RcppDist Introduction

JB Duck-Mayr

2018-10-02

The Rcpp package provides a C++ library to make it easier to use C++ with R. R and Rcpp provide functions for a variety of statistical distributions. Several R packages make functions available to R for additional statistical distributions. However, to access these functions from C++ code, a costly call to the R functions must be made.[1]

RcppDist provides a C++ header-only library with functions for additional statistical distributions that can be called from C++ when writing code using Rcpp or RcppArmadillo. Functions are available that return NumericVectors as well as doubles, and for multivariate or matrix distributions, Armadillo vectors and matrices.

RcppDist provides functions for the following distributions:

- the four parameter beta distribution
- the location-scale t distribution
- the truncated normal distribution
- the truncated t distribution
- a truncated location-scale t distribution
- the triangle distribution
- the multivariate normal distribution*
- the multivariate t distribution*
- the Wishart distribution*
- the inverse Wishart distribution*

Distributions marked with an asterisk rely on RcppArmadillo.

After discussing how to include `RcppDist` headers into your code to use these functions, each function provided by the package is listed (such that you can see the function and argument names, as well as the return and argument types), with a brief explanation.

Including RccpDist in Your Code

RcppDist provides several header files you can include in your code. `RcppDist.h` includes all of them, as well as `Rcpp.h` or `RcppArmadillo.h` as appropriate (see "Use Rpp, or RcppArmadillo?" below). So, in any C++ file you need to use functions from RcppDist, you can simply use

```
#include <RcppDist.h>
```

which will also take care of Rcpp(Armadillo) headers for you. You can alternatively pull in only the header(s) you need; for example, if you only need functions for the four parameter beta distribution, you can just use

```
#include <4beta.h>
```

However, you'll then be responsible for pulling in Rcpp/RcppArmadillo headers as appropriate. The header names that correspond to the various distributions are as follows:

[1] Some of the R packages alluded to have written these functions in C++ (in some cases using Rcpp). However, these packages do not make such functions available as a header library for other package writers intending to make use of the functions in C++ code, thus the motivation for this package.

Distribution	Header
Four Parameter Beta	4beta.h
Location-Scale t	lst.h
Truncated Normal	truncnorm.h
Truncated t	trunct.h
Truncated Location-Scale t	trunclst.h
Triangle	triangular.h
Multivariate Normal	mvnorm.h
Multivariate t	mvt.h
Wishart and Inverse Wishart	wishart.h

Using Rcpp, or RcppArmadillo?

Including `RcppDist.h` by default will pull in the RcppArmadillo headers (and therefore the Rcpp headers), as well as the RcppDist headers. If you would prefer to use Rcpp but *not* RcppArmadillo (i.e. include the Rcpp headers but not the RcppArmadillo headers), include the line

```
#define RCPPDIST_DONT_USE_ARMA
```

before any inclusion of `RcppDist.h`, though this will make the asterisked (multivariate and Wishart) distributions unavailable.

Using RcppDist in a Package

With Rcpp

To use RcppDist in a package that does not link to RcppArmadillo, you must

- Set up your package to use `Rcpp`, such as via `Rcpp::Rcpp.package.skeleton(your_package)` or `devtools::use_rcpp(your_package)`.[2]
- Add RcppDist to the LinkingTo field of your DESCRIPTION file.
- In any C++ file that calls a `RcppDist` function, add `#include <RcppDist.h>` (or a more specific header from the package such as `lst.h` and `Rcpp.h`)
- Remember to define `RCPPDIST_DONT_USE_ARMA` before any include of `RcppDist.h`.

With RcppArmadillo

To use RcppDist in a package that links to RcppArmadillo, you must

- Set up your package to `RcppArmadillo`, such as via `RcppArmadillo::RcppArmadillo.package.skeleton(your_package)`[3]
- Add RcppDist to the LinkingTo field of your DESCRIPTION file.
- In any C++ file that calls a `RcppDist` function, add `#include <RcppDist.h>` (or a more specific header from the package such as `mvt.h` and `RcppArmadillo.h`)

[2] See the Rcpp-package vignette from Rcpp for more details.

[3] See the RcppArmadillo manual or help files for more details, in particular the 'RcppArmadillo-package' and 'RcppArmadillo.package.skeleton' entries.

