

# Package: fntl (via r-universe)

September 25, 2026

**Title** Numerical Tools for 'Rcpp' and Lambda Functions

**Version** 0.1.3.2

**Description** Provides a 'C++' API for routinely used numerical tools such as integration, root-finding, and optimization, where function arguments are given as lambdas. This facilitates 'Rcpp' programming, enabling the development of 'R'-like code in 'C++' where functions can be defined on the fly and use variables in the surrounding environment.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**URL** <https://github.com/andrewraim/fntl>

**Depends** R (>= 4.3)

**Suggests** knitr, Matrix, numDeriv, rmarkdown, quarto, RcppArmadillo, tidyverse, tinytest

**LinkingTo** Rcpp

**Imports** Rcpp

**VignetteBuilder** quarto

**Config/roxygen2/version** 8.1.0

**Repository** <https://andrewraim.r-universe.dev>

**Date/Publication** 2026-08-26 12:57:02 UTC

**RemoteUrl** <https://github.com/andrewraim/fntl>

**RemoteRef** HEAD

**RemoteSha** a17f0c99bf498f86317f866e125b67fb8265ca88

## Contents

|                        |   |
|------------------------|---|
| fntl-package . . . . . | 2 |
| args . . . . .         | 2 |
| deriv . . . . .        | 3 |

2

args

findroot . . . . .

gradient0 . . . . .

hessian0 . . . . .

integrate0 . . . . .

jacobian0 . . . . .

matrix\_apply . . . . .

multivariate-optimization . . . . .

outer . . . . .

solve\_cg . . . . .

trunc . . . . .

univariate-optimization . . . . .

which0 . . . . .

Index

19

---

|              |             |
|--------------|-------------|
| fntl-package | <i>fntl</i> |
|--------------|-------------|

---

Description

Package documentation

Author(s)

**Maintainer:** Andrew M. Raim <andrew.raim@gmail.com> ([ORCID](#))  
**Authors:**  

- Andrew M. Raim <andrew.raim@gmail.com> ([ORCID](#))

See Also

Useful links:  

- <https://github.com/andrewraim/fntl>

---

|      |                  |
|------|------------------|
| args | <i>Arguments</i> |
|------|------------------|

---

Description

Get an arguments list for internal methods with the default settings. This object can be adjusted and passed to the respective function.

**Usage**

```
findroot_args()

optimize_args()

integrate_args()

cg_args()

bfgs_args()

lbfgsb_args()

neldermead_args()

nlm_args()

richardson_args()
```

**Value**

An argument list corresponding to the specified function. The elements of the list are named and supplied with default values. See the package vignette for further details.

- `findroot_args` is documented in the section "Root-Finding".
- `optimize_args` is documented in the section "Univariate Optimization".
- `integrate_args` is documented in the section "Integration".
- `cg_args` is documented in the section "Conjugate Gradient".
- `bfgs_args` is documented in the section "BFGS".
- `lbfgsb_args` is documented in the section "L-BFGS-B".
- `neldermead_args` is documented in the section "Nelder-Mead".
- `nlm_args` is documented in the section "Newton-Type Algorithm for Nonlinear Optimization".
- `richardson_args` is documented in the section "Richardson Extrapolated Finite Differences".

---

`deriv`*Numerical Derivatives via Finite Differences*

---

**Description**

Numerical Derivatives via Finite Differences

**Usage**

```
fd_deriv1(f, x, i, h, fd_type)

fd_deriv2(f, x, i, j, h_i, h_j, fd_type)

deriv1(f, x, i, args, fd_type)

deriv2(f, x, i, j, args, fd_type)
```

**Arguments**

|         |                                                                                                          |
|---------|----------------------------------------------------------------------------------------------------------|
| f       | Function to differentiate.                                                                               |
| x       | Scalar at which to evaluate the derivative.                                                              |
| i       | First coordinate to differentiate.                                                                       |
| h       | Step size in the first coordinate.                                                                       |
| fd_type | Type of derivative: 0 for symmetric difference, 1 for forward difference, and 2 for backward difference. |
| j       | Second coordinate to differentiate.                                                                      |
| h_i     | Step size in the first coordinate.                                                                       |
| h_j     | Step size in the second coordinate.                                                                      |
| args    | List of additional arguments from the function richardson_args.                                          |

