
gcem

Keith O'Hara

Aug 26, 2023

EXAMPLES

| | |
|--------------------------------------|-----------|
| 1 Status | 3 |
| 2 General Syntax | 5 |
| 3 Contents | 7 |
| 3.1 Examples | 7 |
| 3.2 Mathematical functions | 8 |
| Index | 25 |

GCE-Math (**G**eneralized **C**onstant **E**xpression **M**ath) is a templated C++ library enabling compile-time computation of mathematical functions.

- The library is written in C++11 `constexpr` format, and is C++11/14/17/20 compatible.
- Continued fraction and series expansions are implemented using recursive templates.
- The `gcm::` syntax is identical to that of the C++ standard library (`std::`).
- Tested and accurate to floating-point precision against the C++ standard library.
- Released under a permissive, non-GPL license.

Author: Keith O'Hara

License: Apache 2.0

**CHAPTER
ONE**

STATUS

The library is actively maintained, and is still being extended. A list of features includes:

- **basic library functions:**
 - `abs, max, min, pow, sqrt, inv_sqrt`
 - `ceil, floor, round, trunc, fmod,`
 - `exp, expm1, log, log1p, log2, log10` and more
- **trigonometric functions:**
 - basic: `cos, sin, tan`
 - inverse: `acos, asin, atan, atan2`
- **hyperbolic (area) functions:**
 - `cosh, sinh, tanh, acosh, asinh, atanh`
- **algorithms:**
 - `gcd, lcm`
- **special functions:**
 - factorials and the binomial coefficient: `factorial, binomial_coef`
 - beta, gamma, and multivariate gamma functions: `beta, lbeta, lgamma, tgamma, lmgamma`
 - the Gaussian error function and inverse error function: `erf, erf_inv`
 - (regularized) incomplete beta and incomplete gamma functions: `incomplete_beta, incomplete_gamma`
 - inverse incomplete beta and incomplete gamma functions: `incomplete_beta_inv, incomplete_gamma_inv`

CHAPTER
TWO

GENERAL SYNTAX

GCE-Math functions are written as C++ templates with `constexpr` specifiers. For example, the Gaussian error function (`erf`) is defined as:

```
template<typename T>
constexpr
return_t<T>
erf(const T x) noexcept;
```

A set of internal templated `constexpr` functions will implement a continued fraction expansion and return a value of type `return_t<T>`. The output type ('`return_t<T>`') is generally determined by the input type, e.g., `int`, `float`, `double`, `long double`, etc; when `T` is an intergral type, the output will be upgraded to `return_t<T> = double`, otherwise `return_t<T> = T`. For types not covered by `std::is_integral`, recasts should be used.

CONTENTS

3.1 Examples

To calculate 10!:

```
#include "gcem.hpp"

int main()
{
    constexpr int x = 10;
    constexpr int res = gcem::factorial(x);

    return 0;
}
```

Inspecting the assembly code generated by Clang:

```
push    rbp
mov     rbp,  rsp
xor     eax,  eax
mov     dword ptr [rbp - 4],  0
mov     dword ptr [rbp - 8],  10
mov     dword ptr [rbp - 12], 3628800
pop    rbp
ret
```

We see that a function call has been replaced by a numeric value ($10! = 3628800$).

Similarly, to compute the log-Gamma function at a point:

```
#include "gcem.hpp"

int main()
{
    constexpr long double x = 1.5;
    constexpr long double res = gcem::lgamma(x);

    return 0;
}
```

Assembly code:

```
.LCPI0_0:
    .long    1069547520          # float 1.5
.LCPI0_1:
    .quad    -622431863250842976   # x86_fp80 -0.120782237635245222719
    .short    49147
    .zero     6
main:                                # @main
    push    rbp
    mov     rbp, rsp
    xor     eax, eax
    mov     dword ptr [rbp - 4], 0
    fld     dword ptr [rip + .LCPI0_0]
    fstp   tbyte ptr [rbp - 32]
    fld     tbyte ptr [rip + .LCPI0_1]
    fstp   tbyte ptr [rbp - 48]
    pop    rbp
    ret
```

3.1.1 Test suite

To build the full test suite:

```
# clone gcem from GitHub
git clone -b master --single-branch https://github.com/kthohr/gcem ./gcm
# compile tests
cd ./gcm/tests
make
./run_tests
```

3.2 Mathematical functions

3.2.1 Algorithms

```
template<typename T1, typename T2>
constexpr common_t<T1, T2> gcd(const T1 a, const T2 b) noexcept
```

Compile-time greatest common divisor (GCD) function.