Using RcppDist in a Standalone file

With Rcpp

If you are using RcppDist in a standalone file (i.e., not as part of a package), and you don't want to pull in the Armadillo headers, you'll need

```
#define RCPPDIST_DONT_USE_ARMA
#include <RcppDist.h>
// [[Rcpp::depends(RcppDist)]]
```

at the top of your file. If you want to pull in just one or more of the distribution specific headers, you won't need the define, but don't forget to also include `Rcpp.h`; as an example:

```
#include <Rcpp.h>
#include <triangular.h>
// [[Rcpp::depends(RcppDist)]]
```

With RcppArmadillo

If you are using RcppDist in a standalone file (i.e., not as part of a package), and you do want the Armadillo headers, you'll need

```
#include <RcppDist.h>
// [[Rcpp::depends(RcppArmadillo, RcppDist)]]
```

at the top of your file. If you want to pull in just one or more of the distribution specific headers, don't forget to also include `RcppArmadillo.h`; as an example:

```
#include <RcppArmadillo.h>
#include <mvnorm.h>
// [[Rcpp::depends(RcppArmadillo, RcppDist)]]
```

RcppDist Functions

Much like distributions in R, functions are prefixed by d, p, q, and r to mean density, distribution, quantile, and random number generating functions respectively. Functions that are return double rather than, say, a NumericVector are instead prefixed by d_, p_, q_, and r_. Below are more detailed descriptions of the functions provided by each header.

4beta.h (Four Parameter Beta Distribution)

The four parameter beta distribution is a beta distribution supported over an interval $[a, b]$ rather than only $[0, 1]$. The functions provided in this header are:

```
Rcpp::NumericVector d4beta(const Rcpp::NumericVector& x,
        const double shape1, const double shape2, const double a,
        const double b, const bool log_p = false)

Rcpp::NumericVector p4beta(const Rcpp::NumericVector& q,
        const double shape1, const double shape2, const double a,
        const double b, const bool lower_tail = true,
        const bool log_p = false)
```

```
Rcpp::NumericVector q4beta(const Rcpp::NumericVector& p,
        const double shape1, const double shape2, const double a,
        const double b, const bool lower_tail = true,
        const bool log_p = false)

Rcpp::NumericVector r4beta(const int n, const double shape1,
        const double shape2, const double a, const double b)

double d_4beta(const double x, const double shape1, const double shape2,
        const double a, const double b, const int log_p = 0)

double p_4beta(const double q, const double shape1, const double shape2,
        const double a, const double b, const int lower_tail = 1,
        const int log_p = 0)

double q_4beta(const double p, const double shape1, const double shape2,
        const double a, const double b, const int lower_tail = 1,
        const int log_p = 0)

double r_4beta(const double shape1, const double shape2, const double a,
        const double b)
```

Where

- `x` and `q` are quantiles (either a single value or a vector depending)
- `p` is a single probability or a vector of probabilities
- `n` is the numer of observations to draw
- `shape1` and `shape2` are the positive shape parameters of the Beta distribution
- `a` and `b` are the minimum and maximum values of the distribution respectively
- `log_p` is a `bool` or `int` (the default is `false`/0); if `true` (or > 0), the probabilities are given as $\log(p)$
- `lower_tail` is a `bool` or `int`; if `true` (or > 0), the probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

lst.h (Location-Scale t Distribution)

The location-scale t distribution is a t distribution shifted by a location parameter μ and scaled by a scaling parameter σ. The functions provided in this header are:

```
Rcpp::NumericVector dlst(const Rcpp::NumericVector& x, const double df,
        const double mu, const double sigma, const bool log_p = false)

Rcpp::NumericVector plst(const Rcpp::NumericVector& q, const double df,
        const double mu, const double sigma, const bool lower_tail = true,
        const bool log_p = false)

Rcpp::NumericVector qlst(const Rcpp::NumericVector& p, const double df,
        const double mu, const double sigma, const bool lower_tail = true,
        const bool log_p = false)

Rcpp::NumericVector rlst(const int n, const double df, const double mu,
        const double sigma)

double d_lst(const double x, const double df, const double mu,
        const double sigma, const int log_p = 0)
```

```
double p_lst(const double q, const double df, const double mu,
        const double sigma, const int lower_tail = 1, const int log_p = 0)

double q_lst(const double p, const double df, const double mu,
        const double sigma, const int lower_tail = 1, const int log_p = 0)

double r_lst(const double df, const double mu, const double sigma)
```

Where

- `x` and `q` are quantiles (either a single value or a vector depending)
- `p` is a single probability or a vector of probabilities
- `n` is the numer of observations to draw
- `df` is the positive degrees of freedom
- `mu` is the location/shifting parameter
- `sigma` is the scaling parameter
- `log_p` is a `bool` or `int` (the default is `false`/0); if `true` (or > 0), the probabilities are given as $\log(p)$
- `lower_tail` is a `bool` or `int`; if `true` (or > 0), the probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

truncnorm.h (Truncated Normal Distribution)

