The code to generate data for CEC 2015 Special Session: Competition on Bound Constrained Single Objective Numerical Optimization

Qin Chen

Facility Design and Instrument Institute, China Aerodynamic Research & Development Centre, China

chenqin1980@gmail.com, chenqin@fdiicardc.org

Last updated: 2014-10-07

Introduction

There are three parts of data required in the novel composition test functions for numerical global optimization propose by J.J. Liang[1], which will be adopted in the Competition on Bound Constrained Single Objective Numerical Optimization in CEC 2015. The first data are orthogonal rotation matrix, the second are bounded vector to shift the optimum position, and the last is a randomly permutation of dimensional indexes used to generating hybrid functions.

To fulfil the potential use of the test functions in Java, MATLAB and C, we have to make sure all the versions for different languages show the exact same results when generating the data file for the test functions. Thus we define the basic rand number generator in C, and reuse the code in Java as Java Native Interface and in MATLAB with a MEX-function interface. The random number generator used here is the JKISS RNG proposed by David Jones[2]. The returned random double numbers with values uniformly distributed between 0 and 1 are actually 32-bits, which don't cover all double precision real numbers, but are regarded as sufficient for our purpose.

For orthogonal rotation matrix, QR decomposition algorithm is used. Also the same code are shared by C version and Java version. While for MATLAB version, the qr function show the same results as the QR algorithm which we borrowed from Meschach Library[3].

The software packages are validated with MATLAB 7.12.0.635 (R2011a), Microsoft (R) C/C++ Optimizing Compiler Version 16.00.30319.01, gcc for Debian 4.7.2-5, Oracle JDK 1.8 and openJDK 1.7.

The lasted version of the code package can be downloaded at GitHub:

```
https://github.com/withstand/cec15datagenerator
```

Software packages

Folders' structures of the package are like Table 1, in which the folders named meshach and

datagenerator are C implementation of the main functionality, commonly used by C version and Java version. Other folders and files are explained in Table 1.

Table 1 Software package components

Folders/Files	
meschach	Source files for Meschach library
matlab	Matlab version
java	Java version
dg_test_vs	VS solution files
datagenerator	C source files
makefile	Makefile for linux test for c
datagenerator_test.c	C test source file
DataGenerator.pdf	This document

MATLAB version

MATLAB version is the simple one. It consists of MATLAB wrapper functions and a jkiss.c, which must be compiles into shared library with the command line in MATLAB command prompt:

```
>>mex jkiss.c
```

Table 2 MATLAB Interface

```
cec15rand (m, n) Return a matrix with the size [m, n] filled with rand number
between 0 and 1.

cec15reseed([x, y, z, c]) Reseed JKISS RNG with four unsigned integer number
between [0, 2^32-1].

cec15rotate (n) Return the orthogonal rotation matrix with the size [n, n].

cec15shuffle (n) Return permutation of 1: n, the algorithm is a implementation
of Fisher-Yates shuffle [4].
```

C version

The C version consists of two main parts, the code borrowed from Meschach library to do QR decomposition and the code to generate required random numbers.

The functions to generate the required random numbers are mainly listed in datagenerator.h.

Table 3 C interface

```
#ifndef DATAGENERATOR_H
#define DATAGENERATOR_H

//rand number generator
void cec15_reseed(unsigned int x1, unsigned int y1, unsigned int z1, unsigned int c1);
double cec15_rand();
void cec15_rand_matrix(int m, int n, double* mat);

void cec15_rotate(int m, double* mat);
void cec15_shift(int m, double* vec);
void cec15_shuffle(int m, int * shuf);

#endif DATAGENERATOR_H
```

The functions listed above basically do the same jobs as the equivalent MATLAB version. For output arguments in the function, the user should manage the memory, i.e. allocate sufficient memory before call these functions and free the memory to complete their life cycle.

Other C header files and source files are borrowed from Meschach library, which should be included when compiling the package. There is also a datagenerator_test.c to show the basic usages of the package. A makefile is presented for GNU make/gcc and the folder dg_test_vs contains the solution files and projects file for MS Visual C++.

