

Applying Sparse Matrix Solvers to a Glacial Ice Sheet Model

Rodney Jacobs

Computer Science Department
University of Maine
Orono, ME, USA

James Fastook

Professor of Computer Science
University of Maine
Orono, ME, USA

Aitbala Sargent

Computer Science Department
University of Maine
Orono, ME, USA

***Abstract** - Two software packages for solving sparse systems of linear equations, SuperLU and UMFPACK, have been integrated with the University of Maine Ice Sheet Model for predicting the formation and disappearance of glacial ice sheets. Using a library of basic linear algebra subprograms, BLAS, tuned for the underlying hardware, these packages perform significantly better than our banded Gaussian elimination routine. Also, they are able to solve non-banded systems that can be produced by the ice sheet model, but are impractical to solve with straightforward Gaussian elimination. A modified compressed column data structure is presented for interfacing the ice sheet model with the two software packages. Test results are presented that indicate careful consideration must be given to the column ordering methods used by the packages when solving specific problems.*

Keywords: Sparse matrix solvers; SuperLU; UMFPACK

1 Background

The University of Maine Ice Sheet Model (UMISM) is a mathematical model for the formation and disappearance of glacial ice sheets. The model, developed by James Fastook, has been applied to the Antarctica, Greenland, Scandinavian, and Laurentide ice sheets as well as tropical mountain glaciers on the flanks of the Tharsis Montes Volcanos of Mars. The model uses conservation of mass, energy and momentum, as a well as constitutive relations between ice stress versus ice strain rates and energy versus temperature to form partial differential equations describing ice sheet thickness and velocity as functions of time and position. The PDE's are solved numerically using the finite element method (FEM) [6]. Early models were done on a 2-dimensional map plane with a rectangular FEM grid. These models generate systems of simultaneous linear equations that are sparse and diagonally dominant. Systems of 40,000 equations are common. The equations are efficiently solved using iterative methods and matrix storage requirements are limited to the 18 non-zero coefficients per equation generated by FEM.

More recent models have attempted to use better physics to describe ice velocities in regions where velocities vary significantly over short distances. These models use a 3-dimensional, rectangular FEM grid. While the 2-dimensional model is used for an entire ice sheet, the 3-dimensional model is used over a limited area of interest with a smaller distance between grid points. Output of the 2-dimensional model is used to establish boundary conditions for the 3-dimensional model. When solving for 3D velocities, FEM generates systems of banded linear equations with 81 non-zero coefficients per equation. Unlike the 2D model, these equations are not diagonally dominant. For this reason, the decision was made to investigate direct methods based on Gaussian elimination for solving these systems of equations.

An additional complication in the 3D model is internal pressure within the ice sheet. In one version of the model the internal pressure is eliminated, resulting in a system of linear equations that is purely banded. In a second version of the 3D model, pressure is explicitly calculated and results in a banded system of equations with lower and right borders. Figure 1 depicts the non-zero entries in these systems of equations.

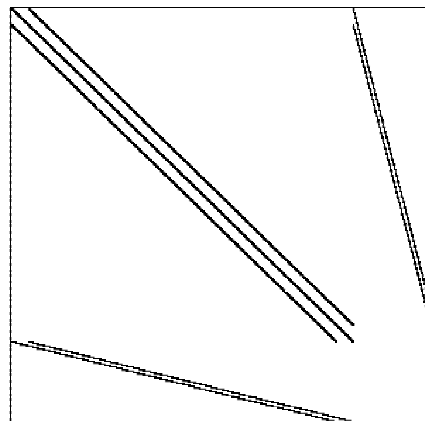


Figure 1. Non-zero entries for 3D model with explicit pressure calculations.

Initially, banded Gaussian elimination was used to solve 3D problems without pressure. However, bandwidths can be relatively large (~1,200 columns for 24,000 equations) and the method does not work for 3D problems with pressure. For these reasons the decision was made to investigate other direct solvers. Our goal was to find available software that was fast, had low memory overhead, and could be integrated into the ice sheet model [7].

