Project 12: Sparse Matrix Solvers

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 the number of zero entries *significantly* out number of non-zero entries. For example, the matrix that needs to be inverted to create cubic splines is tridiagonal. Thus for a large number (i.e. $>> 3$) interpolation nodes, the matrix that needs to be inverted is sparse. 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 a LU solvers which pick pivots in a manner such that the fill in of the factorization is minimized. These methods are call nested dissection or multi-frontal methods. You will explore some of the different pivoting techniques and investigate where these methods are used in practice and their stability.

2 Project Background

2.1 Motivation - Tridiagonal Matricies and the Thomas Algorithm

As has been discussed in class, the computational cost of using Gaussian elmination to solve 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 ofteen seen as too expensive to be useful. To see the impact of this cost, consider solving a system of 1000 equations would need 10^9 operations to fully solve. When the system that needs to solved is sparse there exists direct solution techniques that are more efficient than Gaussian elimination.

One of the most simple examples of a sparse matrix is a tridiagonal matrix. An example of this system is given as

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

Thus $\mathbf{A}$ is a matrix whose entries along the three main diagonals are mostly non-zero, while all other entries of the matrix are zero. To gain some intuition about how we can exploit the sparse structure of the matrix $\mathbf{A}$ to cut down on complexity, consider the 5×5 case of an augmented tridiagonal matrix system:

$$\mathbf{A}\mathbf{x} = \mathbf{y} \implies \left[\begin{array}{ccccc|c} b_0 & c_0 & 0 & 0 & 0 & y_0 \\ a_1 & b_1 & c_1 & 0 & 0 & y_1 \\ 0 & a_2 & b_2 & c_2 & 0 & y_2 \\ 0 & 0 & a_3 & b_3 & c_3 & y_3 \\ 0 & 0 & 0 & a_4 & b_4 & y_4 \end{array}\right] \tag{2.1}$$

Up to this point the way that we would go about solving this system would be to apply Gaussian elimination to the first row of the matrix to eliminate the a_1 term in the second row. This would result in the following operation

$$\begin{array}{rc} & \frac{a_1}{b_0} \times \begin{bmatrix} b_0 & c_0 & 0 & 0 & 0 & y_0 \end{bmatrix} \\ - & \begin{bmatrix} a_1 & b_1 & c_1 & 0 & 0 & y_1 \end{bmatrix} \\ \hline & \begin{bmatrix} 0 & b_1 - \frac{c_0 a_1}{b_0} & c_1 & 0 & 0 & y_1 - \frac{y_0 a_1}{b_0} \end{bmatrix} \end{array}$$

As you may notice, only 3 of the six entries in the above system were actually modified by the Gaussian operations, although six multiplications and six subtractions were performed. Since there are zeros entries involved with these operations, they were completely unnecessary and we could have performed the step of Gaussian elimination with half the operations. Now imagine a much larger system. For example, let $N = 10,000$. Then the number of zero entries that are operated is large and it dominates the cost of one step in Gaussian elimination.

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

In short, we want 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.

$$\left[\begin{array}{ccccc|c} b_0 & c_0 & 0 & 0 & 0 & y_0 \\ a_1 & b_1 & c_1 & 0 & 0 & y_1 \\ 0 & a_2 & b_2 & c_2 & 0 & y_2 \\ 0 & 0 & a_3 & b_3 & c_3 & y_3 \\ 0 & 0 & 0 & a_4 & b_4 & y_4 \end{array}\right] \implies \text{Row Operations} \implies \left[\begin{array}{ccccc|c} 1 & c_0' & 0 & 0 & 0 & y_0* \\ 0 & 1 & c_1' & 0 & 0 & y_1* \\ 0 & 0 & 1 & c_2' & 0 & y_2* \\ 0 & 0 & 0 & 1 & c_3' & y_3* \\ 0 & 0 & 0 & 0 & 1 & y_4* \end{array}\right] \tag{2.2}$$

Once we have rewrtitten our system in the form on the right it can then be solved using back substitution to find each of the x_i solutions.

2.1.1 Questions to Investigate

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

2. Using the patterns you observed in question 2.1.1.1 derive formulas for the values c'_i, y_i*, and x_i from equations (2.1) and (2.2). 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 and 100×100. 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?

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 you apply the Thomas algorithm to a singular tridiagonal matrix **A**? How will you know in the Thomas algorithm that you have encountered a singular matrix?

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 we solve our linear system?

