

# The **libcoin** Package

Torsten Hothorn  
Universität Zürich

October 7, 2021

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>R Code</b>	<b>3</b>
2.1	R User Interface . . . . .	3
2.1.1	One-Dimensional Case (“1d”) . . . . .	4
2.1.2	Two-Dimensional Case (“2d”) . . . . .	7
2.1.3	Methods and Tests . . . . .	10
2.1.4	Tabulations . . . . .	15
2.2	Manual Pages . . . . .	18
<b>3</b>	<b>C Code</b>	<b>21</b>
3.1	Header and Source Files . . . . .	21
3.2	Variables . . . . .	24
3.2.1	Example Data and Code . . . . .	29
3.3	Conventions . . . . .	31
3.4	C User Interface . . . . .	31
3.4.1	One-Dimensional Case (“1d”) . . . . .	31
3.4.2	Two-Dimensional Case (“2d”) . . . . .	42
3.5	Tests . . . . .	53
3.6	Test Statistics . . . . .	60
3.7	Linear Statistics . . . . .	81
3.8	Expectation and Covariance . . . . .	82
3.8.1	Linear Statistic . . . . .	82
3.8.2	Influence . . . . .	84
3.8.3	X . . . . .	88
3.9	Computing Sums . . . . .	93
3.9.1	Simple Sums . . . . .	94
3.9.2	Kronecker Sums . . . . .	98
3.9.3	Column Sums . . . . .	112
3.9.4	Tables . . . . .	117
3.10	Utilities . . . . .	131
3.10.1	Blocks . . . . .	131
3.10.2	Permutation Helpers . . . . .	136
3.10.3	Other Utils . . . . .	139
3.11	Memory . . . . .	151
<b>4</b>	<b>Package Infrastructure</b>	<b>162</b>

# Licence

Copyright (C) 2017-2021 Torsten Hothorn

This file is part of the **libcoin** R add-on package.

**libcoin** is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 2.

**libcoin** is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with **libcoin**. If not, see [<http://www.gnu.org/licenses/>](http://www.gnu.org/licenses/).

# Chapter 1

## Introduction

The **libcoin** package implements a generic framework for permutation tests. We assume that we are provided with  $n$  observations

$$(\mathbf{Y}_i, \mathbf{X}_i, w_i, \text{block}_i), \quad i = 1, \dots, N.$$

The variables  $\mathbf{Y}$  and  $\mathbf{X}$  from sample spaces  $\mathcal{Y}$  and  $\mathcal{X}$  may be measured at arbitrary scales and may be multivariate as well. In addition to those measurements, case weights  $w_i \in \mathbb{N}$  and a factor  $\text{block}_i \in \{1, \dots, B\}$  coding for  $B$  independent blocks may be available. We are interested in testing the null hypothesis of independence of  $\mathbf{Y}$  and  $\mathbf{X}$

$$H_0 : D(\mathbf{Y} \mid \mathbf{X}) = D(\mathbf{Y})$$

against arbitrary alternatives. [Strasser and Weber \(1999\)](#) suggest to derive scalar test statistics for testing  $H_0$  from multivariate linear statistics of a specific linear form. Let  $\mathcal{A} \subseteq \{1, \dots, N\}$  denote some subset of the observation numbers and consider the linear statistic

$$\mathbf{T}(\mathcal{A}) = \text{vec} \left( \sum_{i \in \mathcal{A}} w_i g(\mathbf{X}_i) h(\mathbf{Y}_i, \{\mathbf{Y}_i \mid i \in \mathcal{A}\})^\top \right) \in \mathbb{R}^{pq}. \quad (1.1)$$

Here,  $g : \mathcal{X} \rightarrow \mathbb{R}^p$  is a transformation of  $\mathbf{X}$  known as the *regression function* and  $h : \mathcal{Y} \times \mathcal{Y}^n \rightarrow \mathbb{R}^q$  is a transformation of  $\mathbf{Y}$  known as the *influence function*, where the latter may depend on  $\mathbf{Y}_i$  for  $i \in \mathcal{A}$  in a permutation symmetric way. We will give specific examples on how to choose  $g$  and  $h$  later on.

With  $\mathbf{x}_i = g(\mathbf{X}_i) \in \mathbb{R}^p$  and  $\mathbf{y}_i = h(\mathbf{Y}_i, \{\mathbf{Y}_i, i \in \mathcal{A}\}) \in \mathbb{R}^q$  we write

$$\mathbf{T}(\mathcal{A}) = \text{vec} \left( \sum_{i \in \mathcal{A}} w_i \mathbf{x}_i \mathbf{y}_i^\top \right) \in \mathbb{R}^{pq}. \quad (1.2)$$

The **libcoin** package doesn't handle neither  $g$  nor  $h$ , this is the job of **coin** and we therefore continue with  $\mathbf{x}_i$  and  $\mathbf{y}_i$ .

The distribution of  $\mathbf{T}$  depends on the joint distribution of  $\mathbf{Y}$  and  $\mathbf{X}$ , which is unknown under almost all practical circumstances. At least under the null hypothesis one can dispose of this dependency by fixing  $\mathbf{X}_i, i \in \mathcal{A}$  and conditioning on all possible permutations  $S(\mathcal{A})$  of the responses  $\mathbf{Y}_i, i \in \mathcal{A}$ . This principle leads to test procedures known as *permutation tests*. The conditional expectation  $\mu(\mathcal{A}) \in \mathbb{R}^{pq}$  and covariance  $\Sigma(\mathcal{A}) \in \mathbb{R}^{pq \times pq}$  of  $\mathbf{T}$  under  $H_0$  given all permutations  $\sigma \in S(\mathcal{A})$  of the responses are derived by [Strasser](#)

and Weber (1999):

$$\begin{aligned}
\mu(\mathcal{A}) &= \mathbb{E}(\mathbf{T}(\mathcal{A}) \mid S(\mathcal{A})) = \text{vec} \left( \left( \sum_{i \in \mathcal{A}} w_i \mathbf{x}_i \right) \mathbb{E}(h \mid S(\mathcal{A}))^\top \right), \\
\Sigma(\mathcal{A}) &= \mathbb{V}(\mathbf{T}(\mathcal{A}) \mid S(\mathcal{A})) \\
&= \frac{\mathbf{w} \cdot}{\mathbf{w} \cdot(\mathcal{A}) - 1} \mathbb{V}(h \mid S(\mathcal{A})) \otimes \left( \sum_{i \in \mathcal{A}} w_i \mathbf{x}_i \otimes w_i \mathbf{x}_i^\top \right) \\
&\quad - \frac{1}{\mathbf{w} \cdot(\mathcal{A}) - 1} \mathbb{V}(h \mid S(\mathcal{A})) \otimes \left( \sum_{i \in \mathcal{A}} w_i \mathbf{x}_i \right) \otimes \left( \sum_{i \in \mathcal{A}} w_i \mathbf{x}_i \right)^\top
\end{aligned} \tag{1.3}$$

where  $\mathbf{w} \cdot(\mathcal{A}) = \sum_{i \in \mathcal{A}} w_i$  denotes the sum of the case weights, and  $\otimes$  is the Kronecker product. The conditional expectation of the influence function is

$$\mathbb{E}(h \mid S(\mathcal{A})) = \mathbf{w} \cdot(\mathcal{A})^{-1} \sum_{i \in \mathcal{A}} w_i \mathbf{y}_i \in \mathbb{R}^Q$$

with corresponding  $Q \times Q$  covariance matrix

$$\mathbb{V}(h \mid S(\mathcal{A})) = \mathbf{w} \cdot(\mathcal{A})^{-1} \sum_{i \in \mathcal{A}} w_i (\mathbf{y}_i - \mathbb{E}(h \mid S(\mathcal{A}))) (\mathbf{y}_i - \mathbb{E}(h \mid S(\mathcal{A})))^\top.$$

With  $A_b = \{i \mid \text{block}_i = b\}$  we get  $\mathbf{T} = \sum_{b=1}^B T(\mathcal{A}_b)$ ,  $\mu = \sum_{b=1}^B \mu(\mathcal{A}_b)$  and  $\Sigma = \sum_{b=1}^B \Sigma(\mathcal{A}_b)$ .

Having the conditional expectation and covariance at hand we are able to standardize a linear statistic  $\mathbf{T} \in \mathbb{R}^{PQ}$  of the form (1.2). Univariate test statistics  $c$  mapping an observed linear statistic  $\mathbf{t} \in \mathbb{R}^{PQ}$  into the real line can be of arbitrary form. An obvious choice is the maximum of the absolute values of the standardized linear statistic

$$c_{\max}(\mathbf{t}, \mu, \Sigma) = \max \left| \frac{\mathbf{t} - \mu}{\text{diag}(\Sigma)^{1/2}} \right|$$

utilizing the conditional expectation  $\mu$  and covariance matrix  $\Sigma$ . The application of a quadratic form  $c_{\text{quad}}(\mathbf{t}, \mu, \Sigma) = (\mathbf{t} - \mu) \Sigma^+ (\mathbf{t} - \mu)^\top$  is one alternative, although computationally more expensive because the Moore-Penrose inverse  $\Sigma^+$  of  $\Sigma$  is involved.

The definition of one- and two-sided  $p$ -values used for the computations in the **libcoin** package is

$$\begin{aligned}
P(c(\mathbf{T}, \mu, \Sigma) \leq c(\mathbf{t}, \mu, \Sigma)) &\quad (\text{less}) \\
P(c(\mathbf{T}, \mu, \Sigma) \geq c(\mathbf{t}, \mu, \Sigma)) &\quad (\text{greater}) \\
P(|c(\mathbf{T}, \mu, \Sigma)| \leq |c(\mathbf{t}, \mu, \Sigma)|) &\quad (\text{two-sided}).
\end{aligned}$$

Note that for quadratic forms only two-sided  $p$ -values are available and that in the one-sided case maximum type test statistics are replaced by

$$\min \left( \frac{\mathbf{t} - \mu}{\text{diag}(\Sigma)^{1/2}} \right) \quad (\text{less}) \text{ and } \max \left( \frac{\mathbf{t} - \mu}{\text{diag}(\Sigma)^{1/2}} \right) \quad (\text{greater}).$$

This single source file implements and documents the **libcoin** package following the literate programming paradigm. The keynote lecture on literate programming by Donald E. Knuth given at useR! 2016 in Stanford very much motivated this little experiment.

# Chapter 2

## R Code

### 2.1 R User Interface

"libcoin.R" 3a≡

```
⟨ R Header 166a ⟩
⟨ LinStatExpCov 4 ⟩
⟨ LinStatExpCov1d 6 ⟩
⟨ LinStatExpCov2d 8 ⟩
⟨ vcov LinStatExpCov 10 ⟩
⟨ doTest 12 ⟩
⟨ Contrasts 14 ⟩
◇
```

The **libcoin** package implements two functions, `LinStatExpCov` and `doTest` for the computation of linear statistics, their expectation and covariance as well as for the computation of test statistics and  $p$ -values. There are two interfaces: One (labelled “1d”) when the data is available as matrices **X** and **Y**, both with the same number of rows  $N$ . The second interface (labelled “2d”) handles the case when the data is available in aggregated form; details will be explained later.

```
⟨ LinStatExpCov Prototype 3b ⟩ ≡
(X, Y, ix = NULL, iy = NULL, weights = integer(0),
 subset = integer(0), block = integer(0), checkNAs = TRUE,
 varonly = FALSE, nresample = 0, standardise = FALSE,
 tol = sqrt(.Machine$double.eps))◇
```

Fragment referenced in 4, 18.

Uses: `block` 28bd, `subset` 27be, 28a, `weights` 26c.

$\langle \text{LinStatExpCov } 4 \rangle \equiv$

```

LinStatExpCov <-
function(LinStatExpCov Prototype 3b)
{
  if (missing(X) & !is.null(ix) & is.null(iy)) {
    X <- ix
    ix <- NULL
  }

  if (missing(X)) X <- integer(0)

  ## <FIXME> for the time being only!!! </FIXME>
  ## if (length(subset) > 0) subset <- sort(subset)

  if (is.null(ix) & is.null(iy))
    return(.LinStatExpCov1d(X = X, Y = Y, weights = weights,
                           subset = subset, block = block,
                           checkNAs = checkNAs,
                           varonly = varonly, nresample = nresample,
                           standardise = standardise, tol = tol))

  if (!is.null(ix) & !is.null(iy))
    return(.LinStatExpCov2d(X = X, Y = Y, ix = ix, iy = iy,
                           weights = weights, subset = subset,
                           block = block, varonly = varonly,
                           checkNAs = checkNAs, nresample = nresample,
                           standardise = standardise, tol = tol))

  stop("incorrect call to LinStatExpCov")
}

```

Fragment referenced in 3a.

Uses: block 28bd, subset 27be, 28a, weights 26c, weights, 26de.

### 2.1.1 One-Dimensional Case (“1d”)

We assume that  $\mathbf{x}_i$  and  $\mathbf{y}_i$  for  $i = 1, \dots, N$  are available as numeric matrices  $\mathbf{X}$  and  $\mathbf{Y}$  with  $N$  rows as well as  $P$  and  $Q$  columns, respectively. The special case of a dummy matrix  $\mathbf{X}$  with  $P$  columns can also be represented by a factor at  $P$  levels. The vector of case weights `weights` can be stored as `integer` or `double` (possibly resulting from an aggregation of  $N > \text{INT\_MAX}$  observations). The subset vector `subset` may contain the elements  $1, \dots, N$  as `integer` or `double` (for  $N > \text{INT\_MAX}$ ) and can be longer than  $N$ . The `subset` vector MUST be sorted. `block` is a factor at  $B$  levels of length  $N$ .

*⟨ Check weights, subset, block 5a ⟩ ≡*

```

if (is.null(weights)) weights <- integer(0)

if (length(weights) > 0) {
  if (!(N == length(weights)) && all(weights >= 0))
    stop("incorrect weights")
  if (checkNAs) stopifnot(!anyNA(weights))
}

if (is.null(subset)) subset <- integer(0)

if (length(subset) > 0 && checkNAs) {
  rs <- range(subset)
  if (anyNA(rs)) stop("no missing values allowed in subset")
  if (!(rs[2] <= N) && (rs[1] >= 1L))
    stop("incorrect subset")
}

if (is.null(block)) block <- integer(0)

if (length(block) > 0) {
  if (!(N == length(block)) && is.factor(block))
    stop("incorrect block")
  if (checkNAs) stopifnot(!anyNA(block))
}

```

Fragment referenced in 6, 8, 16.

Uses: block 28bd, N 24bc, subset 27be, 28a, weights 26c.

Missing values are only allowed in X and Y, all other vectors must not contain NAs. Missing values are dealt with by excluding the corresponding observations from the subset vector.

*⟨ Handle Missing Values 5b ⟩ ≡*

```

ms <- !complete.cases(X, Y)
if (all(ms))
  stop("all observations are missing")
if (any(ms)) {
  if (length(subset) > 0) {
    if (all(subset %in% which(ms)))
      stop("all observations are missing")
    subset <- subset[!(subset %in% which(ms))]
  } else {
    subset <- (1:N)[-which(ms)]
  }
}

```

Fragment referenced in 6.

Uses: N 24bc, subset 27be, 28a.

The logical argument `varonly` triggers the computation of the diagonal elements of the covariance matrix  $\Sigma$  only. `nresample` permuted linear statistics under the null hypothesis  $H_0$  are returned on the original and standardised scale (the latter only when `standardise` is `TRUE`). Variances smaller than `tol` are treated as being zero.



$\langle \text{LinStatExpCov1d } 6 \rangle \equiv$

```
.LinStatExpCov1d <-
function(X, Y, weights = integer(0), subset = integer(0), block = integer(0),
  checkNAs = TRUE, varonly = FALSE, nresample = 0, standardise = FALSE,
  tol = sqrt(.Machine$double.eps))
{
  if (NROW(X) != NROW(Y))
    stop("dimensions of X and Y don't match")
  N <- NROW(X)

  if (is.integer(X)) {
    if (is.null(attr(X, "levels")) || checkNAs) {
      rg <- range(X)
      if (anyNA(rg))
        stop("no missing values allowed in X")
      stopifnot(rg[1] > 0) # no missing values allowed here!
      if (is.null(attr(X, "levels")))
        attr(X, "levels") <- 1:rg[2]
    }
  }

  if (is.factor(X) && checkNAs)
    stopifnot(!anyNA(X))

   $\langle \text{Check weights, subset, block } 5a \rangle$ 

  if (checkNAs) {
     $\langle \text{Handle Missing Values } 5b \rangle$ 
  }

  ret <- .Call(R_ExpectationCovarianceStatistic, X, Y, weights, subset,
    block, as.integer(varonly), as.double(tol))
  ret$varonly <- as.logical(ret$varonly)
  ret$Xfactor <- as.logical(ret$Xfactor)
  if (nresample > 0) {
    ret$PermutedLinearStatistic <-
      .Call(R_PermutedLinearStatistic, X, Y, weights, subset,
        block, as.double(nresample))
    if (standardise)
      ret$StandardisedPermutedLinearStatistic <-
        .Call(R_StandardisePermutedLinearStatistic, ret)
  }
  class(ret) <- c("LinStatExpCov1d", "LinStatExpCov")
  ret
}
◇
```

Fragment referenced in 3a.

Uses: block 28bd, N 24bc, NROW 139b, R\_ExpectationCovarianceStatistic 32c, R\_PermutedLinearStatistic 40, subset 27be, 28a, weights 26c, weights, 26de.

Here is a simple example. We have five groups and a uniform outcome (rounded to one digit) and want to test independence of group membership and outcome. The simplest way is to set-up the dummy matrix explicitly:

```
> isequal <- function(a, b) {
+   attributes(a) <- NULL
```

```

+     attributes(b) <- NULL
+     if (!isTRUE(all.equal(a, b))) {
+         print(a, digits = 10)
+         print(b, digits = 10)
+         FALSE
+     } else
+         TRUE
+ }
> library("libcoin")
> set.seed(290875)
> x <- gl(5, 20)
> y <- round(runif(length(x)), 1)
> ls1 <- LinStatExpCov(X = model.matrix(~ x - 1), Y = matrix(y, ncol = 1))
> ls1$LinearStatistic

[1]  8.8  9.5 10.3  9.8 10.5

> tapply(y, x, sum)

      1      2      3      4      5
8.8  9.5 10.3  9.8 10.5

```

The linear statistic is simply the sum of the response in each group. Alternatively, we can compute the same object without setting-up the dummy matrix:

```

> ls2 <- LinStatExpCov(X = x, Y = matrix(y, ncol = 1))
> all.equal(ls1[-grep("Xfactor", names(ls1))],
+          ls2[-grep("Xfactor", names(ls2))])

[1] TRUE

```

The results are identical, except for a logical indicating that a factor was used to represent the dummy matrix  $X$ .

