Numerical Linear Algebra

Preconditioning, numerical software and parallelisation

Gerard Sleijpen and Martin van Gijzen

November 27, 2007

Program November 27

- Preconditioning
  - Diagonal scaling, Gauss-Seidel, SOR and SSOR
  - Incomplete Choleski and Incomplete LU
- Numerical software
- Parallelisation
  - Shared memory versus distributed memory
  - Domain decomposition

Introduction

We already saw that the performance of iterative methods can be improved by applying a preconditioner. Preconditioners are a key to successful iterative methods. In general they are very problem dependent.

Today we will discuss some standard preconditioners and some ideas behind advanced preconditioning techniques.

We will also discuss some efforts to standardise numerical software.

Finally we will discuss how to perform scientific computations on a parallel computer.

Preconditioning (1)

A preconditioned iterative solver solves the system

$$M^{-1}Ax = M^{-1}b \ .$$

The matrix M is called the preconditioner.

The preconditioner should satisfy certain requirements:

- Convergence should be much faster for the preconditioned system than for the original system. Normally this means that M is constructed as an easily invertible approximation to A. Note that if $M = A$ any iterative method converges in one iteration.
- Operations with M^{-1} should be easy to perform ("cheap").

Preconditioning (2)

Of course the matrix $M^{-1}A$ is not explicitly formed. The multiplication $u = M^{-1}Av$ can simply be carried out by the operations

$$t = Av; \quad u = M^{-1}t$$

Preconditioning (3)

Preconditioners can be applied in different ways:

From the left

$$M^{-1}Ax = M^{-1}b \ ,$$

centrally

$$M = LU; \quad L^{-1}AU^{-1}y = L^{-1}b; \quad x = U^{-1}y \ ,$$

or from the right

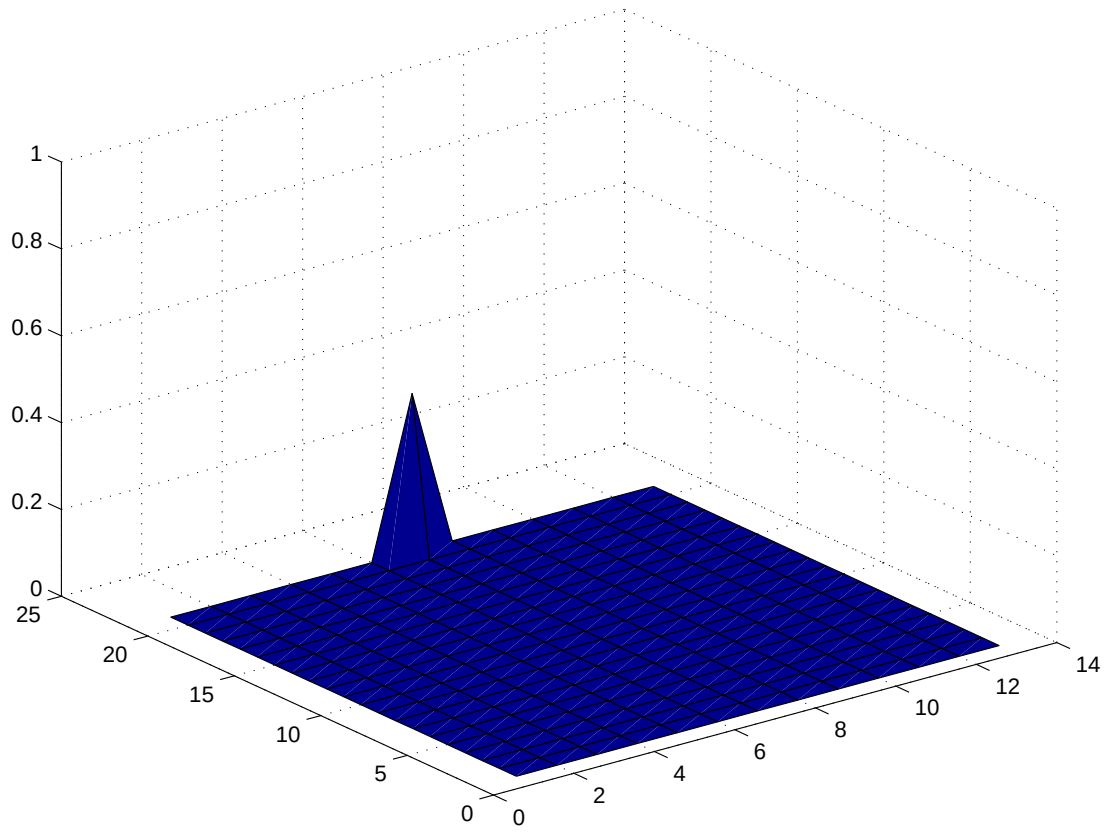
$$AM^{-1}y = b; \quad x = M^{-1}y \ .$$

Preconditioning (4)

