

03/15/2005

Solution of Linear Systems: Sparse Direct Methods (pg. 1-5 of part 1 notes)

We'll think about these methods in terms of our problems:

Our usual FE problem: $A_h u_h = f_h$

or the eigenvalue problem $A_h \phi_j = \lambda_j \phi_j$

But these SD methods work for any linear system problem with a sparse matrix.

We solved our problem using the backslash operator “\” in Matlab. We want to know what's behind the “\” operator.

We will now think of general linear system: $Ax = b$ with A large and sparse.

Algorithms for solving this system:

Assumed background - (for more information: <http://www.cs.umd.edu/~oleary/c460/old/> or Van Loan or Moler)

1) Gauss Elimination and the LU decomposition ($A = LU$, L is lower triangular, U is upper triangular)

2) Partial pivoting for stability

3) Cholesky – Used for symmetric positive-definite matrices. In our case, elliptic self-adjoint matrices work.

Factor: $A = LL^T$ where L is lower triangular

4) Forward and backward substitution

Direct vs. Indirect methods

Direct methods: LU, Gauss-elimination, Cholesky

Direct	Indirect
Method of choice 2-d problems	Method of choice 3-d problems
Exact solution (except round-off error)	Approximate soln. with given tolerance
More storage than original (maybe much more)	Only a few extra vectors of storage
Elements of A are modified by algorithm	Only need function that can form Ax given x
If you change b, easy to solve new system	Not easy to solve new system
Good software exists	Software is incomplete and hard to use

Storage:

Matlab stores matrices in a column oriented manner. To store sparse matrices it stores the indices of the nonzero elements in the matrix and the element for that index. It does so going down the columns:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 5 & 7 & 0 \\ 1 & 0 & 6 & 0 \\ 0 & 0 & 0 & 8 \end{bmatrix} \text{ is stored as } \begin{array}{l} (1,1) \ 2 \\ (3,1) \ 1 \\ (2,2) \ 5 \\ (2,3) \ 7 \\ (3,3) \ 6 \\ (4,4) \ 8 \end{array}$$

nz = the number of nonzero elements in the matrix, so there are $3nz$ or 18 storage locations used. If you were using C or another row-oriented language you would store by going across the rows instead.

Sparse Direct Methods:

Run demo in Matlab, “demo” then choose “Matrices”, “Sparse Matrices”

Or

“demo”, “Matrices”, “Orderings and Separators for a Finite Element Matrix”

Back to the problem:

$$Ax = b$$

A is symmetric and positive definite

So we can use Cholesky. We can scramble the rows and columns and the matrix will still be symmetric and positive definite and Cholesky will still work. This is not true if A is not symmetric.

Fill-in:

When you want to solve $Ax = b$ and A is a sparse matrix, you don’t want your method to change zeros to nonzeros when solving. Changing zeros to nonzeros using a solving method is the problem called fill-in.

Look at the arrowhead problem on page 3 of the notes:

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & 0 & 0 & 0 & 0 \\ \times & 0 & \times & 0 & 0 & 0 \\ \times & 0 & 0 & \times & 0 & 0 \\ \times & 0 & 0 & 0 & \times & 0 \\ \times & 0 & 0 & 0 & 0 & \times \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

Here using Gauss-elimination causes a fill-in problem because the zeros in the matrix are turned into nonzeros as we try to make the first column have zeros.

Fixing fill-in:

Scramble the problem above so that we come up with the new problem on the bottom of page 3:

$$A = \begin{bmatrix} \times & 0 & 0 & 0 & 0 & \times \\ 0 & \times & 0 & 0 & 0 & \times \\ 0 & 0 & \times & 0 & 0 & \times \\ 0 & 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ \times & \times & \times & \times & \times & \times \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_1 \end{bmatrix}$$

This is done by moving the top row to the bottom and the first column to the last column.

This is the same problem, just reordered. And now, when you perform Gauss-elimination you don't have the fill-in problem.

