Sparse Matrix Solvers

APPM 4600 Project

University of Colorado Boulder
Department of Applied Mathematics

1 Project Summary

Many applications, such as constructing cubic splines and solving differential equations using the finite element method, result in a system of linear equations that can be represented by a sparse matrix. A matrix $\mathbf{A}$ is called sparse if zero entries *significantly* out number the non-zero entries. For example, the matrix that needs to be inverted to create cubic splines is tridiagonal, and thus is sparse for a large number (i.e. $>> 3$) of interpolation nodes. In this project, you will derive an inversion technique for tridiagonal systems whose computational cost scales linearly with respect to the size of the system. This algorithm is called the Thomas algorithm. Next, you will investigate inversion techniques for general sparse linear systems; i.e. sparse systems where you do not know apriori the sparisty pattern. These solvers are LU solvers which pick pivots in a manner such that the sparsity of the factorization is maximized. These methods are call nested dissection or multi-frontal methods. You will explore some of the different pivoting techniques and investigate the stability of these methods and where they are used in practice.

2 Project Background

2.1 Motivation - Tridiagonal Matricies and the Thomas Algorithm

Recall that the computational cost of Gaussian elmination on a linear system of size N is $O(N^3)$. For many applications the system that needs to be inverted is tens of thousands (or larger) in size, thus a direct solver is often too expensive to be useful. However, when the system is sparse, there exist direct solution techniques that are more efficient than Gaussian elimination.

One of the most simple examples of a sparse matrix is a tridiagonal matrix. The general form of a tridiagonal system is given by

$$\mathbf{Ax} = \begin{bmatrix} b_0 & c_0 & 0 & \dots & 0 \\ a_1 & b_1 & c_1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & a_{n-2} & b_{n-2} & c_{n-2} \\ 0 & \dots & 0 & a_{n-1} & b_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \mathbf{y}. \tag{2.1}$$

where $\{a_j\}_{j=1}^{n-1}$, $\{b_j\}_{j=0}^{n-1}$, and $\{c_j\}_{j=0}^{n}-2$ are the real valued entries of the $n \times n$ matrix $\mathbf{A}$. The a_j and c_j values lie in the lower and upper diagonal of the matrix while the b_j values lie on the diagonal. These are the only entries of the matrix that can have a value other than zero.

To gain intuition about how to exploit the sparse structure of the matrix $\mathbf{A}$ to cut down on complexity, consider the 5×5 case of a tridiagonal matrix system:

$$\mathbf{A}\mathbf{x} = \mathbf{y} \implies \left[\begin{array}{ccccc|c} b_0 & c_0 & 0 & 0 & 0 & y_0 \\ a_1 & b_1 & c_1 & 0 & 0 & y_1 \\ 0 & a_2 & b_2 & c_2 & 0 & y_2 \\ 0 & 0 & a_3 & b_3 & c_3 & y_3 \\ 0 & 0 & 0 & a_4 & b_4 & y_4 \end{array}\right] \tag{2.2}$$

The Gaussian elimination approach for solving this linear system starts with eliminating the a_1 term in the second row. This results in the following operation:

$$\left[\begin{array}{ccccc|c} b_0 & c_0 & 0 & 0 & 0 & y_0 \\ a_1 & b_1 & c_1 & 0 & 0 & y_1 \\ 0 & a_2 & b_2 & c_2 & 0 & y_2 \\ 0 & 0 & a_3 & b_3 & c_3 & y_3 \\ 0 & 0 & 0 & a_4 & b_4 & y_4 \end{array}\right] \xrightarrow{\frac{a_1}{b_0}R_1 - R_2} \left[\begin{array}{ccccc|c} b_0 & c_0 & 0 & 0 & 0 & y_0 \\ 0 & b_1 - \frac{c_0 a_1}{b_0} & c_1 & 0 & 0 & y_1 - \frac{y_0 a_1}{b_0} \\ 0 & a_2 & b_2 & c_2 & 0 & y_2 \\ 0 & 0 & a_3 & b_3 & c_3 & y_3 \\ 0 & 0 & 0 & a_4 & b_4 & y_4 \end{array}\right] \tag{2.3}$$

Note that only three of the six entries in the above system were modified by the Gaussian operations, although six multiplications and six subtractions were performed. Since there are zero entries involved with this step, the same computation can be performed with half the operations. Now imagine a much larger system, for example $N = 10,000$. Then the number of zero entries that are operated on is large and it dominates the cost of one step in Gaussian elimination.

By eliminating the operations involving zero entries, a more efficient direct solver called the *Thomas algorithm* can be derived. This solver scales linearly with the size of the system.

