Google Summer of Code 2013 Project Proposal
Bernoulli Numbers and Applications to Special Function

Boost.Math

May, 1

I am very grateful to Christopher Kormanyos for the assistance in preparing this proposal.

1 Introduction

The **Bernoulli numbers** B_n are a sequence of rational numbers with deep connections to number theory. It can be defined by the generating functions [1]:

(1) $$\sum_{k=0}^{\infty} \frac{B_k x^k}{k!} = \frac{x}{e^x - 1}$$

It can be used for in the Euler–MacLaurin formula [2]:

(2) $$\sum_{a \le k < b} f(k) = \int_a^b f(x)\,dx + \sum_{k=1}^{n} \frac{B_k}{k!} f^{(k-1)}(x)\Big|_a^b + R_m$$

(3) $$where\ R_m = (-1)^{m+1} \int_a^b \frac{B_m(\{x\})}{m!}\, f^{(m)}(x)\,dx$$

Unfortunately, the corresponding recurrence (1) for the Bernoulli numbers is numerically unstable. An even better, and perfectly stable, way to compute Bernoulli numbers is to exploit their relationship with the tangent numbers T_j, defined by:

(4) $$\tan x = \sum_{j \ge 1}^{\infty} T_j \frac{x^{2j-1}}{(2j-1)!}$$

The Bernoulli numbers can be expressed in terms of tangent numbers:

(5) $$B_{j=} \begin{cases} 1 & j = 0 \\ -1/2 & j = 1 \\ \dfrac{(-1)^{\frac{j}{2}-1} j T_{\frac{j}{2}}}{(4^j - 2^j)} & j > 0\ and\ j\ is\ even \\ 0 & otherwise \end{cases}$$

The **polygamma function** of order m is a meromorphic function on and defined as the *(m+1)* derivative of the logarithm of the gamma function [3]:

(6) $$\psi^{(m)}(z) = \frac{d^m}{dz^m}\psi(z) = \frac{d^{(m+1)}}{dz^{(m+1)}} ln\Gamma(z)$$

These non-converging series can be used to get quickly an approximation value with a certain numeric at-least-precision for large arguments:

(7) $$\psi^{(m)}(z) = (-1)^{(m+1)} \sum_{k=0}^{\infty} \frac{(k+m-1)!}{k!} \frac{B_k}{z^{k+m}}, \qquad m \ge 1$$

$$\psi^{(0)}(z) = ln(z) - \sum_{k=1}^{\infty} \frac{B_k}{kz^k}, \qquad m = 0$$

Gamma function is widely used. The gamma function finds application in such diverse areas as quantum physics, astrophysics and fluid dynamics. Also, it is related with Bessel function which are used in spectral analysis and implementing linear filters as well as other special functions: beta function, K-function, Barnes G-function. In the statistics commonly used gamma distribution, which special cases are the exponential distribution and chi-squared distribution. For example, the gamma distribution was used to predict the time between occurrences of earthquakes.
If the real part of the complex number z is positive (*Re(z) > 0*), then the integral [4]:

$$\Gamma(z) = \int_0^{\infty} \frac{t^z e^{-t}}{t} dt \tag{8}$$

Complex values of the gamma function can be computed numerically with arbitrary precision using two types of approximation:

1. Lanczos approximation:

$$\Gamma(z+1) = \sqrt{2\pi}\left(z + g + \frac{1}{2}\right)^{z+\frac{1}{2}} e^{-\left(z+g+\frac{1}{2}\right)} A_g(z)$$
$$A_g(z) = \frac{1}{2}p_0(g) + p_1(g)\frac{z}{z+1} + p_2(g)\frac{z(z+1)}{(z+1)(z+2)} + \cdots \tag{9}$$

Here g is a constant that may be chosen arbitrarily subject to the restriction that *Re(z+g+1/2) > 0*. The coefficients *p*, which depend on *g*, are slightly more difficult to calculate:

$$p_k(g) = \sum_{a=0}^{k} C(2k+1, 2a+1)\frac{\sqrt{2\pi}}{\pi}\left(a - \frac{1}{2}\right)!\left(a + g + \frac{1}{2}\right)^{-\left(a+\frac{1}{2}\right)} e^{a+g+\frac{1}{2}} \tag{10}$$

with *C(i,j)* denoting the *(i, j)* element of the Chebyshev polynomial coefficient matrix.