Left, right and central preconditioning gives the same spectrum.
Yet there are other differences:

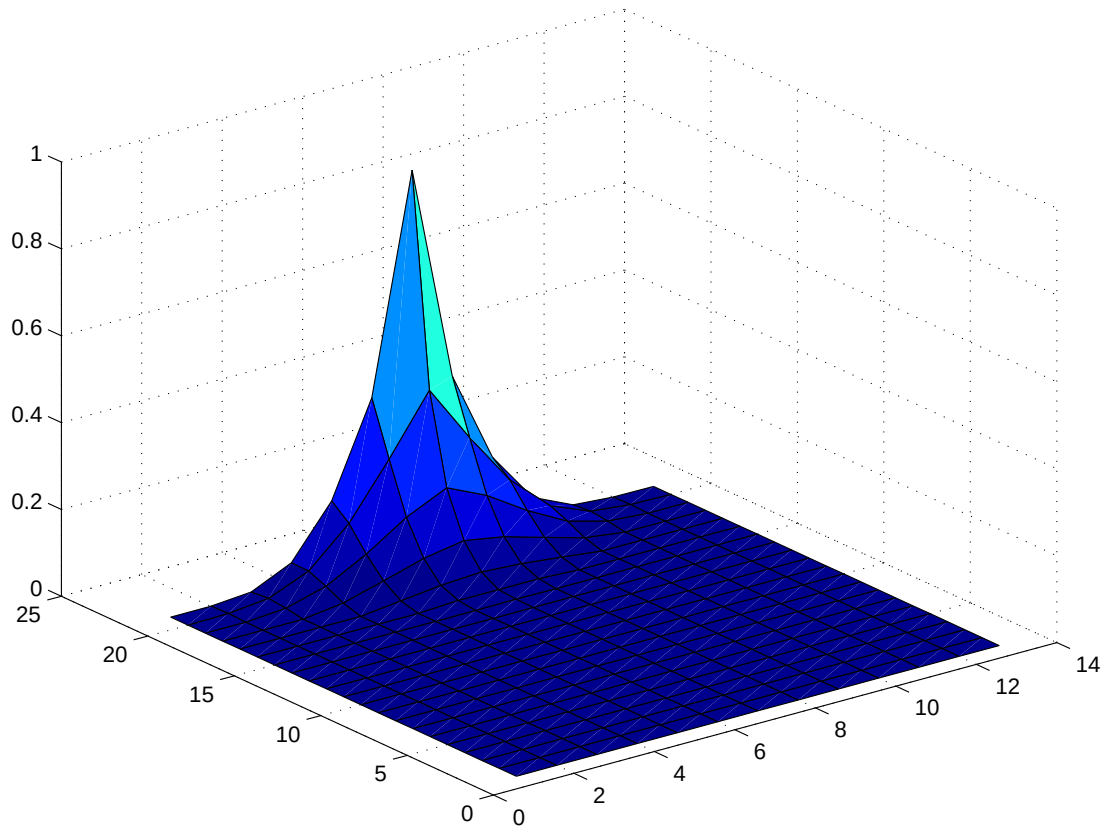
- Left preconditioning is most natural: no extra step is required to compute x;
- Central preconditioning preserves symmetry;
- Right preconditioning does not affect the residual norm.

Why preconditioners?