The idea behind the Thomas Algorithm is to modify the non-zero entries of $\mathbf{A}$ using row operations to change our linear system from the form shown below on the left to the form shown on the right.

$$\left[\begin{array}{ccccc|c} b_0 & c_0 & 0 & 0 & 0 & y_0 \\ a_1 & b_1 & c_1 & 0 & 0 & y_1 \\ 0 & a_2 & b_2 & c_2 & 0 & y_2 \\ 0 & 0 & a_3 & b_3 & c_3 & y_3 \\ 0 & 0 & 0 & a_4 & b_4 & y_4 \end{array}\right] \implies \text{Row Operations} \implies \left[\begin{array}{ccccc|c} 1 & c'_0 & 0 & 0 & 0 & y'_0 \\ 0 & 1 & c'_1 & 0 & 0 & y'_1 \\ 0 & 0 & 1 & c'_2 & 0 & y'_2 \\ 0 & 0 & 0 & 1 & c'_3 & y'_3 \\ 0 & 0 & 0 & 0 & 1 & y'_4 \end{array}\right] \tag{2.4}$$

Then the form on the right can be solved using back substitution to find x_i for $i = 0, \ldots, 4$.

2.1.1 Questions to Investigate

1. Solve a general 3×3, 4×4, and 5×5 tridiagonal matrix using Gaussian elimination. Are there any patterns in the solutions?

2. Using the patterns observed in question 2.1.1.1 derive formulas for the values c_i', y_i', and x_i from equation 2.4. Does this method provide an exact solution to the tridiagonal system?

3. Compare the performance of the Thomas algorithm with Gaussian elimination for matrices that are 3×3, 10×10, 100×100, etc. You should code up your own Thomas solver but can use the LU factorization provided by calling lu() in matlab or scipy.linalg.lu() in python. For what size matrices is the Thomas algorithm faster than GE? Verify that the time cost of the Thomas Algorithm scales linearly.

4. What happens to the Thomas algorithm if the first entry in the matrix **A** is 0; i.e. $b_0 = 0$? Is it possible to correct any problems that arise?

5. Can the Thomas algorithm be applied to a singular tridiagonal matrix **A**? What in the Thomas algorithm would indicate that the matrix is singular?

6. The Thomas algorithm is known to to be unstable for certain values of matrix entries $\{a_j\}_{j=0}^N$, $\{b_j\}_{j=0}^N$ and $\{c_j\}_{j=0}^N$. What are the conditions on these constants which make the Thomas algorithm unstable? When the thomas algorithm is unstable, how else can the linear system be solved?

2.2 The Multi-frontal Method

A key concept in understanding the theory of sparse solvers is the idea of matrix *connectivity.* To understand connectivity consider the 4×4 matrix below:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}. \tag{2.5}$$

If row 1 is added to rows 2 or 4, then a non-zero entry in that row would be modified, while adding row 3 to any other row does not effect any of the non zero entries. Because of these properties, row 1 is said to be connected to rows 2 and 4, while row three is disconnected from the rest of the matrix. The following definition rigorously defines connectivity of row i to row j:

Definition 2.1. For $i \neq j$, the i^{th} and j^{th} rows of a symmetric matrix $\mathbf{A}$ are said to be *connected* if $a_{ij} \neq 0$.

Graph theory can be a useful tool to understand these types of structure. Figure 2.1 depicts a graph constructed to represent the connectivity of the matrix from equation 2.5.

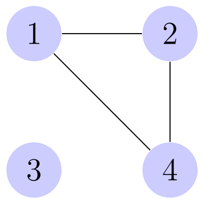


Figure 2.1: The connectivity graph of 2.5

As Figure 2.1 shows, the first, second and fourth row are all connected while the third row is unconnected. This suggest applying two pivots to $\mathbf{A}$, exchanging the 1^{st} and 3^{rd} columns and rows of the matrix, resulting in the form

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{2.6}$$

If A is the coefficient matrix for a system of equations, the system can be solved with $\mathcal{O}(4^3)$ complexity by inverting the 4×4 matrix system. However, equation 2.6 can be rewritten as

$$\begin{bmatrix} \mathbf{M_{1\times1}} & 0 \\ 0 & \mathbf{M_{3\times3}} \end{bmatrix} \tag{2.7}$$

where

$$\mathbf{M_{1\times1}} = \begin{bmatrix}1\end{bmatrix}, \qquad \mathbf{M_{3\times3}} = \begin{bmatrix}1 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 1\end{bmatrix}. \tag{2.8}$$

Since $\mathbf{M_{1\times1}}$ is not connected to $\mathbf{M_{3\times3}}$, $\mathbf{A^{-1}}$ can be expressed as

$$\mathbf{A^{-1}} = \begin{bmatrix}\mathbf{M_{1\times1}^{-1}} & 0\\ 0 & \mathbf{M_{3\times3}^{-1}}\end{bmatrix} \tag{2.9}$$

which requires $\mathcal{O}(3^3) + \mathcal{O}(1^3) = O(3^3)$ complexity to solve. In other words, by understanding the underlying connectivity of the matrix, an algorithm was created to reduce the computational cost.

