statslib

Keith O'Hara

May 14, 2023

GUIDE

StatsLib is a templated C++ library of statistical distribution functions, featuring unique compile-time computing capabilities and seamless integration with several popular linear algebra libraries.

Features

- A header-only library of probability density functions, cumulative distribution functions, quantile functions, and random sampling methods.
- Functions are written in C++11 `constexpr` format, enabling the library to operate as both a compile-time and run-time computation engine.
- Designed with a simple **R**-like syntax.
- Optional vector-matrix functionality, with wrappers to support:
- STL Vectors (`std::vector`)
- Armadillo
- Blaze
- Eigen
- Matrix-based operations are parallelizable with OpenMP.
- Released under a permissive, non-GPL license.

Author: Keith O'Hara

License: Apache Version 2.0

CHAPTER

ONE

INSTALLATION

StatsLib is a header-only library. Simply add the header files to your project using:

```
#include "stats.hpp"
```

The only dependency is the latest version of GCEM

CHAPTER

TWO

CONTENTS

2.1 Syntax

Functions are called using an **R**-like syntax. Some general rules:

- density functions: `stats::d*`. For example, the Normal (Gaussian) density function is called using:

```
stats::dnorm(<value>,<mean parameter>,<standard deviation>);
```

- cumulative distribution functions: `stats::p*`. For example, the Gamma CDF is called using:

```
stats::pgamma(<value>,<shape parameter>,<scale parameter>);
```

- quantile functions: `stats::q*`. For example, the Beta quantile function is called using:

```
stats::qbeta(<value>,<a parameter>,<b parameter>);
```

- random sampling: `stats::r*`. For example, to generate a single draw from the Logistic distribution:

```
stats::rlogis(<location parameter>,<scale parameter>,<seed value or random number engine>
↪);
```

All of these functions have matrix-based equivalents using Armadillo, Blaze, and Eigen dense matrices.

- The pdf, cdf, and quantile functions can take matrix-valued arguments. For example,

```
// Using Armadillo:
arma::mat norm_pdf_vals = stats::dnorm(arma::ones(10,20),1.0,2.0);
```

- The randomization functions (`r*`) can output random matrices of arbitrary size. For example, the following code will generate a 100-by-50 matrix of iid draws from a Gamma(3,2) distribution:

```
// Armadillo:
arma::mat gamma_rvs = stats::rgamma<arma::mat>(100,50,3.0,2.0);

// Blaze:
blaze::DynamicMatrix<double> gamma_rvs = stats::rgamma<blaze::DynamicMatrix<double>>(100,
↪50,3.0,2.0);

// Eigen:
Eigen::MatrixXd gamma_rvs = stats::rgamma<Eigen::MatrixXd>(100,50,3.0,2.0);
```

- All matrix-based operations are parallelizable with OpenMP. For GCC and Clang compilers, simply include the `-fopenmp` option during compilation.

2.2 RNG Seeding

Random number generator seeding is available in two forms: seed values and random number engines.

- Seed values are passed as unsigned integers. For example, to generate a draw from a normal distribution N(1,2) with seed value 1776:

```
stats::rnorm(1,2,1776);
```

- Random engines in StatsLib use the 64-bit Mersenne-Twister generator (`std::mt19937_64`) by default, and are passed by reference. For example:

```
std::mt19937_64 engine(1776);
stats::rnorm(1,2,engine);
```

Notes:

- To use a different random engine type with StatsLib, define `STATS_RNG_ENGINE_TYPE` before including the StatsLib header files. (For example, `#define STATS_RNG_ENGINE_TYPE std::mt19937_64`.)
- Random number generators should be the preferred option over seed values; passing seed values requires generating a new random engine with each function call, which can be computationally intensive if repeated many times.

2.3 Compile-time Functionality

StatsLib is designed to operate equally well as a compile-time computation engine. Compile-time computation allows the compiler to replace function calls (e.g., `dnorm(0,0,1)`) with static values in the source code. That is, functions are evaluated during the compilation process, rather than at run-time. This capability is made possible due to the templated `constexpr` design of the library and can be verified by inspecting the assembly code generated by the compiler.

The compile-time features are enabled using the `constexpr` specifier. The example below computes the pdf, cdf, and quantile function of the Laplace distribution.

```
#include "stats.hpp"

int main()
{

    constexpr double dens_1  = stats::dlaplace(1.0,1.0,2.0); // answer = 0.25
    constexpr double prob_1  = stats::plaplace(1.0,1.0,2.0); // answer = 0.5
    constexpr double quant_1 = stats::qlaplace(0.1,1.0,2.0); // answer = -2.218875...

    return 0;
}
```

Inspecting the assembly code generated by Clang (without any optimization):

```
LCPI0_0:
    .quad   -4611193153885729483    ## double -2.2188758248682015
LCPI0_1:
    .quad   4602678819172646912     ## double 0.5
LCPI0_2:
    .quad   4598175219545276417     ## double 0.25000000000000006
```

(continues on next page)

(continued from previous page)

```
    .section        __TEXT,__text,regular,pure_instructions
    .globl  _main
    .p2align        4, 0x90
_main:                                  ## @main
    push    rbp
    mov     rbp, rsp
    xor     eax, eax
    movsd   xmm0, qword ptr [rip + LCPI0_0] ## xmm0 = mem[0],zero
    movsd   xmm1, qword ptr [rip + LCPI0_1] ## xmm1 = mem[0],zero
    movsd   xmm2, qword ptr [rip + LCPI0_2] ## xmm2 = mem[0],zero
    mov     dword ptr [rbp - 4], 0
    movsd   qword ptr [rbp - 16], xmm2
    movsd   qword ptr [rbp - 24], xmm1
    movsd   qword ptr [rbp - 32], xmm0
    pop     rbp
    ret
```

We see that functions call have been replaced by numeric values.

2.4 Compiler Options

The following options should be declared **before** including the StatsLib header files.

- For inline-only functionality (i.e., no `constexpr` specifiers):

```
#define STATS_GO_INLINE
```

- OpenMP functionality is enabled by default if the `_OPENMP` macro is detected (e.g., by invoking `-fopenmp` with GCC or Clang). To explicitly enable OpenMP features use:

```
#define STATS_USE_OPENMP
```

- To disable OpenMP functionality:

```
#define STATS_DONT_USE_OPENMP
```

- To use StatsLib with Armadillo, Blaze or Eigen:

```
#define STATS_ENABLE_ARMA_WRAPPERS
#define STATS_ENABLE_BLAZE_WRAPPERS
#define STATS_ENABLE_EIGEN_WRAPPERS
```

- To enable wrappers for `std::vector`:

```
#define STATS_ENABLE_STDVEC_WRAPPERS
```

- To specify a random engine type (`stats::rand_engine_t`) to something other than the default of `std::mt19937_64`:

```
#define STATS_RNG_ENGINE_TYPE <your-engine-type>
```

2.5 Examples and Tests

2.5.1 Examples

```
// evaluate the normal PDF at x = 1, mu = 0, sigma = 1
double dval_1 = stats::dnorm(1.0,0.0,1.0);

// evaluate the normal PDF at x = 1, mu = 0, sigma = 1, and return the log value
double dval_2 = stats::dnorm(1.0,0.0,1.0,true);

// evaluate the normal CDF at x = 1, mu = 0, sigma = 1
double pval = stats::pnorm(1.0,0.0,1.0);

// evaluate the Laplacian quantile at p = 0.1, mu = 0, sigma = 1
double qval = stats::qlaplace(0.1,0.0,1.0);

// draw from a t-distribution dof = 30
double rval = stats::rt(30);

// matrix output
arma::mat beta_rvs = stats::rbeta<arma::mat>(100,100,3.0,2.0);

// matrix input
arma::mat beta_cdf_vals = stats::pbeta(beta_rvs,3.0,2.0);
```

2.5.2 Test suite

You can build the tests suite as follows:

```
# compile tests
cd ./tests
./setup
./dens
./configure
make
./dnorm.test
```

2.6 Distributions

2.6.1 Bernoulli Distribution

Table of contents

Density Function

The density function of the Bernoulli distribution:

$$f(x;p) = p^x(1-p)^{1-x} \times \mathbf{1}[x \in \{0,1\}]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T**>
constexpr return_t<*T*> **dbern**(const llint_t x, const *T* prob_par, const bool log_form = false) noexcept

Density function of the Bernoulli distribution.

Example:

```
stats::dbern(1,0.6,false);
```

Parameters

- **x** – an integral-valued input, equal to 0 or 1.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **dbern**(const std::vector<*eT*> &x, const *T1* prob_par, const bool log_form = false)

Density function of the Bernoulli distribution.

Example:

```
std::vector<int> x = {0, 1, 0};
stats::dbern(x,0.5,false);
```

Parameters

- **x** – a standard vector.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **mT**, typename **tT**, typename **T1**>
inline *mT* **dbern**(const ArmaGen<*mT*, *tT*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Bernoulli distribution.

Example:

```
arma::mat X = { {1, 0, 1},
                {0, 1, 0} };
stats::dbern(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dbern**(const BlazeMat<*eT*, *To*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Bernoulli distribution.

Example:

```
stats::dbern(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dbern**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Bernoulli distribution.

Example:

```
stats::dbern(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Bernoulli distribution:

$$F(x; p) = \sum_{z \leq x} f(z; p) = \begin{cases} 0 & \text{if } x < 0 \\ 1 - p & \text{if } 0 \leq x < 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T**>
constexpr return_t<*T*> **pbern**(const llint_t x, const *T* prob_par, const bool log_form = false) noexcept

Distribution function of the Bernoulli distribution.

Example:

```
stats::pbern(1,0.6,false);
```

Parameters

- **x** – a value equal to 0 or 1.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **pbern**(const std::vector<*eT*> &x, const *T1* prob_par, const bool log_form = false)

Density function of the Bernoulli distribution.

Example:

```
std::vector<int> x = {0, 1, 0};
stats::pbern(x,0.5,false);
```

Parameters

- **x** – a standard vector.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **pbern**(const ArmaMat<*eT*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Bernoulli distribution.

Example:

```
arma::mat X = { {1, 0, 1},
                {0, 1, 0} };
stats::pbern(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pbern**(const BlazeMat<*eT*, *To*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Bernoulli distribution.

Example:

```
stats::pbern(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pbern**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Bernoulli distribution.

Example:

```
stats::pbern(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Bernoulli distribution:

$$q(r;p) = \begin{cases} 0 & \text{if } r \leq 1-p \\ 1 & \text{else} \end{cases}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **qbern**(const *T1* p, const *T2* prob_par) noexcept

Quantile function of the Bernoulli distribution.

Example:

```
stats::qbern(0.5,0.4);
```

Parameters

- **p** – a real-valued input.
- **prob_par** – the probability parameter, a real-valued input.

Returns

the quantile function evaluated at `p`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **qbern**(const std::vector<*eT*> &x, const *T1* prob_par)

Quantile function of the Bernoulli distribution.

Example:

```
std::vector<int> x = {0.4, 0.5, 0.9};
stats::qbern(x,0.5);
```

Parameters

- **x** – a standard vector.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **qbern**(const ArmaMat<*eT*> &X, const *T1* prob_par)

Quantile function of the Bernoulli distribution.

Example:

```
arma::mat X = { {0.4, 0.5, 0.9},
                {0.3, 0.6, 0.7} };
stats::qbern(X,0.5);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qbern**(const BlazeMat<*eT*, *To*> &X, const *T1* prob_par)

Quantile function of the Bernoulli distribution.

Example:

```
stats::qbern(X,0.5);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qbern**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* prob_par)

Quantile function of the Bernoulli distribution.

Example:

```
stats::qbern(X,0.5);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Bernoulli distribution is achieved via the inverse probability integral transform.

Scalar Output

1. Random number engines

template<typename **T**>
inline return_t<*T*> **rbern**(const *T* prob_par, rand_engine_t &engine)

Random sampling function for the Bernoulli distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rbern(0.7,engine);
```

Parameters

- **prob_par** – the probability parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Bernoulli distribution.

2. Seed values

template<typename **T**>
inline return_t<*T*> **rbern**(const *T* prob_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Bernoulli distribution.

Example:

```
stats::rbern(0.7,1776);
```

Parameters

- **prob_par** – the probability parameter, a real-valued input.

- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns
a pseudo-random draw from the Bernoulli distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**>
inline *mT* **rbern**(const ullint_t n, const ullint_t k, const *T1* prob_par, rand_engine_t &engine)

Random matrix sampling function for the Bernoulli distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rbern<std::vector<double>>(5,4,0.7,engine);
// Armadillo matrix
stats::rbern<arma::mat>(5,4,0.7,engine);
// Blaze dynamic matrix
stats::rbern<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,0.7,engine);
// Eigen dynamic matrix
stats::rbern<Eigen::MatrixXd>(5,4,0.7,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **prob_par** – the probability parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns
a matrix of pseudo-random draws from the Bernoulli distribution.

2. Seed values

template<typename **mT**, typename **T1**>
inline *mT* **rbern**(const ullint_t n, const ullint_t k, const *T1* prob_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Bernoulli distribution.

Example:

```
// std::vector
stats::rbern<std::vector<double>>(5,4,0.7);
```

(continues on next page)

(continued from previous page)

```
// Armadillo matrix
stats::rbern<arma::mat>(5,4,0.7);
// Blaze dynamic matrix
stats::rbern<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,0.7);
// Eigen dynamic matrix
stats::rbern<Eigen::MatrixXd>(5,4,0.7);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **prob_par** – the probability parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Bernoulli distribution.

dbern	density function of the Bernoulli distribution
pbern	distribution function of the Bernoulli distribution
qbern	quantile function of the Bernoulli distribution
rbern	random sampling function of the Bernoulli distribution

2.6.2 Beta Distribution

Table of contents

Density Function

The density function of the Beta distribution:

$$f(x; a, b) = \frac{1}{\mathcal{B}(a, b)} x^{a-1}(1-x)^{b-1} \times \mathbf{1}[0 \leq x \leq 1]$$

where $\mathcal{B}(a, b)$ denotes the Beta function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dbeta**(const *T1* x, const *T2* a_par, const *T3* b_par, const bool log_form = false) noexcept

Density function of the Beta distribution.

Example:

```
stats::dbeta(0.5,3.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dbeta**(const std::vector<*eT*> &x, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Density function of the Beta distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::dbeta(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dbeta**(const ArmaMat<*eT*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Density function of the Beta distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9,  0.3,  0.87} };
stats::dbeta(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dbeta**(const BlazeMat<*eT*, *To*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Density function of the Beta distribution.

