Juzhen

Manual

Version 1.0

Hui Chen

2011

Table of Contents

1 Introduction

1.1 What's Juzhen

Juzhen is a C++ template library that provides

- Classes for matrices, vectors and complex numbers
- Algorithms for linear algebra and statistics.

The word Juzhen means 矩阵, or 'matrix' in Chinese. Pronunciation of Juzhen sounds like *jyu-jen.* You can check the sound from Google translate (copy-n-paste the url to a browser):

http://translate.google.com/#en|en|jyu-jen

1.2 License

Juzhen is published under Apache License 2.0:

Basically the license says Juzhen is a free (as in “free beer”) software for both commercial and non-commercial use. However, the software is provided “as is” thus the author is not responsibility for any damages it may cause.

1.3 Juzhen on the Web

Website of Juzhen is at

http://huichen.org/juzhen

The source code is hosted at github:

https://github.com/huichen/juzhen

Please submit any problems on the github issue page or drop me a message at usa.chen@gmail.com if you have comments, bugs, patches, etc.

2 Installation

2.1 Download

You can pull the source from github by entering following command in terminal:

```
git clone https://github.com/huichen/juzhen.git
```

Or just download the tarball of latest snapshot from the link

https://github.com/huichen/juzhen/tarball/master

2.2 Set Environs (Optional)

You need a package that provides solver functions such as solving eigen values and matrix inverse. I recommend optimized BLAS libraries such as GotoBLAS2 and Intel's Math Kernel Library. Set `JUZHEN_BLASLIB` environ if you want to use MKL

```
export JUZHEN_BLASLIB=mkl
```

or GotoBLAS2

```
export JUZHEN_BLASLIB=gotoblas2
```

If you don't set `JUZHEN_BLASLIB`, the library will use its own implementation of BLAS (to be specific, gemm) and the solvers (see section 7) won't work.

Set `OMP_NUM_THREADS` if multi-cores are available (if it's unset, only one processer is used):

```
export OMP_NUM_THREADS=4
```

2.3 Run Unit Tests

Enter the test folder and type in terminal

```
make
```

Make sure it passes every unit tests. If not, send me a bug report with what's displayed in your terminal.

2.4 Run a Simple Example

Content of a test program *main.cc* is shown below. The syntax of Juzhen is very intuitive.

```
#include <juzhen.h>  // Include the header

#include <iostream>

using namespace std;

using namespace juzhen;  // The namespace juzhen must be used.

int main() {

  Matrix<double> matrix(4, 4);

  // Define a 4x4 matrix of double precision.

  matrix << 1, 2, 3, 4,

            2, 3, 4, 1,

            3, 4, 1, 2,

            4, 1, 2, 3;

  /* Fill the matrix with comma-initializer. Cool! */

  cout << "matrix = \n" << matrix << endl;

  /* Print the matrix. Simple, isn't it? */

  return 0;

}
```

Compile and run *a.out* (need to include Juzhen's *src* folder in compile flag):

```
g++ main.cc -I your_juzhen_src_folder
```

Output:

```
matrix =

1 2 3 4

2 3 4 1

3 4 1 2

4 1 2 3
```

3 Complex Class

Complex numbers are defined in following ways:

```
Complex<T> c;  // c = 0 + 0i

Complex<T> c(3); // c = 3 + 0i

Complex<T> c(3, 4); // c = 3 + 4i
```

T can be type of *float* or *double*. For convenience, two typedefs are available:

```
typedef Complex<float> CS;

typedef Complex<double> CD;
```

Complex's real and imaginary part can be get/set by

```
CS c(3, 4);       // c = 3 + 4i

c.real = 10; float f1 = Real(c);  // c's real part

c.imag = 11; float f2 = Imag(c);  // c's imaginary

CS c1 = Conjugate(c);     // c's conjugate

Abs(c);           // absolute value

Abs2(c);          // square of absolute value
```