Parameters

- **a** – integral-valued input.
- **b** – integral-valued input.

Returns

the greatest common divisor between integers a and b using a Euclidean algorithm.

```
template<typename T1, typename T2>
constexpr common_t<T1, T2> lcm(const T1 a, const T2 b) noexcept
```

Compile-time least common multiple (LCM) function.

Parameters

- **a** – integral-valued input.

- **b** – integral-valued input.

Returns

the least common multiple between integers **a** and **b** using the representation

$$\text{lcm}(a, b) = \frac{|ab|}{\text{gcd}(a, b)}$$

where $\text{gcd}(a, b)$ denotes the greatest common divisor between a and b .

| | |
|------------------|-------------------------|
| <code>gcd</code> | greatest common divisor |
| <code>lcm</code> | least common multiple |

3.2.2 Basic functions

template<typename **T**>
constexpr **T abs**(const **T** **x**) noexcept

Compile-time absolute value function.

Parameters

x – a real-valued input.

Returns

the absolute value of **x**, $|x|$.

template<typename **T**>
constexpr return_t<**T**> **ceil**(const **T** **x**) noexcept

Compile-time ceil function.

Parameters

x – a real-valued input.

Returns

computes the ceiling-value of the input.

template<typename **T1**, typename **T2**>
constexpr **T1 copysign**(const **T1** **x**, const **T2** **y**) noexcept

Compile-time copy sign function.

Parameters

- **x** – a real-valued input
- **y** – a real-valued input

Returns

replace the signbit of **x** with the signbit of **y**.

template<typename **T**>
constexpr return_t<**T**> **exp**(const **T** **x**) noexcept

Compile-time exponential function.

Parameters

x – a real-valued input.

Returns

$\exp(x)$ using

$$\exp(x) = \frac{1}{1 - \frac{x}{1 + x - \frac{\frac{1}{2}x}{1 + \frac{1}{2}x - \frac{\frac{1}{3}x}{1 + \frac{1}{3}x - \ddots}}}}$$

The continued fraction argument is split into two parts: $x = n + r$, where n is an integer and $r \in [-0.5, 0.5]$.

```
template<typename T>
constexpr return_t<T> expm1(const T x) noexcept
```

Compile-time exponential-minus-1 function.

Parameters

x – a real-valued input.

Returns

$\exp(x) - 1$ using

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

```
template<typename T>
constexpr T factorial(const T x) noexcept
```

Compile-time factorial function.

Parameters

x – a real-valued input.

Returns

Computes the factorial value $x!$. When **x** is an integral type (`int`, `long int`, etc.), a simple recursion method is used, along with table values. When **x** is real-valued, `factorial(x) = tgamma(x+1)`.

```
template<typename T>
constexpr return_t<T> floor(const T x) noexcept
```

Compile-time floor function.

Parameters

x – a real-valued input.

Returns

computes the floor-value of the input.

```
template<typename T1, typename T2>
constexpr common_return_t<T1, T2> fmod(const T1 x, const T2 y) noexcept
```

Compile-time remainder of division function.

Parameters

- **x** – a real-valued input.
- **y** – a real-valued input.

Returns

computes the floating-point remainder of x/y (rounded towards zero) using

$$\text{fmod}(x, y) = x - \text{trunc}(x/y) \times y$$

```
template<typename T1, typename T2>
constexpr common_return_t<T1, T2> hypot(const T1 x, const T2 y) noexcept
```

Compile-time Pythagorean addition function.

Parameters

- **x** – a real-valued input.
- **y** – a real-valued input.

Returns

Computes $x \oplus y = \sqrt{x^2 + y^2}$.

```
template<typename T>
constexpr return_t<T> log(const T x) noexcept
```

Compile-time natural logarithm function.

Parameters

x – a real-valued input.