Example:

```
stats::dbeta(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dbeta**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Density function of the Beta distribution.

Example:

```
stats::dbeta(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Beta distribution:

$$F(x; a, b) = \int_0^x f(z; a, b) dz = I_x(a, b)$$

where $I_x(a, b)$ denotes the regularized incomplete Beta function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **pbeta**(const *T1* x, const *T2* a_par, const *T3* b_par, const bool log_form = false) noexcept

Distribution function of the Beta distribution.

Example:

```
stats::pbeta(0.5,3.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **pbeta**(const std::vector<*eT*> &x, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Distribution function of the Beta distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::pbeta(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **a_par** – a real-valued shape parameter.

- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **pbeta**(const ArmaMat<*eT*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Distribution function of the Beta distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9, -0.3,  1.3} };
stats::pbeta(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pbeta**(const BlazeMat<*eT*, *To*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Distribution function of the Beta distribution.

Example:

```
stats::pbeta(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pbeta**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Distribution function of the Beta distribution.

Example:

```
stats::pbeta(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Quantile Function

The quantile function of the Beta distribution:

$$q(p; a, b) = \inf \{x : p \leq I_x(a, b)\}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qbeta**(const *T1* p, const *T2* a_par, const *T3* b_par) noexcept

Quantile function of the Beta distribution.

Example:

```
stats::qbeta(0.5,3.0,2.0);
```

Parameters

- **p** – a real-valued input.

- **a_par** – shape parameter, a real-valued input.
- **b_par** – shape parameter, a real-valued input.

Returns
the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qbeta**(const std::vector<*eT*> &x, const *T1* a_par, const *T2* b_par)

Quantile function of the Beta distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::qbeta(x,3.0,2.0);
```

Parameters

- **x** – a standard vector.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.

Returns
a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qbeta**(const ArmaMat<*eT*> &X, const *T1* a_par, const *T2* b_par)

Quantile function of the Beta distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9,  0.3,  0.87} };
stats::qbeta(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.

Returns
a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qbeta**(const BlazeMat<*eT*, *To*> &X, const *T1* a_par, const *T2* b_par)

Quantile function of the Beta distribution.

Example:

```
stats::qbeta(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qbeta**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* a_par, const *T2* b_par)

Quantile function of the Beta distribution.

Example:

```
stats::qbeta(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Beta distribution is achieved by simulating two independent gamma-distributed random variables, $X \sim G(a, 1), Y \sim G(a, 1)$, then returning:

$$Z = \frac{X}{X + Y} \sim B(a, b)$$

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rbeta**(const *T1* a_par, const *T2* b_par, rand_engine_t &engine)

Random sampling function for the Beta distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rbeta(3.0,2.0,engine);
```

Parameters

- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Beta distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rbeta**(const *T1* a_par, const *T2* b_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Beta distribution.

Example:

```
stats::rbeta(3.0,2.0,1776);
```

Parameters

- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Beta distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rbeta**(const ullint_t n, const ullint_t k, const *T1* a_par, const *T2* b_par, rand_engine_t &engine)

Random matrix sampling function for the Beta distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rbeta<std::vector<double>>(5,4,3.0,2.0,engine);
// Armadillo matrix
stats::rbeta<arma::mat>(5,4,3.0,2.0,engine);
// Blaze dynamic matrix
stats::rbeta<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0,engine);
// Eigen dynamic matrix
stats::rbeta<Eigen::MatrixXd>(5,4,3.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **a_par** – a real-valued shape parameter.
- **b_par** – a real-valued shape parameter.
- **engine** – a random engine, passed by reference.

Returns
a matrix of pseudo-random draws from the Beta distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rbeta**(const ullint_t n, const ullint_t k, const *T1* a_par, const *T2* b_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Beta distribution.

Example:

```
// std::vector
stats::rbeta<std::vector<double>>(5,4,3.0,2.0);
// Armadillo matrix
stats::rbeta<arma::mat>(5,4,3.0,2.0);
// Blaze dynamic matrix
stats::rbeta<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0);
```

(continues on next page)

(continued from previous page)

```
// Eigen dynamic matrix
stats::rbeta<Eigen::MatrixXd>(5,4,3.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **`n`** – the number of output rows
- **`k`** – the number of output columns
- **`a_par`** – a real-valued shape parameter.
- **`b_par`** – a real-valued shape parameter.
- **`seed_val`** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Beta distribution.

dbeta	density function of the Beta distribution
pbeta	distribution function of the Beta distribution
qbeta	quantile function of the Beta distribution
rbeta	random sampling function of the Beta distribution

2.6.3 Binomial Distribution

Table of contents

Density Function

The density function of the Binomial distribution:

$$f(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x} \times \mathbf{1}[x \in \{0, \dots, n\}]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T**>
constexpr return_t<*T*> **dbinom**(const llint_t x, const llint_t n_trials_par, const *T* prob_par, const bool log_form = false) noexcept

Density function of the Binomial distribution.

Example:

```
stats::dbinom(2,4,0.4,false);
```

Parameters

- **x** – a real-valued input.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **dbinom**(const std::vector<*eT*> &x, const llint_t n_trials_par, const *T1* prob_par, const bool log_form = false)

Density function of the Binomial distribution.

Example:

```
std::vector<int> x = {2, 3, 4};
stats::dbinom(x,5,0.5,false);
```

Parameters

- **x** – a standard vector.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **dbinom**(const ArmaMat<*eT*> &X, const llint_t n_trials_par, const *T1* prob_par, const bool log_form = false)

Density function of the Binomial distribution.

Example:

```
stats::dbinom(X,5,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dbinom**(const BlazeMat<*eT*, *To*> &X, const llint_t n_trials_par, const *T1* prob_par, const bool log_form = false)

Density function of the Binomial distribution.

Example:

```
stats::dbinom(X,5,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dbinom**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const llint_t n_trials_par, const *T1* prob_par, const bool log_form = false)

Density function of the Binomial distribution.

Example:

```
stats::dbinom(X,5,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Binomial distribution:

$$F(x; n, p) = \sum_{z \leq x} f(z; n, p)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T**>
constexpr *T* **pbinom**(const llint_t x, const llint_t n_trials_par, const *T* prob_par, const bool log_form = false) noexcept

Distribution function of the Binomial distribution.

Example:

```
stats::pbinom(2,4,0.4,false);
```

Parameters

- **x** – a real-valued input.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **pbinom**(const std::vector<*eT*> &x, const llint_t n_trials_par, const *T1* prob_par, const bool log_form = false)

Distribution function of the Binomial distribution.

Example:

```
std::vector<int> x = {2, 3, 4};
stats::pbinom(x,5,0.5,false);
```

Parameters

- **x** – a standard vector.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.

- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **pbinom**(const ArmaMat<*eT*> &X, const llint_t n_trials_par, const *T1* prob_par, const bool log_form = false)

Distribution function of the Binomial distribution.

Example:

```
stats::pbinom(X,5,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pbinom**(const BlazeMat<*eT*, *To*> &X, const llint_t n_trials_par, const *T1* prob_par, const bool log_form = false)

Distribution function of the Binomial distribution.

Example:

```
stats::pbinom(X,5,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pbinom**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const llint_t n_trials_par, const *T1* prob_par, const bool log_form = false)

Distribution function of the Binomial distribution.

Example:

```
stats::pbinom(X,5,0.5,false);
```

Parameters

- `X` – a matrix of input values.
- `n_trials_par` – the number of trials, a non-negative integral-valued input.
- `prob_par` – the probability parameter, a real-valued input.
- `log_form` – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of `X`.

Quantile Function

The quantile function of the Binomial distribution:

$$q(r; n, p) = \inf \{x : r \leq F(x; n, p)\}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **qbinom**(const *T1* p, const llint_t n_trials_par, const *T2* prob_par) noexcept

Quantile function of the Binomial distribution.

Example:

```
stats::qbinom(0.4,4,0.4);
```

Parameters

- `p` – a real-valued input.
- `n_trials_par` – the number of trials, a non-negative integral-valued input.
- `prob_par` – the probability parameter, a real-valued input.

Returns

the quantile function evaluated at `p`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **qbinom**(const std::vector<*eT*> &x, const llint_t n_trials_par, const *T1* prob_par)

Quantile function of the Binomial distribution.

Example:

```
std::vector<int> x = {2, 3, 4};
stats::qbinom(x,5,0.5);
```

Parameters

- **x** – a standard vector.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **qbinom**(const ArmaMat<*eT*> &X, const llint_t n_trials_par, const *T1* prob_par)

Quantile function of the Binomial distribution.

Example:

```
stats::qbinom(X,5,0.5);
```

Parameters

- **X** – a matrix of input values.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qbinom**(const BlazeMat<*eT*, *To*> &X, const llint_t n_trials_par, const *T1* prob_par)

Quantile function of the Binomial distribution.

Example:

```
stats::qbinom(X,5,0.5);
```

Parameters

- **X** – a matrix of input values.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qbinom**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const llint_t n_trials_par, const *T1* prob_par)

Quantile function of the Binomial distribution.

Example:

```
stats::qbinom(X,5,0.5);
```

Parameters

- **X** – a matrix of input values.
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Binomial distribution is achieved by summing the results of simulating n Bernoulli-distributed random variables.

Scalar Output

1. Random number engines

template<typename **T**>
inline return_t<*T*> **rbinom**(const llint_t n_trials_par, const *T* prob_par, rand_engine_t &engine)

Random sampling function for the Binomial distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rbinom(4,0.4,engine);
```

Parameters

- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Beta distribution.

2. Seed values

template<typename **T**>
inline return_t<*T*> **rbinom**(const llint_t n_trials_par, const *T* prob_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Binomial distribution.

Example:

```
stats::rbinom(4,0.4,1776);
```

Parameters

- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Beta distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**>
inline *mT* **rbinom**(const ullint_t n, const ullint_t k, const llint_t n_trials_par, const *T1* prob_par, rand_engine_t &engine)

Random matrix sampling function for the Binomial distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rbinom<std::vector<double>>(5,4,5,0.7,engine);
// Armadillo matrix
stats::rbinom<arma::mat>(5,4,5,0.7,engine);
// Blaze dynamic matrix
stats::rbinom<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,5,0.7,engine);
// Eigen dynamic matrix
stats::rbinom<Eigen::MatrixXd>(5,4,5,0.7,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Binomial distribution.

2. Seed values

template<typename **mT**, typename **T1**>
inline *mT* **rbinom**(const ullint_t n, const ullint_t k, const llint_t n_trials_par, const *T1* prob_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Binomial distribution.

Example:

```
// std::vector
stats::rbinom<std::vector<double>>(5,4,5,0.7);
// Armadillo matrix
stats::rbinom<arma::mat>(5,4,5,0.7);
// Blaze dynamic matrix
```

(continues on next page)

(continued from previous page)

```
stats::rbinom<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,5,0.7);
// Eigen dynamic matrix
stats::rbinom<Eigen::MatrixXd>(5,4,5,0.7);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **n_trials_par** – the number of trials, a non-negative integral-valued input.
- **prob_par** – the probability parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns
a matrix of pseudo-random draws from the Binomial distribution.

dbinom	density function of the Binomial distribution
pbinom	distribution function of the Binomial distribution
qbinom	quantile function of the Binomial distribution
rbinom	random sampling function of the Binomial distribution

2.6.4 Cauchy Distribution

Table of contents

Density Function

The density function of the Cauchy distribution:

$$f(x;\mu,\sigma)=\frac{1}{\pi\sigma\left[1+\left(\frac{x-\mu}{\sigma}\right)^2\right]}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dcauchy**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Density function of the Cauchy distribution.

Example:

```
stats::dcauchy(2.5,1.0,3.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dcauchy**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Cauchy distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::dcauchy(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dcauchy**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Cauchy distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::dcauchy(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dcauchy**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Cauchy distribution.

Example:

```
stats::dcauchy(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dcauchy**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Cauchy distribution.

Example:

```
stats::dcauchy(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Cauchy distribution:

$$F(x; \mu, \sigma) = \int_{-\infty}^{x} f(z; \mu, \sigma) dz = 0.5 + \frac{1}{\pi} \text{arctan} \left(\frac{x - \mu}{\sigma} \right)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **pcauchy**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Distribution function of the Cauchy distribution.

Example:

```
stats::pcauchy(2.5,1.0,3.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **pcauchy**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Cauchy distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::pcauchy(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.

- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **pcauchy**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Cauchy distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::pcauchy(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pcauchy**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Cauchy distribution.

Example:

```
stats::pcauchy(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pcauchy**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Cauchy distribution.

Example:

```
stats::pcauchy(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Quantile Function

The quantile function of the Cauchy distribution:

$$q(p; \mu, \sigma) = \mu + \gamma \tan\left(\pi(p - 0.5)\right)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qcauchy**(const *T1* p, const *T2* mu_par, const *T3* sigma_par) noexcept

Quantile function of the Cauchy distribution.

Example:

```
stats::qcauchy(0.5,1.0,3.0);
```

Parameters

- **p** – a real-valued input.

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns
the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qcauchy**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Cauchy distribution.

Example:

```
std::vector<double> x = {0.1, 0.3, 0.7};
stats::qcauchy(x,1.0,2.0);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns
a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qcauchy**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Cauchy distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qcauchy(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qcauchy**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Cauchy distribution.

Example:

```
stats::qcauchy(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qcauchy**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Cauchy distribution.

Example:

```
stats::qcauchy(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Cauchy distribution is achieved via the inverse probability integral transform.

