce the condition number by a factor, say* 4*, and that they cluster most of the eigenvalues around* 1*. Do experiments with diagonal matrices and study the effects. If preconditioning would make each iteration step twice as expensive, is preconditioning then likely to lead to efficient iteration processes? Do these experiments prove that preconditioning will be effective? What do they prove?*

The coefficients α_j and β_j, generated by the Conjugate Gradient algorithms, as in Figures 4.1 and 4.2, can be used to build the matrix $T_{i,i}$ in the

x_0 is initial guess, $r_0 = b - Ax_0$
for $i = 1, 2, ...$
  Solve $Kw_{i-1} = r_{i-1}$
  $\rho_{i-1} = r_{i-1}^T w_{i-1}$
  if $i = 1$
    $p_i = w_{i-1}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p_i = w_{i-1} + \beta_{i-1} p_{i-1}$
  endif
  $q_i = Ap_i$
  $\alpha_i = \rho_{i-1}/p_i^T q_i$
  $x_i = x_{i-1} + \alpha_i p_i$
  $r_i = r_{i-1} - \alpha_i q_i$
  if x_i accurate enough **then quit**
end

Figure 4.2: Conjugate Gradients with preconditioning K

following way:

$$T_{i,i} = \begin{pmatrix} \ddots & & & \\ \ddots & -\frac{\beta_{j-1}}{\alpha_{j-1}} & & \\ \ddots & \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}} & \ddots & \\ & -\frac{1}{\alpha_j} & \ddots & \\ & & \ddots & \end{pmatrix}. \tag{4.10}$$

With the matrix $T_{i,i}$ one can easily compute approximations for eigenvalues of A, the so-called *Ritz values.* To this end, note that

$$R_i^T A R_i = D_i T_{i,i},$$

where D_i is a diagonal matrix with diagonal elements $\|r_0\|_2^2$, up to $\|r_i\|_2^2$. Multiplication from the left and the right by the matrix $D_i^{-\frac{1}{2}}$ leads to

$$D_i^{-\frac{1}{2}} R_i^T A R_i D_i^{-\frac{1}{2}} = D_i^{\frac{1}{2}} T_{i,i} D_i^{-\frac{1}{2}}.$$

Exercise 4.12 *Show that the n by $i+1$ matrix $R_i D_i^{-\frac{1}{2}}$ is an orthonormal matrix, that is $(R_i D_i^{-\frac{1}{2}})^T R_i D_i^{-\frac{1}{2}} = I_{i+1,i+1}$.*

Apparently, the columns of $R_i D_i^{-\frac{1}{2}}$ span an orthonormal basis for $\mathcal{K}^{i+1}(A; r_0)$. A vector x from that subspace can be expressed as

$$x = R_i D_i^{-\frac{1}{2}} y.$$

If we take Raleigh quotients for A with vectors $x \neq 0$ from the Krylov subspace, we have

$$\begin{aligned} \rho(A, x) &\equiv \frac{(Ax, x)}{(x, x)} \\ &= \frac{(A R_i D_i^{-\frac{1}{2}} y, R_i D_i^{-\frac{1}{2}} y)}{(y, y)} \\ &= \frac{(D_i^{\frac{1}{2}} T_{i,i} D_i^{-\frac{1}{2}} y, y)}{(y, y)}. \end{aligned}$$

The eigenvalues of the matrix $D_i^{\frac{1}{2}} T_{i,i} D_i^{-\frac{1}{2}}$ represent the local extrema of the Rayleigh quotient for A with respect to the Krylov subspace. We denote the eigenpairs of $T_{i,i}$ by

$$\theta_j^{(i)}, z_j^{(i)}.$$

Exercise 4.13 *Show that if $D_i^{\frac{1}{2}} T_{i,i} D_i^{-\frac{1}{2}} z_j^{(i)} = \theta_j^{(i)} z_j^{(i)}$, then*

$$A R_i D_i^{-\frac{1}{2}} z_j^{(i)} - \theta_j^{(i)} R_i D_i^{-\frac{1}{2}} z_j^{(i)} \perp \mathcal{K}^{i+1}(A; r_0).$$

We conclude that the residual for the approximate eigenpair

$$\theta_j^{(i)}, x_j^{(i)} \equiv R_i D_i^{-\frac{1}{2}} z_j^{(i)}$$

is orthogonal to the current Krylov subspace. For this reason, this pair is usually referred to as a (Rayleigh -) Ritz value and its corresponding Ritz vector.

Exercise 4.14 *Show that the matrix $\widetilde{T}_{i,i} \equiv D_i^{\frac{1}{2}} T_{i,i} D_i^{\frac{1}{2}}$ is a symmetric matrix* *that can be expressed as the following symmetric tridiagonal matrix:*

$$\widetilde{T}_i = \begin{pmatrix} \ddots & & \\ \ddots & -\frac{\sqrt{\beta_{j-1}}}{\alpha_{j-1}} & \\ \ddots & \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}} & \ddots \\ & -\frac{\sqrt{\beta_j}}{\alpha_j} & \ddots \\ & & \ddots \end{pmatrix}.$$

The eigenvalues of the leading i^{th} order minor of this matrix are the Ritz values of A (for Figure 4.1) or the preconditioned matrix $K^{-1}A$ (for Figure 4.2) with respect to the i-dimensional Krylov subspace spanned by the first i residual vectors. The Ritz values approximate the (extremal) eigenvalues of the (preconditioned) matrix increasingly well. These approximations can be used to get an impression of the relevant eigenvalues. They can also be used to construct upperbounds for the error in the delivered approximation with respect to the solution [115, 104]. According to the results in [168] the eigenvalue information can also be used in order to understand or explain delays in the convergence behavior.

Exercise 4.15 *Do experiments with CG for several linear systems with diagonal matrices. Compute the Ritz values for T_i for some values of i. What observations can one make? Can some relation between the Ritz values and convergence of CG be discerned?*

4.2 The convergence of Conjugate Gradients

Exercise 4.16 *Show that $x - u = Q_i(A)(x - x_0)$, for some polynomial Q_i of degree i with $Q_i(0) = 1$, if $u \in x_0 + \mathcal{K}^i(A; r_0)$.*

The conjugate gradient method (here with $K = I$) constructs in the i^{th} iteration step an x_i, that can be written as

$$x_i - x = P_i(A)(x_0 - x), \tag{4.11}$$

such that $\|x_i - x\|_A$ is minimal over all polynomials P_i of degree i, with $P_i(0) = 1$. Let us denote the eigenvalues and the orthonormalized eigenvectors of A by λ_j, z_j. We write $r_0 = \sum_j \gamma_j z_j$. It follows that

$$r_i = P_i(A) r_0 = \sum_j \gamma_j P_i(\lambda_j) z_j$$

and hence

$$\|x_i - x\|_A^2 = \sum_j \frac{\gamma_j^2}{\lambda_j} P_i^2(\lambda_j).$$

Note that only those λ_j play a role in this process for which $\gamma_j = 0$. In particular, if A happens to be semidefinite, i.e., there is a $\lambda = 0$, then this is no problem for the minimization process as long as the corresponding coefficient γ is zero as well. The situation where γ is small, due to rounding errors, is discussed in [115].