**Value**

fd\_deriv1 and fd\_deriv2 return a single numeric value corresponding to the first and second derivative via finite differences. deriv1 and deriv2 return a list with the form of a richardson\_result described in section "Richardson Extrapolated Finite Differences" of the package vignette.

**Examples**

```
args = richardson_args()

f = sin # Try 2nd derivatives of a univariate function
x0 = 0.5
print(-sin(x0)) ## Exact answer for f''(x0)

fd_deriv2(f, x0, i = 0, j = 0, h_i = 0.001, h_j = 0.001, fd_type = 0)
fd_deriv2(f, x0, i = 0, j = 0, h_i = 0.001, h_j = 0.001, fd_type = 1)
fd_deriv2(f, x0, i = 0, j = 0, h_i = 0.001, h_j = 0.001, fd_type = 2)

deriv2(f, x0, i = 0, j = 0, args, fd_type = 0)

# Try 2nd derivatives of a bivariate function
f = function(x) { sin(x[1]) + cos(x[2]) }
x0 = c(0.5, 0.25)

print(-sin(x0[1])) ## Exact answer for f_xx(x0)
print(-cos(x0[2])) ## Exact answer for f_yy(x0)
```

```

print(0)          ## Exact answer for f_xy(x0,y0)

numDeriv::hessian(f, x0)

fd_deriv2(f, x0, i = 0, j = 0, h_i = 0.001, h_j = 0.001, fd_type = 0)
fd_deriv2(f, x0, i = 0, j = 0, h_i = 0.001, h_j = 0.001, fd_type = 1)
fd_deriv2(f, x0, i = 0, j = 0, h_i = 0.001, h_j = 0.001, fd_type = 2)

fd_deriv2(f, x0, i = 0, j = 1, h_i = 0.001, h_j = 0.001, fd_type = 0)
fd_deriv2(f, x0, i = 0, j = 1, h_i = 0.001, h_j = 0.001, fd_type = 1)
fd_deriv2(f, x0, i = 0, j = 1, h_i = 0.001, h_j = 0.001, fd_type = 2)

fd_deriv2(f, x0, i = 1, j = 1, h_i = 0.001, h_j = 0.001, fd_type = 0)
fd_deriv2(f, x0, i = 1, j = 1, h_i = 0.001, h_j = 0.001, fd_type = 1)
fd_deriv2(f, x0, i = 1, j = 1, h_i = 0.001, h_j = 0.001, fd_type = 2)

deriv2(f, x0, i = 1, j = 1, args, fd_type = 0)
deriv2(f, x0, i = 1, j = 1, args, fd_type = 1)
deriv2(f, x0, i = 1, j = 1, args, fd_type = 2)

```

---

findroot

*Find Root*


---

## Description

Find Root

## Usage

```
findroot_bisect(f, lower, upper, args)
```

```
findroot_brent(f, lower, upper, args)
```

## Arguments

|       |                                                               |
|-------|---------------------------------------------------------------|
| f     | Function for which a root is desired.                         |
| lower | Lower limit of search interval. Must be finite.               |
| upper | Upper limit of search interval. Must be finite.               |
| args  | List of additional arguments from the function findroot_args. |

## Value

A list with the form of a findroot\_result described in section "Root-Finding" of the package vignette.

**Examples**

```
f = function(x) { x^2 - 1 }
args = findroot_args()
findroot_bisect(f, 0, 10, args)
findroot_brent(f, 0, 10, args)
```

---

gradient0

*Numerical Gradient Vector*


---

**Description**

Numerical Gradient Vector

**Usage**

```
gradient0(f, x, args)
```

**Arguments**

|      |                                                                 |
|------|-----------------------------------------------------------------|
| f    | Function to differentiate.                                      |
| x    | Vector at which to evaluate the gradient.                       |
| args | List of additional arguments from the function richardson_args. |

**Value**

A list with the form of a `gradient_result` described in section "Gradient" of the package vignette.

