MeshFree++

A C++ class that provides a unified meshfree basis function computation

User's Reference Manual

Version 1.0

September 23, 2009

MeshFree++: A C++ class that provides a unified meshfree basis function computation

Releases

Code version	Description	Date
1.0	Code released	09/23/09

Copyright and License

TABLE OF CONTENTS

1 Scope of the manual

The aim of this manual is to explain how to use MeshFree++. I assume that you have some background on meshfree methods. This is the first version of the meshfree class and its capabilities include the computation of first order maximum entropy (maxent) basis functions [1,2,3,4] and first order moving least square (mls) basis functions [5,6] along with their derivatives. It is a generic code in the sense that it can compute the aforementioned basis functions in one, two and three dimensions with just one code. If you need more details on meshfree methods, see additional References [7,8].

The complete source code of MeshFree++ can be downloaded from http://www.computationalmechanics.org/~aortizb/codes.

2 Building MeshFree++

This manual explains how to build MeshFree++ under Windows® and Linux/Unix. MeshFree++ consists of two files available in 'src' folder: meshfree.cpp and meshfree.h. It has been written using the matrix-vector template MathVec++ 1.0 which is the header file 'mathvec.h' located in 'include\mathvec' folder. MathVec++ uses TAUCS[1] library which needs to be linked. I have provided precompiled TAUCS libraries along with ATLAS libraries used by TAUCS, for both Windows® and Linux in folder 'libs\taucs'. You do not need to build them again unless you want optimized libraries for your system. If you are in need of information on building TAUCS, you may find more information at TAUCS manual and http://matrixprogramming.com/.

2.1 Building MeshFree++ under Windows

If you decide to build MeshFree++ in Windows®, I highly recommend using Microsoft Visual Studio®. There is an express edition available at Microsoft® web site for free. Also, Professional edition of MVS is free for students. I have provided the MVS 2008 project for the Windows version of MeshFree++ in this distribution. Unless you use other compiler there is nothing more to do. See MeshFree++ project properties in the MVS 2008 project to find out includes and linking options to precompiled TAUCS and ATLAS libraries. Note that runtime library under C-C++ -> Code Generation has been specified to /MTd which is how precompiled TAUCS libraries were built. Also, LIBCMT library in Linker -> Input has been ignored since it conflicts with TAUCS libraries. If you need to rebuild TAUCS libraries in MVS, please see http://matrixprogramming.com/ for details. Then link them considering the observations provided above. Linking TAUCS libraries to MeshFree++ in a compiler other than MVS might require rebuilding of TAUCS libraries.

Once the MVS project is in your machine, then you can try the meshfree example I have included in 'src/main.cpp'. You just need to build the solution in the MVS GUI: Build -> Build solution or by pressing F7. If you want to try a different example, the only thing you need to do is to modify the code in main.cpp and rebuilt it. When you build or rebuild the code, an executable file called 'MeshFree++' is placed in 'bin' folder. You have to run that executable from command line (cmd). This will print the meshfree computation to the screen. Also, a file called 'meshfreeresults.txt' is placed in 'bin' folder which contains the same information that was printed to the screen. Additionally, if needed, there will be a file called 'messages.txt' which contains all warnings and errors that appeared during the meshfree computation. Alternatively, you may want to just execute 'MeshFree++' by double-clicking on it. This will create the 'meshfreeresults.txt' and 'messages.txt' files.

There is another alternative to install MeshFree++ under Win32. For example, you may use gcc compiler for Windows®, such as MinGW or Cygwin. Though, I will not cover these options here since I have not tried a compiler other than MVS and it might require the rebuilding of TAUCS libraries with gcc. If you need to do it, please download the source files from TAUCS web site and consult TAUCS manual for more information.

[1] TAUCS, A Library of Sparse Linear Solvers is used by permission of Sivan Toledo. Original files of TAUCS are available at http://www.tau.ac.il/~stoledo/taucs/.

2.2 Building MeshFree++ under Linux/Unix

Building MeshFree++ under Linux/Unix might be the easiest way. Once you have the Linux/Unix version of MeshFree++ distribution in your machine, you just need to type 'make' or 'make MeshFree++' in the main folder of MeshFree++ distribution. You may find all the options in the makefiles that are provided in the main folder and in 'src' folder. You need to modify the paths to TAUCS and ATLAS libraries if you want to use your own libraries. Instructions are provided in the makefile of 'src' folder.

If you need to rebuild TAUCS libraries, download the source code from TAUCS web site and read TAUCS manual for instructions. Building of TAUCS libraries in Linux/Unix is quite straightforward.

