Iterative Methods for Linear Systems of Equations

Preconditioning techniques

ITMAN PhD-course, DTU, 20-10-08 till 24-10-08

Martin van Gijzen

Overview day 5

- Preconditioning
- Some popular preconditioners
  - Diagonal scaling (Jacobi)
  - Gauss-Seidel, SOR and SSOR
  - Incomplete Choleski and Incomplete LU
- Advanced preconditioners
  - Multigrid
  - Domain decomposition
- Preconditioning of KKT systems

Introduction

Already on the first day we saw that Richardson’s method could 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.

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

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

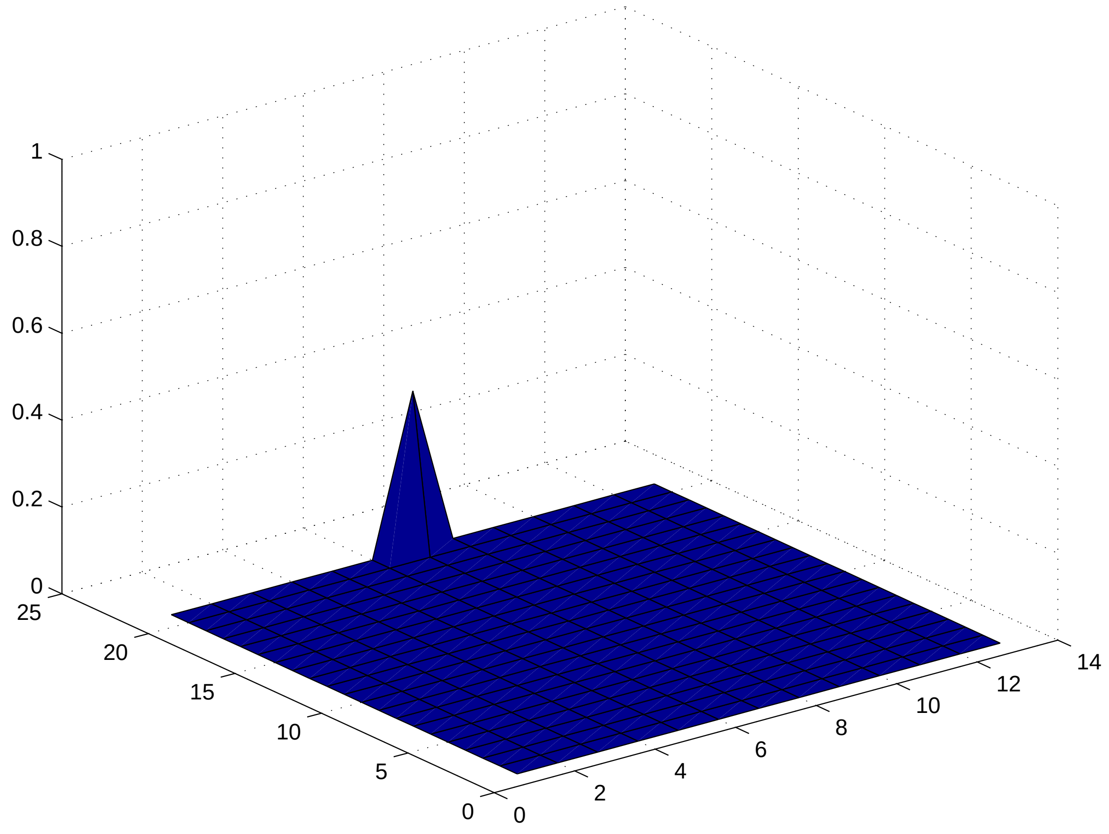
Preconditioning (4)

Left, right and central preconditioning gives the same spectrum.

Yet there are other differences:

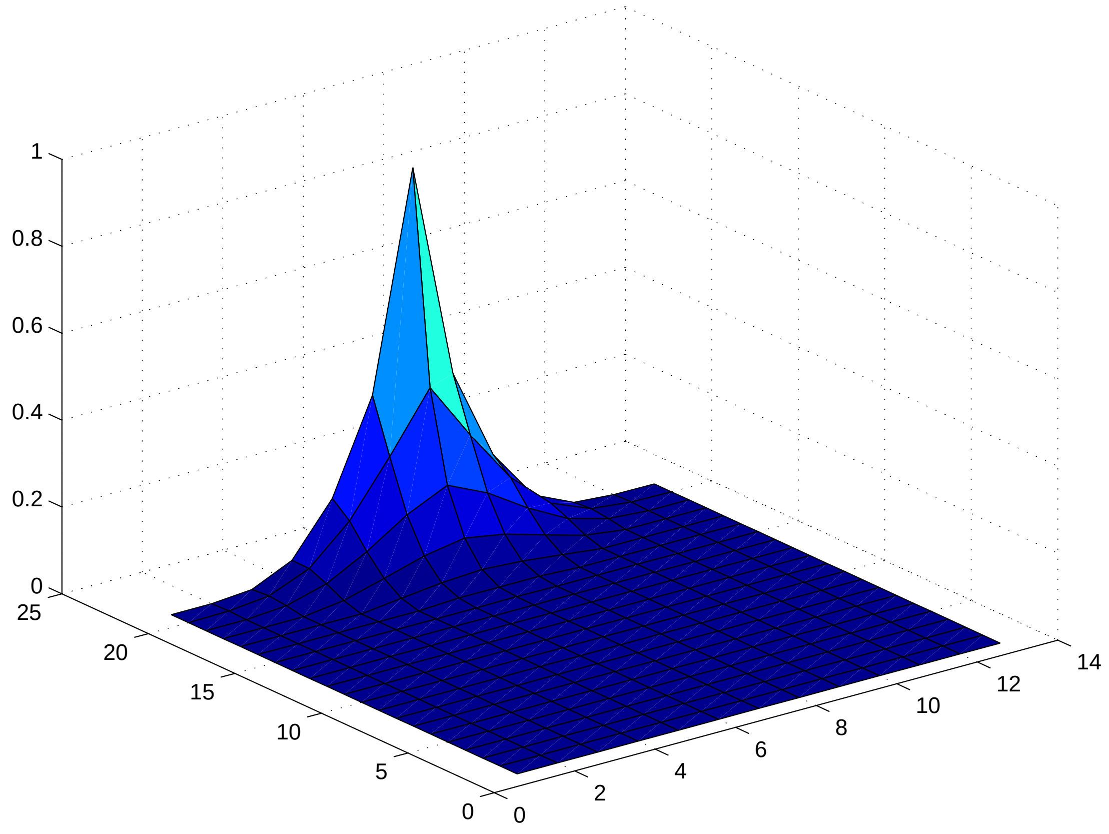
- Left preconditioning is most natural: no extra step is required to compute x;
- Central preconditioning preserves symmetry;
- Right preconditioning does not affect the residual norm.