**Examples**

```
f = function(x) { sum(sin(x)) }
args = richardson_args()
x0 = seq(0, 1, length.out = 5)
cos(x0) ## Exact answer
gradient0(f, x0, args)
numDeriv::grad(f, x0)
```

---

|          |                          |
|----------|--------------------------|
| hessian0 | <i>Numerical Hessian</i> |
|----------|--------------------------|

---

**Description**

Numerical Hessian

**Usage**

```
hessian0(f, x, args)
```

**Arguments**

|      |                                                                 |
|------|-----------------------------------------------------------------|
| f    | Function to differentiate.                                      |
| x    | Vector at which to evaluate the Hessian.                        |
| args | List of additional arguments from the function richardson_args. |

**Value**

A list with the form of a hessian\_result described in section "Hessian" of the package vignette.

**Examples**

```
f = function(x) { sum(x^2) }
x0 = seq(1, 10, length.out = 5)
args = richardson_args()
hessian0(f, x0, args)
numDeriv::hessian(f, x0)
```

---

|            |                    |
|------------|--------------------|
| integrate0 | <i>Integration</i> |
|------------|--------------------|

---

**Description**

Compute the integral  $\int_a^b f(x)dx$ .

**Usage**

```
integrate0(f, lower, upper, args)
```

**Arguments**

|       |                                                                |
|-------|----------------------------------------------------------------|
| f     | Function to integrate.                                         |
| lower | Lower limit of integral.                                       |
| upper | Upper limit of integral.                                       |
| args  | List of additional arguments from the function integrate_args. |

**Value**

A list with the form of a `integrate_result` described in section "Integration" of the package vignette.

**Examples**

```
f = function(x) { exp(-x^2 / 2) }
args = integrate_args()
integrate0(f, 0, 10, args)
```

---

jacobian0

*Numerical Jacobian Matrix*


---

**Description**

Numerical Jacobian Matrix

**Usage**

```
jacobian0(f, x, args)
```

**Arguments**

|      |                                                                               |
|------|-------------------------------------------------------------------------------|
| f    | Function to differentiate.                                                    |
| x    | Vector at which to evaluate the Jacobian.                                     |
| args | List of additional arguments from the function <code>richardson_args</code> . |

**Value**

A list with the form of a `jacobian_result` described in section "Jacobian" of the package vignette.

**Examples**

```
f = function(x) { cumsum(sin(x)) }
x0 = seq(1, 10, length.out = 5)
args = richardson_args()
out = jacobian0(f, x0, args)
print(out$value)
numDeriv::jacobian(f, x0)
```

**Description**

Matrix Apply Functions

**Usage**

```
mat_apply(X, f)
```

```
row_apply(X, f)
```

```
col_apply(X, f)
```

**Arguments**

|   |                        |
|---|------------------------|
| X | A matrix               |
| f | The function to apply. |

**Details**

The `mat_apply`, `row_apply`, and `col_apply` C++ functions are intended to operate like the following calls in R, respectively.

```
apply(x, c(1,2), f)
apply(x, 1, f)
apply(x, 2, f)
```

The R functions exposed here are specific to numeric-valued matrices, but the underlying C++ functions are intended to work with any type of Rcpp Matrix.

**Value**

`mat_apply` returns a matrix. `row_apply` and `col_apply` return a vector. See section "Apply" of the package vignette for details.