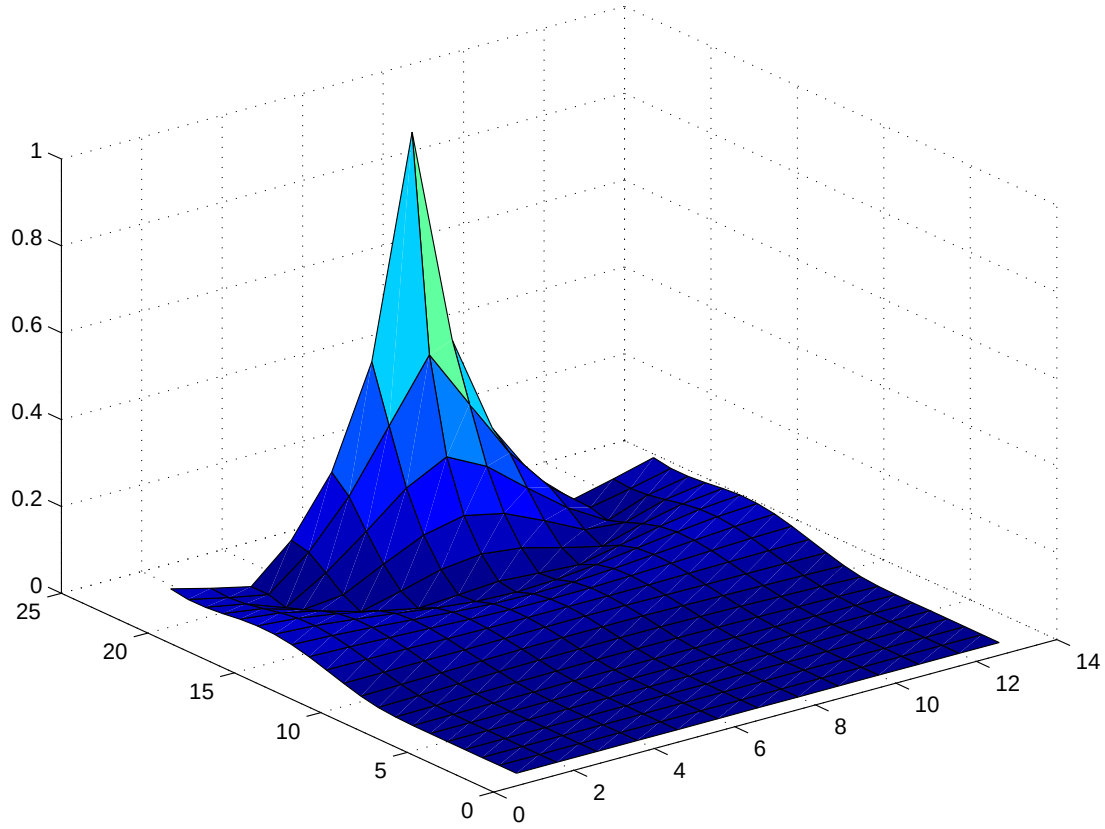
Information after 1 iteration

Why preconditioners?



Information after 7 iterations

Why preconditioners?



Information after 21 iterations

Why preconditioning?

From the previous pictures it is clear that we need $O(1/h) = O(\sqrt{n})$ iterations to move information from one end to the other end of the grid.

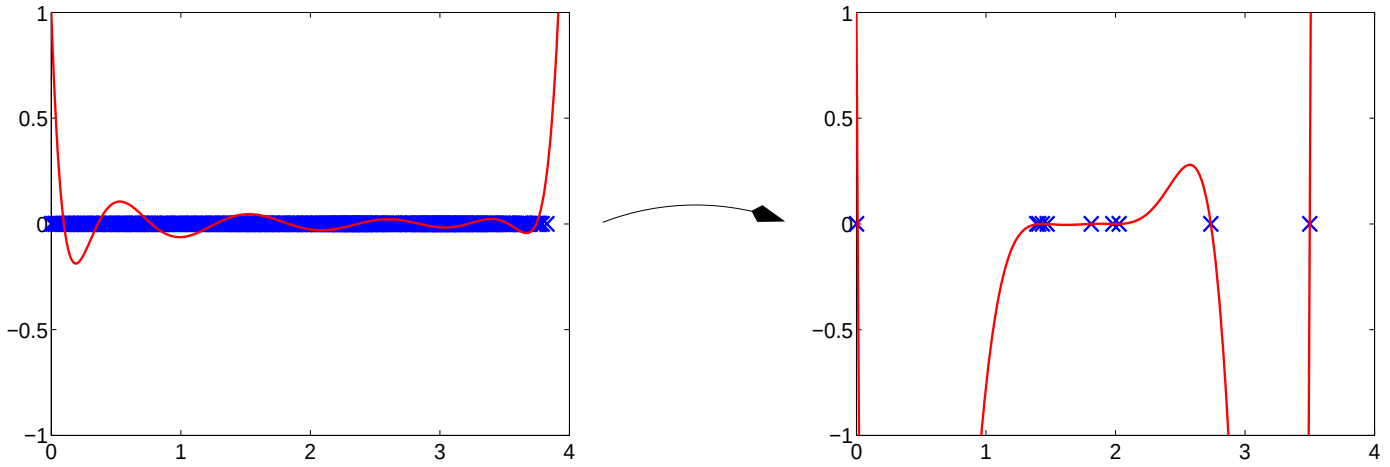
So *at best* it takes $O(n^{3/2})$ operations to compute the solution with an iterative method.

In order to improve this we need a preconditioner that enables fast propagation of information through the mesh.

Clustering the spectrum

In lesson 7 we saw that CG performs better when the spectrum of A is clustered.

Therefore, a good preconditioner clusters the spectrum.