Returns

$\log_e(x)$ using

$$\begin{aligned} \log\left(\frac{1+x}{1-x}\right) &= \frac{2x}{1 - \frac{x^2}{3 - \frac{4x^2}{5 - \frac{9x^3}{7 - \ddots}}}}, \quad x \in [-1, 1] \end{aligned}$$

The continued fraction argument is split into two parts: $x = a \times 10^c$, where c is an integer.

```
template<typename T>
constexpr return_t<T> log1p(const T x) noexcept
```

Compile-time natural-logarithm-plus-1 function.

Parameters

x – a real-valued input.

Returns

$\log_e(x + 1)$ using

$$\log(x + 1) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} x^k}{k}, \quad |x| < 1$$

```
template<typename T>
constexpr return_t<T> log2(const T x) noexcept
```

Compile-time binary logarithm function.

Parameters

x – a real-valued input.

Returns

$\log_2(x)$ using

$$\log_2(x) = \frac{\log_e(x)}{\log_e(2)}$$

```
template<typename T>
constexpr return_t<T> log10(const T x) noexcept
```

Compile-time common logarithm function.

Parameters

- **x** – a real-valued input.

Returns

$\log_{10}(x)$ using

$$\log_{10}(x) = \frac{\log_e(x)}{\log_e(10)}$$

```
template<typename T1, typename T2>
constexpr common_t<T1, T2> max(const T1 x, const T2 y) noexcept
```

Compile-time pairwise maximum function.

Parameters

- **x** – a real-valued input.
- **y** – a real-valued input.

Returns

Computes the maximum between **x** and **y**, where **x** and **y** have the same type (e.g., `int`, `double`, etc.)

```
template<typename T1, typename T2>
constexpr common_t<T1, T2> min(const T1 x, const T2 y) noexcept
```

Compile-time pairwise minimum function.

Parameters

- **x** – a real-valued input.
- **y** – a real-valued input.

Returns

Computes the minimum between **x** and **y**, where **x** and **y** have the same type (e.g., `int`, `double`, etc.)

```
template<typename T1, typename T2>
constexpr common_t<T1, T2> pow(const T1 base, const T2 exp_term) noexcept
```

Compile-time power function.

Parameters

- **base** – a real-valued input.
- **exp_term** – a real-valued input.

Returns

Computes **base** raised to the power **exp_term**. In the case where **exp_term** is integral-valued, recursion by squaring is used, otherwise $\text{base}^{\text{exp_term}} = e^{\text{exp_term} \log(\text{base})}$

```
template<typename T>
```

```
constexpr return_t<T> round(const T x) noexcept
```

Compile-time round function.

Parameters

x – a real-valued input.

Returns

computes the rounding value of the input.

```
template<typename T>
```

```
constexpr bool signbit(const T x) noexcept
```

Compile-time sign bit detection function.

Parameters

x – a real-valued input

Returns

return true if **x** is negative, otherwise return false.

```
template<typename T>
```

```
constexpr int sgn(const T x) noexcept
```

Compile-time sign function.

Parameters

x – a real-valued input

Returns

a value y such that

$$y = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

```
template<typename T>
```

```
constexpr return_t<T> sqrt(const T x) noexcept
```

Compile-time square-root function.

Parameters

x – a real-valued input.

Returns

Computes \sqrt{x} using a Newton-Raphson approach.

```
template<typename T>
```

```
constexpr return_t<T> inv_sqrt(const T x) noexcept
```

Compile-time inverse-square-root function.

Parameters

x – a real-valued input.

Returns

Computes $1/\sqrt{x}$ using a Newton-Raphson approach.

```
template<typename T>
```

```
constexpr return_t<T> trunc(const T x) noexcept
```

Compile-time trunc function.

Parameters

x – a real-valued input.

