gcem

Feb 01, 2019

EXAMPLES

GCE-Math (**G**eneralized **C**onstant **E**xpression Math) 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 compatible.
- Continued fraction and series expansions are implemented using recursive templates.
- The `gcem::` syntax is identical to the C++ standard library (`std::`).
- Tested and accurate to machine precision against the C++ standard library.
- Released under a permissive, non-GPL license.

Author: Keith O'Hara

License: Apache 2.0

CHAPTER 1

Status

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

- **basic library functions:**
  - `abs`, `max`, `min`, `pow`, `sqrt`
  - `exp`, `expm1`, `log`, `log1p`, 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 2

General Syntax

GCE-Math functions are written as C++ templates with `constexpr` specifiers, the format of which might be confusing to users unfamiliar with template-based programming. As an example, the Gaussian error function (`erf`) is defined as:

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

where a set of internal templated `constexpr` functions will implement a continued fraction expansion to return a value of type `return_t<T>`. This 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.

CHAPTER 3

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()
{
```

(continues on next page)

(continued from previous page)

```
    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 -0.12078223763524522719
        .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 ./gcem
# compile tests
cd ./gcem/tests
make
./run_tests
```

3.2 Mathematical functions

3.2.1 Algorithms

template <**typename** T1, **typename** T2>
constexpr common_t<T1, T2> gcem::**gcd**(**const** T1 *a*, **const** T2 *b*)
Compile-time greatest common divisor (GCD) function.

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

Parameters

- a: integral-valued input.
- b: integral-valued input.

template <**typename** T1, **typename** T2>
constexpr common_t<T1, T2> gcem::**lcm**(**const** T1 a, **const** T2 b)
Compile-time least common multiple (LCM) function.

Return 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.

Parameters

- a: integral-valued input.
- b: integral-valued input.

gcd	greatest common divisor
lcm	least common multiple

3.2.2 Basic functions

template <**typename** T>
constexpr T gcem::**abs**(**const** T x)
Compile-time absolute value function.

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

Parameters

- x: a real-valued input.

template <**typename** T>
constexpr return_t<T> gcem::**exp**(**const** T x)
Compile-time exponential function.

Return $\exp(x)$ using

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

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

Parameters

- x: a real-valued input.

template <**typename** T>
constexpr return_t<T> gcem::**expm1**(**const** T x)
Compile-time exponential-minus-1 function.

Return $\exp(x) - 1$ using

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

Parameters

- x: a real-valued input.

template <typename T>
constexpr T gcem::**factorial**(**const** T *x*)
Compile-time factorial function.

Return 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).

Parameters

- x: a real-valued input.

template <typename T>
constexpr return_t<T> gcem::**log**(**const** T *x*)
Compile-time natural logarithm function.

Return $\log_e(x)$ using

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

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

Parameters

- x: a real-valued input.

template <typename T>
constexpr return_t<T> gcem::**log1p**(**const** T *x*)
Compile-time natural-logarithm-plus-1 function.

Return $\log_e(x + 1)$ using

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

Parameters

- x: a real-valued input.

template <typename T1, **typename** T2>
constexpr common_t<T1, T2> gcem::**max**(**const** T1 *x*, **const** T2 *y*)
Compile-time pairwise maximum function.

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

Parameters

- `x`: a real-valued input.
- `y`: a real-valued input.

template <**typename** T1, **typename** T2>
constexpr common_t<T1, T2> `gcem::min`(**const** T1 *x*, **const** T2 *y*)
Compile-time pairwise minimum function.

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

Parameters

- `x`: a real-valued input.
- `y`: a real-valued input.

template <**typename** T1, **typename** T2>
constexpr common_t<T1, T2> `gcem::pow`(**const** T1 *base*, **const** T2 *exp_term*)
Compile-time power function.

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

Parameters

- `base`: a real-valued input.
- `exp_term`: a real-valued input.

template <**typename** T>
constexpr int `gcem::sgn`(**const** T *x*)
Compile-time sign function.

Return a value y such that

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

Parameters

- `x`: a real-valued input

template <**typename** T>
constexpr return_t<T> `gcem::sqrt`(**const** T *x*)
Compile-time square-root function.

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

Parameters

- `x`: a real-valued input.

abs	absolute value
exp	exponential function
expm1	exponential minus 1 function
factorial	factorial function
log	natural logarithm function
log1p	natural logarithm 1 plus argument function
max	maximum between two numbers
min	minimum between two numbers
pow	power function
sgn	sign function
sqrt	square root function

3.2.3 Hyperbolic functions

template <**typename** T>
constexpr return_t<T> gcem::**cosh**(**const** T x)
Compile-time hyperbolic cosine function.

Return the hyperbolic cosine function using

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

Parameters

- x: a real-valued input.

template <**typename** T>
constexpr return_t<T> gcem::**sinh**(**const** T x)
Compile-time hyperbolic sine function.

Return the hyperbolic sine function using

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

Parameters

- x: a real-valued input.

template <**typename** T>
constexpr return_t<T> gcem::**tanh**(**const** T x)
Compile-time hyperbolic tangent function.

Return the hyperbolic tangent function using

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

Parameters

- x: a real-valued input.

Inverse hyperbolic functions

template <**typename** T>
constexpr return_t<T> gcem::**acosh**(**const** T *x*)
Compile-time inverse hyperbolic cosine function.

Return the inverse hyperbolic cosine function using

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

Parameters

- x: a real-valued input.

template <**typename** T>
constexpr return_t<T> gcem::**asinh**(**const** T *x*)
Compile-time inverse hyperbolic sine function.

Return the inverse hyperbolic sine function using

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

Parameters

- x: a real-valued input.

template <**typename** T>
constexpr return_t<T> gcem::**atanh**(**const** T *x*)
Compile-time inverse hyperbolic tangent function.

Return the inverse hyperbolic tangent function using

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

Parameters

- x: a real-valued input.

cosh	hyperbolic cosine function
sinh	hyperbolic sine function
tanh	hyperbolic tangent function
acosh	inverse hyperbolic cosine function
asinh	inverse hyperbolic sine function
atanh	inverse hyperbolic tangent function

3.2.4 Special functions