Why preconditioners?



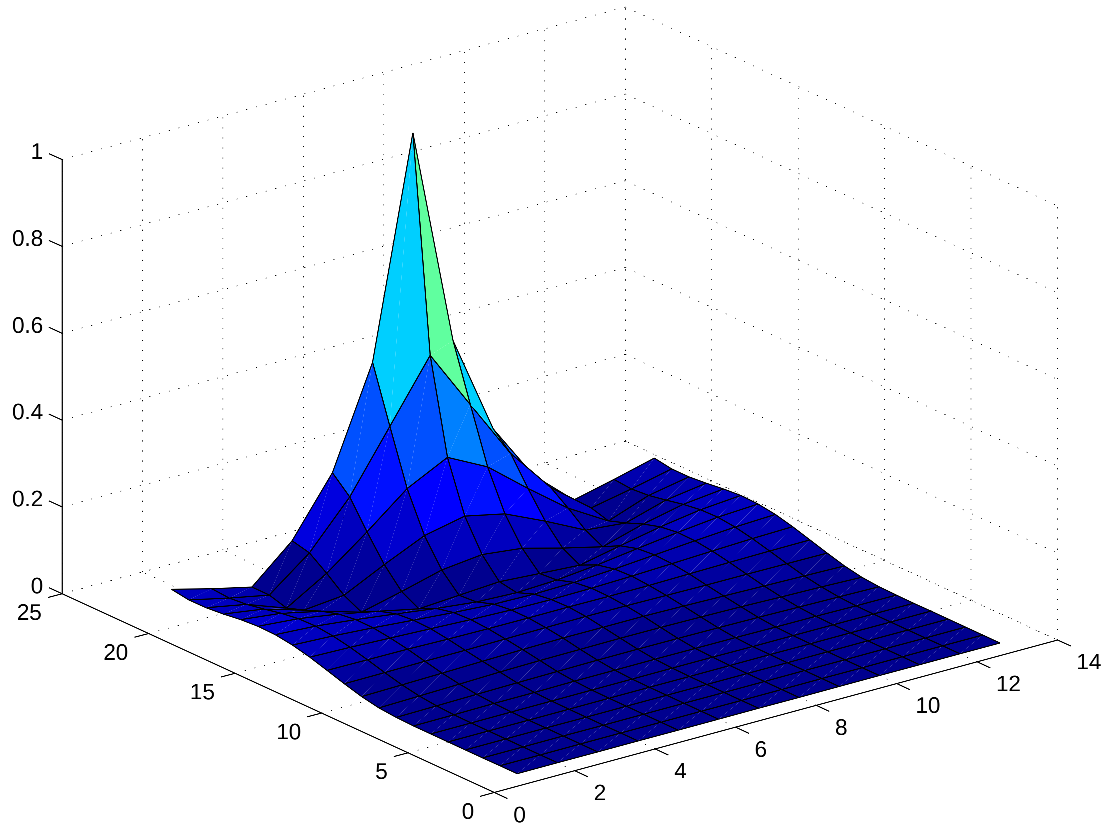
Information after 1 iteration

Why preconditioners?



Information after 7 iterations

Why preconditioners?



Information after 21 iterations

TU Delft

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.

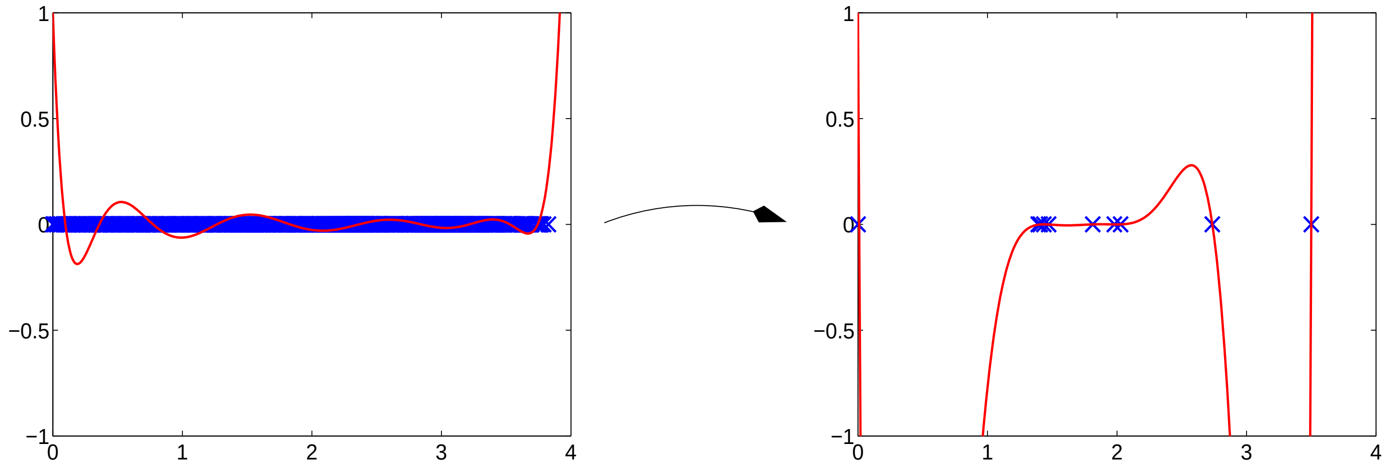
So *at best* it takes $O(n^{3/2})$ operations to compute the solution with an iterative method.

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

Clustering the spectrum

On day 2 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 LU-preconditioner are approximately as costly as multiplications with A.