2 Software selection

Two freely available direct solvers for sparse systems of equations were selected for testing with the 3D ice sheet model. The first, SuperLU [3, 4, 8], is available from the Lawrence Berkeley National Laboratory. The second, UMFPACK [1, 2], is available from the University of Florida. A catalog of freely available software for linear algebra is maintained by Jack Dongarra and can be found at <http://www.netlib.org/utk/people/JackDongarra/la-sw.html>.

SuperLU is a library of ANSI C subroutines for performing LU factorization. It comes in three versions: sequential SuperLU for uniprocessors, SuperLU-MT for shared memory multiprocessors, and SuperLU-DIST for distributed memory multiprocessors. For this work we only tested sequential SuperLU, but the multiprocessor versions open the possibility of migrating the ice sheet model to parallel computing. SuperLU uses row permutations to reduce round off errors and maintain stability. Except for SuperLU-DIST, SuperLU uses partial pivoting with a user-defined threshold to determine row permutations. It uses column permutations to reduce non-zero fill-ins in the L and U factors, thus reducing computation time and memory usage. Four column permutation algorithms are provided: natural column ordering; multiple minimum degree applied to $\mathbf{A}^T\mathbf{A}$, where \mathbf{A} is the coefficient matrix of the system of equations; multiple minimum degree applied to $\mathbf{A}^T+\mathbf{A}$; and column approximate minimum degree. An additional option allows the user to explicitly specify the column ordering through a user-supplied algorithm. SuperLU supports row-column equilibration, a method of scaling rows and columns of \mathbf{A} so that the infinity-norm of each row and column is between $[1/2, 1]$. Column equilibration may increase the accuracy of the solution. Once a solution has been determined, SuperLU can use double precision or extended precision iterative refinement to improve the accuracy of the solution.

SuperLU can optionally compute various error measures including the component-wise relative backward error, *BERR*; an estimate of the forward error bound, *FERR*; and an estimate of the matrix condition number of \mathbf{A} . If $\bar{\mathbf{x}}$ is the computed solution of the system of equations $\mathbf{Ax} = \mathbf{b}$, then $\bar{\mathbf{x}}$ is the exact solution of $\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}}$ where the

relative difference between corresponding elements of \mathbf{A} , $\bar{\mathbf{A}}$, \mathbf{b} , and $\bar{\mathbf{b}}$ is no larger than the *BERR*. *FERR* is the upper bound of the absolute value of the error in any component of $\bar{\mathbf{x}}$ divided by the maximum of the absolute values of the components of \mathbf{x} . The matrix condition number relates uncertainties in the values of the components of \mathbf{A} and \mathbf{b} to uncertainties in the components of $\bar{\mathbf{x}}$. If the condition number is large, the system of equations is ill-conditioned and small uncertainties in the component values of \mathbf{A} and \mathbf{b} can result in large uncertainties in the components of $\bar{\mathbf{x}}$.

UMFPACK is also a library of C subroutines for performing LU factorization. It is only available for uniprocessors. UMFPACK also uses partial pivoting to determine row permutations that reduce round off errors and maintain stability. UMFPACK can automatically select its column-ordering algorithm. Available ordering algorithms are column approximate minimum degree and approximate minimum degree. Instead of doing row-column equilibration, UMFPACK can optionally perform row scaling. Rows are scaled so that either the 1-norm of each row is 1 or the infinity-norm of each row is 1. Double precision iterative refinement is also available. Like SuperLU, UMFPACK can compute *BERR* and an estimate of the matrix condition number.

Both SuperLU and UMFPACK rely on a library of basic linear algebra subroutines, BLAS, tuned for the underlying hardware to deliver optimum performance. Davis, the author of UMFPACK, recommends using Kazushige Goto's BLAS available from the Texas Advanced Computing Center at the University of Texas. Goto's BLAS was used with both SuperLU and UMFPACK on an Intel IA32 processor in our testing.

