Morpheus

Generated by Doxygen 1.6.1

Tue Aug 2 12:51:39 2016

Contents

Chapter 1

Welcome to Morpheus!

Morpheus is an exciting new linear algebra package. It currently contains a dense matrix and dense vector class, but we hope to expand its features as it grows in popularity.

Okay, so Morpheus is really just a test project for an ATPESC tutorial...but at least it has a cool name!

1.1 gcov tutorial

1. Please log in to vesta

   `ssh <username>@vesta.alcf.anl.gov`

2. Clone the Morpheus repository

   `git clone https://github.com/amklinv/morpheus.git`

3. Build the tests

   `cd morpheus`

   `make`

   Note that the proper coverage flags have been added to your makefile. Also note that this step generates a set of .gcno files for you.

4. Run the tests

   `./runtests`

   `runtests` is a perl script which runs the three tests for you. This step generates the .gcda files.

5. Run gcov on the Morpheus source code

   `gcov *.cpp`

   Ignore the system files; we are not responsible for testing them. This will generate our .gcov files

6. Examine Morpheus_Vector.cpp.gcov and Morpheus_Matrix.cpp.gcov

   These are regular text files, so you may use your text editor of choice (vim, emacs, eclipse...or you can just `cat` the file). Lines that have been tested are marked by the number of times they were executed. Lines that have NOT been tested are preceeded by `#####`. Dashes denote lines that contain no instructions, such as blank lines or curly braces.

   Example:

```
     5:   85:bool Matrix::isSymmetric() const
     -:   86:{
     5:   87:  if(nrows_ != ncols_)
 #####:   88:    return false;
     -:   89:
    30:   90:  for(int r=0; r<nrows_; r++)
     -:   91:  {
   150:   92:    for(int c=0; c<ncols_; c++)
     -:   93:    {
   125:   94:      if(data_[r][c] != data_[c][r])
 #####:   95:        return false;
     -:   96:    }
     -:   97:  }
     -:   98:
     5:   99:  return true;
     -:  100:}
```

1.1.1 Food for thought..

- What percentage of Morpheus_Vector.cpp and Morpheus_Matrix.cpp did gcov report was being tested?
- How much confidence do you have that my code is correct?
- If we had 100% code coverage, would that mean there were no bugs? Why?
- Which of the following Vector functions are tested?
  - Constructor
  - Destructor
  - Subscript operator
  - Const subscript operator
  - getNumElements
  - setValue
  - scale
  - add
  - dot
  - norm1
  - normInf
  - norm2
- Are the two tests that exist for the Vector class good?
- What types of data did I ignore?
- What kinds of errors could occur as a result of me ignoring those types of data?
- For which of the following vectors would the 1-norm function produce the correct result? The infinity-norm? The 2-norm?

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

- Which of the following Matrix functions are tested?

  - Constructor
  - Destructor
  - Subscript operator
  - getNumRows
  - getNumCols
  - getNumEntries
  - Matrix-vector multiply
  - Matrix-matrix multiply
  - isSymmetric
  - isUpperTriangular
  - approxEqual
  - norm1
  - normInf
  - print

- Is it sufficient to test the isUpperTriangular function with the identity matrix? Why?
- If my tests failed, would it be easy to track down the source of the problem? What could I do to make it easier?

1.2 Doxygen

To create the html pages you're currently looking at, all you have to do is type `doxygen` in the source directory. (Doxygen is already installed on Vesta.) It reads `Doxyfile`, which I generated with `doxywizard` on my workstation and checked into the repository. Alternatively, you can generate such files by hand.

1.2.1 Didn't you say Doxygen can generate LaTeX manuals?

Yes. You're looking at it.

1.3 What about the beautiful GUIs you showed us?

`lcov` and `doxywizard` are great tools for a personal workstation, but not-so-great tools for computing clusters. I personally prefer using the GUIs, but it's important to know how to use `gcov` and `doxygen` too, since they're available on more systems. If you desperately need the HTML files generated by `lcov`, they are available here.