## 2.1.2 Two-Dimensional Case (“2d”)

Sometimes the data takes only a few unique values and considerable computational speedups can be achieved taking this information into account. Let  $\mathbf{ix}$  denote an integer vector with elements  $0, \dots, L_x$  of length  $N$  and  $\mathbf{iy}$  an integer vector with elements  $0, \dots, L_y$ , also of length  $N$ . The matrix  $\mathbf{X}$  is now of dimension  $(L_x + 1) \times P$  and the matrix  $\mathbf{Y}$  of dimension  $(L_y + 1) \times Q$ . The combination of  $\mathbf{X}$  and  $\mathbf{ix}$  means that the  $i$ th observation corresponds to the row  $\mathbf{X}[\mathbf{ix}[i] + 1, ]$ . This looks cumbersome in R notation but is a very efficient way of dealing with missing values at C level. By convention, elements of  $\mathbf{ix}$  being zero denote a missing value (NAs are not allowed in  $\mathbf{ix}$  and  $\mathbf{iy}$ ). Thus, the first row of  $\mathbf{X}$  corresponds to a missing value. If the first row is simply zero, missing values do not contribute to any of the sums computed later. Even more important is the fact that all entities, such as linear statistics etc., can be computed from the two-way tabulation (therefore the abbreviation “2d”) over the  $N$  elements of  $\mathbf{ix}$  and  $\mathbf{iy}$ . Once such a table was computed, the remaining computations can be performed in dimension  $L_x \times L_y$ , typically much smaller than  $N$ .

⟨ *LinStatExpCov2d 8* ⟩ ≡

```
.LinStatExpCov2d <-
function(X = numeric(0), Y, ix, iy, weights = integer(0), subset = integer(0),
        block = integer(0), checkNAs = TRUE, varonly = FALSE, nresample = 0,
        standardise = FALSE, tol = sqrt(.Machine$double.eps))
{
  IF <- function(x) is.integer(x) || is.factor(x)

  if (!(length(ix) == length(iy)) && IF(ix) && IF(iy)))
    stop("incorrect ix and/or iy")
  N <- length(ix)

  ⟨ Check ix 9a ⟩

  ⟨ Check iy 9b ⟩

  if (length(X) > 0) {
    if (!(NROW(X) == (length(attr(ix, "levels")) + 1) &&
          all(complete.cases(X))))
      stop("incorrect X")
  }

  if (!(NROW(Y) == (length(attr(iy, "levels")) + 1) &&
        all(complete.cases(Y))))
    stop("incorrect Y")

  ⟨ Check weights, subset, block 5a ⟩

  ret <- .Call(R_ExpectationCovarianceStatistic_2d, X, ix, Y, iy,
              weights, subset, block, as.integer(varonly), as.double(tol))
  ret$varonly <- as.logical(ret$varonly)
  ret$Xfactor <- as.logical(ret$Xfactor)
  if (nresample > 0) {
    ret$PermutedLinearStatistic <-
      .Call(R_PermutedLinearStatistic_2d, X, ix, Y, iy, block, nresample, ret$Table)
    if (standardise)
      ret$StandardisedPermutedLinearStatistic <-
        .Call(R_StandardisePermutedLinearStatistic, ret)
  }
  class(ret) <- c("LinStatExpCov2d", "LinStatExpCov")
  ret
}
◇
```

Fragment referenced in 3a.

Uses: block 28bd, N 24bc, NROW 139b, R\_ExpectationCovarianceStatistic\_2d 44, R\_PermutedLinearStatistic\_2d 51, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc.

ix can be a factor but without any missing values

$\langle \text{Check ix 9a} \rangle \equiv$

```

if (is.null(attr(ix, "levels"))) {
  rg <- range(ix)
  if (anyNA(rg))
    stop("no missing values allowed in ix")
  stopifnot(rg[1] >= 0)
  attr(ix, "levels") <- 1:rg[2]
} else {
  ## lev can be data.frame (see inum::inum)
  lev <- attr(ix, "levels")
  if (!is.vector(lev)) lev <- 1:NROW(lev)
  attr(ix, "levels") <- lev
  if (checkNAs) stopifnot(!anyNA(ix))
}
◇

```

Fragment referenced in [8](#), [16](#).

Uses: [NROW 139b](#).

$\langle \text{Check iy 9b} \rangle \equiv$

```

if (is.null(attr(iy, "levels"))) {
  rg <- range(iy)
  if (anyNA(rg))
    stop("no missing values allowed in iy")
  stopifnot(rg[1] >= 0)
  attr(iy, "levels") <- 1:rg[2]
} else {
  ## lev can be data.frame (see inum::inum)
  lev <- attr(iy, "levels")
  if (!is.vector(lev)) lev <- 1:NROW(lev)
  attr(iy, "levels") <- lev
  if (checkNAs) stopifnot(!anyNA(iy))
}
◇

```

Fragment referenced in [8](#), [16](#).

Uses: [NROW 139b](#).

In our small example, we can set-up the data in the following way

```

> X <- rbind(0, diag(nlevels(x)))
> ix <- unclass(x)
> ylev <- sort(unique(y))
> Y <- rbind(0, matrix(ylev, ncol = 1))
> iy <- .bincode(y, breaks = c(-Inf, ylev, Inf))
> ls3 <- LinStatExpCov(X = X, ix = ix, Y = Y, iy = iy)
> all.equal(ls1[-grep("Table", names(ls1))],
+          ls3[-grep("Table", names(ls3))])

[1] TRUE

> ### works also with factors
> ls3 <- LinStatExpCov(X = X, ix = factor(ix), Y = Y, iy = factor(iy))
> all.equal(ls1[-grep("Table", names(ls1))],
+          ls3[-grep("Table", names(ls3))])

```

```
[1] TRUE
```

Similar to the one-dimensional case, we can also omit the  $X$  matrix here

```
> ls4 <- LinStatExpCov(ix = ix, Y = Y, iy = iy)
> all.equal(ls3[-grep("Xfactor", names(ls3))],
+          ls4[-grep("Xfactor", names(ls4))])
```

```
[1] TRUE
```

It is important to note that all computations are based on the tabulations

```
> ls3$Table

, , 1

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]    0    0    0    0    0    0    0    0    0    0    0    0
[2,]    0    0    4    4    1    2    3    0    1    2    3    0
[3,]    0    2    2    1    2    2    5    0    1    1    3    1
[4,]    0    1    1    4    0    1    5    2    0    2    3    1
[5,]    0    0    2    2    4    2    2    1    3    2    1    1
[6,]    0    1    3    1    1    1    2    2    2    6    1    0

> xtabs(~ ix + iy)

      iy
ix  1 2 3 4 5 6 7 8 9 10 11
  1 0 4 4 1 2 3 0 1 2  3  0
  2 2 2 1 2 2 5 0 1 1  3  1
  3 1 1 4 0 1 5 2 0 2  3  1
  4 0 2 2 4 2 2 1 3 2  1  1
  5 1 3 1 1 1 2 2 2 6  1  0
```

where the former would record missing values in the first row / column.

### 2.1.3 Methods and Tests

Objects of class `LinStatExpCov` returned by `LinStatExpCov()` contain the symmetric covariance matrix as a vector of the lower triangular elements. The `vcov` method allows to extract the full covariance matrix from such an object.

$\langle vcov \text{ LinStatExpCov } 10 \rangle \equiv$

```
vcov.LinStatExpCov <-
function(object, ...)
{
  if (object$varonly)
    stop("cannot extract covariance matrix")
  drop(.Call(R_unpack_sym, object$Covariance, NULL, 0L))
}
◇
```

Fragment referenced in [3a](#).

Uses: `R_unpack_sym` [149](#).

```
> ls1$Covariance
```

```

[1] 1.3572364 -0.3393091 -0.3393091 -0.3393091 -0.3393091 1.3572364
[7] -0.3393091 -0.3393091 -0.3393091 1.3572364 -0.3393091 -0.3393091
[13] 1.3572364 -0.3393091 1.3572364

```

```
> vcov(ls1)
```

```

      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.3572364 -0.3393091 -0.3393091 -0.3393091 -0.3393091
[2,] -0.3393091 1.3572364 -0.3393091 -0.3393091 -0.3393091
[3,] -0.3393091 -0.3393091 1.3572364 -0.3393091 -0.3393091
[4,] -0.3393091 -0.3393091 -0.3393091 1.3572364 -0.3393091
[5,] -0.3393091 -0.3393091 -0.3393091 -0.3393091 1.3572364

```

The most important task is, however, to compute test statistics and  $p$ -values. `doTest()` allows to compute the statistics  $c_{\max}$  (taking `alternative` into account) and  $c_{\text{quad}}$  along with the corresponding  $p$ -values. If `nresample = 0` was used in the call to `LinStatExpCov()`,  $p$ -values are obtained from the limiting asymptotic distribution whenever such a thing is available at reasonable costs. Otherwise, the permutation  $p$ -value is returned (along with the permuted test statistics when `PermutedStatistics` is `TRUE`). The  $p$ -values (`lower = FALSE`) or  $(1 - p)$ -values (`lower = TRUE`) can be computed on the log-scale.

```

⟨ doTest Prototype 11 ⟩ ≡
  (object, teststat = c("maximum", "quadratic", "scalar"),
   alternative = c("two.sided", "less", "greater"), pvalue = TRUE,
   lower = FALSE, log = FALSE, PermutedStatistics = FALSE,
   minbucket = 10L, ordered = TRUE, maxselect = object$Xfactor,
   pargs = GenzBretz())◇

```

Fragment referenced in [12](#), [19](#).

`< doTest 12 > ≡`

```
### note: lower = FALSE => p-value; lower = TRUE => 1 - p-value
doTest <-
function(doTest Prototype 11)
{
  teststat <- match.arg(teststat, choices = c("maximum", "quadratic", "scalar"))
  if (!any(teststat == c("maximum", "quadratic", "scalar")))
    stop("incorrect teststat")
  alternative <- alternative[1]
  if (!any(alternative == c("two.sided", "less", "greater")))
    stop("incorrect alternative")

  if (maxselect)
    stopifnot(object$Xfactor)

  if (teststat == "quadratic" || maxselect) {
    if (alternative != "two.sided")
      stop("incorrect alternative")
  }

  test <- which(c("maximum", "quadratic", "scalar") == teststat)
  if (test == 3) {
    if (length(object$LinearStatistic) != 1)
      stop("scalar test statistic not applicable")
    test <- 1L # scalar is maximum internally
  }
  alt <- which(c("two.sided", "less", "greater") == alternative)

  if (!pvalue & (NCOL(object$PermutedLinearStatistic) > 0))
    object$PermutedLinearStatistic <- matrix(NA_real_, nrow = 0, ncol = 0)

  if (!maxselect) {
    if (teststat == "quadratic") {
      ret <- .Call(R_QuadraticTest, object, as.integer(pvalue), as.integer(lower),
        as.integer(log), as.integer(PermutedStatistics))
    } else {
      ret <- .Call(R_MaximumTest, object, as.integer(alt), as.integer(pvalue),
        as.integer(lower), as.integer(log), as.integer(PermutedStatistics),
        as.integer(pargs$maxpts), as.double(pargs$releps),
        as.double(pargs$abseps))
      if (teststat == "scalar") {
        var <- if (object$varonly) object$Variance else object$Covariance
        ret$TestStatistic <- object$LinearStatistic - object$Expectation
        ret$TestStatistic <-
          if (var > object$tol) ret$TestStatistic / sqrt(var) else NaN
      }
    }
  } else {
    ret <- .Call(R_MaximallySelectedTest, object, as.integer(ordered), as.integer(test),
      as.integer(minbucket), as.integer(lower), as.integer(log))
  }
  if (!PermutedStatistics) ret$PermutedStatistics <- NULL
  ret
}
◇
```

Fragment referenced in [3a](#).

Uses: [NCOL 139c](#).

```

> ### c_max test statistic
> ### no p-value
> doTest(ls1, teststat = "maximum", pvalue = FALSE)

$TestStatistic
[1] 0.8411982

$p.value
[1] NA

> ### p-value
> doTest(ls1, teststat = "maximum")

$TestStatistic
[1] 0.8411982

$p.value
[1] 0.8852087

> ### log(p)-value
> doTest(ls1, teststat = "maximum", log = TRUE)

$TestStatistic
[1] 0.8411982

$p.value
[1] 0.108822

> ### (1-p)-value
> doTest(ls1, teststat = "maximum", lower = TRUE)

$TestStatistic
[1] 0.8411982

$p.value
[1] 0.1150168

> ### log(1 - p)-value
> doTest(ls1, teststat = "maximum", lower = TRUE, log = TRUE)

$TestStatistic
[1] 0.8411982

$p.value
[1] -2.164164

> ### quadratic
> doTest(ls1, teststat = "quadratic")

$TestStatistic
[1] 1.077484

$p.value
[1] 0.897828

```



Sometimes we are interested in contrasts of linear statistics and their corresponding properties. Examples include linear-by-linear association tests, where we assign numeric scores to each level of a factor. To implement this, we implement `lmult` so that we can then left-multiply a matrix to an object of class `LinStatExpCov`.

( *Contrasts 14* )  $\equiv$

```
lmult <-
function(x, object)
{
  stopifnot(!object$varonly)
  stopifnot(is.numeric(x))
  if (is.vector(x)) x <- matrix(x, nrow = 1)
  P <- object$dimension[1]
  stopifnot(ncol(x) == P)
  Q <- object$dimension[2]
  ret <- object
  xLS <- x %%% matrix(object$LinearStatistic, nrow = P)
  xExp <- x %%% matrix(object$Expectation, nrow = P)
  xExpX <- x %%% matrix(object$ExpectationX, nrow = P)
  if (Q == 1) {
    xCov <- tcrossprod(x %%% vcov(object), x)
  } else {
    zmat <- matrix(0, nrow = P * Q, ncol = nrow(x))
    mat <- rbind(t(x), zmat)
    mat <- mat[rep(1:nrow(mat), Q - 1),,drop = FALSE]
    mat <- rbind(mat, t(x))
    mat <- matrix(mat, ncol = Q * nrow(x))
    mat <- t(mat)
    xCov <- tcrossprod(mat %%% vcov(object), mat)
  }
  if (!is.matrix(xCov)) xCov <- matrix(xCov)
  if (length(object$PermutedLinearStatistic) > 0) {
    xPS <- apply(object$PermutedLinearStatistic, 2, function(y)
      as.vector(x %%% matrix(y, nrow = P)))
    if (!is.matrix(xPS)) xPS <- matrix(xPS, nrow = 1)
    ret$PermutedLinearStatistic <- xPS
  }
  ret$LinearStatistic <- as.vector(xLS)
  ret$Expectation <- as.vector(xExp)
  ret$ExpectationX <- as.vector(xExpX)
  ret$Covariance <- as.vector(xCov[lower.tri(xCov, diag = TRUE)])
  ret$Variance <- diag(xCov)
  ret$dimension <- c(NROW(x), Q)
  ret$Xfactor <- FALSE
  if (length(object$StandardisedPermutedLinearStatistic) > 0)
    ret$StandardisedPermutedLinearStatistic <-
      .Call(R_StandardisePermutedLinearStatistic, ret)
  ret
}
◇
```

Fragment referenced in 3a.

Uses: `NROW` 139b, `P` 25a, `Q` 25e, `x` 24d, 25bc, `y` 25d, 26ab.

Here is an example for a linear-by-linear association test.

```
> set.seed(29)
```

```

> ls1d <- LinStatExpCov(X = model.matrix(~ x - 1), Y = matrix(y, ncol = 1),
+                       nresample = 10, standardise = TRUE)
> set.seed(29)
> ls1s <- LinStatExpCov(X = as.double(1:5)[x], Y = matrix(y, ncol = 1),
+                       nresample = 10, standardise = TRUE)
> ls1c <- lmult(c(1:5), ls1d)
> stopifnot(isequal(ls1c, ls1s))
> set.seed(29)
> ls1d <- LinStatExpCov(X = model.matrix(~ x - 1), Y = matrix(c(y, y), ncol = 2),
+                       nresample = 10, standardise = TRUE)
> set.seed(29)
> ls1s <- LinStatExpCov(X = as.double(1:5)[x], Y = matrix(c(y, y), ncol = 2),
+                       nresample = 10, standardise = TRUE)
> ls1c <- lmult(c(1:5), ls1d)
> stopifnot(isequal(ls1c, ls1s))

```

### 2.1.4 Tabulations

The tabulation of `ix` and `iy` can be computed without necessarily computing the corresponding linear statistics via `ctabs()`.

```

⟨ ctabs Prototype 15 ⟩ ≡
  (ix, iy = integer(0), block = integer(0), weights = integer(0),
   subset = integer(0), checkNAs = TRUE)◇

```

Fragment referenced in [16](#), [20](#).

Uses: [block 28bd](#), [subset 27be](#), [28a](#), [weights 26c](#).