Exercise 4.17 *Write $b = \sum_{j=1}^{n} \gamma_j z_j$. Consider only those values of j for which $\gamma_j \neq 0$. Let ℓ denote the number of* different *eigenvalues associated with these nonzero γ's. Show that CG delivers the exact solution in ℓ iteration steps. Hint: use the polynomial expression for the iterates.*

Upper bounds on the error (in A-norm) are obtained by observing that

$$\begin{aligned} \|x_i - x\|_A^2 &= \sum_j \frac{\gamma_j^2}{\lambda_j} P_i^2(\lambda_j) \leq \sum_j \frac{\gamma_j^2}{\lambda_j} Q_i^2(\lambda_j) \\ &\leq \max_{\lambda_j} Q_i^2(\lambda_j) \sum_j \frac{\gamma_j^2}{\lambda_j}, \end{aligned} \tag{4.12}$$

for any arbitrary polynomial Q_i of degree i with $Q_i(0) = 1$, where the maximum is taken, of course, only over those λ for which the corresponding $\gamma \neq 0$.
When P_i has zeros at all the different λ_j then $r_i = 0$. The conjugate gradients method tries to spread the zeros in such a way that $P_i(\lambda_j)$ is small in a weighted sense, i.e., $\|x_i - x\|_A$ is as small as possible.

We get suitable upper bounds by selecting appropriate polynomials for Q_i. A very well-known upper bound arises by taking for Q_i the i^{th} degree Chebyshev polynomial C_i transformed to the interval $[\lambda_{min}, \lambda_{max}]$ and scaled such that its value at 0 is equal to 1:

$$Q_i(\lambda) = \frac{C_i(\frac{2\lambda - (\lambda_{min} + \lambda_{max})}{\lambda_{max} - \lambda_{min}})}{C_i(-\frac{\lambda_{min} + \lambda_{max}}{\lambda_{max} - \lambda_{min}})}.$$

For the Chebychev polynomials C_i we have the following properties:

$$C_i(x) = \cos(i \arccos(x)) \quad \text{for} \quad -1 \leq x \leq 1,$$

$$|C_i(x)| \leq 1 \quad \text{for} \quad -1 \leq x \leq 1,$$

$$
\begin{aligned}
C_i(x) &= \frac{1}{2}\left[(x+\sqrt{x^2-1})^i + (x+\sqrt{x^2-1})^{-i}\right] \quad \text{for } x \geq 1, && (4.15)\\
|C_i(x)| &= |C_i(-x)|. && (4.16)
\end{aligned}
$$

With Q_i instead of the optimal CG-polynomial P_i, we have that

$$
\begin{aligned}
\|x_i - x\|_A^2 &\leq \sum \frac{\gamma_j^2}{\lambda_j} Q_i^2(\lambda_j)\\
&\leq \max_{\lambda_{min},\lambda_{max}} |Q_i^2(\lambda_j)|\ \|x_0 - x\|_A^2. && (4.17)
\end{aligned}
$$

Using properties of the Chebychev polynomials, we can now derive an elegant upperbound for the error in A-norm . First we note that

$$
|Q_i(\lambda)| \leq \frac{1}{|C_i(\frac{\lambda_{max}+\lambda_{min}}{\lambda_{max}-\lambda_{min}})|} \quad \text{for} \quad \lambda_{min} \leq \lambda \leq \lambda_{max}.
$$

Exercise 4.18 *Show that for* $0 < a < b$ *we have that*

$$
C_i(\frac{b+a}{b-a}) \geq \frac{1}{2}(x+\sqrt{x^2-1})^i,
$$

with $x = (b+a)/(b-a)$. *Note that this lower bound is increasingly sharp for increasing* i.

With $a \equiv \lambda_{min}$ and $b \equiv \lambda_{max}$, we have for $a \leq \lambda \leq b$ that

$$
\begin{aligned}
|Q_i \lambda| &= \frac{1}{|C_i(\frac{b+a}{b-a})|} && (4.18)\\
&\leq 2\left(\frac{1}{x+\sqrt{x^2-1}}\right)^i \quad \text{with} \quad x \equiv \frac{b+a}{b-a} && (4.19)\\
&= \left(x - \sqrt{x^2-1}\right)^i && (4.20)\\
&= \left(\frac{b+a-2\sqrt{ba}}{b-a}\right)^i && (4.21)\\
&= \left(\frac{\sqrt{b}-\sqrt{a}}{\sqrt{b}+\sqrt{a}}\right)^i. && (4.22)
\end{aligned}
$$

With $\kappa \equiv b/a = \lambda_{max}/\lambda_{min}$, we so obtain a well known upper bound for the A-norm of the error [40, 90, 8]:

$$
\|x_i - x\|_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|x_0 - x\|_A. \qquad (4.23)
$$

This upper bound shows that we have fast convergence for small condition numbers. It was shown by Axelsson [8] that similar upper bounds can be obtained for specific eigenvalue distributions. For instance, let us consider the situation that $\lambda_n > \lambda_{n-1}$. Then we take for Q_i, the polynomial

$$Q_i(\lambda) = \frac{\lambda_n - \lambda}{\lambda_n} C_{i-1}\left(\frac{2\lambda - (a+b)}{b-a}\right) / C_{i-1}\left(\frac{-(a+b)}{b-a}\right),$$

with $a \equiv \lambda_{min}$, $b \equiv \lambda_{n-1}$.
This shows that for the situation that λ_{n-1} is relatively much smaller than λ_n, the upper bound for the error for the CG process lags only one step behind the upper bound for a process with a condition number λ_{n-1}/λ_1.

Exercise 4.19 *Derive an upper bound for the A-norm of the error that shows that this error is at most two steps behind the upper bound for a process with condition number λ_{n-2}/λ_1.*

These types of upper bounds show that it is important to have small condition numbers, or, in the case of larger condition numbers, to have eigenvalue distributions with well separated eigenvalues that cause these large condition numbers. In that case we say that the (remaining) part of the spectrum is (relatively) clustered. The purpose of preconditioning is to reduce the condition number κ and/or clustering of the eigenvalues.

In [8] the situation is analysed where the eigenvalues of $K^{-1}A$ are in disjoint intervals.

