MATH521: Numerical Analysis of Partial Differential Equations
Winter 2018/19, Term 2

Due Date: Thursday, 28 March 2019
Timm Treskatis

Homework Assignment 11

Please submit the following files as indicated below: source code PDF file image file video file

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

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

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

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

Comparing LU decomposition to Cholesky decomposition, we get that LU is more general, even working for non-square matrices. It works by factoring a matrix A into lower and upper triangular matrices such that $A = LU$ then solving the system $Ax = b$ by solving instead $Ly = b$ by forward substitution then solving $Ux = y$ via backward substitution. This has a computational cost of $2/3n^3 + \mathcal{O}(n^2)$, which is about the same as simple Gaussian elimination. Cholesky decomposition takes advantage of the properties of symmetric positive definite matrices by factoring into a lower triangular matrix and the transpose of that, such that $A = LL^T$, then solving in the same way as LU. This method avoids some of the computations necessary for calculating the U matrix and results in a computational cost of $1/3n^3 + \mathcal{O}(n^2)$, about twice as fast as LU (Liesen and Strakoš, 2000).

Since we already know that the mass and stiffness matrices are both symmetric and positive definite (as shown in HW5), which is the requirement for Cholesky decomposition, Cholesky is the recommended method for solving this system.

J. Liesen, Z. Strakoš. On the computational cost of Krylov subspace methods for solving linear algebraic systems. 2000.

(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.

HW10 Solve time: 166.95s
Cholesky time: 43.91s
LU time: 43.60s

In theory the Cholesky method should be faster than LU, but for this specific problem I guess they're about the same.

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?

Between CG, MINRES, and GMRES, the conditions are as follows: symmetric positive definite for CG, symmetric for MINRES, and nonsingular for GMRES. So based on CG having the strictest requirements, if our system fits it should be the best, right? From what I've read, while that logic may not always hold, CG is in fact the best solver for our system. It has a computational cost of $\mathcal{O}(n^{3/2})$ (Shewchuk, 1994) and is guaranteed to reach the exact solution using exact arithmetic after n-iterations, if not sooner. The MINRES method is a bit slower than CG, requiring an additional seven simple vector operations per iteration, while GMRES is about the same as MINRES, however it can be rather storage prohibitive as it must store the Krylov space from each previous step (Wathen, n.d.).

Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain. 1994.

A. J. Wathen. Preconditioning. n.d.

(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.

Method	No Pre	ICC	ILU	Jacobi
CG	45.47s	47.18s	46.19s	46.17s
MINRES	45.92s	49.01s	46.36s	47.60s
GMRES	46.07s	47.22s	46.37s	46.50s

(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? Also give a brief reason to motivate your proposed preconditioner:

Jacobi preconditioning simply multiplies our matrix A by its own diagonal, that is $P = diag(A)$. This is useful for a well conditioned matrix with bad scaling, but in reality usually doesn't do a whole lot. Incomplete Cholesky preconditioning, like Cholesky factorisation, breaks our matrix A into a lower triangular matrix L and its transpose L^T such that $A = LL^T$, except that instead of filling in the matrix L ICC leaves any cell that is zero alone, creating a much sparser triangular matrix. This method works well for diagonally dominant matrices (Wathen, n.d.).

A better preconditioner for the mass matric M^h may be simply to combine the Jacobi and ICC, that is fill the lower triangular matrix sparsely and use the diagonal of M^h. Hopefully, this would have the properties of each, as the mass matrix is indeed diagonally dominant (HW5), and scaling can be an issue (more so with the stiffness matrix).

A. J. Wathen. Preconditioning. n.d.

Your Learning Progress | 0 marks, but -1 mark if unanswered | What is the one most important thing that you have learnt from this assignment?

For this rather small problem, it doesn't really matter which method or type of preconditioning you use.

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

While some methods should perform one way in theory, they frequently perform very different in practice. I'm still kinda of confused on why though, as many of our practice problems are not very different from the theory. Maybe just something to do with the limited precision of the computer?