"ctabs.R" 16≡

```
⟨ R Header 166a ⟩
ctabs <-
function⟨ ctabs Prototype 15 ⟩
{
  stopifnot(is.integer(ix) || is.factor(ix))
  N <- length(ix)

  ⟨ Check ix 9a ⟩

  if (length(iy) > 0) {
    stopifnot(length(iy) == N)
    stopifnot(is.integer(iy) || is.factor(iy))
    ⟨ Check iy 9b ⟩
  }

  ⟨ Check weights, subset, block 5a ⟩

  if (length(iy) == 0 && length(block) == 0)
    return(.Call(R_OneTableSums, ix, weights, subset))
  if (length(block) == 0)
    return(.Call(R_TwoTableSums, ix, iy, weights, subset))
  if (length(iy) == 0)
    return(.Call(R_TwoTableSums, ix, block, weights, subset)[,-1,drop = FALSE])
  return(.Call(R_ThreeTableSums, ix, iy, block, weights, subset))
}
◇
```

Uses: block 28bd, N 24bc, R\_OneTableSums 118a, R\_ThreeTableSums 127b, R\_TwoTableSums 122b, subset 27be, 28a, weights 26c, weights, 26de.

```
> t1 <- ctabs(ix = ix, iy = iy)
> t2 <- xtabs(~ ix + iy)
> max(abs(t1[-1, -1] - t2))
```

```
[1] 0
```



## 2.2 Manual Pages

"LinStatExpCov.Rd" 18

```

\name{LinStatExpCov}
\alias{LinStatExpCov}
\alias{lmult}
\title{
  Linear Statistics with Expectation and Covariance
}
\description{
  Strasser-Weber type linear statistics and their expectation
  and covariance under the independence hypothesis
}
\usage{
LinStatExpCov(LinStatExpCov Prototype 3b)
lmult(x, object)
}
\arguments{
  \item{X}{numeric matrix of transformations.}
  \item{Y}{numeric matrix of influence functions.}
  \item{ix}{an optional integer vector expanding \code{X}.}
  \item{iy}{an optional integer vector expanding \code{Y}.}
  \item{weights}{an optional integer vector of non-negative case weights.}
  \item{subset}{an optional integer vector defining a subset of observations.}
  \item{block}{an optional factor defining independent blocks of observations.}
  \item{checkNAs}{a logical for switching off missing value checks. This
    included switching off checks for suitable values of \code{subset}.
    Use at your own risk.}
  \item{varonly}{a logical asking for variances only.}
  \item{nresample}{an integer defining the number of permuted statistics to draw.}
  \item{standardise}{a logical asking to standardise the permuted statistics.}
  \item{tol}{tolerance for zero variances.}
  \item{x}{a contrast matrix to be left-multiplied in case \code{X} was a factor.}
  \item{object}{an object of class \code{LinStatExpCov}.}
}
\details{
  The function, after minimal preprocessing, calls the underlying C code
  and computes the linear statistic, its expectation and covariance and,
  optionally, \code{nresample} samples from its permutation distribution.

  When both \code{ix} and \code{iy} are missing, the number of rows of
  \code{X} and \code{Y} is the same, ie the number of observations.

  When \code{X} is missing and \code{ix} a factor, the code proceeds as
  if \code{X} were a dummy matrix of \code{ix} without explicitly
  computing this matrix.

  Both \code{ix} and \code{iy} being present means the code treats them
  as subsetting vectors for \code{X} and \code{Y}. Note that \code{ix = 0}
  or \code{iy = 0} means that the corresponding observation is missing
  and the first row of \code{X} and \code{Y} must be zero.

  \code{lmult} allows left-multiplication of a contrast matrix when \code{X}
  was (equivalent to) a factor.
}
\value{
  A list.
}
\references{
  Strasser, H. and Weber, C. (1999). On the asymptotic theory of permutation
  statistics. Mathematical Methods of Statistics 8(2), 220--250.
}
\examples{
wilcox.test(Ozone ~ Month, data = airquality, subset = Month \%in\% c(5, 8))

aq <- subset(airquality, Month \%in\% c(5, 8))
X <- as.double(aq$Month == 5)
Y <- as.double(rank(aq$Ozone))
doTest(LinStatExpCov(X, Y))

```

"doTest.Rd" 19≡

```

\name{doTest}
\alias{doTest}
\title{
  Permutation Test
}
\description{
  Perform permutation test for a linear statistic
}
\usage{
doTest(doTest Prototype 11)
}
\arguments{
  \item{object}{an object returned by \link{LinStatExpCov}.}
  \item{teststat}{type of test statistic to use.}
  \item{alternative}{alternative for scalar or maximum-type statistics.}
  \item{pvalue}{a logical indicating if a p-value shall be computed.}
  \item{lower}{a logical indicating if a p-value (lower is FALSE)
    or 1 - p-value (lower is TRUE) shall be returned.}
  \item{log}{a logical, if TRUE probabilities are log-probabilities.}
  \item{PermutedStatistics}{a logical, return permuted test statistics.}
  \item{minbucket}{minimum weight in either of two groups for maximally selected
    statistics.}
  \item{ordered}{a logical, if TRUE maximally selected statistics assume
    that the cutpoints are ordered.}
  \item{maxselect}{a logical, if TRUE maximally selected statistics are
    computed. This requires that X was an implicitly defined design
    matrix in \link{LinStatExpCov}.}
  \item{pargs}{arguments as in \link[mvtnorm:algorithms]{GenzBretz}.}
}
\details{
  Computes a test statistic, a corresponding p-value and, optionally, cutpoints
  for maximally selected statistics.
}
\value{
  A list.
}
\keyword{htest}
◇

```

"ctabs.Rd" 20≡

```
\name{ctabs}
\alias{ctabs}
\title{
  Cross Tabulation
}
\description{
  Efficient weighted cross tabulation of two factors and a block
}
\usage{
ctabs(ctabs Prototype 15)
}
\arguments{
  \item{ix}{a integer of positive values with zero indicating a missing.}
  \item{iy}{an optional integer of positive values with zero indicating a
    missing.}
  \item{block}{an optional blocking factor without missings.}
  \item{weights}{an optional vector of weights, integer or double.}
  \item{subset}{an optional integer vector indicating a subset.}
  \item{checkNAs}{a logical for switching off missing value checks.}
}
\details{
  A faster version of \code{xtabs(weights ~ ix + iy + block, subset)}.
}
\value{
  If \code{block} is present, a three-way table. Otherwise,
  a one- or two-dimensional table.
}
\examples{
ctabs(ix = 1:5, iy = 1:5, weights = 1:5 / 5)
}
\keyword{univar}
◇
```

Uses: block [28bd](#), subset [27be](#), [28a](#), weights [26c](#), weights, [26de](#).

# Chapter 3

## C Code

The main motivation to implement the **libcoin** package comes from the demand to compute high-dimensional linear statistics (with large  $P$  and  $Q$ ) and the corresponding test statistics very often, either for sampling from the permutation null distribution  $H_0$  or for different subsets of the data. Especially the latter task can be performed *without* actually subsetting the data via the **subset** argument very efficiently (in terms of memory consumption and, depending on the circumstances, speed).

We start with the definition of some macros and global variables in the header files.

### 3.1 Header and Source Files

```
"libcoin_internal.h" 21a≡  
  
  ⟨ C Header 166b⟩  
  ⟨ R Includes 21b⟩  
  ⟨ C Macros 22a⟩  
  ⟨ C Global Variables 22b⟩  
  ◇
```

These includes provide some R infrastructure at C level.

```
⟨ R Includes 21b⟩ ≡  
  
#define USE_FC_LEN_T  
#include <R.h>  
#include <Rinternals.h>  
#include <Rmath.h>  
#include <Rdefines.h>  
#include <R_ext/stats_package.h> /* for S_rcont2 */  
#include <Rversion.h>           // for R_VERSION  
#include <R_ext/Lapack.h> /* for dspev */  
#ifndef FCONE  
# define FCONE  
#endif  
◇
```

Fragment referenced in 21a.

We need three macros: **S** computes the element  $\Sigma_{ij}$  of a symmetric  $n \times n$  matrix when only the lower triangular elements are stored. **LE** implements  $\leq$  with some tolerance, **GE** implements  $\geq$ .



$\langle C \text{ Macros } 22a \rangle \equiv$

```
#define S(i, j, n) ((i) >= (j) ? (n) * (j) + (i) - (j) * ((j) + 1) / 2 : (n) * (i) + (j) -
(i) * ((i) + 1) / 2)
#define LE(x, y, tol) ((x) < (y)) || (fabs((x) - (y)) < (tol))
#define GE(x, y, tol) ((x) > (y)) || (fabs((x) - (y)) < (tol))
◇
```

Fragment referenced in 21a.

Defines: GE 55, 57, LE 57, S 37b, 38b, 47, 48, 60b, 61b, 62b, 65, 67a, 71, 72a, 76a, 80b, 93a, 105, 144, 145a, 147, 153b.

Uses: x 24d, 25bc, y 25d, 26ab.

$\langle C \text{ Global Variables } 22b \rangle \equiv$

```
#define ALTERNATIVE_twosided 1
#define ALTERNATIVE_less 2
#define ALTERNATIVE_greater 3

#define TESTSTAT_maximum 1
#define TESTSTAT_quadratic 2

#define LinearStatistic_SLOT 0
#define Expectation_SLOT 1
#define Covariance_SLOT 2
#define Variance_SLOT 3
#define ExpectationX_SLOT 4
#define varonly_SLOT 5
#define dim_SLOT 6
#define ExpectationInfluence_SLOT 7
#define CovarianceInfluence_SLOT 8
#define VarianceInfluence_SLOT 9
#define Xfactor_SLOT 10
#define tol_SLOT 11
#define PermutedLinearStatistic_SLOT 12
#define StandardisedPermutedLinearStatistic_SLOT 13
#define TableBlock_SLOT 14
#define Sumweights_SLOT 15
#define Table_SLOT 16

#define DoSymmetric 1
#define DoCenter 1
#define DoVarOnly 1
#define Power1 1
#define Power2 2
#define Offset0 0
◇
```

Fragment referenced in 21a.

Defines: CovarianceInfluence\_SLOT 155a, 158b, 159, Covariance\_SLOT 153b, 154a, 158b, 159, dim\_SLOT 151c, 152a, 158b, 159, DoCenter 81d, 86a, 88a, 90, 93a, 100a, 113b, DoSymmetric 81d, 88a, 93a, DoVarOnly 37bc, 38a, 47, ExpectationInfluence\_SLOT 154c, 158b, 159, ExpectationX\_SLOT 154b, 158b, 159, Expectation\_SLOT 153a, 158b, 159, LinearStatistic\_SLOT 152d, 158b, 159, Offset0 35b, 36a, 40, 44, 46c, 47, 85b, 87a, 89a, 92a, 95b, 100a, 109b, 113b, 118a, 122b, 127b, 132b, 136a, PermutedLinearStatistic\_SLOT 157bc, 158b, 159, Power1 86a, 90, 113b, Power2 88a, 93a, StandardisedPermutedLinearStatistic\_SLOT 158b, 159, Sumweights\_SLOT 156a, 157a, 158b, 159, 160b, TableBlock\_SLOT 36a, 155c, 157a, 158b, 159, 160b, Table\_SLOT 156bc, 158b, 159, 161, tol\_SLOT 157d, 158b, 159, VarianceInfluence\_SLOT 155b, 158b, 159, Variance\_SLOT 153b, 158b, 159, varonly\_SLOT 152b, 158b, 159, Xfactor\_SLOT 152c, 158b, 159.

The corresponding header file contains definitions of functions that can be called via `.Call()` from the **libcoin**

package. In addition, packages linking to **libcoin** can access these function at C level (at your own risk, of course!).

"libcoin.h" 23a≡

```

< C Header 166b>
#include "libcoin_internal.h"
< Function Prototypes 23b>
◇

```

< Function Prototypes 23b> ≡

```

extern < R_ExpectationCovarianceStatistic Prototype 32b>;
extern < R_PermutedLinearStatistic Prototype 38c>;
extern < R_StandardisePermutedLinearStatistic Prototype 41c>;
extern < R_ExpectationCovarianceStatistic_2d Prototype 43a>;
extern < R_PermutedLinearStatistic_2d Prototype 50a>;
extern < R_QuadraticTest Prototype 54b>;
extern < R_MaximumTest Prototype 56b>;
extern < R_MaximallySelectedTest Prototype 58>;
extern < R_ExpectationInfluence Prototype 85a>;
extern < R_CovarianceInfluence Prototype 86b>;
extern < R_ExpectationX Prototype 88b>;
extern < R_CovarianceX Prototype 91>;
extern < R_Sums Prototype 95a>;
extern < R_KronSums Prototype 99>;
extern < R_KronSums_Permutation Prototype 109a>;
extern < R_colSums Prototype 113a>;
extern < R_OneTableSums Prototype 117b>;
extern < R_TwoTableSums Prototype 122a>;
extern < R_ThreeTableSums Prototype 127a>;
extern < R_order_subset_wrt_block Prototype 132a>;
extern < R_quadform Prototype 64b>;
extern < R_kronecker Prototype 141c>;
extern < R_MPinv_sym Prototype 146a>;
extern < R_unpack_sym Prototype 148b>;
extern < R_pack_sym Prototype 150b>;
◇

```

Fragment referenced in 23a.

The C file libcoin.c contains all C functions and corresponding R interfaces.

"libcoin.c" 23c≡

```

< C Header 166b>
#include "libcoin_internal.h"
#include <R_ext/stats_stubs.h> /* for S_rcont2 */
#include <mvtnormAPI.h> /* for calling mvtnorm */
< Function Definitions 24a>
◇

```

$\langle \text{Function Definitions 24a} \rangle \equiv$

$\langle \text{MoreUtils 139a} \rangle$   
 $\langle \text{Memory 151a} \rangle$   
 $\langle \text{P-Values 67b} \rangle$   
 $\langle \text{KronSums 98b} \rangle$   
 $\langle \text{colSums 112c} \rangle$   
 $\langle \text{SimpleSums 94c} \rangle$   
 $\langle \text{Tables 117a} \rangle$   
 $\langle \text{Utils 131b} \rangle$   
 $\langle \text{LinearStatistics 81b} \rangle$   
 $\langle \text{Permutations 136b} \rangle$   
 $\langle \text{ExpectationCovariances 82a} \rangle$   
 $\langle \text{Test Statistics 60a} \rangle$   
 $\langle \text{User Interface 31a} \rangle$   
 $\langle \text{2d User Interface 42b} \rangle$   
 $\langle \text{Tests 53b} \rangle$   
 $\diamond$

Fragment referenced in 23c.

## 3.2 Variables

$N$  is the number of observations

$\langle R \ N \ \text{Input 24b} \rangle \equiv$

SEXP  $N$ ,  
 $\diamond$

Fragment referenced in 95a.

Defines:  $N$  5ab, 6, 8, 16, 24c, 35ab, 36ab, 37abc, 38a, 40, 44, 70, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 94a, 95b, 96a, 98a, 100a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 113b, 114a, 116b, 118a, 119a, 122b, 123b, 127b, 128b, 132b, 133b, 134ab, 135a, 136a, 145a.

which at C level is represented as  $R\_xlen\_t$  to allow for  $N > \text{INT\_MAX}$

$\langle C \ \text{integer } N \ \text{Input 24c} \rangle \equiv$

$R\_xlen\_t \ N$   
 $\diamond$

Fragment referenced in 25bc, 34, 40, 44, 81c, 85bc, 87ab, 89ab, 92ab, 95c, 96b, 97abc, 100a, 101b, 109bc, 113b, 118a, 122b, 127b, 132b, 133a, 134ab, 135b.

Defines:  $N$  5ab, 6, 8, 16, 24b, 35ab, 36ab, 37abc, 38a, 40, 44, 70, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 94a, 95b, 96a, 98a, 100a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 113b, 114a, 116b, 118a, 119a, 122b, 123b, 127b, 128b, 132b, 133b, 134ab, 135a, 136a, 145a.

The regressors  $\mathbf{x}_i, i = 1, \dots, N$

$\langle R \ x \ \text{Input 24d} \rangle \equiv$

SEXP  $\mathbf{x}$ ,  
 $\diamond$

Fragment referenced in 31b, 42c, 50a, 81c, 88b, 89b, 91, 92b, 99, 101b, 109ac, 113a, 117b, 122a, 127a.

Defines:  $\mathbf{x}$  8, 14, 18, 22a, 25bc, 32ac, 33, 35ab, 37ac, 38ad, 40, 43b, 44, 45ab, 46c, 47, 50b, 51, 81d, 89a, 90, 92a, 93a, 100a, 101a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 113b, 114a, 116b, 118a, 119a, 121b, 122b, 123b, 126, 127b, 128b, 131a, 139bc, 140a, 145ab, 146ab, 147, 148ab, 149, 150abc.

are either represented as a real matrix with  $N$  rows and  $P$  columns

$\langle C \text{ integer } P \text{ Input 25a} \rangle \equiv$

**int** P

◇

Fragment referenced in 25bc, 34, 81c, 82b, 83, 84, 89b, 92b, 101b, 109c, 160b, 161.

Defines: P 14, 32c, 33, 35ab, 36a, 37ac, 38ab, 40, 44, 45ab, 46c, 47, 48, 49, 51, 55, 56a, 57, 59, 73, 74, 75, 76a, 78, 79ab, 80ab, 81d, 82b, 83, 84, 88b, 89a, 90, 91, 92a, 93a, 99, 100a, 102, 103a, 105, 108, 109ab, 110a, 111b, 112b, 113b, 114a, 116b, 118a, 119a, 121b, 122b, 123b, 126, 127b, 128b, 131a, 140b, 141a, 145a, 158a, 159.

$\langle C \text{ real } x \text{ Input 25b} \rangle \equiv$

**double** \*x,

$\langle C \text{ integer } N \text{ Input 24c} \rangle,$

$\langle C \text{ integer } P \text{ Input 25a} \rangle,$

◇

Fragment referenced in 101c, 110b, 111a, 114b, 145a.

Defines: x 8, 14, 18, 22a, 24d, 25c, 32ac, 33, 35ab, 37ac, 38ad, 40, 43b, 44, 45ab, 46c, 47, 50b, 51, 81d, 89a, 90, 92a, 93a, 100a, 101a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 113b, 114a, 116b, 118a, 119a, 121b, 122b, 123b, 126, 127b, 128b, 131a, 139bc, 140a, 145ab, 146ab, 147, 148ab, 149, 150abc.

or as a factor (an integer at C level) at  $P$  levels

$\langle C \text{ integer } x \text{ Input 25c} \rangle \equiv$

**int** \*x,

$\langle C \text{ integer } N \text{ Input 24c} \rangle,$

$\langle C \text{ integer } P \text{ Input 25a} \rangle,$

◇

Fragment referenced in 106a, 111c, 112a, 119b, 123c, 128c.

Defines: x 8, 14, 18, 22a, 24d, 25b, 32ac, 33, 35ab, 37ac, 38ad, 40, 43b, 44, 45ab, 46c, 47, 50b, 51, 81d, 89a, 90, 92a, 93a, 100a, 101a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 113b, 114a, 116b, 118a, 119a, 121b, 122b, 123b, 126, 127b, 128b, 131a, 139bc, 140a, 145ab, 146ab, 147, 148ab, 149, 150abc.

The influence functions are also either a  $N \times Q$  real matrix

$\langle R \text{ y Input 25d} \rangle \equiv$

**SEXP** y,

◇

Fragment referenced in 31b, 42c, 50a, 85ac, 86b, 87b, 99, 109a, 122a, 127a, 132a.

Defines: y 14, 22a, 26ab, 32ac, 33, 35b, 37ab, 38d, 40, 43b, 44, 45ab, 46c, 47, 50b, 81d, 85b, 86a, 87a, 88a, 100a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 122b, 123b, 126, 127b, 128b, 131a, 132b, 143, 144.

$\langle C \text{ integer } Q \text{ Input 25e} \rangle \equiv$

**int** Q

◇

Fragment referenced in 26ab, 34, 82b, 83, 84, 85bc, 87ab, 100a, 109b, 160b, 161.

