
Then in no more than 1/2 page discuss the main differences between the method described in this paper and that used in the standard reverse Cuthill Mc Kee ordering.

2. Consider the matrix (and adjacency graph) shown at the top of Page 6-18 of the lecture notes. 1) Let \( \pi = [1, 6, 5, 3, 2, 4] \). Show the adjacency graph and new pattern of the matrix \( B \) obtained by symmetrically permuting the original matrix according to the permutation \( \pi \). 2) Find all fill-ins generated by Gaussian elimination on the matrix \( B \) by showing all fill-paths in the graph. 3) Find the Cuthill McKee (CMK) ordering for \( B \) (For tie breaking assume that adjacency lists are sorted increasingly). 4) Find its reverse Cuthill McKee ordering and show the adjacency graph as well as the corresponding reordered matrix \( C \). 5) Repeat question 2 when \( B \) is replaced by \( C \).

3. Problem 3-11 from textbook (p. 103 of on-line edition, p. 100 of the SIAM edition). [Hint for question b: a graph that is 2-colorable cannot have a cycle of odd length]

4. (Question in P. 7-26 of lecture notes). Show that the size of the independent set \( I \) found by the greedy algorithm (P. 7-24 of notes) satisfies: \( |I| \geq n/(d_I + 1) \), where \( d_I \) is the max. degree of all nodes in \( I \) (self-cycles \((i, i)\) are not counted). How does this compare with the actual size obtained for a graph associated with a 5-point (rectangular) finite difference grid.

5. Part of this exercise is a continuation of HW1. Recall how we exploited Depth-First Search (DFS) for solving triangular systems with sparse right-hand sides (RHSs). We saw in class that this can be used to implement a sparse LU factorization algorithm based on a left-looking Gaussian elimination (called Gilbert and Peierls algorithm). The goal of this question is to program the algorithm in matlab by paying attention to efficiency. So the algorithm should be coded in matlab but with an understanding of how you would do it in C/C++, or Fortran.

a. Implement a topological sorting algorithm. For simplicity you will just use a recursive (inefficient) version for now so you can use the one you developed in HW1. The script should return the array Lst which lists the order in which to determine each unknown.

b. Based on the script JKI1.m [posted in matlab page] develop a sparse version of the LU factorization. There are comments in the script indicating the main changes you will need to make. There are two main differences with the dense JKI script. First the loop for \( k=1:j-1 \) now becomes for \( l=1:length(Lst) \) since we are solving only for unknowns in Lst and in the order given by Lst. Second, if you program the code in C, you would allocate the \( j \)th column of \([L, U]\) (called \( w \) in the notes and the JKI1.m script). So you need to explicitly use a temporary column \( w \) and when this column is computed copy the appropriate parts to \( L \) and \( U \) from it.

c. Once you have debugged your script try it on a sample Laplacian problem using the fd3d code you already used. Use a \( 20 \times 20 \times 1 \) mesh.

d. Learn how to use profiling in matlab (profile on) Profile your execution. How much time is spent where? How would you recode the dfs part?

e. Optional [10 bonus points out of 100]. Recode the DFS code into a non-recursive version and redo the profiling.