More Mathematical Functions, especially for Statistics.

Progress report 30 Nov 2005

Paul A Bristow

Aficionados of mathematical functions may remember that I proposed some additional functions

http://www2.open-std.org/JTC1/SC22/WG21/docs/papers/2004/n1668.pdf

to the choices of Walter Brown which have been added to the C++ TR1 (latest draft)

http://www2.open-std.org/JTC1/SC22/WG21/docs/papers/2005/n1905.pdf

The rationale for my proposed functions is that previously standardized functions do not include several functions which have very much wider utility in statistical problems, even the most elementary, like Student's t, Fisher and Chi-sqr tests.

To recap, a long list of basic and essential functions were added with the C99 for which rationale is given in

http://www2.open-std.org/JTC1/SC22/WG14/www/C99RationaleV5.10.pdf

Briefly the total list of functions consists of

1 Original math.h functions, log, exp, pow, sin, cos...
1 So-called IEEE recommended functions, signbit, isnan, isfinite ...
2 Some simplish math functions like lgamma, tgamma, cbrt, acosh.
3 TR1 Bessel functions, Reimann, hypergeometric...
4 TR2 proposed additions, incomplete beta, Fisher, Student, Chisqr probability distributions...

I have now produced a first demo of a few of these functions to carry out some very simple, but almost universally applicable statistical tests - Student's t and Fisher F-ratio. The example is from analytical chemistry, but could equally well be of school exams, economic comparisons, measurement of lengths, brightness of stars - the list is endless. The demo shows computation of the probability that two methods of extracting tin from foodstuffs give a different result and the probability that one method is more precise than the other. It does NOT require the use of published tables.

All the hard and difficult work has been done by Stephen Moshier, author of a respected book on mathematical functions, and a long supported (since 1984) and extensive library of mathematical functions in C called Cephes. Moshier has kindly offered to change the library license terms to the Boost terms to support this project.

I have merely packaged a few examples of the C functions as C++ functions and placed them in namespaces std::tr2 to show what they would look like in final form.
The C versions are required to remain in global namespace to follow the example of the C99 functions which can be used by both C and C++ programs. I am advised that this is a requirement to get the proposal accepted by C WG14 and C++ WG21 Standard groups.

I have compiled all the Cephes modules (with a few name changes using macros in the mconf.h configuration header file) and built a library from them.

In unitTestfunc1.cpp, I show declarations and definitions of a few sample functions

```
       float cbrt(float x)
       { // Loss of speed over C float version cbrtf?
              // But not less accurate.
              // return float(::cbrt(x));
              return ::cbrtf(x); // C99 implementation from Cephes,
// 32-bit IEEE 754 single precision.
       } //float cbrt(float x)
```

Some of the 'issues' are:

1 Floating-point format. I have assumed IEEE formats, but try to check and warn if not. All built-in types can vary in precision, especially double and long double: I have shown an example of selecting long double having 53, 80 or 128 significand bits (all IEEE formats).

2 I have only used MSVC 8.0 (but I expect older versions to work) where long double == 64-bit double.

3 Cephes can be configured to deal with 'wrong' endianness using a macro definition in the configuration header. This might be derived from a Boost header?

4 User Defined Types

I have also used Victor Shoup's NTL 128-bit and arbitrary precision library to show that the function definitions can be successfully extended to include User Defined Types such as these. There are a small but significant group of users who want a higher precision than the native long double provides, especially Microsoft compiler users where long double is identical to double and only 64-bit (about 15 decimal digit). (I'm sorry that this makes the sample code rather messy, but you can skip over the NTL part).

5 I have shown some sample tests using the Boost Unit Test framework. These are only a few spot values, calculated using Cephes DOSbox qcalc.exe 100 decimal digit calculator (output to 40 decimal digits) or NTL. These are intended merely to test a few obvious, random, or exact values, a few corner cases, and around some points at which the algorithms are known to change. They aim to check for general algorithm 'sanity' rather than attempt to evaluate accuracy in detail over the whole range. The Cephes functions have already been tested quite fully and their accuracy and tests are reported in the package.

6 In the test example, for simplicity I have concatenated all the declarations and definitions in the same file. These will of course be in separate header and/or source files.

7 I have not concerned myself with efficiency or speed and assumed that the compiler will inline and otherwise optimise away inefficiencies from wrapping. In some cases, the float versions may simply do the calculation in double and return a conversion to float: it may be the best speed and/or accuracy anyway.

8 Implementations will always have to make some compromises between size, speed and accuracy.

9 Cephes deals with errors (for example, from wrong parameters values) by the then conventional method of calling matherr, setting a global math error no rather than throwing exceptions, and printing a message. But I note that the system-wide error numbers (XENIX) in the Microsoft error.h include files are undocumented, and conflict with the Cephes codes. And C99 (and thus TR1?) no longer REQUIRE a math error to set errno. The simplest solution would be to change the existing matherr function to set errno and NOT print any message: users would always be free to change this module to suit their other requirements. I would NOT wish to implement the C99 exception handling, nor C++ floating_point exceptions.

Detailed recommendations about if and how this should be made more 'standard', perhaps using error.h would be welcomed.

10 Cephes could provide test and values for NaNs and infinity etc, but only for some floating- point layouts. These C99 functions are _essential_, but outside the scope of my work at present.

11 I have assumed that compilation and link option choices will deal with floating point instruction set variations, including 'intrinsic' functions like log, exp, sin, cos. This will complicate building the math library and provide potential for confusion at link time.

12 The Cephes code produces a lot of warnings when compiled in 'strict' mode. I have ignored these, although they suggest that the C code does not meet todays 'picky' standards. However a full revision would be a big undertaking with risks of introducing errors: my impression is that most of the warnings look spurious.

13 Some have suggested that a statistics package might be more useful, but I believe this is NOT what should be in the Standard library. (Even producing means and variances is difficult to generalise when the data may be in many different containers, some partly filled: Boost developments like range may make this more practical in future). The functions proposed are the building blocks with which many different statistics packages can be constructed.

14 Finally may I reiterate that the objective of this implementation is to persuade the C and C++ Standards groups to standardize the signatures of these functions: it will be possible for better implementations to be produced in the future by both open source and commercial organisations like Dinkumware.

Before I undertake the tedious task of wrapping all the proposed TR2 functions (and as many of the C99 and TR1 functions as possible), and presenting the collection for review, I would like feedback from Boosters critical eyes.

Bear in mind that, in all, there are nearly ***500*** function signatures!

(This is 500 signatures without dealing with complex, which I have ignored so far - though many are implemented in the Cephes library).

I would especially like to avoid a lot of bright new ideas emerging at the review stage, as seems to happen all too often. Speak now or forever hold thy peace, please!

Thank you.

Paul

For those want to see code, a simple demo is at
http://www.hetp.u-net.com/public/mathFuncDemo1.cpp
http://www.hetp.u-net.com/public/math2.cpp
http://www.hetp.u-net.com/public/math2.hpp

and two samples, simple and with UDTs of unit testing
http://www.hetp.u-net.com/public/unitTestFunc1.cpp
http://www.hetp.u-net.com/public/unitTestFunc2.cpp