3 Using MeshFree++

The meshfree class of MeshFree++ version 1.0 consists of the following:

A parent class:

- MeshFree class

Two child classes:

- Maxent class
- Mls class

Parent class MeshFree contains all the information that is common to meshfree basis functions construction. Maxent class contains information that is particular to maximum entropy basis functions. Similarly, Mls class contains information that is particular to moving least squares basis functions. Using MeshFree++ is quite easy. You need to construct Maxent or Mls objects from the available constructors for both child classes. You can only run a meshfree computation form the child classes. You may attempt to create an object of the parent class MeshFree. The latter will not run any meshfree computation as the parent class has not any method to do so. Thus, unless you are implementing a new child class, you must always work with the child classes to compute meshfree basis functions. All the data you may obtain from the meshfree class after the meshfree basis functions construction will be available through an object of a child class using public methods of the parent class MeshFree. The only constructors for each of these child classes are:

```
Maxent(int dim, int nNodes, const Mat<double>& nodeCoord,
       const Vec<double>& xSample, const Vec<double>& nodeSpacing,
       const string& weightName, double gamma, int order, int compute,
       bool autoIncreaseSupportSize, bool forceConsistencyCheck,
       bool variousSamplePoints, bool printAllResults,
       double rtol, int maxIter);

Mls(int dim, int nNodes, const Mat<double>& nodeCoord,
    const Vec<double>& xSample, const Vec<double>& nodeSpacing,
    const string& weightName, double gamma, int order, int compute,
    bool autoIncreaseSupportSize, bool forceConsistencyCheck,
    bool variousSamplePoints, bool printAllResults);
```

Note that objects 'Vec' and 'Mat' are defined in MathVec++ template. For details see 'math-vec.h' located in folder 'include\mathvec'. A preview of MathVec++ capabilities and its usage is provided in Section 4 of this manual. The parameters of the above constructors stand for:

```
dim                     =   Problem dimension.

nNodes                  =   Total number of nodes in the domain.

nodeCoord               =   Nodal coordinates.
```

```
xSample                   =   Sample (evaluation) point.

nodeSpacing               =   Characteristic nodal spacing.

weightName                =   Name of the weight (kernel) function. Only
                              radial kernels are used: cubic, quartic,
                              gaussian or constant.
gamma                     =   Support size parameter.

order                     =   Order of the meshfree approximation.

compute                   =   Specify if basis functions and their de-
                              rivatives will be computed (1, for basis
                              function only; and 2 for both).

autoIncreaseSupportSize   =   If true, support radius will be automati-
                              cally and repeatedly increased by 10% until
                              sufficient number of neighbors are avail-
                              able for evaluation at xSample.

forceConsistencyCheck     =   If true, it will force PU consistency check
                              of computed basis functions and their de-
                              rivatives, even if not needed.

variousSamplePoints       =   If true, it will append all output data to
                              previous output file. This might be useful
                              when meshfree computation at various sample
                              (evaluation) points is needed.

printAllResults           =   If true, it will print all meshfree compu-
                              tations to the screen and to output file.
                              If false, it will print to the screen and
                              to output file only if a PU consistency
                              check is required/needed.

rtol                      =   Required Newton's tolerance for maxent com-
                              putation.

maxIter                   =   Maximum allowed Newton's iterations in max-
                              ent computation.
```

Once you have created the Maxent or Mls object, you just need to compute the meshfree basis functions with the following public method:

```
object.computeMeshfree();
```

Apart from the output files, you may also retrieve the basis functions, basis functions derivatives, list of contributing (neighbor) nodes and the number of contributing (neighbor) nodes with the following public methods:

```
object.phi();           // vector of doubles containing the meshfree basis
                        // functions evaluated at xSample

object.phiDer();        // matrix of doubles containing the meshfree basis
                        // functions evaluated at xSample

object.contribute();    // vector of integers containing the indices of the
                        // contributing (neighbor) nodes at xSample

object.length();        // integer which is the length of contribute, i.e.,
                        // the number of contributing nodes
```

or entry by entry as

```
object.phi(ind);            // a double
```

```
object.phiDer(ind_1,ind_2); // a double

object.contribute(ind);     // a double
```

where ind, ind_1 and ind_2 are 0-based indices to the entries in the vector and matrices. A complete catalog of public methods to retrieve information from the meshfree class is given in Section 3.3.

3.1 MAXENT example

The following example can be found in 'main.cpp' located in 'src' folder. You just need to code there if you want to vary the constructor parameters. You can also put 'main.cpp' (or even give it a different name) outside that folder provided that you compile and link the source code accordingly. For instance, the available makefile for Linux/Unix has been written for 'main.cpp' located inside 'src' folder.