4.2.1 Local effects in the convergence behavior

Upper bounds as in (4.23) show that we have global convergence, but they do not help us to explain all sorts of local effects in the convergence behavior of CG. A very well known effect is the so-called super linear convergence: in many situations one observes that the average speed of convergence seems to increase as the iteration proceeds. As we have seen, the conjugate gradients algorithm is just an efficient implementation of the Lanczos algorithm. The eigenvalues of the implicitly generated tridiagonal matrix T_i are the Ritz values of A with respect to the current Krylov subspace. It is known from Lanczos theory that these Ritz values converge towards the eigenvalues of A and that in general the extreme eigenvalues of A are well approximated during early iterations [118, 133, 139]. Furthermore, the speed of convergence depends on how well these eigenvalues are separated from the others (gap ratio) [139]. This will help us to understand the super linear convergence behavior of the conjugate gradient method (as well as other Krylov subspace methods). It can be shown that as soon as one of the extreme eigenvalues is modestly well approximated by a Ritz value, the procedure converges from then on as a process in which this eigenvalue is absent, i.e., a process with a reduced condition number. Note that super linear convergence behavior in this connection is used to indicate linear convergence with a factor that is gradually decreased during the process as more and more of the extreme eigenvalues are sufficiently well approximated.

The local convergence behavior of CG, and especially the occurrence of super linear convergence, was first explained in a qualitative sense in [40], and later in a quantitative sense in [168]. In both papers it was linked to the convergence of eigenvalues (Ritz values) of $T_{i,i}$ towards eigenvalues of $K^{-1}A$, for increasing i. We will follow here the exposition given in [168]. The main idea in that paper is to link the convergence of CG in a certain phase with a process in which one or more of the Ritz values are replaced by eigenvalues of the matrix. To that end we first have to make a link with the Ritz values.

As we have seen, the residual r_i can be expressed as

$$r_i = P_i(A)b, \tag{4.24}$$

with $r_i \perp \mathcal{K}^i(A, r_0)$. Let the orthonormal Lanczos basis for this Krylov subspace be given by v_1, v_2, $\ldots$, v_i. We have seen that r_i is a multiple of v_{i+1}. In the generic case that $v_{i+1} \neq 0$ (if the equal sign holds then we have exact termination), the polynomial P_i has exact degree i. We will now show that the Ritz values are zeros of P_i. We start with the basic relation (2.16), which is rewritten for the symmetric case as

$$AV_i = V_i T_{i,i} + t_{i+1,i} v_{i+1} e_i^T, \tag{4.25}$$

with $T_{i,i}$ a symmetric tridiagonal matrix. After post-multiplication with e_1 (the first canonical unit vector of length i) we obtain

$$Av_1 = V_i T_{i,i} e_1.$$

Pre-multiplying this with A leads to

$$\begin{aligned} A^2 v_1 &= AV_i T_{i,i} e_1 \\ &= (V_i T_{i,i} + t_{i+1,i} v_{i+1} e_i^T) T_{i,i} e_1 \\ &= V_i T_{i,i}^2 e_1 \quad \text{if} \quad i > 2 \end{aligned}$$

Exercise 4.20 *Show that*

$$A^i v_1 = V_i T_{i,i} e_1 + c v_{i+1}, \tag{4.26}$$

for some constant c.

Exercise 4.21 *Show that*

$$V_i^T P_i(A) v_1 = P_i(T_{i,i}) e_1 = 0, \tag{4.27}$$

with P_i the polynomial of (4.24).

Equation (4.27) shows that $P_i(T_{i,i})$ is at least singular. We will now show that $P_i(T_{i,i}$ is identical to zero for all eigenvectors of $T_{i,i}$.

The Ritz pairs θ, z of A with respect to $\mathcal{K}^i(A; v_1)$ are defined by

$$Az - \theta z \perp \mathcal{K}^i(A; v_1),$$

for $z \in \mathcal{K}^i(A; v_1)$. With $z = V_i y$ it follows that

$$V_i^T A V_i y - \theta y = 0,$$

so that the pair θ, y is an eigenpair of $T_{i,i}$. Let the eigenpairs of $T_{i,i}$ be denoted as

$$\theta_j^{(i)}, y_j^{(i)}.$$

Since $T_{i,i}$ is symmetric the normalized eigenvectors span a complete orthonormal basis for $\mathbb{R}^i$. This implies that we can express e_1 in terms of this eigenvector basis

$$e_1 = \sum_j \gamma_j y_j^{(i)}. \tag{4.28}$$

Assume that $\gamma_j = (y_j^{(i)}, e_1) = 0$, then on account of

$$T_{i,i} y_j^{(i)} = \theta_j^{(i)} y_j^{(i)},$$

we have that $(T_{i,i} y_j^{(i)}, e_1) = (y_j^{(i)}, T_{i,i} e_1) = 0$ and this implies that $(y_j^{(i)}, e_2) = 0$. It is now easy to prove that $y_j^{(i)} \perp \{e_1, e_2, \ldots, e_i\}$ and this leads to a contradiction.

A combination of (4.27) with (4.28) gives

$$\sum_j \gamma_j P_i(\theta_j^{(i)}) y_j^{(i)} = 0,$$

and since all γ_j are nonzero it follows that the i Ritz values $\theta_j^{(i)}$ are the zeros of P_i. Because $P_i(0) = 1$ (cf. (2.11), this polynomial can be fully characterized as

$$P_i(t) = \frac{(\theta_1^{(i)} - t)(\theta_2^{(i)} - t) \cdots (\theta_i^{(i)} - t)}{\theta_1^{(i)} \theta_2^{(i)} \cdots \theta_i^{(i)}}. \tag{4.29}$$

We can now further analyze the convergence behavior of CG by exploiting the characterization of the residual polynomial P_i. Let λ_j, z_j denote the eigenpairs of A, with normalized z_j. We write the initial error $x - x_0$ as

$$x - x_0 = \sum_1^n \mu_j z_j, \tag{4.30}$$

and this implies, on account of (4.11),

$$x - x_i = \sum_1^n \mu_j P_i(\lambda_j) z_j.$$

Now we replace the first component of x_i by the first component of x and we take the modified x_i as the starting vector $\bar{x}_0$ for another CG process (with the same A and b):

$$x - \bar{x}_0 = \sum_2^n \mu_j P_i(\lambda_j) z_j.$$

This new CG process generates $\bar{x}_k$, for which the error is characterized by a polynomial $\bar{P}_k$. Because of the optimality property of these polynomials, we may replace any of the residual polynomials by arbitrary other polynomials in order to derive upper bounds. We select

$$q_i(t) = \frac{(\lambda_1 - t)(\theta_2^{(i)} - t) \cdots (\theta_i^{(i)} - t)}{\lambda_1 \theta_2^{(i)} \cdots \theta_i^{(i)}} = \frac{\theta_1^{(i)}(\lambda_1 - t)}{\lambda_1(\theta_1^{(i)} - t)} P_i(t).$$