Returns

computes the trunc-value of the input, essentially returning the integer part of the input.

| | |
|------------------|--|
| <i>abs</i> | absolute value |
| <i>ceil</i> | ceiling function |
| <i>copysign</i> | copy sign function |
| <i>exp</i> | exponential function |
| <i>expm1</i> | exponential minus 1 function |
| <i>factorial</i> | factorial function |
| <i>floor</i> | floor function |
| <i>fmod</i> | remainder of division function |
| <i>hypot</i> | Pythagorean addition function |
| <i>log</i> | natural logarithm function |
| <i>log1p</i> | natural logarithm 1 plus argument function |
| <i>log2</i> | binary logarithm function |
| <i>log10</i> | common logarithm function |
| <i>max</i> | maximum between two numbers |
| <i>min</i> | minimum between two numbers |
| <i>pow</i> | power function |
| <i>round</i> | round function |
| <i>signbit</i> | sign bit function |
| <i>sgn</i> | sign function |
| <i>sqrt</i> | square root function |
| <i>inv_sqrt</i> | inverse square root function |
| <i>trunc</i> | truncate function |

3.2.3 Hyperbolic functions

Table of contents

- *Hyperbolic functions*
- *Inverse hyperbolic functions*

Hyperbolic functions

```
template<typename T>
constexpr return_t<T> cosh(const T x) noexcept
    Compile-time hyperbolic cosine function.
```

Parameters

x – a real-valued input.

Returns

the hyperbolic cosine function using

$$\cosh(x) = \frac{\exp(x) + \exp(-x)}{2}$$

```
template<typename T>
```

```
constexpr return_t<T> sinh(const T x) noexcept
```

Compile-time hyperbolic sine function.

Parameters

x – a real-valued input.

Returns

the hyperbolic sine function using

$$\sinh(x) = \frac{\exp(x) - \exp(-x)}{2}$$

```
template<typename T>
```

```
constexpr return_t<T> tanh(const T x) noexcept
```

Compile-time hyperbolic tangent function.

Parameters

x – a real-valued input.

Returns

the hyperbolic tangent function using

$$\tanh(x) = \frac{x}{1 + \frac{x^2}{3 + \frac{x^2}{5 + \frac{x^2}{7 + \ddots}}}}$$

Inverse hyperbolic functions

```
template<typename T>
```

```
constexpr return_t<T> acosh(const T x) noexcept
```

Compile-time inverse hyperbolic cosine function.

Parameters

x – a real-valued input.

Returns

the inverse hyperbolic cosine function using

$$\text{acosh}(x) = \ln \left(x + \sqrt{x^2 - 1} \right)$$

```
template<typename T>
```

```
constexpr return_t<T> asinh(const T x) noexcept
```

Compile-time inverse hyperbolic sine function.

Parameters

x – a real-valued input.

Returns

the inverse hyperbolic sine function using

$$\text{asinh}(x) = \ln \left(x + \sqrt{x^2 + 1} \right)$$

```
template<typename T>
```

```
constexpr return_t<T> atanh(const T x) noexcept  
    Compile-time inverse hyperbolic tangent function.
```

Parameters

x – a real-valued input.

Returns

the inverse hyperbolic tangent function using

$$\operatorname{atanh}(x) = \frac{1}{2} \ln \left(\frac{1+x}{1-x} \right)$$

| | |
|--------------|-------------------------------------|
| <i>cosh</i> | hyperbolic cosine function |
| <i>sinh</i> | hyperbolic sine function |
| <i>tanh</i> | hyperbolic tangent function |
| <i>acosh</i> | inverse hyperbolic cosine function |
| <i>asinh</i> | inverse hyperbolic sine function |
| <i>atanh</i> | inverse hyperbolic tangent function |

3.2.4 Special functions

Table of contents

- *Binomial function*
- *Beta function*
- *Gamma function*
- *Incomplete integral functions*
- *Inverse incomplete integral functions*

Binomial function

```
template<typename T1, typename T2>  
constexpr common_t<T1, T2> binomial_coef(const T1 n, const T2 k) noexcept  
    Compile-time binomial coefficient.
```

Parameters

- **n** – integral-valued input.
- **k** – integral-valued input.

Returns

computes the Binomial coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

also known as ‘n choose k’.

```
template<typename T1, typename T2>
```

constexpr common_return_t<*T1*, *T2*> **log_binomial_coef**(const *T1* n, const *T2* k) noexcept
Compile-time log binomial coefficient.

Parameters

- **n** – integral-valued input.
- **k** – integral-valued input.

