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 e.g. 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, 'cg') # or use 'minres' or 'gmres' instead of 'cg'
   ```

   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

2.1 LU Factorisation

```
directsolver = LUSolver(A)
directsolver.parameters['symmetric'] = False
```

If the matrix does not change, i.e. if you do not apply the `set_operator` method, the factorisation $A = LU$ will be computed once only and then L and U are kept in memory.

2.2 Cholesky Factorisation

```
directsolver = LUSolver(A)
directsolver.parameters['symmetric'] = True
```

If the matrix does not change, i.e. if you do not apply the `set_operator` method, the factorisation $A = LL^\top$ will be computed once only and then L is kept in memory.

3 Iterative Solvers

3.1 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()
```

3.2 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.

3.3 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
  $$\|r_k\| \leq \texttt{reltol}\|b\|$$
  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
  ```

If you wish to output the residuals, set

```
iterativesolver.parameters['monitor_convergence'] = True
```