Just like scalars, complex numbers have arithmetic operations:

```
// These are all legal operations:

c + 3; c += 3;

c – 3; c -= 3;

c * 3; c *= 3;

c / 3; c /= 3;

3 + a;

3 – a;

3 * a;

3 / a;

a * CS(7, 8);  // a * (7 + 8i)

CD(9, 10) / a; // (9 + 10i)/a, CS and CD can be mixed
```

4 Matrix Class

4.1 Matrix Definition, Assignment and Access

Matrix can be declared in several ways:

```
Matrix<T> matrix();

// The default constructor, returns empty matrix.

// T can be float, double, CS (aka Complex<float>) and

// CD (aka Complex<double>)



Matrix<T> matrix(int col, int row);

// a col by row matrix

Matrix<T> matrix(T *data_ptr, int col, int row);

// declare a col by row matrix, and copy data from data_ptr
```

Four *typedef*s are predefined with LAPACK naming convention:

```
typedef Matrix<float> smatrix;

typedef Matrix<double> dmatrix;

typedef Matrix<Complex<float> > cmatrix;

typedef Matrix<Complex<double> > zmatrix;
```

We apply column-major order to all matrices, ie, if a matrix in mathematical form is

```
1 2 3

4 5 6
```

Then the order of elements stored in memory is, from low to high address,

```
1 4 2 5 3 6
```

Matrix can be initialized by comma separated values (in row major order):

```
Matrix<float> matrix(2, 3);  // 2 rows and 3 columns

matrix << 1, 2, 3,

          4, 5, 6;

std::cout << matrix << endl;
```

```
// This will output
// 1 2 3
// 4 5 6
```

Element in matrix can be accessed by parenthesis and square bracket operators:

```
float f = matrix(1, 2);  // equals 6
matrix(0, 1) = 10;
// matrix's (0,1) element is replaced with 10, gives
// matrix = 1 10 3
//          4 5 6
matrix(1);  // second element in the 1 dimension array, which is 4
matrix[1];  // same as above
```

Numbers of rows and columns of matrix are

```
m.num_row();  // number of rows
m.num_col();  // number of columns
m.size();     // total number of elements = m.num_row() * m.num_col()
```

Matrix can be resized/reshaped

```
// before resizing
// m = 1 2 3
//     4 5 6
m.Resize(3, 2);  // now 3 rows and 2 columns
// m = 1 5
//     4 3
//     2 6
m.Resize(3, 3);  // now 3 rows and 3 columns
// However last columns' values are undetermined
// m = 1 5 x
//     4 3 x
//     2 6 x
```

Matrix's elements can be cleared or set to a single value

```
Matrix<double> matrix(2, 2);

// the values of matrix's elements are undetermined


matrix.Clear();  // all elements are set to zero

matri.Set(3);  // all elements are set to 3
```

Finally, the pointer to the 1 dimensional array in memory can be accessed by

```
double *ptr = matrix.raw_ptr();
```

This is extremely useful when you want to call BLAS/LAPACK functions with the data stored in Matrix classes. However, do remember data storage is in column-major order.

4.2 Basic Matrix Operations

Transpose, conjugate and adjoint operations are provided:

```
zmatrix matrix(2, 2);

Transpose(matrix);

matrix.Transpose();

// both return transpose, matrix itself doesn't change. Same below.


Conjugate(matrix);

matrix.Conjugate(); // return conjugate


Adjoint(matrix);

matrix.Adjoint();  // return adjoint
```

You can also take real or imaginary part of a matrix

```
Real(matrix);  // return type is dmatrix

Imag(matrix);  // return type is dmatrix
```

4.3 Sub-Matrix Operations

Sub-matrix is obtained from Block function:

```
matrix.Block(r1, c1, r2, c2);  // matrix itself is unchanged
```

The sub-matrix has upper-left corner at (r1, c1) and lower-right corner at (contains) (r2, c2).