The truncated normal distribution is a normal distribution supported over an interval $[a, b]$ rather than $(-\infty, \infty)$. The functions provided in this header are:

```
Rcpp::NumericVector dtruncnorm(const Rcpp::NumericVector& x,
        const double mu, const double sigma, const double a, const double b,
        const bool log_p = false)

Rcpp::NumericVector ptruncnorm(const Rcpp::NumericVector& x,
        const double mu, const double sigma, const double a, const double b,
        const bool lower_tail = true, const bool log_p = false)

Rcpp::NumericVector qtruncnorm(const Rcpp::NumericVector& p,
        const double mu, const double sigma, const double a, const double b,
        const bool lower_tail = true, const bool log_p = false)

Rcpp::NumericVector rtruncnorm(const int n, const double mu,
        const double sigma, const double a, const double b)

double d_truncnorm(const double x, const double mu, const double sigma,
        const double a, const double b, const int log_p = 0)

double p_truncnorm(const double x, const double mu, const double sigma,
        const double a, const double b, const int lower_tail = 1,
        const int log_p = 0)

double q_truncnorm(const double p, const double mu, const double sigma,
        const double a, const double b, const int lower_tail = 1,
        const int log_p = 0)

double r_truncnorm(const double mu, const double sigma, const double a,
        const double b)
```

Where

- `x` and `q` are quantiles (either a single value or a vector depending)
- `p` is a single probability or a vector of probabilities
- `n` is the numer of observations to draw
- `mu` is the mean of the distribution
- `sigma` is the standard deviation
- `a` and `b` are the minimum and maximum values of the distribution respectively
- `log_p` is a `bool` or `int` (the default is `false`/0); if `true` (or > 0), the probabilities are given as $\log(p)$
- `lower_tail` is a `bool` or `int`; if `true` (or > 0), the probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

trunct.h (Truncated t Distribution)

The truncated t distribution is a t distribution supported over an interval $[a, b]$ rather than $(-\infty, \infty)$. The functions provided in this header are:

```
Rcpp::NumericVector dtrunct(const Rcpp::NumericVector& x,
        const double df, const double a, const double b,
        const bool log_p = false)

Rcpp::NumericVector ptrunct(const Rcpp::NumericVector& x,
        const double df, const double a, const double b,
        const bool lower_tail = true, const bool log_p = false)

Rcpp::NumericVector qtrunct(const Rcpp::NumericVector& p,
        const double df, const double a, const double b,
        const bool lower_tail = true, const bool log_p = false)

Rcpp::NumericVector rtrunct(const int n, const double df,
        const double a, const double b)

double d_trunct(const double x, const double df, const double a,
        const double b, const int log_p = 0)

double p_trunct(const double x, const double df, const double a,
        const double b, const int lower_tail = 1, const int log_p = 0)

double q_trunct(const double p, const double df, const double a,
        const double b, const int lower_tail = 1, const int log_p = 0)

double r_trunct(const double df, const double a, const double b)
```

Where

- `x` and `q` are quantiles (either a single value or a vector depending)
- `p` is a single probability or a vector of probabilities
- `n` is the numer of observations to draw
- `df` is the positive degrees of freedom
- `a` and `b` are the minimum and maximum values of the distribution respectively
- `log_p` is a `bool` or `int` (the default is `false`/0); if `true` (or > 0), the probabilities are given as $\log(p)$
- `lower_tail` is a `bool` or `int`; if `true` (or > 0), the probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

trunclst.h (Truncated Location-Scale t Distribution)

The truncated location-scale t distribution is a location-scale t distribution supported over an interval $[a, b]$ rather than $(-\infty, \infty)$. The functions provided in this header are:

```
Rcpp::NumericVector dtrunclst(const Rcpp::NumericVector& x,
        const double df, const double mu, const double sigma, const double a,
        const double b, const bool log_p = false)

Rcpp::NumericVector ptrunclst(const Rcpp::NumericVector& x,
        const double df, const double mu, const double sigma, const double a,
        const double b, const bool lower_tail = true,
        const bool log_p = false)

Rcpp::NumericVector qtrunclst(const Rcpp::NumericVector& p,
        const double df, const double mu, const double sigma, const double a,
        const double b, const bool lower_tail = true,
        const bool log_p = false)

Rcpp::NumericVector rtrunclst(const int n, const double df,
        const double mu, const double sigma, const double a, const double b)

double d_trunclst(const double x, const double df, const double mu,
        const double sigma, const double a, const double b,
        const int log_p = 0)

double p_trunclst(const double x, const double df, const double mu,
        const double sigma, const double a, const double b,
        const int lower_tail = 1, const int log_p = 0)

double q_trunclst(const double p, const double df, const double mu,
        const double sigma, const double a, const double b,
        const int lower_tail = 1, const int log_p = 0)

double r_trunclst(const double df, const double mu, const double sigma,
        const double a, const double b)
```

Where

- `x` and `q` are quantiles (either a single value or a vector depending)
- `p` is a single probability or a vector of probabilities
- `n` is the numer of observations to draw
- `df` is the positive degrees of freedom
- `mu` is the location/shifting parameter
- `sigma` is the scaling parameter
- `a` and `b` are the minimum and maximum values of the distribution respectively
- `log_p` is a `bool` or `int` (the default is `false`/0); if `true` (or > 0), the probabilities are given as $\log(p)$
- `lower_tail` is a `bool` or `int`; if `true` (or > 0), the probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

triangular.h (Triangle Distribution)

