Michael Hanke
CSC NA
September 24, 2009

DN2222
Numerical Algebra

KTH Computer Science and Communication

Programming assignment 2

Sparse matrices, direct algorithms

A short introduction is given in the lecture notes Chapter 2!

In MATLAB's `doc` browser you will find an instructive description. Choose `MATLAB -> Mathematics -> Sparse Matrices`. Some details for the present assignment are downloadable as "Extra Tips" from the courses home page.

If you are really interested, read the original paper: Gilbert, John R., 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. (Downloadable as `gilbert92sparse.pdf` from the course's home page).

Test matrices are 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, Henrik Holst, will answer questions concerning these assignments at the scheduled lab sessions.

Hand in a report by November 25, at the Student Expedition! The report may be hand written, but would preferably include plots and matrices from MATLAB.

Do not hand in your entire MATLAB code, only a few lines where the interesting computations are done. *I will not like to see long* MATLAB *programs or* `diary` *files!*

A 1. **Simple example: Compare reorderings!**

Start with the matrix you get from a finite difference approximation of the Laplace equation. You get a grid by the command `G = numgrid` and a matrix by `A = delsq(G)`! Look at the matrix with `spy(A)`.

Take an appropriate size, start with an L-membrane with `n = 15` grid points along one of the long sides . This is the one that you see when you start MATLAB! Look at the grid matrix `G` and see how the points are ordered by columns.

You will get permutation vectors for band width reduction (RCM) and approximate minimal degree (AMD) by the calls `pr = symrcm(A)` and `pm=symamd(A)`. Substructuring, nested dissection, is not implemented,

but there is an option in `numgrid` that gives that order for a square. Look at the reordered matrices with `spy` and compare to what I showed in the lecture!

Look at the grid `G` reordered by Reversed Cuthill McKee. You can use the routine `v2g` as described in Extra tips. *Note* that it cannot be used directly on the permutation vector `pr`, it must have the inverse permutation. It is simple to get the inverse permutation by doing the call

```
iv = 1:n; rp(pr) = iv;
```

which gives the inverse permutation in the vector `rp`.

Does your result look like what I showed in the lecture?

Note: The Approximate Minimum Degree ordering of G is different from the Minimum Degree ordering. Previously, there was an implementation, `symmmd`, available in MATLAB. However, AMD appears to be always better than MMD.

A 2. **When is sparse factorization of 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.

Matrix: Take the matrix `A` as finite difference matrices over a L-membrane as in previous task.

Right hand side b: Take the vector `b` as a 1 (one) in one or a set of contigous positions in the center of the grid `G`.

Solution x: Plotted over the membrane, it will look like a tent with poles in those positions where `b` is one.

You can use the routine

```
X = v2g(x,G);
```

to get a matrix with the vector `x` spread over the grid `G` and plot it by the command `mesh(X)`.

(a) **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 needed for

i. Permutation: Find the permutation of the indices for the reordering

ii. Factorization: Compute the LU factors of the reordered matrix

iii. Solution: Solve a system for a new right hand side `b`

Check the solution: Compute the residual `r = Ax-b` and list $\|\mathtt{r}\|$!

Start with a moderate grid, say `n = 12` points along one side. When your code works, increase the grid size until you either

run out of memory or the experiment takes more time than say 3 minutes. Record for which `n` that happens, and report which machine you use! It may be different on different workstations or PC:s.

As a comparison, take time for the built in implementation of the complete solution of a linear system `x = A\b`. I think it uses AMD reordering. The time may be shorter than the sum of the 3 times above, memory allocation may take time. In industrial codes you always divide the process, because the permutation only needs to be done once for each matrix size, the factorization only once for each matrix and the solution many times for each matrix.

(b) **Compare sparse and full matrix code**

Just call

```
AF = full(A);
```

and you get a full matrix. Do the same computations as in the previous task. Record the time and space needed, and report for which matrix sizes the full matrix code is faster than the sparse one. Be careful, the full matrix may need too much memory for a much smaller `n` than a sparse matrix! On my account, I got `Out of memory` for `n` just above 4000.

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