Diagonal scaling

Diagonal scaling or Jacobi preconditioning uses

$$M = diag(A)$$

as preconditioner. Clearly, this preconditioner does not enable fast propagation through a grid. On the other hand, operations with $diag(A)$ are very easy to perform and diagonal scaling can be useful as a first step, in combination with other techniques.

Gauss-Seidel, SOR and SSOR

The Gauss-Seidel preconditioner is defined by

$$M = L + D$$

in which L is the strictly lower-triangular part of A and $D = diag(A)$. By introducing a relaxation parameter, we get the SOR-preconditioner.

For symmetric problems it is wise to take a symmetric preconditioner. A symmetric variant of Gauss-Seidel is defined by

$$M = (L + D)D^{-1}(L + D)^T$$

By introducing a relaxation parameter we get the so called SSOR-preconditioner.

ILU-preconditioners (1)

ILU-preconditioners are the most popular 'black box' preconditioners. They are constructed by making a standard LU-decomposition

$$A = LU \ .$$

However, during the elimination process some nonzero entries in the factors are discarded. This can be done on basis of two criteria:

- Sparsity pattern: e.g. an entry in a factor is only kept if it corresponds to a nonzero entry in A;
- Size: small entries in the decomposition are dropped.

ILU-preconditioners (2)

The number of nonzero entries that is maintained in the LU-factors is normally of the order of the number of nonzeros in A.
This means that operations with the ILU-preconditioner are approximately as costly as multiplications with A.

For A Symmetric Positive Definite a special variant of ILU exists, called Incomplete Choleski. This preconditioner is based on the Choleski decomposition $A = CC^T$.

Numerical software

To promote the use of good quality software several efforts have been made in the past. We mention:

- Eispack: Fortran 66 package for eigenvalue computations.
- Linpack: F77 package for dense or banded linear systems.
- BLAS: standardisation of basic linear algebra operations. Available in several languages.
- LAPACK: F77 package for dense eigenvalue problems and linear systems. Builds on BLAS and has replaced Eispack and Linpack. F90 and C++ versions also exist.

The above software can be downloaded from http://www.netlib.org.

BLAS

The BLAS libraries provide a standard for basic linear algebra subroutines. The BLAS library is available in optimised form on many (super)computers. Three versions are available:

- BLAS 1: vector-vector operations;
- BLAS 2: matrix-vector operations;
- BLAS 3: matrix-matrix operations;

BLAS 1

Examples of BLAS 1 routines are:

- Vector update: `daxpy`
- Inner product: `ddot`
- Vector scaling:`dscal`

Cache memory

Computers normally have small, fast memory close to the processor, the so called cache memory.

For optimal performance data in the cache should be reused as much as possible.

Level 1 BLAS routines are for this reason not very efficient: for every number that is loaded into the cache only one calculation is made.

BLAS 2 and 3

An example of a BLAS 2 routine is the matrix-vector multiplication routine `dgemv`.
An example of a BLAS 3 routine is the matrix-matrix multiplication routine `dgemm`.

BLAS 2 and in particular BLAS 3 routines make much better use of the cache than BLAS 1 routines.

Paralllel computing

Modern supercomputers may contain many thousands of processors. Another popular type of parallel computer is the cluster of workstations connected via a communication network.

Paralllel computing poses special restrictions on the numerical algorithms. In particular, algorithms that rely on recursions are difficult to parallelise.

