Webnucleo Technical Report: wn_matrix Module

David Adams and Bradley S. Meyer

September 28, 2007

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 provides some example code demonstrating the use of 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 them 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.

3 Example C Code Calling WnMatrix Structure

The following code shows how to use the WnMatrix structure and routines in a C code. It is included in the distribution as `example1.c`. To compile, use Makefile; thus, type `make example1`. Examples 2-9 are also included in the distribution, and may be compiled in the same fashion.

```
#include "WnMatrix.h"

int main( void ) {

  unsigned int i_row, i_col;
  WnMatrix * p_my_matrix;
```

```
/*========================================================================
// Create a 3 x 3 matrix called my_matrix and a pointer to it
// called p_my_matrix.
//======================================================================*/

p_my_matrix = WnMatrix__new( 3, 3 );

/* Assign the following matrix:

   | 10.  0.   3. |
   |  0.  0.   0. |
   | -5.  2.   0. |
*/

WnMatrix__assignElement( p_my_matrix, 1, 2, 1. );  /* Remove this below */

WnMatrix__assignElement( p_my_matrix, 1, 1, 10. );

WnMatrix__assignElement( p_my_matrix, 3, 1, -2.5 );

WnMatrix__assignElement( p_my_matrix, 3, 2, 2. );

WnMatrix__assignElement( p_my_matrix, 3, 1, -2.5 );

WnMatrix__assignElement( p_my_matrix, 1, 3, 3. );

if( WnMatrix__removeElement( p_my_matrix, 1, 2 ) == -1 ) {
  fprintf( stderr, "Couldn't remove element!\n" );
  return EXIT_FAILURE;
}

/*========================================================================
// Print out the matrix.
//======================================================================*/

printf( "\nThe elements of the matrix are:\n\n" );
printf( "Row   Column   Value\n" );
printf( "---   ------   -----\n" );

i_row = 1;
i_col = 1;

for ( i_row = 1; i_row <= 3; i_row++ ) {

  for ( i_col = 1; i_col <= 3; i_col++ ) {
    printf( "%3d   %6d   %5.1f\n",
      i_row,
      i_col,
      WnMatrix__getElement( p_my_matrix, i_row, i_col )
    );

  }

}

printf(
   "\nNumber of non-zero elements = %d\n",
   WnMatrix__getNumberOfElements( p_my_matrix )
);

/*==============================================================================
// Double the elements of the matrix and print out.
//============================================================================*/

WnMatrix__scaleMatrix( p_my_matrix, 2. );

printf( "\nThe elements of the matrix are (when doubled):\n\n" );
printf( "Row   Column   Value\n" );
printf( "---   ------   -----\n" );

i_row = 1;
i_col = 1;

for ( i_row = 1; i_row <= 3; i_row++ ) {

  for ( i_col = 1; i_col <= 3; i_col++ ) {

    printf( "%3d   %6d   %5.1f\n",
      i_row,
      i_col,
      WnMatrix__getElement( p_my_matrix, i_row, i_col )
    );

  }

}

printf(
   "\nNumber of non-zero elements = %d\n",
   WnMatrix__getNumberOfElements( p_my_matrix )
);
```

```
/*============================================================================
// Clear matrix.  Since we only clear p_my_matrix, we can still use it.
//==========================================================================*/

printf( "\nNow clear matrix:\n\n" );

WnMatrix__clear( p_my_matrix );

printf(
   "Number of non-zero elements = %d\n\n",
   WnMatrix__getNumberOfElements( p_my_matrix )
);

/*============================================================================
// Assign element and print number of elements.
//==========================================================================*/

printf( "Now add element:\n\n" );
WnMatrix__assignElement( p_my_matrix, 1, 1, 1. );

printf(
   "Number of non-zero elements = %d\n\n",
   WnMatrix__getNumberOfElements( p_my_matrix )
);

/*============================================================================
// Now free matrix.
//==========================================================================*/

printf( "Now free matrix.\n\n" );
WnMatrix__free( p_my_matrix );

/*============================================================================
// Since p_my_matrix freed, must reallocate to reuse.
//==========================================================================*/

printf( "Now re-create matrix.\n\n" );
p_my_matrix = WnMatrix__new( 3, 3 );

printf(
   "Number of non-zero elements = %d\n\n",
   WnMatrix__getNumberOfElements( p_my_matrix )
);

/*============================================================================
  // Assign element and print number of elements.
  //===========================================================================*/

  printf( "Now add element:\n\n" );
  WnMatrix__assignElement( p_my_matrix, 1, 1, 1. );

  printf(
     "Number of non-zero elements = %d\n\n",
     WnMatrix__getNumberOfElements( p_my_matrix )
  );

  /*============================================================================
  // Clean up and exit.
  //===========================================================================*/

  WnMatrix__free( p_my_matrix );

  return EXIT_SUCCESS;

}
```