The polynomial $\bar{P}_k q_i$ takes the value 1 for $t = 0$, and it follows that

$$\begin{aligned} \|x - x_{i+k}\|_A^2 &\leq \|\bar{P}_k(A) q_i(A)(x - x_0\|_A^2 && (4.31) \\ &= \sum_2^n \lambda_j \bar{P}_k(\lambda_j)^2 q_i(\lambda_j)^2 \gamma_j^2. && (4.32) \end{aligned}$$

Defining

$$F_i \equiv \frac{\theta_1^{(i)}}{\lambda_1} \max_{j\geq 2} \left| \frac{\lambda_j - \lambda_1}{\lambda_j - \theta_1^{(i)}} \right|,$$

it follows that

$$|q_i(\lambda_j)| \leq F_i |P_i(\lambda_j)| \quad \text{for} \quad j \geq 2.$$

We can now further simplify the upper bound in (4.31) as

$$\begin{aligned} \|x - x_{i+k}\|_A^2 &\leq F_i^2 \sum_2^n \lambda_j \bar{P}_k(\lambda_j)^2 R_i(\lambda_j)^2 \gamma_j^2 \\ &= F_i^2 \|x - \bar{x}_j\|_A^2 \\ &\leq F_i^2 \frac{\|x - \bar{x}_j\|_A^2}{\|x - \bar{x}_0\|_A^2} \|x - x_i\|_A^2. \end{aligned} \tag{4.33}$$

We have now proved the result stated in Theorem 3.1 in [168]. The interpretation of this result is as follows. First note that when a Ritz value, say θ_1^i, is close to an eigenvalue, in this case λ_1, for some i then $F_i \approx 1$. The result in (4.33) says that we may then expect a reduction in the next steps that is bounded by the reduction that we obtain for a CG process for $Ax = b$ in which λ_1 is missing. If we define the effective condition number for that process as $\kappa_2 = \lambda_n/\lambda_2$, we have that

$$\|x - x_{i+k}\|_A \leq 2 \left(\frac{\sqrt{\kappa_2} - 1}{\sqrt{\kappa_2} + 1} \right)^k \|x - x_i\|_A.$$

This shows that after the i-th step we have a reduction that is bounded by an expression in which we see the condition number for the remaining part of the spectrum. In fact, expression (4.33 shows more. It shows that we do not need to have accurate approximations for λ_1 by the first Ritz values. For instance, if

$$\frac{\theta_1^{(i)} - \lambda_1}{\lambda_1} < 0.1 \quad \text{and} \quad \frac{\theta_1^{(i)} - \lambda_1}{\lambda_2 - \lambda_1} < 0.1,$$

then we already have that $F_i < 1.25$, which shows that we have from then on a reduction as for the reduced process except for a very modest factor. Things cannot be worse from then on, because it is easy to prove that $\theta_1^{(i)}$ is a strictly decreasing function towards λ_1 for increasing i. This means that F_i becomes also smaller during the further iterations.

In [168] other results are presented that shed more light on all sorts of special situations, such as almost double eigenvalues in the spectrum of A, or situations in which more Ritz values have converged to a number of successive eigenvalues. Not surprisingly, these results show that after (modest) approximations of Ritz values for the smallest m eigenvalues, we will see convergence of the CG process from then on, governed by a condition number $\kappa_m \equiv \lambda_n/\lambda_{m+1}$. Of course, similar results can be derived for convergence of Ritz values towards the largest eigenvalues, but usually removal of small eigenvalues lead to much smaller condition numbers.

Exercise 4.22 *Consider the matrix A of order* 100 *with eigenvalues* 1, 2, ..., 100. *Suppose we carry out a CG process with this A, for some right hand side b. What is the guaranteed speed of convergence after $\theta_1^{(i)} < 1.1$, for some i. How many eigenvalues at the upper end of the spectrum need to be well approximated by Ritz values in order to have a similar reduction with the effective condition number? In view of the fact that the convergence of Ritz values depends on the relative gap in the spectrum, we will see in the generic case that the smallest and the largest Ritz value converge about equally fast to the smallest, largest eigenvalue of A, respectively. Is it in this case meaningful to consider convergence of the largest Ritz value in order to explain an observed faster convergence?*

4.3 Parallelism and data locality in CG

For successful application of CG the matrix A must be symmetric positive definite. In other short recurrence methods, other properties of A may be desirable, but we will not exploit these properties explicitly in the discussion on parallel aspects.

Most often, the conjugate gradients method is used in combination with some kind of preconditioning. This means that the matrix A can be thought of as being multiplied by some suitable approximation K^{-1} for A^{-1}. Usually, K is constructed as an approximation of A, such that systems like $Ky = z$ are much easier to solve than $Ax = b$. Unfortunately, a popular class of preconditioners, based upon incomplete factorization of A, do not lend themselves well for parallel implementation.
For the moment we will assume that the preconditioner is chosen such that the parallelism in solving $Ky = z$ is comparable with the parallelism in computing Ap, for given p.

For CG it is also required that the preconditioner K besymmetric positive definite. This aspect will play a role in our discussions because it shows how some properties of the preconditioner can be used sometimes to our advantage for an efficient implementation.

x_0= initial guess; $r_0 = b - Ax_0$;
$q_{-1} = p_{-1} = 0; \beta_{-1} = 0$;
Solve w_0 from $Kw_0 = r_0$;
$s_0 = Aw_0$;
$\rho_0 = (r_0, w_0); \mu_0 = (s_0, w_0)$;
$\alpha_0 = \rho_0/\mu_0$;
for $i = 0, 1, 2, ...$
    $p_i = w_i + \beta_{i-1}p_{i-1}$;
    $q_i = s_i + \beta_{i-1}q_{i-1}$;
    $x_{i+1} = x_i + \alpha_i p_i$;
    $r_{i+1} = r_i - \alpha_i q_i$;
    if x_{i+1} accurate enough then quit;
    Solve w_{i+1} from $Kw_{i+1} = r_{i+1}$;
    $s_{i+1} = Aw_{i+1}$;
    $\rho_{i+1} = (r_{i+1}, w_{i+1})$;
    $\mu_{i+1} = (s_{i+1}, w_{i+1})$;
    $\beta_i = \frac{\rho_{i+1}}{\rho_i}$;
    $\alpha_{i+1} = \frac{\rho_{i+1}}{\mu_{i+1} - \rho_{i+1}\beta_i/\alpha_i}$;
end i;

Figure 4.3: Parallel CG; Chronopoulos and Gear variant

The scheme for preconditioned CG is given in Fig. 4.1. Note that in that scheme the updating of x and r can only start after the completion of the inner product required for α_i. Therefore, this inner product is a so-called synchronization point: all computation has to wait for completion of this operation. One can try to avoid such synchronization points as much as possible, or to formulate CG in such a way that synchronization points can be taken together. We will see such approaches further on.

