step-2

Step-2 shows:

- How degrees of freedom are defined with finite elements
- The *DoFHandler* class
- How DoFs are connected by bilinear forms
- Sparsity patterns of matrices
- How to visualize a sparsity pattern

step-2

Sparsity of system matrices:

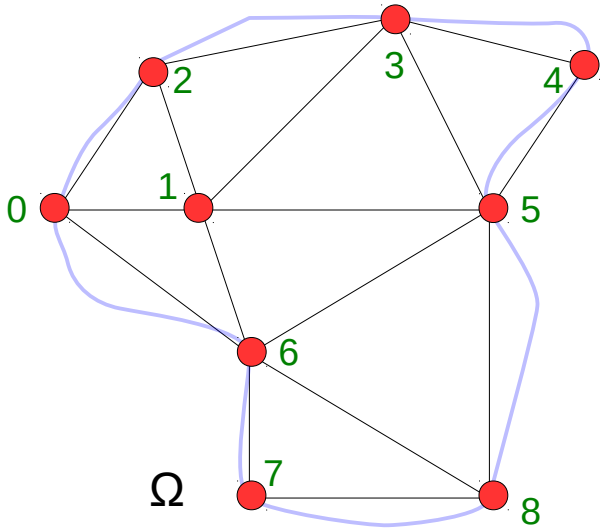
- For PDEs, finite element matrices are *always sparse*
- Result of
  – *local* definition of shape functions
  – *locality* of the differential operator

Sparsity is not a coincidence. It is a design choice of the finite element method.

Sparsity can not be overestimated as a factor in the success of the FEM!

step-2

Example: Consider this mesh and bilinear form:



$$A_{ij} = (\nabla\varphi_i, \nabla\varphi_j) = \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j \,\mathrm{d}x$$

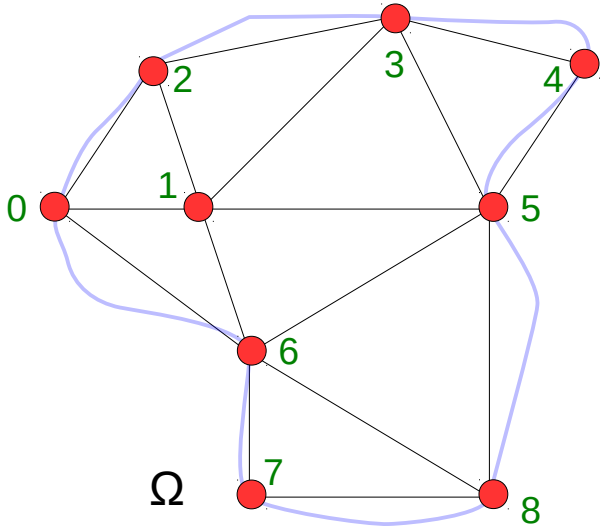
Note: In general we have that

- $A_{00}\neq 0, A_{01}\neq 0, A_{02}\neq 0, A_{06}\neq 0$
- $A_{03}=A_{04}=A_{05}=A_{07}=A_{08}=0$

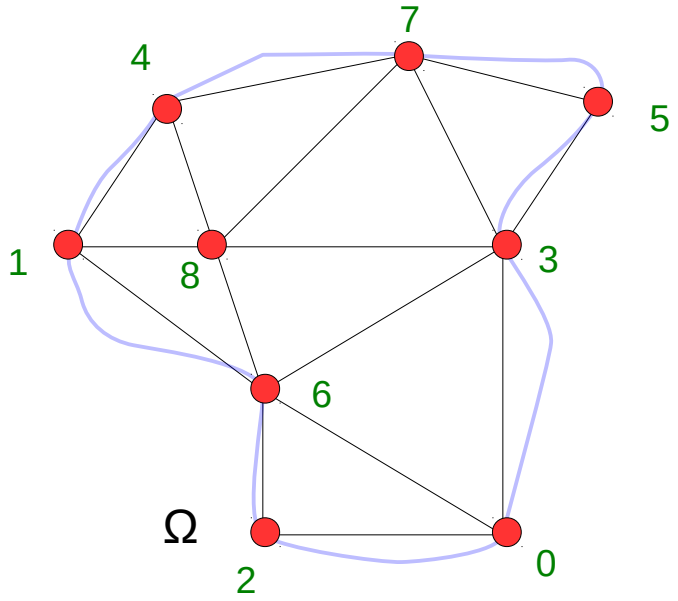
The bigger the mesh, the more zeros there are per row!