Defines: Q 14, 32c, 33, 35ab, 37abc, 38ab, 40, 44, 45ab, 46c, 47, 48, 49, 51, 55, 56a, 57, 73, 74, 75, 76abc, 78, 80ab, 81ad, 82b, 83, 84, 85b, 86a, 87a, 88a, 100a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 122b, 123b, 126, 127b, 128b, 131a, 141a, 158a, 159, 160a.

$\langle C \text{ real } y \text{ Input 26a} \rangle \equiv$

```
double *y,
 $\langle C \text{ integer } Q \text{ Input 25e} \rangle$ ,
◇
```

Fragment referenced in 81c, 101bc, 106a, 109c, 110b, 111ac, 112a.

Defines: `y` 14, 22a, 25d, 26b, 32ac, 33, 35b, 37ab, 38d, 40, 43b, 44, 45ab, 46c, 47, 50b, 81d, 85b, 86a, 87a, 88a, 100a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 122b, 123b, 126, 127b, 128b, 131a, 132b, 143, 144.

or a factor at  $Q$  levels

$\langle C \text{ integer } y \text{ Input 26b} \rangle \equiv$

```
int *y,
 $\langle C \text{ integer } Q \text{ Input 25e} \rangle$ ,
◇
```

Fragment referenced in 123c, 128c.

Defines: `y` 14, 22a, 25d, 26a, 32ac, 33, 35b, 37ab, 38d, 40, 43b, 44, 45ab, 46c, 47, 50b, 81d, 85b, 86a, 87a, 88a, 100a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 122b, 123b, 126, 127b, 128b, 131a, 132b, 143, 144.

The weights  $w_i, i = 1, \dots, N$

$\langle R \text{ weights Input 26c} \rangle \equiv$

```
SEXP weights
◇
```

Fragment referenced in 31b, 42c, 81c, 85ac, 86b, 87b, 88b, 89b, 91, 92b, 95ac, 99, 100b, 113ac, 117b, 118b, 122a, 123a, 127a, 128a, 132a, 135b.

Defines: `weights` 3b, 4, 5a, 6, 8, 15, 16, 18, 20, 26de, 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 52a, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 95b, 96a, 100a, 102, 103a, 113b, 114a, 118a, 119a, 122b, 123b, 127b, 128b, 132b, 136a.

can be constant one (`XLENGTH(weights) == 0` or `weights = integer(0)`) or integer-valued, with `HAS_WEIGHTS == 0` in the former case

$\langle C \text{ integer weights Input 26d} \rangle \equiv$

```
int *weights,
int HAS_WEIGHTS,
◇
```

Fragment referenced in 97ab, 104ab, 106c, 107a, 115bc, 120bc, 124c, 125a, 129c, 130a.

Defines: `HAS_WEIGHTS` 26e, 98a, 105, 108, 116b, 121b, 126, 131a, `weights`, 4, 6, 8, 16, 20, 26e, 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93a, 95b, 100a, 113b, 118a, 122b, 127b, 132b, 136a.

Uses: `weights` 26c.

Weights larger than `INT_MAX` are stored as double

$\langle C \text{ real weights Input 26e} \rangle \equiv$

```
double *weights,
int HAS_WEIGHTS,
◇
```

Fragment referenced in 96b, 97c, 103b, 104c, 106b, 107b, 115a, 116a, 120a, 121a, 124b, 125b, 129b, 130b.

Defines: `HAS_WEIGHTS` 26d, 98a, 105, 108, 116b, 121b, 126, 131a, `weights`, 4, 6, 8, 16, 20, 26d, 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93a, 95b, 100a, 113b, 118a, 122b, 127b, 132b, 136a.

Uses: `weights` 26c.

The sum of all weights is a double

$\langle C \text{ sumweights Input 27a} \rangle \equiv$

```
double sumweights
◇
```

Fragment referenced in 83, 84, 85c, 87b.

Defines: **sumweights** 34, 36ab, 37abc, 38a, 46bc, 47, 49, 51, 52b, 53a, 74, 75, 76b, 81a, 83, 84, 85b, 86a, 87a, 88a, 136a, 156a.

Subsets  $\mathcal{A} \subseteq \{1, \dots, N\}$  are R style indices

$\langle R \text{ subset Input 27b} \rangle \equiv$

```
SEXP subset
◇
```

Fragment referenced in 31b, 42c, 81c, 85ac, 86b, 87b, 88b, 89b, 91, 92b, 95ac, 99, 100b, 109ac, 113ac, 117b, 118b, 122a, 123a, 127a, 128a, 132a, 133a, 135ab.

Defines: **subset** 3b, 4, 5ab, 6, 8, 15, 16, 18, 20, 27e, 28a, 32ac, 34, 35b, 36ab, 38d, 40, 43b, 44, 46c, 47, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 94b, 95b, 96a, 100a, 102, 103a, 109b, 110a, 111b, 112b, 113b, 114a, 118a, 119a, 122b, 123b, 127b, 128b, 132b, 133b, 135a, 136a, 137ab, 138ab.

are either not existent (**XLENGTH(subset) == 0**) or of length

$\langle C \text{ integer Nsubset Input 27c} \rangle \equiv$

```
R_xlen_t Nsubset
◇
```

Fragment referenced in 27d, 40, 44, 85b, 87a, 89a, 92a, 95b, 100a, 109b, 113b, 118a, 122b, 127b, 137ab, 138b.

Defines: **Nsubset** 36b, 40, 44, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 94ab, 95b, 96a, 98a, 100a, 102, 103a, 109b, 110a, 111b, 112b, 113b, 114a, 118a, 119a, 122b, 123b, 127b, 128b, 137ab, 138b.

Optionally, one can specify a subset of the subset via

$\langle C \text{ subset range Input 27d} \rangle \equiv$

```
R_xlen_t offset,
⟨ C integer Nsubset Input 27c ⟩
◇
```

Fragment referenced in 27e, 28a, 81c, 85c, 87b, 89b, 92b, 95c, 100b, 109c, 113c, 118b, 123a, 128a.

Defines: **offset** 34, 36b, 37abc, 38a, 81d, 86a, 88a, 90, 93ab, 96a, 102, 103a, 110a, 111b, 112b, 114a, 119a, 123b, 128b.

where **offset** is a C style index for **subset**.

Subsets are stored either as integer

$\langle C \text{ integer subset Input 27e} \rangle \equiv$

```
int *subset,
⟨ C subset range Input 27d ⟩
◇
```

Fragment referenced in 97bc, 104bc, 107ab, 111a, 112a, 115c, 116a, 120c, 121a, 125ab, 130ab.

Defines: **subset** 3b, 4, 5ab, 6, 8, 15, 16, 18, 20, 27b, 28a, 32ac, 34, 35b, 36ab, 38d, 40, 43b, 44, 46c, 47, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 94b, 95b, 96a, 100a, 102, 103a, 109b, 110a, 111b, 112b, 113b, 114a, 118a, 119a, 122b, 123b, 127b, 128b, 132b, 133b, 135a, 136a, 137ab, 138ab.

or double (to allow for indices larger than INT\_MAX)

$\langle C \text{ real subset Input 28a} \rangle \equiv$

```
double *subset,
 $\langle C \text{ subset range Input 27d} \rangle$ 
◇
```

Fragment referenced in 96b, 97a, 103b, 104a, 106bc, 110b, 111c, 115ab, 120ab, 124bc, 129bc.

Defines: **subset** 3b, 4, 5ab, 6, 8, 15, 16, 18, 20, 27be, 32ac, 34, 35b, 36ab, 38d, 40, 43b, 44, 46c, 47, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 94b, 95b, 96a, 100a, 102, 103a, 109b, 110a, 111b, 112b, 113b, 114a, 118a, 119a, 122b, 123b, 127b, 128b, 132b, 133b, 135a, 136a, 137ab, 138ab.

Blocks  $\text{block}_i, i = 1, \dots, N$

$\langle R \text{ block Input 28b} \rangle \equiv$

```
SEXP block
◇
```

Fragment referenced in 31b, 42c, 50a, 127a, 132a, 133a, 134b, 135a.

Defines: **block** 3b, 4, 5a, 6, 8, 15, 16, 18, 20, 28d, 32ac, 33, 36ab, 38d, 40, 43b, 44, 45a, 50b, 127b, 128b, 131a, 132b, 133b, 134b, 135a, 155c.

at  $B$  levels

$\langle C \text{ integer } B \text{ Input 28c} \rangle \equiv$

```
int B
◇
```

Fragment referenced in 28d, 34, 160b, 161.

Defines: **B** 32c, 33, 34, 35a, 36a, 40, 44, 45a, 46a, 48, 49, 51, 52b, 73, 74, 78, 127b, 128b, 131a, 141abc, 142, 143, 144, 158a, 159, 160b, 161.

are stored as a factor

$\langle C \text{ integer block Input 28d} \rangle \equiv$

```
int *block,
 $\langle C \text{ integer } B \text{ Input 28c} \rangle$ ,
◇
```

Fragment referenced in 128c.

Defines: **block** 3b, 4, 5a, 6, 8, 15, 16, 18, 20, 28b, 32ac, 33, 36ab, 38d, 40, 43b, 44, 45a, 50b, 127b, 128b, 131a, 132b, 133b, 134b, 135a, 155c.

The tabulation of block (potentially in subsets) is

$\langle R \text{ blockTable Input 28e} \rangle \equiv$

```
SEXP blockTable
◇
```

Fragment referenced in 133a, 134b, 135a.

Defines: **blockTable** 40, 132b, 133b, 134b, 135a.

where the table is of length  $B + 1$  and the first element counts the number of missing values (although these are NOT allowed in block).

### 3.2.1 Example Data and Code

We start with setting-up some toy data sets to be used as test bed. The data over both the 1d and the 2d case, including weights, subsets and blocks.

```
> N <- 20L
> P <- 3L
> Lx <- 10L
> Ly <- 5L
> Q <- 4L
> B <- 2L
> iX2d <- rbind(0, matrix(runif(Lx * P), nrow = Lx))
> ix <- sample(1:Lx, size = N, replace = TRUE)
> levels(ix) <- 1:Lx
> ixf <- factor(ix, levels = 1:Lx, labels = 1:Lx)
> x <- iX2d[ix + 1,]
> Xfactor <- diag(Lx)[ix,]
> iY2d <- rbind(0, matrix(runif(Ly * Q), nrow = Ly))
> iy <- sample(1:Ly, size = N, replace = TRUE)
> levels(iy) <- 1:Ly
> iyf <- factor(iy, levels = 1:Ly, labels = 1:Ly)
> y <- iY2d[iy + 1,]
> weights <- sample(0:5, size = N, replace = TRUE)
> block <- sample(gl(B, ceiling(N / B))[1:N])
> subset <- sort(sample(1:N, floor(N * 1.5), replace = TRUE))
> subsety <- sample(1:N, floor(N * 1.5), replace = TRUE)
> r1 <- rep(1:ncol(x), ncol(y))
> r1Xfactor <- rep(1:ncol(Xfactor), ncol(y))
> r2 <- rep(1:ncol(y), each = ncol(x))
> r2Xfactor <- rep(1:ncol(y), each = ncol(Xfactor))
```

As a benchmark, we implement linear statistics, their expectation and covariance, taking weights, subsets and blocks into account, at R level. In a sense, the core of the **libcoin** package is “just” a less memory-hungry and sometimes faster version of this simple function.

```
> LECV <- function(X, Y, weights = integer(0), subset = integer(0), block = integer(0)) {
+
+   if (length(weights) == 0) weights <- rep(1, NROW(X))
+   if (length(subset) == 0) subset <- 1:NROW(X)
+   idx <- rep(subset, weights[subset])
+   X <- X[idx,,drop = FALSE]
+   Y <- Y[idx,,drop = FALSE]
+   sumweights <- length(idx)
+
+   if (length(block) == 0) {
+     ExpX <- colSums(X)
+     ExpY <- colSums(Y) / sumweights
+     yc <- t(t(Y) - ExpY)
+     CovY <- crossprod(yc) / sumweights
+     CovX <- crossprod(X)
+     Exp <- kronecker(ExpY, ExpX)
+     Cov <- sumweights / (sumweights - 1) * kronecker(CovY, CovX) -
+       1 / (sumweights - 1) * kronecker(CovY, tcrossprod(ExpX))
+
+     ret <- list(LinearStatistic = as.vector(crossprod(X, Y)),
```



```

+             Expectation = as.vector(Exp),
+             Covariance = Cov,
+             Variance = diag(Cov))
+   } else {
+     block <- block[idx]
+     ret <- list(LinearStatistic = 0, Expectation = 0, Covariance = 0, Variance = 0)
+     for (b in levels(block)) {
+       tmp <- LECV(X = X, Y = Y, subset = which(block == b))
+       for (l in names(ret)) ret[[l]] <- ret[[l]] + tmp[[l]]
+     }
+   }
+   return(ret)
+ }

> cmpr <- function(ret1, ret2) {
+   if (inherits(ret1, "LinStatExpCov")) {
+     if (!ret1$varonly)
+       ret1$Covariance <- vcov(ret1)
+   }
+   ret1 <- ret1[!sapply(ret1, is.null)]
+   ret2 <- ret2[!sapply(ret2, is.null)]
+   nm1 <- names(ret1)
+   nm2 <- names(ret2)
+   nm <- c(nm1, nm2)
+   nm <- names(table(nm))[table(nm) == 2]
+   isequal(ret1[nm], ret2[nm])
+ }

```

We now compute the linear statistic along with corresponding expectation, variance and covariance for later reuse.

```

> LECVxyws <- LinStatExpCov(x, y, weights = weights, subset = subset)
> LEVxyws <- LinStatExpCov(x, y, weights = weights, subset = subset, varonly = TRUE)

```

The following tests compare the high-level R implementation (function LECV()) with the 1d and 2d C level implementations in the two situations with and without specification of X (ie, the dummy matrix in the latter case).

```

> ### with X given
> testit <- function(...) {
+   a <- LinStatExpCov(x, y, ...)
+   b <- LECV(x, y, ...)
+   d <- LinStatExpCov(X = iX2d, ix = ix, Y = iY2d, iy = iy, ...)
+   return(cmpr(a, b) && cmpr(d, b))
+ }
> stopifnot(
+   testit() && testit(weights = weights) &&
+   testit(subset = subset) && testit(weights = weights, subset = subset) &&
+   testit(block = block) && testit(weights = weights, block = block) &&
+   testit(subset = subset, block = block) &&
+   testit(weights = weights, subset = subset, block = block))
> ### without dummy matrix X
> testit <- function(...) {
+   a <- LinStatExpCov(X = ix, y, ...)
+   b <- LECV(Xfactor, y, ...)

```

```

+   d <- LinStatExpCov(X = integer(0), ix = ix, Y = iY2d, iy = iy, ...)
+   return(cmp(r(a, b) && cmp(d, b))
+ }
> stopifnot(
+   testit() && testit(weights = weights) &&
+   testit(subset = subset) && testit(weights = weights, subset = subset) &&
+   testit(block = block) && testit(weights = weights, block = block) &&
+   testit(subset = subset, block = block) &&
+   testit(weights = weights, subset = subset, block = block))

```

All three implementations give the same results.

### 3.3 Conventions

Functions starting with `R_` are C functions callable via `.Call()` from R. That means they all return `SEXP`. These functions allocate memory handled by R.

Functions starting with `RC_` are C functions with `SEXP` or pointer arguments and possibly an `SEXP` return value.

Functions starting with `C_` only take pointer arguments and return a scalar nor nothing.

Return values (arguments modified by a function) are named `ans`, sometimes with dimension (for example: `PQ_ans`).

### 3.4 C User Interface

#### 3.4.1 One-Dimensional Case (“1d”)

*⟨ User Interface 31a ⟩* ≡

```

  ⟨ RC_ExpectationCovarianceStatistic 34 ⟩
  ⟨ R_ExpectationCovarianceStatistic 32c ⟩
  ⟨ R_PermutedLinearStatistic 40 ⟩
  ⟨ R_StandardisePermutedLinearStatistic 42a ⟩
  ◇

```

Fragment referenced in 24a.

The data are given as  $\mathbf{x}_i$  and  $\mathbf{y}_i$  for  $i = 1, \dots, N$ , optionally with weights, subset and blocks. The latter three variables are ignored when specified as `integer(0)`.

*⟨ User Interface Input 31b ⟩* ≡

```

  ⟨ R x Input 24d ⟩
  ⟨ R y Input 25d ⟩
  ⟨ R weights Input 26c ⟩,
  ⟨ R subset Input 27b ⟩,
  ⟨ R block Input 28b ⟩,
  ◇

```

Fragment referenced in 32b, 34, 38c.

This function can be called from other packages.

"libcoinAPI.h" 32a≡

```
⟨ C Header 166b ⟩
#include <R_ext/Rdynload.h>
#include <libcoin.h>

extern SEXP libcoin_R_ExpectationCovarianceStatistic(
    SEXP x, SEXP y, SEXP weights, SEXP subset, SEXP block, SEXP varonly,
    SEXP tol
) {
    static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;
    if (fun == NULL)
        fun = (SEXP(*) (SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP))
            R_GetCCallable("libcoin", "R_ExpectationCovarianceStatistic");
    return fun(x, y, weights, subset, block, varonly, tol);
}
◇
```

File defined by 32a, 38d, 41b, 43b, 50b, 54a, 64a, 141b, 145b, 148a, 150a.

Uses: block 28bd, R\_ExpectationCovarianceStatistic 32c, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

⟨ R\_ExpectationCovarianceStatistic Prototype 32b ⟩ ≡

```
SEXP R_ExpectationCovarianceStatistic
(
    ⟨ User Interface Input 31b ⟩
    SEXP varonly,
    SEXP tol
)
◇
```

Fragment referenced in 23b, 32c.

Uses: R\_ExpectationCovarianceStatistic 32c.

The C interface essentially sets-up the necessary memory and calls a C level function for the computations.

⟨ R\_ExpectationCovarianceStatistic 32c ⟩ ≡

```
⟨ R_ExpectationCovarianceStatistic Prototype 32b ⟩
{
    SEXP ans;

    ⟨ Setup Dimensions 33 ⟩

    PROTECT(ans = RC_init_LECV_1d(P, Q, INTEGER(varonly)[0], B, TYPEOF(x) == INTSXP, REAL(tol)[0]));

    RC_ExpectationCovarianceStatistic(x, y, weights, subset, block, ans);

    UNPROTECT(1);
    return(ans);
}
◇
```

Fragment referenced in 31a.

Defines: R\_ExpectationCovarianceStatistic 6, 32ab, 164, 165.

