Matrix Class

Generated by Doxygen 1.8.14

Contents

Chapter 1

Description

Templatized Matrix class provides cache friendly 1d aray implementation of Matrix Multiplication and Matrix Transpose. the helper functions include printing the matrix to a file, outputting to terminal, reading from console terminal, populating a Matrix using random numbers, using an array to fill the Matrix etc.,

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 Matrix< T > Class Template Reference

Public Member Functions

- Matrix ()

  constructor which initializes the rows and cols to 0
- Matrix (size_t rows, size_t cols)

  constructor which takes two arguments of type size_t to set the rows and cols of the Matrix
- ~Matrix ()

  destructor for deleting Matrix
- T operator() (size_t i, size_t j) const

  getter function which returns value at the position of 'i'th row and 'j'th column of the Matrix
- void set (size_t i, size_t j, T val)

  setter function to modify the value at the position of 'i'th row and 'j'th column of the Matrix
- T & operator() (size_t i, size_t j)

  getter function which returns value at the position of 'i'th row and 'j'th column of the Matrix
- void fillMatrix (const T lower_bound, const T upper_bound)

  populates a matrix using random numbers between the bounds provided by the user
- void feedArray (T *arr)

  populates a Matrix using the provided array elements in row major order
- void printToFile (ofstream &out) const

  prints the Matrix to the output file ofstream object given by the user
- Matrix transpose () const

  Transposes Matrix without tiling.
- Matrix transposeBlock () const

  Uses tiling/blocking to transpose a Matrix.
- Matrix operator* (const Matrix &B) const

  Uses overloaded operator to perform Matrix Multiplication.*
- Matrix operator= (const Matrix &B)

  Uses overloaded operator= to perform Matrix Assignment.

Public Attributes

- size_t rows

  *Matrix class contains 3 public members, 1.rows, 2.cols and 3. *Mat.*
- size_t cols
- T * Mat

Private Member Functions

- T & operator[ ] (int index) const

  getter function which returns value at the index positon in 1d array sequence of the Matrix

3.1.1 Constructor & Destructor Documentation

3.1.1.1 Matrix() [1/2]

```
template<class T>
Matrix< T >::Matrix ( )  [inline]
```

constructor which initializes the rows and cols to 0

USAGE:

```
Matrix <double> M;
```

creates Matrix of type double with 0 rows and 0 columns

3.1.1.2 Matrix() [2/2]

```
template<class T>
Matrix< T >::Matrix (
            size_t rows,
            size_t cols )  [inline]
```

constructor which takes two arguments of type size_t to set the rows and cols of the Matrix

USAGE:

```
Matrix <double> M(2,3);
```

creates Matrix of type double with 2 rows and 3 columns < rows and cols should be positive numbers

allocate the array size for the respective datatype provided by the user

initialize all the values to zeros

3.1.2 Member Function Documentation

3.1.2.1 feedArray()

```
template<class T>
void Matrix< T >::feedArray (
             T * arr )  [inline]
```

populates a Matrix using the provided array elements in row major order

USAGE:

```
int arr[6] = {1,2,3,4,5,6};
M.feedArray(arr);
```

populates M with the values of arr (which is of the same datatype as M)

3.1.2.2 fillMatrix()

```
template<class T>
void Matrix< T >::fillMatrix (
             const T lower_bound,
             const T upper_bound )  [inline]
```

populates a matrix using random numbers between the bounds provided by the user

USAGE:

```
M.fillMatrix(0.3, 9.1);
```

populates M with values between 0.3 and 9.1 checks if the bounds are nan values

3.1.2.3 operator()() [1/2]

```
template<class T>
T Matrix< T >::operator() (
             size_t i,
             size_t j ) const  [inline]
```

getter function which returns value at the position of 'i'th row and 'j'th column of the Matrix

USAGE:

```
double M23 = M(2,3);
```

double variable M23 gets assigned the value of M(2,3) checks for the validity of i and j values

3.1.2.4 operator()() [2/2]

```
template<class T>
T& Matrix< T >::operator() (
            size_t i,
            size_t j )  [inline]
```

getter function which returns value at the position of 'i'th row and 'j'th column of the Matrix

checks for the validity of i and j values

3.1.2.5 operator∗()

```
template<class T>
Matrix Matrix< T >::operator* (
            const Matrix< T > & B ) const  [inline]
```

Uses overloaded operator∗ to perform Matrix Multiplication.

USAGE:

```
Matrix<double> AB = A*B;
```

Note

The AB matrix initialization should be done as shown above

< using Block Matrix Transpose for faster computation

3.1.2.6 operator=()

```
template<class T>
Matrix Matrix< T >::operator= (
            const Matrix< T > & B )  [inline]
```

Uses overloaded operator= to perform Matrix Assignment.

USAGE:

```
Matrix<double> A = B;
```

Note

The AB matrix initialization should be done as shown above

3.1.2.7 operator[]()

```
template<class T>
T& Matrix< T >::operator[] (
            int index ) const  [inline], [private]
```

getter function which returns value at the index positon in 1d array sequence of the Matrix

checks for the validity of index

3.1.2.8 printToFile()

```
template<class T>
void Matrix< T >::printToFile (
            ofstream & out ) const  [inline]
```

prints the Matrix to the output file ofstream object given by the user

USAGE:

```
ofstream output_file("output.txt");
M.printToFile(output_file);
```

prints the Matrix to a ofstream file object

3.1.2.9 set()

```
template<class T>
void Matrix< T >::set (
            size_t i,
            size_t j,
            T val )  [inline]
```

setter function to modify the value at the position of 'i'th row and 'j'th column of the Matrix

USAGE:

```
M.set(2,3,9.5);
```

the value at M(2,3) gets changed to 9.5 checks for the validity of i and j values

3.1.2.10 transpose()

```
template<class T>
Matrix Matrix< T >::transpose ( ) const  [inline]
```

Transposes Matrix without tiling.

USAGE:

```
Matrix<double> Mtrans = M.transpose();
```

using the regular transpose

```
Matrix<double> Mtrans = M.transposeBlock();
```

using the block transpose

Note

The Mtrans matrix initialization should be done as shown above

3.1.2.11 transposeBlock()

```
template<class T>
Matrix Matrix< T >::transposeBlock ( ) const  [inline]
```

Uses tiling/blocking to transpose a Matrix.

USAGE:

```
Matrix<double> Mtrans = M.transposeBlock();
```

using the block transpose

Note

The Mtrans matrix initialization should be done as shown above

block size 32 has been used

3.1.3 Member Data Documentation

3.1.3.1 cols

```
template<class T>
size_t Matrix< T >::cols
```

1. cols (columns of the Matrix)

3.1.3.2 Mat

```
template<class T>
T* Matrix< T >::Mat
```

1. ∗Mat (pointer to the 1d array, storing the matrix elements)

3.1.3.3 rows

```
template<class T>
size_t Matrix< T >::rows
```

Matrix class contains 3 public members, 1.rows, 2.cols and 3. ∗Mat.

1. rows (rows of the Matrix)

The documentation for this class was generated from the following file:

- /home/anoop/Documents/brain_corporation/prob2_Matrix/Transpose/src/Matrix.h

Index