This package could be used in C++, while using, have to tell the C++ compiler this functions are defined in C as follows:

Table 4 Used in C++

```
extern "C" {
  #include "datagenerator.h" //a C header, so wrap it in extern "C"
}
```

Java version

For Java, there is no unsigned int, so we decide to use the same code for JKISS RNG and QR decomposition through Java Native Interface (JNI). You must have both JDK (openJDK or Oracle JDK) and C compiler to make the code work.

The package is basically a Java wrapper for the C version. It consists of a DataGenertorJNI.java, which can be compiled to class file with javac, and a datageneratorjni.c, which implements the native functions defined in DataGeneratorJNI.java.

Table 5 Java Interface

```
// Competition on Bound Constrained Single Objective Numerical Optimization
package org.ieee.cec15.cbcsono;

public class DataGeneratorJNI {

   // Load native library at runtime
   //    datageneratorjni.dll (Windows) or
   //    libdatageneratorjni.so (Unix/Linux)
   static {
      System.loadLibrary("datageneratorjni");
   }
   public native double rand();
   //Use long as unsigned int 0 - 2^32-1
   public native void reseed(long x, long y, long z, long c);
   public native double[][] rand(int m, int n);
   public native int [] shuffle(int n);
   public native double[][] rotate(int n);
}
```

The compiling process of the Java Native Interface code is more complicate. A good tutorial could be found in [5].

For our purpose, the following procedure is sufficient:

1. Compile DataGeneratorJNI.java with the following command line:

```
javac DataGeneratorJNI.java –d .
```

The command javac will create related folder structure in the folder specified by –d. Here the folder is ".", i.e., the current working folder.

We can check the result class file DataGeneratorJNI.class in the ./org/ieee/cec15/cbcsono/ to make sure the compiling is completed.

2. Generator C header file for JNI implementation using javah:

```
javah –jni org.ieee.cec15.cbcsono.DataGeneratorJNI
```

We will have a C header file org_ieee_cec15_cbcsono_DataGeneratorJNI.h in the current working folder.

3. Compile the implementation C source file datageneratorjni.c with any C compiler with required Meschach library. The command line will be like the following under Linux with gcc:

```
gcc –lc –shared –lm –fPIC –I$(JAVA_HOME)/include –I$(JAVA_HOME)/include/linux –o libdatadeneratorjni.so datageneratorjni.c datagenerator.c (all other sourcefile…)
```

Where $(JAVA_HOME) should be the home folder of the Java Development Kits. We can use the following command line to locate it:

```
which java                #result -> /usr/bin/javac
ls –l /usr/bin/javac      #result -> /usr/lib/jvm/java-1.7.0-openjdk-amd64/javac
```

We may have to use ls –l for several times to finally locate the folder.

In Windows Platform, we could browser to locate the folder of Java Home.

There are two batch files for gcc under Linux and for cl under MS Windows to show the detailed procedure. Just make sure substitute the correct folders in them.

While using the Java Package, just import org.ieee.cec15.cbcsono.DataGenerator and use the class as regular Java class. Only you have to make sure the folder that contains the shared library, libdatagenerator.so or datagenerator.dll for Windows or Linux respectively, in Java's library path, which can be defined when running Java program as follows:

```
java –Djava.library.path=$(FOLDER_THAT_CONTAINS_DATADENERATOR_LIBRARY)
```

Contacts

If you have any problems, recommendations, advices and bugs to report, please don't hesitate to contact the following email addresses:


```
chenqin1980@gmail.com
chenqin@fdiicardc.org
```


We will try our best to make the code workable for possible platforms and make it bug-free.

References

[1] Liang, J. J., P. N. Suganthan, and K. Deb. "Novel composition test functions for numerical global optimization." *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*. IEEE, 2005.

[2] Jones, David, and UCL Bioinformatics Unit. "Good practice in (pseudo) random number generation for bioinformatics applications." *URL http://www. cs. ucl. ac. uk/staff/d. jones/GoodPracticeRNG. pdf* (2010).

[3] Stewart, D. E., and Zbigniew Leyk. "Meschach library." *Australian National University, Canberra, Australia* (1994).

[4] Black, Paul E. "Fisher-Yates shuffle." *Dictionary of Algorithms and Data Structures* 19 (2005).

[5] Chua Hock Chuan. “Java Programming Tutorial-Java Native Interface (JNI)”. *URL https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html*