step-2

Renumbering: The order of enumerating degrees of freedom is arbitrary



vs.



Notes:

- Resulting matrices are just permutations of each other
- Both sparse; most algorithms don’t care about ordering

Gaussian elimination

End result: finite element matrices are sparse. So how do we solve linear systems AU=F ?

Recall:

- Gaussian elimination adds one row to the ones below
- The rows below therefore gain nonzero entries
- What started out as a matrix with cN nonzero entries will result in something with $O(N^2)$ entries!
- In other words, the result is a *full* matrix!

Gaussian elimination

Gaussian elimination leads to a full matrix.

Why is this bad:

- We need to store $O(N^2)$ entries instead of cN entries
- Computing $O(N^2)$ entries generally costs $O(N^3)$ ops

Imagine cases where $N=1,000,000$.

Remark: There are “sparse direct solvers” that try to be smarter. But the basic problem remains.

Iterative solvers

Idea: Do not try to actually factor/invert the matrix *A*.

Instead: Use methods that

- solve $AU=F$ by starting with a guess for U and *iteratively* improving on it
- only ever *multiply* by A

Benefits:

- $O(N)$ memory consumption
- $O(N)$ cost for each iteration
- $O(N)$ overall cost if we could get away with a fixed number of iterations

Iterative solvers

Idea: Do not try to actually factor/invert the matrix *A*.

Instead: Use methods that

- solve $AU=F$ by starting with a guess for U and *iteratively* improving on it
- only ever *multiply* by A

Examples of methods:

- Fixed point iterations: Richardson, Gauss-Seidel, SOR, ...
- Krylov subspace methods: Conjugate Gradients (CG), GMRES, BiCGStab, ...

Direct vs iterative

Guidelines for direct solvers vs iterative solvers:

Direct solvers:

- ✔ *Always* work, for any invertible matrix
- ✔ No need to think about preconditioners
- ✔ Faster for problems with <100k unknowns
- ✘ Need too much memory + CPU time for larger problems

Iterative solvers:

- ✔ Need *O(N)* memory
- ✔ Can solve *very* large problems
- ✔ Often parallelize well
- ✘ Choice of solver/preconditioner depends on problem

Advice for iterative solvers

There is a wide variety of iterative solvers:

- CG: Conjugate gradients
- MinRes: Minimal residuals
- GMRES: Generalized minimal residuals
- F-GMRES: Flexible GMRES
- SymmLQ: Symmetric LQ decomposition
- BiCGStab: Biconjugate gradients stabilized
- QMR: Quasi-minimal residual
- TF-QMR: Transpose-free QMR
- ...

Which solver to choose depends on the properties of the matrix, primarily *symmetry* and *definiteness*!

Advice for iterative solvers

Guidelines for use:

- CG: Matrix is symmetric, positive definite
- MinRes: –
- GMRES: Catch-all
- F-GMRES: Catch-all with variable preconditioners
- SymmLQ: –
- BiCGStab: Matrix is non-symmetric but positive definite
- QMR: –
- TF-QMR: –
- All others: –

In reality, only CG, BiCGStab and (F-)GMRES are used much.

Advice for iterative solvers

Note:

All iterative solvers are bad without a good preconditioner!

The art of devising a good iterative solver is to devise a good preconditioner!