Uses: B 28c, block 28bd, P 25a, Q 25e, RC\_ExpectationCovarianceStatistic 34, 48, RC\_init\_LECV\_1d 160b, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

$P$ ,  $Q$  and  $B$  are first extracted from the data. The case where  $\mathbf{X}$  is an implicitly specified dummy matrix, the dimension  $P$  is the number of levels of  $\mathbf{x}$ .

*⟨ Setup Dimensions 33 ⟩*  $\equiv$

```
int P, Q, B;

if (typeof(x) == INTSXP) {
    P = NLEVELS(x);
} else {
    P = NCOL(x);
}
Q = NCOL(y);

B = 1;
if (LENGTH(block) > 0)
    B = NLEVELS(block);
◇
```

Fragment referenced in [32c](#), [40](#).

Uses:  $B$  [28c](#),  $\mathbf{block}$  [28bd](#),  $\mathbf{NCOL}$  [139c](#),  $\mathbf{NLEVELS}$  [140a](#),  $P$  [25a](#),  $Q$  [25e](#),  $\mathbf{x}$  [24d](#), [25bc](#),  $\mathbf{y}$  [25d](#), [26ab](#).

The core function first computes the linear statistic (as there is no need to pay attention to blocks) and, in a second step, starts a loop over potential blocks.

FIXME:  $\mathbf{x}$  being an integer (Xfactor) with some 0 elements is not handled correctly (as `sumweights` doesn't take this information into account; use `subset` to exclude these missings (as done in `libcoin::LinStatExpCov`))

$\langle RC\_ExpectationCovarianceStatistic\ 34 \rangle \equiv$

```

void RC_ExpectationCovarianceStatistic
(
   $\langle$  User Interface Input 31b  $\rangle$ 
  SEXP ans
) {
   $\langle$  C integer N Input 24c  $\rangle$ ;
   $\langle$  C integer P Input 25a  $\rangle$ ;
   $\langle$  C integer Q Input 25e  $\rangle$ ;
   $\langle$  C integer B Input 28c  $\rangle$ ;
  double *sumweights, *table;
  double *ExpInf, *VarInf, *CovInf, *ExpX, *ExpXtotal, *VarX, *CovX;
  double *tmpV, *tmpCV;
  SEXP nullvec, subset_block;

   $\langle$  Extract Dimensions 35a  $\rangle$ 

   $\langle$  Compute Linear Statistic 35b  $\rangle$ 

   $\langle$  Setup Memory and Subsets in Blocks 36a  $\rangle$ 

  /* start with subset[0] */
  R_xlen_t offset = (R_xlen_t) table[0];

  for (int b = 0; b < B; b++) {

     $\langle$  Compute Sum of Weights in Block 36b  $\rangle$ 

    /* don't do anything for empty blocks or blocks with weight 1 */
    if (sumweights[b] > 1) {

       $\langle$  Compute Expectation Linear Statistic 37a  $\rangle$ 

       $\langle$  Compute Covariance Influence 37b  $\rangle$ 

      if (C_get_varonly(ans)) {
         $\langle$  Compute Variance Linear Statistic 37c  $\rangle$ 
      } else {
         $\langle$  Compute Covariance Linear Statistic 38a  $\rangle$ 
      }
    }

    /* next iteration starts with subset[cumsum(table[1:(b + 1)])] */
    offset += (R_xlen_t) table[b + 1];
  }

   $\langle$  Compute Variance from Covariance 38b  $\rangle$ 

  Free(ExpX); Free(VarX); Free(CovX);
  UNPROTECT(2);
}

```

Fragment referenced in 31a.

Defines: RC\_ExpectationCovarianceStatistic 32c.

Uses: B 28c, C\_get\_varonly 152b, offset 27d, subset 27be, 28a, sumweights 27a.

The dimensions are available from the return object:

$\langle \text{Extract Dimensions 35a} \rangle \equiv$

```
P = C_get_P(ans);
Q = C_get_Q(ans);
N = NROW(x);
B = C_get_B(ans);
◇
```

Fragment referenced in 34.

Uses: B 28c, C\_get\_B 157a, C\_get\_P 151c, C\_get\_Q 152a, N 24bc, NROW 139b, P 25a, Q 25e, x 24d, 25bc.

The linear statistic  $\mathbf{T}(\mathcal{A})$  can be computed without taking blocks into account.

$\langle \text{Compute Linear Statistic 35b} \rangle \equiv$

```
RC_LinearStatistic(x, N, P, REAL(y), Q, weights, subset,
                  Offset0, XLENGTH(subset),
                  C_get_LinearStatistic(ans));
◇
```

Fragment referenced in 34.

Uses: C\_get\_LinearStatistic 152d, N 24bc, Offset0 22b, P 25a, Q 25e, RC\_LinearStatistic 81d, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

We next extract memory from the return object and allocate some additional memory. The most important step is to tabulate blocks and to order the subset with respect to blocks. In absense of block, this just returns subset.

$\langle \text{Setup Memory and Subsets in Blocks 36a} \rangle \equiv$

```

ExpInf = C_get_ExpectationInfluence(ans);
VarInf = C_get_VarianceInfluence(ans);
CovInf = C_get_CovarianceInfluence(ans);
ExpXtotal = C_get_ExpectationX(ans);
for (int p = 0; p < P; p++) ExpXtotal[p] = 0.0;
ExpX = Calloc(P, double);
/* Fix by Joanidis Kristoforos: P > INT_MAX is possible
   for maximally selected statistics (when X is an integer).
   2018-12-13
*/
if (C_get_varonly(ans)) {
    VarX = Calloc(P, double);
    CovX = Calloc(1, double);
} else {
    VarX = Calloc(1, double);
    CovX = Calloc(PP12(P), double);
}
table = C_get_TableBlock(ans);
sumweights = C_get_Sumweights(ans);
PROTECT(nullvec = allocVector(INTSXP, 0));

if (B == 1) {
    table[0] = 0.0;
    table[1] = RC_Sums(N, nullvec, subset, Offset0, XLENGTH(subset));
} else {
    RC_OneTableSums(INTEGER(block), N, B + 1, nullvec, subset, Offset0,
                    XLENGTH(subset), table);
}
if (table[0] > 0)
    error("No missing values allowed in block");
PROTECT(subset_block = RC_order_subset_wrt_block(N, subset, block,
                                                  VECTOR_ELT(ans, TableBlock_SLOT)));
◇

```

Fragment referenced in 34.

Uses: B 28c, block 28bd, C\_get\_CovarianceInfluence 155a, C\_get\_ExpectationInfluence 154c, C\_get\_ExpectationX 154b, C\_get\_Sumweights 156a, C\_get\_TableBlock 155c, C\_get\_VarianceInfluence 155b, C\_get\_varonly 152b, N 24bc, Offset0 22b, P 25a, PP12 140b, RC\_OneTableSums 119a, RC\_order\_subset\_wrt\_block 133b, RC\_Sums 96a, subset 27be, 28a, sumweights 27a, TableBlock\_SLOT 22b.

We compute  $\mu(\mathcal{A})$  based on  $\mathbb{E}(h \mid S(\mathcal{A}))$  and  $\sum_{i \in \mathcal{A}} w_i \mathbf{x}_i$  for the subset given by subset and the  $b$ th level of block. The expectation is initialised zero when  $b = 0$  and values add-up over blocks.

$\langle \text{Compute Sum of Weights in Block 36b} \rangle \equiv$

```

/* compute sum of weights in block b of subset */
if (table[b + 1] > 0) {
    sumweights[b] = RC_Sums(N, weights, subset_block,
                           offset, (R_xlen_t) table[b + 1]);
} else {
    /* offset = something and Nsubset = 0 means Nsubset = N in
       RC_Sums; catch empty or zero-weight block levels here */
    sumweights[b] = 0.0;
}

```

Fragment referenced in 34.

Uses: block 28bd, N 24bc, Nsubset 27c, offset 27d, RC\_Sums 96a, subset 27be, 28a, sumweights 27a, weights 26c, weights, 26de.

$\langle \text{Compute Expectation Linear Statistic 37a} \rangle \equiv$

```

RC_ExpectationInfluence(N, y, Q, weights, subset_block, offset,
                        (R_xlen_t) table[b + 1], sumweights[b], ExpInf + b * Q);
RC_ExpectationX(x, N, P, weights, subset_block, offset,
                (R_xlen_t) table[b + 1], ExpX);
for (int p = 0; p < P; p++) ExpXtotal[p] += ExpX[p];
C_ExpectationLinearStatistic(P, Q, ExpInf + b * Q, ExpX, b,
                             C_get_Expectation(ans));

```

Fragment referenced in 34.

Uses: C\_ExpectationLinearStatistic 82b, C\_get\_Expectation 153a, N 24bc, offset 27d, P 25a, Q 25e, RC\_ExpectationInfluence 86a, RC\_ExpectationX 90, sumweights 27a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

The covariance  $\mathbb{V}(h \mid S(\mathcal{A}))$  is now computed for the subset given by subset and the  $b$ th level of block. Note that CovInf stores the values for each block in the return object (for later reuse).

$\langle \text{Compute Covariance Influence 37b} \rangle \equiv$

```

/* C_ordered_Xfactor and C_unordered_Xfactor need both VarInf and CovInf */
RC_CovarianceInfluence(N, y, Q, weights, subset_block, offset,
                      (R_xlen_t) table[b + 1], ExpInf + b * Q, sumweights[b],
                      !DoVarOnly, CovInf + b * Q * (Q + 1) / 2);
/* extract variance from covariance */
tmpCV = CovInf + b * Q * (Q + 1) / 2;
tmpV = VarInf + b * Q;
for (int q = 0; q < Q; q++) tmpV[q] = tmpCV[S(q, q, Q)];

```

Fragment referenced in 34.

Uses: C\_ordered\_Xfactor 73, C\_unordered\_Xfactor 78, DoVarOnly 22b, N 24bc, offset 27d, Q 25e, RC\_CovarianceInfluence 88a, S 22a, sumweights 27a, weights 26c, weights, 26de, y 25d, 26ab.

We can now compute the variance or covariance of the linear statistic  $\Sigma(\mathcal{A})$ :



*⟨ Compute Variance Linear Statistic 37c ⟩ ≡*

```
RC_CovarianceX(x, N, P, weights, subset_block, offset,
               (R_xlen_t) table[b + 1], ExpX, DoVarOnly, VarX);
C_VarianceLinearStatistic(P, Q, VarInf + b * Q, ExpX, VarX, sumweights[b],
                          b, C_get_Variance(ans));
```

◇

Fragment referenced in 34.

Uses: C\_get\_Variance 153b, C\_VarianceLinearStatistic 84, DoVarOnly 22b, N 24bc, offset 27d, P 25a, Q 25e, RC\_CovarianceX 93a, sumweights 27a, weights 26c, weights, 26de, x 24d, 25bc.

*⟨ Compute Covariance Linear Statistic 38a ⟩ ≡*

```
RC_CovarianceX(x, N, P, weights, subset_block, offset,
               (R_xlen_t) table[b + 1], ExpX, !DoVarOnly, CovX);
C_CovarianceLinearStatistic(P, Q, CovInf + b * Q * (Q + 1) / 2,
                           ExpX, CovX, sumweights[b], b,
                           C_get_Covariance(ans));
```

◇

Fragment referenced in 34.

Uses: C\_CovarianceLinearStatistic 83, C\_get\_Covariance 154a, DoVarOnly 22b, N 24bc, offset 27d, P 25a, Q 25e, RC\_CovarianceX 93a, sumweights 27a, weights 26c, weights, 26de, x 24d, 25bc.

*⟨ Compute Variance from Covariance 38b ⟩ ≡*

```
/* always return variances */
if (!C_get_varonly(ans)) {
    for (int p = 0; p < mPQB(P, Q, 1); p++)
        C_get_Variance(ans)[p] = C_get_Covariance(ans)[S(p, p, mPQB(P, Q, 1))];
}
```

◇

Fragment referenced in 34.

Uses: C\_get\_Covariance 154a, C\_get\_Variance 153b, C\_get\_varonly 152b, mPQB 141a, P 25a, Q 25e, S 22a.

The computation of permuted linear statistics is done outside this general function. The user interface is the same, except for an additional number of permutations to be specified.

*⟨ R\_PermutedLinearStatistic Prototype 38c ⟩ ≡*

```
SEXP R_PermutedLinearStatistic
(
    ⟨ User Interface Input 31b ⟩
    SEXP nresample
)
```

◇

Fragment referenced in 23b, 40.

Uses: R\_PermutedLinearStatistic 40.

"libcoinAPI.h" 38d≡

```
extern SEXP libcoin_R_PermutedLinearStatistic(  
  SEXP x, SEXP y, SEXP weights, SEXP subset, SEXP block, SEXP nresample  
) {  
  static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;  
  if (fun == NULL)  
    fun = (SEXP*)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP)  
      R_GetCCallable("libcoin", "R_PermutedLinearStatistic");  
  return fun(x, y, weights, subset, block, nresample);  
}  
◇
```

File defined by 32a, 38d, 41b, 43b, 50b, 54a, 64a, 141b, 145b, 148a, 150a.

Uses: block 28bd, R\_PermutedLinearStatistic 40, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

The dimensions are extracted from the data in the same ways as above. The function differentiates between the absense and presense of blocks. Weights are removed by expanding subset accordingly. Once within-block permutations were set-up the Kronecker product of **X** and **Y** is computed. Note that this function returns the matrix of permuted linear statistics; the R interface assigns this matrix to the corresponding element of the **LinStatExpCov** object (because we are not allowed to modify existing R objects at C level).

$\langle R\_PermutedLinearStatistic\ 40 \rangle \equiv$

```

 $\langle R\_PermutedLinearStatistic\ Prototype\ 38c \rangle$ 
{
  SEXP ans, expand_subset, block_subset, perm, tmp, blockTable;
  double *linstat;
  int PQ;
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
   $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;
  R_xlen_t inresample;

   $\langle Setup\ Dimensions\ 33 \rangle$ 
  PQ = mPQB(P, Q, 1);
  N = NROW(y);
  inresample = (R_xlen_t) REAL(nresample)[0];

  PROTECT(ans = allocMatrix(REALSXP, PQ, inresample));
  PROTECT(expand_subset = RC_setup_subset(N, weights, subset));
  Nsubset = XLENGTH(expand_subset);
  PROTECT(tmp = allocVector(REALSXP, Nsubset));
  PROTECT(perm = allocVector(REALSXP, Nsubset));

  GetRNGstate();
  if (B == 1) {
    for (R_xlen_t np = 0; np < inresample; np++) {
       $\langle Setup\ Linear\ Statistic\ 41a \rangle$ 
      C_doPermute(REAL(expand_subset), Nsubset, REAL(tmp), REAL(perm));
      RC_KronSums_Permutation(x, NROW(x), P, REAL(y), Q, expand_subset,
                             Offset0, Nsubset, perm, linstat);
    }
  } else {
    PROTECT(blockTable = allocVector(REALSXP, B + 1));
    /* same as RC_OneTableSums(block, noweights, expand_subset) */
    RC_OneTableSums(INTEGER(block), XLENGTH(block), B + 1, weights, subset, Offset0,
                    XLENGTH(subset), REAL(blockTable));
    PROTECT(block_subset = RC_order_subset_wrt_block(XLENGTH(block), expand_subset,
                                                       block, blockTable));

    for (R_xlen_t np = 0; np < inresample; np++) {
       $\langle Setup\ Linear\ Statistic\ 41a \rangle$ 
      C_doPermuteBlock(REAL(block_subset), Nsubset, REAL(blockTable),
                       B + 1, REAL(tmp), REAL(perm));
      RC_KronSums_Permutation(x, NROW(x), P, REAL(y), Q, block_subset,
                             Offset0, Nsubset, perm, linstat);
    }
    UNPROTECT(2);
  }
  PutRNGstate();

  UNPROTECT(4);
  return(ans);
}

```

Fragment referenced in 31a.

Defines:  $R\_PermutedLinearStatistic$  6, 38cd, 164, 165.

Uses: B 28c, block 28bd, blockTable 28e, C\_doPermute 137b, C\_doPermuteBlock 138b, mPQB 141a, N 24bc, NROW 139b, Nsubset 27c, Offset0 22b, P 25a, Q 25e, RC\_KronSums\_Permutation 110a, RC\_OneTableSums 119a, RC\_order\_subset\_wrt\_block 133b, RC\_setup\_subset 136a, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

$\langle \textit{Setup Linear Statistic 41a} \rangle \equiv$

```

    if (np % 256 == 0) R_CheckUserInterrupt();
    linstat = REAL(ans) + PQ * np;
    for (int p = 0; p < PQ; p++)
        linstat[p] = 0.0;
    ◇

```

Fragment referenced in [40](#), [51](#).

"libcoinAPI.h" 41b $\equiv$

```

extern SEXP libcoin_R_StandardisePermutedLinearStatistic(
    SEXP LECV
) {
    static SEXP(*fun)(SEXP) = NULL;
    if (fun == NULL)
        fun = (SEXP(*) (SEXP))
            R_GetCCallable("libcoin", "R_StandardisePermutedLinearStatistic");
    return fun(LECV);
}
◇

```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).  
 Uses: LECV [151b](#).

This small function takes an object containing permuted linear statistics and returns the matrix of standardised linear statistics.

$\langle \textit{R_StandardisePermutedLinearStatistic Prototype 41c} \rangle \equiv$

```

SEXP R_StandardisePermutedLinearStatistic
(
    SEXP LECV
)
◇

```

Fragment referenced in [23b](#), [42a](#).  
 Uses: LECV [151b](#).

$\langle R\_StandardisePermutedLinearStatistic\ 42a \rangle \equiv$

```

 $\langle R\_StandardisePermutedLinearStatistic\ Prototype\ 41c \rangle$ 
{
    SEXP ans;
    R_xlen_t nresample = C_get_nresample(LECV);
    double *ls;
    if (!nresample) return(R_NilValue);
    int PQ = C_get_P(LECV) * C_get_Q(LECV);

    PROTECT(ans = allocMatrix(REALSXP, PQ, nresample));

    for (R_xlen_t np = 0; np < nresample; np++) {
        ls = REAL(ans) + PQ * np;
        /* copy first; standarisation is in place */
        for (int p = 0; p < PQ; p++)
            ls[p] = C_get_PermutedLinearStatistic(LECV)[p + PQ * np];
        if (C_get_varonly(LECV)) {
            C_standardise(PQ, ls, C_get_Expectation(LECV),
                          C_get_Variance(LECV), 1, C_get_tol(LECV));
        } else {
            C_standardise(PQ, ls, C_get_Expectation(LECV),
                          C_get_Covariance(LECV), 0, C_get_tol(LECV));
        }
    }
    UNPROTECT(1);
    return(ans);
}

```

Fragment referenced in [31a](#).