2. **Stirling's approximation**:

$$ln\left(\Gamma(z)\right) = \left(z - \frac{1}{2}\right) ln(z) - z + \frac{1}{2}\ln(2\pi) + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n(2n-1)z^{2n-1}}$$
$$for\ |arg(z)| < \pi \tag{11}$$

The current implementation of gamma function in the Boost library uses a Lanczos-type approximation that loses digits for high precision and is also limited to about 100 decimal digits. Stirling's approximation gives more precision but requires Bernoulli numbers.

All the above shows that, currently existing functions are not absolutely accurate. It limits the range of implementation and delays the development of the related areas. I believe that further research and improvements are required and the Bernoulli numbers project is the most promising in the field.

2 Goals in this GSoC

- Implement thread-safe caching function of calculation the Bernoulli numbers which will be based on tangent numbers calculation.
- Implement polygamma functions of positive integer order based on calculation Bernoulli numers.
- Improve tgamma/lgamma functions for multiprecision types based on Stirling's approximation.
- Develop documentation, tests and manual for these functions.
- Optional: add support for the Hurwitz zeta function.

3 Code design

```
/*
 * This class is for Bernoulli number calculation. It uses caching concept.
 * It uses tangent numbers numbers to get Berniulli numbers.
 * Formal description of algorithm (Tn is n-th tangent number):
 * T1 <-1
 * for k from 2 to n
 *      Tk <-(k-1)Tk-1
 *  for k from 2 to n
 *      for j from k to n
 *          Tj <-(j-k)Tj-1+ (j-k+2)T
 *  return T1,T2,...,Tn
 *
 *  Bernoulli numbers are calculated by formula (5).
 *  For more information see Richard P. Brent and David Harvey,
 *  "Fast Computation of Bernoulli, Tangent and Secant Numbers", page 12, 6.1.
 */
template <class T, class Policy>
class bernulli_numbers
{

private:

    //Data

    typedef std::vector<T> Cache;
    const Cache & cache;

    // Methods

    /*
     * This function calculates the first n Bernoulli numbers and stores it to cache.
     *
     * Parameter: amount >= 0, it is amount of Bernoulli numbers which will be saved in cache.
     */
    void calculate_first_n_numbers(const unsigned amount)
    {
        //...
        // this function uses mutable singleton, for example
        boost::serialization::singleton<Cache>::get_mutable_instance().resize(n);
        // boost::call_once in initialize method guarantees thread safety.
        //...
    };
    /*
     * This method returns n-th bernoulli number from the cache.
     * It can be used only if cache table has been initialized.
     * Note: This method has to be called after 'calculate_first_n_numbers' method.

     * Parameter: n >= 0, n < cache_size
     * Returns: n-rh Bernoulli number from cache
     */
    T get_number_from_cache(const unsigned n) const;
```

```
52.
53. public:
54.
55.     // Constructor
56.     bernoulli_numbers()
57.     {
58.         // Because of only one instance of cache is needed the singleton pattern is used.
59.         cache = boost::serialization::singleton< Cache >::get_const_instance();
60.         // ...
61.     };
62.
63.     //This method fills cache with Bernoulli numbers.
64.     void initialize(const unsigned amount_num_in_cache,
65.                     const Policy& pol = policies::policy<>())
66.     {
67.         boost::once_flag once = BOOST_ONCE_INIT;
68.         boost::call_once(boost::bind(&bernoulli_numbers::calculate_first_n_numbers,
69.                                      amount_num_in_cache_table),
70.                          once_flag& flag);
71.         // ...
72.     };
73.
74.     /*
75.      * If the class has not been initialized or if the parameter n is more than cache size
76.      * this method will calculate Bernoulli number singly.
77.      * Otherwise it calls get_number_from_cache() and returns value from cache.
78.      *
79.      * Parameter: n >= 0
80.      * Returns: n-rh Bernoulli number
81.      */
82.     T () (const unsigned n,
83.           const Policy& pol = policies::policy<>()) const;
84. };
85.
86. // This function calculate gamma function of z using Bernoulli numbers for Stirlings approxima
    tion, formula (11).
87. template <class T, class Policy>
88. T gamma_bn (T z, const Policy& pol = policies::policy<>());
89.
90. /*
91.  * This function uses Bernoulli numbers to calculate
92.  * polygamma function of z for positive order m, formula (7).
93.  */
94. template <class T, class Policy>
95. T polygamma_bn (T z, unsigned m, const Policy& pol = policies::policy<>());
```