Returns

computes the log Binomial coefficient

$$\ln \frac{n!}{k!(n-k)!} = \ln \Gamma(n+1) - [\ln \Gamma(k+1) + \ln \Gamma(n-k+1)]$$

Beta function

template<typename *T1*, typename *T2*>
constexpr common_return_t<*T1*, *T2*> **beta**(const *T1* a, const *T2* b) noexcept
Compile-time beta function.

Parameters

- **a** – a real-valued input.
- **b** – a real-valued input.

Returns

the beta function using

$$B(\alpha, \beta) := \int_0^1 t^{\alpha-1} (1-t)^{\beta-1} dt = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

where Γ denotes the gamma function.

template<typename *T1*, typename *T2*>
constexpr common_return_t<*T1*, *T2*> **lbeta**(const *T1* a, const *T2* b) noexcept
Compile-time log-beta function.

Parameters

- **a** – a real-valued input.
- **b** – a real-valued input.

Returns

the log-beta function using

$$\ln B(\alpha, \beta) := \ln \int_0^1 t^{\alpha-1} (1-t)^{\beta-1} dt = \ln \Gamma(\alpha) + \ln \Gamma(\beta) - \ln \Gamma(\alpha+\beta)$$

where Γ denotes the gamma function.

Gamma function

```
template<typename T>
constexpr return_t<T> tgamma(const T x) noexcept
```

Compile-time gamma function.

Parameters

x – a real-valued input.

Returns

computes the true gamma function

$$\Gamma(x) = \int_0^\infty y^{x-1} \exp(-y) dy$$

using a polynomial form:

$$\Gamma(x + 1) \approx (x + g + 0.5)^{x+0.5} \exp(-x - g - 0.5) \sqrt{2\pi} \left[c_0 + \frac{c_1}{x+1} + \frac{c_2}{x+2} + \cdots + \frac{c_n}{x+n} \right]$$

where the value g and the coefficients (c_0, c_1, \dots, c_n) are taken from Paul Godfrey, whose note can be found here: <http://my.fit.edu/~gabdo/gamma.txt>

```
template<typename T>
constexpr return_t<T> lgamma(const T x) noexcept
```

Compile-time log-gamma function.

Parameters

x – a real-valued input.

Returns

computes the log-gamma function

$$\ln \Gamma(x) = \ln \int_0^\infty y^{x-1} \exp(-y) dy$$

using a polynomial form:

$$\Gamma(x + 1) \approx (x + g + 0.5)^{x+0.5} \exp(-x - g - 0.5) \sqrt{2\pi} \left[c_0 + \frac{c_1}{x+1} + \frac{c_2}{x+2} + \cdots + \frac{c_n}{x+n} \right]$$

where the value g and the coefficients (c_0, c_1, \dots, c_n) are taken from Paul Godfrey, whose note can be found here: <http://my.fit.edu/~gabdo/gamma.txt>

```
template<typename T1, typename T2>
constexpr return_t<T1> lmgamma(const T1 a, const T2 p) noexcept
```

Compile-time log multivariate gamma function.

Parameters

- **a** – a real-valued input.
- **p** – integral-valued input.

Returns

computes log-multivariate gamma function via recursion

$$\Gamma_p(a) = \pi^{(p-1)/2} \Gamma(a) \Gamma_{p-1}(a - 0.5)$$

where $\Gamma_1(a) = \Gamma(a)$.

Incomplete integral functions

```
template<typename T>
constexpr return_t<T> erf(const T x) noexcept
```

Compile-time Gaussian error function.

Parameters

- x** – a real-valued input.

Returns

computes the Gaussian error function

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt$$

using a continued fraction representation:

$$\text{erf}(x) = \frac{2x}{\sqrt{\pi}} \exp(-x^2) \cfrac{1}{1 - 2x^2 + \cfrac{4x^2}{3 - 2x^2 + \cfrac{8x^2}{5 - 2x^2 + \cfrac{12x^2}{7 - 2x^2 + \ddots}}}}$$

```
template<typename T1, typename T2, typename T3>
constexpr common_return_t<T1, T2, T3> incomplete_beta(const T1 a, const T2 b, const T3 z) noexcept
```

Compile-time regularized incomplete beta function.

Parameters

- a** – a real-valued, non-negative input.
- b** – a real-valued, non-negative input.
- z** – a real-valued, non-negative input.

