September 13, 2017

## Matlab Assignments – Lecture 2, Fall 2016

In these assignments we will investigate how the performance of MATLAB's LU-decomposition can be improved by applying a suitable reordering scheme. We will also investigate the effect of pivoting on the final accuracy.

Download the files `ocean.m, topo.mat` and `ocean.mat` from the course webpage. The file `ocean.mat` contains the matrix **A**, the right-hand side vector **b** and a matrix **P**. These files define an ocean circulation model on a grid with a resolution of $3^0$. **P** maps every entry in the solution vector **x** onto a corresponding gridpoint on the map of the world.

**Assignment 2.1**. In this assignment you will device a renumbering scheme to minimise the fill in of the matrix of the ocean problem.

(a) Run the script `ocean`. You will see the nonzero structure of the matrix **A**. Determine the percentage of nonzero elements in $A$. The matrix **A** is banded, except for a few arrows. These arrows are related to circulation conditions around islands and continents, hence each arrow in **A** corresponds to a continent. Notice that **A** is stored using MATLAB's sparse matrix storage. Only the nonzero elements of **A** and the corresponding indices are stored.

(b) Make an LU-decomposition of **A** using the commands

```
thresh = 0;     LU = lu(A,thresh);.
```

The parameter `thresh` determines a pivoting threshold (`help lu`). Since in the first assignment we only study reordering of the equations to minimise fill in, we suppress reordering of the equation to enhance numerical stability by putting `thresh = 0;`. Check the nonzero pattern in the LU-decomposition of **A**. Determine the percentage of nonzero elements.

(c) Clearly, the 'arrows' in the matrix cause a lot of fill in. A simple idea to overcome this problem is to change the direction of the arrows. This comes down to eliminating the unknowns in reverse order.
Reorder the matrix in this way. To reorder you can define an index vector `I` with entries `I(i)=j` in which `i` is the original number of the unknown and `j` the new number of the unknown after reordering. The statement `B = A(I,:)` reorders the rows of the matrix, the statement `B = A(:,I)`, and `B = A(I,I)` reorders both the rows and the columns and preserves the symmetric structure of the matrix. Note that this is much cheaper than defining a permutation matrix, and multiplying with it! Check the sparsity pattern of the permuted matrix. Compute its LU-decomposition and check the resulting nonzero pattern. What is the percentage of the nonzeros in the **L** and **U** factors?

(d) Row with many zeros cause a lot of fill in (why?). It is therefore a good idea to permute the rows with the most nonzeros to the bottom of the matrix. Make an algorithm that constructs an index vector $I$ that permutes the rows with the most nonzero elements to the bottom of the matrix. Apply this permutation symmetrically to preserve the symmetric structure in the matrix. Check the sparsity pattern of the permuted matrix. Compute its LU-decomposition and check the resulting nonzero pattern. What is the percentage of the nonzeros in the **L** and **U** factors?

(e) The **Reverse Cuthill–McKee** algorithm aims to minimise the bandwidth of the matrix. A Reverse Cuthill–McKee ordering can be computed with the command `I = symrcm(A)`. Apply this permutation, check the resulting sparsity pattern of the permuted matrix and of the LU-factorisation of this matrix. What is the percentage of the nonzeros in the **L** and **U** factors?

(f) The **minimum degree** ordering aims to minimise the fill in. This ordering can be computed with the command `I = realmmd(A)` (see also MATLAB's command `amd` and `symamd`). Apply this permutation, check the resulting sparsity pattern of the permuted matrix and of the LU-factorisation of this matrix. What is the percentage of the nonzeros in the **L** and **U** factors?

(g) You have used different permutation strategies. Which one worked best for our problem?

**Assignment 2.2**. In this assignment you will investigate the effect of pivoting on the fill in and on the accuracy of the solution.

(a) Use the Reverse Cuthill-McKee algorithm. Make an LU-decomposition while you suppress pivoting `thresh = 0`. Calculate the solution of the system using this LU-decomposition. Check the term $|\widehat{\mathbf{L}}|\,|\widehat{\mathbf{U}}|$. Solve the system using this decomposition. Permute your solution back to the original ordering and compare it with the solution that you get with MATLAB's backslash command \\. Repeat this assignment, but now perform partial pivoting by setting `thresh = 1`. Does partial pivoting cause considerable additional fill in *in this case*?

(b) Same assignment, but now use the minimum degree ordering.