Examples of use:

```
1.  // Create Bernoulli numbers object
2.  bernoulli_numbers Bn;
3.  // Cache first 100 numbers
4.  Bn.initialize(100);
5.  // Get fast(with cache) thread safe access to Berniulli numbers
6.  for (int i = 0; i < precision_limit; i += 2) {
7.  //...
8.  double bn = bernoulli_numbers(6);
9.  //...
10. }
11.
12. // Get gamma function of 5;
13. double gamma_5 = gamma_bn<double>(5.0);
14.
15. //Get polygamma function of order 2 of 3
16. double polygamma_5 = polygamma<double>(3.0, 2);
```

4 Schedule

1. Present - May 27:	Reviewing all materials about Bernoulli numbers, tangent numbers and Stirling's approximation, particularly its calculation methods, including modern methods. Examining the code that exists to adapt it later to Boost requirements. Analyzing if Boost.Math and other Boost tools must be used to implement the task.
2. May 27 – June 17:	(Close collaboration with my mentor.) Completing the detailed description of my task. Analyzing the algorithms which I investigated in May. Making an overview of theirs advantages and disadvantages. Deciding on the final design.
3. June 17 – July 1:	Developing Bernoulli numbers function. Providing draft versions to my mentor. Correcting the draft according to mentor's remarks on code style and implementation.
4. July 1 – July 22	Developing polygamma function.
5. July 22 – July 29:	Refactoring. Prepare for mid-term evaluations.
6. July 29 – August 12:	Investigating tgamma/lgamma implementation, multiprecision types and materials about Hurwitz zeta function. Discuss algorithms for its implementation with mentor.
7. August 12 –August 26:	Implementing these algorithms and updating the current functions.
8. August 26 – September 2:	Refactoring.
9. September 2 - September 16	Completing the documentation, tests and manuals for all functions I have implemented.
10. September 16 - September 23	Finalization.

5 Background Information

1. *Personal information:*

Name:	Artemy Margaritov
University	Moscow Institute of Physics and Technology
Course:	Applied physics and mathematics
Degree/program:	I am in B.Sc. program, third year undergraduate student
Email:	artemy.margaritov@gmail.com
Availability:	I am ready to work on the project 60 hours in May and 100 hours per month in summer.

2. *Objective*

My ambition is to join open source code community and to take part in Bernoulli numbers and Special Function project which is important for modeling a variety of natural and technological processes.

Due to my experience in mathematics and programming and my profound interest in the area I believe I can contribute with full efforts for the development of the project.

3. *Educational background*

The curriculum included the following courses:

- mathematical analysis, including complex analysis and differential equations
- discrete mathematics, analytic geometry
- computer science
- calculus mathematics
- concurrent computing, including OpenMP and OpenMPI
- operating systems

Having completed the courses I have significantly improved my mathematical programming skills.

4. *Programming background*

You can see the rating of my knowledge in range 0 – 5:

C++:	4
C++ Standard Library:	3
Boost C++ Libraries:	2
Subversion:	4

5. *Work Experience*

I have been working for Intel Company as an intern in software and service group for one year. My responsibility is C++ programming and during last year I got skills to work on big project and developing clear, robust and flexible design of c++ program. Also, I have learned to organize and plan my work and make reports and presentation.

6. *Technical skills*

I am experienced in the use of Linux. I am closely familiar with Eclipse and I am proficient in a range of shell environment including vim, bash, gcc. Also I have worked with Doxygen, SVN, Bugzilla

References

1 R. P. Brent, P. Zimmermann, Modern Computer Arithmetic

2 Graham, Ronald L.; Knuth, Donald E.; Patashnik, Concrete Mathematics

3 Polygamma function http://en.wikipedia.org/wiki/Polygamma_function

4 Gamma function http://en.wikipedia.org/wiki/Gamma_function

5 Bernoulli numbers http://en.wikipedia.org/wiki/Bernoulli_number

6 Richard P. Brent , David Harvey, Fast Computation of Bernoulli, Tangent and Secant Numbers,