Recall that the process of Gaussian elmination is the same as constructing the LU factorization. The advantage of the LU factorization is that once the factors are constructed, the solution to the linear system can be found for a reduced computational cost.

When the connectivity based pivoting is used to construct the LU factorization, there is minimal fill-in in the L and U factors. In other words, the L and U factors will be sparse. Thus solving the linear system via the LU factorization will be less expensive than using the LU factorization from the not reordered system. The sparsity of the L and U factors will be illustrated later in this section.

The heart of the nested dissection algorithm is to construct a LU factorization with sparse factors. Let $\mathbf{S}$ denote a sparse matrix. The basic idea of the nested dissection algorithm is to determine an optimal permutation of the rows and columns of $\mathbf{S}$ so the sparsity of the the resulting LU factorization is maximized. By doing this, each independent part of the graph can be solved in parallel. It can then be synthesized together to build the final solution to the linear system. Another, perhaps more intuitive explanation is the nested dissection algorithm provides a method for determining the ordering of a system of equations such that the LU factorization will be optimally sparse.

To demonstrate the nested dissection method, consider the matrix shown below:

$$\mathbf{A} = \begin{bmatrix}1 & 1 & 0 & 1 & 0\\ 1 & 1 & 1 & 0 & 1\\ 0 & 1 & 1 & 1 & 0\\ 1 & 0 & 1 & 1 & 1\\ 0 & 1 & 0 & 1 & 1\end{bmatrix}. \tag{2.10}$$

This is known as a banded matrix structure. Matrices of this form arise in many different applications including approximating the solution to differential equations. If one tries the approach of the Thomas algorithm, which is to try to avoid touch zero entries in the matrix in the elimination procedure, the zero entries will start of fill in. For example, subtracting row 1 from row 2 results in $a_{2,4} = -1$ instead of preserving $a_{2,4} = 0$. Meanwhile, if rows 2 and 3 are swapped no elimination is required to find the LU factorization of A.

Consider the connectivity graph for the matrix A in Figure 2.2.

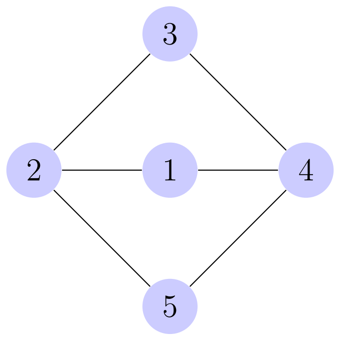


Figure 2.2: The connectivity graph for the matrix 2.10

To minimize the amount of operations needed to construct the LU factorization, the matrix should be ordered to minimize the connections between the first node and second node. This is because since row 1 is attached to row 2, an operation involving both of them needs to be performed in order to build the LU factorization. Instead the nodes can be re-ordered a illustrated in Figure 2.3.

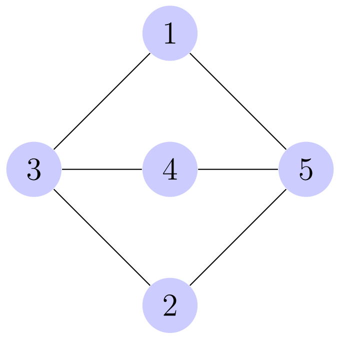


Figure 2.3: The connectivity graph of the reordered system 2.11

Notice that now rows 1 and 2 only have to two connections each. In the original ordering row 2 had three connections. The permuted **A** is given by

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{2.11}$$

As expected, operations between rows 1 and 2 are no longer needed to create the LU factorization. However, the matrix is not fully reordered to have optimal sparsity in the L and U factors yet. At the beginning, it was stated that the goal is to minimize the connections between nodes corresponding to the uppermost rows, but the third node (corresponding to the third row) has 3 connections while the forth node only has 2 connections. Fortunately, this can be adjusted by applying bissection to the graph and relabeling the nodes as illusrated in Figure 2.4.

