

MIT 18.335, Fall 2006: Homework 6

Due December 12

Fill-Reducing Permutations

1. Write a MATLAB function `p=nestdiff(n)` that computes a nested dissection permutation p for a regular n -by- n grid. To compute a separator for a rectangular block, simply split across the longest edge size.

As an example, for $n = 5$ you could renumber the nodes according to below:

$$\begin{array}{ccccc} 21 & 22 & 23 & 24 & 25 \\ 16 & 17 & 18 & 19 & 20 \\ 11 & 12 & 13 & 14 & 15 \\ 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 \end{array} \longrightarrow \begin{array}{ccccc} 7 & 8 & 25 & 17 & 18 \\ 5 & 6 & 24 & 15 & 16 \\ 9 & 10 & 23 & 19 & 20 \\ 3 & 4 & 22 & 13 & 14 \\ 1 & 2 & 21 & 11 & 12 \end{array}$$

which gives the permutation

$$p = [1, 2, 6, 7, 16, 17, 21, 22, 11, 12, 4, 5, 9, 10, 19, 20, 24, 25, 14, 15, 3, 8, 13, 18, 23].$$

Try to verify that your code is correct, for example by studying the output for small n or by plotting the sparsity pattern of a permuted Poisson model problem. Also make sure it is fast enough to handle sizes up to $n = 160$.

2. Study the fill-in for the Poisson model problem of size n ,

```
A=delsq(numgrid('S',n+2));
```

using four different reordering schemes:

- (a) Natural ordering (no reordering)
- (b) Reverse Cuthill-McKee (`symrcm`)
- (c) Approximate minimum degree (`symamd`)
- (d) Nested dissection (from problem 1.)

Permute A using each of the schemes and compute Cholesky factorizations of the permuted matrices. Use the problem sizes $n = 10, 20, 40, 80$, and 160 , and plot the number of non-zeros in the Cholesky factor as a function of n in a log-log graph. Try to determine the dependency on n , that is, $O(n^\alpha)$ or $O(n^\alpha \log n)$ for some value of α .

Incomplete Factorizations

3. Let L, U be an ILU(0) factorization of A , and set $B = LU$. Show that if $a_{ij} \neq 0$ then $a_{ij} = b_{ij}$, that is, A and B only differ in elements outside the non-zero pattern of A .

Turn the page \implies

Preconditioned Conjugate Gradients

Problems 4-7 use the linear elasticity problem from Homework 5. In class we showed how to write a CG solver in MATLAB and how to solve the elasticity problem. All required functions are available on the course web pages. The following lines solve a typical problem and plot the convergence of the residuals:

```
[K,L]=mkmodel(20);  
[x,res]=cg_stats(K,L,0*L);  
semilogy(1:length(res),res)
```

In this homework you will solve this problem using preconditioned conjugate gradients with three different preconditioners.

4. Write a preconditioned conjugate gradients solver `pccg_stats` based on `cg_stats`. Pass the preconditioner to the function by an additional input argument `R`, corresponding to the Cholesky factor of the preconditioner $M = R^T R$. You can then apply M^{-1} efficiently by back-substitutions using MATLAB's backslash.
5. Write code that produces the Cholesky factors for the following three preconditioners:
 - (a) $M =$ tridiagonal part of K (see `spdiags`, use `chol` for R)
 - (b) $M =$ incomplete Cholesky with no fill-in (see `cholinc`, parameter '0')
 - (c) $M =$ one step of SSOR =

$$\frac{1}{2-\omega}(D/\omega - L)(D/\omega)^{-1}(D/w - L)^T$$

with D, L as defined for the classical iterative methods and $\omega = 1.6$. Do not form M explicitly, instead figure out its Cholesky factor analytically.

6. Create K, L for $n = 20$ as in the MATLAB lines above, and run your `pccg_stats` code with the three preconditioners. Plot the convergence of the residuals in a lin-log graph. Estimate the convergence rates of the three cases and for standard CG without preconditioning (assuming linear convergence as in Homework 5). Compare with the upper bound $1 - 2/\sqrt{\kappa}$, where κ is the condition number of $M^{-1}K$ (or just K for standard CG). You can compute κ using `cond(full(A))` since the matrices are relatively small.

In some of the cases the convergence rates are much better than the bounds based on κ . Briefly explain this by studying the eigenvalue distributions of the preconditioned matrices. Again you can use full matrices, e.g. `e=sort(eig(full(A))); plot(e,1:length(e),'.')`;

7. If you have implemented the code efficiently (no dense matrices and only triangular solves), you should be able to solve up to $n = 160$ (although `cholinc` is very slow for large problems). Solve using the two preconditioners (b) and (c) for $n = 10, 20, 40, 80$, and 160. Plot the number of iterations required for convergence versus n in a log-log graph, and estimate the slope.

For standard CG and "pure" SSOR with optimal ω we expected the number of iterations to grow linearly with n (since the convergence rates are $1 - C/n$). Do these preconditioners seem to improve on this exponent 1?