Replacing a sub-block of matrix with another matrix is easy:

```
matrix.Replace(r1, c1, another_matrix);  // matrix has been changed
```

The sub-matrix with upper-left corner at (r1, c1) has be replaced with another_matrix.

You can also get/swap columns and rows in a matrix;

```
matrix.GetRow(r);  // get row r, returns a row matrix
                   // matrix's value doesn't change, same for GetCol()
matrix.GetCol(c);  // get column c, returns a column matrix
matrix.SwapRow(r1, r2); // swap row r1 and row r2
                        // matrix's value changes, same for SwapCol
matrix.SwapCol(c1, c2); // swap column c1 and column c2
```

4.4 Matrix Arithmetic and Boolean Operations

Matrices have arithmetic operations.

```
dmatrix m1(3,3), m2(3,3);
// unary operations
+m1; // returns itself
-m1; // returns opposite elements


// with scalars
m1 + 2;  // every elements add one
m1 – 2;
m1 * 2;
m1 / 2;
```

```
2 + m1;

2 – m1;

2 * m1;

m1 += 2;

m1 -= 2;

m1 *= 2;

m1 /= 2;


// with matrix

m1 + m2;

m1 – m2;

m1 * m2; // matrix multiplication

m1 *= m2; // m1 = m1 * m2;


// Boolean operations

m1 == m2;  // dimensions and every elements must be equal

m1 != m2;  // opposite of ==
```

5 Vector Class

Vector class is a derived class of Matrix so they have a lot of similarities in member functions.

5.1 Vector Definition, Assignment and Access

Vector can be declared in several forms.

```
Vector<T> vector();

// The default constructor, returns an empty vector.

// T can be float, double, CS (aka Complex<float>) and

// CD (aka Complex<double>)


Vector<T> vector(int s);

// a vector of size s

Vector<T> vector(T *data_ptr, int s);

// declare a vector of size s, and copy data from data_ptr
```

Like the Matrix class, fours *typedef*s are predefined with LAPACK naming convention:

```
typedef Vector<float> svector;

typedef Vector<double> dvector;

typedef Vector<Complex<float> > cvector;

typedef Vector<Complex<double> > zvector;
```

Vector can be initialized by comma separated values

```
dvector vector(3);  // length of 3

vector << 1, 2, 3;

std::cout << vector << endl;

// This will output (notice the difference from matrix’s)

// {1, 2, 3}
```

Elements in vector can be accessed by

```
double f = vector(1);  // equals 2

vector(1) = 10;
```

```
// vector’s second element is replaced with 10

vector[1] = 10;  // same as above
```

Size of a vector is

```
vector.size();     // total number of elements in vector
```

Vector can also be resized

```
// before resizing

// vector = {1, 2, 3}

vector.Resize(5);  // now vector has 5 elements

// Howevere last two elements’ values are undetermined

// vector = {1, 2, 3, x, x}
```

Vector’s elements can be cleared or set

```
dvector vector(2);

// Initially, the values of vector’s elements are undetermined

vector.Clear();  // all elements are zero now

vector.Set(3);  // all elements are 3 now
```

Finally, the pointer to the 1 dimensional array in memory can be accessed by

```
double *ptr = vector.raw_ptr();
```

5.2 Basic Vector Operations

Conjugate, transpose and adjoint operations are provided though transpose has no effect and adjoint is equivalent to conjugate:

```
zvector vector(2);

Conjugate(vector);

vector.Conjugate(); // return conjugate

Adjoint(vector);

vector.Adjoint(); // return conjugate
```

You can also get real or imaginary part of a vector by

```
Real(vector);  // return type is dvector
```

```
Imag(vector);  // return type is dvector
```

5.3 Sub-Vector Operations

Sub-vector is obtained from Block member function:

```
vector.Block(i1, i2);  // vector is unchanged
```

The sub-vector starts at index i1 and ends contains at (contains) index i2.