Finite element methods in scientific computing

Wolfgang Bangerth, Colorado State University

Lecture 35:

What preconditioner to use

Introduction

Parts 1+2: Simple preconditioners for simple problems

Observations on iterative solvers

The finite element method provides us with a linear system

$$A\,x \;=\; b$$

that we then need to solve.

Basic observations:

- For sparse direct solvers, speed of solution only depends on sparsity pattern
- For iterative solvers, performance also depends on the *values* in *A*
- Performance measures:
  – number of iterations
  – cost of every iteration

Observations on iterative solvers

The finite element method provides us with a linear system

$$A\,x \;=\; b$$

that we then need to solve.

Factors affecting performance of iterative solvers:

- Symmetry of a matrix
- Whether *A* is definite
- Condition number of *A*
- How the eigenvalues of *A* are clustered

- Whether *A* is reducible/irreducible

Observations on iterative solvers

Example 1: Using CG to solve

$$A\,x \;=\; b$$

where A is SPD, each iteration reduces the residual by a factor of

$$r \;=\; \frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1} \;<\; 1$$

- For a tolerance ε we need $n \;=\; \frac{\log \epsilon}{\log r}$ iterations
- For the Laplace matrix, this is $r \;=\; \frac{c-h}{c+h} \;<\; 1$

$$n \;=\; O\left(\frac{\log \epsilon}{h}\right)$$

Observations on iterative solvers

Example 2: When solving

$$A\,x \;=\; b$$

where A has the form

$$A \;=\; \begin{pmatrix} a_{11} & 0 & 0 & \cdots \\ 0 & a_{22} & 0 & \cdots \\ 0 & 0 & a_{33} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

then every decent iterative solver converges in 1 iteration.

Note 1: This, even though condition number may be large
Note 2: This is true, in particular, if $A=I$.

The idea of preconditioners

Idea: When solving

$$A\,x \;=\; b$$

maybe we can find a matrix P^{-1} and instead solve

$$P^{-1}\,A\,x \;=\; P^{-1}\,b$$

Observation 1: If $P^{-1}A \approx D$ then solving should require fewer iterations

Corollary: The perfect preconditioner is a multiple of the inverse matrix, i.e., $P^{-1}=A^{-1}$.

The idea of preconditioners

Idea: When solving

$$A\,x \;=\; b$$

maybe we can find a matrix P^{-1} and instead solve

$$P^{-1}\,A\,x \;=\; P^{-1}\,b$$

Observation 2: Iterative solvers only need matrix-vector multiplications, no element-by-element access.

The idea of preconditioners

Idea: When solving

$$A\,x \;=\; b$$

maybe we can find a matrix P^{-1} and instead solve

$$P^{-1}\,A\,x \;=\; P^{-1}\,b$$

Observation 3: There is a tradeoff:
fewer iterations vs *cost of preconditioner.*

Corollary: Preconditioning only works if P^{-1} is cheap to compute and if P^{-1} is cheap to *apply* to a vector.

Example: $P^{-1}=A^{-1}$ does not qualify.

Constructing preconditioners

Part 1:
Constructing simple preconditioners for elliptic problems

(1952 – 1980s)

Constructing preconditioners

Remember: When solving the preconditioned system

$$P^{-1} A x = P^{-1} b$$

then the best preconditioner is $P^{-1}=A^{-1}$.

Problem: (i) We can't compute it efficiently. (ii) If we could, we would not need an iterative solver.

But: Maybe we can approximate $P^{-1}\approx A^{-1}$.

Idea 1: Do we know of other iterative solution techniques?

Idea 2: Use incomplete decompositions.

Constructing preconditioners

Approach 1: Remember the oldest iterative techniques!

To solve $Ax = b$ we can use *defect correction*:

- Under certain conditions, the iteration:

$$x^{(k+1)} = x^{(k)} - P^{-1}\left(Ax^{(k)} - b\right)$$

  will converge to the exact solution x

- Unlike Krylov-space methods, convergence is linear
- The best preconditioner is again $P^{-1} \approx A^{-1}$

Constructing preconditioners

Approach 1: Remember the oldest iterative techniques!