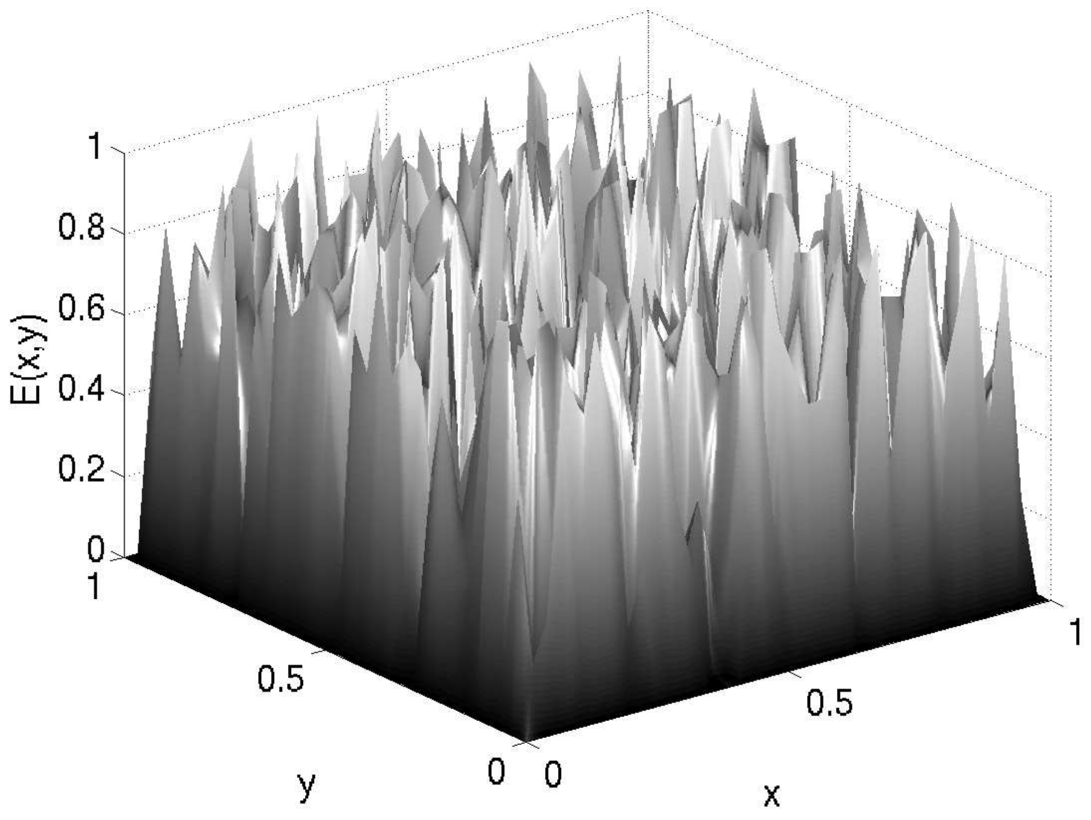
For A Symmetric Positive Definite a special variant of ILU exists, called Incomplete Choleski. This preconditioner is based on the Choleski decomposition $A = CC^T$.

Basic preconditioners

Although the preconditioners discussed before can considerably reduce the number of iterations, they do not normally reduce the mesh-dependency of the number of iterations.

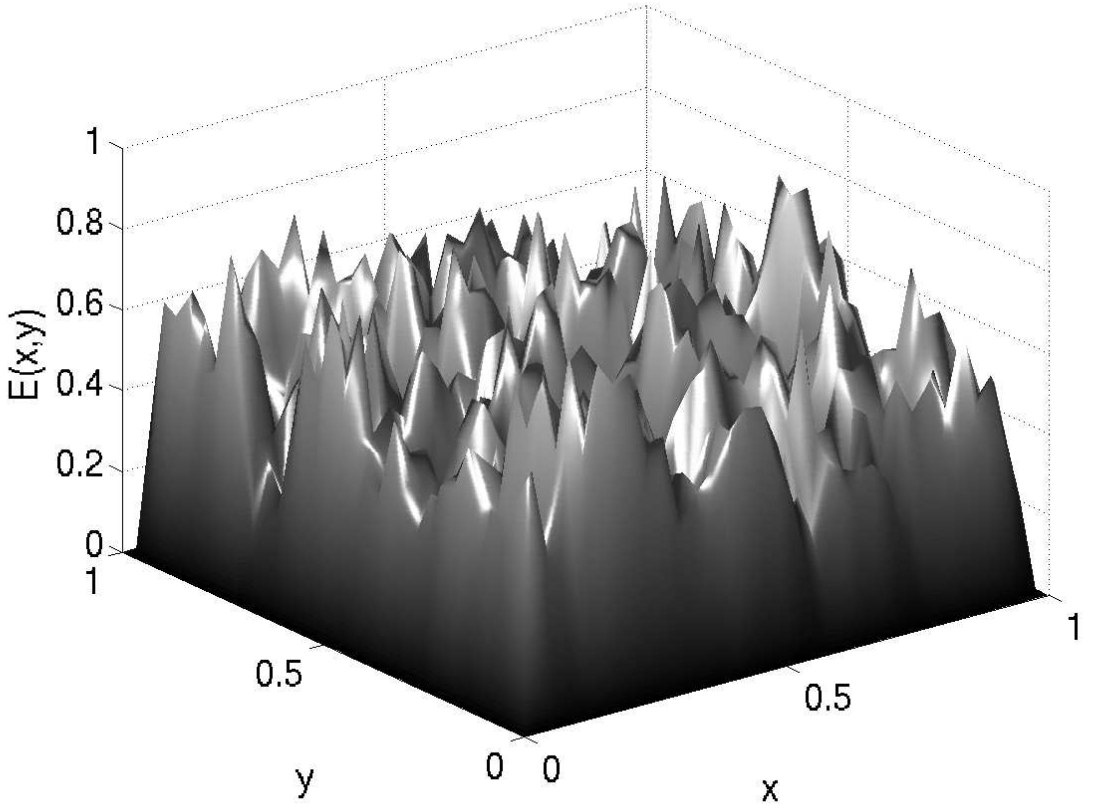
In the next slides we take a closer look at how basic iterative methods reduce the error. From the observations we make, we will develop the idea that is at the basis of one of the fastest techniques: multigrid.