3 Interfacing with the Ice Sheet Model

Compressed column format is a data format for \mathbf{A} that is compatible with both SuperLU and UMFPACK. Three one-dimensional arrays are used to store a matrix in this format. One array is used to store the non-zero entries of \mathbf{A} in column-row order. The second array is used to store the row number of each corresponding non-zero entry in the values array. The third array contains the index values of the first and second arrays where the first non-zero entry for each column of \mathbf{A} is stored. The matrix column number is the index to this array.

The FEM computation requires access to entries in \mathbf{A} by row and column number as well as the ability to insert new non-zero values in \mathbf{A} . Compressed column format allows efficient access to non-zero entries in \mathbf{A} by row and column numbers. The column number can be used to determine the range of index values in the row number and value arrays that contain non-zero entries for the specified column. Since entries in this index range are stored in

order by row number, a binary search can be used to quickly find the index value for a specific row. If the row number is located, the entry value can be read from the values array. If the row number is not found, the entry value must be zero. With this scheme the time to retrieve an entry by row and column number was less than 100 nanoseconds for matrices with 2.9 million non-zero entries.

However, inserting a new non-zero entry in a compressed column format data structure is costly. In order to maintain entries in column-row order, all entries in the row number and value arrays above the new entry must be moved and all entries in the column index array above the column number of the new entry must be updated. To alleviate this performance bottleneck, a modified compressed column format with supporting routines was developed. Our rectangular FEM grid allows us to compute the maximum number of non-zero entries per row of \mathbf{A} . This maximum amount of space is allocated for each column in the row number and value arrays. Instead of having a single column pointer array that points to the first entry for each column in the row number and value arrays, two column pointer arrays are used. The first array points to the first row number and value in each column and the second array points to the last row number and value in each column. The free space for each column in the row number and value arrays allows a new entry to be added by moving only entries in the column of the new entry. Once FEM has completed the computation of \mathbf{A} , the remaining free space is removed from the modified compressed column data structure by moving entries down in the row number and value arrays and updating the column pointers in the column pointer arrays. Once the free space has been squeezed out of the data structure, the starting column pointer array, the row numbers array, and the values array are in fact a compressed column format data structure that is compatible with SuperLU and UMFPACK. With a 40,000 row matrix, 2.9 million non-zero entries were inserted in an empty modified compressed column data structure in 0.35 seconds, or about 120 nanoseconds per entry. The squeeze operation to remove free space took an additional 0.05 seconds.

N	3		Bucket Size	3							
Column Start	1	4	7	-	-	...					
Column End	2	5	7	-	-	...					
Row	1	2	-	2	3	-	1	-	-	-	...
Value	2.5	10.1	-	5.3	7.8	-	-1.3	-	-	-	...

Figure 2. Modified compressed column format data structure before squeezing.

The ice sheet model requires performing the FEM calculation for each time step of the model. Once the row and column numbers of non-zero entries produced by FEM is determined in the first iteration, they remain constant in all subsequent iterations. After the first iteration is complete, the values array is zeroed and the column pointer and row number arrays are left unchanged. No new entries are added to the data structure in subsequent iterations, so there is no need for additional free space or the attendant squeeze operation.

The ice sheet model is written in Fortran 77. A C interface routine that is callable from the ice sheet model was written to invoke the needed SuperLU functionality to solve the system of equations. A similar routine was written to call UMFPACK. Both routines have the same calling parameters and conventions, so the two methods can be interchanged with ease. The routines allow user selectable options within each package to be specified as well as produce optional debugging traces and operational statistics. A third routine was written to dump the matrix \mathbf{A} and vector \mathbf{b} computed by the ice sheet model to a disk file. With sample systems of equations stored in disk files, simple programs could be written for testing SuperLU and UMFPACK without having to run the entire ice sheet model.

