Sparse Matrix Software Package Design Proposal

CS 365: Mathematics of Computer Science

Kyle Lee

[kylelee91@me.com](mailto:kylelee91@me.com)

Terence Henriod

[thenriod@gmail.com](mailto:thenriod@gmail.com)

12/5/2013

1. **SparTrix Introduction**

SparTrix will be a software package designed to handle a particular class of matrices known as *sparse matrices*. This package will handle storage, representation, and basic matrix operations for square sparse matrices. This package will focus primarily on the matrix computations, while the representation of the matrices will be primitive in the current version.

In the first version, SparTrix will be designed for those experienced in dealing with sparse matrix operations, and need software that can perform at large scales. This software will not be intended for beginners, but future versions of SparTrix will likely become more user friendly.

Sparse matrices when, most loosely defined, are matrices with fewer significant elements than zero (or not meaningful) elements. In practical terms, sparse matrices are generally thought of as matrices with far fewer significant elements relative to the size of the matrix. SparTrix will be designed for particularly sparse, or low density, matrices.

In order to ensure that a concise, well-designed software package is built, the scope of SparTrix will be kept narrow. This means that user interface will be kept minimal (largely text based) and there will be few operations implemented. The package will be designed tightly and error free, utilizing the best known algorithms that are tuned for efficient computing performance for certain cases rather than flexibility. Storage methods will lend themselves to computational methods, rather than being optimized for storage capacity or matrix construction. Data input methods will be few and defined.

Although SparTrix will ideally be able to handle different classes of sparse matrices, in keeping with the spirit of narrow scope, SparTrix should be designed to be best suited for *banded matrices*. Banded matrices, briefly defined, are matrices where the significant elements are concentrated near the main diagonal. A banded matrix has a relatively narrow range from the main diagonal where significant entries may appear; all entries outside this range must be zero (or considered insignificant) for a matrix to be considered banded.

Despite this notion of concision, however, some features of SparTrix will be designed in a forward looking manner, laying groundwork to expand the package with relative ease.

1. **Definitions**

Before proceeding, significant sparse matrix related terms should be defined so that there is little confusion as to what is meant when discussing the concepts of the design of SparTrix.

|  |  |
| --- | --- |
| Sparse Matrix | For the purposes of the SparTrix design, a matrix will be considered sparse if it has fewer than significant elements. Ideally, there will be fewer elements for SparTrix to operate optimally until more functionality is added in future versions. [1] |
|  |  |
| Banded Matrix | A banded (or band) matrix is a matrix whose significant entries are confined to a diagonal band that includes the main diagonal and possibly diagonals above and below the main diagonal. Formally, a bandwidth is defined as the lowest integer k where all significant an element aij is zero when i > (j + p). Put simply, this means that all significant elements exist within a range of k units away from the main diagonal. Often, an upper and lower bandwidth are defined, and the resulting total bandwidth is found by k1 + k2 + 1. [1] |
|  |  |
| Significant Element | Typically a significant element is defined as a non-zero element. This will be the default in SparTrix. Users may set a threshold where values below the threshold are regarded as zero. |
|  |  |
| Matrix Storage Format | When storing matrices, particularly sparse ones, it is important to store the elements in a way that takes advantage of being able to neglect non-significant, or zero, elements, thus reducing the memory required. Such formats will be discussed later, but it is important to acknowledge here that a matrix storage format is one that assumes that un-stored data is 0, and only the elements that are stored are 0. |
|  |  |

1. **SparTrix Interface**

**Display and User Interface**

In general, the SparTrix display will appear as a simple command line window (similar to that often seen in common mathematical analysis software packages). A menu bar across the top of the window will be available, but it will present only essential options. This menu bar will contain all typical program menus (including a help menu) as well as those essential to operating SparTrix’s most basic functions. The command line interface, however, will afford the user access to the full range of SparTrix’s functionality, including primitive output.