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

#include "meshfree.h"

int main()
{
  /*********************************************************
  * EXAMPLE OF MAXENT COMPUTATION USING THE MESHFREE CLASS *
  *********************************************************/
  int dim = 3;                      // problem dimension, must be <= 3
  int nNodes = 8;                   // total number of nodes in the domain
  string weightName = "cubic";      // name of the weight function (prior in
                                    // maxent context): "cubic", "quartic",
                                    // "gaussian" or "constant"
  double gamma = 3.0;               // support size parameter. Typical values: 1
                                    // to 8 (gaussian), 1 to 3 (quartic or cubic)
  double rtol = 1e-8;               // required tolerance for newton method in
                                    // maxent computation
  int maxIter = 20;                 // maximum allowed number of newton's
                                    // iterations
  int compute = 2;                  // =1 for basis functions, =2 for both basis
                                    // functions and their derivatives
  bool printAllResults = 1;         // =1 plot a summary of the maxent
                                    // computation, =0 nothing is plotted but in
                                    // case of maxent failure
  int order = 1;                    // order of the approximation
  bool forceConsistencyCheck = 1;   // =0, don't run consistency check if not
                                    // needed. =1, run consistency check even if
                                    // not needed
  bool autoIncreaseSupportSize = 1; // =1, auto increase support size when there
                                    // are insufficient neighbors. =0, otherwise

  Mat<double> nodeCoord(8,3);       // nodal coordinates
  nodeCoord(0,0) = 0.0; nodeCoord(0,1) = 0.0; nodeCoord(0,2) = 0.0;
  nodeCoord(1,0) = 1.0; nodeCoord(1,1) = 0.0; nodeCoord(1,2) = 0.0;
  nodeCoord(2,0) = 1.0; nodeCoord(2,1) = 1.0; nodeCoord(2,2) = 0.0;
  nodeCoord(3,0) = 0.0; nodeCoord(3,1) = 1.0; nodeCoord(3,2) = 0.0;
  nodeCoord(4,0) = 0.0; nodeCoord(4,1) = 0.0; nodeCoord(4,2) = 1.0;
  nodeCoord(5,0) = 1.0; nodeCoord(5,1) = 0.0; nodeCoord(5,2) = 1.0;
  nodeCoord(6,0) = 1.0; nodeCoord(6,1) = 1.0; nodeCoord(6,2) = 1.0;
  nodeCoord(7,0) = 0.0; nodeCoord(7,1) = 1.0; nodeCoord(7,2) = 1.0;

  Vec<double> xSample(3);  // sample point
  xSample(0) = 0.25;
  xSample(1) = 0.25;
  xSample(2) = 0.25;

  Vec<double> nodeSpacing(1,0.6); // characteristic nodal spacing. can set
                                  // different values for every node in the domain
                                  // by changing the length of the vector and
                                  // setting up the values of the vector entries
```

```
  bool variousSamplePoints = 0;   // =0, output file will be deleted in a further
                                  // computation
                                  // =1, append information to the output file
                                  // (example: a loop over several evaluation
                                  // points)

  // create a maxent object
  Maxent maxentObj(dim,nNodes,nodeCoord,xSample,nodeSpacing,weightName,gamma,
                   order,compute,autoIncreaseSupportSize,forceConsistencyCheck,
                   variousSamplePoints,printAllResults,rtol,maxIter);

  // start maxent computation
  maxentObj.computeMeshfree();

  // retrieve what you need from the meshfree class
  // in order to use it in a meshfree code
  cout << "*********************************************************" << endl;
  cout << "                    MAXENT OBJECT                        " << endl;
  cout << "THE FOLLOWING WAS OBTAINED FROM THE MESHFREE PARENT CLASS:" << endl;
  cout << "*********************************************************" << endl;
  cout << "BASIS FUNCTIONS:" << endl;
  Vec<double> phi = maxentObj.phi(); // type must be DOUBLE
  cout << phi << endl << endl;

  cout << "GRADIENT OF BASIS FUNCTIONS:" << endl;
  Mat<double> phiDer = maxentObj.phiDer(); // type must be DOUBLE
  cout << phiDer << endl << endl;

  cout << "INDICES OF THE CONTRIBUTING NODES (NEIGHBORS):" << endl;
  Vec<int> contribute = maxentObj.contribute(); // type must be INTEGER
  cout << contribute << endl << endl;