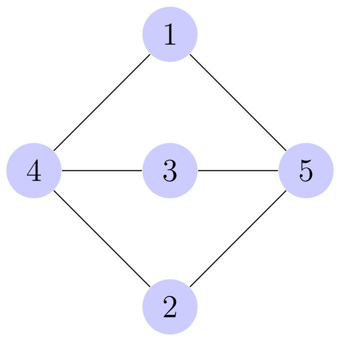


Figure 2.4: The connectivity graph of the twice reordered matrix 2.12

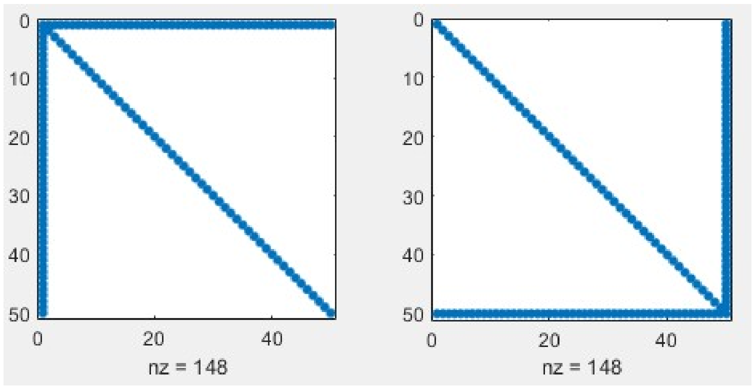
Notice that the connectivity of the nodes increases with the number label of the node. Nodes 1-3 each have two connections, while nodes 4 and 5 have three connections. This corresponds with a final matrix form of

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{2.12}$$

Under this new arragnement of the nodes, the permuted **A** has a downward arrow structure. This structure is known to produce a sparse LU factorization, which allows for efficient solution to a corresponding system of equations.

2.2.1 Questions to consider

1. One method for solving a matrix system efficiently on a machine is with an LU factorization, which can cut the computational time of a matrix solve down to $O(N^2)$. Calculate and visualize (using the spy() command) the LU factorization of two large matrices ($N = 100$ should do it) with the following sparsity structures:



where nz is the number of nonzero entries in the given matrices.

2. Of the two matrix structures in the previous question, which has the more desirable LU factorization? Why does this matter?

3. The method outlined in the background is just one procedure for selecting how to pivot a matrix to optimize the LU factorization. Explore some of the methods outlined in different software packages. How do they select their pivots? Are some methods more effective than others? Some sparse matrices to test these techniques on can be found at `https://sparse.tamu.edu/about`.

4. What happens to the nested dissection method when the matrix being permuted is not symmetric? Can the method still be applied?

5. What happens if the connectivity graph is disjoint? Does this pose an issue for this method?

3 Software Expectations

You may use software packages for this project. You should implement your own Thomas algorithm. You can find some sparsity structures that arise in applications in the Sparse Matrix Library: `https://sparse.tamu.edu/about` to test the performance of your solvers.

4 Possible Independent Directions

1. Apply the sparse solvers to solving differential equations by the method of finite differences. Possible problems to approach are the 1 or 2D-Poisson equation.

2. What different methods exist for solving other sparsity structures?

3. How does the performance of a direct sparse solver compare to that of an iterative solver like the Jacobi method, conjugate gradient method, or GMRES? **Note** : If you

choose this option, please employ software to test the iterative solver. Your paper is on sparse solvers so you do not need to spend your time building the codes for an iterative solve.

4. There exists a class of solvers known as "banded solvers" that are known to be more efficient than sparse solvers when applied to banded matrices. Explore what these banded solvers do and why they are more efficient in certain cases.

5. Explore how parallel computing can be employed to further improve the performance of nested dissection.

6. Explore the incomplete LU factorization and how it performs against the other methods presented in this project.

5 Helpful Sources

1. W. T. Lee, "Tridiagonal Matrices: Thomas Algorithm." University of Limerick.

   `http://www.industrial-maths.com/ms6021_thomas.pdf`

2. M. S. Khaira, G. L. Miller, T. J. Sheffler. "Nested Dissection: A su rvey and comparison of various nested dissection algorithms." Carnegie Mellon University (1992).

   `https://www.cs.cmu.edu/~glmiller/Publications/CMU-CS-92-106R.pdf`

3. Golub, Van Loan, Matrix Computations, Chapter 11, Large Sparse Linear System Problems.

4. Latex graph/nodes help: `https://www.javatpoint.com/latex-node-graphs-using-tikz`

5. Sparse Matrix Matlab tutorial and Sparse matrix Scipy tutorial

6. Sparse Matrix Library: `https://sparse.tamu.edu/about`