Webnucleo Technical Report: wn_matrix Module

David Adams and Bradley S. Meyer

June 4, 2008

The following will provide information regarding the routines that comprise the wn_matrix Module. The wn_matrix Module is divided into two parts: routines that deal with creating and managing the WnMatrix structure, and routines for performing matrix operations with the WnMatrix structures. This report describes the WnMatrix structures and details of some of the wn_matrix routines. The module itself can be found at:

http://www.webnucleo.org/home/modules/wn_matrix/

1 wn_matrix Structures Overview

WnMatrix is a structure for storing a matrix. As of version 0.2, elements of the matrix are stored in a multi-dimensional hash, which provides excellent flexibility in adding and removing elements from the matrix. The hash routines used are those from libxml, the xml C parser and toolkit of the Gnome. Version 0.1, now no longer supported, used doubly-linked lists, which could require $\mathcal{O}$(N) operations, where N is the number of columns in the matrix, to store and retrieve elements. The hash only requires $\mathcal{O}(1)$ operations.

The WnMatrix structure is central to the module, as the majority of the routines contained within the module are used to either operate on the structure or retrieve information about its contents. The structure itself is principally comprised of a pointer to a libxml xmlHashTable pointer, in which the non-zero elements are stored. Other structure elements are unsigned ints containing the number of rows and columns in the matrix (the number of non-zero elements is not kept but rather is retrieved by the libxml xmlHashSize routine). Finally, there is a pointer to an internal data structure for use in callback routines on the hash.

As of version 0.3, the clear and scale matrix routines are correct. Version 0.2 used hash callbacks for these routines but modified the hash during the callbacks. This naturally led to undefined behavior. Version 0.3 also adds getCopy and getTranspose routines to the API. These return new matrices that the caller must free with WnMatrix__free when no longer needed.

WnMatrix__Line is a structure for storing data relevant to the non-zero elements of a line in the matrix, that is, a row or a column. These data are stored in the structure as arrays.

As of release 0.4, we have introduced three new structures, namely, *WnMatrix__Coo*, *WnMatrix__Csr*, and *WnMatrix__Yale* for storing the sparse matrix in coordinate, compressed sparse row, and Yale sparse matrix format, respectively. We have also rearranged the API to accomodate these changes, which means there is a backward incompatibilty between release 0.4 and earlier versions. The reason for the change is to relieve the user of the burden of allocating memory for these alternative sparse matrix formats. Thus, for example, when the user calls *WnMatrix__getCoo()*, a pointer to a coordinate matrix is returned. The *WnMatrix__getCoo()* routine does all the memory allocation. The routine *WnMatrix__Coo__getRowVector()* then returns the coordinate row matrix array (with a number of elements equal to the number of non-zero elements in the matrix). The user frees the memory for the coordinate matrix (and, consequently, the row array) by calling the *WnMatrix__Coo__free()* routine. Example codes in the src/examples directory of the distribution demonstrate how this works.

Our philosophy has been that the user should not have to worry about the data structures used by wn_matrix. Instead, we intend that the user should interact with the wn_matrix structures via the API routines. For this reason, the API does not make the content of the wn_matrix structures public. Nevertheless, the interested user may find their prototypes located in the WnMatrix header file `WnMatrix.h`.

2 wn_matrix Routines

For documentation on the wn_matrix routines, see the WnMatrix.h file in the Overview in the Technical Resources for the current release. The documentation provides a brief description of each routine, the prototype, pre- and post-conditions, and examples on using the routines.

As of release 0.5, we have introduced routines to read input matrix data from XML and output matrix data to XML. For example, the routine *WnMatrix__new_from_xml()* inputs matrix data in row, column, value triplets from an XML file and creates a matrix based on those data. The routines *WnMatrix__Coo__writeToXmlFile()*, *WnMatrix__Coo__writeToXmlFile()*, and *WnMatrix__Coo__writeToXmlFile()* output coordinate, compressed sparse row, and Yale matrix forms of the matrix to XML output. A new routine also allows the user to validate the input XML file against Webnucleo.org's input matrix schema for version 0.5 (future releases will have similar xsd_pub directories). For more details, the user should consult the API documentation.

As of version 0.7, wn_matrix uses the gnu gsl scientific library vector structure when interacting with vectors. This change of course means that wn_matrix now depends on gsl as well as libxml. We have added routines to parse in vector data from an xml file (*WnMatrix__new_gsl_vector_from_xml()*), to validate the xml (*WnMatrix__is_valid_vector_input_xml()*), and to dump a gsl_vector to an xml file (*WnMatrix__write_gsl_vector_to_xml_file()*). While it is possible to interact with the elements of the gsl_vector data structure directly (see the gsl documentation), we have also added API routines to get the size of the vec-

tor (*WnMatrix__get_gsl_vector_size()*) and to retrieve the data array from the gsl_vector (*WnMatrix__get_gsl_vector_array()*).

Our change over to using gsl_vectors simplifies the wn_matrix API for the matrix times vector and transpose matrix times vector routines. It nevertheless introduces a backwards incompatibility. While we could have kept the routines

WnMatrix__getMatrixTimesVector()

and

WnMatrix__getTransposeMatrixTimesVector()

and deprecated them, we decided to remove them and replace them with

WnMatrix__computeMatrixTimesVector()

and

WnMatrix__computeTransposeMatrixTimesVector().

This is desirable since the old routines did not in fact retrieve (get) pre-existing data but rather computed them. The new routines not only have simplified calls because they use gsl_vectors, but their names also better reflect their functionality.

Because wn_matrix now depends on gsl, it made sense in version 0.7 for us also to replace *WnMatrix__getDenseMatrix()* that returned a double** with *WnMatrix__getGslMatrix()* that returns a gsl_matrix *. This makes for a cleaner interface in the API and easier allocations. Also, we added *WnMatrix__solve()*, a routine that solves a matrix equation $Ax = b$, given input vector A and right-hand-side vector b. The routine uses gsl linear algebra routines to solve the equation.