  cout << " NUMBER OF CONTRIBUTING NODES:" << endl;
  int length = maxentObj.length(); // type must be INTEGER
  cout << length << endl;
  //
  return 0;
}
```

3.2 MLS example

The following example can also be found in 'main.cpp' located in 'src' folder.

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

#include "meshfree.h"

int main()
{
  /*********************************************************
  *   EXAMPLE OF MLS COMPUTATION USING THE MESHFREE CLASS  *
  *********************************************************/
  int dim = 3;                      // problem dimension, must be <= 3
  int nNodes = 8;                   // total number of nodes in the domain
  string weightName = "cubic";      // name of the weight function (prior in
                                    // maxent context): "cubic", "quartic",
                                    // "gaussian" or "constant"
  double gamma = 3.0;               // support size parameter. Typical values: 1
                                    // to 8 (gaussian), 1 to 3 (quartic or cubic)
  int compute = 2;                  // =1 for basis functions, =2 for both basis
                                    // functions and their derivatives
  bool printAllResults = 1;         // =1 plot a summary of the maxent
                                    // computation, =0 nothing is plotted but in
                                    // case of maxent failure
  int order = 1;                    // order of the approximation
  bool forceConsistencyCheck = 1;   // =0, don't run consistency check if not
                                    // needed. =1, run consistency check even if
                                    // not needed
```

```
  bool autoIncreaseSupportSize = 1; // =1, auto increase support size when there
                                    // are insufficient neighbors. =0, otherwise

  Mat<double> nodeCoord(8,3);       // nodal coordinates
  nodeCoord(0,0) = 0.0; nodeCoord(0,1) = 0.0; nodeCoord(0,2) = 0.0;
  nodeCoord(1,0) = 1.0; nodeCoord(1,1) = 0.0; nodeCoord(1,2) = 0.0;
  nodeCoord(2,0) = 1.0; nodeCoord(2,1) = 1.0; nodeCoord(2,2) = 0.0;
  nodeCoord(3,0) = 0.0; nodeCoord(3,1) = 1.0; nodeCoord(3,2) = 0.0;
  nodeCoord(4,0) = 0.0; nodeCoord(4,1) = 0.0; nodeCoord(4,2) = 1.0;
  nodeCoord(5,0) = 1.0; nodeCoord(5,1) = 0.0; nodeCoord(5,2) = 1.0;
  nodeCoord(6,0) = 1.0; nodeCoord(6,1) = 1.0; nodeCoord(6,2) = 1.0;
  nodeCoord(7,0) = 0.0; nodeCoord(7,1) = 1.0; nodeCoord(7,2) = 1.0;

  Vec<double> xSample(3);  // sample point
  xSample(0) = 0.25;
  xSample(1) = 0.25;
  xSample(2) = 0.25;

  Vec<double> nodeSpacing(1,0.6); // characteristic nodal spacing. can set
                                  // different values for every node in the domain
                                  // by changing the length of the vector and
                                  // setting up the values of the vector entries
  bool variousSamplePoints = 0;   // =0, output file will be deleted in a further
                                  // computation
                                  // =1, append information to the output file
                                  // (example: a loop over several evaluation
                                  // points)

  // create a mls object
  Mls mlsObj(dim,nNodes,nodeCoord,xSample,nodeSpacing,weightName,gamma,
             order,compute,autoIncreaseSupportSize,forceConsistencyCheck,
             variousSamplePoints,printAllResults);

  // start mls computation
  mlsObj.computeMeshfree();

  // retrieve what you need from the meshfree class
  // in order to use it in a meshfree code
  cout << "*********************************************************" << endl;
  cout << "                        MLS OBJECT                       " << endl;
  cout << "THE FOLLOWING WAS OBTAINED FROM THE MESHFREE PARENT CLASS:" << endl;
  cout << "*********************************************************" << endl;
  cout << "BASIS FUNCTIONS:" << endl;
  phi = mlsObj.phi();
  cout << phi << endl << endl;

  cout << "GRADIENT OF BASIS FUNCTIONS:" << endl;
  phiDer = mlsObj.phiDer();
  cout << phiDer << endl << endl;

  cout << "INDICES OF THE CONTRIBUTING NODES (NEIGHBORS):" << endl;
  contribute = mlsObj.contribute();
  cout << contribute << endl << endl;

  cout << " THE NUMBER OF CONTRIBUTING NODES:" << endl;
  length = mlsObj.length(); // type must be INTEGER
  cout << length << endl;
  //
  return 0;
}
```

3.3 A catalog of public methods to retrieve information from the meshfree class

As a user of the meshfree class, you can run meshfree basis functions computations only from objects of the child classes. After meshfree computation, you may only retrieve data through an object of a child class using public methods of the parent class MeshFree. The following are all public methods in MeshFree class that allow you to obtain computed data. For instance, if you create a Maxent object 'mxt', then you can use any of the public methods listed below as

```
mxt.name_of_the_method( … )
```