Smoothing Property



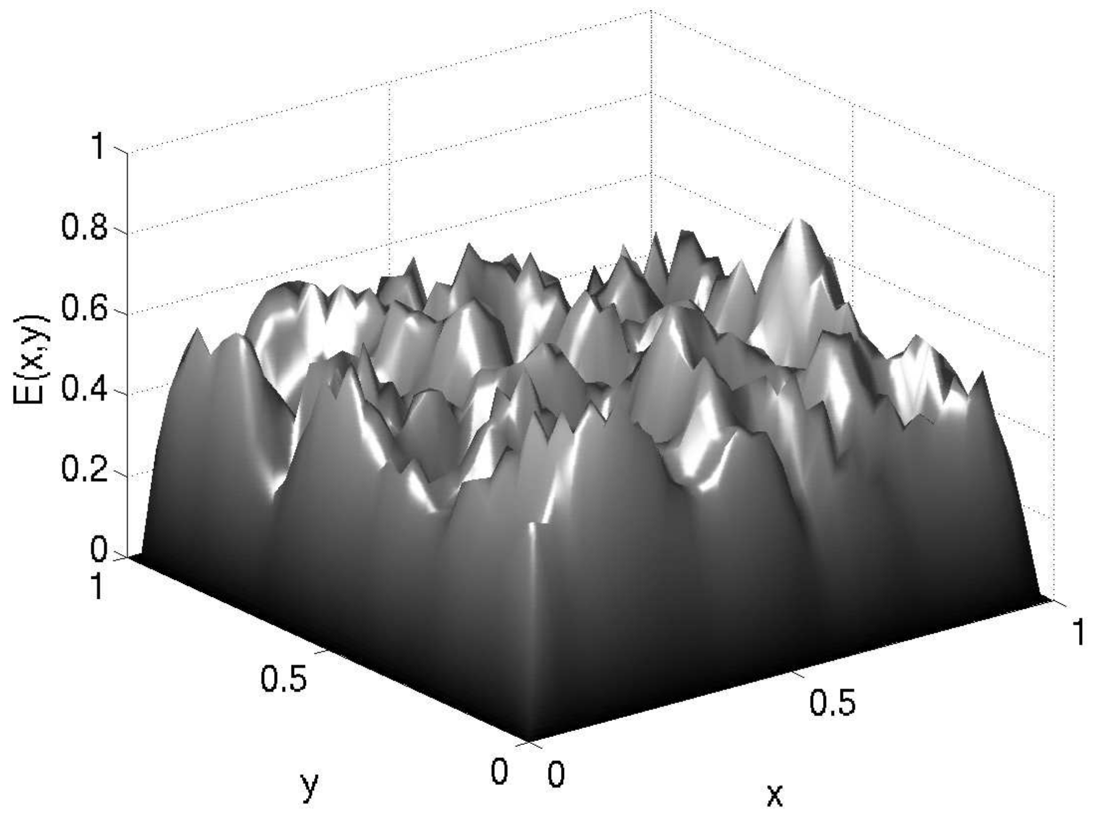
Random initial error

Smoothing Property



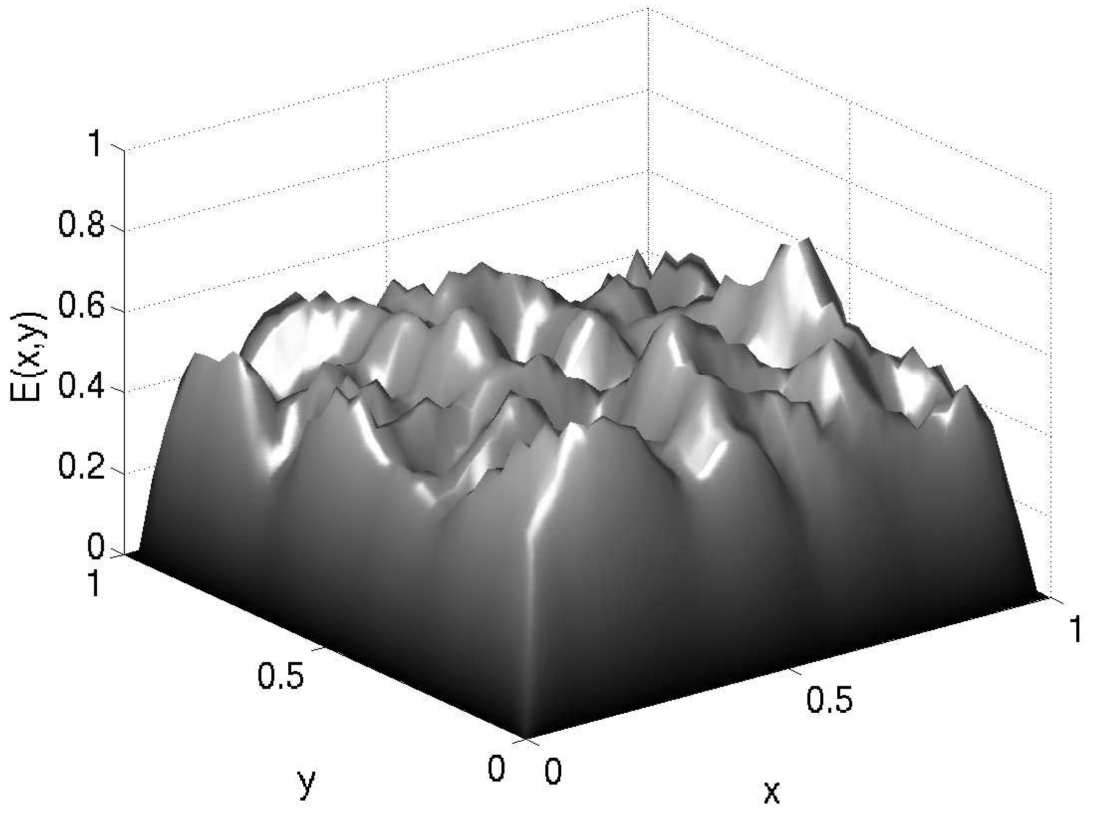
Error after 1 Jacobi iterations

Smoothing Property



Error after 2 Jacobi iterations

Smoothing Property



Error after 3 Jacobi iterations

Complementarity

- Error after a few weighted Jacobi iterations has structure, this is the same for the other basic iterative methods.
- Instead of discarding the method, look to complement its failings

How can we best correct errors that are slowly reduced by basic iterative method?

Complementarity

- Error after a few weighted Jacobi iterations has structure, this is the same for the other basic iterative methods.
- Instead of discarding the method, look to complement its failings

How can we best correct errors that are slowly reduced by basic iterative method?