Returns

computes the regularized incomplete beta function,

$$\frac{B(z; \alpha, \beta)}{B(\alpha, \beta)} = \frac{1}{B(\alpha, \beta)} \int_0^z t^{\alpha-1} (1-t)^{\beta-1} dt$$

using a continued fraction representation, found in the Handbook of Continued Fractions for Special Functions, and a modified Lentz method.

$$\frac{B(z; \alpha, \beta)}{B(\alpha, \beta)} = \frac{z^\alpha (1-t)^\beta}{\alpha B(\alpha, \beta)} \cfrac{a_1}{1 + \cfrac{a_2}{1 + \cfrac{a_3}{1 + \cfrac{a_4}{1 + \ddots}}}}$$

where $a_1 = 1$ and

$$a_{2m+2} = -\frac{(\alpha+m)(\alpha+\beta+m)}{(\alpha+2m)(\alpha+2m+1)}, \quad m \geq 0$$

$$a_{2m+1} = \frac{m(\beta-m)}{(\alpha+2m-1)(\alpha+2m)}, \quad m \geq 1$$

The Lentz method works as follows: let f_j denote the value of the continued fraction up to the first j terms; f_j is updated as follows:

$$c_j = 1 + a_j/c_{j-1}, \quad d_j = 1/(1 + a_j d_{j-1})$$

$$f_j = c_j d_j f_{j-1}$$

```
template<typename T1, typename T2>
constexpr common_return_t<T1, T2> incomplete_gamma(const T1 a, const T2 x) noexcept
```

Compile-time regularized lower incomplete gamma function.

Parameters

- **a** – a real-valued, non-negative input.
- **x** – a real-valued, non-negative input.

Returns

the regularized lower incomplete gamma function evaluated at (a, x),

$$\frac{\gamma(a, x)}{\Gamma(a)} = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} \exp(-t) dt$$

When a is not too large, the value is computed using the continued fraction representation of the upper incomplete gamma function, $\Gamma(a, x)$, using

$$\begin{aligned} \Gamma(a, x) = \Gamma(a) - & \frac{x^a \exp(-x)}{ax} \\ & a - \frac{x}{a+1 + \frac{x}{a+2 - \frac{(a+1)x}{a+3 + \frac{2x}{a+4 - \ddots}}}} \end{aligned}$$

where $\gamma(a, x)$ and $\Gamma(a, x)$ are connected via

$$\frac{\gamma(a, x)}{\Gamma(a)} + \frac{\Gamma(a, x)}{\Gamma(a)} = 1$$

When $a > 10$, a 50-point Gauss-Legendre quadrature scheme is employed.

Inverse incomplete integral functions

```
template<typename T>
constexpr return_t<T> erf_inv(const T p) noexcept
```

Compile-time inverse Gaussian error function.

Parameters

- p** – a real-valued input with values in the unit-interval.

Returns

Computes the inverse Gaussian error function, a value x such that

$$f(x) := \text{erf}(x) - p$$

is equal to zero, for a given p. GCE-Math finds this root using Halley's method:

$$x_{n+1} = x_n - \frac{f(x_n)/f'(x_n)}{1 - 0.5 \frac{f(x_n)}{f'(x_n)} \frac{f''(x_n)}{f'(x_n)}}$$

where

$$\frac{\partial}{\partial x} \text{erf}(x) = \exp(-x^2), \quad \frac{\partial^2}{\partial x^2} \text{erf}(x) = -2x \exp(-x^2)$$

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_t<T1, T2, T3> incomplete_beta_inv(const **T1** a, const **T2** b, const **T3** p) noexcept
 Compile-time inverse incomplete beta function.

Parameters

- **a** – a real-valued, non-negative input.
- **b** – a real-valued, non-negative input.
- **p** – a real-valued input with values in the unit-interval.

