KTH Engineering Sciences

Computer Exercise 5

Iterative methods for linear systems of equations

In this exercise you shall make some experiments with different iterative methods for linear systems of equations. In particular you shall study the convergence and computational cost of the methods.

Part 1: Convergence of Jacobi and conjugate gradient

We consider the *Jacobi* and the *conjugate gradient* methods applied to matrices that come from finite difference discretizations of the Poisson equation in 1D, 2D or 3D. We let n be the number of grid points in one coordinate direction and d be the dimension. (Thus $\Delta x \sim 1/n$ and n is therefore a measure of how well resolved the problem is.) The matrices are then of size $N \times N$ where $N = n^d$. You can use the function `lap.m` available on the homepage to generate the matrices for different resolutions and dimensions with the command `A=lap(n,d)`.

Background theory. Suppose the linear system is $Ax = b$ and that the iterates are denoted x_k. The error in x_k then decays as

$$||x_k - x|| \leq C\beta^k, \tag{1}$$

where C is independent of k and $0 < \beta < 1$ is the *convergence rate*. A small β thus means fast convergence, while β close to one means slower convergence. The latter case is typical for matrices of interest, and we therefore write $\beta = 1 - \delta$ where $0 < \delta \ll 1$. In general, the number of iterations needed to achieve a fixed accuracy will then be inversely proportional to δ. More precisely,

$$||x_k - x|| \leq \varepsilon \qquad \text{if} \qquad k \geq \tilde{C}/\delta, \qquad \tilde{C} = |\log(\varepsilon/C)|. \tag{2}$$

This follows since $C(1-\delta)^k \leq \varepsilon$ when $k|\log(1-\delta)| \geq |\log(\varepsilon/C)|$ and $|\log(1-\delta)| \approx \delta$.

In the Jacobi case β in (1) is the 2-norm of the matrix $D^{-1}(A - D)$ where D is the diagonal part of A (when A is symmetric). For the Poisson matrices one can show that this norm is well approximated by $1 - \bar{c}/n^2$ for some constant $\bar{c}$ when $n \to \infty$. Hence, by (2) the number of iterations needed for a fixed accuracy grows as $O(n^2)$.

For the conjugate gradient method, $\beta \leq 1 - 2/\sqrt{\kappa}$, where κ is the condition number of A. Hence, the convergence of conjugate gradient is in general slow for ill-conditioned matrices. In the Poisson case, κ grows as $O(n^2)$, and therefore by (2) the number of iterations needed for a fixed accuracy grows as $O(n)$.

Tasks. Implement the Jacobi and the conjugate gradient methods. Apply them to discretizations of the Poisson equation given by `lap(n,d)`. Start the iterations from $x_0 = 0$. Let the the right hand side vector b be a random vector obtained with the command `b=rand(N,1)`. Do not use the built in `pcg` function here.

(a) Make convergence plots, where the relative size of the residual, $||Ax_k - b||_2/||b||_2 = ||r_k||_2/||b||_2$, is plotted as a function of iteration number k using the MATLAB command `semilogy`. The goal is to compare the convergence of the methods for different resolutions n and dimensions d. Results for varying n and/or d should therefore be overlaid in the same figure to make such comparisons easy.

Draw conclusions and discuss:

- Is the convergence faster or slower with larger n (finer resolution)?
- Which method converges faster? Does this depend on the dimension d?
- Does the number of iterations needed for a fixed accuracy agree with the theory above?
- Is the convergence faster or slower with larger d, for a fixed matrix size N?

You should not plot all combinations of methods, n and d. Select a few cases that you think are the most relevant, where the plots best help you support the conclusions that you draw and illustrate the discussion around it. Make sure to include information about what methods, n and d are used in all plots.

Hints:

- The matrices should be quite large. Always take $N \geq 2000$.
- Make the code efficient. Use sparse format for the matrix and make sure only *one* matrix-vector multiply is performed in each iteration. (This is the largest cost of the iteration and should be made as small as possible.) In the end, there should be no problem using matrices of size $N \approx 40000$ for instance.

(b) Change your conjugate gradient code such that it continues to iterate until the relative residual size is less than 10^{-10}. Your task is now to compare the computational time for this code against the computational time for MATLAB's backslash command. (Verify numerically that the two solutions are almost the same.) Experiment with different choices of resolution n and dimension d. When is your conjugate gradient code faster than backslash? For what choices of n and d? Give one example when it is faster and one where it is slower.

Try to explain your results based on the theoretical costs of solving banded linear systems by direct methods and the theoretical convergence rates of the iterative methods given above.