Note: remember to unscramble the answer at the end

Reordering the problem saves a lot of work.

Strategies for overcoming fill-in:

Reorder the rows and columns using a permutation matrix P and solve

$$PAP^T(Px) = Pb \quad \text{instead of} \quad Ax = b$$

When you multiply by P in front of A, you rearrange the rows, behind you rearrange the columns.

If we call A the original arrowhead matrix from the middle of page 3, then to get to the bottom of page 3 picture we have PAP^T where

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

When we multiply $P*A$ we move the first row to the last. When we multiply $A*P^T$ we move the first column to the last.

Note: PAP^T is symmetric if A is symmetric because –
 $(PAP^T)^T = (P^T)^T A^T P^T = PAP^T$

So we keep symmetry and can still use Cholesky on the permuted matrix.

Reordering:

It's too expensive to find the optimal reordering. Instead, use heuristics – cute algorithms that work on some problems. Sometimes heuristics cause bad re-orderings.

Important insights:

1) Band matrices – there is never any fill outside of the band.

2) Profile of the matrix – there is never any fill outside the profile:

To determine the profile of a matrix, find the first nonzero element in every column. Add to the profile every entry between these elements and the main diagonal. Then find the first nonzero element in every row. Add all elements between these nonzero entries and the main diagonal. The profile consists of all of these elements together. Below is a picture from the notes. All x and ⊗ elements are in the profile. Once again, there is never any fill outside the profile. (see below for picture from notes)

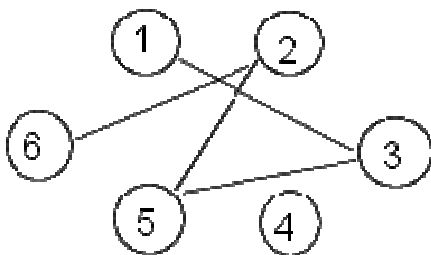
Example: Zeros within the profile marked with ⊗

$$A = \begin{bmatrix} \times & 0 & \times & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & \times & 0 \\ 0 & 0 & \times & 0 & \times & 0 \\ \times & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & \times & 0 & \times & 0 \\ 0 & \times & 0 & 0 & 0 & \times \end{bmatrix}, \text{ profile}(A) = \begin{bmatrix} \times & 0 & \times & 0 & 0 & 0 \\ 0 & \times & \otimes & 0 & \times & 0 \\ 0 & 0 & \times & 0 & \times & 0 \\ \times & \otimes & \otimes & \times & \otimes & 0 \\ 0 & 0 & \times & \otimes & \times & 0 \\ 0 & \times & \otimes & \otimes & \otimes & \times \end{bmatrix}$$

3) The sparsity of a matrix can be encoded in a graph. For example the sparsity of the symmetric matrix:

$$A = \begin{bmatrix} \times & 0 & \times & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & \times & \times \\ \times & 0 & \times & 0 & \times & 0 \\ 0 & 0 & 0 & \times & 0 & 0 \\ 0 & \times & \times & 0 & \times & 0 \\ 0 & \times & 0 & 0 & 0 & \times \end{bmatrix}$$

can be converted to the following graph by drawing an arc between the indices of any nonzero element:



Here the connections represent the coordinates of the nonzero elements in the matrix. Since it is symmetric, we do not need direction arrows, so this is an undirected graph. For instance, there is a link between 1 and 3 so we know the elements with indices (1, 3) and (3, 1) of the matrix are nonzero. We know the diagonal elements are nonzero.

Reordering strategies:

Cuthill-McKee:

“degree of a node in a graph” = the number of edges touching the node

Strategy:

- 1) Start at the node with minimum degree. If not unique, choose one with minimum degree. Number this 1.
- 2) Find neighbors of node 1 and number them in order of least degree, again if not unique, you can pick the nodes with the same degree in any order.
- 3) Repeat 2 for the new nodes.
- 4) Continue ordering this way until all nodes are ordered.

Ex (the start of ordering the grid):