- Slow-to-converge errors are smooth
- Smooth vectors can be accurately represented using fewer degrees of freedom

Coarse-Grid Correction

- Smooth vectors can be accurately represented using fewer degrees of freedom
- Idea: transfer job of resolving smooth components to a coarser grid version of the problem
- Need:
  - Complementary process for resolving smooth components of the error on the coarse grid
  - Way to combine the results of the two processes

Multigrid

- Relaxation is the name for applying one or a few basic iteration steps.
- Idea is to correct the approximation after relaxation, $x^{(1)}$, from a coarse-grid version of the problem
- Need interpolation map, P, from coarse grid to fine grid
- Corrected approximation will be $x^{(2)} = x^{(1)} + Px_c$
- x_c is the solution of the coarse-grid problem and satisfies $(P^TAP)x_c = P^TA(x - x^{(1)}) = P^Tr^{(1)}$

Two-grid cycle

Two-grid cycle

Multigrid Components

- Relaxation

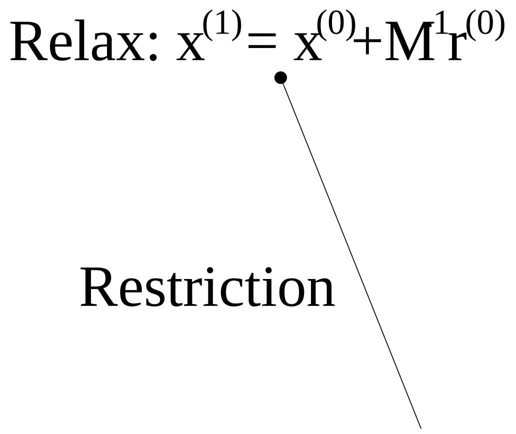
Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$

- Use a smoothing process (such as Jacobi or Gauss-Seidel) to eliminate oscillatory errors
- Remaining error satisfies $Ae^{(1)} = r^{(1)} = b - Ax^{(1)}$

Two-grid cycle

Multigrid Components

- Relaxation
- Restriction

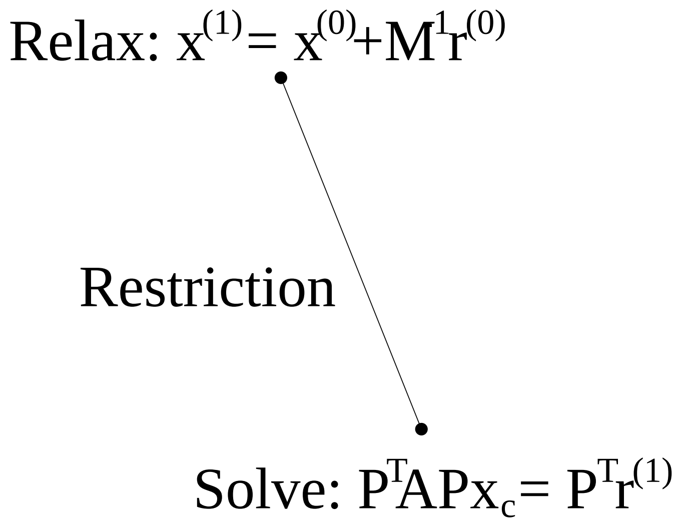


- Transfer residual to coarse grid
- Compute $P^T r^{(1)}$

Two-grid cycle

Multigrid Components

- Relaxation
- Restriction
- Coarse-Grid Correction


Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
Restriction
Solve: $P^T A P x_c = P^T r^{(1)}$

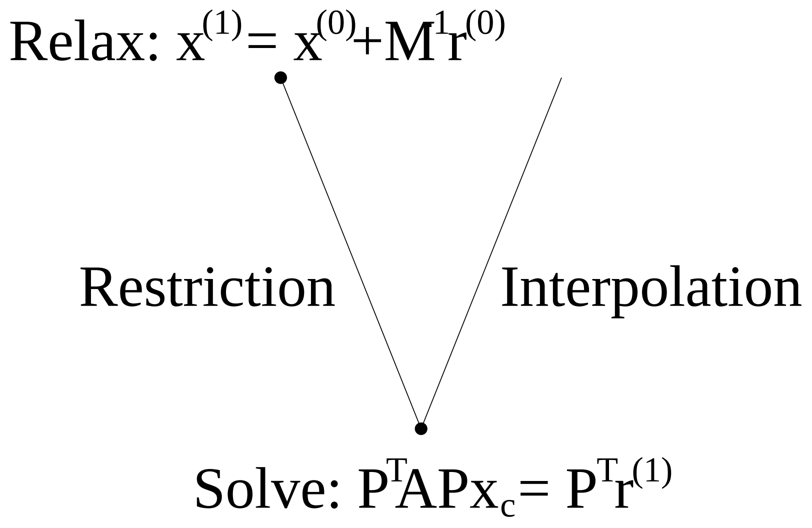
- Use coarse-grid correction to eliminate smooth errors
- Best correction x_c satisfies

$$P^T A P x_c = P^T r^{(1)}$$

Two-grid cycle

Multigrid Components

- Relaxation
- Restriction
- Coarse-Grid Correction
- Interpolation


Relax: $x^{(1)} = x^{(0)} + M^{-1}r^{(0)}$
Restriction
Interpolation
Solve: $P^TAPx_c = P^Tr^{(1)}$

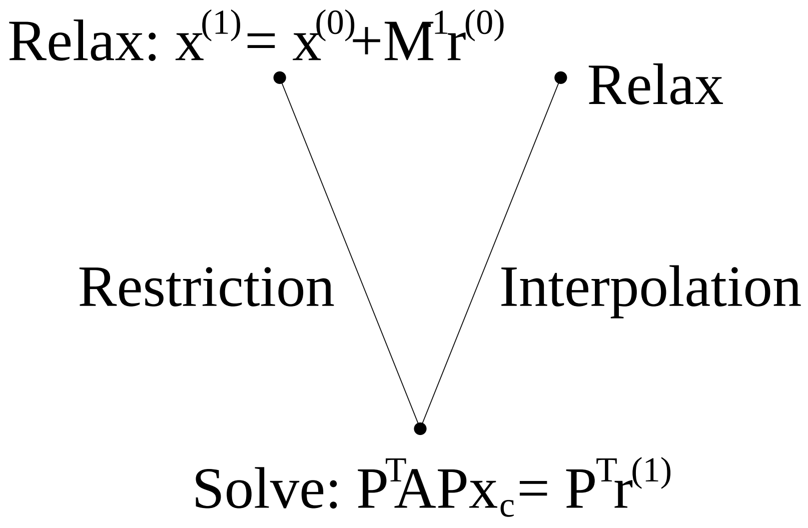
- Transfer correction to fine grid
- Compute $x^{(2)} = x^{(1)} + Px_c$