Replacing a sub-block of vector with another vector is easy

```
vector.Replace(i1, another_vector);  // matrix changed
```

The sub-vector starting at index i1 will be replaced with another_vector.

You can also swap elements in a vector;

```
vector.Swap(i1, i2); // swap i1 element and i2 element
```

5.4 Vector Arithmetic and Boolean Operations

Vectors have arithmetic operations.

```
dvector v1(3), v2(3);

// unary operations

+v1; // returns itself

-v1; // returns opposite elements



// with scalars

v1 + 2;  // every elements add one

v1 – 2;

v1 * 2;

v1 / 2;

2 + v1;

2 – v1;

2 * v1;

v1 += 2;

v1 -= 2;
```

```
v1 *= 2;

v1 /= 2;



// with vector

v1 + v2;

v1 – v2;

v1 * v2; // vector dot product, returns a scalar



// with matrix

dmatrix m(2,2);

v1 * m; // row vector times matrix, returns a vector of same type

m * v1; // matrix times column vector, returns a vector of same type



// Boolean operations

v1 == v2;  // must have same size and every element must be equal

v1 != v2;  // opposite of ==
```

6 Important Note on Matrix/Vector Copy Constructor

A copy constructor of Matrix class is only allowed with simple variables. Copy construct a matrix from return of a function or an expression will give undetermined results:

```
Matrix<double> m1(m2 + m3);            // This is NOT ALLOWED

Matrix<double> m1 = m2 + m3;           // This is NOT ALLOWED

Matrix<double> m1 = Conjugate(m2);     // This is NOT ALLOWED

Matrix<double> m1(m2);                 // Allowed

Matrix<double> m1 = m2;                // Allowed
```

Same for vectors:

```
Vector<double> v1(v2 + v3);            // This is NOT ALLOWED

Vector<double> v1 = v2 + v3;           // This is NOT ALLOWED

Vector<double> v1 = Conjugate(v2);     // This is NOT ALLOWED

Vector<double> v1(v2);                 // Allowed

Vector<double> v1 = v2;                // Allowed
```

7 Solvers

Three classes of solvers are available:

```
// Linear equation solver
// A * X = B
// Return X if successful
// Return an empty matrix if it fails
Matrix<T> LinearSolver(const Matrix<T> &A, const Matrix<T> &B);


// Inverse matrix
// X = inv(A)
// Return X if successful
// Return an empty matrix if X is not invertable
Matrix<T> Inverse(const Matrix<T> &A);


// Eigen solvers
// There are three versions: right, left and both
// Right: A * Xr = lambda * Xr
// Left:  Xl * A = lambda * Xl
// Return lambda (eigen values) as a complex vector, Xr and Xl as square
matrices with all eigen vectors
void RightEigenSolver(const Matrix<T> &A,
              Matrix<Complex<T> > &lambda, Matrix<T> &Xr);
void LeftEigenSolver(const Matrix<T> &A,
              Matrix<Complex<T> > &lambda, Matrix<T> &Xl) {


void EigenSolver(const Matrix<T> &A,
              Matrix<Complex<T> > &lambda, Matrix<T> &Xl, Matrix<T> &Xr);
```

8 Statistical and Fitting Functions

There are several statistical functions available:

Target Classes	Function usage	Meaning
Vector<T> and Matrix<T>	Sum(target)	Sum of all elements
Vector<T> and Matrix<T>	Max(target)	Max element
Vector<T> and Matrix<T>	Min(target)	Min element
Vector<T> and Matrix<T>	Mean(target)	Mean of elements
Vector<T> and Matrix<T>	Prod(target)	Product of all elements
Vector<T>	StdDev(target)	Standard deviation
Vector<T>	Var(target)	Variance
Vector<T>	Cov(target1, target2)	Covariance of two vectors
Vector<T> and Matrix<T>	Norm(target)	Euclidean norm of a vector
Vector<T> and Matrix<T>	NormSquare(target)	Square of Euclidean norm
Vector<T> and Matrix<T>	NormOne(target)	Taxicab (Manhattan) norm
Vector<T> and Matrix<T>	NormInfinity(target)	Infinity (maximum) norm
Vector<T>	Sort(target)	Sort a vector
Vector<T>	CorrCoeff(target1, target2)	Correlation Coefficient of two vectors