Chapter 2

Todo List

Namespace Morpheus Add a sparse matrix class

Class Morpheus::Matrix Add a function for computing the Frobenius norm

Add a function for computing the 2-norm

Add a function for reading a matrix from a file

Member Morpheus::Matrix::multiply(const Vector &X, Vector &Y) const Write a test for this function

Class Morpheus::Vector Consider whether Vector should be a subclass of Matrix

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 Morpheus::Matrix Class Reference

Stores a dense matrix.

`#include <Morpheus_Matrix.h>`

Public Member Functions

Constructors and destructors

- Matrix (const int nrows, const int ncols)
  Constructor.
- ~Matrix ()
  Destructor.

Accessor functions

- double & operator() (const int row, const int col)
  Accesses a single entry of the matrix.
- int getNumRows () const
  Returns the number of rows.
- int getNumCols () const
  Returns the number of columns.
- int getNumEntries () const
  Returns the number of entries.

Multiplication routines

- void multiply (const Vector &X, Vector &Y) const
  Computes a matrix-vector multiplication.

- void multiply (const Matrix &X, Matrix &Y) const
  Computes a matrix-matrix multiplication.

Matrix property query methods

- bool isSymmetric () const
  Determines whether the matrix is symmetric.
- bool isUpperTriangular () const
  Determines whether the matrix is upper triangular.
- bool approxEqual (const Matrix &m, const double tol) const
  Determines whether this matrix is approximately equal to another matrix.

Norms

- double norm1 () const
  Maximum absolute column sum.
- double normInf () const
  Maximum absolute row sum.

I/O functions

- void print () const
  Prints matrix to console.

Private Attributes

- int nrows_
  Number of rows.
- int ncols_
  Number of columns.
- double ** data_
  Pointer to raw data.

5.1.1 Detailed Description

Stores a dense matrix.

Todo

Add a function for computing the Frobenius norm
Add a function for computing the 2-norm
Add a function for reading a matrix from a file

Examples:

Morpheus_Matrix_Tests.cpp.

Definition at line 25 of file Morpheus_Matrix.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Morpheus::Matrix::Matrix (const int *nrows*, const int *ncols*)

Constructor. Allocates memory for a dense matrix. If either nrows or ncols is not positive, the program terminates.

Parameters:

← ***nrows*** Number of rows

← ***ncols*** Number of columns

Definition at line 15 of file Morpheus_Matrix.cpp.

5.1.2.2 Morpheus::Matrix::∼Matrix ()

Destructor. Deallocates memory allocated in the constructor

Definition at line 31 of file Morpheus_Matrix.cpp.

5.1.3 Member Function Documentation

5.1.3.1 double & Morpheus::Matrix::operator() (const int *row*, const int *col*)

Accesses a single entry of the matrix.

Parameters:

← ***row*** Row

→ ***col*** Column

Note:

This is 0-based, not 1-based indexing

Usage:

```
Matrix m(4,3);
m(0,0) = 1; m(0,1) = 0; m(0,2) = 0;
m(1,0) = 0; m(1,1) = 1; m(1,2) = 0;
m(2,0) = 0; m(2,1) = 0; m(2,2) = 1;
m(3,0) = 0; m(3,1) = 0; m(3,2) = 0;
```

creates the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Definition at line 41 of file Morpheus_Matrix.cpp.

5.1.3.2 int Morpheus::Matrix::getNumRows () const

Returns the number of rows.

Definition at line 159 of file Morpheus_Matrix.cpp.

5.1.3.3 int Morpheus::Matrix::getNumCols () const

Returns the number of columns.

Definition at line 165 of file Morpheus_Matrix.cpp.

5.1.3.4 int Morpheus::Matrix::getNumEntries () const

Returns the number of entries.

Definition at line 171 of file Morpheus_Matrix.cpp.