Shared versus distributed memory

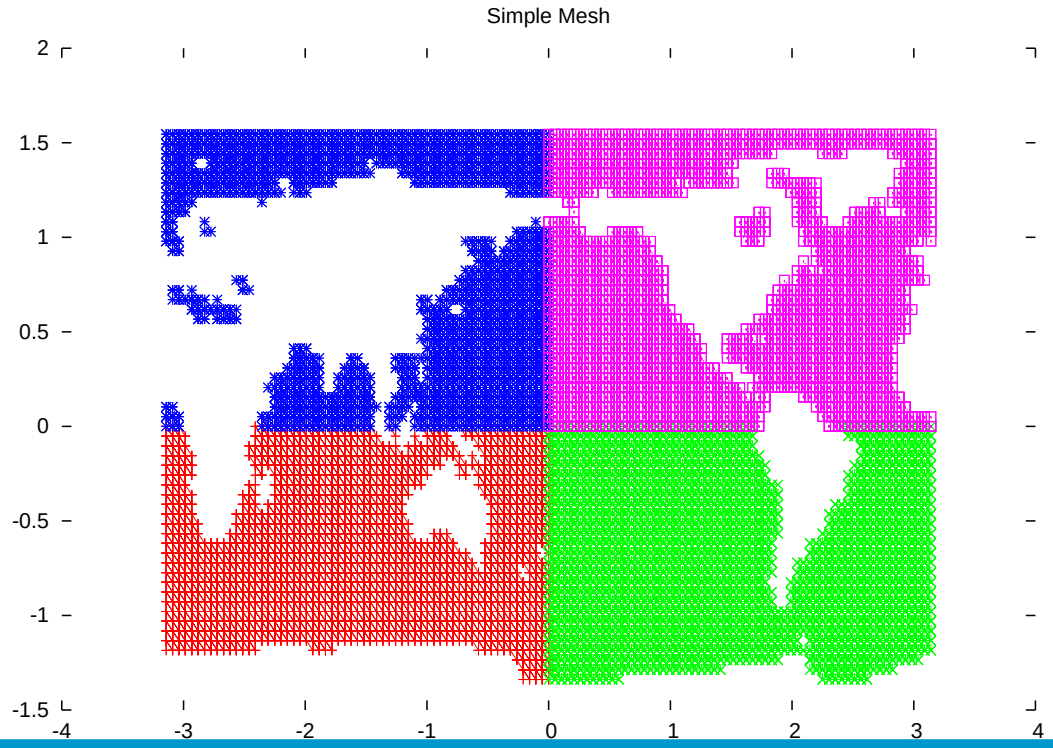
An important distinction in computer architecture is the memory organisation.

Shared memory machines have a single address space: all processors read from and write to the same memory. This type of machines can be parallelised using fine grain, loop level parallelisation techniques.

On distributed memory computers every processors has its own local memory. Data is exchanged via a message passing mechanism. Parallelisation should be done using a coarse grain approach. This is often achieved by making a domain decomposition.

Domain decomposition

A standard way to parallelise a grid-based computation is to split the domain into p subdomains, and to map each subdomain on a processor.



Domain decomposition (2)

The domain decomposition decomposes the system $Ax = b$ into blocks. For two subdomains one obtains

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix},$$

x_1 and x_2 represent the subdomain unknowns, A_{11} and A_{22} the subdomain discretization matrices and A_{12} and A_{21} the coupling matrices between subdomains.

Parallel matrix-vector multiplication

For the multiplication $u = Av$ the operations $u_1 = A_{11}v_1 + A_{12}v_2$ and $u_2 = A_{22}v_2 + A_{21}v_1$ can be performed in parallel.

However, processor 1 has to send (part of) v_1 to processor 2 before the computation, and processor 2 (part of) v_2 to processor 1.

This kind of communication is **local**, only informations from neighbouring subdomains is needed.

Parallel vector operations

Inner products $u^H v$ are calculated by computing the local inner products $u_1^H v_1$ and $u_2^H v_2$. The final results is obtained by adding all local inner products. This requires **global** communication.

Vector updates $u = u + av$ can be performed locally, without any communication.

Domain decomposition preconditioners

It is a natural idea to solve a linear system $Ax = b$ by solving the subdomain problems independently and to iterate to correct for the error.

This idea has given rise to the family of domain decomposition preconditioners.
The theory for domain decomposition preconditioners is vast, here we only discuss some important ideas.

Block-Jacobi preconditioner

A simple domain decomposition preconditioner is defined by

$$M = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix}$$ (Block-Jacobi preconditioner).

or, in general by

$$M = \begin{pmatrix} A_{11} & & \\ & \ddots & \\ & & A_{MM} \end{pmatrix}$$

The subdomain systems can be solved in parallel.

Solution of the subdomain problems

There are several ways to solve the subdomain problems:

- Exact solves. This is in general (too) expensive.
- Inexact solves using an incomplete decomposition (block-ILU).
- Inexact solves using an iterative method to solve the subproblems. Since in this case the preconditioner is variable, the outer iteration should be flexible, for example GCR.

On the scalability of block-Jacobi

Without special techniques, the number of iterations increases with the number of subdomains. The algorithm is not scalable.

To overcome this problems techniques can be applied to enable the exchange of information between subdomains.

Improving the scalability

Two popular techniques improve the flow of information are:

- Use an overlap between subdomains. Of course one has to ensure that the value of unknowns in gridpoints that belong to multiple domains is unique.
- Use a coarse grid correction. The solution of the coarse grid problem is added to the subdomain solutions.
  This idea is closely related to multigrid, since the coarse-grid solution is the non-local, smooth part of the solution that cannot be represented on a single subdomain.

Concluding remarks

Preconditioners are the key to a successful iterative method.

Today we saw some of the most important preconditioners. The 'best' preconditioner, however, depends completely on the problem.