Finite element methods in scientific computing

Wolfgang Bangerth, Colorado State University

Lecture 21:

Linear solvers for problems with more than one solution variable

Linear systems for vector-valued problems

Recall the mixed form of the Laplace equation:

$$\begin{aligned} K^{-1}\boldsymbol{u} + \nabla p &= \boldsymbol{0} \\ -\nabla\cdot\boldsymbol{u} &= -f \end{aligned}$$

The associated weak form is

$$(\boldsymbol{\phi}_{\boldsymbol{u}}, K^{-1}\boldsymbol{u}) - (\nabla\cdot\boldsymbol{\phi}_{\boldsymbol{u}}, p) - (\phi_p, \nabla\cdot\boldsymbol{u}) = (\phi_p, -f)$$

Expanding the solution and testing with discrete test functions yields this matrix:

$$A_{ij} = (\boldsymbol{\phi}_{\boldsymbol{i},\boldsymbol{u}}, K^{-1}\boldsymbol{\phi}_{\boldsymbol{j},\boldsymbol{u}}) - (\nabla\cdot\boldsymbol{\phi}_{\boldsymbol{i},\boldsymbol{u}}, \phi_{j,p}) - (\phi_{i,p}, \nabla\cdot\boldsymbol{\phi}_{\boldsymbol{j},\boldsymbol{u}})$$

Linear systems for vector-valued problems

Let us look at this more closely:

$$A_{ij} = (\boldsymbol{\phi}_{\boldsymbol{i,u}}, K^{-1}\boldsymbol{\phi}_{\boldsymbol{j,u}}) - (\nabla\cdot\boldsymbol{\phi}_{\boldsymbol{i,u}}, \phi_{j,p}) - (\phi_{i,p}, \nabla\cdot\boldsymbol{\phi}_{\boldsymbol{j,u}})$$

- We build the bilinear form like this in code because it's convenient
- However, in reality, shape functions are nonzero only in either the ***u*** or *p* components!
- I.e., they are either

$$\Phi_i = \begin{pmatrix} \boldsymbol{\phi}_{\boldsymbol{i,u}} \\ 0 \end{pmatrix} \qquad \text{or} \qquad \Phi_i = \begin{pmatrix} \mathbf{0} \\ \phi_{i,p} \end{pmatrix}$$

Note: See step-8/the "vector-valued module" on this.

Linear systems for vector-valued problems

Let us look at this more closely:

$$A_{ij} = (\boldsymbol{\phi}_{\boldsymbol{i},\boldsymbol{u}}, K^{-1}\boldsymbol{\phi}_{\boldsymbol{j},\boldsymbol{u}}) - (\nabla\cdot\boldsymbol{\phi}_{\boldsymbol{i},\boldsymbol{u}}, \phi_{j,p}) - (\phi_{i,p}, \nabla\cdot\boldsymbol{\phi}_{\boldsymbol{j},\boldsymbol{u}})$$

- If index *i* is so that the shape function is of ***u***-type, then we get a matrix row that is in fact

$$A_{ij} = (\boldsymbol{\phi}_{\boldsymbol{i},\boldsymbol{u}}, K^{-1}\boldsymbol{\phi}_{\boldsymbol{j},\boldsymbol{u}}) - (\nabla\cdot\boldsymbol{\phi}_{\boldsymbol{i},\boldsymbol{u}}, \phi_{j,p})$$

- If index *i* is of *p*-type, then we get

$$A_{ij} = -(\phi_{i,p}, \nabla\cdot\boldsymbol{\phi}_{\boldsymbol{j},\boldsymbol{u}})$$

- We can do the same for index *j*.

Linear systems for vector-valued problems

Let us look at this more closely:

$$A_{ij} = (\boldsymbol{\phi}_{i,u}, K^{-1}\boldsymbol{\phi}_{j,u}) - (\nabla\cdot\boldsymbol{\phi}_{i,u}, \phi_{j,p}) - (\phi_{i,p}, \nabla\cdot\boldsymbol{\phi}_{j,u})$$

- If we enumerate degrees of freedom so that all the ***u***-types come first and then the *p*-types, then this leads to a *block matrix*
- Here, we get:

$$A = \begin{pmatrix} M & B \\ B^T & 0 \end{pmatrix}$$

- How does one approach solving linear systems with this matrix?

Linear systems for vector-valued problems

What is difficult about this?