Scalar Output

Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rcauchy**(const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random sampling function for the Cauchy distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rcauchy(1.0,2.0,engine);
```

Parameters

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Cauchy distribution.

Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rcauchy**(const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Cauchy distribution.

Example:

```
stats::rcauchy(1.0,2.0,1776);
```

Parameters

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Cauchy distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rcauchy**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random matrix sampling function for the Cauchy distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rcauchy<std::vector<double>>(5,4,1.0,2.0,engine);
// Armadillo matrix
stats::rcauchy<arma::mat>(5,4,1.0,2.0,engine);
// Blaze dynamic matrix
stats::rcauchy<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0,engine);
// Eigen dynamic matrix
stats::rcauchy<Eigen::MatrixXd>(5,4,1.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element types `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Cauchy distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rcauchy**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Cauchy distribution.

Example:

```
// std::vector
stats::rcauchy<std::vector<double>>(5,4,1.0,2.0);
// Armadillo matrix
stats::rcauchy<arma::mat>(5,4,1.0,2.0);
// Blaze dynamic matrix
stats::rcauchy<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0);
```

(continues on next page)

(continued from previous page)

```
// Eigen dynamic matrix
stats::rcauchy<Eigen::MatrixXd>(5,4,1.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element types `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Cauchy distribution.

dcauchy	density function of the Cauchy distribution
pcauchy	distribution function of the Cauchy distribution
qcauchy	quantile function of the Cauchy distribution
rcauchy	random sampling function of the Cauchy distribution

2.6.5 Chi-squared Distribution

Table of contents

Density Function

The density function of the Chi-squared distribution:

$$f(x;k) = \frac{x^{k/2-1}\exp(-x/2)}{2^{k/2}\Gamma(k/2)} \times \mathbf{1}[x \geq 0]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **dchisq**(const *T1* x, const *T2* dof_par, const bool log_form = false) noexcept

Density function of the Chi-squared distribution.

Example:

```
stats::dchisq(4,5,false);
```

Parameters

- **x** – a real-valued input.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **dchisq**(const std::vector<*eT*> &x, const *T1* dof_par, const bool log_form = false)

Density function of the Chi-squared distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::dchisq(x,4,false);
```

Parameters

- **x** – a standard vector.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **dchisq**(const ArmaMat<*eT*> &X, const *T1* dof_par, const bool log_form = false)

Density function of the Chi-squared distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::dchisq(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dchisq**(const BlazeMat<*eT*, *To*> &X, const *T1* dof_par, const bool log_form = false)

Density function of the Chi-squared distribution.

Example:

```
stats::dchisq(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dchisq**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* dof_par, const bool log_form = false)

Density function of the Chi-squared distribution.

Example:

```
stats::dchisq(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Chi-squared distribution:

$$F(x;k) = \int_0^x f(z;k) dz = \frac{\gamma(k/2, x/2)}{\Gamma(k/2)}$$

where $\Gamma(\cdot)$ denotes the gamma function and $\gamma(\cdot, \cdot)$ denotes the incomplete gamma function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **pchisq**(const *T1* x, const *T2* dof_par, const bool log_form = false) noexcept

Distribution function of the Chi-squared distribution.

Example:

```
stats::pchisq(4,5,false);
```

Parameters

- **x** – a real-valued input.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **pchisq**(const std::vector<*eT*> &x, const *T1* dof_par, const bool log_form = false)

Distribution function of the Chi-squared distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::pchisq(x,4,false);
```

Parameters

- **x** – a standard vector.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **pchisq**(const ArmaMat<*eT*> &X, const *T1* dof_par, const bool log_form = false)

Distribution function of the Chi-squared distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::pchisq(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pchisq**(const BlazeMat<*eT*, *To*> &X, const *T1* dof_par, const bool log_form = false)

Distribution function of the Chi-squared distribution.

Example:

```
stats::pchisq(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pchisq**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* dof_par, const bool log_form = false)

Distribution function of the Chi-squared distribution.

Example:

```
stats::pchisq(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Chi-squared distribution:

$$q(p; k) = \inf \{x : p \leq \gamma(k/2, x/2)/\Gamma(k/2)\}$$

where $\Gamma(\cdot)$ denotes the gamma function and $\gamma(\cdot, \cdot)$ denotes the incomplete gamma function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **qchisq**(const *T1* p, const *T2* dof_par) noexcept

Quantile function of the Chi-squared distribution.

Example:

```
stats::qchisq(0.5,5);
```

Parameters

- **p** – a real-valued input.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns

the quantile function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **qchisq**(const std::vector<*eT*> &x, const *T1* dof_par)

Quantile function of the Chi-squared distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.8};
stats::qchisq(x,4);
```

Parameters

- **x** – a standard vector.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **qchisq**(const ArmaMat<*eT*> &X, const *T1* dof_par)

Quantile function of the Chi-squared distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qchisq(X,4);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qchisq**(const BlazeMat<*eT*, *To*> &X, const *T1* dof_par)

Quantile function of the Chi-squared distribution.

Example:

```
stats::qchisq(X,4);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qchisq**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* dof_par)

Quantile function of the Chi-squared distribution.

Example:

```
stats::qchisq(X,4);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Scalar Output

1. Random number engines

template<typename **T**>

inline return_t<*T*> **rchisq**(const *T* dof_par, rand_engine_t &engine)

Random sampling function for the Chi-squared distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rchisq(4,engine);
```

Parameters

- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Chi-squared distribution.

2. Seed values

template<typename **T**>
inline return_t<*T*> **rchisq**(const *T* dof_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Chi-squared distribution.

Example:

```
stats::rchisq(4,1776);
```

Parameters

- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Chi-squared distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**>
inline *mT* **rchisq**(const ullint_t n, const ullint_t k, const *T1* dof_par, rand_engine_t &engine)

Random matrix sampling function for the Chi-squared distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rchisq<std::vector<double>>(5,4,4,engine);
// Armadillo matrix
stats::rchisq<arma::mat>(5,4,4,engine);
// Blaze dynamic matrix
```

(continues on next page)

(continued from previous page)

```
stats::rchisq<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,4,engine);
// Eigen dynamic matrix
stats::rchisq<Eigen::MatrixXd>(5,4,4,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Chi-squared distribution.

2. Seed values

template<typename **mT**, typename **T1**>
inline *mT* **rchisq**(const ullint_t n, const ullint_t k, const *T1* dof_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Chi-squared distribution.

Example:

```
// std::vector
stats::rchisq<std::vector<double>>(5,4,4);
// Armadillo matrix
stats::rchisq<arma::mat>(5,4,4);
// Blaze dynamic matrix
stats::rchisq<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,4);
// Eigen dynamic matrix
stats::rchisq<Eigen::MatrixXd>(5,4,4);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Chi-squared distribution.

dchisq	density function of the Chi-squared distribution
pchisq	distribution function of the Chi-squared distribution
qchisq	quantile function of the Chi-squared distribution
rchisq	random sampling function of the Chi-squared distribution

2.6.6 Exponential Distribution

Table of contents

Density Function

The density function of the Exponential distribution:

$$f(x; \lambda) = \lambda \exp(-\lambda x) \times \mathbf{1}[x \geq 0]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **dexp**(const *T1* x, const *T2* rate_par, const bool log_form = false) noexcept

Density function of the Exponential distribution.

Example:

```
stats::dexp(1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **dexp**(const std::vector<*eT*> &x, const *T1* rate_par, const bool log_form = false)

Density function of the Exponential distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::dexp(x,4,false);
```

Parameters

- **x** – a standard vector.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **dexp**(const ArmaMat<*eT*> &X, const *T1* rate_par, const bool log_form = false)

Density function of the Exponential distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::dexp(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dexp**(const BlazeMat<*eT*, *To*> &X, const *T1* rate_par, const bool log_form = false)

Density function of the Exponential distribution.

Example:

```
stats::dexp(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dexp**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* rate_par, const bool log_form = false)

Density function of the Exponential distribution.

Example:

```
stats::dexp(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Exponential distribution:

$$\int_0^x f(z; \lambda) dz = 1 - \exp(-\lambda x \times \mathbf{1}[x \geq 0])$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **pexp**(const *T1* x, const *T2* rate_par, const bool log_form = false) noexcept

Distribution function of the Exponential distribution.

Example:

```
stats::pexp(1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **pexp**(const std::vector<*eT*> &x, const *T1* rate_par, const bool log_form = false)

Distribution function of the Exponential distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::pexp(x,2.0,false);
```

Parameters

- **x** – a standard vector.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **pexp**(const ArmaMat<*eT*> &X, const *T1* rate_par, const bool log_form = false)

Distribution function of the Exponential distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::pexp(X,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pexp**(const BlazeMat<*eT*, *To*> &X, const *T1* rate_par, const bool log_form = false)

Distribution function of the Exponential distribution.

Example:

```
stats::pexp(X,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pexp**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* rate_par, const bool log_form = false)

Distribution function of the Exponential distribution.

Example:

```
stats::pexp(X,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Exponential distribution:

$$q(p; \lambda) = -\ln(1-p)/\lambda$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **qexp**(const *T1* p, const *T2* rate_par) noexcept

Quantile function of the Exponential distribution.

Example:

```
stats::qexp(0.5,4.0);
```

Parameters

- **p** – a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.

Returns

the quantile function evaluated at `p`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **qexp**(const std::vector<*eT*> &x, const *T1* rate_par)

Quantile function of the Exponential distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.8};
stats::qexp(x,4);
```

Parameters

- **x** – a standard vector.
- **rate_par** – the rate parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **qexp**(const ArmaMat<*eT*> &X, const *T1* rate_par)

Quantile function of the Exponential distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qexp(X,4);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qexp**(const BlazeMat<*eT*, *To*> &X, const *T1* rate_par)

Quantile function of the Exponential distribution.

Example:

```
stats::qexp(X,4);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qexp**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* rate_par)

Quantile function of the Exponential distribution.

Example:

```
stats::qexp(X,4);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Cauchy distribution is achieved via the inverse probability integral transform.

Scalar Output

1. Random number engines

template<typename **T**>
inline return_t<*T*> **rexp**(const *T* rate_par, rand_engine_t &engine)

Random sampling function for the Exponential distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rexp(4,engine);
```

Parameters

- **rate_par** – the rate parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Exponential distribution.

2. Seed values

template<typename **T**>
inline return_t<*T*> **rexp**(const *T* rate_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Exponential distribution.

Example:

```
stats::rexp(4,1776);
```

Parameters

- **rate_par** – the rate parameter, a real-valued input.

- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns
a pseudo-random draw from the Exponential distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**>
inline *mT* **rexp**(const ullint_t n, const ullint_t k, const *T1* rate_par, rand_engine_t &engine)

Random matrix sampling function for the Exponential distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rexp<std::vector<double>>(5,4,4,engine);
// Armadillo matrix
stats::rexp<arma::mat>(5,4,4,engine);
// Blaze dynamic matrix
stats::rexp<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,4,engine);
// Eigen dynamic matrix
stats::rexp<Eigen::MatrixXd>(5,4,4,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **rate_par** – the rate parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns
a matrix of pseudo-random draws from the Exponential distribution.

2. Seed values

template<typename **mT**, typename **T1**>
inline *mT* **rexp**(const ullint_t n, const ullint_t k, const *T1* rate_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Exponential distribution.

Example:

```
// std::vector
stats::rexp<std::vector<double>>(5,4,4,engine);
// Armadillo matrix
stats::rexp<arma::mat>(5,4,4,engine);
```

(continues on next page)

(continued from previous page)

```
// Blaze dynamic matrix
stats::rexp<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,4,engine);
// Eigen dynamic matrix
stats::rexp<Eigen::MatrixXd>(5,4,4,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **rate_par** – the rate parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Exponential distribution.

dexp	density function of the Exponential distribution
pexp	distribution function of the Exponential distribution
qexp	quantile function of the Exponential distribution
rexp	random sampling function of the Exponential distribution

2.6.7 F-Distribution

Table of contents

Density Function

The density function of the F distribution:

$$f(x; d_1, d_2) = \frac{1}{\mathcal{B}\left(\frac{d_1}{2}, \frac{d_2}{2}\right)} \left(\frac{d_1}{d_2}\right)^{\frac{d_1}{2}} x^{d_1/2-1} \left(1 + \frac{d_1}{d_2} x\right)^{-\frac{d_1+d_2}{2}} \times \mathbf{1}[x \geq 0]$$

where $\mathcal{B}(a, b)$ denotes the Beta function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **df**(const *T1* x, const *T2* df1_par, const *T3* df2_par, const bool log_form = false) noexcept

Density function of the F-distribution.

Example:

```
stats::df(1.5,10.0,12.0,false);
```

Parameters

- **x** – a real-valued input.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **df**(const std::vector<*eT*> &x, const *T1* df1_par, const *T2* df2_par, const bool log_form = false)

Density function of the F-distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::df(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **df**(const ArmaMat<*eT*> &X, const *T1* df1_par, const *T2* df2_par, const bool log_form = false)

Density function of the F-distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9, -0.3,  1.3} };
stats::df(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **df**(const BlazeMat<*eT*, *To*> &X, const *T1* df1_par, const *T2* df2_par, const bool log_form = false)

Density function of the F-distribution.

Example:

```
stats::df(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **df**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* df1_par, const *T2* df2_par, const bool log_form = false)

Density function of the F-distribution.

Example:

```
stats::df(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the F distribution:

$$F(x; d_1, d_2) = \int_0^x f(z; d_1, d_2) dz = I_{\frac{d_1 x}{d_2 + d_1 x}}(d_1/2, d_2/2)$$

where $I_x(a, b)$ denotes the regularized incomplete Beta function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **pf**(const *T1* x, const *T2* df1_par, const *T3* df2_par, const bool log_form = false) noexcept

Distribution function of the F-distribution.

Example:

```
stats::pf(1.5,10.0,12.0,false);
```

Parameters

- **x** – a real-valued input.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **pf**(const std::vector<*eT*> &x, const *T1* df1_par, const *T2* df2_par, const bool log_form = false)

Distribution function of the Beta distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::pf(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.

- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **pf**(const ArmaMat<*eT*> &X, const *T1* df1_par, const *T2* df2_par, const bool log_form = false)

Distribution function of the Beta distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9, -0.3,  1.3} };
stats::pf(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pf**(const BlazeMat<*eT*, *To*> &X, const *T1* df1_par, const *T2* df2_par, const bool log_form = false)

Distribution function of the Beta distribution.

Example:

```
stats::pf(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pf**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* df1_par, const *T2* df2_par, const bool log_form = false)

Distribution function of the Beta distribution.

Example:

```
stats::pf(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Quantile Function

The quantile function of the F distribution:

$$q(p; a, b) = \inf \left\{ x : p \leq I_{\frac{d_1 x}{d_2 + d_1 x}}(d_1/2, d_2/2) \right\}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qf**(const *T1* p, const *T2* df1_par, const *T3* df2_par) noexcept

Quantile function of the F-distribution.