A command line interface was chosen to cut down on graphical overhead. This choice was made to allow the initial version of SparTrix to focus on functionality. The command line interface is also an ergonomic choice. Often, those experienced with mathematical software packages are not interested in hunting through menus to find an option or procedure command; it is easier to simply type the command. This conclusion was reached based on both the personal preferences of the SparTrix Design Team, as well as data collected from academic professionals. There will be topics found in help menus to guide users through the syntax of the SparTrix commands.

In general, SparTrix output will be simple ASCII character output that is typical of command line interfaces. Matrices will be represented in a separate window that is ideal for scanning the general structure of a matrix and representing matrices in a printable format. This matrix display window will display the entries that appear near the main diagonal of the matrix horizontally, one section at a time, moving from the upper left to the lower right. That is, the rows of the matrix will be displayed, left justified in a column, even though the entries will be offset from one another to form a diagonal band in reality. This will neglect the bulk of a matrix that is considered non-significant. This does, introduce the possibility that entries far from the diagonal will be neglected. This choice was made to reduce overhead by displaying non-significant parts of the matrix that usually consist entirely of zero entries.

While a more developed Graphical User Interface (GUI) was considered for SparTrix, it has been decided that in early versions of the package, those most likely to use SparTrix in its early versions would not be looking for a package with bells and whistles, but rather, raw functionality. In later versions of SparTrix it would make sense to add a well-developed GUI to the package, but ideally an optional one. Experienced users would still be able to revert to the command line interface. In the event that a GUI (like the one described) is added, it would largely consist of menus and an interactive output window so that matrix operations or commands could be easily modified or repeated.

**Data Input**

There will be two data input modalities: manual input and digital input from files. Using digital input methods is HIGHLY encouraged to maintain data integrity.

*Manual Input*

It is possible for users to input data manually, if necessary. Given the large data set sizes characteristic of the sparse matrices used in practice, this is not recommended as data fidelity is risked.

Should the user decide to input a matrix manually, a second window will open that will guide the user through the data entry process. This window will present 4 data entry fields required for defining a sparse matrix in Yale format (discussed in the section on matrix storage and representation), and strict instructions will be provided for the user. These fields will be: one field for the dimension of one side of a matrix (which will be squared, of course), and then three fields for the data that goes in each of the three arrays of the Yale format. This very rigid input structure will help to mitigate entry errors given the complexity and length of the task of entering sparse matrices (which are typically). Entering an entire matrix is disallowed, as entering an entire matrix would be much too arduous for the user.

*Digital Input*

SparTrix will support two file formats: the common .csv and the SparTrix specified .stx.

SparTrix will be designed to read data in from the simple Comma Separated Value format because it is accessible, versatile, and can be easily modified by hand if necessary. In fact, if users are forced to input or modify data manually, spreadsheet software can be used to conduct these operations. Only the first matrix stored in a .csv file will be recognized by SparTrix.

The SparTrix file type, .stx, will be preferable once a data set has been defined digitally because then the data can be stored in a protected module that cannot be accessed by other programs. This format also reduces the likelihood of data corruption because data irregularities that may arise due to the changing of data formats (for example .csv to .stx). There is a smaller probability that errant data will be able to find its way into the data. Further, this format allows for multiple matrices to be stored in the same file.

SparTrix is designed to accept data from an entire matrix from a CSV file if the special option to accept an entire matrix is selected. SparTrix will read in the data and convert it to the Yale storage format appropriately.

There are really no alternative methods whose implementations would be would be worth discussing, as other methods that were considered provided no advantages.

*Matrix Representation*

Matrices will, by necessity, be represented in a separate window, unless they are small enough to be displayed in the

1. **Matrix Storage and Representation**

All matrix entries will be stored as the double type, since many matrix operations can easily result in non-integer numbers. It would make the package much less effective if it was constructed using the integer type, and the programming to accommodate for this to keep the software package effective would be an unnecessary, unrewarding challenge. This of course only applies to the data element array of the Yale format (the first array described below), while it is only reasonable to store position indices as integers (as in the second and third array described below).