2.2 The Multi-frontal Method

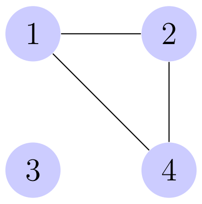
A key concept in understanding the theory of sparse solvers is the idea of matrix *connectivity.* To understand connectivity consider the 4×4 matrix below

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

As can be seen above, 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. The i^{th} and j^{th} rows of a sparse, symmetric matrix $\mathbf{A}$ are said to be *connected* if the off-diagonal entry $a_{ij} \neq 0$

Conveniently, pure mathematicians happen to have developed a tool for understanding these types of structures, graph theory. Employing this deep and beautiful field, we can construct a graph to represent our matrix from equation (2.3).



As the graph above shows, the first, second and fourth row are all connected while the third row is unconnected. With this understanding we can apply two pivots to $\mathbf{A}$, exchanging the 1^{st} and 3^{rd} columns and rows of the matrix, to give the form

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.4}$$

if A us the coefficient matrix for a system of equations, we could solve the system, by inverting the 4×4 matrix system, which would require $\mathcal{O}(4^3)$ complexity. Meanwhile, an astute reader may notice that (2.4) can be rewritten as

$$\begin{bmatrix} \mathbf{M_{1\times1}} & 0 \\ 0 & \mathbf{M_{3\times3}} \end{bmatrix} \tag{2.5}$$

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

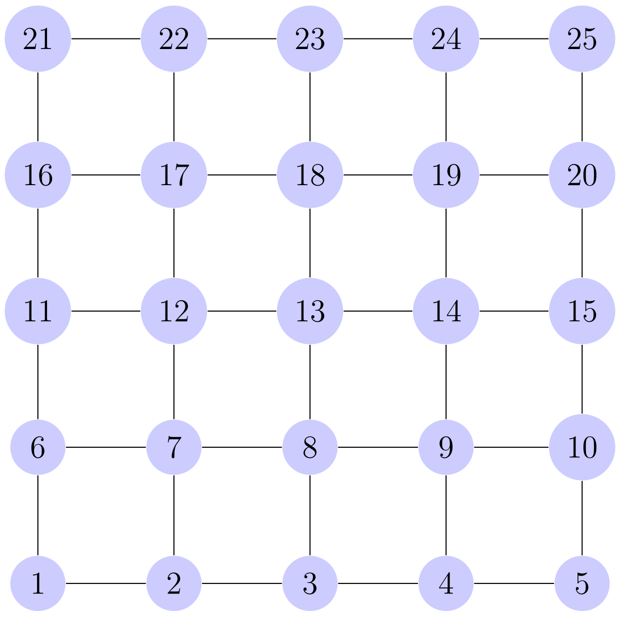
$$\begin{bmatrix} \mathbf{M}_{\mathbf{1\times1}}^{\mathbf{-1}} & 0 \\ 0 & \mathbf{M}_{\mathbf{3\times3}}^{\mathbf{-1}} \end{bmatrix} \tag{2.6}$$

Which would require $\mathcal{O}(3^3)+\mathcal{O}(1^3) = O(3^3)$ complexity. In other words, by understanding the underlying connectivity of our matrix, we are able to cut down on computational costs.

While Gaussian elimination provides one method for solving a linear system, it is often desirable to use the LU decomposition to further improve computational time. As it turns out, for a given matrix $\mathbf{A}$, if we apply the same method outlined above to rearrange the rows and columns of $\mathbf{A}$ to group connected sets, the resulting LU factorization will itself be sparse, which can further optimize the LU factorization for solving a linear system. This will be demonstrated later in this section.

This is the heart of the nested dissection algorithm. The basic idea of the algorithm is to determine an optimal permutation of the rows and columns of a sparse matrix S_0 such that we are able to maximize the sparsity of the resulting LU factorization. By doing this each branch of the graph can be solved in parallel, then 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 variable and equation order for a system of equations such that the LU factorization will be optimally sparse.

To demonstrate the nested dissection method, consider the connectivity graph shown below:



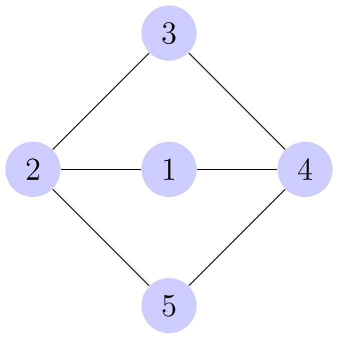
This graph corresponds to a 25×25 matrix with a structure similar to the 5×5 matrix

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{2.7}$$

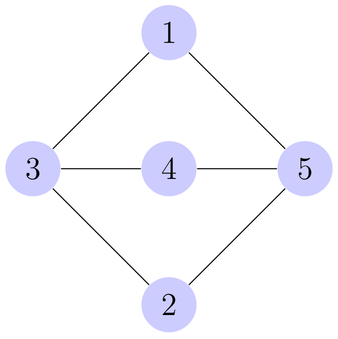
.

