MATH521
Numerical Analysis of Partial Differential Equations

Winter 2017/18, Term 2
Timm Treskatis

Homework Assignment 11

Even though the lion share of the computational expense when solving a PDE is the solution of the discrete linear system $Ax = b$, we have never spent a lot of thought on how to solve these systems. So far, we have simply used some high-level commands like the backslash in GNU Octave / MATLAB and `solve` in FEniCS. These commands would first run some initial tests to try and detect certain structure in the matrix, and then they choose a suitable method.

This can be done more efficiently. Since we already have a lot of knowledge about properties of the matrix A, we can select the best possible solver ourselves so that no additional testing is necessary at runtime.

Therefore, you will now learn about numerical methods for solving linear systems that stem from discretisations of PDEs. To choose a method that is (i) guaranteed to converge and (ii) as efficient as possible, you will have to use all your knowledge about the matrix A.

Question 1 considers *direct solvers*, which are useful for linear systems of moderate size, e.g. discretised 1D or 2D PDEs. Question 2 deals with *iterative solvers*, which are needed for very large systems, e.g. from 3D PDE problems.

Question 1 | 2 marks Read about LU factorisation (aka Gaussian elimination) and Cholesky factorisation.

(a) Which of these direct solvers is most appropriate for solving the two linear systems

$$\left(M^h + (\theta\Delta tc)^2 K^h\right)\vec{u}^h_+ = M^h\left(\vec{u}^h_\circ + \Delta t\vec{v}^h_\circ\right) - \left(\theta\,(1-\theta)\,(\Delta tc)^2\right)K^h\vec{u}^h_\circ \tag{1a}$$

$$M^h\vec{v}^h_+ = M^h\vec{v}^h_\circ - \Delta tc^2 K^h\left(\theta\vec{u}^h_+ + (1-\theta)\,\vec{u}^h_\circ\right) \tag{1b}$$

in the finite-element discretisation of the wave equation (cf Assignment 10) and why?

Solution. LU factorisation is a general matrix factorisation algorithm which works for any matrix A. Cholesky factorisation is a special case of LU factorisation for the case of symmetric positive definite matrices A, and is approximately twice as efficient.

The matrices M^h and K^h are both sparse, symmetric, positive definite matrices when arising from the discretisation of the homogenous wave equation. Then, the matrix $M^h + (\theta\Delta tc)^2 K^h$ is also symmetric positive definite, and so the matrix inversion in both equations 1a and 1b can be computed using the more efficient Cholesky factorisation.

□

(b) Make a copy of `hw10.py` (you may use the program from the model answers). The new script should integrate the wave equation as in Assignment 10, but it should solve the linear systems with the method you selected in part (a).

Hint: You can find some useful FEniCS commands on the enclosed cheat sheet. Create a solver object for (1a) and another solver object for (1b). Your code should run approximately three times faster than in Assignment 10.

Solution. TODO: see attached code □

Question 2 | 3 marks There are two main classes of iterative solvers: Krylov subspace methods and multigrid methods. We will look at Krylov subspace methods here.

Read about the conjugate gradient method (CG), the minimal residual method (MINRES) and the generalised minimal residual method (GMRES).

(a) Which of these iterative solvers is most appropriate for the linear systems in (1) and why?

Solution. The basic requirements of the listed Krylov subspace solvers are:

- CG: symmetric, positive definite matrix
- MINRES: symmetric matrix
- GMRES: no requirements; solver is fully generic

These methods are, naturally, also listed in decreasing order of efficiency.

Now, since the matrices M^h and K^h are both symmetric positive definite matrices, and therefore so is the sum $M^h + (\theta \Delta t c)^2 K^h$, the matrix inversion in both equations 1a and 1b should be solved using the conjugate gradient method. □

(b) Modify your FEniCS script from Question 1 to now solve the linear systems with the iterative method you selected in part (a).

Hint: Comment out the lines where you defined the direct solver objects. Create two iterative solver objects instead.

Solution. **TODO: see attached code** □

(c) Iterative methods typically converge significantly faster if they are applied to a preconditioned problem: instead of

$$Ax = b, \tag{2}$$

one solves the mathematically equivalent, but numerically advantageous problem

$$P^{-1}Ax = P^{-1}b. \tag{3}$$

The invertible matrix P, the so-called preconditioner, should on the one hand approximate A as closely as possible, but on the other hand it should be easier to invert than A. Such preconditioners are designed based on the specific properties of the linear system or the original PDE.

Note that if $P \approx A$, then $P^{-1}A \approx \text{id}$. This is what makes (3) more amenable to iterative solvers than (2).

Read about diagonal aka Jacobi preconditioning and incomplete Cholesky factorisation. Can you think of an even better preconditioner specifically for the mass matrix M^h than these two generic preconditioners?

Solution. The basic properties of the listed preconditioners are:

- Jacobi: The precondition P is given simply by the diagonal of the matrix A, and therefore P^{-1} is extremely easy to compute. This method only requires that the diagonal entries of A are non-zero.
- Incomplete Cholesky: The preconditioner P is given by an efficient to compute sparse approximation of the Cholesky factorisation, and simiarly requires that the matrix A is symmetric positive definite. This P is sparsely constructed in such a way that it is inexpensive to invert

Now, both of these generic preconditioners could be reasonably efficient for inverting the sparse symmetric positive definite mass matrix M^h. A more efficient preconditioner, however, would be the mass lumping matrix $\tilde{M}^h$, which is the diagonal matrix with entries $\tilde{M}^h_{ii} = \sum_j M^h_{ij}$. This matrix would be a more efficient preconditioner because not only is it an easy to invert diagonal matrix (with non-zero diagonal entries), but mass lumping only incurs a $\mathcal{O}(h^2)$ error in the first place and so we should expect $(\tilde{M}^h)^{-1}M^h \approx \text{id}$ to be a good approximation.

□

Your Oral Presentation You can find a worksheet attached to the presentation assignment on Canvas which you may want to use to prepare for your talk. Also familiarise yourself with the marking criteria which can be found on the same page to make sure you will cover everything that is needed.

You do not necessarily have to include final results in your talk. If you already have results to share, this is only to your advantage as it allows me to give you some feedback before you submit your written work where they will be graded.

Come up with an interesting title and write a succinct mini-abstract of at most three short sentences. This will also be made available to the public and should thus assume no specialised knowledge, but arouse interest for your talk. Please use the presentation assignment on Canvas to submit your title and your mini-abstract.

Your Learning Progress What is the one most important thing that you have learnt from this assignment?

Any new discoveries or achievements towards the objectives of your course project?

What is the most substantial new insight that you have gained from this course this week? Any *aha moment*?