Linear least-squares method is also provided for vectors:

```
Vector<T> LinearLeastSquares(const Vector<T> &X,
                             const Vector<T> &Y);
```

The return is a 2-vector with the parameters (k1, k2) which fit the relation with least squares.

$$Y = k1*X + k2$$

9 Python Interface

A Python interface to Juzhen was generated from SWIG. Thanks to its efficient C++ core, the Python module runs much faster than some numerical libraries such as Numpy.

9.1 Compile Python Module

You need to compile the Python module before using it. Make sure you have installed development package for Python (version>=2.6), which has the *Python.h* header.

```
cd python

make
```

Two files are generated from SWIG: *_juzhen.so* (shared library) and *juzhen.py* (package declaration). Copy them to the folder you want to import juzhen from.

9.2 Classes and Functions

Five classes are available:

Complex class (complex of double precision)

Matrix and **CMatrix** classes (float and complex double matrix)

Vector and **CVector** classes (float and complex double vector)

There are also solver functions to solve **inverse matrix**, **linear equation**, **eigen values/vectors**, etc.

9.3 Complex Class Example

```
from juzhen import Complex

# a = 3 + 4i

a = Complex(3, 4)



# a = -1 + 0i

b = Complex(-1)



# This is equivalent to c = b

c = Complex(b)
```

```
print "a = ", a
print "b = ", b
print "c = ", c

# Conjugate of a
print "a.conj() = ", a.conj()

# Absolute value of a
print "a.abs() = ", a.abs()

# Real part of a
print "a.real = ", a.real

# Imaginary part of a
print "a.imag = ", a.imag

print "a + b = ", a + b
print "a + 3 = ", a + 3

print "a - b = ", a - b
print "a - 3 = ", a - 3

print "a * b = ", a * b
print "a * 3 = ", a * 3

print "a / b = ", a / b
print "a / 3 = ", a / 3
```

9.4 Matrix and CMatrix Class Example

Replace Matrix with CMatrix in following code to use complex matrix.

```
from juzhen import Matrix, CMatrix, Identity, CIdentity


# Define a matrix of 3 rows (1st parameter) and 3 columns (2nd parameter)
# Row and column indices start from zero
#
#     x    x    x    (row 0)
#     x    x    x    (row 1)
#     x    x    x    (row 2)
#
# Col 0    1    2
#
a = Matrix(3, 3)


# Number of rows and columns
print "a.num_row() = ", a.num_row()
print "a.num_col() = ", a.num_col()


# Set zero to a's elements
a.clear()


# Set 3 to all elements
a.set(3)
print "a.set(3); a = "
print a
print
```

```
# 7 x 7 identity matrix

print "Identity(7) = "

print Identity(7)

print


for i in range(0, 3):

  for j in range(0, 3):

    a.set(i, j, i*j+1) # Set a's element at i-th row & j-th row to be
i*j+1

print "a = "

print a

print


# Get a's (1,2) element

print "a.get(1, 2) = "

print a.get(1, 2)

print


# Get a copy of a and save to b

# b = a doesn't copy the values

b = a.copy();

b.set(1, 1, 3)

print "b = "

print b

print


print "a + b = "

print a + b

print
```

```
print "a - b = "
print a - b
print

print "a * b = "
print a * b
print

print "a * 3 = "
print a * 3
print

print "a / 3 = "
print a / 3
print

# Get column 2
print "a.get_col(2) = "
print a.get_col(2)
print

# Get row 1
print "a.get_row(1) = "
print a.get_row(1)
print

# Get a block between row 1 (include) to row 3 (not include) and column 0
(include) to column 2 (not include).
```

```
print "a.block(1, 3, 0, 2) = "

print a.block(1, 3, 0, 2)

print


c = Matrix(2, 2)

c.clear()


# Replace the sub matrix with upper-left corner at (1, 0) with matrix c

print "a.replace(1, 0, c) = "

print a.replace(1, 0, c)

print


# Swap column 0 and column 1

print "a.swap_col(0, 1) = "

print a.swap_col(0, 1)

print


# Swap row 0 and row 1

print "a.swap_row(0, 1) = "

print a.swap_row(0, 1)

print


# Resize the dimension of a

print "a.resize(2, 2); a = "

a.resize(2, 2)

print a

print


# Get the real part of a matrix
```

```
print "a.real() = "

print a.real()

print


# Get the imaginary part of a matrix

print "a.imag() = "

print a.imag()

print


# Get the transpose of a matrix

print "a.trans() = "

print a.trans()

print


# Get the conjugate of a matrix

print "a.conj() = "

print a.conj()

print


# Get the adjoint of a matrix

print "a.adj() = "

print a.adj()

print
```