Part 2: Built-in MATLAB Krylov methods and preconditioning

Consider the stiffnes matrix that COMSOL uses in the *Cooling Flange* example, found in the COMSOL application libraries. (See Figure 1.) It is a FEM discretization of an elliptic equation modeling steady heat flow. The matrix is available in the `cooling_flange.mat` file on Canvas. It is a sparse, symmetric positive definite matrix of size 35296×35296. Visualize its sparsity structure with the `spy` command.

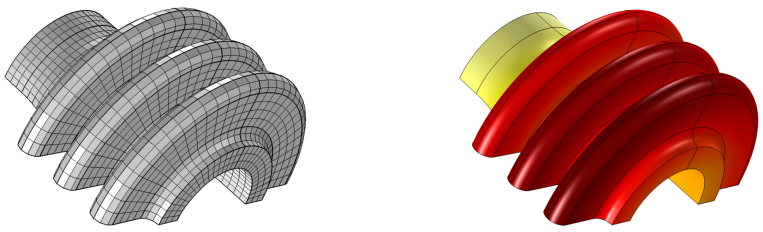

Figure 1. Cooling flange: mesh (left), temperature distribution (right).

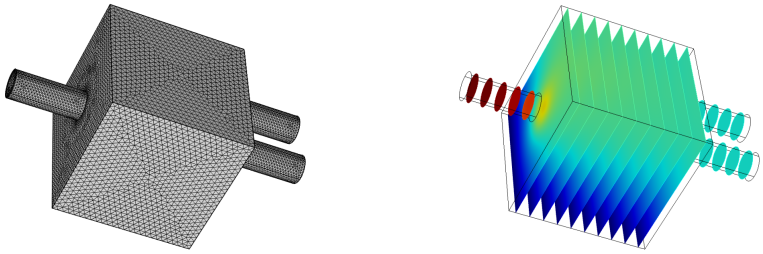

Figure 2. Convection diffusion: Hot liquid enters through the left pipe. Bottom plate is kept at a low temperature. Cooled liquid exits to the right. Mesh (left), temperature distribution (right).

(a) Use the built-in function `pcg` for the conjugate gradient method to solve $Ax = b$ with b random. Do `help pcg` to learn about the syntax. Use a tolerance of 10^{-4} and all return values `[X,FLAG,RELRES,ITER,RESVEC]`. Choose a large enough value for max iterations to converge (`FLAG=0`). Plot the convergence history (`RESVEC`) as in Part 1. Report the number of iterations used (`ITER`) and the size of the final relative residual (`RELRES`).

Compare the computational time of `pcg` with that of MATLAB's backslash command for this matrix.

The convergence rate of conjugate gradient method can be improved by *preconditioning* with a matrix M, which should be an approximation of A that is easy to invert. Essentially[1], the preconditioned conjugate gradient then solves $M^{-1}Ax = M^{-1}b$ instead of $Ax = b$. If M approximates A well, then $M^{-1}A \approx I$. The convergence rate depends on the condition number of the system matrix, and since the condition number of I is one, the convergence will be very fast. In practice, the approximation M of A can, however, be rather crude and still give good effect.

(b) Try M=the diagonal part of A. (In `pcg` you add M as an argument after `MAXITER`.) Also try $M = LL^T$ where L is the *incomplete Cholesky factorization* of A given by the `ichol` command. (Here you add both L and L^T to the `pcg` arguments.)

Repeat the experiments in (2a) and the comparison with MATLAB's backslash for both preconditioners.

Consider finally another COMSOL application: the cooling of a liquid advected through a pipe-splitter (see Figure 2). This is governed by a convection-diffusion equation which contains both first and second order derivatives. First order derivatives give rise to skew-symmetric matrices. The final COMSOL discretization matrix is therefore unsymmetric. The size is 55096×55096. The matrix is available in the `convdiff.mat` file on Canvas.

(c) The conjugate gradient method cannot be used for unsymmetric matrices. Verify that `pcg` indeed does not converge for this matrix. Use instead *GMRES*. The MATLAB command is `gmres` and the syntax is essentially the same as for `pcg`. Do not use the restarted version. Repeat the experiments in (2a) and (2b) for this matrix with GMRES. For the preconditioning you need to replace incomplete Cholesky with *incomplete LU factorization*, command `ilu`, since the matrix is not symmetric. (Add the two different matrices L and U to the `gmres` arguments.)

Remark: It may be difficult to use backslash for this matrix, if you do not have a lot of memory in your computer.

[1]Note that $M^{-1}A$ may not be symmetric positive definite even if both M and A is. The preconditioned conjugate gradient therefore actually implicitly solves the symmetric positive definite system $M^{-1/2}AM^{-1/2}\tilde{x} = M^{-1/2}b$, although in the algorithm only solutions of $Mz = d$ are needed.