	Return:
`const string name() const`	Name of the meshfree type.
`int dim() const`	Problem dimension.
`int nNodes() const`	Total number of nodes.
`double gamma() const`	Support size parameter gamma.
`int order() const`	Order of the approximation.
`int compute() const`	Value of compute.
`bool variousSamplePoints() const`	Value of variousSamplePoints.
`double nodeSpacing(int index) const`	Entry at index (0-based) in characteristic nodal spacing vector.
`const Vec<double> nodeSpacing() const`	Characteristic nodal spacing vector.
`double nodeCoord(int index_i, int index_j) const`	Entry at (index_i,index_j) (0-based) of nodal coordinates matrix.
`double nodeCoordContribute(int index_i, int index_j) const`	Entry at (index_i,index_j) (0-based) of the contributing nodal coordinates matrix.
`const Mat<double> nodeCoord() const`	Nodal coordinates matrix.
`const Mat<double> nodeCoordContribute()const`	Nodal coordinates matrix of contributing nodes.
`double xSample(int index) const`	Entry at index (0-based) of sample (evaluation) point.
`const Vec<double> xSample() const`	Sample (evaluation) point vector.
`double xShift(int index_i, int index_j) const`	Entry at (index_i,index_j) (0-based) of the shifted nodal coordinates matrix.
`double xShiftContribute(int index_i, int index_j) const`	Entry at (index_i,index_j) (0-based) of the shifted nodal coordinate matrix of contributing nodes.
`const Mat<double> xShift() const`	Shifted nodal coordinates matrix.
`const Mat<double> xShiftContribute() const`	Shifted nodal coordinates matrix of contributing nodes.
`double phi(int index) const`	Entry at index (0-based) of basis functions vector.
`const Vec<double> phi() const`	Basis functions vector.
`double phiDer(int index_i, int index_j) const`	Entry at (index_i,index_j) (0-based) of basis function derivatives matrix.
`const Mat<double> phiDer() const`	Basis function derivatives matrix.
`int contribute(int index) const`	Entry at index (0-based) of contribute vector.
`const Vec<int> contribute() const`	Contributing vector containing the list of contributing (neighbor) nodes.
`int length() const`	Number of contributing (neighbor) nodes.
`double weightFun(int index) const`	Entry at index (0-based) of weight (kernel) function vector.
`const Vec<double> weightFun() const`	Vector containing weight (kernel) functions associated to each node.

`double weightFunDer(int index_i, int index_j) const`	Entry at (index_i,index_j) (0-based) of weight (kernel) function derivatives matrix.
`const Mat<double> weightFunDer() const`	Matrix containing weight (kernel) function derivatives associated to each node.

If you are in need of other public methods, please consult the class header 'meshfree.h' and the class definition file 'meshfree.cpp' both located in 'src' folder of this distribution. Every function definition in 'meshfree.cpp' is preceded by a brief description of what the function does.

4 MathVec++ template

MeshFree++ vector and matrices operations are based on MathVec++ 1.0. That means you should also use it if you want to work with this meshfree class. If you elect to use your own matrix-vector class, or any other library or even std::vector class, you need at least to declare the meshfree objects and retrieve the basis functions, derivatives and contributing information with MathVec++ objects. Then, you can use MathVec++ to manipulate the meshfree object throughout your code, or you have to copy all the entries of 'Mat' and 'Vec' objects to your specific matrix and vector objects or arrays. Following, I provide an example code using MathVec++ where you may find some of (not all) its capabilities. If you want to know all the capabilities, see "mathvec.h" in folder 'include/mathvec'.

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

// include the MATRIX-VECTOR template
#include "mathvec.h"

/*****************************************************************************
*         MAIN FUNCTION TO TEST THE MATRIX-VECTOR TEMPLATE MathVec++          *
*****************************************************************************/

int main()
{
  // initiate a vector of three components (all elements are 2)
  Vec<double> x(3,2); // create a vector of size 3 with all entries = 2

  // modifiy components of vector x, so that all are 3
  x(0) = 3;
  x(1) = 3;
  x(2) = 3;

  // multiply a vector by a scalar
  Vec<double> d1 = 2*x;
  Vec<double> d2 = 2.2*x;
  Vec<double> d3 = x*2;
  Vec<double> d4 = x*2.2;

  // divide a vector by a scalar
  Vec<double> c1 = x/2;
  Vec<double> c2 = x/2.2;

  // create an empty vector and use push_back method to fill it
  Vec<double> c3; // create an empty vector which has to be filled with numbers
                  // before doing any other operation.
                  // let's make the following vector: {2.3, 4.7, 8}
  c3.push_back(2.3); // push the first number
  c3.push_back(4.7); // push the second number
  c3.push_back(8);   // push the third number

  // generate a vector of all entries equal to a number
  Vec<double> c4 = c3; // let's copy c3 into c4 (so, c4 = {2.3, 4.7, 8})
  c4 = 7.5; // change all entries of c4 to 7.5

  // substract two vectors
  Vec<double> e1 = x - 2*x;
```

```
// add two vectors
Vec<double> e2 = x + x;