Example:

```
stats::qf(0.5,10.0,12.0);
```

Parameters

- **p** – a real-valued input.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.

Returns

the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qf**(const std::vector<*eT*> &x, const *T1* df1_par, const *T2* df2_par)

Quantile function of the F-distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::qf(x,3.0,2.0);
```

Parameters

- **x** – a standard vector.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qf**(const ArmaMat<*eT*> &X, const *T1* df1_par, const *T2* df2_par)

Quantile function of the F-distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9,  0.3,  0.87} };
stats::qf(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qf**(const BlazeMat<*eT*, *To*> &X, const *T1* df1_par, const *T2* df2_par)

Quantile function of the F-distribution.

Example:

```
stats::qf(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qf**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* df1_par, const *T2* df2_par)

Quantile function of the F-distribution.

Example:

```
stats::qf(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Beta distribution is achieved by simulating two independent χ^2-distributed random variables, $X \sim \chi^2(d_1), Y \sim \chi^2(d_2)$, then returning:

$$Z = \frac{d_1}{d_2} \frac{X}{Y}$$

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rf**(const *T1* df1_par, const *T2* df2_par, rand_engine_t &engine)

Random sampling function for the F-distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rf(3.0,2.0,engine);
```

Parameters

- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the F-distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rf**(const *T1* df1_par, const *T2* df2_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the F-distribution.

Example:

```
stats::rf(3.0,2.0,1776);
```

Parameters

- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the F-distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rf**(const ullint_t n, const ullint_t k, const *T1* df1_par, const *T2* df2_par, rand_engine_t &engine)

Random matrix sampling function for the F-distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rf<std::vector<double>>(5,4,3.0,2.0,engine);
// Armadillo matrix
stats::rf<arma::mat>(5,4,3.0,2.0,engine);
// Blaze dynamic matrix
stats::rf<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0,engine);
// Eigen dynamic matrix
stats::rf<Eigen::MatrixXd>(5,4,3.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the F-distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rf**(const ullint_t n, const ullint_t k, const *T1* df1_par, const *T2* df2_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the F-distribution.

Example:

```
// std::vector
stats::rf<std::vector<double>>(5,4,3.0,2.0);
// Armadillo matrix
stats::rf<arma::mat>(5,4,3.0,2.0);
// Blaze dynamic matrix
stats::rf<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0);
```

(continues on next page)

(continued from previous page)

```
// Eigen dynamic matrix
stats::rf<Eigen::MatrixXd>(5,4,3.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **df1_par** – a degrees of freedom parameter, a real-valued input.
- **df2_par** – a degrees of freedom parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the F-distribution.

df	density function of the F-distribution
pf	distribution function of the F-distribution
qf	quantile function of the F-distribution
rf	random sampling function of the F-distribution

2.6.8 Gamma Distribution

Table of contents

Density Function

The density function of the Gamma distribution:

$$f(x;k,\theta) = \frac{x^{k-1}\exp(-x/\theta)}{\theta^k \Gamma(k)} \times \mathbf{1}[x \geq 0]$$

where $\Gamma(\cdot)$ denotes the Gamma function, k is the shape parameter, and θ is the scale parameter.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dgamma**(const *T1* x, const *T2* shape_par, const *T3* scale_par, const bool log_form = false) noexcept

Density function of the Gamma distribution.

Example:

```
stats::dgamma(2,2,3,false);
```

Parameters

- **x** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dgamma**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Density function of the Gamma distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::dgamma(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dgamma**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Density function of the Gamma distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::dgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dgamma**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Density function of the Gamma distribution.

Example:

```
stats::dgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dgamma**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Density function of the Gamma distribution.

Example:

```
stats::dgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Gamma distribution:

$$F(x; k, \theta) = \int_0^x f(z; k, \theta) dz = \frac{\gamma(k, x\theta)}{\Gamma(k)}$$

where $\Gamma(\cdot)$ denotes the gamma function and $\gamma(\cdot, \cdot)$ denotes the incomplete gamma function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **pgamma**(const *T1* x, const *T2* shape_par, const *T3* scale_par, const bool log_form = false) noexcept

Distribution function of the Gamma distribution.

Example:

```
stats::pgamma(2,2,3,false);
```

Parameters

- **x** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **pgamma**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Distribution function of the Gamma distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::pgamma(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.

- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **pgamma**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Distribution function of the Gamma distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::pgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pgamma**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Distribution function of the Gamma distribution.

Example:

```
stats::pgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pgamma**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Distribution function of the Gamma distribution.

Example:

```
stats::pgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Gamma distribution:

$$q(p; k, \theta) = \inf \left\{ x : p \leq \frac{\gamma(k, x\theta)}{\Gamma(k)} \right\}$$

where $\Gamma(\cdot)$ denotes the gamma function and $\gamma(\cdot, \cdot)$ denotes the incomplete gamma function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qgamma**(const *T1* p, const *T2* shape_par, const *T3* scale_par) noexcept

Quantile function of the Gamma distribution.

Example:

```
stats::qgamma(0.4,2,3);
```

Parameters

- **p** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns
the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qgamma**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* scale_par)

Quantile function of the Gamma distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::qgamma(x,3.0,2.0);
```

Parameters

- **x** – a standard vector.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns
a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qgamma**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* scale_par)

Quantile function of the Gamma distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9,  0.3,  0.87} };
stats::qgamma(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.

- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qgamma**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* scale_par)

Quantile function of the Gamma distribution.

Example:

```
stats::qgamma(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qgamma**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* scale_par)

Quantile function of the Gamma distribution.

Example:

```
stats::qgamma(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Gamma distribution is achieved via the Ziggurat method of Marsaglia and Tsang (2000).

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rgamma**(const *T1* shape_par, const *T2* scale_par, rand_engine_t &engine)

Random sampling function for the Gamma distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rgamma(3.0,2.0,engine);
```

Parameters

- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Gamma distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rgamma**(const *T1* shape_par, const *T2* scale_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Gamma distribution.

Example:

```
stats::rgamma(3.0,2.0,1776);
```

Parameters

- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Gamma distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rgamma**(const ullint_t n, const ullint_t k, const *T1* shape_par, const *T2* scale_par, rand_engine_t &engine)

Random matrix sampling function for the Gamma distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rgamma<std::vector<double>>(5,4,3.0,2.0,engine);
// Armadillo matrix
stats::rgamma<arma::mat>(5,4,3.0,2.0,engine);
// Blaze dynamic matrix
stats::rgamma<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0,engine);
// Eigen dynamic matrix
stats::rgamma<Eigen::MatrixXd>(5,4,3.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Gamma distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rgamma**(const ullint_t n, const ullint_t k, const *T1* shape_par, const *T2* scale_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Gamma distribution.

Example:

```
// std::vector
stats::rgamma<std::vector<double>>(5,4,3.0,2.0);
// Armadillo matrix
stats::rgamma<arma::mat>(5,4,3.0,2.0);
// Blaze dynamic matrix
stats::rgamma<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0);
```

(continues on next page)

(continued from previous page)

```
// Eigen dynamic matrix
stats::rgamma<Eigen::MatrixXd>(5,4,3.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Gamma distribution.

dgamma	density function of the Gamma distribution
pgamma	distribution function of the Gamma distribution
qgamma	quantile function of the Gamma distribution
rgamma	random sampling function of the Gamma distribution

2.6.9 Inverse-Gamma Distribution

Table of contents

Density Function

The density function of the inverse-Gamma distribution:

$$f(x; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} \exp\left(-\frac{\beta}{x}\right) \times \mathbf{1}[x \geq 0]$$

where $\Gamma(\cdot)$ denotes the Gamma function, α is the shape parameter, and β is the rate parameter.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dinvgamma**(const *T1* x, const *T2* shape_par, const *T3* rate_par, const bool log_form = false) noexcept

Density function of the Inverse-Gamma distribution.

Example:

```
stats::dinvgamma(1.5,2,1,false);
```

Parameters

- **x** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dinvgamma**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* rate_par, const bool log_form = false)

Density function of the Inverse-Gamma distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::dinvgamma(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dinvgamma**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* rate_par, const bool log_form = false)

Density function of the Inverse-Gamma distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::dinvgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dinvgamma**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* rate_par, const bool log_form = false)

Density function of the Inverse-Gamma distribution.

Example:

```
stats::dinvgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dinvgamma**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* rate_par, const bool log_form = false)

Density function of the Inverse-Gamma distribution.

Example:

```
stats::dinvgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the inverse-Gamma distribution:

$$F(x; \alpha, \beta) = \int_0^x f(z; \alpha, \beta) dz = 1 - \frac{\gamma(1/x, \beta/x)}{\Gamma(\alpha)}$$

where $\Gamma(\cdot)$ denotes the gamma function and $\gamma(\cdot, \cdot)$ denotes the incomplete gamma function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **pinvgamma**(const *T1* x, const *T2* shape_par, const *T3* rate_par, const bool log_form = false) noexcept

Distribution function of the Inverse-Gamma distribution.

Example:

```
stats::pinvgamma(1.5,2,1,false);
```

Parameters

- **x** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **pinvgamma**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* rate_par, const bool log_form = false)

Distribution function of the Inverse-Gamma distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::pinvgamma(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.

- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **pinvgamma**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* rate_par, const bool log_form = false)

Distribution function of the Inverse-Gamma distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::pinvgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pinvgamma**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* rate_par, const bool log_form = false)

Distribution function of the Inverse-Gamma distribution.

Example:

```
stats::pinvgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.

- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pinvgamma**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* rate_par, const bool log_form = false)

Distribution function of the Inverse-Gamma distribution.

Example:

```
stats::pinvgamma(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the inverse-Gamma distribution:

$$q(p; \alpha, \beta) = \inf \left\{ x : p \leq 1 - \frac{\gamma(1/x, \beta/x)}{\Gamma(\alpha)} \right\}$$

where $\Gamma(\cdot)$ denotes the gamma function and $\gamma(\cdot, \cdot)$ denotes the incomplete gamma function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qinvgamma**(const *T1* p, const *T2* shape_par, const *T3* rate_par) noexcept

Quantile function of the Inverse-Gamma distribution.

Example:

```
stats::qinvgamma(0.5,2,1);
```

Parameters

- **p** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.

Returns

the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qinvgamma**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* rate_par)

Quantile function of the Inverse-Gamma distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::qinvgamma(x,3.0,2.0);
```

Parameters

- **x** – a standard vector.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qinvgamma**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* rate_par)

Quantile function of the Inverse-Gamma distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9,  0.3,  0.87} };
stats::qinvgamma(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.

- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qinvgamma**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* rate_par)

Quantile function of the Inverse-Gamma distribution.

Example:

```
stats::qinvgamma(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qinvgamma**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* rate_par)

Quantile function of the Inverse-Gamma distribution.

Example:

```
stats::qinvgamma(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the inverse-Gamma distribution is achieved by simulating $X \sim G(\alpha, 1/\beta)$, then returning

$$Z = \frac{1}{X} \sim \text{IG}(\alpha, \beta)$$

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rinvgamma**(const *T1* shape_par, const *T2* rate_par, rand_engine_t &engine)

Random sampling function for the Inverse-Gamma distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rinvgamma(3.0,2.0,engine);
```

Parameters

- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Inverse-Gamma distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rinvgamma**(const *T1* shape_par, const *T2* rate_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Inverse-Gamma distribution.

Example:

```
stats::rinvgamma(3.0,2.0,1776);
```

Parameters

- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Inverse-Gamma distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rinvgamma**(const ullint_t n, const ullint_t k, const *T1* shape_par, const *T2* rate_par, rand_engine_t &engine)

Random matrix sampling function for the Inverse-Gamma distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rinvgamma<std::vector<double>>(5,4,3.0,2.0,engine);
// Armadillo matrix
stats::rinvgamma<arma::mat>(5,4,3.0,2.0,engine);
// Blaze dynamic matrix
stats::rinvgamma<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0,
↪engine);
// Eigen dynamic matrix
stats::rinvgamma<Eigen::MatrixXd>(5,4,3.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Inverse-Gamma distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rinvgamma**(const ullint_t n, const ullint_t k, const *T1* shape_par, const *T2* rate_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Inverse-Gamma distribution.

Example:

```
// std::vector
stats::rinvgamma<std::vector<double>>(5,4,3.0,2.0);
// Armadillo matrix
stats::rinvgamma<arma::mat>(5,4,3.0,2.0);
```

(continues on next page)

(continued from previous page)

```
// Blaze dynamic matrix
stats::rinvgamma<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0);
// Eigen dynamic matrix
stats::rinvgamma<Eigen::MatrixXd>(5,4,3.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **shape_par** – the shape parameter, a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Inverse-Gamma distribution.

dinvgamma	density function of the inverse Gamma distribution
pinvgamma	distribution function of the inverse Gamma distribution
qinvgamma	quantile function of the inverse Gamma distribution
rinvgamma	random sampling function of the inverse Gamma distribution

2.6.10 Inverse-Gaussian Distribution

Table of contents

Density Function

The density function of the inverse Gaussian distribution:

$$f(x;\mu,\lambda) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left[-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right] \times \mathbf{1}[x \geq 0]$$

where μ is the mean parameter and λ is the shape parameter.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dinvgauss**(const *T1* x, const *T2* mu_par, const *T3* lambda_par, const bool log_form = false) noexcept

Density function of the inverse Gaussian distribution.

Example:

```
stats::dinvgauss(0.5,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dinvgauss**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* lambda_par, const bool log_form = false)

Density function of the inverse Gaussian distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::dinvgauss(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dinvgauss**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* lambda_par, const bool log_form = false)

Density function of the inverse Gaussian distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::dinvgauss(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dinvgauss**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* lambda_par, const bool log_form = false)

Density function of the inverse Gaussian distribution.