4 Test results

Five test matrices ranging in size from 1,500 rows to 24,000 rows were obtained from the ice sheet model with internal pressures eliminated. Four additional test matrices ranging in size from 1,824 rows to 16,864 were obtained with explicit pressures. These are the matrices with lower and right borders mentioned earlier. This set of nine matrices was used exclusively to produce the following test results.

As a basis for comparison, a hand-coded banded Gaussian elimination routine and a banded LU factorization routine were written in Fortran. These routines could solve systems with internal pressures eliminated. All computations were performed by Fortran code without the assistance of BLAS.

The end user statistics of interest in these tests are compute time, memory overhead and calculation stability. In addition to these statistics, we looked at the number of floating point operations performed and the number of non-zero entries in $L+U$. *BERR* was used as a measure of calculation stability. With iterative refinement, *BERR* was usually reduced to near machine precision. If iterative refinement did not reduce *BERR* to near machine precision, we assumed the calculation was not stable.

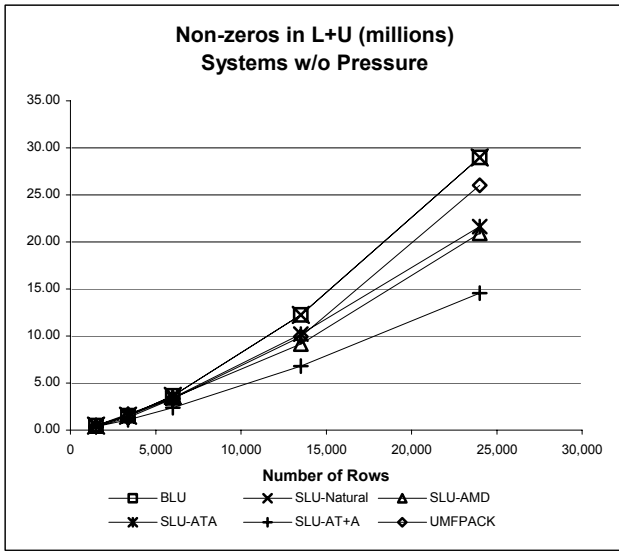


Figure 3.

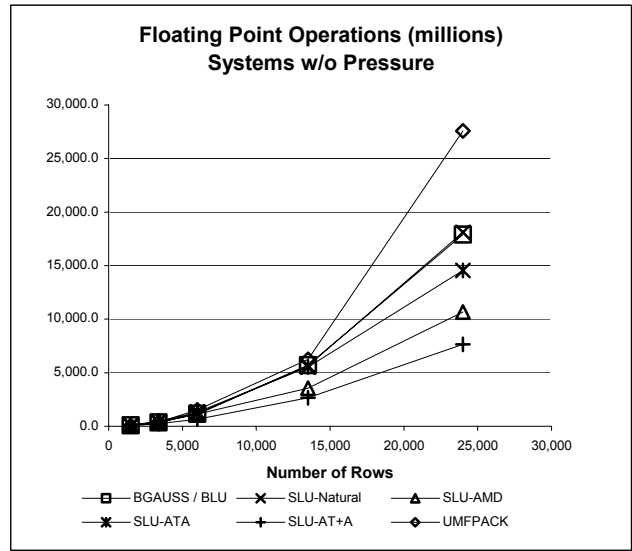


Figure 5.

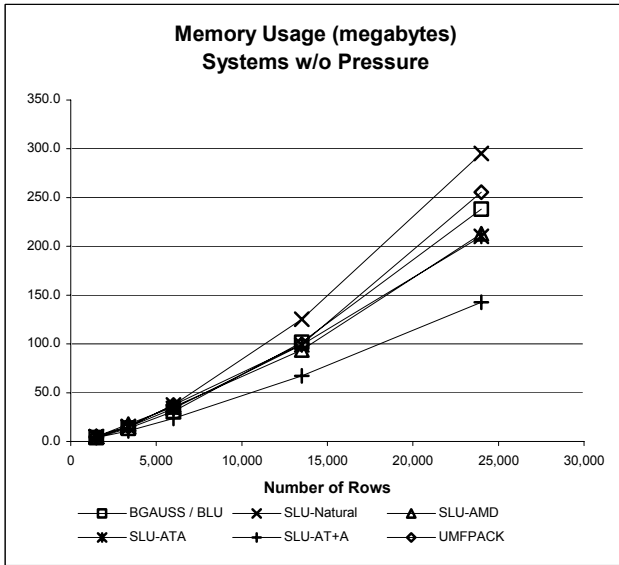


Figure 4.