Since on a distributed memory machine communication is required to assemble the inner product, it would be nice if we could proceed with other useful computation while the communication takes place. However, as we see from our CG scheme, there is no possibility to overlap this communication time with useful computation. The same observation can be made for the updating of p, which can only take place after the completion of the inner product for β_i. Apart from the computation of Ap and the computations with K, we need to load 7 vectors for 10 vector floatingpoint operations. This means that for this part of the computation only 10/7 floating point operation can be carried out per memory reference on average.

Several authors [36, 127, 128] have attempted to improve this ratio, and to reduce the number of synchronization points. In our formulation of CG there are two such synchronization points, namely the computation of both inner products. Meurant [127] (see also [147]) has proposed a variant in which there is only one synchronization point, though at the cost of possibly reduced numerical stability, and one additional inner product. In this scheme the ratio between computations and memory references is about 2.

In Figure 4.3, we show a variant that was proposed by Chronopoulos and Gear [36]. In this scheme all vectors need be loaded only once per pass of the loop, which leads to better exploitation of the data (improved data locality). However, the price is that we need $2n$ flops more per iteration step. Chronopoulos and Gear [36] claim stability, based upon their numerical experiments.

Instead of 2 synchronization points, as in the standard version of CG, we have now only one synchronization point, as the next loop can only be started when the inner products at the end of the previous loop have been assembled. Another slight advantage is that these inner products can be computed in parallel.

Chronopoulos and Gear [36] propose to improve further the data locality and parallelism in CG by combining s successive steps. Their algorithm is based upon the following property of CG. The residual vectors $r_0, ..., r_i$ form an orthogonal basis (assuming exact arithmetic) for the Krylov subspace spanned by $r_0, Ar_0, ..., A^{i-1}r_0$. When arrived at r_j, the vectors $r_0, r_1, ..., r_j$, $Ar_j, ..., A^{i-j-1}r_j$ also form a basis for this subspace. Hence, we may combine s successive steps by generating $r_j, Ar_j, ..., A^{s-1}r_j$ first, and then do the orthogonalization and the updating of the current solution with this blockwise extended subspace. This approach leads to a slight increase in flops in comparison with s successive steps of the standard CG, and also one additional matrix vector product is required per s steps.

The main drawback in this approach seems to be potential numerical instability. Depending on the spectral properties of A, the set $r_j, ..., A^{s-1}r_j$ may tend to converge to a vector in the direction of a dominating eigenvector, or, in other words, may tend to dependence for increasing values of s. The authors claim to have seen successful completion of this approach, with no serious stability problems, for small values of s. Nevertheless, it seems that s-step CG, because of these problems, has a bad reputation (see also [148]). However, a similar approach, suggested by Chronopoulos and Kim [37] for other processes such as GMRES, seems to be more promising. Several authors have pursued this research direction, and we will come back to this in section 7.3.

We consider still another variant of CG, in which there is a possibility to overlap all of the communication time with useful computations [52]. This variant, represented in Figure 4.4, is just a reorganized version of the original CG scheme, and is therefore roughly as stable. The key trick in this approach is to delay the updating of the solution vector by one iteration step. In fact, it is somewhat more stable because the inner product for ρ is computed in a more stable way. The computation guarantees that the inner product is always positive, even for highly ill-conditioned K.
Another advantage over the previous scheme is that no additional operations are required.
It is assumed that the preconditioner K can be written as $K = LL^T$. Furthermore, it is assumed that the preconditioner has a block structure, corresponding to the gridblocks assigned to the processors, so that communication (if necessary) can be overlapped with computation.

Now we discuss how this scheme may lead to an efficient parallel scheme, and how local memory (vector registers, cache, ...) can be exploited.

1. All computing intensive operations can be carried out in parallel. Only for the operations (2), (3), (7), (8), (9), and (0), communication between processors is required. We have assumed that the communication in (2), (7), and (0) can be largely overlapped with computation.

2. The communication required for the assembly of the inner product in (3) can be overlapped with the update for x (which could have been done in the previous iteration step).

3. The assembly of the inner product in (8) can be overlapped with the computation in (0). Also step (9) usually requires information such as the norm of the residual, which can be overlapped with (0).

4. Steps (1), (2), and (3) can be combined: the computation of a segment of p_i can be followed immediately by the computation of a segment of

$$
\begin{array}{ll}
x_0 = \text{ initial guess; } r_0 = b - Ax_0; & \\
p_{-1} = 0; \beta_{-1} = 0; \alpha_{-1} = 0; & \\
s = L^{-1} r_0; & \\
\rho_0 = (s, s); & \\
\textsf{for } i = 0, 1, 2, ... & \\
\quad w_i = L^{-T} s; & (0) \\
\quad p_i = w_i + \beta_{i-1} p_{i-1}; & (1) \\
\quad q_i = A p_i; & (2) \\
\quad \gamma = (p_i, q_i); & (3) \\
\quad x_i = x_{i-1} + \alpha_{i-1} p_{i-1}; & (4) \\
\quad \alpha_i = \frac{\rho_i}{\gamma}; & (5) \\
\quad r_{i+1} = r_i - \alpha_i q_i; & (6) \\
\quad s = L^{-1} r_{i+1}; & (7) \\
\quad \rho_{i+1} = (s, s); & (8) \\
\quad \textsf{if } r_{i+1} \textit{ small enough } \textsf{then} & (9) \\
\qquad x_{i+1} = x_i + \alpha_i p_i & \\
\qquad \text{quit;} & \\
\quad \beta_i = \frac{\rho_{i+1}}{\rho_i}; & \\
\textsf{end } i; &
\end{array}
$$

Figure 4.4: Parallel CG; Demmel *et al* variant

q_i (2), and this can be followed by the computation of a part of the inner product in (3). This saves on load operations for segments of p_i and q_i.

5. Depending on the structure of L, the computation of segments of r_{i+1} in (6) can be followed by operations in (7), which can be followed by the computation of parts of the inner product in (8), and the computation of the norm of r_{i+1}, required for (9).

6. The computation of β_i can be done as soon as the computation in (8) has been completed. At that moment, the computation for (1) can be started if the requested parts of w_i have been completed in (0).