- The matrix

$$A = \begin{pmatrix} M & B \\ B^T & 0 \end{pmatrix}$$

  is symmetric but indefinite (i.e., has positive & negative eigenvalues)

- Conjugate Gradients (CG) requires the matrix to be symmetric & positive definite → won't work

- Preconditioners such as Jacobi or SSOR need to divide by diagonal elements → won't work

Linear systems for vector-valued problems

What should we do?

- Could use a direct solver that computes a sparse LU factorization of *A*
- Could use iterative solver alternatives to CG:
  - MINRES
  - SymmLQ
  - GMRES
- Would need to think more about preconditioners

More on this: Lectures 34-38.
For now: Exploit the block structure $A = \begin{pmatrix} M & B \\ B^T & 0 \end{pmatrix}$

Linear systems for vector-valued problems

Block structures in deal.II:

- Block structured matrices like

$$A = \begin{pmatrix} M & B \\ B^T & 0 \end{pmatrix}$$

  are represented by classes *BlockSparsityPattern/ BlockSparseMatrix*

- Block structured vectors like

$$U = \begin{pmatrix} u \\ p \end{pmatrix}$$

  are represented by class *BlockVector.*

Note: From here on, *u,p* are *vectors*, not solution variables.

Solvers

Let us consider the linear system:

$$\begin{pmatrix} M & B \\ B^T & 0 \end{pmatrix}\begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} F \\ G \end{pmatrix}$$

- This can equivalently be written as

$$\begin{aligned} Mu+Bp &= F \\ B^T u &= G \end{aligned}$$

- We can eliminate this variable by variable:

$$\begin{aligned} u &= -M^{-1}Bp+M^{-1}F \\ B^T M^{-1} B p &= -G+B^T M^{-1} F \end{aligned}$$

Solvers

Let us consider the linear system:

$$\begin{pmatrix} M & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} F \\ G \end{pmatrix}$$

- We have transformed this into a *decoupled system*:

$$\begin{aligned} B^T M^{-1} B p &= -G + B^T M^{-1} F \\ M u &= -Bp + F \end{aligned}$$

- We call $S = B^T M^{-1} B$ the *Schur complement* of the matrix
- S is a symmetric, positive definite matrix
- We can solve the equation for p with CG

Solvers

In order to solve the solve the system

$$B^T M^{-1} B p = -G + B^T M^{-1} F$$
$$M u = -Bp + F$$

we first need to solve with $S = B^T M^{-1} B$ and then with M.

Advantages:

- Both S and M are symmetric and positive definite
- We can use Conjugate Gradients and our usual preconditioners

Solvers

In order to solve the solve the system

$$\begin{aligned} B^T M^{-1} B p &= -G + B^T M^{-1} F \\ M u &= -Bp + F \end{aligned}$$

we first need to solve with $S = B^T M^{-1} B$ and then with M.

Problem:

- S is not known element by element
- We need to represent it in code
- We need to access individual blocks of A

But: The deal.II solvers do not need to know matrix elements. They only need operator actions!

step-20

Representing solvers for

$$\begin{aligned} S\,p &= -G + B^T M^{-1} F \\ M\,u &= F - Bp \end{aligned}$$

in deal.II requires us to **represent** *S,M* and present it to the CG solver.

Note: We also need a preconditioner. See the tutorial program for more information!

Solvers in deal.II

In deal.II, solvers are templated on both the vector and *operator* class:

```
template <class Vector>
class SolverCG {
    …
    public:
        template <class Matrix, class Preconditioner>
        void
        solve (const Matrix             &A,
               Vector                   &x,
               const Vector             &b,
               const Preconditioner &precondition);
};
```

Requirements: (i) Operator and vector need to be compatible; (ii) operator needs to have a function

vmult (Vector &out, const Vector &in);

Example

Here: Schur complement operator for mixed Laplace:

```
class SchurComplement {
   public:
      SchurComplement (const BlockSparseMatrix &A);

      void vmult (Vector<double>       &dst,
                  const Vector<double> &src) const
      {
        Vector<double> tmp1(src.size());
        Vector<double> tmp2(src.size());

        A.block(0,1).vmult (tmp1, src);
        some_inverse_of(A.block(0,0))->vmult (tmp2, tmp1);
        A.block(1,0).vmult (dst, tmp2);
      }
};
```

Example

Also necessary:

- Need to represent the inverse of *M*
- Need to come up with a way to precondition the Schur complement

Let's just take a look at the code!

Finite element methods in scientific computing

Wolfgang Bangerth, Colorado State University