Example:

```
stats::dinvgauss(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dinvgauss**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* lambda_par, const bool log_form = false)

Density function of the inverse Gaussian distribution.

Example:

```
stats::dinvgauss(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the inverse Gaussian distribution:

$$F(x; \mu, \lambda) = \Phi\left(\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} - 1\right)\right) + \exp\left(\frac{2\lambda}{\mu}\right)\Phi\left(-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} + 1\right)\right)$$

where $\Phi(\cdot)$ denotes the standard Normal CDF.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **pinvgauss**(const *T1* x, const *T2* mu_par, const *T3* lambda_par, const bool log_form = false) noexcept

Distribution function of the inverse Gaussian distribution.

Example:

```
stats::pinvgauss(2.0,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **pinvgauss**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* lambda_par, const bool log_form = false)

Distribution function of the inverse Gaussian distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::pinvgauss(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.

- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **pinvgauss**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* lambda_par, const bool log_form = false)

Distribution function of the inverse Gaussian distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::pinvgauss(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pinvgauss**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* lambda_par, const bool log_form = false)

Distribution function of the inverse Gaussian distribution.

Example:

```
stats::pinvgauss(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.

- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pinvgauss**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* lambda_par, const bool log_form = false)

Distribution function of the inverse Gaussian distribution.

Example:

```
stats::pinvgauss(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the inverse Gaussian distribution:

$$q(p; \mu, \lambda) = \inf \{x : p \leq F(x; \mu, \lambda)\}$$

where $F(\cdot)$ denotes the CDF of the inverse Gaussian distribution.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qinvgauss**(const *T1* p, const *T2* mu_par, const *T3* lambda_par) noexcept

Quantile function of the inverse Gaussian distribution.

Example:

```
stats::qinvgauss(0.5,1.0,2.0);
```

Parameters

- **p** – a real-valued input.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.

Returns
the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qinvgauss**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* lambda_par)

Quantile function of the inverse Gaussian distribution.

Example:

```
std::vector<double> x = {0.1, 0.3, 0.7};
stats::qinvgauss(x,1.0,2.0);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.

Returns
a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qinvgauss**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* lambda_par)

Quantile function of the inverse Gaussian distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qinvgauss(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qinvgauss**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* lambda_par)

Quantile function of the inverse Gaussian distribution.

Example:

```
stats::qinvgauss(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qinvgauss**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* lambda_par)

Quantile function of the inverse Gaussian distribution.

Example:

```
stats::qinvgauss(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Random Sampling

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rinvgauss**(const *T1* mu_par, const *T2* lambda_par, rand_engine_t &engine)

Random sampling function for the inverse Gaussian distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rinvgauss(1.0,2.0,engine);
```

Parameters

- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the inverse Gaussian distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rinvgauss**(const *T1* mu_par, const *T2* lambda_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the inverse Gaussian distribution.

Example:

```
stats::rinvgauss(1.0,2.0,1776);
```

Parameters

- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the inverse Gaussian distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1** = double, typename **T2** = double>
inline *mT* **rinvgauss**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* lambda_par, rand_engine_t &engine)

Random matrix sampling function for the inverse Gaussian distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rinvgauss<std::vector<double>>(5,4,1.0,2.0,engine);
// Armadillo matrix
stats::rinvgauss<arma::mat>(5,4,1.0,2.0,engine);
// Blaze dynamic matrix
stats::rinvgauss<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0,
↪engine);
// Eigen dynamic matrix
stats::rinvgauss<Eigen::MatrixXd>(5,4,1.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the inverse Gaussian distribution.

2. Seed values

template<typename **mT**, typename **T1** = double, typename **T2** = double>
inline *mT* **rinvgauss**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* lambda_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the inverse Gaussian distribution.

Example:

```
// std::vector
stats::rinvgauss<std::vector<double>>(5,4,1.0,2.0);
// Armadillo matrix
stats::rinvgauss<arma::mat>(5,4,1.0,2.0);
```

(continues on next page)

(continued from previous page)

```
// Blaze dynamic matrix
stats::rinvgauss<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0);
// Eigen dynamic matrix
stats::rinvgauss<Eigen::MatrixXd>(5,4,1.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the mean parameter, a real-valued input.
- **lambda_par** – the shape parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the inverse Gaussian distribution.

dinvgauss	density function of the inverse Gaussian distribution
pinvgauss	distribution function of the inverse Gaussian distribution
qinvgauss	quantile function of the inverse Gaussian distribution
rinvgauss	random sampling function of the inverse Gaussian distribution

2.6.11 Inverse-Wishart Distribution

Table of contents

Density Function

The density function of the inverse-Wishart distribution:

$$f(\mathbf{X}; \mathbf{\Psi}, \nu) = \frac{|\mathbf{\Psi}|^{\frac{\nu}{2}}}{2^{\frac{\nu p}{2}} \Gamma_p \left(\frac{\nu}{2} \right)} |\mathbf{X}|^{-\frac{\nu + p + 1}{2}} \exp \left(-\frac{1}{2} \text{tr}(\mathbf{\Psi} \mathbf{X}^{-1}) \right)$$

where Γ_p is the Multivariate Gamma function, $| \cdot |$ denotes the matrix determinant, and $\text{tr}(\cdot)$ denotes the matrix trace.

template<typename **mT**, typename **pT**, typename not_arma_mat<*mT*>::type* = nullptr>

inline return_t<*pT*> **dinvwish**(const *mT* &X, const *mT* &Psi_par, const *pT* nu_par, const bool log_form = false)

Density function of the Inverse-Wishart distribution.

Parameters

- **X** – a positive semi-definite matrix.
- **Psi_par** – a positive semi-definite scale matrix.
- **nu_par** – the degrees of parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at X.

Random Sampling

Random sampling for the inverse-Wishart distribution is achieved via the method of Feiveson and Odell (1966).

template<typename **mT**, typename **pT**, typename not_arma_mat<*mT*>::type* = nullptr>
inline *mT* **rinvwish**(const *mT* &Psi_par, const *pT* nu_par, rand_engine_t &engine, const bool pre_inv_chol = false)

Random sampling function for the Inverse-Wishart distribution.

Parameters

- **Psi_par** – a positive semi-definite scale matrix.
- **nu_par** – the degrees of parameter, a real-valued input.
- **engine** – a random engine, passed by reference.
- **pre_inv_chol** – indicate whether Psi_par has been inverted and passed in lower triangular (Cholesky) format.

Returns

a pseudo-random draw from the Inverse-Wishart distribution.

dinvwish	density function of the inverse Wishart distribution
rinvwish	random sampling function of the inverse Wishart distribution

2.6.12 Laplace Distribution

Table of contents

Density Function

The density function of the Laplace distribution:

$$f(x; \mu, \sigma) = \frac{1}{2\sigma} \exp\left(- \frac{|x - \mu|}{\sigma} \right)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dlaplace**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Density function of the Laplace distribution.

Example:

```
stats::dlaplace(0.7,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dlaplace**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Laplace distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::dlaplace(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dlaplace**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Laplace distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::dlaplace(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dlaplace**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Laplace distribution.

Example:

```
stats::dlaplace(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dlaplace**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Laplace distribution.

Example:

```
stats::dlaplace(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a matrix of density function values corresponding to the elements of `X`.

Cumulative Distribution Function

The cumulative distribution function of the Laplace distribution:

$$F(x;\mu,\sigma)=\int_{-\infty}^{x} f(z;\mu,\sigma)dz=\frac{1}{2}+\frac{1}{2}\times \text{sign}(x-\mu)\times\left(1-\exp\left(-\frac{|x-\mu|}{\sigma}\right)\right)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **plaplace**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Distribution function of the Laplace distribution.

Example:

```
stats::plaplace(0.7,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **plaplace**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Laplace distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::plaplace(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **plaplace**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Laplace distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::plaplace(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **plaplace**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Laplace distribution.

Example:

```
stats::plaplace(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **plaplace**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Laplace distribution.

Example:

```
stats::plaplace(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Laplace distribution:

$$q(p; \mu, \sigma) = \mu - \sigma \times \text{sign}(p - 0.5) \times \ln(1 - 2|p - 0.5|)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qlaplace**(const *T1* p, const *T2* mu_par, const *T3* sigma_par) noexcept

Quantile function of the Laplace distribution.

Example:

```
stats::qlaplace(0.7,1.0,2.0);
```

Parameters

- **p** – a real-valued input.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qlaplace**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Laplace distribution.

Example:

```
std::vector<double> x = {0.1, 0.3, 0.7};
stats::qlaplace(x,1.0,2.0);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qlaplace**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Laplace distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qlaplace(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qlaplace**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Laplace distribution.

Example:

```
stats::qlaplace(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qlaplace**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Laplace distribution.

Example:

```
stats::qlaplace(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Laplace distribution is achieved via the inverse probability integral transform.

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rlaplace**(const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random sampling function for the Laplace distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rlaplace(1.0,2.0,engine);
```

Parameters

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Laplace distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rlaplace**(const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Lapalce distribution.

Example:

```
stats::rlaplace(1.0,2.0,1776);
```

Parameters

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Laplace distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rlaplace**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random matrix sampling function for the Laplace distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rlaplace<std::vector<double>>(5,4,1.0,2.0,engine);
// Armadillo matrix
stats::rlaplace<arma::mat>(5,4,1.0,2.0,engine);
// Blaze dynamic matrix
stats::rlaplace<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0,
↪engine);
// Eigen dynamic matrix
stats::rlaplace<Eigen::MatrixXd>(5,4,1.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Laplace distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rlaplace**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Laplace distribution.

Example:

```
// std::vector
stats::rlaplace<std::vector<double>>(5,4,1.0,2.0);
// Armadillo matrix
stats::rlaplace<arma::mat>(5,4,1.0,2.0);
// Blaze dynamic matrix
```

(continues on next page)

(continued from previous page)

```
stats::rlaplace<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0);
// Eigen dynamic matrix
stats::rlaplace<Eigen::MatrixXd>(5,4,1.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Laplace distribution.

dlaplace	density function of the Laplace distribution
plaplace	distribution function of the Laplace distribution
qlaplace	quantile function of the Laplace distribution
rlaplace	random sampling function of the Laplace distribution

2.6.13 Log-Normal Distribution

Table of contents

Density Function

The density function of the log-Normal distribution:

$$f(x;\mu,\sigma)=\frac{1}{x}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(\ln x-\mu)^2}{2\sigma^2}\right)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dlnorm**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Density function of the Log-Normal distribution.

Example:

```
stats::dlnorm(2.0,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dlnorm**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Log-Normal distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::dlnorm(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dlnorm**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Log-Normal distribution.

Example:

```
arma::mat X = { {0.2, 1.7, 0.1},
                {0.9, 4.0, 0.3} };
stats::dlnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dlnorm**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Log-Normal distribution.

Example:

```
stats::dlnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dlnorm**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Log-Normal distribution.

Example:

```
stats::dlnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the log-Normal distribution:

$$F(x;\mu,\sigma) = \int_0^x f(z;\mu,\sigma)dz = \frac{1}{2} + \frac{1}{2} \times \text{erf}\left(\frac{\ln(x) - \mu}{\sigma}\right)$$

where $\text{erf}(\cdot)$ denotes the Gaussian error function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **plnorm**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Distribution function of the Log-Normal distribution.

Example:

```
stats::plnorm(2.0,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **plnorm**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Log-Normal distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::plnorm(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **plnorm**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Log-Normal distribution.

Example:

```
arma::mat X = { {0.2, 1.7, 0.1},
                {0.9, 4.0, 0.3} };
stats::plnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **plnorm**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Log-Normal distribution.

Example:

```
stats::plnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **plnorm**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Log-Normal distribution.

Example:

```
stats::plnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the log-Normal distribution:

$$q(p; \mu, \sigma) = \exp\left(\mu + \sqrt{2}\sigma \times \text{erf}^{-1}\left(2p - 1\right)\right)$$

where $\text{erf}^{-1}(\cdot)$ denotes the inverse Gaussian error function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qlnorm**(const *T1* p, const *T2* mu_par, const *T3* sigma_par) noexcept

Quantile function of the Log-Normal distribution.

Example:

```
stats::qlnorm(0.6,1.0,2.0);
```

Parameters

- **p** – a real-valued input.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.

Returns

the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qlnorm**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Log-Normal distribution.

Example:

```
std::vector<double> x = {0.1, 0.3, 0.7};
stats::qlnorm(x,1.0,2.0);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qlnorm**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Log-Normal distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qlnorm(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qlnorm**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Log-Normal distribution.

Example:

```
stats::qlnorm(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qlnorm**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Log-Normal distribution.

Example:

```
stats::qlnorm(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the log-Normal distribution is achieved by simulating $X \sim N(\mu, \sigma^2)$, then returning

$$Z = \exp(X) \sim \text{Lognormal}(\mu, \sigma^2)$$

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rlnorm**(const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random sampling function for the Log-Normal distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rlnorm(1.0,2.0,engine);
```

Parameters

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Log-Normal distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rlnorm**(const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Log-Normal distribution.

Example:

```
stats::rlnorm(1.0,2.0,1776);
```

Parameters

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Log-Normal distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rlnorm**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random matrix sampling function for the Log-Normal distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rlnorm<std::vector<double>>(5,4,1.0,2.0,engine);
// Armadillo matrix
stats::rlnorm<arma::mat>(5,4,1.0,2.0,engine);
// Blaze dynamic matrix
stats::rlnorm<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0,engine);
// Eigen dynamic matrix
stats::rlnorm<Eigen::MatrixXd>(5,4,1.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Log-Normal distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rlnorm**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Log-Normal distribution.

Example:

```
// std::vector
stats::rlnorm<std::vector<double>>(5,4,1.0,2.0);
// Armadillo matrix
stats::rlnorm<arma::mat>(5,4,1.0,2.0);
// Blaze dynamic matrix
stats::rlnorm<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0);
```

(continues on next page)

(continued from previous page)

```
// Eigen dynamic matrix
stats::rlnorm<Eigen::MatrixXd>(5,4,1.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Log-Normal distribution.

dlnorm	density function of the log Normal distribution
plnorm	distribution function of the log Normal distribution
qlnorm	quantile function of the log Normal distribution
rlnorm	random sampling function of the log Normal distribution

2.6.14 Logistic Distribution

Table of contents

Density Function

The density function of the Logistic distribution:

$$f(x;\mu,\sigma) = \frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma\left(1+\exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dlogis**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Density function of the Logistic distribution.

