DN2222 - Autumn 2011

Programming assignment 2

Direct Methods for Sparse Matrices

Study the relevant topics before preparing solutions to the assignments. **D:2.7** and **L:2**. In the reading advice **D** means the book *Applied Numerical Linear Algebra* by **JW Demmel** and **L** (and sometimes **R**) means the lecture notes *Topics in Numerical Linear Algebra* by **A Ruhe**.

Extra reading:

In Matlab's document browser you will find an instructive description. Choose `MATLAB -> User Guide -> Mathematics -> Sparse Matrices`. Some extra details and help for this assignment are given on the course home page as "Extra tips".

If you are interested, you can read the original paper: **John R Gilbert, Cleve Moler and Robert Schreiber:** *Sparse matrices in MATLAB: Design and Implementation*, SIAM J Matrix Anal. Appl. Vol 13, No 1, January 1992, pp 333-356. (Download-able from the course's home page).

Test matrices can be found on the *Matrix Market* of the *National Institute of Standards and Technology (NIST)*. (http://math.nist.gov/MatrixMarket/).

You are encouraged to work in groups of two and you may work at any time. The course assistant Ashraful Kadir will answer questions concerning these assignments at the scheduled lab sessions.

Hand in a report by November 28 at the Student expedition. The report should be short and show your conclusions. It may be hand written and should preferably include MATLAB-plots, etc.

Do not include all of your MATLAB code or printout. Only include a few lines showing the most interesting parts/results. I do not want to see long MATLAB programs or diary files!

Exercise 1. Compare re-orderings

We will start with a matrix coming from applying a finite difference approximation on the Laplace equation (the regular five-point stencil) over some different 2D domains, both square and L-shaped.

You will compare the standard numbering with those obtained by RCM, MMD (really AMD, see below) and ND. You will compare node numbering order, matrix structure, fill in, etc.

Exercise 1a. Standard numbering

The standard matrix can be obtained by Matlab from the routines `numgrid` (to obtain the grid matrix G, ie node numbering) and `delsq` (to obtain the corresponding FDM-matrix, A). Use both square- and L-shaped domains and try different sizes `n`. (The L-shaped domain with a `n=15` grid is the picture you see when Matlab starts!)

Check the structure of the matrix A with the routine `spy`. Then do Cholesky factorization and check the fill in.

Exercise 1b. RCM

Now renumber the nodes using the Reverse Cuthill McKee method. (Use the Matlab routine `symrcm`). Check the structure of the new matrix A. Then do Cholesky factorization and check the fill in.

Look at the grid reordered by RCM. You may use the routine `v2g` as described in "Extra tips". Note that you need the inverse permutation vector (which can be obtained by: `pr=symrcm(A); iv=1:n; rp(pr)=iv;`)

Exercise 1c. MMD/AMD

Now renumber the nodes using the Approximate Minimal Degree (Use the Matlab routine `symamd`). Check the grid numbering. Check the structure of the new matrix A. Then do Cholesky factorization and check the fill in.

Exercise 1d. ND

Now renumber the nodes using the Nested Dissection. Unfortunately there is no routine for re-ordering but the Matlab routine `nested` does a direct numbering. It only works for square domains though. Check the grid numbering. Check the structure of the new matrix A. Then do Cholesky factorization and check the fill in.

Try to explain your observations! Also, try to find an illustrative way to show the different grid numbering.

Note: The Approximate Minimum Degree ordering (AMD) is different from the Minimum Degree ordering (MMD). Previous there was a function `symmmd` in Matlab but it has been replaced by `symamd`. However, AMD appears to be always better than MMD.

Exercise 2. When is sparse factorization an advantage?

The sparse factorization will use much less storage space and fewer arithmetic operations than a full matrix code. On the other hand, it needs quite a lot of book-keeping to track at which places fill in occurs. Now we want to see when it is of advantage to use a sparse matrix code.

In this exercise we will use an L-shaped domain. Make the matrix `A` the standard finite difference matrix. Take the right hand side `b` as a one (1) in one or a set of contiguous points (or just close points) in the center of the grid `G`. Then the solution `x` will look like a circus tent with poles in the positions where b is one.

You may use the routine `X=v2g(x,G)` to get a matrix `X` from the vector `x` spread over the grid `G` and plot it by the command `mesh(X)`. (The non-Matlab routine `v2g` can be found on the course home page).

Exercise 2a. Experiment with permutations in sparse code

Let the number of grid points vary. Make up a table of the number of nonzeros in the original matrix A and the Cholesky factors for the original ordering and the reorderings RCM and AMD. Determine the permutations and measure the time (using Matlab's `tic` and `toc`) needed for permutation, factorization and solution respectively. Also determine the storage space needed (using `whos`).

Then check the solution: Compute the residual `r=A*x-b` and list the norm of `r`.

Start with a moderate size, say `n=12` along one side. Then (when your code is debugged) increase the grid size, say by a factor of 2, until you either run out of memory or the experiment takes longer than say 3 minutes. Record for which `n` this happens, and report which machine you use! Provide in your report a table over `n` and the different run-times.

As a comparison, measure the time for Matlab's built-in implementation for solving a linear system, \ ("backslash"). Include also these values in your table. (It will probably be the fastest in your table - not only because it is not divided into three separate calls.)

Exercise 2b. Experiment with permutations in full code

Do and report the same computations as in 2a but with full matrices. Report also for which matrix sizes the full code is faster than the sparse one.

A note on timing: The Matlab clock ticks very slowly. You might need to run the quick operations several times and the divide the total time by the number of repeats to get accurate values.

Good luck! /Ninni