// multiply the current vector by a scalar
Vec<double> y = x;
y *= 4; // can also do: Vec<double> y1 = (y *= 4);

// divide the current vector by a scalar
Vec<double> z = x;
z /= 2.5;

// dot (scalar) product
Vec<double> t(3); // create a vector of size 3 with all entries = default for
                  // type double
t(0) = 2; // modify vector entries
t(1) = 2;
t(2) = 2;
double scalar = x*t; // dot product

// euclidean norm of a vector
Vec<double> k = t;
double enorm = k.norm();

// substract a vector from the current vector
Vec<double> s1(4,5); // create a vector of size 4, all entries = 5
Vec<double> s2(4,6); // create a vector of size 4, all entries = 6
Vec<double> s3 = (s1 -= s2); // or simply s1 -= s2 and then print the value of
                             // s1

// add a vector from the current vector
Vec<double> m1(4,5); // create a vector of size 4, all entries = 5
Vec<double> m2(4,6); // create a vector of size 4, all entries = 6
m1 += m2; // can also do: Vec<double> m3 = (m1 += m2);

// the negative of a vector
Vec<double> m4(4,1); // create a vector of size 4, all entries = 1
Vec<double> m5 = -(m4);

// sum all entries of a vector
Vec<double> m6(5,1); // create a vector of size 5, all entries = 1
m6(2) = 6.2;
double sum1 = m6.sum();

// test begin() and end() iterators with max() and min() functions
Vec<double> m7;
m7.push_back(2.7);
m7.push_back(1.2);
m7.push_back(3.5);
m7.push_back(4.256);
double maxNumber = m7.max();
double minNumber = m7.min();

// declare a matrix
Mat<double> H(3,3,6); // 3x3 matrix, all entries = 6
H(0,0) = 1; // modify
H(0,1) = 0; // to create
H(0,2) = 0; // an
H(1,0) = 0; // identity matrix
H(1,1) = 1;
H(1,2) = 0;
H(2,0) = 0;
H(2,1) = 0;
H(2,2) = 1;

// create an empty matrix and use push_back method to fill it
Mat<double> H1; // create an empty matrix which has to be filled with row
                // vectors
                // before doing any other operation.
                // let's do it for a 3x3 matrix, all entries = 4.7
Vec<double> vec1(3,4.7); // the row vector, all entries = 4.7
H1.push_back(vec1); // push the first row
H1.push_back(vec1); // push the second row
H1.push_back(vec1); // push the third row

// create a matrix filled of a number
```

```
Mat<double> H2 = H1; // let's copy H1 into H2 (so 3x3 matrix, all entries = 4.7)
H2 = 3.9; // let's change all entries of H1 to 3.9

// add two matrices
Mat<double> M(2,2); // 2x2 matrix all entries = default value for type double
Mat<double> N(2,2,2); // 2x2 matrix all entries are = 2
M(0,0) = 4; // modify
M(0,1) = 2; // the entries
M(1,0) = 2; // of
M(1,1) = 4; // the matrix
Mat<double> F1 = M + N;

// substract two matrices
Mat<double> F2 = M - N;

// add a matrix to the current matrix
Mat<double> F3 = (M += N); // or simply M += N and then print the value of M

// substract a matrix to the current matrix
Mat<double> F4 = (N -= N); // or simply N -= N and then print the value of N

// multiply a matrix by a scalar
Mat<double> P(3,3,2); // 3x3 matrix, all entries are = 2
Mat<double> P1 = P * 2.2;

// multiply a scalar by a matrix
Mat<double> P2 = 2.2 * P;

// multiply current matrix by a scalar
Mat<double> P3 = (P *= 3);

// divide a matrix by a scalar
Mat<double> V(3,3,2); // 3x3 matrix, all entries are = 2
Mat<double> V1 = V / 1.5;

// multiply a matrix by a vector
Mat<double> Z(3,2,1);  // 3x2 matrix, all entries are = 1
Vec<double> b(2,3);    // vector of size 2, all entries = 3
Vec<double> b1 = Z*b;

// multiply a vector by a matrix
Mat<double> J = Z.transpose();
Vec<double> b2 = b*J;

// multiply a matrix by a matrix
Mat<double> S1(2,3,2); // 2x3 matrix, all entries = 2
Mat<double> S2(4,2,2); // 4x2 matrix, all entries = 2
Mat<double> S3 = S2 * S1;

// multiply current matrix by a matrix
Mat<double> S4(3,2,2); // 3x2 matrix, all entries = 2
Mat<double> S5(2,3,3); // 2x2 matrix, all entries = 3
Mat<double> S6 = (S4 *= S5); // or simply S4 *= S5 and then print the value of
                             // S4

