**MATH521**
**Numerical Analysis of Partial Differential Equations**

Winter 2017/18, Term 2
Timm Treskatis

# Specifying Solvers for Linear Systems in FEniCS

## 1 General Workflow

1. Assemble the matrix $A$ and the right hand side $b$ of the linear system $Ax = b$.

   *Example:*

   ```
   A = assemble(dot(grad(u), grad(v))*dx) # stiffness matrix
   b = assemble(f*v*dx) # load vector
   ```

2. Apply Dirichlet boundary conditions, if applicable.

   *Example:*

   ```
   bc.apply(A) # apply boundary conditions to A only
   bc.apply(b) # apply boundary conditions to b only
   bc.apply(A,b) # apply boundary conditions to both A and b
   ```

   If $A$ or $b$ change in a time loop, remember to re-apply the boundary conditions after each such change.

3. Create a solver object for linear systems with the matrix $A$.

   *Example:*

   ```
   # solve linear systems by matrix factorisation (direct solver):
   solver = LUSolver(A)
   # solve linear systems with a Krylov subspace method (iterative solver):
   solver = KrylovSolver(A,'alg') # replace 'alg' with 'cg', 'minres' or 'gmres'
   ```

   If $A$ changes in a time loop, remember to update the solver after each such change:

   ```
   solver.set_operator(A)
   ```

4. Specify additional parameters (see below).

5. Solve the linear system.

   *Example:*

   ```
   u = Function(V) # if not already defined
   solver.solve(u.vector(), b) # solve Ax = b and write the solution to u
   ```

## 2 Direct Solvers

### LU Factorisation

```
directsolver = LUSolver(A)
directsolver.parameters.symmetric = False
```

If you have to solve multiple linear systems with the same matrix $A$ but different right hand sides $b$, use

```
directsolver.parameters.reuse_factorization = True
```

to keep the matrices $L$ and $U$ in memory.

### Cholesky Factorisation

```
directsolver = LUSolver(A)
directsolver.parameters.symmetric = True
```

If you have to solve multiple linear systems with the same matrix $A$ but different right hand sides $b$, use

```
directsolver.parameters.reuse_factorization = True
```

to keep the matrix $L$ in memory.

## 3 Iterative Solvers

### CG, MINRES and GMRES

```
# CG-method:
iterativesolver = KrylovSolver(A,'cg')
# CG-method preconditioned with incomplete Cholesky factorisation
iterativesolver = KrylovSolver(A,'cg','icc')
# Or use 'minres' or 'gmres' instead of 'cg'. To list all alternatives:
list_linear_solver_methods()
# Or use 'ilu' instead of 'icc' (for incomplete LU factorisation). To list all alternatives:
list_krylov_solver_preconditioners()
```

**Initial Guess** These iterative methods require an initial guess for the solution of the linear system. By default, the initial guess is the vector of all zeros. If you know of a better initial guess (e.g. the solution from a previous time step) set

```
iterativesolver.parameters.nonzero_initial_guess = True
```

Then

```
iterativesolver.solve(u.vector(), b)
```

will use the vector stored in `u.vector()` to start the CG, MINRES or GMRES iteration.

**Stopping Criterion** The above iterative solvers monitor the residual $\|r_k\| = \|Ax_k - b\|$ (where $k$ is the iteration counter).

- To stop when
$$\|r_k\| \leq \texttt{abstol}$$
use
```
iterativesolver.parameters.absolute_tolerance = abstol # e.g. 1E-9
```
- To stop when
$$\|r_k\| \leq \texttt{reltol}\|r_0\|$$
use
```
iterativesolver.parameters.relative_tolerance = reltol # e.g. 1E-6
```
- To stop when
$$k = \texttt{maxiter}$$
use
```
iterativesolver.parameters.maximum_iterations = maxiter # e.g. 1000
```