Compilation of GSL 1.15 under Microsoft Visual C++ 2010 Express

Tristan Dagobert

9 janvier 2013

First we present compilation of the GNU Scientific Library version 1-15 **from source code**, under the IDE (integrated development environment) Microsoft Visual C++ 2010 Express under the distribution Windows 7 Professional both for 32 and 64 architectures.

We have not tested GSL compilation with the professionnal IDE Microsoft Visual C++ Studio 2010 (i.e. MSVC Studio) because it requires license fee. The inner differences between these two IDE concern number of functionnalities. It does not call into question the fact that if compilation works fine under MSVC Express, then it will work fine under MSVC Studio.

Then we show the numerical tests results computed from the GSL test packages.

0.1 Required sofwares

- **Microsoft Visual C++ 2010 Express**. (i.e. MSVC Express) This is the IDE of Windows which allows writing C and C++ programs. This is an evaluation version reserved to students or beginners in programming. By default, programs are compiled in 32 bits. MSVC Express is available from `http://msdn.microsoft.com/fr-fr/gg699327`.
- **Microsoft Windows SDK 7.1** It is the development kit which allows to compile programs both in 32 and 64 bits from MSVC Express. Available from `http://www.microsoft.com/en-us/download/details.aspx?id=8279`.
- **The GNU Scientific Library**. The last version is 1.15. Source code of it comes into package `gsl-1.15.tar.gz` available from `ftp://ftp.gnu.org/gnu/gsl/`.
- **gsl-1.15-vc10.zip** It is a set of scripts in C, Python and MSVC macros written by Brian Gladman letting to
  - add to source code some options needed by Windows compilers, into several headers files
  - replace the file Makefile (specific to Unix/Linux environment) of the GSL by a file consistent with MSVC.

  These scripts are not intrusive in sense they do not modify data types nor algorithms. This package is available from `http://gladman.plushost.co.uk/oldsite/computing/gnu_scientific_library.php`.
- **Python library**. It is used to run the conversion scripts for some of the GSL header files. It is available from `www.python.org` for installation using the Windows x86-64 MSI installer.

0.2 Installation des logiciels

Please respect the chronology of installation. Following instructions come from help files of the package `gsl-1.15-vc10.zip`. They allow to build dynamic version of the GSL under the object files `cblas.dll` and `gsl.dll` (corresponding respectively to the files `libgslcblas.so` and `libgsl.so` under UNIX/Linux environment) and the static versions, i.e. `cblas.lib` and `gsl.lib`. For more details, consult these files.

0.2.1 Installation of Microsoft Visual C++ 2010 Express

Download then install the required package.

0.2.2 Installation of Microsoft Windows SDK 7.1

Download then install the required package.

0.2.3 Installation of Python Library

Download then install the required package.

0.2.4 Installation of dynamic version GSL 1.15

GSL package usage

Download package `gsl-1.15.tar.gz`. Decompress it in an appropriate directory. We then obtain a tree like this :

```
gsl-1.15\blas\
gsl-1.15\block\
...
```

GSL converter package usage

Download package `gsl-1.15-vc10.zip`. Decompress it into the previous directory, i.e. `gsl-1.15\`. During decompression, some files will be indicated as being replaced : accept replacement of all these files. After compression, we then obtain tree like :

```
...
gsl-1.15\build.vc10\
...
```

Python usage

Run Python script `add.express.py` located in directory `gsl-1.15\build.vc10\`. **Be careful :** this script modify compilation files in order to take into account the SDK 7.1 compilers and MVSC Express specificities. It must not be ran if you use MVSC Studio !

MSVC usage

Run MSVC Express IDE.

A Before compiling anything, two modifications need to be done in source code. Indeed, it had been shown in `http://4fire.wordpress.com/2012/03/18/gsl-1-15-building-with-visual-studio-2010/` that it came from Windows preprocessor subtilities... Note that the second modification concern a problem occuring during the static compilation and not during the dynamic however we describes it here. Notice too, that these modifications do not affect algorithms.

Modify file `gsl-1.15\ode-initval2\rk4imp.c` replacing each instruction `sqrt(3)` by `M_SQRT3`. In file `gsl-1.15\bspline\bspline.c`, modify function `gsl_bspline_greville_abscissa()` by moving the three declarations

```
const size_t stride = w->knots->stride;
size_t km1 = w->km1;
double * data = w->knots->data + (i+1)*stride;
```

at the first line of this function, i.e. before the preprocessor macro.

B Open project `gsl.dll.sln` located in `gsl-1.15\build.vc10\`. This project contains 4 sub-projects :

```
cblasdll
gsldefs
gsldll
gslhdrs
```

Note that sub-project `gsldefs` will not be used here. It is an alternative to the `gslhdrs` usage.

C By default compilation is done with the MSVC compiler v100 in Debug mode and for 32 bits architecture. If you want to built the 4 differents versions i.e. {Debug, Release} × {Win32, x64} you will have to process from C0 to C3 each time.