Returns

Computes the inverse incomplete beta function, a value x such that

$$f(x) := \frac{B(x; \alpha, \beta)}{B(\alpha, \beta)} - p$$

equal to zero, for a given p. GCE-Math finds this root using Halley's method:

$$x_{n+1} = x_n - \frac{f(x_n)/f'(x_n)}{1 - 0.5 \frac{f(x_n)}{f'(x_n)} \frac{f''(x_n)}{f'(x_n)}}$$

where

$$\begin{aligned} \frac{\partial}{\partial x} \left(\frac{B(x; \alpha, \beta)}{B(\alpha, \beta)} \right) &= \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \\ \frac{\partial^2}{\partial x^2} \left(\frac{B(x; \alpha, \beta)}{B(\alpha, \beta)} \right) &= \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \left(\frac{\alpha-1}{x} - \frac{\beta-1}{1-x} \right) \end{aligned}$$

template<typename **T1**, typename **T2**>
constexpr common_return_t<T1, T2> incomplete_gamma_inv(const **T1** a, const **T2** p) noexcept
 Compile-time inverse incomplete gamma function.

Parameters

- **a** – a real-valued, non-negative input.
- **p** – a real-valued input with values in the unit-interval.

Returns

Computes the inverse incomplete gamma function, a value x such that

$$f(x) := \frac{\gamma(a, x)}{\Gamma(a)} - p$$

equal to zero, for a given p. GCE-Math finds this root using Halley's method:

$$x_{n+1} = x_n - \frac{f(x_n)/f'(x_n)}{1 - 0.5 \frac{f(x_n)}{f'(x_n)} \frac{f''(x_n)}{f'(x_n)}}$$

where

$$\frac{\partial}{\partial x} \left(\frac{\gamma(a, x)}{\Gamma(a)} \right) = \frac{1}{\Gamma(a)} x^{a-1} \exp(-x)$$
$$\frac{\partial^2}{\partial x^2} \left(\frac{\gamma(a, x)}{\Gamma(a)} \right) = \frac{1}{\Gamma(a)} x^{a-1} \exp(-x) \left(\frac{a-1}{x} - 1 \right)$$

| | |
|-----------------------------|-----------------------------------|
| <i>binomial_coef</i> | binomial coefficient |
| <i>log_binomial_coef</i> | log binomial coefficient |
| <i>beta</i> | beta function |
| <i>lbeta</i> | log-beta function |
| <i>tgamma</i> | gamma function |
| <i>lgamma</i> | log-gamma function |
| <i>lmgamma</i> | log-multivariate gamma function |
| <i>erf</i> | error function |
| <i>incomplete_beta</i> | incomplete beta function |
| <i>incomplete_gamma</i> | incomplete gamma function |
| <i>erf_inv</i> | inverse error function |
| <i>incomplete_beta_inv</i> | inverse incomplete beta function |
| <i>incomplete_gamma_inv</i> | inverse incomplete gamma function |

3.2.5 Trigonometric functions

Table of contents

- *Trigonometric functions*
 - *Inverse trigonometric functions*
-

Trigonometric functions

template<typename T>
constexpr return_t<T> **cos**(const T x) noexcept

Compile-time cosine function.

Parameters

x – a real-valued input.

Returns

the cosine function using

$$\cos(x) = \frac{1 - \tan^2(x/2)}{1 + \tan^2(x/2)}$$

template<typename T>
constexpr return_t<T> **sin**(const T x) noexcept

Compile-time sine function.

Parameters

x – a real-valued input.

Returns

the sine function using

$$\sin(x) = \frac{2 \tan(x/2)}{1 + \tan^2(x/2)}$$

```
template<typename T>
constexpr return_t<T> tan(const T x) noexcept
    Compile-time tangent function.
```

Parameters

x – a real-valued input.

Returns

the tangent function using

$$\begin{aligned} \tan(x) = & \frac{x}{x^2} \\ & 1 - \frac{x^2}{3 - \frac{x^2}{5 - \ddots}} \end{aligned}$$

To deal with a singularity at $\pi/2$, the following expansion is employed:

$$\tan(x) = -\frac{1}{x - \pi/2} - \sum_{k=1}^{\infty} \frac{(-1)^k 2^{2k} B_{2k}}{(2k)!} (x - \pi/2)^{2k-1}$$

where B_n is the n-th Bernoulli number.

Inverse trigonometric functions

```
template<typename T>
constexpr return_t<T> acos(const T x) noexcept
    Compile-time arccosine function.
```

Parameters

x – a real-valued input, where $x \in [-1, 1]$.