Uses: [C\\_get\\_Covariance 154a](#), [C\\_get\\_Expectation 153a](#), [C\\_get\\_nresample 157b](#), [C\\_get\\_P 151c](#),  
[C\\_get\\_PermutedLinearStatistic 157c](#), [C\\_get\\_Q 152a](#), [C\\_get\\_tol 157d](#), [C\\_get\\_Variance 153b](#), [C\\_get\\_varonly 152b](#),  
[C\\_standardise 67a](#), [LECV 151b](#).

### 3.4.2 Two-Dimensional Case (“2d”)

$\langle 2d\ User\ Interface\ 42b \rangle \equiv$

```

 $\langle RC\_ExpectationCovarianceStatistic\_2d\ 48 \rangle$ 
 $\langle R\_ExpectationCovarianceStatistic\_2d\ 44 \rangle$ 
 $\langle R\_PermutedLinearStatistic\_2d\ 51 \rangle$ 

```

Fragment referenced in [24a](#).

$\langle 2d \text{ User Interface Input } 42c \rangle \equiv$

```

     $\langle R \ x \text{ Input } 24d \rangle$ 
    SEXP ix,
     $\langle R \ y \text{ Input } 25d \rangle$ 
    SEXP iy,
     $\langle R \ weights \text{ Input } 26c \rangle$ ,
     $\langle R \ subset \text{ Input } 27b \rangle$ ,
     $\langle R \ block \text{ Input } 28b \rangle$ ,
     $\diamond$ 

```

Fragment referenced in [43a](#), [48](#).

$\langle R\_ExpectationCovarianceStatistic\_2d \text{ Prototype } 43a \rangle \equiv$

```

    SEXP R_ExpectationCovarianceStatistic_2d
    (
         $\langle 2d \text{ User Interface Input } 42c \rangle$ 
        SEXP varonly,
        SEXP tol
    )
     $\diamond$ 

```

Fragment referenced in [23b](#), [44](#).

Uses: [R\\_ExpectationCovarianceStatistic\\_2d](#) [44](#).

"libcoinAPI.h" [43b](#)  $\equiv$

```

extern SEXP libcoin_R_ExpectationCovarianceStatistic_2d(
    SEXP x, SEXP ix, SEXP y, SEXP iy, SEXP weights, SEXP subset, SEXP block,
    SEXP varonly, SEXP tol
) {
    static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;
    if (fun == NULL)
        fun = (SEXP*)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP)
        R_GetCCallable("libcoin", "R_ExpectationCovarianceStatistic_2d");
    return fun(x, ix, y, iy, weights, subset, block, varonly, tol);
}
 $\diamond$ 

```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).

Uses: [block](#) [28bd](#), [R\\_ExpectationCovarianceStatistic\\_2d](#) [44](#), [subset](#) [27be](#), [28a](#), [weights](#) [26c](#), [weights](#), [26de](#), [x](#) [24d](#), [25bc](#), [y](#) [25d](#), [26ab](#).

$\langle R\_ExpectationCovarianceStatistic\_2d\ 44 \rangle \equiv$

```

 $\langle R\_ExpectationCovarianceStatistic\_2d\ Prototype\ 43a \rangle$ 
{
    SEXP ans;
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;
    int Xfactor;

    N = XLENGTH(ix);
    Nsubset = XLENGTH(subset);
    Xfactor = XLENGTH(x) == 0;

     $\langle Setup\ Dimensions\ 2d\ 45a \rangle$ 

    PROTECT(ans = RC_init_LECV_2d(P, Q, INTEGER(varonly)[0],
                                   Lx, Ly, B, Xfactor, REAL(tol)[0]));

    if (B == 1) {
        RC_TwoTableSums(INTEGER(ix), N, Lx + 1, INTEGER(iy), Ly + 1,
                        weights, subset, Offset0, Nsubset,
                        C_get_Table(ans));
    } else {
        RC_ThreeTableSums(INTEGER(ix), N, Lx + 1, INTEGER(iy), Ly + 1,
                          INTEGER(block), B, weights, subset, Offset0, Nsubset,
                          C_get_Table(ans));
    }
    RC_ExpectationCovarianceStatistic_2d(x, ix, y, iy, weights,
                                         subset, block, ans);

    UNPROTECT(1);
    return(ans);
}

```

Fragment referenced in 42b.

Defines: `R_ExpectationCovarianceStatistic_2d` 8, 43ab, 164, 165.

Uses: `B` 28c, `block` 28bd, `C_get_Table` 156b, `N` 24bc, `Nsubset` 27c, `Offset0` 22b, `P` 25a, `Q` 25e, `RC_init_LECV_2d` 161, `RC_ThreeTableSums` 128b, `RC_TwoTableSums` 123b, `subset` 27be, 28a, `weights` 26c, `weights`, 26de, `x` 24d, 25bc, `y` 25d, 26ab.

⟨ *Setup Dimensions 2d 45a* ⟩ ≡

```

int P, Q, B, Lx, Ly;

if (XLENGTH(x) == 0) {
    P = NLEVELS(ix);
} else {
    P = NCOL(x);
}
Q = NCOL(y);

B = 1;
if (XLENGTH(block) > 0)
    B = NLEVELS(block);

Lx = NLEVELS(ix);
Ly = NLEVELS(iy);
◇

```

Fragment referenced in [44](#), [51](#).

Uses: B [28c](#), block [28bd](#), NCOL [139c](#), NLEVELS [140a](#), P [25a](#), Q [25e](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

⟨ *Linear Statistic 2d 45b* ⟩ ≡

```

if (Xfactor) {
    for (int j = 1; j < Lyp1; j++) { /* j = 0 means NA */
        for (int i = 1; i < Lxp1; i++) { /* i = 0 means NA */
            for (int q = 0; q < Q; q++)
                linstat[q * (Lxp1 - 1) + (i - 1)] +=
                    btab[j * Lxp1 + i] * REAL(y)[q * Lyp1 + j];
        }
    }
} else {
    for (int p = 0; p < P; p++) {
        for (int q = 0; q < Q; q++) {
            int qPp = q * P + p;
            int qLy = q * Lyp1;
            for (int i = 0; i < Lxp1; i++) {
                int pLxi = p * Lxp1 + i;
                for (int j = 0; j < Lyp1; j++)
                    linstat[qPp] += REAL(y)[qLy + j] * REAL(x)[pLxi] * btab[j * Lxp1 + i];
            }
        }
    }
}
◇

```

Fragment referenced in [48](#), [53a](#).

Uses: P [25a](#), Q [25e](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).



$\langle 2d \text{ Total Table } 46a \rangle \equiv$

```

for (int i = 0; i < Lxp1 * Lyp1; i++)
  table2d[i] = 0.0;
for (int b = 0; b < B; b++) {
  for (int i = 0; i < Lxp1; i++) {
    for (int j = 0; j < Lyp1; j++)
      table2d[j * Lxp1 + i] += table[b * Lxp1 * Lyp1 + j * Lxp1 + i];
  }
}

```

Fragment referenced in [48](#).

Uses: [B 28c](#).

$\langle \text{Col Row Total Sums } 46b \rangle \equiv$

```

/* Remember: first row / column count NAs */
/* column sums */
for (int q = 1; q < Lyp1; q++) {
  csum[q] = 0;
  for (int p = 1; p < Lxp1; p++)
    csum[q] += btab[q * Lxp1 + p];
}
csum[0] = 0; /* NA */
/* row sums */
for (int p = 1; p < Lxp1; p++) {
  rsum[p] = 0;
  for (int q = 1; q < Lyp1; q++)
    rsum[p] += btab[q * Lxp1 + p];
}
rsum[0] = 0; /* NA */
/* total sum */
sumweights[b] = 0;
for (int i = 1; i < Lxp1; i++) sumweights[b] += rsum[i];

```

Fragment referenced in [48](#), [51](#).

Uses: [sumweights 27a](#).

$\langle 2d \text{ Expectation } 46c \rangle \equiv$

```

RC_ExpectationInfluence(NROW(y), y, Q, Rcsum, subset, Offset0, 0, sumweights[b], ExpInf);

if (LENGTH(x) == 0) {
  for (int p = 0; p < P; p++)
    ExpX[p] = rsum[p + 1];
} else {
  RC_ExpectationX(x, NROW(x), P, Rrsum, subset, Offset0, 0, ExpX);
}

C_ExpectationLinearStatistic(P, Q, ExpInf, ExpX, b, C_get_Expectation(ans));

```

Fragment referenced in [48](#).

Uses: [C\\_ExpectationLinearStatistic 82b](#), [C\\_get\\_Expectation 153a](#), [NROW 139b](#), [Offset0 22b](#), [P 25a](#), [Q 25e](#),

[RC\\_ExpectationInfluence 86a](#), [RC\\_ExpectationX 90](#), [subset 27be, 28a](#), [sumweights 27a](#), [x 24d, 25bc](#), [y 25d, 26ab](#).

$\langle 2d \text{ Covariance } 47 \rangle \equiv$

```

/* C_ordered_Xfactor needs both VarInf and CovInf */
RC_CovarianceInfluence(NROW(y), y, Q, Rcsum, subset, Offset0, 0, ExpInf, sumweights[b],
    !DoVarOnly, C_get_CovarianceInfluence(ans));
for (int q = 0; q < Q; q++)
    C_get_VarianceInfluence(ans)[q] = C_get_CovarianceInfluence(ans)[S(q, q, Q)];

if (C_get_varonly(ans)) {
    if (LENGTH(x) == 0) {
        for (int p = 0; p < P; p++) CovX[p] = ExpX[p];
    } else {
        RC_CovarianceX(x, NROW(x), P, Rrsum, subset, Offset0, 0, ExpX, DoVarOnly, CovX);
    }
    C_VarianceLinearStatistic(P, Q, C_get_VarianceInfluence(ans),
        ExpX, CovX, sumweights[b], b,
        C_get_Variance(ans));
} else {
    if (LENGTH(x) == 0) {
        for (int p = 0; p < PP12(P); p++) CovX[p] = 0.0;
        for (int p = 0; p < P; p++) CovX[S(p, p, P)] = ExpX[p];
    } else {
        RC_CovarianceX(x, NROW(x), P, Rrsum, subset, Offset0, 0, ExpX, !DoVarOnly, CovX);
    }
    C_CovarianceLinearStatistic(P, Q, C_get_CovarianceInfluence(ans),
        ExpX, CovX, sumweights[b], b,
        C_get_Covariance(ans));
}
◇

```

Fragment referenced in 48.

Uses: C\_CovarianceLinearStatistic 83, C\_get\_Covariance 154a, C\_get\_CovarianceInfluence 155a, C\_get\_Variance 153b, C\_get\_VarianceInfluence 155b, C\_get\_varonly 152b, C\_ordered\_Xfactor 73, C\_VarianceLinearStatistic 84, DoVarOnly 22b, NROW 139b, Offset0 22b, P 25a, PP12 140b, Q 25e, RC\_CovarianceInfluence 88a, RC\_CovarianceX 93a, S 22a, subset 27be, 28a, sumweights 27a, x 24d, 25bc, y 25d, 26ab.

$\langle RC\_ExpectationCovarianceStatistic\_2d\ 48 \rangle \equiv$

```

void RC_ExpectationCovarianceStatistic_2d
(
   $\langle 2d\ User\ Interface\ Input\ 42c \rangle$ 
  SEXP ans
) {
   $\langle 2d\ Memory\ 49 \rangle$ 

   $\langle 2d\ Total\ Table\ 46a \rangle$ 

  linstat = C_get_LinearStatistic(ans);
  for (int p = 0; p < mPQB(P, Q, 1); p++)
    linstat[p] = 0.0;

  for (int b = 0; b < B; b++) {
    btab = table + Lxp1 * Lyp1 * b;

     $\langle Linear\ Statistic\ 2d\ 45b \rangle$ 

     $\langle Col\ Row\ Total\ Sums\ 46b \rangle$ 

     $\langle 2d\ Expectation\ 46c \rangle$ 

     $\langle 2d\ Covariance\ 47 \rangle$ 
  }

  /* always return variances */
  if (!C_get_varonly(ans)) {
    for (int p = 0; p < mPQB(P, Q, 1); p++)
      C_get_Variance(ans)[p] = C_get_Covariance(ans)[S(p, p, mPQB(P, Q, 1))];
  }

  Free(CovX);
  Free(table2d);
  UNPROTECT(2);
}

```

Fragment referenced in 42b.

Defines: RC\_ExpectationCovarianceStatistic 32c, 34.

Uses: B 28c, C\_get\_Covariance 154a, C\_get\_LinearStatistic 152d, C\_get\_Variance 153b, C\_get\_varonly 152b, mPQB 141a, P 25a, Q 25c, S 22a.

⟨ 2d Memory 49 ⟩ ≡

```

SEXP Rcsum, Rrsum;
int P, Q, Lxp1, Lyp1, B, Xfactor;
double *ExpInf, *ExpX, *CovX;
double *table, *table2d, *csum, *rsum, *sumweights, *btabs, *linstat;

P = C_get_P(ans);
Q = C_get_Q(ans);

ExpInf = C_get_ExpectationInfluence(ans);
ExpX = C_get_ExpectationX(ans);
table = C_get_Table(ans);
sumweights = C_get_Sumweights(ans);

Lxp1 = C_get_dimTable(ans)[0];
Lyp1 = C_get_dimTable(ans)[1];
B = C_get_B(ans);
Xfactor = C_get_Xfactor(ans);

if (C_get_varonly(ans)) {
    CovX = Calloc(P, double);
} else {
    CovX = Calloc(PP12(P), double);
}

table2d = Calloc(Lxp1 * Lyp1, double);
PROTECT(Rcsum = allocVector(REALSXP, Lyp1));
csum = REAL(Rcsum);
PROTECT(Rrsum = allocVector(REALSXP, Lxp1));
rsum = REAL(Rrsum);

```

◇

Fragment referenced in 48.

Uses: B [28c](#), C\_get\_B [157a](#), C\_get\_dimTable [156c](#), C\_get\_ExpectationInfluence [154c](#), C\_get\_ExpectationX [154b](#),  
C\_get\_P [151c](#), C\_get\_Q [152a](#), C\_get\_Sumweights [156a](#), C\_get\_Table [156b](#), C\_get\_varonly [152b](#), C\_get\_Xfactor [152c](#),  
P [25a](#), PP12 [140b](#), Q [25e](#), sumweights [27a](#).

```

> LinStatExpCov(X = iX2d, ix = ix, Y = iY2d, iy = iy,
+               weights = weights, subset = subset, nresample = 10)$PermutedLinearStatistic

      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
[1,] 15.486226 15.432786 15.474636 15.434733 15.515989 15.421226 15.523577
[2,]  7.864472  7.948006  8.105228  8.390763  8.212044  8.493673  8.415919
[3,] 12.398382 12.290212 11.905712 12.506639 12.143855 12.549147 12.590900
[4,] 31.244688 31.476627 31.257920 31.264541 31.096744 31.405390 31.259421
[5,] 17.500047 17.688850 17.055915 15.065147 16.586396 15.315949 16.382058
[6,] 24.466142 24.647923 25.464893 24.239801 25.488434 24.296588 23.694321
[7,] 43.423057 43.421097 43.330443 43.612924 43.424099 43.430492 43.309780
[8,] 24.311651 23.474319 22.844531 23.490053 23.541204 22.749540 22.388328
[9,] 34.180046 34.430423 35.397534 33.988759 34.366957 33.293748 33.389741
[10,] 13.461330 13.490553 13.492064 13.434007 13.447127 13.491634 13.476994
[11,]  6.973432  7.169633  7.259611  6.943487  7.017767  7.094398  7.241183
[12,] 10.672723 10.658055 10.574382 10.675107 10.743783 10.837748 10.788257

      [,8]      [,9]     [,10]
[1,] 15.434192 15.491818 15.398248
[2,]  7.834800  8.223809  7.925817

```

```

[3,] 12.362877 11.997518 12.169918
[4,] 31.510352 31.284964 31.576643
[5,] 18.211173 16.969768 17.197270
[6,] 23.773081 25.183959 24.742788
[7,] 43.375471 43.374905 43.496870
[8,] 23.445718 22.372659 23.729797
[9,] 34.264857 35.341197 34.487409
[10,] 13.498456 13.473376 13.482370
[11,] 7.221054 7.329256 7.097392
[12,] 10.669965 10.540119 10.702889

```

$\langle R\_PermutedLinearStatistic\_2d \text{ Prototype } 50a \rangle \equiv$

```

SEXP R_PermutedLinearStatistic_2d
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
  SEXP ix,
   $\langle R \text{ } y \text{ Input } 25d \rangle$ 
  SEXP iy,
   $\langle R \text{ block Input } 28b \rangle$ ,
  SEXP nresample,
  SEXP itable
)
◇

```

Fragment referenced in [23b](#), [51](#).

Uses: [R\\_PermutedLinearStatistic\\_2d 51](#).

"libcoinAPI.h" 50b $\equiv$

```

extern SEXP libcoin_R_PermutedLinearStatistic_2d(
  SEXP x, SEXP ix, SEXP y, SEXP iy, SEXP block, SEXP nresample,
  SEXP itable
) {
  static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;
  if (fun == NULL)
    fun = (SEXP(*) (SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP))
      R_GetCCallable("libcoin", "R_PermutedLinearStatistic_2d");
  return fun(x, ix, y, iy, block, nresample, itable);
}
◇

```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).

Uses: [block 28bd](#), [R\\_PermutedLinearStatistic\\_2d 51](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

$\langle R\_PermutedLinearStatistic\_2d\ 51 \rangle \equiv$

$\langle R\_PermutedLinearStatistic\_2d\ Prototype\ 50a \rangle$

```
{
  SEXP ans, Ritable;
  int *csum, *rsum, *sumweights, *jwork, *table, *rtable2, maxn = 0, Lxp1, Lyp1, *btab, PQ, Xfactor;
  R_xlen_t inresample;
  double *fact, *linstat;

  ⟨ Setup Dimensions 2d 45a ⟩

  PQ = mPQB(P, Q, 1);
  Xfactor = XLENGTH(x) == 0;
  Lxp1 = Lx + 1;
  Lyp1 = Ly + 1;
  inresample = (R_xlen_t) REAL(nresample)[0];

  PROTECT(ans = allocMatrix(REALSXP, PQ, inresample));

  ⟨ Setup Working Memory 52b ⟩

  ⟨ Convert Table to Integer 52a ⟩

  for (int b = 0; b < B; b++) {
    btab = INTEGER(Ritable) + Lxp1 * Lyp1 * b;
    ⟨ Col Row Total Sums 46b ⟩
    if (sumweights[b] > maxn) maxn = sumweights[b];
  }

  ⟨ Setup Log-Factorials 52c ⟩

  GetRNGstate();

  for (R_xlen_t np = 0; np < inresample; np++) {
    ⟨ Setup Linear Statistic 41a ⟩

    for (int p = 0; p < Lxp1 * Lyp1; p++)
      table[p] = 0;

    for (int b = 0; b < B; b++) {
      ⟨ Compute Permuted Linear Statistic 2d 53a ⟩
    }
  }

  PutRNGstate();

  Free(csum); Free(rsum); Free(sumweights); Free(rtable2);
  Free(jwork); Free(fact); Free(table);
  UNPROTECT(2);
  return(ans);
}
◇
```

Fragment referenced in [42b](#).