7. If no preconditioner is used, then $w_i = r_i$, and steps (7) and (0) have to be skipped. Step (8) has to be replaced by $\rho_i = (r_{i+1}, r_{i+1})$. Now we

need useful computation in order to overlap the communication for this inner product. To this end, one might split the computation in (4) per processor in two parts. The first of these parts is computed in parallel in overlap with (3), while the parallel computation of the other parts is used in order to overlap the communication for the computation of ρ_i.

4.4 Parallel performance of CG

Some realistic 3D computational fluid dynamics simulation problems, as well as other problems, lead to the necessity to solve linear systems $Ax = b$ with a matrix of very large order, billions of unknowns, say. If not of very special structure, such systems are not likely to be solved by direct elimination methods. For such very large (sparse) systems we must exploit parallelism in combination with suitable solution techniques, such as iterative solution methods.

From a parallel point of view CG mimics very well parallel performance properties of a variety of iterative methods such as Bi-CG, CGS, BiCGSTAB, QMR, and others.
In this section we study the performance of CG on parallel distributed memory systems and we report on some supporting experiments on actual existing machines. Guided by our experiments we will discuss the suitability of CG for Massively Parallel Processing systems.

All computationally intensive elements in preconditioned CG (updates, inner products, and matrix vector operations) are trivially parallelizable for shared memory machines [56], except possibly for the preconditioning step: *Solve* w_{i+1} *from* $Kw_{i+1} = r_{i+1}$. For the latter operation parallelism depends very much on the choice for K. In this section we restrict ourselves to block Jacobi preconditioning, where the blocks have been chosen so that each processor can handle one block independently of the others. For other preconditioners that allow some degree of parallelism, see [56].

For a distributed memory machine at least some of the steps require communication between processors: the accumulation of inner products and the computation of Ap_i (depending on the non-zero structure of A and the distribution of the non-zero elements over the processors). We consider in some more detail the situation where A is a block-tridiagonal matrix of order N, and we assume that all blocks are of order $\sqrt{N}$:

$$A = \begin{pmatrix} A_1 & D_1 & & & \\ D_1 & A_2 & D_2 & & \\ & D_2 & \ddots & \ddots & \\ & & \ddots & & \\ & & & & \end{pmatrix},$$

in which the D_i are diagonal matrices, and the A_i are tridiagonal matrices. Such systems occur quite frequently in finite difference approximations in 2 space dimensions. Our discussion can easily be adapted to 3 space dimensions.

4.4.1 Processor configuration and data distribution

For simplicity we assume that the processors are connected as a 2D grid with $p \times p = P$ processors.

The data have been distributed in a straightforward manner over the processor memories and we have not attempted to exploit fully the underlying grid structure for the given type of matrix (in order to reduce communication as much as possible). In fact it will turn out that in our case the communication for the matrix vector product plays only a minor role for matrix systems of large size .

Because of symmetry only the 3 non-zero diagonals in the upper triangular part of A need to be stored, and we have chosen to store successive parts of length N/P of each diagonal in consecutive neighbouring processors. In Figure 4.5 we see which part of A is represented by the data in the memory of a given processor.

The blocks for block Jacobi are chosen to be the diagonal blocks that are available on each processor, and the various vectors (r_i, p_i, etc.), have been distributed likewise, i.e., each processor holds a section of length N/P of these vectors in its local memory.

4.4.2 Required Communication

Matrix vector product It is easily seen for a $2D$ processor grid (as well as for many other configurations, including hypercube and pipeline), that the matrix vector product can be completed with only neighbour-neighbour communication. This means that the communication costs do not increase for

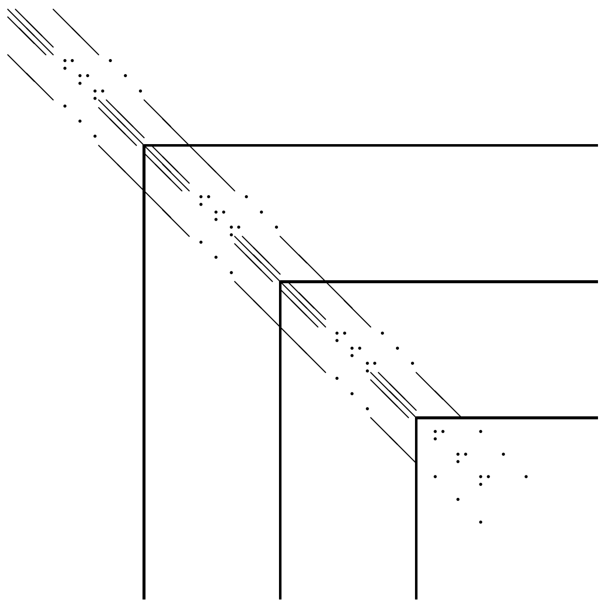

Figure 4.5: Distribution of A over the processors.

increasing values of p. If one follows a domain decomposition approach, in which the finite difference discretization grid is subdivided into p by p subgrids (p in x-direction and p in y-direction), then the communication costs are smaller than the computational costs by a factor of $\mathcal{O}(\frac{\sqrt{N}}{p})$.

In [47] much attention is given to this sparse matrix vector product and it is shown that the time for communication can almost completely be overlapped with computational work. Therefore, with adequate coding the matrix vector products do not necessarily lead to serious communication problems, even not for relatively small-sized problems.

Vector update In our case these operations do not require any communication and we should expect linear speed up when increasing the number of processors P.

Inner product For the inner product we need global communication for assembly and we need global communication for the distribution of the assembled inner product over the processors. For a $p \times p$ processor grid these communication costs are proportional to p. This means that for a constant length of the vector parts per processor, these communication costs will dominate for values of p large enough. This is quite unlike the situation for the matrix vector product and as we will see it may be a severely limiting factor in achieving high speed-ups in a massively parallel environment.

Our experiments, reported in [43, 44], and our modelling approach (see also [51]) clearly show that even a method like CG, which might be anticipated to be highly parallel, may suffer severely from the communication overhead due to the required inner products. Our study indicates that if we want reasonable speed-up in a massively parallel environment then the local memories should also be much larger when the number of processors is increased in order to accommodate systems large enough to compensate for the increased global communication costs.

Another approach for the reduction of the relative costs for communication is to do more useful computational work per iteration step, so that the communication for the two inner products takes relatively less time. One way to do this is to use polynomial preconditioning, i.e., the preconditioner consists of a number of matrix vector products with the matrix A. This may work well in situations where the matrix vector product requires only little (local) communication. Another way is to apply domain decomposition: the given domain is split into P, say, subdomains with estimated values for the solutions on the interfaces. Then all the subproblems are solved independently and in parallel. This way of approximating the solution may be viewed as a preconditioning step in an iterative method. In this way we do more computational work per communication step. Unfortunately, depending on the problem and on the way of decoupling the subdomains one may need a larger number of iteration steps for larger values of P, which may then, of course, diminish the overall efficiency of the domain decomposition approach. For more information on this approach we also refer to references given in [18].