// the negative of a matrix
Mat<double> S7(3,3,1); // 3x3 matrix, all entries = 1
Mat<double> S8 = -(S7);

// transpose of a matrix
Mat<double> S9(3,3,6); // 3x3 matrix, all entries = 6
S9(2,0) = 1; // change
S9(2,1) = 1; // last row
S9(2,2) = 1; // of S9
Mat<double> S10 = S9.transpose(); // will NOT change S9. it only returns a new
                                  // object

// transform the current matrix into its transpose
Mat<double> T1(3,2,8); // 3x3 matrix, all entries = 8
T1(2,0) = 1; // change
T1(2,1) = 1; // last row
T1.makeTranspose(); // will change T1 and return a reference, so can also do:
                    // Mat<T> T2 = T1.makeTranspose(); in which case both
                    // T2 and T1 will be the transposed matrix

// get a row of a matrix
```

```
Mat<double> D1(3,3,7); // 3x3 matrix, all entries = 7
D1(2,0) = 2;  // change
D1(2,1) = -3; // the third row
D1(2,2) = -6; // of D1
Vec<double> a1 = D1.getRow(3); // the third row. note that the the number of
                               // rows starts from 1, but the index from 0.

// get a column of a matrix
Mat<double> D2(3,3,7); // 3x3 matrix, all entries = 7
D2(0,1) = 1;  // change
D2(1,1) = -4; // the second column
D2(2,1) = -1; // of D2
Vec<double> a2 = D2.getCol(2); // the second row. note that the the number of
                               // rows starts from 1, but the index from 0.

// transform a matrix into a null matrix
Mat<double> D3(3,3,2.4); // 3x3 matrix, all entries = 2.4
D3.makeNull();           // change the entries of D3 and returns a reference,
                         // so can also do: Mat<double> D4 = D3.makeNull();
                         // in which case both D4 and D3 are null matrices.

// transform a matrix into a unit matrix
Mat<double> D4(3,3,6.2); // 3x3 matrix, all entries = 6.2
D4.makeUnit();           // change the entries of D4 and returns a reference,
                         // so can also do: Mat<double> D5 = D4.makeUnit();
                         // in which case both D5 and D4 are unit matrices.

// transform a matrix into an identity matrix
Mat<double> D5(3,3,9.8); // 3x3 matrix, all entries = 9.8
D5.makeIdentity();       // change the entries of D5 and returns a reference,
                         // so can also do: Mat<double> D6 = D5.makeIdentity();
                         // in which case both D6 and D5 are identity matrices.

// compute the 1-norm of a square matrix
// (naive code ... not good for large matrices)
Mat<double> D6(2,2);    // 2x2 matrix, all entries = default double type
D6(0,0) = 1;            // set the entries
D6(0,1) = 1;            // such that
D6(1,0) = 1;            // the matrix is
D6(1,1) = 1.000000001;  // ill-conditioned
double norm1 = D6.norm1();

// compute the condition number of a small square ill-conditioned matrix
double condSmall = D6.smallCond1();   // small matrices are up to 3x3

// compute the condition number of a square ill-conditioned matrix
// using the function for large matrices
double condLarge = D6.largeSymmCond1();

// compute the determinant of a square ill-conditioned matrix.
// small matrix max size = 3x3
double det = D6.smallDet();

// compute the inverse of a small square ill-conditioned matrix
Mat<double> D7 = D6.smallInv();  // small matrices are up to 3x3

// compute the inverse using the function for large square matrix
Mat<double> D7a = D6.largeSymmInv();

// sum all the entries of a row of a matrix
Mat<double> D8(4,4,1); // 4x4 matrix, all entries = 1
D8(0,1) = 3.2;
D8(1,0) = 6.2;
double sum2 = D8.sumRow(2); // the second row. note that the the number of
                            // rows starts from 1, but the index from 0.

// sum all the entries of a column of a matrix
double sum3 = D8.sumCol(2); // the second column. note that the the number of
                            // columns starts from 1, but the index from 0.

// print size of a vector
cout << "Size of x: " << x.size() << endl;
cout << "Size of d1: " << d1.size() << endl;

// print size of a matrix
cout << "Number of rows of H: " << H.rows() << endl;
```

```
cout << "Number of columns of H: " << H.cols() << endl;