C0 Select the 4 sub-projects once, then :
- Replace compiler : Projet → Propriétés de configuration → Général → Ensemble d'outils de plateforme := Windows SDK 7.1
- Replace mode : mode := Release or Debug
- Replace architecture : plate-forme := x64 or Win32

C1 Select the project `gslhdrs`. Compile **and** run this project : Déboguer → Générer la solution. After compiling, executable `gslhdrs.exe` modifies some headers file of the GSL.

C2 Select the project `cblasdll`. Compile this project : Select line → Right click → Générer. Object file `cblas.dll` will be produced and located into subdirectory
`gsl-1.15\build.vc10\dll\{Win32 | x64}\{Release | Debug}\`.

C3 Select the project `gsldll`. Compile this project : Select line → Right click → Générer. Object file `gsl.dll` will be produced and located into subdirectory
`gsl-1.15\build.vc10\dll\{Win32 | x64}\{Release | Debug}\`.

0.3 Test of dynamic version of GSL 1.15

Note that all the functions of the GSL have been compiled correctly. Some of them have showed warning but no compilation error occured.

Open project `gsl-1.15\build.vc10\test.gsl.dll.sln` containing 44 sub-projects. As previously, process as C0 to select the good compiler, architecture and mode and then compile them all as C2 (not C1).

Run tests by executing the python script `gsl-1.15\build.vc10\dlltest\run-tests.py`. Each of the 44 test projects runs and puts its output in the appropriate subdirectory :

```
gsl-1.15\build.vc10\dll\Win32\Release
gsl-1.15\build.vc10\dll\Win32\Debug
gsl-1.15\build.vc10\dll\x64\Release
gsl-1.15\build.vc10\dll\x64\Debug
```

as a `name.out` where `name` indicates the test in question. These output files will contain failure messages if there are any problems.

The table 1 show that numerical problems occured, concerning matrix and vector test. In both case, it seems to come from a subscript out of range when C `complex` types are tested. The file `testmatrix.out` contains :

```
FAIL : gsl_matrix_set traps 1st index above upper bound [751]
FAIL : gsl_matrix_set traps 2nd index above upper bound [752]
FAIL : gsl_matrix_set traps 1st index at upper bound [753]
FAIL : gsl_matrix_set traps 2nd index at upper bound [754]
```

The file `testvector.out` contains :

```
 FAIL: gsl_vector_set traps index below lower bound stride=1, N=1 [28344]
FAIL: gsl_vector_set traps index above upper bound stride=1, N=1 [28345]
FAIL: gsl_vector_set traps index at upper bound stride=1, N=1 [28346]
FAIL: gsl_vector_get traps index below lower bound stride=1, N=1 [28347]
FAIL: gsl_vector_get traps index above upper bound stride=1, N=1 [28349]
FAIL: gsl_vector_get traps index at upper bound stride=1, N=1 [28351]
FAIL: gsl_vector_float_set traps index below lower bound stride=1, N=1 [28353]
FAIL: gsl_vector_float_set traps index above upper bound stride=1, N=1 [28354]
FAIL: gsl_vector_float_set traps index at upper bound stride=1, N=1 [28355]
FAIL: gsl_vector_float_get traps index below lower bound stride=1, N=1 [28356]
FAIL: gsl_vector_float_get traps index above upper bound stride=1, N=1 [28358]
FAIL: gsl_vector_float_get traps index at upper bound stride=1, N=1 [28360]
```

However Debug and Release compilations are only differents from their compilation and preprocessor options. According to some discussions on the Net, some Windows compilers may redefine the C `complex` type into their own `math.h` file, without respecting the ISO recommendations. But this point is unclear.

	Debug	Release
Win32	testmatrix and testvector failed	all 44 tests are OK
x64	testmatrix and testvector failed	all 44 tests are OK

Tab. 1 – Result of numerical tests for the dynamic version of GSL, depending on architecture and mode.

0.3.1 Installation and test of static version GSL 1.15

Installation process is the same as for the dynamic version except that the projects to be open are `gsl.lib.sln` and `test.gsl.lib.sln`. The object files produced are
`gsl-1.15\build.vc10\lib\{Win32 | x64}\{Release | Debug}\cblas.lib` and
`gsl-1.15\build.vc10\lib\{Win32 | x64}\{Release | Debug}\gsl.lib`.

Run tests by executing the python script `gsl-1.15\build.vc10\libtest\run-tests.py`.

The table 2 shows that the same numerical problems as described previously occur.

	Debug	Release
Win32	testmatrix and testvector failed	all 44 tests are OK
x64	testmatrix and testvector failed	all 44 tests are OK

Tab. 2 – Result of numerical tests for the static version of GSL, depending on architecture and mode.

0.4 Conclusion

Compilation of GSL 1.15 from its source code on Microsoft Visual C++ 2010 Express has been described precisely. With minor corrections, compilation works fine, both in dynamic and static modes and both in 32 and 64 bits. The standard numerical tests have been made. They show that all the tests are correct in release mode on the two architectures.

Two errors occurs in Debug mode, but which seem to come more from the compilation option subtilities than from the source code de GSL itself. So we can say these errors are irrelevant.