4.5 CG and the Normal Equations

When faced with an unsymmetric system, then it seems rather obvious to form the system of normal equations $A^T Ax = A^T b$, and to solve this iteratively with conjugate gradients. However, this may have severe disadvantages because of the squaring of the condition number. This has as effects that the solution is more susceptible to errors in the right-hand side and that the rate of convergence of the CG procedure is much slower as for a comparable symmetric system with a matrix with the same condition number as A. Moreover, the amount of work per iteration step, necessary for the matrix vector product, is doubled. GMRES avoids to form the normal equations, but needs an increasing amount of storage and computational work per iteration. Therefore, the normal equations approach may still be attractive if only a few iterations suffice to get an acceptable small residual. To illustrate this, we note that when A is orthonormal, then CG solves $A^TAx = A^Tb$ in one single iteration because $A^TA = I$. GMRES would need many iterations for such an extremely well-conditioned system if A is indefinite (which is the case for nontrivial situations). Of course, solving orthonormal systems is an easy matter, but this example shows that if A is in some sense close to orthogonal then the normal equations approach with CG may be very attractive. Another way of stating this is that CG with the normal equations, in a way as formulated below in CGLS, may be attractive if the *singular values* of A are clustered. The Krylov subspace methods for the equation $Ax = b$ are attractive if the *eigenvalues of* A are clustered.

Paige and Saunders [138] point out that applying CG (see Fig. 4.1, with A replaced by A^TA) naively to the normal equations is not advisable, largely because the method would then generate vectors of the form A^TAp_i. This vector does not lead to to sufficiently accurate iteration coefficients α_i. An algorithm with better numerical properties is obtained after a slight algebraic rearrangement, where we make use of the intermediate vector Ap_i (see also [25], that report contains an early FORTRAN implementation). This leads to the CGLS algorithm in Fig. 4.6. Our formulation is slightly different from the one in [138], in oder to have it compatible with the CG algorithm in Fig. 4.1. The vector r_i denotes the residual for the system $Ax = b$.

The use of conjugate gradients (LSQR) in a least squares context, as well as a theoretical comparison with SIRT type methods, is discussed in [169] and [170].

Exercise 4.23 *Show that, in exact arithmetic, CGLS leads to x_i for which $\|b - Ax_i\|_2$ is minimal over the shifted Krylov subspace $x_0 + \mathcal{K}^i(A^TA; A^Tb)$.*

For overdetermined, or underdetermined, systems, solving the normal equations may be a quite natural approach. Now we have to face the situation that A may be ill-conditioned. In that case it may be numerically unattractive to apply the Lanczos procedure with A^TA to form the reduced tridiagonal system with respect to the Krylov subspace. This tridiagonal system may also be ill-conditioned. Paige and Saunders have proposed a method that behaves numerically better. Their LSQR [138] is in exact arithmetic equivalent to CGLS but gives better results for ill-conditioned systems (in

x_0 is initial guess, $r_0 = b - Ax_0$, $s_0 = A^T r_0$
for $i = 1, 2, \ldots.$
    $\rho_{i-1} = s_{i-1}^T s_{i-1}$
    if $i = 1$
        $p_i = s_{i-1}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $p_i = s_{i-1} + \beta_{i-1} p_{i-1}$
    endif
    $q_i = Ap_i$
    $\alpha_i = \rho_{i-1}/p_i^T q_i$
    $x_i = x_{i-1} + \alpha_i p_i$
    $r_i = r_{i-1} - \alpha_i q_i$
    $s_i = A^T r_i$
    if x_i accurate enough **then quit**
end

Figure 4.6: CGLS

particular, when the singular vectors associated with small singular vectors carry important information of the solution). We will now discuss LSQR in more detail.

We first note that the least squares problem $Ax = b$ is equivalent to the auxiliary square linear system

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{4.34}$$

Forming an orthonormal basis for the Krylov subspace for the system in (4.34, and starting with the vector

$$w_1 = \frac{1}{\|b\|_2} \begin{pmatrix} b \\ 0 \end{pmatrix},$$

we obtain as the second vector for the Krylov subspace:

$$\frac{1}{\|b\|_2}\begin{pmatrix} b \\ A^T b \end{pmatrix}.$$

After orthogonalizing this vector to w_1 and normalizing of the result, we obtain the second orthogonal basis vector

$$w_2 = \frac{1}{\|A^T b\|_2}\begin{pmatrix} 0 \\ A^T b \end{pmatrix}.$$

Exercise 4.24 *Show that we get alternatingly orthogonal basis vectors for the Krylov subspace of the form*

$$\begin{pmatrix} u \\ 0 \end{pmatrix} \quad \textit{and} \quad \begin{pmatrix} 0 \\ v \end{pmatrix}. \tag{4.35}$$

Let U_k denote the matrix of the first k u-vectors and let V_k denote the matrix of the first k v-vectors in their proper order. Then we have the following relation between these vectors:

$$\begin{array}{rcl} \beta_1 u_1 & = & b \quad \alpha_1 v_1 = A^T u_1 \\ i = 1, 2, \ldots & & \left\{ \begin{array}{l} \beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i \\ \alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i \end{array} \right. \end{array} \tag{4.36}$$

The scalars $\alpha_i > 0$ and $\beta_i > 0$ are chosen so that $\|u_i\|_2 = \|v_i\|_2 = 1$. These relations are also known as the bidiagonalization procedure, due to Golub and Kahan [86], for reasons that will now be made clear. With

$$\begin{array}{l} U_k \equiv [u_1, u_2, \ldots, u_k], \\ V_k \equiv [v_1, v_2, \ldots, v_k], \end{array} \quad B_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix},$$

it follows that

$$\beta_1 U_{k+1} e_1 = b, \tag{4.37}$$

$$A V_k = U_{k+1} B_k, \tag{4.38}$$

$$A^T U_{k+1} = V_k {B_k}^T + \alpha_{k+1} v_{k+1} e_{k+1}^T. \tag{4.39}$$

Exercise 4.25 *Show that $T_k = {B_k}^T B_k$, with T_k the reduced tridiagonal matrix obtained with the Lanczos process for $A^T Ax = A^T b$, and that this Lanczos process is characterized by*

$$A^T AV_k = V_k T_k + \gamma_{k+1} v_{k+1} e_{k+1}^T, \tag{4.40}$$

with $\gamma = \beta_{k+1}\alpha_{k+1}$.