Preconditioned defect correction for $Ax = b$, $A = L+D+U$:

- Jacobi iteration:

$$x^{(k+1)} = x^{(k)} - \omega D^{-1}\left(A x^{(k)} - b\right)$$

- The Jacobi preconditioner is then

$$P^{-1} = \omega D^{-1}$$

  which is easy to compute and apply.

Note: We don't need the scaling (“relaxation”) factor.

Constructing preconditioners

Approach 1: Remember the oldest iterative techniques!

Preconditioned defect correction for $Ax = b$, $A = L+D+U$:

- Gauss-Seidel iteration:

$$x^{(k+1)} = x^{(k)} - \omega \left(L+D\right)^{-1} \left(A x^{(k)} - b\right)$$

- The Gauss-Seidel preconditioner is then

$$P^{-1} = \omega \left(L+D\right)^{-1} \qquad \text{i.e. } h = P^{-1} r \text{ solves } \left(L+D\right) h = \omega r$$

  which is easy to compute and apply as $L+D$ is triangular.

Note 1: We don't need the scaling (“relaxation”) factor.
Note 2: This preconditioner is not symmetric.

Constructing preconditioners

Approach 1: Remember the oldest iterative techniques!

Preconditioned defect correction for $Ax = b$, $A = L+D+U$:

- SOR (Successive Over-Relaxation) iteration:

$$x^{(k+1)} = x^{(k)} - \omega \left(D+\omega L\right)^{-1} \left(A x^{(k)} - b\right)$$

- The SOR preconditioner is then

$$P^{-1} = \left(D+\omega L\right)^{-1}$$

Note: This preconditioner is not symmetric.

Constructing preconditioners

Approach 1: Remember the oldest iterative techniques!

Preconditioned defect correction for $Ax = b$, $A = L+D+U$:

- SSOR (Symmetric Successive Over-Relaxation) iteration:

$$x^{(k+1)} = x^{(k)} - \frac{1}{\omega(2-\omega)}(D+\omega U)^{-1} D (D+\omega L)^{-1}(A x^{(k)} - b)$$

- The SSOR preconditioner is then

$$P^{-1} = (D+\omega U)^{-1} D (D+\omega L)^{-1}$$

Note: This preconditioner is now symmetric if *A* is symmetric!

Constructing preconditioners

Approach 1: Remember the oldest iterative techniques!

Common observations about preconditioners from stationary iterations:

- Have been around for a long time
- Generally useful for small problems (<100,000 DoFs)
- Not particularly useful for larger problems

Constructing preconditioners

Approach 2: Approximations to A^{-1}

Idea 1: Incomplete decompositions

- Incomplete LU (ILU):
  Perform an LU decomposition on A but only keep elements of L, U that fit into the sparsity pattern of A

- Incomplete Cholesky (IC):
  LL^T decomposition if A is symmetric

- Many variants:
  – strengthen diagonal
  – augment sparsity pattern
  – thresholding of small/large elements

Summary

Preconditioners for “simple” problem:

- Defect correction-based preconditioners are the simplest choice
- Limited by slow speed of information propagation
- ILU/IC frequently better but more complex and limited by memory requirements/CPU time
- Both kinds work reasonably well for “small” problems
- For elliptic problems we today have much better methods: *Geometric* and *Algebraic Multigrid (GMG, AMG)*

Overall summary

Sparsity has many implications:

- Finite element matrices
  – can be stored efficiently
  – can be multiplied by efficiently
  – can not be inverted efficiently
- Iterative methods work around that:
  – only require multiplication
  – iteratively improve the solution
- **But:** Speed of convergence depends on properties of A
- "Preconditioners" address this by solving $P^{-1}A\,U = P^{-1}F$ instead of $A\,U = F$.