Example:

```
stats::dlogis(2.0,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dlogis**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Logistic distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::dlogis(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dlogis**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Logistic distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::dlogis(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dlogis**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Logistic distribution.

Example:

```
stats::dlogis(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dlogis**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Logistic distribution.

Example:

```
stats::dlogis(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Logistic distribution:

$$F(x;\mu,\sigma) = \int_{-\infty}^{x} f(z;\mu,\sigma)dz = \frac{1}{1+\exp\left(-\frac{x-\mu}{\sigma}\right)}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **plogis**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Distribution function of the Logistic distribution.

Example:

```
stats::plogis(2.0,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **plogis**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Logistic distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::plogis(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.

- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **plogis**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Logistic distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::plogis(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **plogis**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Logistic distribution.

Example:

```
stats::plogis(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **plogis**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Logistic distribution.

Example:

```
stats::plogis(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Logistic distribution:

$$q(p; \mu, \sigma) = \mu + \sigma \times \ln \left(\frac{p}{1-p} \right)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qlogis**(const *T1* p, const *T2* mu_par, const *T3* sigma_par) noexcept

Quantile function of the Logistic distribution.

Example:

```
stats::qlogis(0.5,1.0,2.0);
```

Parameters

- **p** – a real-valued input.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns
the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qlogis**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Logistic distribution.

Example:

```
std::vector<double> x = {0.1, 0.3, 0.7};
stats::qlogis(x,1.0,2.0);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns
a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qlogis**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Logistic distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qlogis(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qlogis**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Logistic distribution.

Example:

```
stats::qlogis(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of `X`.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qlogis**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Logistic distribution.

Example:

```
stats::qlogis(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of `X`.

Random Sampling

Random sampling for the Logistic distribution is achieved via the inverse probability integral transform.

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rlogis**(const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random sampling function for the Logistic distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rlogis(1.0,2.0,engine);
```

Parameters

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Logistic distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rlogis**(const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Logistic distribution.

Example:

```
stats::rlogis(1.0,2.0,1776);
```

Parameters

- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Logistic distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rlogis**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random matrix sampling function for the Logistic distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rlogis<std::vector<double>>(5,4,1.0,2.0,engine);
// Armadillo matrix
stats::rlogis<arma::mat>(5,4,1.0,2.0,engine);
// Blaze dynamic matrix
stats::rlogis<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0,engine);
// Eigen dynamic matrix
stats::rlogis<Eigen::MatrixXd>(5,4,1.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Logistic distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rlogis**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Logistic distribution.

Example:

```
// std::vector
stats::rlogis<std::vector<double>>(5,4,1.0,2.0);
// Armadillo matrix
stats::rlogis<arma::mat>(5,4,1.0,2.0);
// Blaze dynamic matrix
stats::rlogis<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0);
```

(continues on next page)

(continued from previous page)

```
// Eigen dynamic matrix
stats::rlogis<Eigen::MatrixXd>(5,4,1.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the location parameter, a real-valued input.
- **sigma_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Logistic distribution.

dlogis	density function of the Logistic distribution
plogis	distribution function of the Logistic distribution
qlogis	quantile function of the Logistic distribution
rlogis	random sampling function of the Logistic distribution

2.6.15 Multivariate-Normal Distribution

Table of contents

Density Function

The density function of the Multivariate-Normal distribution:

$$f(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right)$$

where k is the dimension of the real-valued vector $\mathbf{x}$ and $| \cdot |$ denotes the matrix determinant.

template<typename **vT**, typename **mT**, typename **eT** = double>
inline *eT* **dmvnorm**(const *vT* &X, const *vT* &mu_par, const *mT* &Sigma_par, const bool log_form = false)

Density function of the Multivariate-Normal distribution.

Parameters

- **X** – a column vector.

- **mu_par** – mean vector.
- **Sigma_par** – the covariance matrix.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at `X`.

Random Sampling

template<typename **vT**, typename **mT**, typename not_arma_mat<*mT*>::type* = nullptr>
inline *vT* **rmvnorm**(const *vT* &mu_par, const *mT* &Sigma_par, rand_engine_t &engine, const bool pre_chol = false)

Random sampling function for the Multivariate-Normal distribution.

Parameters

- **mu_par** – mean vector.
- **Sigma_par** – the covariance matrix.
- **engine** – a random engine, passed by reference.
- **pre_chol** – indicate whether `Sigma_par` is passed in lower triangular (Cholesky) format.

Returns

a pseudo-random draw from the Multivariate-Normal distribution.

dmvnorm	density function of the Multivariate Normal Distribution
rmvnorm	random sampling function of the Multivariate Normal distribution

2.6.16 Normal Distribution

Table of contents

Density Function

The density function of the Normal (Gaussian) distribution:

$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dnorm**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Density function of the Normal distribution.

Example:

```
stats::dnorm(0.5,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at `x`.

template<typename **T**>
constexpr return_t<*T*> **dnorm**(const *T* x, const bool log_form = false) noexcept

Density function of the standard Normal distribution.

Example:

```
stats::dnorm(0.5,false);
```

Parameters

- **x** – a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dnorm**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Normal distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::dnorm(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dnorm**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Normal distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::dnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dnorm**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Normal distribution.

Example:

```
stats::dnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dnorm**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Density function of the Normal distribution.

Example:

```
stats::dnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Normal (Gaussian) distribution:

$$F(x;\mu,\sigma) = \int_{-\infty}^{x} f(z;\mu,\sigma)dz = \frac{1}{2} \times \left(1 + \text{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma} \right) \right)$$

where $\text{erf}(\cdot)$ denotes the Gaussian error function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **pnorm**(const *T1* x, const *T2* mu_par, const *T3* sigma_par, const bool log_form = false) noexcept

Distribution function of the Normal distribution.

Example:

```
stats::pnorm(2.0,1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.

- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at `x`.

template<typename **T**>
constexpr return_t<*T*> **pnorm**(const *T* x, const bool log_form = false) noexcept

Distribution function of the standard Normal distribution.

Example:

```
stats::pnorm(1.0,false);
```

Parameters

- **x** – a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **pnorm**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Normal distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::pnorm(x,1.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **pnorm**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Normal distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::pnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pnorm**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Normal distribution.

Example:

```
stats::pnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pnorm**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par, const bool log_form = false)

Distribution function of the Normal distribution.

Example:

```
stats::pnorm(X,1.0,1.0,false);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the log-Normal distribution:

$$q(p; \mu, \sigma) = \mu + \sqrt{2}\sigma \times \text{erf}^{-1}\left(2p - 1\right)$$

where $\text{erf}^{-1}(\cdot)$ denotes the inverse Gaussian error function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qnorm**(const *T1* p, const *T2* mu_par, const *T3* sigma_par) noexcept

Quantile function of the Normal distribution.

Example:

```
stats::qnorm(0.5,1.0,2.0);
```

Parameters

- **p** – a real-valued input.
- **mu_par** – the mean parameter, a real-valued input.

- **sigma_par** – the standard deviation parameter, a real-valued input.

Returns
the quantile function evaluated at p.

template<typename **T**>
constexpr return_t<*T*> **qnorm**(const *T* p) noexcept

Quantile function of the standard Normal distribution.

Example:

```
stats::qnorm(0.5);
```

Parameters
p – a real-valued input.

Returns
the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qnorm**(const std::vector<*eT*> &x, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Normal distribution.

Example:

```
std::vector<double> x = {0.1, 0.3, 0.7};
stats::qnorm(x,1.0,2.0);
```

Parameters

- **x** – a standard vector.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.

Returns
a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qnorm**(const ArmaMat<*eT*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Normal distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qnorm(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qnorm**(const BlazeMat<*eT*, *To*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Normal distribution.

Example:

```
stats::qnorm(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qnorm**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* mu_par, const *T2* sigma_par)

Quantile function of the Normal distribution.

Example:

```
stats::qnorm(X,1.0,1.0);
```

Parameters

- **X** – a matrix of input values.
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Random Sampling

Random sampling for the Normal distribution is achieved via the `normal_distribution` class from the C++ standard library, imported from <`random`>.

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rnorm**(const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random sampling function for the Normal distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rnorm(1.0,2.0,engine);
```

Parameters

- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Normal distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rnorm**(const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Normal distribution.

Example:

```
stats::rnorm(1.0,2.0,1776);
```

Parameters

- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns
a pseudo-random draw from the Normal distribution.

3. Convenience

template<typename **T** = double>
inline *T* **rnorm**()

Random sampling function for the standard Normal distribution.

Example:

```
stats::rnorm();
```

Returns
a pseudo-random draw from the standard Normal distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1** = double, typename **T2** = double>
inline *mT* **rnorm**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, rand_engine_t &engine)

Random matrix sampling function for the Normal distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rnorm<std::vector<double>>(5,4,1.0,2.0,engine);
// Armadillo matrix
stats::rnorm<arma::mat>(5,4,1.0,2.0,engine);
// Blaze dynamic matrix
stats::rnorm<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0,engine);
```

(continues on next page)

(continued from previous page)

```
// Eigen dynamic matrix
stats::rnorm<Eigen::MatrixXd>(5,4,1.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Normal distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rnorm**(const ullint_t n, const ullint_t k, const *T1* mu_par, const *T2* sigma_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Normal distribution.

Example:

```
// std::vector
stats::rnorm<std::vector<double>>(5,4,1.0,2.0);
// Armadillo matrix
stats::rnorm<arma::mat>(5,4,1.0,2.0);
// Blaze dynamic matrix
stats::rnorm<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,1.0,2.0);
// Eigen dynamic matrix
stats::rnorm<Eigen::MatrixXd>(5,4,1.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **mu_par** – the mean parameter, a real-valued input.
- **sigma_par** – the standard deviation parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns
a matrix of pseudo-random draws from the Normal distribution.

dnorm	density function of the Normal distribution
pnorm	distribution function of the Normal distribution
qnorm	quantile function of the Normal distribution
rnorm	random sampling function of the Normal distribution

2.6.17 Poisson Distribution

Table of contents

Density Function

The density function of the Poisson distribution:

$$f(x;\lambda) = \frac{\lambda^x \exp(-\lambda)}{x!} \times \mathbf{1}[x \geq 0]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T**>
constexpr return_t<*T*> **dpois**(const llint_t x, const *T* rate_par, const bool log_form = false) noexcept

Density function of the Poisson distribution.

Example:

```
stats::dpois(8.0,10.0,false);
```

Parameters

- **x** – a non-negative integral-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **dpois**(const std::vector<*eT*> &x, const *T1* rate_par, const bool log_form = false)

Density function of the Poisson distribution.

Example:

```
std::vector<int> x = {2, 3, 4};
stats::dpois(x,4,false);
```

Parameters

- **x** – a standard vector.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **dpois**(const ArmaMat<*eT*> &X, const *T1* rate_par, const bool log_form = false)

Density function of the Poisson distribution.

Example:

```
arma::mat X = { {2, 1, 4},
                {3, 5, 6} };
stats::dpois(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dpois**(const BlazeMat<*eT*, *To*> &X, const *T1* rate_par, const bool log_form = false)

Density function of the Poisson distribution.

Example:

```
stats::dpois(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dpois**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* rate_par, const bool log_form = false)

Density function of the Poisson distribution.

Example:

```
stats::dpois(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Poisson distribution:

$$F(x;\lambda) = \sum_{z \leq x} f(z;\lambda) = \exp(-\lambda) \sum_{z \leq x} \frac{\lambda^z}{z!} \times \mathbf{1}[z \geq 0]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T**>
constexpr return_t<*T*> **ppois**(const llint_t x, const *T* rate_par, const bool log_form = false) noexcept

Distribution function of the Poisson distribution.

Example:

```
stats::ppois(8.0,10.0,false);
```

Parameters

- **x** – a non-negative integral-valued input.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **ppois**(const std::vector<*eT*> &x, const *T1* rate_par, const bool log_form = false)

Distribution function of the Poisson distribution.

Example:

```
std::vector<int> x = {2, 3, 4};
stats::ppois(x,2.0,false);
```

Parameters

- **x** – a standard vector.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **ppois**(const ArmaMat<*eT*> &X, const *T1* rate_par, const bool log_form = false)

Distribution function of the Poisson distribution.

Example:

```
arma::mat X = { {2, 1, 4},
                {3, 5, 6} };
stats::ppois(X,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **ppois**(const BlazeMat<*eT*, *To*> &X, const *T1* rate_par, const bool log_form = false)

Distribution function of the Poisson distribution.

Example:

```
stats::ppois(X,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **ppois**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* rate_par, const bool log_form = false)

Distribution function of the Poisson distribution.

Example:

```
stats::ppois(X,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Poisson distribution:

$$q(p; \lambda) = \inf \{x : p \leq F(x; \lambda)\}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **qpois**(const *T1* p, const *T2* rate_par) noexcept

Quantile function of the Poisson distribution.

Example:

```
stats::qpois(0.6,10.0);
```

Parameters

- **p** – a real-valued input.
- **rate_par** – the rate parameter, a real-valued input.

Returns

the quantile function evaluated at `p`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **qpois**(const std::vector<*eT*> &x, const *T1* rate_par)

Quantile function of the Poisson distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.8};
stats::qpois(x,4);
```

Parameters

- **x** – a standard vector.
- **rate_par** – the rate parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **qpois**(const ArmaMat<*eT*> &X, const *T1* rate_par)

Quantile function of the Poisson distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qpois(X,4);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qpois**(const BlazeMat<*eT*, *To*> &X, const *T1* rate_par)

Quantile function of the Poisson distribution.

Example:

```
stats::qpois(X,4);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qpois**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* rate_par)

Quantile function of the Poisson distribution.