Due to its versatility and memory efficiency, the Yale format will be used. The Yale format for storing sparse matrices is defined using three arrays. The first of these arrays contains all of the significant entries of the matrix, listed in the order they appear in the matrix, working from left to right, from the top row to the bottom row. The second array’s entries indicate the indices of the elements in the first array that are the first elements of each new row. Finally, the third array contains the column indices of all the elements in the first array. Any element not indicated by the data in the array trio is assumed to be non-significant. [2]

The Yale format was chosen primarily for its similarity to one of the more common matrix representations for performing computations on sparse matrices, the Compressed Sparse Row (CSR) format. It should be noted that the CSR format is nearly identical to the Yale format except that the column array is normally stored before the row array, as opposed to storing the row index array before the column one in Yale format. This format allows matrices to be stored and represented in memory efficiently, assuming the matrix is sparse enough. [2]

Unfortunately, the Yale format is an inferior format in terms of construction complexity when compared to other typical storage formats, formats such as the List of Lists (LIL), where a list of significant elements is stored for each row, or the Coordinate list (COO), which is a list of coordinate pair and value groupings. These storage format alternatives are simpler to construct as computations to determine where the first element of a row falls in a separate list are not required, and the access of individual elements does not require any sort of decoding, as might occur with the Yale format. These other formats were not chosen, though, because their data must be converted to another format for arithmetic operations to take place (like CSR), or to at least carry out arithmetic efficiently. SparTrix is designed to be optimized for mathematical operations and not for storage operations. Thus the Yale format was chosen because a matrix only needs to be constructed once in Yale format and then is ready for computations every time, while the other formats may be quickly constructed, but must be converted back and forth to other formats every time a computation occurs. Implementing the alternative formats for purposes of matrix data construction or for user preference would be worth consideration in future versions. [1]

1. **Supported Operations**

In the first version, SparTrix will only be able to handle the most primitive operations, but suitable algorithms for the operations will be applied. Focus will be on operations that are necessary operations that are required for more advanced, but useful, operations.

**Overview**

*Basic Arithmetic*

Basic addition and multiplication will be supported. These are standard algorithms, so there is not much decision making to be done here. The typical algorithms will be used.

*Transposition*

Matrix transposition will be performed using the TRANSP algorithm defined in the SMMP created by Bank and Douglas. Foregoing a lengthy complexity analysis of the algorithm, suffice it to say that the algorithm came from Yale, the source of the storage format that SparTrix will use, and is supported by experts in the field. [2]. Given the length of the algorithm, it will not be discussed here. Matrix transposition will be important for implementing future matrix operations, so it will be implemented in the first version of SparTrix.

*Determinant*

It may be necessary for the user to know what the determinant of a matrix is. SparTrix will be able to compute the determinant of a matrix using methods that apply best to banded matrices of low density.

**A Listing of Basic Interface Functions**

The following table lists the basic commands that the user may call on SparTrix to perform via the command line interface or any of the top of the screen menus. When these commands are used, the result is computed “in place.” This means that if the command is part of a larger expression the result is passed as an argument to the greater expression as part of the computation. If another variable/matrix uses the command as part of an assignment, the result will simply be stored in the variable/matrix and as long as there are no errors or warnings to report, the action occurs “silently.” If the command is used by itself, then the result is displayed to the user on screen (if possible, will result cannot be displayed for large matrices).

|  |  |
| --- | --- |
| **Operation Name** | **Brief Description** |
| add( A, B ) | Adds matrix A to matrix B, then returns the result “in place.” If matrices A and B do not have compatible dimensions, then an exception occurs and the entire operation/computation is halted and the error is reported. If the resulting matrix is small enough to display on the screen, it will be displayed. |
|  |  |
| subtract( A, B ) | Subtracts matrix B from matrix A and returns the result “in place.” If matrices A and B do not have compatible dimensions, then an exception occurs and the entire operation/computation is halted and the error is reported. If the resulting matrix is small enough to display on the screen, it will be displayed. |
|  |  |
| multiply( A, B ) | Multiplies matrices A and B and returns the result “in place.” If matrices A and B do not have compatible dimensions, then an exception occurs and the entire operation/computation is halted and the error is reported. If the resulting matrix is small enough to display on the screen, it will be displayed. |
|  |  |
| determinant( A ) | Computes the determinant of matrix A. The result is returned “in place.” If the command is given independently, then the result is simply displayed for the user to see. |
|  |  |
| transpose( A ) | Transposes matrix A and returns the result “in place.” If matrices A and B do not have compatible dimensions, then an exception occurs and the entire operation/computation is halted and the error is reported. If the resulting matrix is small enough to display on the screen, it will be displayed. |
|  |  |
| sparsity( A ) | Computes the sparsity coefficient (ratio of significant elements to non-significant ones) of matrix A. The result is returned “in place.” If the command is given independently, then the result is simply displayed for the user to see. |
|  |  |