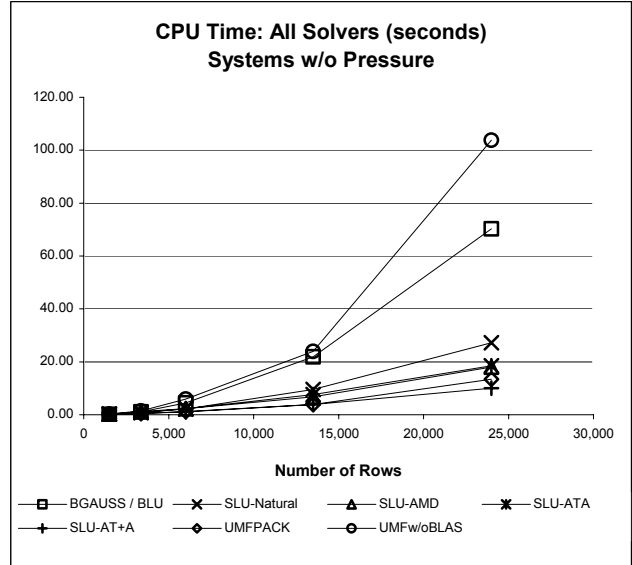


Figure 6.

UMFPACK selects the approximate minimum degree algorithm for ordering columns of banded matrices without pressure and produces slightly fewer non-zero entries in L+U than banded LU factorization. Compared with banded LU factorization, SuperLU and UMFPACK require memory in excess of the memory required to store non-zero entries of L+U. SuperLU with the multiple minimum degree algorithm on A^T+A produces the fewest non-zeros in L+U and uses the least amount of memory. All column-ordering methods produced stable computations for these matrices.

UMFPACK performs significantly more floating point operations than SuperLU and banded LU factorization. However, UMFPACK makes significantly better use of BLAS by using matrix-matrix multiply operations, producing overall CPU times that are comparable to the best SuperLU CPU times. UMFPACK without BLAS performs approximately the same number of floating point operations per second as banded LU factorization, which does not use BLAS. SuperLU natural column ordering performs the same number of floating point operations as banded LU factorization, but only uses 40% as much CPU time due to the performance enhancement of the BLAS.

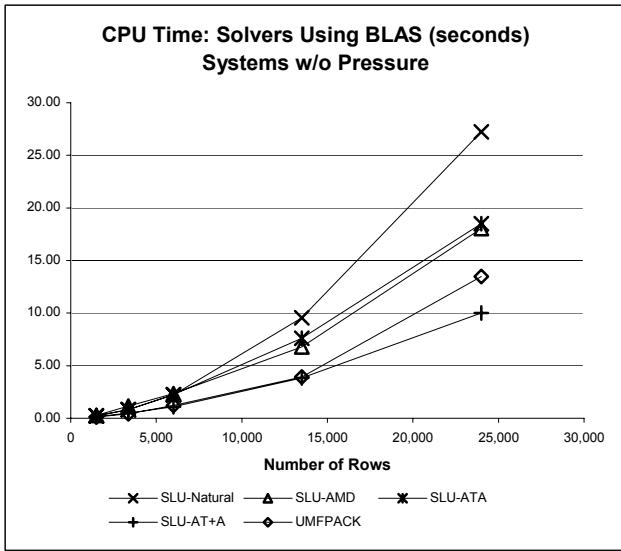


Figure 7.