Returns

the inverse cosine function using

$$\text{acos}(x) = \text{atan} \left(\frac{\sqrt{1 - x^2}}{x} \right)$$

```
template<typename T>
constexpr return_t<T> asin(const T x) noexcept
    Compile-time arcsine function.
```

Parameters

x – a real-valued input, where $x \in [-1, 1]$.

Returns

the inverse sine function using

$$\text{asin}(x) = \text{atan} \left(\frac{x}{\sqrt{1 - x^2}} \right)$$

```
template<typename T>
constexpr return_t<T> atan(const T x) noexcept
```

Compile-time arctangent function.

Parameters

- x** – a real-valued input.

Returns

the inverse tangent function using

$$\begin{aligned} \text{atan}(x) = & \frac{x}{1 + \frac{x^2}{3 + \frac{4x^2}{5 + \frac{9x^2}{7 + \ddots}}}} \end{aligned}$$

```
template<typename T1, typename T2>
constexpr common_return_t<T1, T2> atan2(const T1 y, const T2 x) noexcept
```

Compile-time two-argument arctangent function.

Parameters

- **y** – a real-valued input.
- **x** – a real-valued input.

Returns

$$\text{atan2}(y, x) = \begin{cases} \text{atan}(y/x) & \text{if } x > 0 \\ \text{atan}(y/x) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \text{atan}(y/x) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ +\pi/2 & \text{if } x = 0 \text{ and } y > 0 \\ -\pi/2 & \text{if } x = 0 \text{ and } y < 0 \end{cases}$$

The function is undefined at the origin, however the following conventions are used.

$$\text{atan2}(y, x) = \begin{cases} +0 & \text{if } x = +0 \text{ and } y = +0 \\ -0 & \text{if } x = +0 \text{ and } y = -0 \\ +\pi & \text{if } x = -0 \text{ and } y = +0 \\ -\pi & \text{if } x = -0 \text{ and } y = -0 \end{cases}$$

| | |
|--------------|----------------------------------|
| <i>cos</i> | cosine function |
| <i>sin</i> | sine function |
| <i>tan</i> | tangent function |
| <i>acos</i> | arccosine function |
| <i>asin</i> | arcsine function |
| <i>atan</i> | arctangent function |
| <i>atan2</i> | two-argument arctangent function |

INDEX

A

`abs (C++ function)`, 9
`acos (C++ function)`, 23
`acosh (C++ function)`, 15
`asin (C++ function)`, 23
`asinh (C++ function)`, 15
`atan (C++ function)`, 24
`atan2 (C++ function)`, 24
`atanh (C++ function)`, 15

B

`beta (C++ function)`, 17
`binomial_coef (C++ function)`, 16

C

`ceil (C++ function)`, 9
`copysign (C++ function)`, 9
`cos (C++ function)`, 22
`cosh (C++ function)`, 14

E

`erf (C++ function)`, 19
`erf_inv (C++ function)`, 20
`exp (C++ function)`, 9
`expm1 (C++ function)`, 10

F

`factorial (C++ function)`, 10
`floor (C++ function)`, 10
`fmod (C++ function)`, 10

G

`gcd (C++ function)`, 8

H

`hypot (C++ function)`, 11

I

`incomplete_beta (C++ function)`, 19
`incomplete_beta_inv (C++ function)`, 21
`incomplete_gamma (C++ function)`, 20

`incomplete_gamma_inv (C++ function)`, 21
`inv_sqrt (C++ function)`, 13

L

`lbeta (C++ function)`, 17
`lcm (C++ function)`, 8
`lgamma (C++ function)`, 18
`lmgamma (C++ function)`, 18
`log (C++ function)`, 11
`log10 (C++ function)`, 12
`log1p (C++ function)`, 11
`log2 (C++ function)`, 11
`log_binomial_coef (C++ function)`, 16

M

`max (C++ function)`, 12
`min (C++ function)`, 12

P

`pow (C++ function)`, 12

R

`round (C++ function)`, 12

S

`sgn (C++ function)`, 13
`signbit (C++ function)`, 13
`sin (C++ function)`, 22
`sinh (C++ function)`, 14
`sqrt (C++ function)`, 13

T

`tan (C++ function)`, 23
`tanh (C++ function)`, 15
`tgamma (C++ function)`, 18
`trunc (C++ function)`, 13