**Examples**

```
X = matrix(1:12, nrow = 4, ncol = 3)
mat_apply(X, f = function(x) { x^(1/3) })
row_apply(X, f = function(x) { sum(x^2) })
col_apply(X, f = function(x) { sum(x^2) })
```

---

`multivariate-optimization`*Multivariate Optimization*

---

**Description**

Multivariate Optimization

**Usage**`cg1(init, f, g, args)``cg2(init, f, args)``bfgs1(init, f, g, args)``bfgs2(init, f, args)``lbfgsb1(init, f, g, args)``lbfgsb2(init, f, args)``neldermead(init, f, args)``nlm1(init, f, g, h, args)``nlm2(init, f, g, args)``nlm3(init, f, args)`**Arguments**

|                   |                                                |
|-------------------|------------------------------------------------|
| <code>init</code> | Initial value                                  |
| <code>f</code>    | Function $f$ to optimize                       |
| <code>g</code>    | Gradient function of $f$ .                     |
| <code>args</code> | List of additional arguments for optimization. |
| <code>h</code>    | Hessian function of $f$ .                      |

**Details**

The argument `args` should be a list constructed from one of the following functions:

- `bfgs_args` for BFGS;
- `lbfgsb_args` for L-BFGS-B;
- `cg_args` for CG;

- `neldermead_args` for Nelder-Mead;
- `nlm_args` for the Newton-type algorithm used in `nlm`.

When `g` or `h` are omitted, the gradient or Hessian will be respectively be computed via finite differences.

## Value

A list with results corresponding to the specified function. See the package vignette for further details.

- `cg1` and `cg2` return a `cg_result` which is documented in the section "Conjugate Gradient".
- `bfgs1` and `bfgs2` return a `bfgs_result` which is documented in the section "BFGS".
- `lbfgsb1` and `lbfgsb2` return a `lbfgsb_result` which is documented in the section "L-BFGS-B".
- `neldermead` returns a `neldermead_result` which is documented in the section "Nelder-Mead".
- `nlm1`, `nlm2`, and `nlm3` return a `nlm_result` which is documented in the section "Newton-Type Algorithm for Nonlinear Optimization".

## Examples

```
f = function(x) { sum(x^2) }
g = function(x) { 2*x }
h = function(x) { 2*diag(length(x)) }
x0 = c(1,1)
```

```
args = cg_args()
cg1(x0, f, g, args)
cg2(x0, f, args)
```

```
args = bfgs_args()
bfgs1(x0, f, g, args)
bfgs2(x0, f, args)
```

```
args = lbfgsb_args()
lbfgsb1(x0, f, g, args)
lbfgsb2(x0, f, args)
```

```
args = neldermead_args()
neldermead(x0, f, args)
```

```
args = nlm_args()
nlm1(x0, f, g, h, args)
nlm2(x0, f, g, args)
nlm3(x0, f, args)
```

outer

*Outer Matrix***Description**

Compute "outer" matrices and matrix-vector products based on a function that operators on pairs of rows. See details.

**Usage**

```
outer1(X, f)
```

```
outer2(X, Y, f)
```

```
outer1_matvec(X, f, a)
```

```
outer2_matvec(X, Y, f, a)
```

```
outer1_triplet(X, f, g)
```

```
outer2_triplet(X, Y, f, g)
```

**Arguments**

|     |                                                                                                                                                                                                                            |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $X$ | A numerical matrix.                                                                                                                                                                                                        |
| $f$ | Function $f(x, y)$ that operates on a pair of rows and returns an element for the result.                                                                                                                                  |
| $Y$ | A numerical matrix.                                                                                                                                                                                                        |
| $a$ | A scalar vector.                                                                                                                                                                                                           |
| $g$ | Function $g(x, y)$ that operates on a pair of rows and returns a logical value; if TRUE, the corresponding output of $f$ is placed into the result. Otherwise, the corresponding output of $f$ is omitted from the result. |

**Details**

Functions  $f(x, y)$  and  $g(x, y)$  operate on pairs of rows. In the functions with prefix `outer1`,  $x$  and  $y$  are pairs of rows of  $X$ . In the functions with prefix `outer2`,  $x$  is a row from  $X$  and  $y$  is a row from  $Y$ .