Solvers that do not use BLAS dominate the CPU time graph in Figure 6. Figure 7 shows CPU times for only those solvers that use BLAS, giving a more detailed graph of their relative performance.

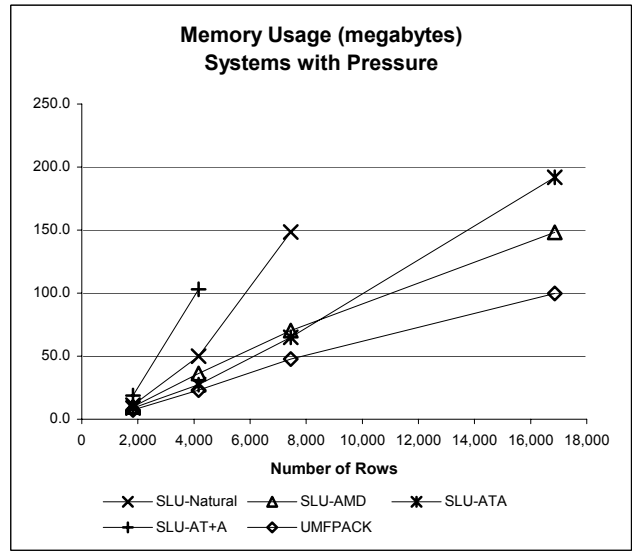


Figure 9.

UMFPACK chooses the column approximate minimum degree algorithm for matrices with pressure. This strategy produced stable computations for all but the largest test matrix. SuperLU with the multiple minimum degree algorithm on $A^T A$ and the approximate minimum degree ordering algorithm produced stable computations except for the largest matrix with multiple minimum degree ordering. SuperLU's other two order methods produced stable computations, but the number of non-zeros in L+U grew too fast for them to be usable for larger problems.

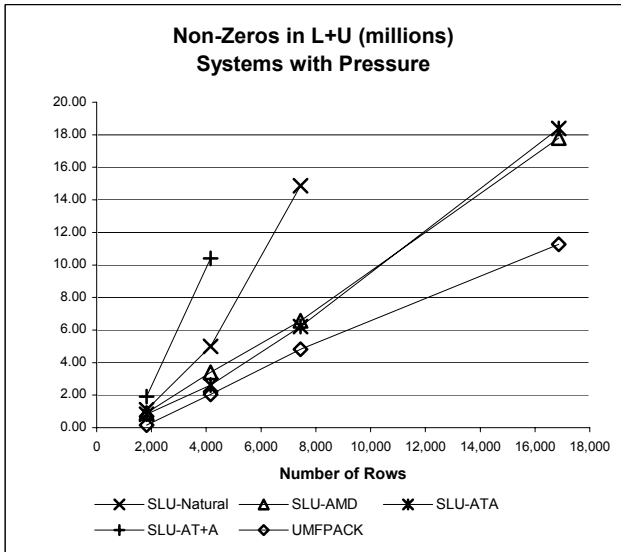


Figure 8.

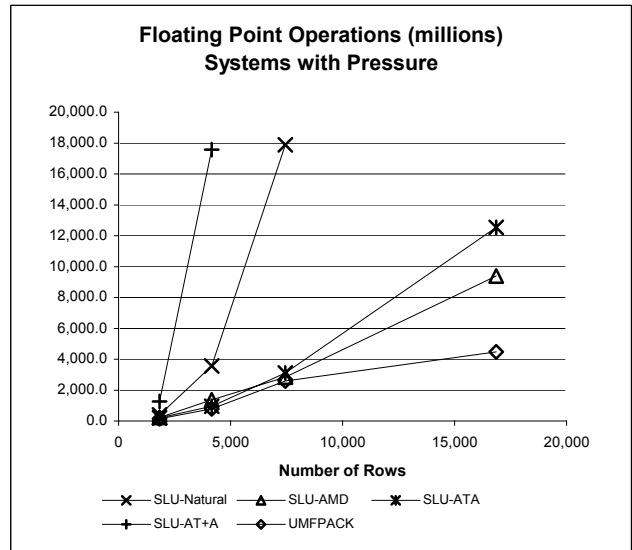


Figure 10.