Defines: [R\\_PermutedLinearStatistic\\_2d 8](#), [50ab](#), [52a](#), [164](#), [165](#).

Uses: [B 28c](#), [mPQB 141a](#), [P 25a](#), [Q 25e](#), [sumweights 27a](#), [x 24d](#), [25bc](#).

*⟨ Convert Table to Integer 52a ⟩ ≡*

```
PROTECT(Ritable = allocVector(INTSXP, LENGTH(itable)));
for (int i = 0; i < LENGTH(itable); i++) {
  if (REAL(itable)[i] > INT_MAX)
    error("cannot deal with weights larger INT_MAX in R_PermutedLinearStatistic_2d");
  INTEGER(Ritable)[i] = (int) REAL(itable)[i];
}
◇
```

Fragment referenced in [51](#).

Uses: `R_PermutedLinearStatistic_2d` [51](#), `weights` [26c](#).

*⟨ Setup Working Memory 52b ⟩ ≡*

```
csum = Calloc(Lyp1 * B, int);
rsum = Calloc(Lxp1 * B, int);
sumweights = Calloc(B, int);
table = Calloc(Lxp1 * Lyp1, int);
rtable2 = Calloc(Lx * Ly , int);
jwork = Calloc(Lyp1, int);
◇
```

Fragment referenced in [51](#).

Uses: `B` [28c](#), `sumweights` [27a](#).

*⟨ Setup Log-Factorials 52c ⟩ ≡*

```
fact = Calloc(maxn + 1, double);
/* Calculate log-factorials. fact[i] = lgamma(i+1) */
fact[0] = fact[1] = 0.;
for (int j = 2; j <= maxn; j++)
  fact[j] = fact[j - 1] + log(j);
◇
```

Fragment referenced in [51](#).

Note: the interface to `S_rcont2` changed in R 4.1-0.

$\langle \text{Compute Permuted Linear Statistic 2d 53a} \rangle \equiv$

```
#if defined(R_VERSION) && R_VERSION >= R_Version(4, 1, 0)
    S_rcont2(Lx, Ly,
             rsum + Lxp1 * b + 1,
             csum + Lyp1 * b + 1,
             sumweights[b], fact, jwork, rtable2);
#else
    S_rcont2(&Lx, &Ly, rsum + Lxp1 * b + 1,
             csum + Lyp1 * b + 1, sumweights + b, fact, jwork, rtable2);
#endif

for (int j1 = 1; j1 <= Lx; j1++) {
    for (int j2 = 1; j2 <= Ly; j2++)
        table[j2 * Lxp1 + j1] = rtable2[(j2 - 1) * Lx + (j1 - 1)];
}
btab = table;
 $\langle \text{Linear Statistic 2d 45b} \rangle$ 
 $\diamond$ 
```

Fragment referenced in 51.

Uses: `sumweights` 27a.

## 3.5 Tests

$\langle \text{Tests 53b} \rangle \equiv$

```
 $\langle R\_QuadraticTest \text{ 55} \rangle$ 
 $\langle R\_MaximumTest \text{ 57} \rangle$ 
 $\langle R\_MaximallySelectedTest \text{ 59} \rangle$ 
 $\diamond$ 
```

Fragment referenced in 24a.



"libcoinAPI.h" 54a≡

```

extern SEXP libcoin_R_QuadraticTest(
  SEXP LEV, SEXP pvalue, SEXP lower, SEXP give_log, SEXP PermutedStatistics
) {
  static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;
  if (fun == NULL)
    fun = (SEXP*)(SEXP, SEXP, SEXP, SEXP, SEXP)
      R_GetCCallable("libcoin", "R_QuadraticTest");
  return fun(LEV, pvalue, lower, give_log, PermutedStatistics);
}

extern SEXP libcoin_R_MaximumTest(
  SEXP LEV, SEXP alternative, SEXP pvalue, SEXP lower, SEXP give_log,
  SEXP PermutedStatistics, SEXP maxpts, SEXP releps, SEXP abseps
) {
  static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;
  if (fun == NULL)
    fun = (SEXP*)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP)
      R_GetCCallable("libcoin", "R_MaximumTest");
  return fun(LEV, alternative, pvalue, lower, give_log, PermutedStatistics, maxpts, releps,
    abseps);
}

extern SEXP libcoin_R_MaximallySelectedTest(
  SEXP LEV, SEXP ordered, SEXP teststat, SEXP minbucket, SEXP lower, SEXP give_log
) {
  static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;
  if (fun == NULL)
    fun = (SEXP*)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP)
      R_GetCCallable("libcoin", "R_MaximallySelectedTest");
  return fun(LEV, ordered, teststat, minbucket, lower, give_log);
}

```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).

$\langle R\_QuadraticTest \text{ Prototype } 54b \rangle \equiv$

```

SEXP R_QuadraticTest
(
   $\langle R \text{ LECV Input } 151b \rangle$ ,
  SEXP pvalue,
  SEXP lower,
  SEXP give_log,
  SEXP PermutedStatistics
)

```

Fragment referenced in [23b](#), [55](#).

$\langle R\_QuadraticTest\ 55 \rangle \equiv$

```

 $\langle R\_QuadraticTest\ Prototype\ 54b \rangle$ 
{
    SEXP ans, stat, pval, names, permstat;
    double *MPinv, *ls, st, pst, *ex;
    int rank, P, Q, PQ, greater = 0;
    R_xlen_t nresample;

     $\langle Setup\ Test\ Memory\ 56a \rangle$ 

    MPinv = Calloc(PP12(PQ), double); /* was: C_get_MPinv(LECV); */
    C_MPinv_sym(C_get_Covariance(LECV), PQ, C_get_tol(LECV), MPinv, &rank);

    REAL(stat)[0] = C_quadform(PQ, C_get_LinearStatistic(LECV),
                               C_get_Expectation(LECV), MPinv);

    if (!PVALUE) {
        UNPROTECT(2);
        Free(MPinv);
        return(ans);
    }

    if (C_get_nresample(LECV) == 0) {
        REAL(pval)[0] = C_chisq_pvalue(REAL(stat)[0], rank, LOWER, GIVELOG);
    } else {
        nresample = C_get_nresample(LECV);
        ls = C_get_PermutedLinearStatistic(LECV);
        st = REAL(stat)[0];
        ex = C_get_Expectation(LECV);
        greater = 0;
        for (R_xlen_t np = 0; np < nresample; np++) {
            pst = C_quadform(PQ, ls + PQ * np, ex, MPinv);
            if (GE(pst, st, C_get_tol(LECV)))
                greater++;
            if (PSTAT) REAL(permstat)[np] = pst;
        }
        REAL(pval)[0] = C_perm_pvalue(greater, nresample, LOWER, GIVELOG);
    }

    UNPROTECT(2);
    Free(MPinv);
    return(ans);
}

```

Fragment referenced in 53b.

Uses: C\_chisq\_pvalue 67c, C\_get\_Covariance 154a, C\_get\_Expectation 153a, C\_get\_LinearStatistic 152d, C\_get\_nresample 157b, C\_get\_PermutedLinearStatistic 157c, C\_get\_tol 157d, C\_perm\_pvalue 68, C\_quadform 65, GE 22a, LECV 151b, P 25a, PP12 140b, Q 25e.

⟨ *Setup Test Memory 56a* ⟩ ≡

```

P = C_get_P(LECV);
Q = C_get_Q(LECV);
PQ = mPQB(P, Q, 1);

if (C_get_varonly(LECV) && PQ > 1)
    error("cannot compute adjusted p-value based on variances only");
/* if (C_get_nresample(LECV) > 0 && INTEGER(PermutedStatistics)[0]) { */
    PROTECT(ans = allocVector(VECSXP, 3));
    PROTECT(names = allocVector(STRSXP, 3));
    SET_VECTOR_ELT(ans, 2, permstat = allocVector(REALSXP, C_get_nresample(LECV)));
    SET_STRING_ELT(names, 2, mkChar("PermutedStatistics"));
/* } else {
    PROTECT(ans = allocVector(VECSXP, 2));
    PROTECT(names = allocVector(STRSXP, 2));
}
*/
SET_VECTOR_ELT(ans, 0, stat = allocVector(REALSXP, 1));
SET_STRING_ELT(names, 0, mkChar("TestStatistic"));
SET_VECTOR_ELT(ans, 1, pval = allocVector(REALSXP, 1));
SET_STRING_ELT(names, 1, mkChar("p.value"));
namesgets(ans, names);
REAL(pval)[0] = NA_REAL;
int LOWER = INTEGER(lower)[0];
int GIVELOG = INTEGER(give_log)[0];
int PVALUE = INTEGER(pvalue)[0];
int PSTAT = INTEGER(PermutedStatistics)[0];
◇

```

Fragment referenced in [55](#), [57](#).

Uses: [C\\_get\\_nresample 157b](#), [C\\_get\\_P 151c](#), [C\\_get\\_Q 152a](#), [C\\_get\\_varonly 152b](#), [LECV 151b](#), [mPQB 141a](#), [P 25a](#), [Q 25e](#).

⟨ *R\_MaximumTest Prototype 56b* ⟩ ≡

```

SEXP R_MaximumTest
(
    ⟨ R LECV Input 151b ⟩,
    SEXP alternative,
    SEXP pvalue,
    SEXP lower,
    SEXP give_log,
    SEXP PermutedStatistics,
    SEXP maxpts,
    SEXP releps,
    SEXP abseps
)
◇

```

Fragment referenced in [23b](#), [57](#).

$\langle R\_MaximumTest\ 57 \rangle \equiv$

```

 $\langle R\_MaximumTest\ Prototype\ 56b \rangle$ 
{
  SEXP ans, stat, pval, names, permstat;
  double st, pst, *ex, *cv, *ls, tl;
  int P, Q, PQ, vo, alt, greater;
  R_xlen_t nresample;

   $\langle Setup\ Test\ Memory\ 56a \rangle$ 

  if (C_get_varonly(LECV)) {
    cv = C_get_Variance(LECV);
  } else {
    cv = C_get_Covariance(LECV);
  }
  REAL(stat)[0] = C_maxtype(PQ, C_get_LinearStatistic(LECV),
    C_get_Expectation(LECV), cv, C_get_varonly(LECV), C_get_tol(LECV),
    INTEGER(alternative)[0]);
  if (!PVALUE) {
    UNPROTECT(2);
    return(ans);
  }

  if (C_get_nresample(LECV) == 0) {
    if (C_get_varonly(LECV) && PQ > 1) {
      REAL(pval)[0] = NA_REAL;
      UNPROTECT(2);
      return(ans);
    }
    REAL(pval)[0] = C_maxtype_pvalue(REAL(stat)[0], cv,
      PQ, INTEGER(alternative)[0], LOWER, GIVELOG, INTEGER(maxpts)[0],
      REAL(releps)[0], REAL(abseps)[0], C_get_tol(LECV));
  } else {
    nresample = C_get_nresample(LECV);
    ls = C_get_PermutedLinearStatistic(LECV);
    ex = C_get_Expectation(LECV);
    vo = C_get_varonly(LECV);
    alt = INTEGER(alternative)[0];
    st = REAL(stat)[0];
    tl = C_get_tol(LECV);
    greater = 0;
    for (R_xlen_t np = 0; np < nresample; np++) {
      pst = C_maxtype(PQ, ls + PQ * np, ex, cv, vo, tl, alt);
      if (alt == ALTERNATIVE_less) {
        if (LE(pst, st, tl)) greater++;
      } else {
        if (GE(pst, st, tl)) greater++;
      }
      if (PSTAT) REAL(permstat)[np] = pst;
    }
    REAL(pval)[0] = C_perm_pvalue(greater, nresample, LOWER, GIVELOG);
  }
  UNPROTECT(2);
  return(ans);
}

```

Fragment referenced in 53b.

Uses: C\_get\_Covariance 154a, C\_get\_Expectation 153a, C\_get\_LinearStatistic 152d, C\_get\_nresample 157b,  
 C\_get\_PermutedLinearStatistic 157c, C\_get\_tol 157d, C\_get\_Variance 153b, C\_get\_varonly 152b, C\_maxtype 66,  
 C\_maxtype\_pvalue 70, C\_perm\_pvalue 68, GE 22a, LE 22a, LECV 151b, P 25a, Q 25e.

$\langle R\_MaximallySelectedTest \text{ Prototype } 58 \rangle \equiv$

```
SEXP R_MaximallySelectedTest
(
  SEXP LECV,
  SEXP ordered,
  SEXP teststat,
  SEXP minbucket,
  SEXP lower,
  SEXP give_log
)
◇
```

Fragment referenced in [23b](#), [59](#).  
Uses: LECV [151b](#).

$\langle R\_MaximallySelectedTest\ 59 \rangle \equiv$

$\langle R\_MaximallySelectedTest\ Prototype\ 58 \rangle$

```
{
  SEXP ans, index, stat, pval, names, permstat;
  int P, mb;

  P = C_get_P(LECV);
  mb = INTEGER(minbucket)[0];

  PROTECT(ans = allocVector(VECSXP, 4));
  PROTECT(names = allocVector(STRSXP, 4));
  SET_VECTOR_ELT(ans, 0, stat = allocVector(REALSXP, 1));
  SET_STRING_ELT(names, 0, mkChar("TestStatistic"));
  SET_VECTOR_ELT(ans, 1, pval = allocVector(REALSXP, 1));
  SET_STRING_ELT(names, 1, mkChar("p.value"));
  SET_VECTOR_ELT(ans, 3, permstat = allocVector(REALSXP, C_get_nresample(LECV)));
  SET_STRING_ELT(names, 3, mkChar("PermutedStatistics"));
  REAL(pval)[0] = NA_REAL;

  if (INTEGER(ordered)[0]) {
    SET_VECTOR_ELT(ans, 2, index = allocVector(INTSXP, 1));
    C_ordered_Xfactor(LECV, mb, INTEGER(teststat)[0],
                      INTEGER(index), REAL(stat), REAL(permstat),
                      REAL(pval), INTEGER(lower)[0],
                      INTEGER(give_log)[0]);
    if (REAL(stat)[0] > 0)
      INTEGER(index)[0]++; /* R style indexing */
  } else {
    SET_VECTOR_ELT(ans, 2, index = allocVector(INTSXP, P));
    C_unordered_Xfactor(LECV, mb, INTEGER(teststat)[0],
                        INTEGER(index), REAL(stat), REAL(permstat),
                        REAL(pval), INTEGER(lower)[0],
                        INTEGER(give_log)[0]);
  }

  SET_STRING_ELT(names, 2, mkChar("index"));
  namesgets(ans, names);

  UNPROTECT(2);
  return(ans);
}
◇
```

Fragment referenced in [53b](#).

Uses: [C\\_get\\_nresample 157b](#), [C\\_get\\_P 151c](#), [C\\_ordered\\_Xfactor 73](#), [C\\_unordered\\_Xfactor 78](#), [LECV 151b](#), [P 25a](#).

## 3.6 Test Statistics

⟨ *Test Statistics* 60a ⟩ ≡

⟨ *C\_maxstand\_Covariance* 60b ⟩  
 ⟨ *C\_maxstand\_Variance* 61a ⟩  
 ⟨ *C\_minstand\_Covariance* 61b ⟩  
 ⟨ *C\_minstand\_Variance* 62a ⟩  
 ⟨ *C\_maxabsstand\_Covariance* 62b ⟩  
 ⟨ *C\_maxabsstand\_Variance* 63 ⟩  
 ⟨ *C\_quadform* 65 ⟩  
 ⟨ *R\_quadform* 64c ⟩  
 ⟨ *C\_maxtype* 66 ⟩  
 ⟨ *C\_standardise* 67a ⟩  
 ⟨ *C\_ordered\_Xfactor* 73 ⟩  
 ⟨ *C\_unordered\_Xfactor* 78 ⟩  
 ◇

Fragment referenced in 24a.

⟨ *C\_maxstand\_Covariance* 60b ⟩ ≡

```
double C_maxstand_Covariance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *covar_sym,
    const double tol
) {
    double ans = R_NegInf, tmp = 0.0;

    for (int p = 0; p < PQ; p++) {
        tmp = 0.0;
        if (covar_sym[S(p, p, PQ)] > tol)
            tmp = (linstat[p] - expect[p]) / sqrt(covar_sym[S(p, p, PQ)]);
        if (tmp > ans) ans = tmp;
    }
    return(ans);
}
◇
```

Fragment referenced in 60a.

Defines: *C\_maxstand\_Covariance* 66.

Uses: S 22a.

$\langle C\_maxstand\_Variance\ 61a \rangle \equiv$

```
double C_maxstand_Variance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *var,
    const double tol
) {
    double ans = R_NegInf, tmp = 0.0;

    for (int p = 0; p < PQ; p++) {
        tmp = 0.0;
        if (var[p] > tol)
            tmp = (linstat[p] - expect[p]) / sqrt(var[p]);
        if (tmp > ans) ans = tmp;
    }
    return(ans);
}
◇
```

Fragment referenced in [60a](#).

Defines: `C_maxstand_Variance` [66](#).

$\langle C\_minstand\_Covariance\ 61b \rangle \equiv$

```
double C_minstand_Covariance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *covar_sym,
    const double tol
) {
    double ans = R_PosInf, tmp = 0.0;

    for (int p = 0; p < PQ; p++) {
        tmp = 0.0;
        if (covar_sym[S(p, p, PQ)] > tol)
            tmp = (linstat[p] - expect[p]) / sqrt(covar_sym[S(p, p, PQ)]);
        if (tmp < ans) ans = tmp;
    }
    return(ans);
}
◇
```

Fragment referenced in [60a](#).

Defines: `C_minstand_Covariance` [66](#).

Uses: `S` [22a](#).