Now we return to the augmented system (4.34) (that is equivalent to the normal equations). With the Lanczos method, we want to find a solution in a Krylov subspace $\mathcal{K}$ so that the residual is orthogonal to $\mathcal{K}$, that is, we want to find a $(\widetilde{r},\ \widetilde{x})^T \in \mathcal{K}$ so that

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \widetilde{r} \\ \widetilde{x} \end{bmatrix} - \begin{bmatrix} b \\ 0 \end{bmatrix} \perp \mathcal{K}. \tag{4.41}$$

When we increase the dimension of $\mathcal{K}$ then $\widetilde{x}$ should approximate x and $\widetilde{r}$ should approximate $r = b - Ax$. In view of relation (4.40) the subspace $\mathcal{K}$ should be such that it permits $\widetilde{x}$ of the form $x_k \equiv \widetilde{x} = V_k y_k$. For the corresponding $r_k = b - Ax_k$, we then should have

$$\begin{aligned} r_k &= b - AV_k x_k \\ &= b - U_{k+1} B_k y_k \\ &= U_{k+1}(\beta_1 - B_k y_k) \\ &\equiv U_{k+1} t_{k+1}. \end{aligned}$$

This shows that $\mathcal{K}$ should be spanned by the columns of

$$\begin{bmatrix} U_{k+1} & 0 \\ 0 & V_k \end{bmatrix},$$

and note that these columns are generated by the first $2k+1$ steps of the Lanczos process for the augmented system (4.34), although not in that order.

With $\widetilde{r} = U_{k+1} t_{k+1}$ and $\widetilde{x} = V_k y_k$, the orthogonality relations (4.41) lead to

$$\begin{bmatrix} U_{k+1}^T & 0 \\ 0 & V_k^T \end{bmatrix} \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} U_{k+1} t_{k+1} \\ V_k x_k \end{bmatrix} - \begin{bmatrix} b \\ 0 \end{bmatrix} = 0.$$

This leads to the reduced system

$$\begin{bmatrix} I & B_k \\ B_k^T & 0 \end{bmatrix} \begin{bmatrix} t_{k+1} \\ y_k \end{bmatrix} = \begin{bmatrix} \beta_1 e_1 \\ 0 \end{bmatrix}. \tag{4.42}$$

Now note that (4.42) is equivalent to the least-squares problem

$$\min \|\beta_1 e_1 - B_k y_k\|_2, \tag{4.43}$$

and this forms the basis for LSQR.

In LSQR [138] the linear least-squares problem (4.43 is solved via the standard QR factorization (see, for instance, [90, Chapter 5]) for the matrix B_k. The better numerical properties of LSQR, with respect to CG for the normal equations, stem from the fact that we form the bidiagonal factors of the Lanczos tridiagonal matrix and that we use the QR method for solving the reduced least-squares problem instead of solving the reduced tridiagonal problem (thus avoiding additional problems if this reduced tridiagonal system is ill-conditioned).

For completeness we give the LSQR algorithm, as formulated in [138], in Fig. 4.7. For further details we refer to [138] and we note that more sophisticated implementations of LSQR are available in netlib:
`http://www.netlib.org`

An interesting variant that is also based on the bidiagonalization of A is the so-called Craig's method [138]. The easiest way to think of this method is to apply Conjugate Gradients to the system $A^T A x = A^T b$, with the following choice for the inner product

$$[x, y] \equiv (x, (A^T A)^{-1} y),$$

which defines a proper inner product if A is of full rank (see section 4.1). First note that the two inner products in CG (as in section 4.1 can be computed without inverting $A^T A$:

$$[p_i, A^T A p_i] = (p_i, p_i),$$

and, assuming that $b \in R(A)$ so that $Ax = b$ has a solution x:

$$[r_i, r_i] = [A^T(Ax_i - b), A^T(Ax_i - b)] = [A^T A(x_i - x), A^T(Ax_i - b)] =$$

$$(x_i - x, A^T(Ax_i - b)) = (Ax_i - b, Ax_i - b).$$

Apparently, with CG we are minimizing

$$[x_i - x, A^T A(x_i - x)] = (x_i - x, x_i - x) = \|x_i - x\|_2^2,$$

that is, in this approach the Euclidean norm of the error is minimized, over the same shifted subspace $x_0 + K^i(A^T A' A^T r_0)$, as in the standard

$$
\begin{array}{l}
x_0 = 0,\ \beta_1 = \|b\|_2,\ u_1 = \|b\|_2/\beta_1, \\
v = A^T u_1,\ \alpha_1 = \|v\|_2,\ w_1 = v_1 = v/\alpha_1 \\
\widetilde{\phi}_1 = \beta_1,\ \widetilde{\rho}_1 = \alpha_1 \\
\text{for } i = 1, 2, \ldots. \\
\quad u = Av_i - \alpha_i u_i,\ \beta_{i+1} = \|u\|_2,\ u_{i+1} = u/\beta_{i+1} \\
\quad v = A^T u_{i+1} - \beta_{i+1} v_i,\ \alpha_{i+1} = \|v\|_2,\ v_{i+1} = v/\alpha_{i+1} \\
\quad \rho_i = \sqrt{\widetilde{\rho}_i^{\,2} + \beta_{i+1}^2} \\
\quad c_i = \widetilde{\rho}_i/\rho_i \\
\quad s_i = \beta_{i+1}/\rho_i \\
\quad \theta_{i+1} = s_i \alpha_{i+1} \\
\quad \widetilde{\rho}_{i+1} = -c_i \alpha_{i+1} \\
\quad \phi_i = c_i \widetilde{\phi}_i \\
\quad \widetilde{\phi}_{i+1} = s_i \widetilde{\phi}_i \\
\quad x_i = x_{i-1} + (\phi_i/\rho_i) w_i \\
\quad w_{i+1} = v_{i+1} - (\theta_{i+1}/\rho_i) w_i \\
\quad \textbf{if } x_i \text{ accurate enough } \textbf{then quit} \\
\textbf{end}
\end{array}
$$

Figure 4.7: LSQR

approach. Note, however, that the rate of convergence of Craig's method is also determined by the condition number of A^TA, so that this method is also only attractive if one has a good preconditioner for A^TA.
It is also possible to interpret Craig's method as a solution method for the system $AA^Tu = b$, with $x = A^Tu$. This leads to the same algorithm as sketched in the approach discussed here. For a convenient introduction to the AA^T-approach, see [150].

4.6 Further references

A more formal presentation of CG, as well as many theoretical properties, can be found in the textbook by Hackbusch [102]. A shorter presentation is given in [90]. An overview of papers, published in the first 25 years of existence of the method, is given in [87]. Vector processing and parallel computing aspects are discussed in [57] and [132].