5.1.3.5 void Morpheus::Matrix::multiply (const Vector & *X*, Vector & *Y*) const

Computes a matrix-vector multiplication.

Parameters:

$\leftarrow$ ***X*** vector to be multiplied

$\rightarrow$ ***Y*** result of multiplication

Note:

This function does not allocate memory for *Y*; it only fills in the values. The number of rows of the matrix must equal the number of entries in *Y*. The number of columns of the matrix must equal the number of entries in *X*. Otherwise, the program will terminate.

Todo

Write a test for this function

Examples:

Morpheus_Matrix_Tests.cpp.

Definition at line 47 of file Morpheus_Matrix.cpp.

5.1.3.6 void Morpheus::Matrix::multiply (const Matrix & *X*, Matrix & *Y*) const

Computes a matrix-matrix multiplication.

Parameters:

$\leftarrow$ ***X*** matrix to be multiplied

$\rightarrow$ ***Y*** result of multiplication

Note:

This function does not allocate memory for *Y*; it only fills in the values. The number of rows of the calling matrix must equal the number of rows of *Y*. The number of columns of the calling matrix must equal the number of rows of *X*. *X* and *Y* must have the same number of columns. Otherwise, the program will terminate.

Definition at line 64 of file Morpheus_Matrix.cpp.

5.1.3.7 bool Morpheus::Matrix::isSymmetric () const

Determines whether the matrix is symmetric.

Note:

The symmetry is not stored as a property of the matrix. Every time this function is called, it will perform the O(n^2) comparison of entries.

Examples:

Morpheus_Matrix_Tests.cpp.

Definition at line 85 of file Morpheus_Matrix.cpp.

5.1.3.8 bool Morpheus::Matrix::isUpperTriangular () const

Determines whether the matrix is upper triangular.

Note:

This is not stored as a property of the matrix. Every time this function is called, it will perform the O(n^2) comparison of entries.

Examples:

Morpheus_Matrix_Tests.cpp.

Definition at line 103 of file Morpheus_Matrix.cpp.

5.1.3.9 bool Morpheus::Matrix::approxEqual (const Matrix & *m*, const double *tol*) const

Determines whether this matrix is approximately equal to another matrix. Returns true if

- *this* and *m* are the same size
- *this(r,c)* == *m(r,c)* for all *r*, *c*

Parameters:

← ***m*** The matrix to be compared
← ***tol*** The tolerance of the comparison

Examples:

Morpheus_Matrix_Tests.cpp.

Definition at line 177 of file Morpheus_Matrix.cpp.

5.1.3.10 double Morpheus::Matrix::norm1 () const

Maximum absolute column sum.

Definition at line 122 of file Morpheus_Matrix.cpp.

5.1.3.11 double Morpheus::Matrix::normInf () const

Maximum absolute row sum.

Definition at line 141 of file Morpheus_Matrix.cpp.

5.1.3.12 void Morpheus::Matrix::print () const

Prints matrix to console. Example:

```
4x3 Matrix
1 0 0
0 1 0
0 0 1
0 0 0
```

Definition at line 194 of file Morpheus_Matrix.cpp.

5.1.4 Member Data Documentation

5.1.4.1 int Morpheus::Matrix::nrows_ [private]

Number of rows.

Definition at line 170 of file Morpheus_Matrix.h.

5.1.4.2 int Morpheus::Matrix::ncols_ [private]

Number of columns.

Definition at line 172 of file Morpheus_Matrix.h.

5.1.4.3 double** Morpheus::Matrix::data_ [private]

Pointer to raw data.

Definition at line 174 of file Morpheus_Matrix.h.

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

- Morpheus_Matrix.h
- Morpheus_Matrix.cpp

5.2 Morpheus::Vector Class Reference

Stores a dense vector.

`#include <Morpheus_Vector.h>`

Public Member Functions

Constructors and destructors

- Vector (const int numElements)
  Constructor.
- ~Vector ()
  Destructor.