TU Delft

Two-grid cycle

Multigrid Components

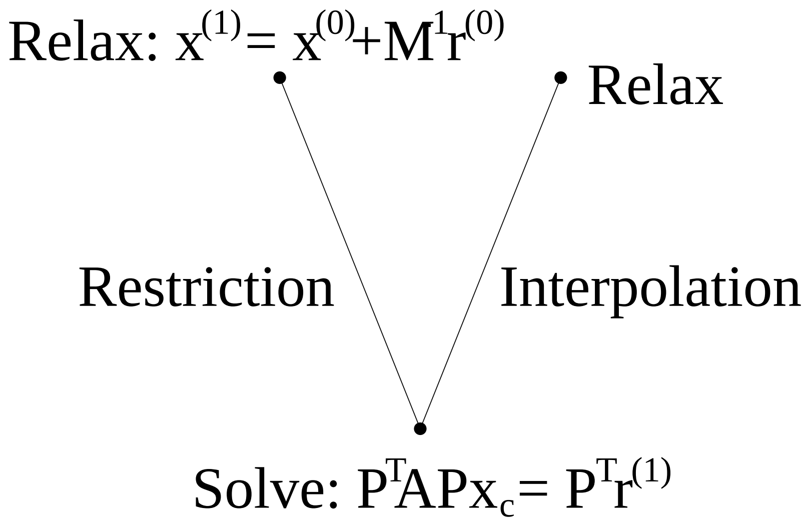
- Relaxation
- Restriction
- Coarse-Grid Correction
- Interpolation
- Relaxation
- Relax once again to remove oscillatory error introduced in coarse-grid correction


Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
Relax
Restriction
Interpolation
Solve: $P^T A P x_c = P^T r^{(1)}$

Two-grid cycle

Multigrid Components

- Relaxation
- Restriction
- Coarse-Grid Correction
- Interpolation
- Relaxation

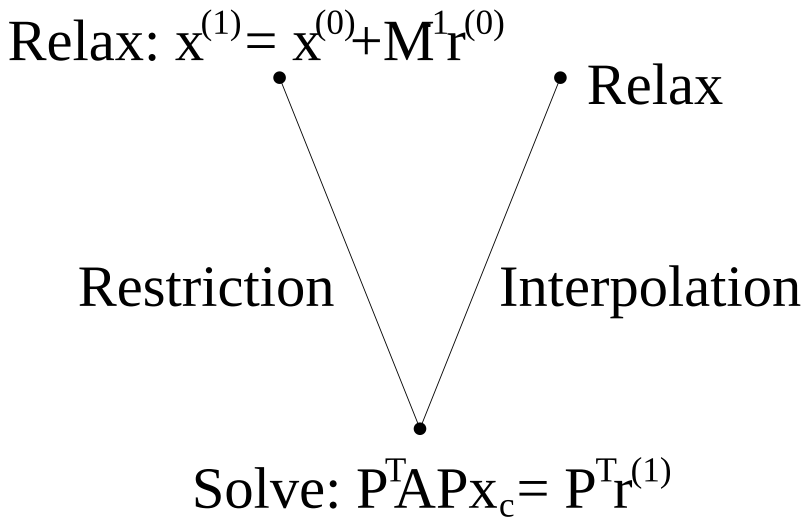

Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
Relax
Restriction
Interpolation
Solve: $P^T A P x_c = P^T r^{(1)}$

Direct solution of coarse-grid problem isn't practical

Two-grid cycle

Multigrid Components

- Relaxation
- Restriction
- Coarse-Grid Correction
- Interpolation
- Relaxation


Relax: $x^{(1)} = x^{(0)} + M^{-1} r^{(0)}$
Relax
Restriction
Interpolation
Solve: $P^T A P x_c = P^T r^{(1)}$

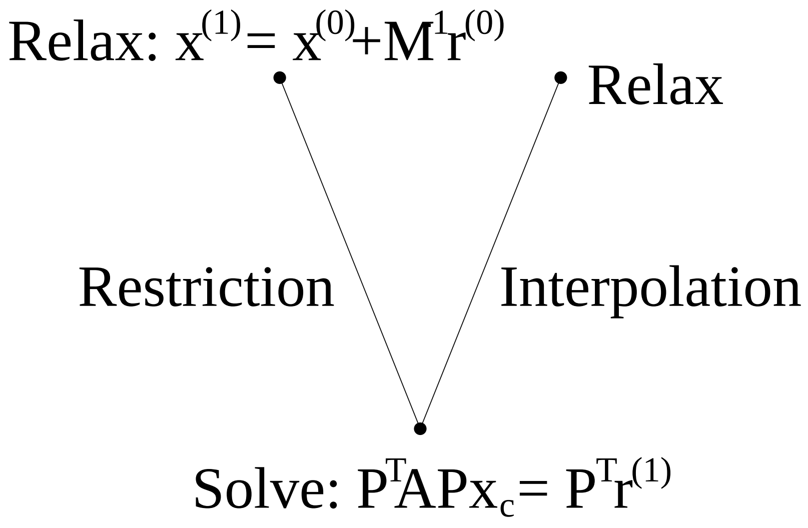
Direct solution of coarse-grid problem isn't practical

Use an iterative method!