Example:

```
stats::qpois(X,4);
```

Parameters

- **X** – a matrix of input values.
- **rate_par** – the rate parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Random Sampling

Scalar Output

1. Random number engines

template<typename **T**>
inline return_t<*T*> **rpois**(const *T* rate_par, rand_engine_t &engine)

Random sampling function for the Poisson distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rchisq(4,engine);
```

Parameters

- **rate_par** – the rate parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns
a pseudo-random draw from the Poisson distribution.

2. Seed values

template<typename **T**>
inline return_t<*T*> **rpois**(const *T* rate_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Poisson distribution.

Example:

```
stats::rchisq(4,1776);
```

Parameters

- **rate_par** – the rate parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns
a pseudo-random draw from the Poisson distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**>
inline *mT* **rpois**(const ullint_t n, const ullint_t k, const *T1* rate_par, rand_engine_t &engine)

Random matrix sampling function for the Poisson distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rpois<std::vector<double>>(5,4,4,engine);
// Armadillo matrix
stats::rpois<arma::mat>(5,4,4,engine);
// Blaze dynamic matrix
stats::rpois<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,4,engine);
// Eigen dynamic matrix
stats::rpois<Eigen::MatrixXd>(5,4,4,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **rate_par** – the rate parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Poisson distribution.

2. Seed values

template<typename **mT**, typename **T1**>
inline *mT* **rpois**(const ullint_t n, const ullint_t k, const *T1* rate_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Poisson distribution.

Example:

```
// std::vector
stats::rpois<std::vector<double>>(5,4,4);
// Armadillo matrix
stats::rpois<arma::mat>(5,4,4);
// Blaze dynamic matrix
stats::rpois<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,4);
// Eigen dynamic matrix
stats::rpois<Eigen::MatrixXd>(5,4,4);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **`n`** – the number of output rows
- **`k`** – the number of output columns
- **`rate_par`** – the rate parameter, a real-valued input.
- **`seed_val`** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Poisson distribution.

dpois	density function of the Poisson distribution
ppois	distribution function of the Poisson distribution
qpois	quantile function of the Poisson distribution
rpois	random sampling function of the Poisson distribution

2.6.18 Rademacher Distribution

Table of contents

Density Function

The density function of the Rademacher distribution:

$$f(x;p) = p \times \mathbf{1}[x = 1] + (1 - p) \times \mathbf{1}[x = -1]$$

Note that this is a somewhat more general definition of the Rademacher distribution than is standard in the statistics literature, which assumes $p = 0.5$.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T**>
constexpr return_t<*T*> **dradem**(const llint_t x, const *T* prob_par, const bool log_form = false) noexcept

Density function of the Rademacher distribution.

Example:

```
stats::dradem(1,0.6,false);
```

Parameters

- **x** – an integral-valued input, equal to -1 or 1.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>

inline std::vector<*rT*> **dradem**(const std::vector<*eT*> &x, const *T1* prob_par, const bool log_form = false)

Density function of the Rademacher distribution.

Example:

```
std::vector<int> x = {-1, 1, 0};
stats::dradem(x,0.5,false);
```

Parameters

- **x** – a standard vector.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **mT**, typename **tT**, typename **T1**>
inline *mT* **dradem**(const ArmaGen<*mT*, *tT*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Rademacher distribution.

Example:

```
arma::mat X = { {-1, 0,  1},
                {-1, 1, -1} };
stats::dradem(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dradem**(const BlazeMat<*eT*, *To*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Rademacher distribution.

Example:

```
stats::dradem(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of **X**.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dradem**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Rademacher distribution.

Example:

```
stats::dradem(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of **X**.

Cumulative Distribution Function

The cumulative distribution function of the Rademacher distribution:

$$F(x;p) = \sum_{z \leq x} f(z;p) = \begin{cases} 0 & \text{if } x < -1 \\ 1-p & \text{if } x < 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T**>
constexpr return_t<*T*> **pradem**(const llint_t x, const *T* prob_par, const bool log_form = false) noexcept

Distribution function of the Rademacher distribution.

Example:

```
stats::pradem(1,0.6,false);
```

Parameters

- **x** – a value equal to 0 or 1.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **pradem**(const std::vector<*eT*> &x, const *T1* prob_par, const bool log_form = false)

Density function of the Rademacher distribution.

Example:

```
std::vector<int> x = {0, 1, 0};
stats::pradem(x,0.5,false);
```

Parameters

- **x** – a standard vector.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **pradem**(const ArmaMat<*eT*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Rademacher distribution.

Example:

```
arma::mat X = { {1, 0, 1},
                {0, 1, 0} };
stats::pradem(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pradem**(const BlazeMat<*eT*, *To*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Rademacher distribution.

Example:

```
stats::pradem(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pradem**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* prob_par, const bool log_form = false)

Density function of the Rademacher distribution.

Example:

```
stats::pradem(X,0.5,false);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Rademacher distribution:

$$q(r; p) = \begin{cases} -1 & \text{if } r \leq 1 - p \\ 1 & \text{else} \end{cases}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **qradem**(const *T1* p, const *T2* prob_par) noexcept

Quantile function of the Rademacher distribution.

Example:

```
stats::qradem(0.5,0.4);
```

Parameters

- **p** – a real-valued input.
- **prob_par** – the probability parameter, a real-valued input.

Returns

the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **qradem**(const std::vector<*eT*> &x, const *T1* prob_par)

Quantile function of the Rademacher distribution.

Example:

```
std::vector<int> x = {0.4, 0.5, 0.9};
stats::qradem(x,0.5);
```

Parameters

- **x** – a standard vector.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **qradem**(const ArmaMat<*eT*> &X, const *T1* prob_par)

Quantile function of the Rademacher distribution.

Example:

```
arma::mat X = { {0.4, 0.5, 0.9},
                {0.3, 0.6, 0.7} };
stats::qradem(X,0.5);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qradem**(const BlazeMat<*eT*, *To*> &X, const *T1* prob_par)

Quantile function of the Rademacher distribution.

Example:

```
stats::qradem(X,0.5);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qradem**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* prob_par)

Quantile function of the Rademacher distribution.

Example:

```
stats::qradem(X,0.5);
```

Parameters

- **X** – a matrix of input values.
- **prob_par** – the probability parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Rademacher distribution is achieved via the inverse probability integral transform.

Scalar Output

1. Random number engines

template<typename **T**>
inline return_t<*T*> **rradem**(const *T* prob_par, rand_engine_t &engine)

Random sampling function for the Rademacher distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rradem(0.7,engine);
```

Parameters

- **prob_par** – the probability parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Rademacher distribution.

2. Seed values

template<typename **T**>
inline return_t<*T*> **rradem**(const *T* prob_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Rademacher distribution.

Example:

```
stats::rradem(0.7,1776);
```

Parameters

- **prob_par** – the probability parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Rademacher distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**>
inline *mT* **rradem**(const ullint_t n, const ullint_t k, const *T1* prob_par, rand_engine_t &engine)

Random matrix sampling function for the Rademacher distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rradem<std::vector<double>>(5,4,0.7,engine);
// Armadillo matrix
stats::rradem<arma::mat>(5,4,0.7,engine);
// Blaze dynamic matrix
stats::rradem<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,0.7,engine);
// Eigen dynamic matrix
stats::rradem<Eigen::MatrixXd>(5,4,0.7,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **prob_par** – the probability parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Rademacher distribution.

2. Seed values

template<typename **mT**, typename **T1**>
inline *mT* **rradem**(const ullint_t n, const ullint_t k, const *T1* prob_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Rademacher distribution.

Example:

```
// std::vector
stats::rradem<std::vector<double>>(5,4,0.7);
// Armadillo matrix
stats::rradem<arma::mat>(5,4,0.7);
// Blaze dynamic matrix
stats::rradem<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,0.7);
// Eigen dynamic matrix
stats::rradem<Eigen::MatrixXd>(5,4,0.7);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns

- **prob_par** – the probability parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns
a matrix of pseudo-random draws from the Rademacher distribution.

dradem	density function of the Rademacher distribution
pradem	distribution function of the Rademacher distribution
qradem	quantile function of the Rademacher distribution
rradem	random sampling function of the Rademacher distribution

2.6.19 Student's t-Distribution

Table of contents

Density Function

The density function of the Student's t distribution:

$$f(x;\nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

where $\Gamma(\cdot)$ denotes the gamma function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **dt**(const *T1* x, const *T2* dof_par, const bool log_form = false) noexcept

Density function of the t-distribution.

Example:

```
stats::dt(0.37,11,false);
```

Parameters

- **x** – a real-valued input.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **dt**(const std::vector<*eT*> &x, const *T1* dof_par, const bool log_form = false)

Density function of the t-distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::dt(x,4,false);
```

Parameters

- **x** – a standard vector.

- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a vector of density function values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **dt**(const ArmaMat<*eT*> &X, const *T1* dof_par, const bool log_form = false)

Density function of the t-distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::dt(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dt**(const BlazeMat<*eT*, *To*> &X, const *T1* dof_par, const bool log_form = false)

Density function of the t-distribution.

Example:

```
stats::dt(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dt**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* dof_par, const bool log_form = false)

Density function of the t-distribution.

Example:

```
stats::dt(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Student's t distribution:

$$F(x;\nu) = \int_{-\infty}^{x} f(z;\nu)dz = \frac{1}{2} + x\Gamma\left(\frac{\nu+1}{2}\right) + \frac{{}_2F_1\left(\frac{1}{2}, \frac{\nu+1}{2}; \frac{3}{\nu}; -\frac{x^2}{\nu}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}$$

where $\Gamma(\cdot)$ denotes the gamma function and ${}_2F_1$ denotes the hypergeometric function.

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **pt**(const *T1* x, const *T2* dof_par, const bool log_form = false) noexcept

Distribution function of the t-distribution.

Example:

```
stats::pt(0.37,11,false);
```

Parameters

- **x** – a real-valued input.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **pt**(const std::vector<*eT*> &x, const *T1* dof_par, const bool log_form = false)

Distribution function of the t-distribution.

Example:

```
std::vector<double> x = {0.0, 1.0, 2.0};
stats::pt(x,4,false);
```

Parameters

- **x** – a standard vector.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **pt**(const ArmaMat<*eT*> &X, const *T1* dof_par, const bool log_form = false)

Distribution function of the t-distribution.

Example:

```
arma::mat X = { {0.2, -1.7,  0.1},
                {0.9,  4.0, -0.3} };
stats::pt(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pt**(const BlazeMat<*eT*, *To*> &X, const *T1* dof_par, const bool log_form = false)

Distribution function of the t-distribution.

Example:

```
stats::pt(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pt**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* dof_par, const bool log_form = false)

Distribution function of the t-distribution.

Example:

```
stats::pt(X,4,false);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Student's t distribution:

$$q(p; \nu) = \inf \{x : p \leq F(x; \nu)\}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**>
constexpr common_return_t<*T1*, *T2*> **qt**(const *T1* p, const *T2* dof_par) noexcept

Quantile function of the t-distribution.

Example:

```
stats::qt(0.5,11);
```

Parameters

- **p** – a real-valued input.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns

the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline std::vector<*rT*> **qt**(const std::vector<*eT*> &x, const *T1* dof_par)

Quantile function of the t-distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.8};
stats::qt(x,4);
```

Parameters

- **x** – a standard vector.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>>
inline ArmaMat<*rT*> **qt**(const ArmaMat<*eT*> &X, const *T1* dof_par)

Quantile function of the t-distribution.

Example:

```
arma::mat X = { {0.2, 0.7, 0.9},
                {0.1, 0.8, 0.3} };
stats::qt(X,4);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qt**(const BlazeMat<*eT*, *To*> &X, const *T1* dof_par)

Quantile function of the t-distribution.

Example:

```
stats::qt(X,4);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **rT** = common_return_t<*eT*, *T1*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qt**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* dof_par)

Quantile function of the t-distribution.

Example:

```
stats::qt(X,4);
```

Parameters

- **X** – a matrix of input values.
- **dof_par** – the degrees of freedom parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Scalar Output

1. Random number engines

template<typename **T**>
inline return_t<*T*> **rt**(const *T* dof_par, rand_engine_t &engine)

Random sampling function for Student's t-distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rt(4,engine);
```

Parameters

- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from Student's t-distribution.

2. Seed values

template<typename **T**>
inline return_t<*T*> **rt**(const *T* dof_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for Student's t-distribution.

Example:

```
stats::rt(4,1776);
```

Parameters

- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from Student's t-distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**>
inline *mT* **rt**(const ullint_t n, const ullint_t k, const *T1* dof_par, rand_engine_t &engine)

Random matrix sampling function for Student's t-distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rt<std::vector<double>>(5,4,12,engine);
// Armadillo matrix
stats::rt<arma::mat>(5,4,12,engine);
// Blaze dynamic matrix
stats::rt<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,12,engine);
// Eigen dynamic matrix
stats::rt<Eigen::MatrixXd>(5,4,12,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the t-distribution.

2. Seed values

template<typename **mT**, typename **T1**>
inline *mT* **rt**(const ullint_t n, const ullint_t k, const *T1* dof_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for Student's t-distribution.

Example:

```
// std::vector
stats::rt<std::vector<double>>(5,4,12);
// Armadillo matrix
stats::rt<arma::mat>(5,4,12);
// Blaze dynamic matrix
stats::rt<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,12);
// Eigen dynamic matrix
stats::rt<Eigen::MatrixXd>(5,4,12);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **dof_par** – the degrees of freedom parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from Student's t-distribution.

dt	density function of the t-distribution
pt	distribution function of the t-distribution
qt	quantile function of the t-distribution
rt	random sampling function of the t-distribution

2.6.20 Uniform Distribution

Table of contents

Density Function

The density function of the Uniform distribution:

$$f(x;a,b) = \frac{1}{b-a} \times \mathbf{1}[a \leq x \leq b]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dunif**(const *T1* x, const *T2* a_par, const *T3* b_par, const bool log_form = false) noexcept

Density function of the Uniform distribution.

Example:

```
stats::dunif(0.5,-1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dunif**(const std::vector<*eT*> &x, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Density function of the Uniform distribution.

Example:

```
std::vector<double> x = {-2.0, 0.0, 2.0};
stats::dunif(x,-1.0,3.0,false);
```

Parameters

- **x** – a standard vector.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a vector of density function values corresponding to the elements of `x`.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dunif**(const ArmaMat<*eT*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Density function of the Uniform distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9, -0.3,  1.3} };
stats::dunif(X,-1.0,3.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dunif**(const BlazeMat<*eT*, *To*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Density function of the Uniform distribution.

Example:

```
stats::dunif(X,-1.0,3.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dunif**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Density function of the Uniform distribution.

Example:

```
stats::dunif(X,-1.0,3.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Cumulative Distribution Function

The cumulative distribution function of the Uniform distribution:

$$F(x;a,b) = \int_a^x f(z;a,b)dz = \frac{x-a}{b-a} \times \mathbf{1}[a \le x \le b] + \times \mathbf{1}[x > b]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **punif**(const *T1* x, const *T2* a_par, const *T3* b_par, const bool log_form = false) noexcept

Distribution function of the Uniform distribution.