Accessor functions

- double & operator[ ] (const int subscript)
  Subscript operator.
- const double & operator[ ] (const int subscript) const
  Const subscript operator.
- int getNumElements () const
  Returns the total number of entries.

Linear algebra functions

- void setValue (const double alpha=0)
  Initializes all entries to alpha.
- void scale (const double alpha)
  Scales the vector.
- void add (const Vector &b, Vector &sum) const
  Vector addition
- double dot (const Vector &b) const
  Dot product.

Norms

- double norm1 () const
  Sum of all entries.
- double normInf () const
  Maximum magnitude entry.
- double norm2 () const
  Length of vector.

I/O functions

- void print () const
  Prints vector to console.

Private Attributes

- int numElements_
  Number of elements in the vector.
- double * data_
  Pointer to raw data.

5.2.1 Detailed Description

Stores a dense vector.

Todo

Consider whether Vector should be a subclass of Matrix

Examples:

Morpheus_Vector_addScaleTest.cpp, and Morpheus_Vector_normTest.cpp.

Definition at line 30 of file Morpheus_Vector.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Morpheus::Vector::Vector (const int *numElements*)

Constructor. Allocates memory for a Vector. If *numElements* is not positive, the program terminates.

Parameters:

← ***numElements*** The number of entries in the vector

Warning:

This function only allocates the memory; it does not initialize the memory.

Definition at line 15 of file Morpheus_Vector.cpp.

5.2.2.2 Morpheus::Vector::~Vector ()

Destructor. Deallocates memory for a Vector

Definition at line 26 of file Morpheus_Vector.cpp.

5.2.3 Member Function Documentation

5.2.3.1 double & Morpheus::Vector::operator[ ] (const int *subscript*)

Subscript operator. Returns a reference to the entry at the location denoted by *subscript*. Example usage:

```
Vector v(3);
v[0] = 1;
v[1] = 0;
v[2] = 0;
```

Parameters:

← ***subscript*** The subscript of interest

Definition at line 33 of file Morpheus_Vector.cpp.

5.2.3.2 const double & Morpheus::Vector::operator[ ] (const int *subscript*) const

Const subscript operator. Returns a const reference to the entry at the location denoted by *subscript*. Example usage:

```
Vector v(3);
v.setValue();
double entry = v[0];
```

Parameters:

← ***subscript*** The subscript of interest

Definition at line 43 of file Morpheus_Vector.cpp.

5.2.3.3 int Morpheus::Vector::getNumElements () const

Returns the total number of entries.

Definition at line 53 of file Morpheus_Vector.cpp.

5.2.3.4 void Morpheus::Vector::setValue (const double *alpha* = 0)

Initializes all entries to *alpha*.

Parameters:

← ***alpha*** The number all entries are set equal to. Default: 0

Definition at line 59 of file Morpheus_Vector.cpp.

5.2.3.5 void Morpheus::Vector::scale (const double *alpha*)

Scales the vector. Every entry in the vector is multiplied by *alpha*

Parameters:

← ***alpha*** The number to scale by

Examples:

Morpheus_Vector_addScaleTest.cpp.

Definition at line 68 of file Morpheus_Vector.cpp.

5.2.3.6 void Morpheus::Vector::add (const Vector & *b*, Vector & *sum*) const

Vector addition Replaces *sum* by *this* + *b*.

Parameters:

← ***b*** Vector to add

→ ***sum*** Sum vector

Note:

This function does not allocate memory; it fills in the existing values of *sum*. If *this*, *b*, and *sum* are not the same size, the function will abort.

Examples:

Morpheus_Vector_addScaleTest.cpp.

Definition at line 77 of file Morpheus_Vector.cpp.

5.2.3.7 double Morpheus::Vector::dot (const Vector & *b*) const

Dot product. If *this* and *b* are not the same size, this function will abort

Parameters:

← ***b*** Vector to use in dot-product

Definition at line 91 of file Morpheus_Vector.cpp.