// print vectors, matrices and variables
cout << "x:" << endl;
cout << x << endl;
cout << "d1:" << endl;
cout << d1 << endl;
cout << "d2:" << endl;
cout << d2 << endl;
cout << "d3:" << endl;
cout << d3 << endl;
cout << "d4:" << endl;
cout << d4 << endl;
cout << "c1:" << endl;
cout << c1 << endl;
cout << "c2:" << endl;
cout << c2 << endl;
cout << "c3:" << endl;
cout << c3 << endl;
cout << "c4:" << endl;
cout << c4 << endl;
cout << "e1:" << endl;
cout << e1 << endl;
cout << "e2:" << endl;
cout << e2 << endl;
cout << "y:" << endl;
cout << y << endl;
cout << "z:" << endl;
cout << z << endl;
cout << "t:" << endl;
cout << t << endl;
cout << "Dot product: " << scalar << endl;
cout << "Euclidean norm: " << enorm << endl;
cout << "s1:" << endl;
cout << s1 << endl;
cout << "s3:" << endl;
cout << s3 << endl;
cout << "m1:" << endl;
cout << m1 << endl;
cout << "m5:" << endl;
cout << m5 << endl;
cout << "sum1:" << endl;
cout << sum1 << endl;
cout << "maxNumber:" << endl;
cout << maxNumber << endl;
cout << "minNumber:" << endl;
cout << minNumber << endl;
cout << "H:" << endl;
cout << H << endl;
cout << "H1:" << endl;
cout << H1 << endl;
cout << "H2:" << endl;
cout << H2 << endl;
cout << "F1:" << endl;
cout << F1 << endl;
cout << "F2:" << endl;
cout << F2 << endl;
cout << "F3:" << endl;
cout << F3 << endl;
cout << "F4:" << endl;
cout << F4 << endl;
cout << "P1:" << endl;
cout << P1 << endl;
cout << "P2:" << endl;
cout << P2 << endl;
cout << "P3:" << endl;
cout << P3 << endl;
cout << "V1:" << endl;
cout << V1 << endl;
cout << "b1:" << endl;
cout << b1 << endl;
cout << "b2:" << endl;
cout << b2 << endl;
cout << "S3:" << endl;
cout << S3 << endl;
cout << "S6:" << endl;
```

```
  cout << S6 << endl;
  cout << "S8:" << endl;
  cout << S8 << endl;
  cout << "S10:" << endl;
  cout << S10 << endl;
  cout << "T1:" << endl;
  cout << T1 << endl;
  cout << "a1:" << endl;
  cout << a1 << endl;
  cout << "a2:" << endl;
  cout << a2 << endl;
  cout << "D3:" << endl;
  cout << D3 << endl;
  cout << "D4:" << endl;
  cout << D4 << endl;
  cout << "D5:" << endl;
  cout << D5 << endl;
  cout << "norm1:" << endl;
  cout << norm1 << endl;
  cout << "condSmall:" << endl;
  cout << condSmall << endl;
  cout << "condLarge:" << endl;
  cout << condLarge << endl;
  cout << "det:" << endl;
  cout << det << endl;
  cout << "D7:" << endl;
  cout << D7 << endl;
  cout << "D7a:" << endl;
  cout << D7a << endl;
  cout << "sum2:" << endl;
  cout << sum2 << endl;
  cout << "sum3:" << endl;
  cout << sum3 << endl;
  //
  return 0;
}
```

5 Acknowledgements

I am grateful to Evgenii Rudnyi from CADFEM GmbH, Germany, and Mark Hoemmen from UC Berkeley for their valuable help on introducing me to general building of scientific libraries and especially TAUCS library.

6 References

[1] Sukumar N. Construction of polygonal interpolants: a maximum entropy approach. *International Journal for Numerical Methods in Engineering* 2004; **61**(12):2159-2181.

[2] Arroyo M, Ortiz M. Local maximum-entropy approximation schemes: a semaless bridge between finite elements and meshfree methods. *International Journal for Numerical Methods in Engineering* 2006; **65**(13):2167-2202.

[3] Sukumar N, Wright R.W. Overview and construction of meshfree shape functions: From moving least squares to entropy approximants. *International Journal for Numerical Methods in Engineering* 2007; **70**(2):181-205.

[4] Yau L.L, Sukumar N, Kunnath S.K. Meshfree co-rotational formulation for two-dimensional continua. *International Journal for Numerical Methods in Engineering* 2009; **79**(8):979-1003.

[5] Belytschko T, Lu Y.Y, Gu L. Element-free Galerkin methods. *International Journal for Numerical Methods in Engineering* 1994; **37**(2):229-256.

[6] Dolbow J, Belytschko T. An introduction to programming the meshless element free Galerkin method. *Archives of Computational Methods in Engineering* 1998; **5**(3):207-242.

[7] Li S, Liu W.K. Meshfree and particle methods and their applications. *Applied Mechanics Reviews* 2002; **55**(1):1-34

[8] Fries TP, Matthies HG. Classification and overview of meshfree methods. *Technical Report Informatikbericht*-Nr. 2003-03, Institute of Scientific Computing, Technical University Braunschweig, Braunschweig, Germany, 2004.