Webnucleo Technical Report: wn_matrix Module

David Adams and Bradley S. Meyer

June 24, 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 and WnMatrix__Line structures. This report describes the function and usage of these routines, as well as the WnMatrix and WnMatrix__Line structures themselves. The module itself can be found at:

http://nucleo.ces.clemson.edu/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 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.

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.

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

The following is a list of the prototypes for all the routines contained within the wn_matrix module API, as well as a brief description of their purpose. For more information, visit the wn_matrix web site noted above.

-
  ```
  WnMatrix *WnMatrix__new(
    unsigned int i_rows,
    unsigned int i_columns
  );
  ```

  Initializes a WnMatrix structure.

-
  ```
  void WnMatrix__assignElement(
    WnMatrix *self, unsigned int i_row,
    unsigned int i_col, double d_val
  );
  ```

  Assigns an element to a WnMatrix structure. If an element already exists at that (row,col), the new value is added to the existing value.

-
  ```
  double WnMatrix__getElementi(
    WnMatrix *self, unsigned int i_row, unsigned int i_col
  );
  ```

  Retrieves the value of the specified matrix element.

-
  ```
  int WnMatrix__removeElement(
    WnMatrix *self, unsigned int i_row, unsigned int i_col
  );
  ```

  Removes the specified matrix element from the matrix.

-
  ```
  void WnMatrix__free( WnMatrix *self );
  ```

  Frees the memory allocated for a WnMatrix structure.

-
  ```
  int WnMatrix__writeMatrixToAsciiFile(
    WnMatrix *self, char * s_ascii_filename, double d_cutoff
  );
  ```

Writes out the elements of a matrix stored in a WnMatrix structure larger than a cutoff value to a file.

- `unsigned int WnMatrix__getNumberOfElements( WnMatrix *self );`

  Returns the number of elements in the matrix.

- `unsigned int WnMatrix__getNumberOfRows( WnMatrix *self );`

  Returns the number of rows in the matrix.

- `unsigned int WnMatrix__getNumberOfColumns( WnMatrix *self );`

  Returns the number of columns in the matrix.

-
  ```
  void WnMatrix__getMatrixTimesVector(
    WnMatrix *self, unsigned int i_vector_size, double a_vector_input[],
    double a_vector_output[]
  );
  ```

  Computes the vector y from the equation y = Ax, given matrix A and vector x.

-
  ```
  void WnMatrix__getTransposeMatrixTimesVector(
    WnMatrix *self, unsigned int i_vector_size, double a_vector_input[],
    double a_vector_output[]
  );
  ```

  Computes the matrix equation Y = A(transpose)x given A and x.

-
  ```
  int WnMatrix__insertMatrix(
    WnMatrix *self, WnMatrix *p_inserted_matrix,
    unsigned int i_row, unsigned int i_col
  );
  ```

  Inserts a smaller WnMatrix into a larger one.

-
  ```
  double *WnMatrix__getDiagonalElements(
    WnMatrix *self
  );
  ```

  Stores the values of the diagonal elements in the double array passed into the function.

- `WnMatrix * WnMatrix__getTransferMatrix( WnMatrix *self );`

  Returns the F matrix (Fij = aij/ajj, but no diagonal elements).

-
  ```
  void WnMatrix__getCsrMatrix(
    WnMatrix *self, unsigned int a_rwptr[], unsigned int a_col[],
    double a_val[]
  );
  ```

  Converts a WnMatrix into Compressed Sparse Row format.

-
  ```
  void WnMatrix__getYaleSparseMatrix(
    WnMatrix *self, int a_ija[], double a_sa[]
  );
  ```

  Stores the matrix in Yale Sparse format in the arrays passed into the function.

-
  ```
  void WnMatrix__addValueToDiagonals( WnMatrix *self, double d_val );
  ```

  Adds a value to the diagonal elements of the matrix.

-
  ```
  WnMatrix__Line *WnMatrix__getRow(
    WnMatrix *self, unsigned int i_row
  );
  ```

  Routine to get a row from a WnMatrix structure.

-
  ```
  WnMatrix__Line *WnMatrix__getRow(
    WnMatrix *self, unsigned int i_col
  );
  ```

  Routine to get a column from a WnMatrix structure.

-
  ```
  unsigned int WnMatrix__Line__getNumberOfElements(
    WnMatrix__Line *self
  );
  ```

  Routine to get the number of non-zero elements in a row or column in a WnMatrix structure.

-
  ```
  unsigned int *WnMatrix__Line__getNonZeroIndices(
    WnMatrix__Line *self
  );
  ```

  Routine to get an array containing the column numbers of the non-zero elements in a row or the row numbers of the non-zero elements in a column in a WnMatrix__Line structure.

-
  ```
  double *WnMatrix__Line__getNonZeroElements(
    WnMatrix__Line *self
  );
  ```

  Routine to get an array containing the values of the non-zero elements in a row or of the non-zero elements in a column in a WnMatrix__Line structure.

-
  ```
  WnMatrix *WnMatrix__extractMatrix( WnMatrix *self,
                                     unsigned int i_row,
                                     unsigned int i_col,
                                     unsigned int i_row_offset,
                                     unsigned int i_col_offset );
  ```

  Extracts a smaller WnMatrix from a larger one.

- `double **WnMatrix__getDenseMatrix( WnMatrix *self );`

  Stores the matrix in dense format in a doubly indexed array.

- `void WnMatrix__scaleMatrix( WnMatrix *self, double d_val );`

  Multiplies all the elements in the matrix by a scalar constant.

- `void WnMatrix__Line__free( WnMatrix__Line *self );`

  Frees the memory allocated for a WnMatrix__Line structure.

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 2-8 are also included in the distribution, and may be compiled in the same fashion.

```
#include "WnMatrix.h"

int main() {

  unsigned int i_row, i_col;
  WnMatrix * p_my_matrix;

  /* Create a matrix called my_matrix and a pointer to it called p_my_matrix. */

  /* Create a 3 by 3 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 )
);

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

WnMatrix__free( p_my_matrix );

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

  return EXIT_SUCCESS;

}
```