Two-grid cycle

Multigrid Components

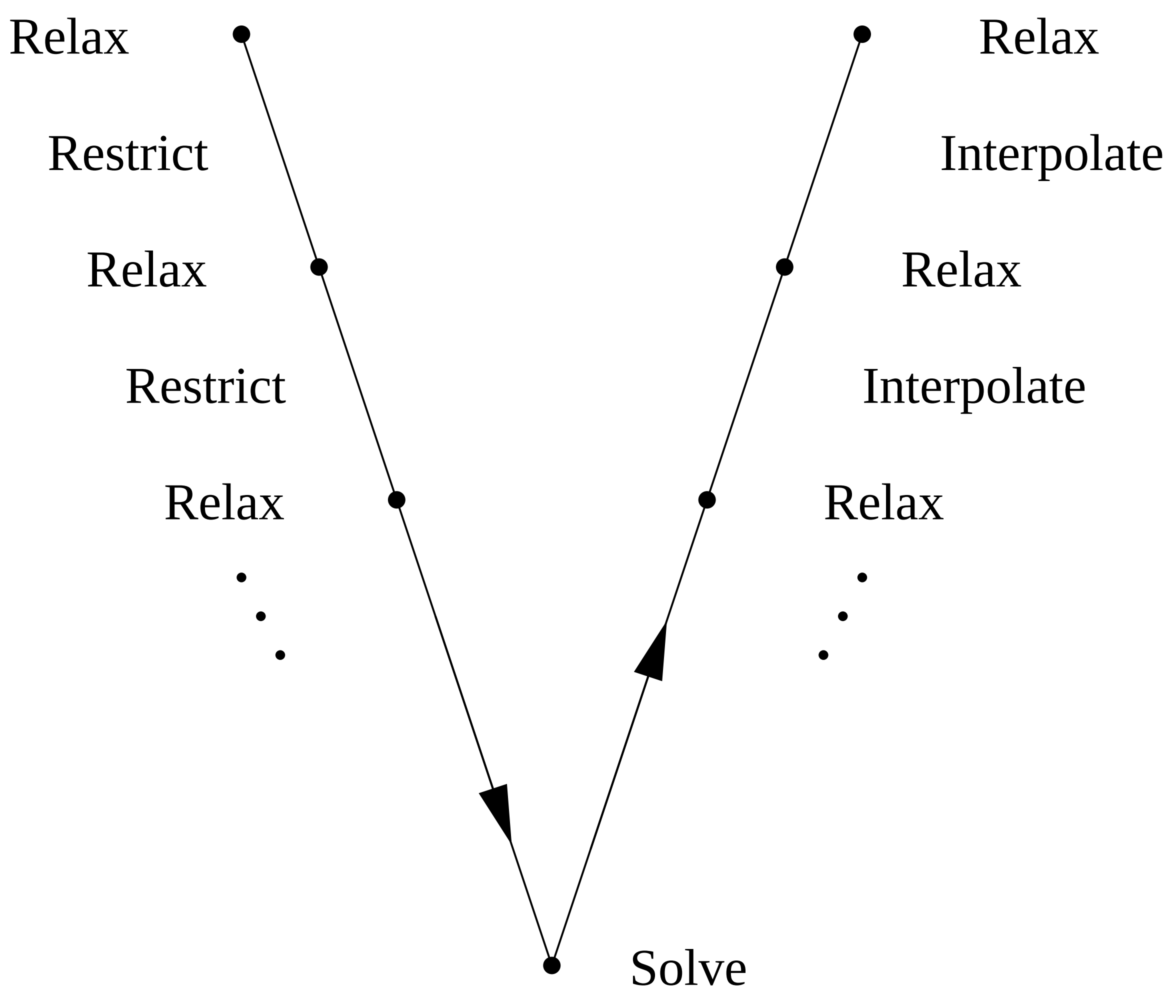
- Relaxation
- Restriction
- Coarse-Grid Correction
- Interpolation
- Relaxation


Relax: $x^{(1)} = x^{(0)} + M^{-1}r^{(0)}$
Relax
Restriction
Interpolation
Solve: $P^TAPx_c = P^Tr^{(1)}$

Recursion!

Apply same methodology to solve coarse-grid problem

The Multigrid V-cycle

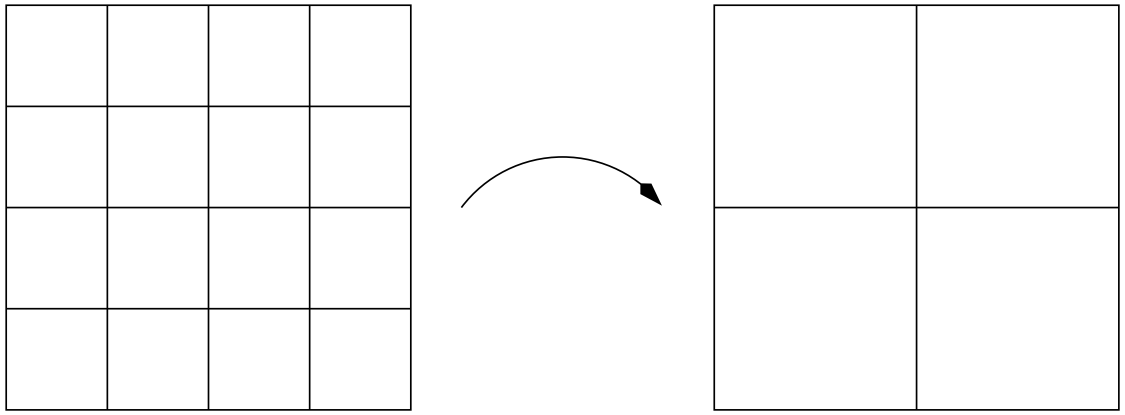


Properties of Effective Cycles

- Fast convergence
  - Effective reduction of all error components
  - On each level, coarse-grid correction must effectively reduce exactly those errors that are slow to be reduced by relaxation alone
  - Hierarchy of coarse-grid operators resolves relevant physics at each scale
- Low iteration cost
  - Simple relaxation scheme (cheap computation of $M^{-1}r$ on all levels)
  - Sparse coarse-grid operators
  - Sparse interpolation/restriction operations

Choosing Coarse Grids

- No *best* strategy to choose coarse grids
- Operator dependent, but also machine dependent
- For structured meshes, often use uniform de-refinement approach



- For unstructured meshes, various weighted independent set algorithms are often used.

What didn’t we talk about?

- How do we choose P?
  - Number of columns
  - Sparsity structure
  - Non-zero values
- Choices depend closely on the properties of the relaxation method

Some remarks about multigrid

Multigrid works well if the problem

- is grid-based. However, matrix-based multigrid methods (Algebraic Multigrid) do exist and are often successful;
- has a smooth solution. An underlying assumption is that the solution can be represented on a coarser grid. Multigrid works particularly well for Poisson-type problems. For these problems the number of operations is $O(n)$.