5.2.3.8 double Morpheus::Vector::norm1 () const

Sum of all entries.

Definition at line 108 of file Morpheus_Vector.cpp.

5.2.3.9 double Morpheus::Vector::normInf () const

Maximum magnitude entry.

Definition at line 122 of file Morpheus_Vector.cpp.

5.2.3.10 double Morpheus::Vector::norm2 () const

Length of vector.

Examples:

Morpheus_Vector_addScaleTest.cpp.

Definition at line 140 of file Morpheus_Vector.cpp.

5.2.3.11 void Morpheus::Vector::print () const

Prints vector to console. Example:

```
Vector with 3 entries
data[0] = 0
data[1] = 0
data[2] = 7
```

Definition at line 154 of file Morpheus_Vector.cpp.

5.2.4 Member Data Documentation

5.2.4.1 int Morpheus::Vector::numElements_ [private]

Number of elements in the vector. Cannot be changed after construction

Definition at line 158 of file Morpheus_Vector.h.

5.2.4.2 double* Morpheus::Vector::data_ [private]

Pointer to raw data. Allocated in the constructor and deallocated in the destructor.

Definition at line 163 of file Morpheus_Vector.h.

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

- Morpheus_Vector.h
- Morpheus_Vector.cpp

Chapter 6

File Documentation

6.1 Morpheus_Matrix.cpp File Reference

Defines a Matrix class. `#include "Morpheus_Matrix.h"`

`#include <cassert>`

`#include <cmath>`

`#include <iostream>`

6.1.1 Detailed Description

Defines a Matrix class.

Author:

Alicia Klinvex

Definition in file Morpheus_Matrix.cpp.

6.2 Morpheus_Matrix.h File Reference

Defines a Matrix class. `#include "Morpheus_Vector.h"`

Classes

- class Morpheus::Matrix

  Stores a dense matrix.

6.2.1 Detailed Description

Defines a Matrix class.

Author:

Alicia Klinvex

Definition in file Morpheus_Matrix.h.

6.3 Morpheus_Vector.cpp File Reference

Defines a Vector class. `#include <iostream>`

`#include <cassert>`

`#include <cmath>`

`#include "Morpheus_Vector.h"`

6.3.1 Detailed Description

Defines a Vector class.

Author:

Alicia Klinvex

Definition in file Morpheus_Vector.cpp.

6.4 Morpheus_Vector.h File Reference

Defines a Vector class.

Classes

- class Morpheus::Vector

  Stores a dense vector.

6.4.1 Detailed Description

Defines a Vector class.

Author:

Alicia Klinvex

Definition in file Morpheus_Vector.h.

Chapter 7

Example Documentation

7.1 Morpheus_Matrix_Tests.cpp

Demonstrates the usage of the matrix class

```
/*
 * Morpheus_Matrix_Tests.cpp
 *
 *  Created on: Jul 28, 2016
 *      Author: amklinv
 */

#include "Morpheus_Matrix.h"
#include <time.h>
#include <stdlib.h>
#include <iostream>

int main()
{
  bool testPassed = true;

  // Seed the random number generator
  srand(time(NULL));

  // Create a random matrix
  Morpheus::Matrix randMat(5,5);
  for(int r=0; r<5; r++)
  {
    for(int c=0; c<5; c++)
    {
      randMat(r,c) = rand();
    }
  }

  // Create an identity matrix
  Morpheus::Matrix eye(5,5);
  for(int r=0; r<5; r++)
  {
    for(int c=0; c<5; c++)
    {
      if(r == c)
        eye(r,c) = 1;
      else
        eye(r,c) = 0;
    }
  }
```

```
  // Multiply the two
  Morpheus::Matrix result(5,5);
  randMat.multiply(eye,result);

  // Assert that randMat is the same as the result
  if(!randMat.approxEqual(result,1e-10))
  {
    std::cout << "ERROR: The matrix product is incorrect\n";
    testPassed = false;
  }

  if(!eye.isSymmetric())
  {
    std::cout << "ERROR: The identity matrix should be symmetric\n";
    testPassed = false;
  }

  if(!eye.isUpperTriangular())
  {
    std::cout << "ERROR: The identity matrix should be upper triangular\n";
    testPassed = false;
  }

  if(testPassed)
    std::cout << "Matrix test: PASSED!\n";
  else
    std::cout << "Matrix test: FAILED!\n";

}
```