Example:

```
stats::punif(0.5,-1.0,2.0,false);
```

Parameters

- **x** – a real-valued input.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
the cumulative distribution function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **punif**(const std::vector<*eT*> &x, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Distribution function of the Uniform distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::punif(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **a_par** – the lower bound parameter, a real-valued input.

- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **punif**(const ArmaMat<*eT*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Distribution function of the Uniform distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9, -0.3,  1.3} };
stats::punif(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **punif**(const BlazeMat<*eT*, *To*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Distribution function of the Uniform distribution.

Example:

```
stats::punif(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **punif**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* a_par, const *T2* b_par, const bool log_form = false)

Distribution function of the Uniform distribution.

Example:

```
stats::punif(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of `X`.

Quantile Function

The quantile function of the Uniform distribution:

$$q(p; a, b) = a + p(b - a)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qunif**(const *T1* p, const *T2* a_par, const *T3* b_par) noexcept

Quantile function of the Uniform distribution.

Example:

```
stats::qunif(0.5,-1.0,2.0);
```

Parameters

- **p** – a real-valued input.

- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.

Returns
the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qunif**(const std::vector<*eT*> &x, const *T1* a_par, const *T2* b_par)

Quantile function of the Uniform distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::qunif(x,3.0,2.0);
```

Parameters

- **x** – a standard vector.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.

Returns
a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qunif**(const ArmaMat<*eT*> &X, const *T1* a_par, const *T2* b_par)

Quantile function of the Uniform distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9,  0.3,  0.87} };
stats::qunif(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.

Returns
a matrix of quantile values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qunif**(const BlazeMat<*eT*, *To*> &X, const *T1* a_par, const *T2* b_par)

Quantile function of the Uniform distribution.

Example:

```
stats::qunif(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qunif**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* a_par, const *T2* b_par)

Quantile function of the Uniform distribution.

Example:

```
stats::qunif(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Uniform distribution is achieved via the `uniform_real_distribution` class from the C++ standard library, imported from `<random>`.

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **runif**(const *T1* a_par, const *T2* b_par, rand_engine_t &engine)

Random sampling function for the Uniform distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::runif(3.0,2.0,engine);
```

Parameters

- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Uniform distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **runif**(const *T1* a_par, const *T2* b_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Uniform distribution.

Example:

```
stats::runif(3.0,2.0,1776);
```

Parameters

- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Uniform distribution.

template<typename **T** = double>

inline *T* **runif**()

Random sampling function for the Uniform distribution on the unit interval.

Example:

```
stats::runif();
```

Returns

a pseudo-random draw from the Uniform distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **runif**(const ullint_t n, const ullint_t k, const *T1* a_par, const *T2* b_par, rand_engine_t &engine)

Random matrix sampling function for the Uniform distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::runif<std::vector<double>>(5,4,-1.0,3.0,engine);
// Armadillo matrix
stats::runif<arma::mat>(5,4,-1.0,3.0,engine);
// Blaze dynamic matrix
stats::runif<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,-1.0,3.0,engine);
// Eigen dynamic matrix
stats::runif<Eigen::MatrixXd>(5,4,-1.0,3.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a matrix of pseudo-random draws from the Uniform distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>

inline *mT* **runif**(const ullint_t n, const ullint_t k, const *T1* a_par, const *T2* b_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Uniform distribution.

Example:

```
// std::vector
stats::runif<std::vector<double>>(5,4,-1.0,3.0);
// Armadillo matrix
stats::runif<arma::mat>(5,4,-1.0,3.0);
// Blaze dynamic matrix
stats::runif<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,-1.0,3.0);
// Eigen dynamic matrix
stats::runif<Eigen::MatrixXd>(5,4,-1.0,3.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **a_par** – the lower bound parameter, a real-valued input.
- **b_par** – the upper bound parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a matrix of pseudo-random draws from the Uniform distribution.

dunif	density function of the Uniform distribution
punif	distribution function of the Uniform distribution
qnorm	quantile function of the Uniform distribution
runif	random sampling function of the Uniform distribution

2.6.21 Weibull Distribution

Table of contents

Density Function

The density function of the Weibull distribution:

$$f(x; k, \theta) = \frac{k}{\theta} \left(\frac{x}{\theta} \right)^{k-1} \exp \left(- \left(\frac{x}{\theta} \right)^{k} \right) \times \mathbf{1}[x \geq 0]$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **dweibull**(const *T1* x, const *T2* shape_par, const *T3* scale_par, const bool log_form = false) noexcept

Density function of the Weibull distribution.

Example:

```
stats::dweibull(1.0,2.0,3.0,false);
```

Parameters

- **x** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
the density function evaluated at x.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **dweibull**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Density function of the Weibull distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::dweibull(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns
a vector of density function values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **dweibull**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Density function of the Weibull distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::dweibull(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **dweibull**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Density function of the Weibull distribution.

Example:

```
stats::dweibull(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

a matrix of density function values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **dweibull**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Density function of the Weibull distribution.

Example:

```
stats::dweibull(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

- **log_form** – return the log-density or the true form.

Returns
a matrix of density function values corresponding to the elements of `X`.

Cumulative Distribution Function

The cumulative distribution function of the Weibull distribution:

$$F(x; k, \theta) = \int_0^x f(z; k, \theta) dz = 1 - \exp\left(-\left(\frac{x}{\theta}\right)^k \times \mathbf{1}[x \geq 0]\right)$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **pweibull**(const *T1* x, const *T2* shape_par, const *T3* scale_par, const bool log_form = false) noexcept

Distribution function of the Weibull distribution.

Example:

```
stats::pweibull(1.0,2.0,3.0,false);
```

Parameters

- **x** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
the cumulative distribution function evaluated at `x`.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **pweibull**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Distribution function of the Weibull distribution.

Example:

```
std::vector<double> x = {1.8, 0.7, 4.2};
stats::pweibull(x,3.0,2.0,false);
```

Parameters

- **x** – a standard vector.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a vector of CDF values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **pweibull**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Distribution function of the Weibull distribution.

Example:

```
arma::mat X = { {1.8, 0.7, 4.2},
                {0.3, 5.3, 3.7} };
stats::pweibull(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns

a matrix of CDF values corresponding to the elements of X.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **pweibull**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Distribution function of the Weibull distribution.

Example:

```
stats::pweibull(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **pweibull**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* scale_par, const bool log_form = false)

Distribution function of the Weibull distribution.

Example:

```
stats::pweibull(X,3.0,2.0,false);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **log_form** – return the log-probability or the true form.

Returns
a matrix of CDF values corresponding to the elements of X.

Quantile Function

The quantile function of the Weibull distribution:

$$q(p; k, \theta) = \lambda \times (-\ln(1-p))^{1/k}$$

Methods for scalar input, as well as for vector/matrix input, are listed below.

Scalar Input

template<typename **T1**, typename **T2**, typename **T3**>
constexpr common_return_t<*T1*, *T2*, *T3*> **qweibull**(const *T1* p, const *T2* shape_par, const *T3* scale_par) noexcept

Quantile function of the Weibull distribution.

Example:

```
stats::qweibull(0.5,2.0,3.0);
```

Parameters

- **p** – a real-valued input.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns

the quantile function evaluated at p.

Vector/Matrix Input

STL Containers

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline std::vector<*rT*> **qweibull**(const std::vector<*eT*> &x, const *T1* shape_par, const *T2* scale_par)

Quantile function of the Weibull distribution.

Example:

```
std::vector<double> x = {0.3, 0.5, 0.9};
stats::qweibull(x,3.0,2.0);
```

Parameters

- **x** – a standard vector.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns

a vector of quantile values corresponding to the elements of x.

Armadillo

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>>
inline ArmaMat<*rT*> **qweibull**(const ArmaMat<*eT*> &X, const *T1* shape_par, const *T2* scale_par)

Quantile function of the Weibull distribution.

Example:

```
arma::mat X = { {0.2,  0.7,  0.1},
                {0.9,  0.3,  0.87} };
stats::qweibull(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Blaze

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, bool **To** = blaze::columnMajor>
inline BlazeMat<*rT*, *To*> **qweibull**(const BlazeMat<*eT*, *To*> &X, const *T1* shape_par, const *T2* scale_par)

Quantile function of the Weibull distribution.

Example:

```
stats::qweibull(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of `X`.

Eigen

template<typename **eT**, typename **T1**, typename **T2**, typename **rT** = common_return_t<*eT*, *T1*, *T2*>, int **iTr** = Eigen::Dynamic, int **iTc** = Eigen::Dynamic>
inline EigenMat<*rT*, *iTr*, *iTc*> **qweibull**(const EigenMat<*eT*, *iTr*, *iTc*> &X, const *T1* shape_par, const *T2* scale_par)

Quantile function of the Weibull distribution.

Example:

```
stats::qweibull(X,3.0,2.0);
```

Parameters

- **X** – a matrix of input values.
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.

Returns

a matrix of quantile values corresponding to the elements of X.

Random Sampling

Random sampling for the Weibull distribution is achieved via the inverse probability integral transform.

Scalar Output

1. Random number engines

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rweibull**(const *T1* shape_par, const *T2* scale_par, rand_engine_t &engine)

Random sampling function for the Weibull distribution.

Example:

```
stats::rand_engine_t engine(1776);
stats::rweibull(3.0,2.0,engine);
```

Parameters

- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns

a pseudo-random draw from the Weibull distribution.

2. Seed values

template<typename **T1**, typename **T2**>
inline common_return_t<*T1*, *T2*> **rweibull**(const *T1* shape_par, const *T2* scale_par, const ullint_t seed_val = std::random_device{}())

Random sampling function for the Weibull distribution.

Example:

```
stats::rweibull(3.0,2.0,1776);
```

Parameters

- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns

a pseudo-random draw from the Weibull distribution.

Vector/Matrix Output

1. Random number engines

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rweibull**(const ullint_t n, const ullint_t k, const *T1* shape_par, const *T2* scale_par, rand_engine_t &engine)

Random matrix sampling function for the Weibull distribution.

Example:

```
stats::rand_engine_t engine(1776);
// std::vector
stats::rweibull<std::vector<double>>(5,4,3.0,2.0,engine);
// Armadillo matrix
stats::rweibull<arma::mat>(5,4,3.0,2.0,engine);
// Blaze dynamic matrix
stats::rweibull<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0,
↪engine);
// Eigen dynamic matrix
stats::rweibull<Eigen::MatrixXd>(5,4,3.0,2.0,engine);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns

- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **engine** – a random engine, passed by reference.

Returns
a matrix of pseudo-random draws from the Weibull distribution.

2. Seed values

template<typename **mT**, typename **T1**, typename **T2**>
inline *mT* **rweibull**(const ullint_t n, const ullint_t k, const *T1* shape_par, const *T2* scale_par, const ullint_t seed_val = std::random_device{}())

Random matrix sampling function for the Weibull distribution.

Example:

```
// std::vector
stats::rweibull<std::vector<double>>(5,4,3.0,2.0);
// Armadillo matrix
stats::rweibull<arma::mat>(5,4,3.0,2.0);
// Blaze dynamic matrix
stats::rweibull<blaze::DynamicMatrix<double,blaze::columnMajor>>(5,4,3.0,2.0);
// Eigen dynamic matrix
stats::rweibull<Eigen::MatrixXd>(5,4,3.0,2.0);
```

Note: This function requires template instantiation; acceptable output types include: `std::vector`, with element type `float`, `double`, etc., as well as Armadillo, Blaze, and Eigen dense matrices.

Parameters

- **n** – the number of output rows
- **k** – the number of output columns
- **shape_par** – the shape parameter, a real-valued input.
- **scale_par** – the scale parameter, a real-valued input.
- **seed_val** – initialize the random engine with a non-negative integral-valued seed.

Returns
a matrix of pseudo-random draws from the Weibull distribution.

dweibull	density function of the Weibull distribution
pweibull	distribution function of the Weibull distribution
qweibull	quantile function of the Weibull distribution
rweibull	random sampling function of the Weibull distribution

2.6.22 Wishart Distribution

Table of contents

Density Function

The density function of the Wishart distribution:

$$f(\mathbf{X}; \mathbf{\Psi}, \nu) = \frac{1}{2^{\frac{\nu p}{2}} |\mathbf{\Psi}|^{\frac{\nu}{2}} \Gamma_p \left(\frac{\nu}{2}\right)} |\mathbf{X}|^{\frac{\nu - p - 1}{2}} \exp\left(-\frac{1}{2} \text{tr}(\mathbf{\Psi}^{-1} \mathbf{X})\right)$$

where Γ_p is the Multivariate Gamma function, $|\cdot|$ denotes the matrix determinant, and $\text{tr}(\cdot)$ denotes the matrix trace.

template<typename **mT**, typename **pT**, typename not_arma_mat<*mT*>::type* = nullptr>
inline return_t<*pT*> **dwish**(const *mT* &X, const *mT* &Psi_par, const *pT* nu_par, const bool log_form = false)

Density function of the Wishart distribution.

Parameters

- **X** – a positive semi-definite matrix.
- **Psi_par** – a positive semi-definite scale matrix.
- **nu_par** – the degrees of parameter, a real-valued input.
- **log_form** – return the log-density or the true form.

Returns

the density function evaluated at `X`.

Random Sampling

template<typename **mT**, typename **pT**, typename not_arma_mat<*mT*>::type* = nullptr>
inline *mT* **rwish**(const *mT* &Psi_par, const *pT* nu_par, rand_engine_t &engine, const bool pre_chol = false)

Random sampling function for the Wishart distribution.

Parameters

- **Psi_par** – a positive semi-definite scale matrix.
- **nu_par** – the degrees of parameter, a real-valued input.
- **engine** – a random engine, passed by reference.
- **pre_chol** – indicate whether `Psi_par` is passed in lower triangular (Cholesky) format.

Returns

a pseudo-random draw from the Wishart distribution.

dwish	density function of the Wishart distribution
rwish	random sampling function of the Wishart distribution

INDEX

D

P

Q

R