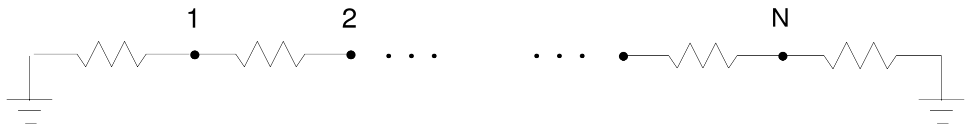
Course 6.336J/18.338J - Introduction to Numerical Algorithms (Fall 2003)
Problem Set Number 2 - Due September 19

1) As the previous problem set made clear, the equations associated with a one-dimensional discretized Poisson's equation can be represented by a line of resistors, as show below (Note, the grounded ends of the resistor line and the fact that there are no current sources implies the circuit models zero temperature boundary conditions for the Laplace's equation (zero right hand side)).



a) Determine the number of nonzero entries in the $N \times N$ conductance matrix associated with a N-long line of resistors.

b) Determine the number of nonzero entries in the $N \times N$ resistance matrix associated with the N-long line of resistors (assume all the resistors are positive). The resistance matrix is the inverse of the conductance matrix.

c) Determine the number of nonzero entries in the the L and U factors of the conductance matrix associated with a N-long line of resistors. Compare to your result from part (b) for $N = 1000$ (Moral: avoid inverting matrices when possible).

d) Assume all the resistors in the line have resistance of one ohm. Determine, as a function of N, the smallest entry in the associated resistance matrix.

2) Prove or find a counter-example to each of the following statements about strictly diagonally dominant tridiagonal (SDDT) matrices. An SDDT matrix is such that $A_{i,j} = 0, |i-j| > 1$ and $|A_{i,i}| > \sum_{j \neq i} |A_{i,j}|$.

a) The magnitude of the pivots used during LU factorization of a SDDT matrix are NEVER larger than the maximum magnitude of the diagonal entries of the original matrix A.

b) The magnitude of the pivots used during LU factorization of a SDDT matrix with postive diagonals and negative off-diagonals are NEVER larger than the maximum magnitude of the diagonal entries of the original matrix A.

3) In this problem you will derive an algorithm for inverting a matrix "in place", that is, without additional storage.

Suppose an $n \times n$ matrix is given by

$$\boldsymbol{N}(\boldsymbol{y}, k) = \boldsymbol{I} + \boldsymbol{y}\boldsymbol{e}_k^T$$

where $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{y}$ is an n-length vector and $\boldsymbol{e}_k$ is the n-length unit vector (that is, all $\boldsymbol{e}_k$ entries are zero except for the k-th, which is one).

a) Give a formula for the inverse of $\boldsymbol{N}(\boldsymbol{y}, k)$, assuming the inverse exists.

b) Use part (a) to determine $\boldsymbol{y}$ as a function of $\boldsymbol{x}$ such that

$$\boldsymbol{N}(\boldsymbol{y}, k)\boldsymbol{x} = \boldsymbol{e}_k$$

c) Use part (b) to develop an algorithm for computing the inverse of any given matrix, assuming the inverse exists. Implement the algorithm and compare its performance to that of matlab's built-in matrix inversion routine. In particular, assuming your computer had infinite memory, how large a matrix could it invert in a week using matlab's and your routines? How about in a year?

d) *Optional:* Try implementing your algorithm in an efficient compiled language like C or Fortran. How does its run time compare to your matlab script?