7.2 Morpheus_Vector_addScaleTest.cpp

Demonstrates the use of the vector addition and vector scale functions

```
/*
 * Morpheus_Vector_addScaleTest.cpp
 *
 *  Created on: Jul 28, 2016
 *      Author: amklinv
 */

#include "Morpheus_Vector.h"
#include <cmath>
#include <iostream>
#include <stdlib.h>
#include <time.h>

int main()
{
  bool testPassed = true;
  int numEntries = 10;

  // Seed the random number generator
  srand(time(NULL));

  // Create three vectors, all with 10 entries
  Morpheus::Vector vecA(numEntries);
  Morpheus::Vector vecB(numEntries);
  Morpheus::Vector vecC(numEntries);

  // Set the entries in vecA
  // Each entry is random
  for(int i=0; i<numEntries; i++) {
    vecA[i] = rand();
  }

  // Copy the entries of vecA to vecB
  for(int i=0; i<numEntries; i++) {
    vecB[i] = vecA[i];
  }

  // Scale vecB by -1
  vecB.scale(-1);

  // Compute vecC = vecA + vecB
  vecA.add(vecB, vecC);

  // vecC should be all 0s, so its norm should be too
  double norm2 = vecC.norm2();

  if(norm2 > 1e-10)
  {
    std::cout << "ERROR: C must be 0\n";
    testPassed = false;
  }

  if(testPassed)
    std::cout << "Add/scale test: PASSED!\n";
  else
    std::cout << "Add/scale test: FAILED!\n";
}
```

7.3 Morpheus_Vector_normTest.cpp

Demonstrates the use of the vector norm functions

```
/*
 * Morpheus_Vector_normTest.cpp
 *
 *  Created on: Jul 28, 2016
 *      Author: amklinv
 */

#include "Morpheus_Vector.h"
#include <cmath>
#include <iostream>

// Returns true if | a-b | < tol, false otherwise
bool approxEqual(double a, double b, double tol);

int main()
{
  bool testPassed = true;
  int numEntries = 5;

  // Create a vector with 10 entries
  Morpheus::Vector vec(numEntries);

  // Set the entries in this vector
  // Each entry is 1/sqrt(n)
  double rootN = std::sqrt(numEntries);
  double invRootN = 1./rootN;
  for(int i=0; i<numEntries; i++) {
    vec[i] = invRootN;
  }

  // 1-norm should be sqrt(n)
  // infinity-norm should be 1/sqrt(n)
  // 2-norm should be 1
  double norm1 = vec.norm1();
  double normInf = vec.normInf();
  double norm2 = vec.norm2();

  if(!approxEqual(norm1, sqrt(numEntries), 1e-10))
  {
    std::cout << "ERROR: The 1-norm is incorrect\n";
    testPassed = false;
  }
  if(!approxEqual(normInf, invRootN, 1e-10))
  {
    std::cout << "ERROR: The infinity-norm is incorrect\n";
    testPassed = false;
  }
  if(!approxEqual(norm2, 1, 1e-10))
  {
    std::cout << "ERROR: The 2-norm is incorrect\n";
    testPassed = false;
  }

  if(testPassed)
    std::cout << "Norm test: PASSED!\n";
  else
    std::cout << "Norm test: FAILED!\n";
}


// Returns true if | a-b | < tol, false otherwise
bool approxEqual(double a, double b, double tol)
```

```
{
  if(std::abs(a-b) < tol)
    return true;
  return false;
}
```