This is known as a banded matrix structure. Matrices of this form arise in many different applications, such as solving differential equations numerically. As we observed with the Thomas algorithm, we can try to efficiently solve our matrix by ignoring the zero entries of our matrix, but we will find that some of the zero entries will get filled in as we try to perform the elimination (In subtracting row 1 from row 2 for example). Meanwhile, if rows 2 and 3 were swapped no such elimination would be required to find the LU factorization of **A**.

Consider the connectivity graph for the matrix A.



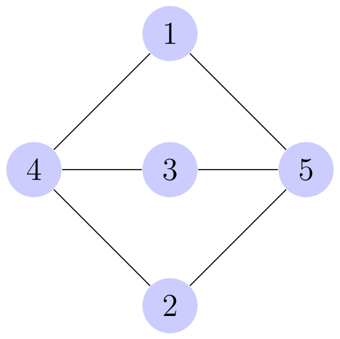
To minimize the amount of operations we need to perform to build our LU factorization we want to minimize the connections between our first nodes to each other (since row 1 is attached to row 2 we need to perform an operation between the two to build the LU factorization). To do this we can instead number the nodes as follows:



Notice that now rows one and 2 only have to two connections each, where in the original form row 2 had three! Returning to the matrix form **A** would now be written as

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{2.8}$$

As expected, we no longer need to perform an operation between rows 1 and 2 since they are no longer connected. However, we have not fully optimized our matrix. At the beginning we said that we wanted to minimize the connections between nodes corresponding to the variables with low index numbers, but the third node (corresponding to the third variable) has 3 connections, which is more than the 2 that the forth node has. Fortunately, we can simply apply another bisection to the graph, relabeling the nodes as follows.



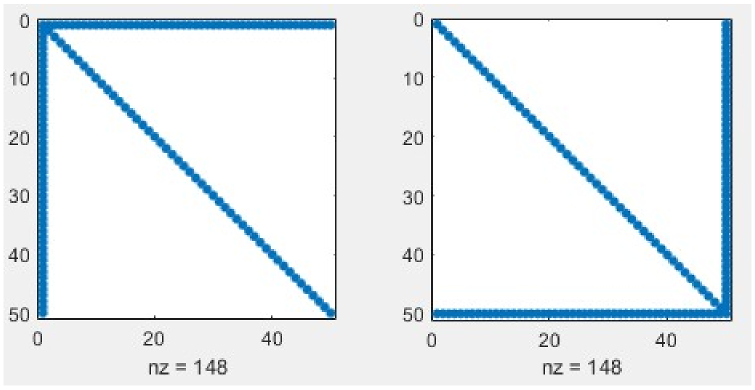
Notice how now we have labeled our nodes in descending order with the number of connections they have. Nodes 1-3 have 2 connections each, while the final two nodes, 4 and 5, have 3. This corresponds with a final matrix form of

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2.9}$$

Under this new arragnement of the nodes of $\mathbf{A}$, the resulting sparse matrix 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



2. Of the two matrix structures you explored for 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 some software packages, how do they select their pivots? Are some methods more effective than others? You can find some sparse matrices to test these techniques on at https://sparse.tamu.edu/abou.

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

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/abou to test the performance of your solvers.

4 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 other method 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. Another class of matrix is a banded matrix. Banded Matrices are known to solve faster when a banded solver is applied to the system than a sparse solver. Explore why this is and what these solvers do that allows them to solve another variety of problem.

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

2. https://www.javatpoint.com/latex-node-graphs-using-tikz

3. https://www.youtube.com/watch?v=r1P4Ze7Yqe0

4. https://www.youtube.com/watch?v=mwX0wPRdw7U

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

6. The Thomas algorithm: an overview

7. Sparse Matrix Matlab tutorial and Sparse matrix Scipy tutorial

8. Lecture: Sparse-Direct Solvers

9. Sparse Matrix Library: https://sparse.tamu.edu/abou

6 Appendix: Matrix Data management