The triangle (or triangular) distribution is supported over an interval $[a, b]$ with a mode c; as the name suggests, the density function is shaped like a triangle with vertices at a, b, and c. The functions provided in this header are:

```
double d_tri(const double x, const double a, const double b,
        const double c, const int log_p = 0)

double p_tri(const double x, const double a, const double b,
```

```
        const double c, const int lower_tail = 1, const int log_p = 0)

double q_tri(const double p, const double a, const double b,
        const double c, const int lower_tail = 1, const int log_p = 0)

double r_tri(const double a, const double b, const double c)

Rcpp::NumericVector dtri(const Rcpp::NumericVector& x, const double a,
        const double b, const double c, const bool log_p = false)

Rcpp::NumericVector ptri(const Rcpp::NumericVector& x, const double a,
        const double b, const double c, const bool lower_tail = true,
        const bool log_p = false)

Rcpp::NumericVector qtri(const Rcpp::NumericVector& p, const double a,
        const double b, const double c, const bool lower_tail = true,
        const bool log_p = false)

Rcpp::NumericVector rtri(const int n, const double a, const double b,
        const double c)
```

Where

- `x` and `q` are quantiles (either a single value or a vector depending)
- `p` is a single probability or a vector of probabilities
- `n` is the numer of observations to draw
- `a` and `b` are the minimum and maximum values of the distribution respectively
- `c` is the mode of the distribution
- `log_p` is a `bool` or `int` (the default is `false`/0); if `true` (or > 0), the probabilities are given as $\log(p)$
- `lower_tail` is a `bool` or `int`; if `true` (or > 0), the probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

mvnorm.h (Multivariate Normal Distribution)

The multivariate normal distribution is a generalization of the normal distribution to multiple dimensions. Then each draw is a vector, the mean parameter μ is a vector, and rather than a scalar standard deviation parameter σ, we have a covariance matrix Σ (or here denoted S). The functions provided in this header are:

```
arma::vec dmvnorm(const arma::mat& x, const arma::vec& mu,
        const arma::mat& S, const bool log_p = false)

arma::mat rmvnorm(const arma::uword n, const arma::vec& mu,
        const arma::mat& S)
```

Where

- `x` is a matrix of quantiles, such that each row is a quantile
- `n` is the numer of observations to draw
- `mu` is the mean vector
- `S` is the covariance matrix
- `log_p` is a `bool` (the default is `false`); if `true`, the probabilities are given as $\log(p)$

mvt.h (Multivariate t Distribution)

The multivariate t distribution is a generalization of the t distribution to multiple dimensions. Then each draw is a vector, and in addition to the degrees of freedom, we have a correlation matrix Σ (or here denoted S), and this implementation allows for a location vector μ. The functions provided in this header are:

```
arma::vec dmvt(const arma::mat& x, const arma::vec& mu,
        const arma::mat& S, const double df, const bool log_p = false)

arma::mat rmvt(const arma::uword n, const arma::vec& mu,
        const arma::mat& S, const double df)
```

Where

- `x` is a matrix of quantiles, such that each row is a quantile
- `n` is the numer of observations to draw
- `mu` is the location vector
- `S` is the correlation matrix
- `log_p` is a `bool` (the default is `false`); if `true`, the probabilities are given as $\log(p)$

wishart.h (Wishart and Inverse Wishart Distributions)

The Wishart distribution is a generalization of the gamma distribution to multiple dimensions defined over symmetric, nonnegative-definite random matrices. Its parameters are the degrees of freedom and a scale matrix S. If $X \sim$ Wishart(df, S), then $X^{-1} \sim$ Inverse Wishart(df, S^{-1}). Due to their use in the density functions for these distributions, a multivariate gamma function and logged multivariate gamma function are also provided. Note that for now, al functions for this distribution are designed to deal with only one random matrix. The functions provided in this header are:

```
double mvgamma(const int p, const double x)

double lmvgamma(const int p, const double x)

double dwish(const arma::mat& X, const int df, const arma::mat& S,
    const bool log_p = false)

arma::mat rwish(const int df, const arma::mat& S)

double diwish(const arma::mat& X, const int df, const arma::mat& S,
    const bool log_p = false)

arma::mat riwish(const int df, const arma::mat& S)
```

Where

- `p` and `x` are the arguments to the multivariate gamma function
- `X` is a matrix, a draw from the Wishart or Inverse Wishart distribution
- `df` is the degrees of feedom
- `S` is the scale matrix
- `log_p` is a `bool` (the default is `false`); if `true`, the probabilities are given as $\log(p)$