# MathVec++

User's Reference Manual

Version 1.0

September 23, 2009

**MathVec++**

## 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 MathVec++. MathVec++ is a template header that simplifies matrix-vector operations. Matrix-vector operations are designed for small matrices since it uses direct implementations. However, it can solve large sparse linear systems very efficiently due to its interface to TAUCS library. The complete source code of MathVec++ can be downloaded from http://www.computationalmechanics.org/~aortizb/codes.

# 2 Building MathVec++

This manual explains how to build MathVec++ under Windows® and Linux/Unix. MathVec++ consists of one template header file available in 'src' folder: mathvec.h. 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 MathVec++ under Windows

If you decide to build MathVec++ 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 MathVec++ in this distribution. Unless you use other compiler there is nothing more to do. See MathVec++ 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 MathVec++ in a compiler other than MVS might require rebuilding of TAUCS libraries.

Once the MVS project is in your machine, you can see and try the test file 'main.cpp' included in 'src' where some of the capabilities of MathVec++ are tested. You just need to build the solution in the MVS GUI: Build -> Build solution or by pressing F7. If you want to try other examples, 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 'MathVec++' is placed in 'bin' folder. You have to run that executable from command line (cmd). This will print the numerical computation to the screen. Since this template was thought to be used along with other's code, no output file is provided.

There is another alternative to install MathVec++ 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.

## 2.2 Building MathVec++ under Linux/Unix

Building MathVec++ 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 MathVec++' in the main folder of MathVec++ 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.

[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/.

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 MathVec++ template

The best way to see how MathVec++ works is through an example. The following code is actually the code provided in 'src' folder. It shows most of the capabilities of MathVec++. For other functions available in MathVec++ see 'mathvec.h' located in 'src' folder. Every function definition in 'mathvec.h' is preceded by a brief description of what the function does.

```
#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;
}
```

## 4 Solving a linear system of equations with MathVec++

This example is not provided in 'src' folder. It shows how to solve a linear system of equations. Although the solvers are especially designed for large sparse linear system, the following small linear system is intended to demonstrate the usage of the solver function.

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
// include the MATRIX-VECTOR template
#include "mathvec.h"

/*****************************************************************************
*                EXAMPLE OF SOLUTION OF LINEAR SYSTEMS IN MathVec++           *
*****************************************************************************/

int main()
{
  Vec<double> kn(10);  // vector containing the upper part of K by rows
  Vec<double> f(4);    // right-hand side vector
  int stat = 0;        // status of the solver execution
  // create CCS matrix structure
  kn(0) = 1.0;
  kn(1) = 0.5;
  kn(2) = 0.0;
  kn(3) = 0.0;
  kn(4) = 1.0;
  kn(5) = 0.5;
  kn(6) = 0.0;
  kn(7) = 1.0;
  kn(8) = 0.5;
  kn(9) = 1.0;
  // create right-hand side vector
  f(0) = 1.0;
  f(1) = 2.0;
  f(2) = 3.0;
  f(3) = 4.0;
  // solve the system
  Vec<double> soln = symmSparseLinearSolver(kn,f,&stat);
  if (stat != 0)
  {
    cout << "Error: Matrix might be singular." << endl;
    exit(1);
  }
  else
    cout << soln << endl;
  //
  return 0;
}
```

It is also possible to create a full matrix instead of a vector format to use it with the following function prototype:

```
const Vec<T> symmSparseLinearSolver(const Mat<T>& K, Vec<T>& f, int* stat)
```

For more details see 'mathvec.h' located in 'src' folder.

## 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.