The `outer1` function computes the  $n \times n$  symmetric matrix

$$\text{outer1}(X, f) = \begin{bmatrix} f(x_1, x_1) & \cdots & f(x_1, x_n) \\ \vdots & \ddots & \vdots \\ f(x_n, x_1) & \cdots & f(x_n, x_n) \end{bmatrix}$$

and the `outer1_matvec` operation computes the  $n$ -dimensional vector

$$\text{outer1\_matvec}(X, f, a) = \begin{bmatrix} f(x_1, x_1) & \cdots & f(x_1, x_n) \\ \vdots & \ddots & \vdots \\ f(x_n, x_1) & \cdots & f(x_n, x_n) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}.$$

The `outer2` operation computes the  $m \times n$  matrix

$$\text{outer2}(X, Y, f) = \begin{bmatrix} f(x_1, y_1) & \cdots & f(x_1, y_n) \\ \vdots & \ddots & \vdots \\ f(x_m, y_1) & \cdots & f(x_m, y_n) \end{bmatrix}$$

and the `outer2_matvec` operation computes the  $m$ -dimensional vector

$$\text{outer2\_matvec}(X, Y, f, a) = \begin{bmatrix} f(x_1, y_1) & \cdots & f(x_1, y_n) \\ \vdots & \ddots & \vdots \\ f(x_m, y_1) & \cdots & f(x_m, y_n) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}.$$

The `outer_triplet1` and `outer_triplet2` operations are analogous to `outer1` and `outer2`, respectively, except return sparse matrices in triplet form. The triplet variants take an additional argument `g`, a function `$g(x_i, y_j)$` that returns TRUE if the result is to be stored in the result and FALSE otherwise. The argument `g` can be used for sparseness.

## Value

`outer1` and `outer2` return a matrix. `outer1_matvec` and `outer2_matvec` return a vector. `outer1_triplet` and `outer2_triplet` return a list with a sparse matrix (triplet) representation. See section "Outer" of the package vignette for details.

## Examples

```
set.seed(1234)
f = function(x,y) { sum( (x - y)^2 ) }
X = matrix(rnorm(12), 6, 2)
Y = matrix(rnorm(10), 5, 2)
outer1(X, f)
outer2(X, Y, f)

a = rep(1, 6)
b = rep(1, 5)
outer1_matvec(X, f, a)
outer2_matvec(X, Y, f, b)
```

---

solve\_cg

*Iteratively Solve a Linear System with Conjugate Gradient*


---

### Description

Solve the system  $l(x) = b$  where  $l(x)$  is a matrix-free representation of the linear operation  $Ax$ .

### Usage

```
solve_cg(l, b, init, args)
```

### Arguments

|                   |                                                          |
|-------------------|----------------------------------------------------------|
| <code>l</code>    | A linear transformation of $x$ .                         |
| <code>b</code>    | A vector.                                                |
| <code>init</code> | Initial value of solution.                               |
| <code>args</code> | List of additional arguments from <code>cg_args</code> . |

### Value

A list with the form of a `solve_cg_result` described in section "Conjugate Gradient" of the package vignette.

### Examples

```
set.seed(1234)

n = 8
idx_diag = cbind(1:n, 1:n)
idx_ldiag = cbind(2:n, 1:(n-1))
idx_udiag = cbind(1:(n-1), 2:n)
b = rep(1, n)

## Solution by explicit computation of solve(A, b)
A = matrix(0, n, n)
A[idx_diag] = 2
A[idx_ldiag] = 1
A[idx_udiag] = 1
solve(A, b)

## Solve iteratively with solve_cg
f = function(x) { A %*% x }
args = cg_args()
init = rep(0, n)
solve_cg(f, b, init, args)
```

trunc

*Functions for Truncated Distributions***Description**

Density, CDF, quantile, and drawing functions for a univariate distribution with density  $f$ , cdf,  $F$ , and quantile function  $F^{-1}$  truncated to the interval  $[a, b]$ .