Multigrid can be used as a separate solver, but is often used as a preconditioner for a Krylov-type method.

Parallel 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.

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

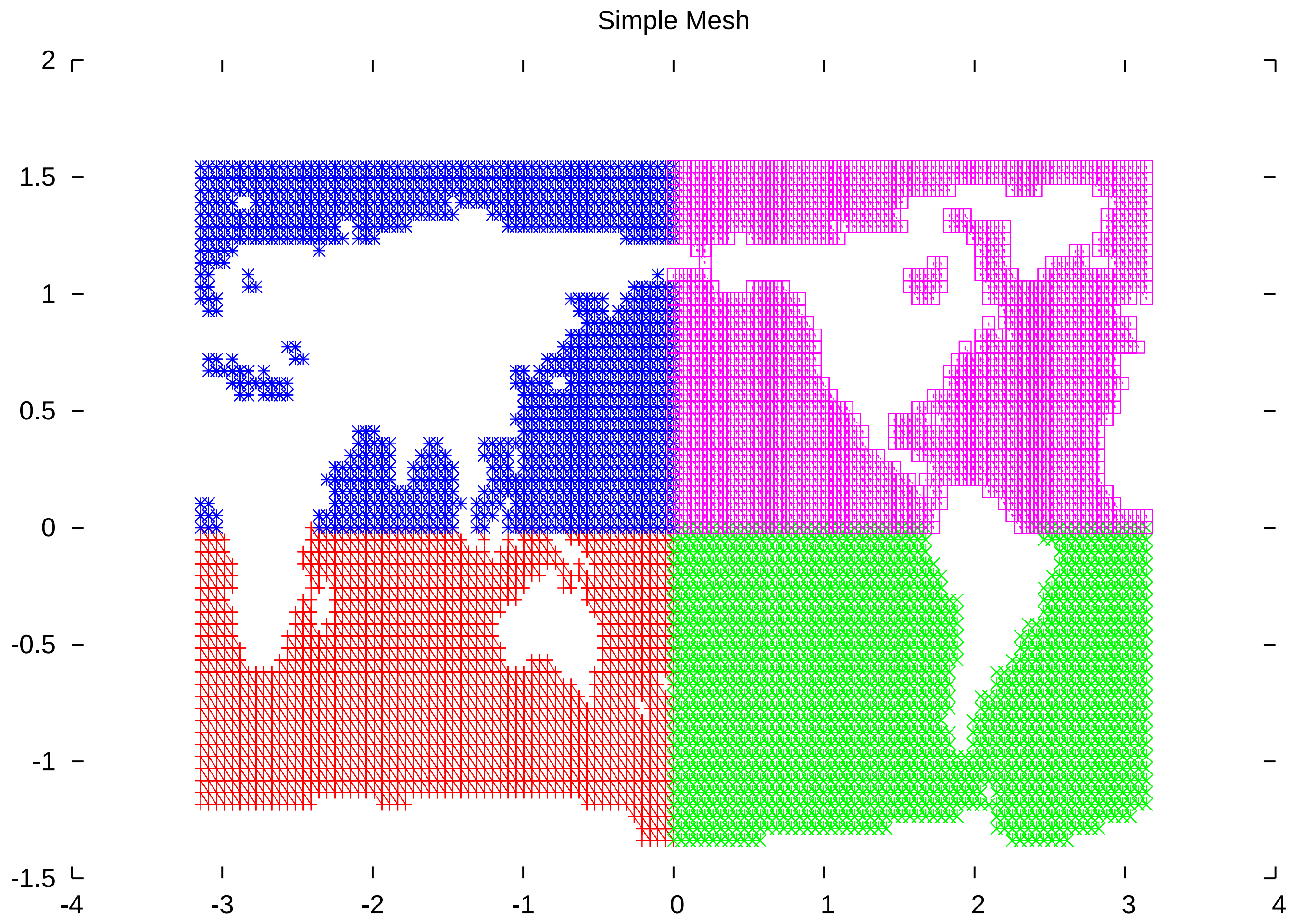
Domain decomposition preconditioners

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.

It is a natural idea to solve 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.

Domain decomposition



Example of domain decomposition

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.

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

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

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 f lexible, 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 problem, techniques can be applied to enable the exchange of information between subdomains.

Improving the scalability

Two popular techniques that 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 error that cannot be represented on a single subdomain.

KKT or saddle point systems

KKT systems frequently arise in optimisation. KKT systems are also know as saddle-point systems. A KKT system has the following block structure

$$A = \begin{pmatrix} F & B^T \\ B & -C \end{pmatrix}$$

with F and C symmetric pos. def matrices.

Systems with such a matrix arise in many other applications, for example in CFD (Stokes problem) and in structural engineering.

Preconditioning a KKT-system (1)

Many preconditonres have been developed that make use of the fact that of the system matrix

$$A = \begin{pmatrix} F & B^T \\ B & -C \end{pmatrix}$$

a block LU decomposition can be made:

$$\begin{pmatrix} F & B^T \\ B & -C \end{pmatrix} = \begin{pmatrix} I & O^T \\ BF^{-1} & I \end{pmatrix} \begin{pmatrix} F & B^T \\ O & -M_S \end{pmatrix} .$$

Here $M_S = BF^{-1}B^T + C$ is the Schur complement.

Preconditioning a KKT-system (2)

Idea (e.g. Elman, Silvester, Wathen): take

$$P = \begin{pmatrix} F & B^T \\ O & -M_S \end{pmatrix}$$

as (right) preconditioner:

$$AP^{-1} = \begin{pmatrix} I & O^T \\ BF^{-1} & I \end{pmatrix}$$

has only eigenvalue 1: GMRES ready in 2 iterations. **BUT**

- Preconditioner is nonsymmetric
- Schur complement is too expensive to compute and has to be approximated

Preconditioning a KKT-system (3)

An SPD block-diagonal preconditioner:

$$P = \begin{pmatrix} F & O^T \\ O & M_S \end{pmatrix}$$

Can be used with MINRES (short recurrences)

Preconditioned matrix has three distinct eigenvalues

$\rightarrow$ MINRES needs three iterations.

The main question is again how to make a cheap approximation to the Schur complement.

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.