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

Return computes the Binomial coefficient

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

also known as 'n choose k '.

Parameters

- n: integral-valued input.
- k: integral-valued input.

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

Return computes the log Binomial coefficient

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

Parameters

- n: integral-valued input.
- k: integral-valued input.

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

Return the beta function using

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

where Γ denotes the gamma function.

Parameters

- a: a real-valued input.
- b: a real-valued input.

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

Return the log-beta function using

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

where Γ denotes the gamma function.

Parameters

- a: a real-valued input.
- b: a real-valued input.

template <typename T>
constexpr return_t<T> gcem::**tgamma**(**const** T x)
Compile-time gamma function.

Return 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 $(c_0, c_1, \ldots, c_n)$ are taken from Paul Godfrey, whose note can be found here: http://my.fit.edu/~gabdo/gamma.txt

Parameters

- x: a real-valued input.

template <typename T>
constexpr return_t<T> gcem::**lgamma**(**const** T x)
Compile-time log-gamma function.

Return 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 $(c_0, c_1, \ldots, c_n)$ are taken from Paul Godfrey, whose note can be found here: http://my.fit.edu/~gabdo/gamma.txt

Parameters

- x: a real-valued input.

template <typename T1, **typename** T2>
constexpr return_t<T1> gcem::**lmgamma**(**const** T1 a, **const** T2 p)
Compile-time log multivariate gamma function.

Return computes log-multivariate gamma function via recursion

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

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

Parameters

- a: a real-valued input.
- p: integral-valued input.

Incomplete integral functions

template <**typename** T>
constexpr return_t<T> gcem::**erf**(**const** T x)

Compile-time Gaussian error function.

Return computes the Gaussian error function

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

using a continued fraction representation:

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

Parameters

- x: a real-valued input.

template <**typename** T1, **typename** T2, **typename** T3>
constexpr common_return_t<T1, T2, T3> gcem::**incomplete_beta**(**const** T1 a, **const** T2 b, **const** T3 z)

Compile-time regularized incomplete beta function.

Return computes the regularized incomplete beta function,

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

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

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

where $a_1 = 1$ and

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

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

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:

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

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

Parameters

- `a`: a real-valued, non-negative input.
- `b`: a real-valued, non-negative input.
- `z`: a real-valued, non-negative input.

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

Compile-time regularized lower incomplete gamma function.

Return 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

$$\Gamma(a,x) = \Gamma(a) - \cfrac{x^a\exp(-x)}{a - \cfrac{ax}{a+1+\cfrac{x}{a+2-\cfrac{(a+1)x}{a+3+\cfrac{2x}{a+4-\ddots}}}}}$$

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.

Parameters

- `a`: a real-valued, non-negative input.
- `x`: a real-valued, non-negative input.

Inverse incomplete integral functions

template <**typename** T>
constexpr return_t<T> `gcem::erf_inv`(**const** T *p*)

Compile-time inverse Gaussian error function.

Return 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

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

Parameters

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

template <**typename** T1, **typename** T2, **typename** T3>
constexpr common_t<T1, T2, T3> gcem::**incomplete_beta_inv**(**const** T1 *a*, **const** T2 *b*, **const** T3 *p*)

Compile-time inverse incomplete beta function.

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

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

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

$$\frac{\partial}{\partial x}\left(\frac{\mathrm{B}(x;\alpha,\beta)}{\mathrm{B}(\alpha,\beta)}\right) = \frac{1}{\mathrm{B}(\alpha,\beta)}x^{\alpha-1}(1-x)^{\beta-1}$$

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

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.

template <**typename** T1, **typename** T2>
constexpr common_return_t<T1, T2> gcem::**incomplete_gamma_inv**(**const** T1 *a*, **const** T2 *p*)

Compile-time inverse incomplete gamma function.

Return 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

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

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

Parameters

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

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

3.2.5 Trigonometric functions

template <typename T>
constexpr return_t<T> gcem::**cos**(**const** T x)
Compile-time cosine function.

Return the cosine function using

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

Parameters

- x: a real-valued input.

template <typename T>
constexpr return_t<T> gcem::**sin**(**const** T x)
Compile-time sine function.

Return the sine function using

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

Parameters

- x: a real-valued input.

template <typename T>
constexpr return_t<T> gcem::**tan**(**const** T x)
Compile-time tangent function.

Return the tangent function using

$$\tan(x) = \cfrac{x}{1 - \cfrac{x^2}{3 - \cfrac{x^2}{5 - \ddots}}}$$

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.

Parameters

- `x`: a real-valued input.

Inverse trigonometric functions

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

Return the inverse cosine function using

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

Parameters

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

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

Return the inverse sine function using

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

Parameters

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

template <typename T>
constexpr return_t<T> `gcem::atan`(**const** T x)
Compile-time arctangent function.

Return the inverse tangent function using

$$\text{atan}(x) = \cfrac{x}{1 + \cfrac{x^2}{3 + \cfrac{4x^2}{5 + \cfrac{9x^2}{7 + \ddots}}}}$$

Parameters

- `x`: a real-valued input.

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

Compile-time two-argument arctangent function.

Return

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

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

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

Parameters

- `y`: a real-valued input.
- `x`: a real-valued input.

cos	cosine function
sin	sine function
tan	tangent function
acos	arccosine function
asin	arcsine function
atan	arctangent function
atan2	two-argument arctangent function

Index

G