**Usage**

```
d_trunc(x, lo, hi, f, F, log = FALSE)

p_trunc(x, lo, hi, F, lower = TRUE, log = FALSE)

q_trunc(p, lo, hi, F, Finv, lower = TRUE, log = FALSE)

r_trunc(n, lo, hi, F, Finv)
```

**Arguments**

|                    |                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>     | Vector of quantiles.                                                                                                                                  |
| <code>lo</code>    | Vector of lower limits.                                                                                                                               |
| <code>hi</code>    | Vector of upper limits.                                                                                                                               |
| <code>f</code>     | Density function with form $f(x, \log)$ .                                                                                                             |
| <code>F</code>     | CDF function with signature $F(x, \text{lower}, \log)$ , where $x$ is numeric and <code>lower</code> and <code>log</code> are logical.                |
| <code>log</code>   | logical; if TRUE, probabilities are given on log-scale.                                                                                               |
| <code>lower</code> | logical; if TRUE, probabilities are $P(X \leq x)$ ; otherwise, $P(X > x)$ .                                                                           |
| <code>p</code>     | Vector of probabilities.                                                                                                                              |
| <code>Finv</code>  | Quantile function with signature $\text{Finv}(x, \text{lower}, \log)$ , where $x$ is numeric and <code>lower</code> and <code>log</code> are logical. |
| <code>n</code>     | Number of draws.                                                                                                                                      |

**Value**

Vector with results. `d_trunc` computes the density, `r_trunc` generates random deviates, `p_trunc` computes the CDF, and `q_trunc` computes quantiles.

**Examples**

```
library(tidyverse)

m = 100 ## Length of sequence for density, CDF, etc
shape1 = 5
shape2 = 2
```

```

lo = 0.5
hi = 0.7

# Density, CDF, and quantile function for untruncated distribution
ff = function(x, log) { dbeta(x, shape1, shape2, log = log) }
FF = function(x, lower, log) { pbeta(x, shape1, shape2, lower.tail = lower, log = log) }
FFinv = function(x, lower, log) { qbeta(x, shape1, shape2, lower.tail = lower, log = log) }

# Compare truncated and untruncated densities
xseq = seq(0, 1, length.out = m)
lo_vec = rep(lo, m)
hi_vec = rep(hi, m)
f0seq = ff(xseq, log = FALSE)
fseq = d_trunc(xseq, lo_vec, hi_vec, ff, FF)
data.frame(x = xseq, f = fseq, f0 = f0seq) %>%
  ggplot() +
  geom_line(aes(xseq, fseq)) +
  geom_line(aes(xseq, f0seq), lty = 2) +
  xlab("x") +
  ylab("Density") +
  theme_minimal()

# Compare truncated densities and empirical density of generated draws
n = 100000
lo_vec = rep(lo, n)
hi_vec = rep(hi, n)
x = r_trunc(n = n, lo_vec, hi_vec, FF, FFinv)
hist(x, probability = TRUE, breaks = 15)
points(xseq, fseq)

# Compare empirical CDF of draws with CDF function
Femp = ecdf(x)
lo_vec = rep(lo, m)
hi_vec = rep(hi, m)
Fseq = p_trunc(xseq, lo_vec, hi_vec, FF)
data.frame(x = xseq, FF = Fseq) %>%
  mutate(F0 = Femp(x)) %>%
  ggplot() +
  geom_line(aes(xseq, FF), lwd = 1.2) +
  geom_line(aes(xseq, F0), col = "orange") +
  xlab("x") +
  ylab("Probability") +
  theme_minimal()

# Compare empirical quantiles of draws with quantile function
pseq = seq(0, 1, length.out = m)
lo_vec = rep(lo, m)
hi_vec = rep(hi, m)
Finvseq = q_trunc(pseq, lo_vec, hi_vec, FF, FFinv)
Finvemp = quantile(x, prob = pseq)
data.frame(p = pseq, Finv = Finvseq, Finvemp = Finvemp) %>%
  ggplot() +
  geom_line(aes(pseq, Finv), lwd = 1.2) +