9.5 Vector and CVector Class Example

Replace Vector with CVector in following code to use complex vector.

```
from juzhen import Vector, CVector, Identity, CIdentity



# Define a vector of length 6
```

```
a = Vector(6)


# Set zero to all elements of a
a.clear()


# Set 3 to all elements
a.set(3)
print "a = "
print a
print


# Set elements
for i in range(0, 6):
    a.set(i, i)
print "a = "
print a
print


# Get size of a vector
print "a.size() = "
print a.size()
print


# Get element 1
print "a.get(1) = "
print a.get(1)
print


# Copy data of a to vector b
```

```
b = a.copy();


# Set element 1 to be 3

b.set(1, 3)

print "b = "

print b

print


print "a + b = "

print a + b

print


print "a - b = "

print a - b

print


print "a * b = "

print a * b

print


print "a * 3 = "

print a * 3

print


print "a / 3 = "

print a / 3

print


print "a * Identity(6) = "
```

```
print a * Identity(6)

print


print "Identity(6) * a= "

print Identity(6) * a

print


# Sum of all elements

print "a.sum() = "

print a.sum()

print


# Find the max elements

print "a.max() = "

print a.max()

print


# Calculate the norm of vector a

# norm(a) = sqrt(a_1*a_1 + a_2*a_2 + ...)

print "a.norm() = "

print a.norm()

print


# Sort a vector

print "b.sort() = "

print b.sort()

print


# Get the real part of a vector
```

```
print "a.real() = "

print a.real()

print


# Get the imaginary part of a vector

print "a.imag() = "

print a.imag()

print
```

9.6 Solver Function Example

```
from juzhen import Complex, Matrix, CMatrix, Identity, CIdentity, Vector,
CVector

from juzhen import inverse

from juzhen import linear_solver

from juzhen import eigen, left_eigen, right_eigen


a = Identity(3)

print "inverse(a) = "

print inverse(a)

print


b = Matrix(3, 3)

b.set(10)


# Solve A * X = B

print "linear_solver(a, b) = "

print linear_solver(a, b)

print
```

```
a.set(1, 2, 3)

a.set(0, 1, 7)

a.set(2, 0, 7)


# Solve eigen problem:

#   e is a column vector contains all eigen values

#   vl is a square matrix that contains all left eigen vectors

#   vr is a square matrix that contains all right eigen vectors

# Similarly,

# e, vl = left_eigen(a) solves left eigen vectors only

# e, vr = right_eigen(a) solves right eigen vectors only

e, vl, vr = eigen(a);

print "e, vl, vr = right_eigen(a); e = "

print e

print

print "vl = "

print vl

print

print "vr = "

print vr

print
```