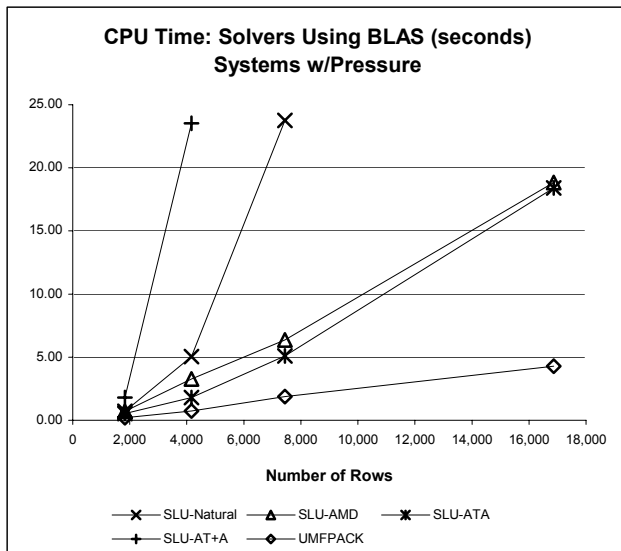


Figure 11.

Floating point operations and CPU time for systems with pressure show trends similar to the number of non-zeros in L+U and memory usage. UMFPACK performance is particularly good, however instability for the largest problem size leaves SuperLU with approximate minimum degree ordering as the better choice.

5 Conclusions

The packages provide significantly better performance and functionality than our hand-coded methods. In addition, they are able to solve systems with pressure, which is impractical to do with our hand-coded methods. The BLAS is a crucial element for high performance. Care must be taken to choose appropriate column ordering algorithms.

The modified compressed column data structure and its supporting routines work well. They are actually easier to use in the ice sheet model than the banded matrix structure. Matrix elements are directly referenced using row and column numbers with the modified compressed column data structure. The banded matrix structure stores the diagonal column of the matrix in the center column of an array, requiring a mapping function between matrix column numbers and array column numbers.

6 References

- [1] Davis, T. A. (2005). *UMFPACK Version 4.3.1 User Guide*. Tech Report TR-04-003, Dept. of Computer and Information Science and Engineering, Univ. of Florida. <http://www.cise.ufl.edu/research/sparse/umfpack>.
- [2] Davis, T. A., and Duff, I. S. (1994). *An Unsymmetric-Pattern Multifrontal Method for Sparse LU Factorization*. Tech report TR-94-038, Dept. of Computer and Information Science and Engineering, Univ. of Florida.
- [3] Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, Xiaoye S., and Liu, J. W. H. (1999). *A Supernodal Approach to Sparse Partial Pivoting*. Siam J. Matrix Analysis and Applications, 20, 720-755.
- [4] Demmel, J. W., Gilbert, J. R., and Li, Xiaoye S. (1999, Last update 2003). *SuperLU Users' Guide*. Tech report LBNL-44289, Computational Research Division, Lawrence Berkley National Laboratory. <http://crd.lbl.gov/~xiaoye/SuperLU/>.
- [5] Duff, I. S., Erisman, A. M., and Reid, J. K. (1986). *Direct Methods for Sparse Matrices*. Oxford University Press.
- [6] Fastook, J. L. (1993). *The Finite Element Method For Solving Conservation Equations in Galciology*. Computational Science and Engineering, Vol. 1, No. 1, 55-67.
- [7] Jacobs, R. A. (2005). *Data Structures & Algorithms for Efficient Solution of Simultaneous Linear Equations from 3-D Ice Sheet Models*. Master thesis, University of Maine.
- [8] Li, Xiaoye S. (2005). *An overview of SuperLU: Algorithms, Implementation, and User Interface*. ACM Transactions on Mathematical Software, Vol. 31, No. 3, 302-325.