```

```
geom_line(aes(pseq, Finvemp), col = "orange") +  
xlab("p") +  
ylab("Quantile") +  
theme_minimal()
```

---

univariate-optimization

*Univariate Optimization*

---

## Description

Univariate Optimization

## Usage

```
goldensection(f, lower, upper, args)
```

```
optimize_brent(f, lower, upper, args)
```

## Arguments

|       |                                                               |
|-------|---------------------------------------------------------------|
| f     | Function to optimize.                                         |
| lower | Lower limit of search interval. Must be finite.               |
| upper | Upper limit of search interval. Must be finite.               |
| args  | List of additional arguments from the function optimize_args. |

## Value

A list with the form of a `optimize_result` described in section "Univariate Optimization" of the package vignette.

## Examples

```
f = function(x) { x^2 - 1 }  
args = optimize_args()  
goldensection(f, 0, 10, args)  
optimize_brent(f, 0, 10, args)
```

---

|        |                              |
|--------|------------------------------|
| which0 | <i>Matrix Which Function</i> |
|--------|------------------------------|

---

**Description**

Matrix Which Function

**Usage**

```
which0(X, f)
```

**Arguments**

|   |                                               |
|---|-----------------------------------------------|
| X | A matrix                                      |
| f | A predicate to apply to each element of $X$ . |

**Details**

The which C++ functions are intended to operate like the following call in R.

```
which(f(X), arr.ind = TRUE) - 1
```

The R functions exposed here are specific to numeric-valued matrices, but the underlying C++ functions are intended to work with any type of Rcpp Matrix.

**Value**

A matrix with two columns. Each row contains a row and column index corresponding to an element of  $X$  that matches the criteria of  $f$ . See section "Which" of the package vignette for details.

**Examples**

```
X = matrix(1:12 / 6, nrow = 4, ncol = 3)
f = function(x) { x < 1 }
which0(X, f)
```

# Index

args, 2

bfgs1 (multivariate-optimization), 10  
bfgs2 (multivariate-optimization), 10  
bfgs\_args (args), 2

cg1 (multivariate-optimization), 10  
cg2 (multivariate-optimization), 10  
cg\_args (args), 2  
col\_apply (matrix\_apply), 9

d\_trunc (trunc), 15  
deriv, 3  
deriv1 (deriv), 3  
deriv2 (deriv), 3

fd\_deriv1 (deriv), 3  
fd\_deriv2 (deriv), 3  
findroot, 5  
findroot\_args (args), 2  
findroot\_bisect (findroot), 5  
findroot\_brent (findroot), 5  
fntl (fntl-package), 2  
fntl-package, 2

goldensection  
    (univariate-optimization), 17  
gradient0, 6

hessian0, 7

integrate0, 7  
integrate\_args (args), 2

jacobian0, 8

lbfgsb1 (multivariate-optimization), 10  
lbfgsb2 (multivariate-optimization), 10  
lbfgsb\_args (args), 2

mat\_apply (matrix\_apply), 9

matrix\_apply, 9  
multivariate-optimization, 10

neldermead (multivariate-optimization),  
    10  
neldermead\_args (args), 2  
nlm1 (multivariate-optimization), 10  
nlm2 (multivariate-optimization), 10  
nlm3 (multivariate-optimization), 10  
nlm\_args (args), 2

optimize\_args (args), 2  
optimize\_brent  
    (univariate-optimization), 17  
outer, 12  
outer1 (outer), 12  
outer1\_matvec (outer), 12  
outer1\_triplet (outer), 12  
outer2 (outer), 12  
outer2\_matvec (outer), 12  
outer2\_triplet (outer), 12

p\_trunc (trunc), 15

q\_trunc (trunc), 15

r\_trunc (trunc), 15  
richardson\_args (args), 2  
row\_apply (matrix\_apply), 9

solve\_cg, 14

trunc, 15

univariate-optimization, 17

which0, 18