1. **Limitations**

**Square Matrices**

The first version of SparTrix will be designed to handle square matrices only. Attempting to input a non-square matrix will result in undesired results. SpartTrix is designed to alert the user in most cases that an entered matrix is not square, but depending on the arrangement of data items, it is possible that SparTrix could interpret a rectangular matrix, one with non-significant (all-zero) rows below the row containing the last significant data item, as a square matrix. This is most likely to happen if the user chooses to enter a matrix manually and enters incorrect dimensions for the matrix.

All of the methods in the current version of SparTrix are designed for square matrices, so it is not advised that the user attempt to “work around” this square matrix requirement. For example, the user should avoid storing a matrix that is actually rectangular that has rows that are all-zero below the row containing the last significant data item. SparTrix will process this matrix as though it were square and unsound computations may result.

**Size Limits**

In accordance with average contemporary computer system capabilities, SparTrix will limit the size of matrices to 1 E6, squared. This may be adjusted in the future as typical system capabilities improve with technological improvements.

**Matrix Density**

Matrices should be relatively sparse in order for SparTrix to perform effectively. Only matrices with a number of significant elements less than significant data items, where n is the row/column length, should be considered. This limit is imposed so that efficient use of the Yale storage format can be utilized, as storing any more data items than the formula indicates can cause the Yale format to require more memory than storing the actual matrix would. This also helps to ensure that SparTrix’s methods so not become inefficient when operating on a given data set. It should be noted that this is only an upper limit, ideally matrices will be even less dense.

**Banded Matrices**

SparTrix may not perform optimally if an entered matrix is not banded. This is not a strict limitation of SparTrix, but it should be noted that the first version of SparTrix will be oriented toward the handling of banded matrices so this limitation should be observed in order to expect ideal performance.

1. **Error Handling**

In general, SparTrix will report errors before any processing is done. When a command is entered, SparTrix will make all possible checks to ensure that valid processing can continue before any computations are begun. Should any checks relevant to the operation fail, processing is halted and a pertinent error message is displayed so the user can correct the problem.

SparTrix will generate regular checkpoints during longer computations (longer than 1 minute) to prevent loss of time or processing work. This will help mitigate runtime errors. For example, should SparTrix reach the limit of available memory or should some other runtime error occur, SparTrix will halt processing and notify the user of the error. SparTrix will keep only the most recent checkpoint in memory, so a user cannot expect to alter the computation they have entered or re-attempt failed computations that are not the most recently attempted computation. This will give the user an opportunity to resolve the issue (close programs, delete data, etc.) and continue the computation without losing too much time or computational work.

1. **Future Developments**

Future developments have been discussed throughout the proposal as a reminder of the forward-looking theme of the development of SparTrix. To re-iterate, major improvements to be added in future versions will include: GUI options, improvements in matrix representations, and addition of matrix operations. Different algorithms for existing operations will be added to accommodate effective processing of more classes of sparse matrices than just banded ones. Increasing SparTrix’ versatility to include non-square matrix handling capabilities would also be worth serious consideration.

1. **References**

|  |  |
| --- | --- |
| [1] | G. H. Golub and C. F. Van Loan, Matrix Computations, 3rd ed., Baltimore: Johns Hopkins, 1996. |
| [2] | R. E. Bank and C. C. Douglas, "Sparse Matrix Multiplication Package (SMMP)," 23 April 2001. [Online]. Available: http://www.cs.yale.edu/homes/douglas-craig/Preprints/pub34.pdf. [Accessed 28 October 2013]. |