$\langle C\_minstand\_Variance\ 62a \rangle \equiv$

```
double C_minstand_Variance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *var,
    const double tol
) {
    double ans = R_PosInf, tmp = 0.0;

    for (int p = 0; p < PQ; p++) {
        tmp = 0.0;
        if (var[p] > tol)
            tmp = (linstat[p] - expect[p]) / sqrt(var[p]);
        if (tmp < ans) ans = tmp;
    }
    return(ans);
}
◇
```

Fragment referenced in [60a](#).

Defines: `C_minstand_Variance` [66](#).

$\langle C\_maxabsstand\_Covariance\ 62b \rangle \equiv$

```
double C_maxabsstand_Covariance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *covar_sym,
    const double tol
) {
    double ans = R_NegInf, tmp = 0.0;

    for (int p = 0; p < PQ; p++) {
        tmp = 0.0;
        if (covar_sym[S(p, p, PQ)] > tol)
            tmp = fabs((linstat[p] - expect[p]) /
                sqrt(covar_sym[S(p, p, PQ)]));
        if (tmp > ans) ans = tmp;
    }
    return(ans);
}
◇
```

Fragment referenced in [60a](#).

Defines: `C_maxabsstand_Covariance` [66](#).

Uses: [S 22a](#).

$\langle C\_maxabsstand\_Variance \ 63 \rangle \equiv$

```
double C_maxabsstand_Variance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *var,
    const double tol
) {
    double ans = R_NegInf, tmp = 0.0;

    for (int p = 0; p < PQ; p++) {
        tmp = 0.0;
        if (var[p] > tol)
            tmp = fabs((linstat[p] - expect[p]) / sqrt(var[p]));
        if (tmp > ans) ans = tmp;
    }
    return(ans);
}
◇
```

Fragment referenced in [60a](#).

Defines: `C_maxabsstand_Variance` [66](#).

```
> MPinverse <- function(x, tol = sqrt(.Machine$double.eps)) {
+   SVD <- svd(x)
+   pos <- SVD$d > max(tol * SVD$d[1L], 0)
+   inv <- SVD$v[, pos, drop = FALSE] %*%
+       ((1/SVD$d[pos]) * t(SVD$u[, pos, drop = FALSE]))
+   list(MPinv = inv, rank = sum(pos))
+ }
> quadform <- function (linstat, expect, MPinv) {
+   censtat <- linstat - expect
+   censtat %*% MPinv %*% censtat
+ }
> linstat <- ls1$LinearStatistic
> expect <- ls1$Expectation
> MPinv <- MPinverse(vcov(ls1))$MPinv
> MPinv_sym <- MPinv[lower.tri(MPinv, diag = TRUE)]
> qf1 <- quadform(linstat, expect, MPinv)
> qf2 <- .Call(libcoin:::R_quadform, linstat, expect, MPinv_sym)
> stopifnot(isequal(qf1, qf2))
```

"libcoinAPI.h" 64a≡

```
extern SEXP libcoin_R_quadform(
  SEXP linstat, SEXP expect, SEXP MPinv_sym
) {
  static SEXP(*fun)(SEXP, SEXP, SEXP) = NULL;
  if (fun == NULL)
    fun = (SEXP*)(SEXP, SEXP, SEXP))
      R_GetCCallable("libcoin", "R_quadform");
  return fun(linstat, expect, MPinv_sym);
}
◇
```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).  
Uses: [R\\_quadform 64c](#).

⟨ *R\_quadform Prototype 64b* ⟩ ≡

```
SEXP R_quadform
(
  SEXP linstat,
  SEXP expect,
  SEXP MPinv_sym
)
◇
```

Fragment referenced in [23b](#), [64c](#).  
Uses: [R\\_quadform 64c](#).

⟨ *R\_quadform 64c* ⟩ ≡

```
⟨ R_quadform Prototype 64b ⟩
{
  SEXP ans;
  int n, PQ;
  double *dlinstat, *dexpect, *dMPinv_sym, *dans;

  n = NCOL(linstat);
  PQ = NROW(linstat);
  dlinstat = REAL(linstat);
  dexpect = REAL(expect);
  dMPinv_sym = REAL(MPinv_sym);

  PROTECT(ans = allocVector(REALSXP, n));
  dans = REAL(ans);
  for (int i = 0; i < n; i++)
    dans[i] = C_quadform(PQ, dlinstat + PQ * i, dexpect, dMPinv_sym);

  UNPROTECT(1);
  return(ans);
}
◇
```

Fragment referenced in [60a](#).  
Defines: [R\\_quadform 64ab](#), [164](#), [165](#).  
Uses: [C\\_quadform 65](#), [NCOL 139c](#), [NROW 139b](#).

$\langle C\_quadform\ 65 \rangle \equiv$

```
double C_quadform
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *MPinv_sym
) {
    double ans = 0.0, tmp = 0.0;

    for (int q = 0; q < PQ; q++) {
        tmp = 0.0;
        for (int p = 0; p < PQ; p++)
            tmp += (linstat[p] - expect[p]) * MPinv_sym[S(p, q, PQ)];
        ans += tmp * (linstat[q] - expect[q]);
    }

    return(ans);
}
◇
```

Fragment referenced in [60a](#).

Defines: `C_quadform` [55](#), [64c](#), [76c](#).

Uses: `S` [22a](#).

$\langle C\_maxtype\ 66 \rangle \equiv$

```
double C_maxtype
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *covar,
    const int varonly,
    const double tol,
    const int alternative
) {
    double ret = 0.0;

    if (varonly) {
        if (alternative == ALTERNATIVE_twosided) {
            ret = C_maxabsstand_Variance(PQ, linstat, expect, covar, tol);
        } else if (alternative == ALTERNATIVE_less) {
            ret = C_minstand_Variance(PQ, linstat, expect, covar, tol);
        } else if (alternative == ALTERNATIVE_greater) {
            ret = C_maxstand_Variance(PQ, linstat, expect, covar, tol);
        }
    } else {
        if (alternative == ALTERNATIVE_twosided) {
            ret = C_maxabsstand_Covariance(PQ, linstat, expect, covar, tol);
        } else if (alternative == ALTERNATIVE_less) {
            ret = C_minstand_Covariance(PQ, linstat, expect, covar, tol);
        } else if (alternative == ALTERNATIVE_greater) {
            ret = C_maxstand_Covariance(PQ, linstat, expect, covar, tol);
        }
    }
    return(ret);
}
◇
```

Fragment referenced in [60a](#).

Defines: [C\\_maxtype 57](#), [76c](#).

Uses: [C\\_maxabsstand\\_Covariance 62b](#), [C\\_maxabsstand\\_Variance 63](#), [C\\_maxstand\\_Covariance 60b](#), [C\\_maxstand\\_Variance 61a](#),  
[C\\_minstand\\_Covariance 61b](#), [C\\_minstand\\_Variance 62a](#).

$\langle C\_standardise\ 67a \rangle \equiv$

```
void C_standardise
(
    const int PQ,
    double *linstat,          /* in place standardisation */
    const double *expect,
    const double *covar,
    const int varonly,
    const double tol
) {
    double var;

    for (int p = 0; p < PQ; p++) {
        if (varonly) {
            var = covar[p];
        } else {
            var = covar[S(p, p, PQ)];
        }
        if (var > tol) {
            linstat[p] = (linstat[p] - expect[p]) / sqrt(var);
        } else {
            linstat[p] = NAN;
        }
    }
}
```

◇

Fragment referenced in [60a](#).  
 Defines: `C_standardise` [42a](#).  
 Uses: `S` [22a](#).

$\langle P\_Values\ 67b \rangle \equiv$

```
 $\langle C\_chisq\_pvalue\ 67c \rangle$ 
 $\langle C\_perm\_pvalue\ 68 \rangle$ 
 $\langle C\_norm\_pvalue\ 69 \rangle$ 
 $\langle C\_maxtype\_pvalue\ 70 \rangle$ 
```

◇

Fragment referenced in [24a](#).

$\langle C\_chisq\_pvalue\ 67c \rangle \equiv$

```
/* lower = 1 means p-value, lower = 0 means 1 - p-value */
double C_chisq_pvalue
(
    const double stat,
    const int df,
    const int lower,
    const int give_log
) {
    return(pchisq(stat, (double) df, lower, give_log));
}
```

◇

Fragment referenced in [67b](#).  
 Defines: `C_chisq_pvalue` [55](#).

$\langle C\_perm\_pvalue\ 68 \rangle \equiv$

```
double C_perm_pvalue
(
    const int greater,
    const double nresample,
    const int lower,
    const int give_log
) {
    double ret;

    if (give_log) {
        if (lower) {
            ret = log1p(- (double) greater / nresample);
        } else {
            ret = log(greater) - log(nresample);
        }
    } else {
        if (lower) {
            ret = 1.0 - (double) greater / nresample;
        } else {
            ret = (double) greater / nresample;
        }
    }
    return(ret);
}
◇
```

Fragment referenced in [67b](#).

Defines: `C_perm_pvalue` [55](#), [57](#), [77](#).

$\langle C\_norm\_pvalue\ 69 \rangle \equiv$

```
double C_norm_pvalue
(
    const double stat,
    const int alternative,
    const int lower,
    const int give_log
) {
    double ret;

    if (alternative == ALTERNATIVE_less) {
        return(pnorm(stat, 0.0, 1.0, 1 - lower, give_log));
    } else if (alternative == ALTERNATIVE_greater) {
        return(pnorm(stat, 0.0, 1.0, lower, give_log));
    } else if (alternative == ALTERNATIVE_twosided) {
        if (lower) {
            ret = pnorm(fabs(stat)*-1.0, 0.0, 1.0, 1, 0);
            if (give_log) {
                return(log1p(- 2 * ret));
            } else {
                return(1 - 2 * ret);
            }
        } else {
            ret = pnorm(fabs(stat)*-1.0, 0.0, 1.0, 1, give_log);
            if (give_log) {
                return(ret + log(2));
            } else {
                return(2 * ret);
            }
        }
    }
    return(NA_REAL);
}
◇
```

Fragment referenced in [67b](#).



$\langle C\_maxtype\_pvalue\ 70 \rangle \equiv$

```
double C_maxtype_pvalue
(
    const double stat,
    const double *Covariance,
    const int n,
    const int alternative,
    const int lower,
    const int give_log,
    int maxpts, /* const? */
    double releps,
    double abseps,
    double tol
) {
    int nu = 0, inform, i, j, sub, nonzero, *infin, *index, rnd = 0;
    double ans, myerror, *lowerbnd, *upperbnd, *delta, *corr, *sd;

    /* univariate problem */
    if (n == 1)
        return(C_norm_pvalue(stat, alternative, lower, give_log));

     $\langle Setup\ mvtnorm\ Memory\ 71 \rangle$ 

     $\langle Setup\ mvtnorm\ Correlation\ 72a \rangle$ 

    /* call mvtnorm's mvtstd C function defined in mvtnorm/include/mvtnormAPI.h */
    mvtnorm_C_mvtstd(&nonzero, &nu, lowerbnd, upperbnd, infin, corr, delta,
        &maxpts, &abseps, &releps, &myerror, &ans, &inform, &rnd);

    /* inform == 0 means: everything is OK */
    switch (inform) {
        case 0: break;
        case 1: warning("cmvnorm: completion with ERROR > EPS"); break;
        case 2: warning("cmvnorm: N > 1000 or N < 1");
            ans = 0.0;
            break;
        case 3: warning("cmvnorm: correlation matrix not positive semi-definite");
            ans = 0.0;
            break;
        default: warning("cmvnorm: unknown problem in MVTSTD");
            ans = 0.0;
    }
    Free(corr); Free(sd); Free(lowerbnd); Free(upperbnd);
    Free(infin); Free(delta); Free(index);

    /* ans = 1 - p-value */
    if (lower) {
        if (give_log)
            return(log(ans)); /* log(1 - p-value) */
        return(ans); /* 1 - p-value */
    } else {
        if (give_log)
            return(log1p(ans)); /* log(p-value) */
        return(1 - ans); /* p-value */
    }
}
◇
```

Fragment referenced in [67b](#).  
 Defines: `C_maxtype_pvalue` [57](#).  
 Uses: `N` [24bc](#).

*⟨ Setup mvtnorm Memory 71 ⟩*  $\equiv$

```
if (n == 2)
  corr = Calloc(1, double);
else
  corr = Calloc(n + ((n - 2) * (n - 1))/2, double);

sd = Calloc(n, double);
lowerbnd = Calloc(n, double);
upperbnd = Calloc(n, double);
infin = Calloc(n, int);
delta = Calloc(n, double);
index = Calloc(n, int);

/* determine elements with non-zero variance */

nonzero = 0;
for (i = 0; i < n; i++) {
  if (Covariance[S(i, i, n)] > tol) {
    index[nonzero] = i;
    nonzero++;
  }
}
◇
```

Fragment referenced in [70](#).

Uses: [S 22a](#).

`mvtdst` assumes the unique elements of the triangular covariance matrix to be passed as argument `CORREL`

⟨ *Setup mvtnorm Correlation 72a* ⟩ ≡

```

for (int nz = 0; nz < nonzero; nz++) {
  /* handle elements with non-zero variance only */
  i = index[nz];

  /* standard deviations */
  sd[i] = sqrt(Covariance[S(i, i, n)]);

  if (alternative == ALTERNATIVE_less) {
    lowerbnd[nz] = stat;
    upperbnd[nz] = R_PosInf;
    infin[nz] = 1;
  } else if (alternative == ALTERNATIVE_greater) {
    lowerbnd[nz] = R_NegInf;
    upperbnd[nz] = stat;
    infin[nz] = 0;
  } else if (alternative == ALTERNATIVE_twosided) {
    lowerbnd[nz] = fabs(stat) * -1.0;
    upperbnd[nz] = fabs(stat);
    infin[nz] = 2;
  }

  delta[nz] = 0.0;

  /* set up vector of correlations, i.e., the upper
     triangular part of the covariance matrix) */
  for (int jz = 0; jz < nz; jz++) {
    j = index[jz];
    sub = (int) (jz + 1) + (double) ((nz - 1) * nz) / 2 - 1;
    if (sd[i] == 0.0 || sd[j] == 0.0)
      corr[sub] = 0.0;
    else
      corr[sub] = Covariance[S(i, j, n)] / (sd[i] * sd[j]);
  }
}
◇

```

Fragment referenced in [70](#).

Uses: [S 22a](#).

⟨ *maxstat Xfactor Variables 72b* ⟩ ≡

```

SEXP LECV,
const int minbucket,
const int teststat,
int *wmax,
double *maxstat,
double *bmaxstat,
double *pval,
const int lower,
const int give_log
◇

```

Fragment referenced in [73](#), [78](#).

Uses: [LECV 151b](#).

$\langle C\_ordered\_Xfactor\ 73 \rangle \equiv$

```

void C_ordered_Xfactor
(
     $\langle maxstat\ Xfactor\ Variables\ 72b \rangle$ 
) {
     $\langle Setup\ maxstat\ Variables\ 74 \rangle$ 

     $\langle Setup\ maxstat\ Memory\ 75 \rangle$ 

    wmax[0] = NA_INTEGER;

    for (int p = 0; p < P; p++) {
        sumleft += ExpX[p];
        sumright -= ExpX[p];

        for (int q = 0; q < Q; q++) {
            mlinstat[q] += linstat[q * P + p];
            for (R_xlen_t np = 0; np < nresample; np++)
                mblinstat[q + np * Q] += blinstat[q * P + p + np * PQ];
            mexpect[q] += expect[q * P + p];
            if (B == 1) {
                 $\langle Compute\ maxstat\ Variance\ /\ Covariance\ Directly\ 76b \rangle$ 
            } else {
                 $\langle Compute\ maxstat\ Variance\ /\ Covariance\ from\ Total\ Covariance\ 76a \rangle$ 
            }
        }

        if ((sumleft >= minbucket) && (sumright >= minbucket) && (ExpX[p] > 0)) {
            ls = mlinstat;
            /* compute MPinv only once */
            if (teststat != TESTSTAT_maximum)
                C_MPinv_sym(mcovar, Q, tol, mMPinv, &rank);
             $\langle Compute\ maxstat\ Test\ Statistic\ 76c \rangle$ 
            if (tmp > maxstat[0]) {
                wmax[0] = p;
                maxstat[0] = tmp;
            }

            for (R_xlen_t np = 0; np < nresample; np++) {
                ls = mblinstat + np * Q;
                 $\langle Compute\ maxstat\ Test\ Statistic\ 76c \rangle$ 
                if (tmp > bmaxstat[np])
                    bmaxstat[np] = tmp;
            }
        }

         $\langle Compute\ maxstat\ Permutation\ P-Value\ 77 \rangle$ 
        Free(mlinstat); Free(mexpect); Free(mblinstat);
        Free(mvar); Free(mcovar); Free(mMPinv);
        if (nresample == 0) Free(blinstat);
    }
}

```

Fragment referenced in 60a.

Defines: C\_ordered\_Xfactor 37b, 47, 59.

Uses: B 28c, P 25a, Q 25e.

⟨ *Setup maxstat Variables 74* ⟩ ≡

```

double *linstat, *expect, *covar, *varinf, *covinf, *ExpX, *blinstat, tol, *ls;
int P, Q, B;
R_xlen_t nresample;

double *mlinstat, *mblinstat, *mexpect, *mvar, *mcovar, *mMPinv,
      tmp, sumleft, sumright, sumweights;
int rank, PQ, greater;

Q = C_get_Q(LECV);
P = C_get_P(LECV);
PQ = mPQB(P, Q, 1);
B = C_get_B(LECV);
if (B > 1) {
    if (C_get_varonly(LECV))
        error("need covariance for maximally statistics with blocks");
    covar = C_get_Covariance(LECV);
} else {
    covar = C_get_Variance(LECV); /* make -Wall happy */
}
linstat = C_get_LinearStatistic(LECV);
expect = C_get_Expectation(LECV);
ExpX = C_get_ExpectationX(LECV);
/* both need to be there */
varinf = C_get_VarianceInfluence(LECV);
covinf = C_get_CovarianceInfluence(LECV);
nresample = C_get_nresample(LECV);
if (nresample > 0)
    blinstat = C_get_PermutedLinearStatistic(LECV);
tol = C_get_tol(LECV);
◇

```

Fragment referenced in 73, 78.

Uses: B 28c, C\_get\_B 157a, C\_get\_Covariance 154a, C\_get\_CovarianceInfluence 155a, C\_get\_Expectation 153a, C\_get\_ExpectationX 154b, C\_get\_LinearStatistic 152d, C\_get\_nresample 157b, C\_get\_P 151c, C\_get\_PermutedLinearStatistic 157c, C\_get\_Q 152a, C\_get\_tol 157d, C\_get\_Variance 153b, C\_get\_VarianceInfluence 155b, C\_get\_varonly 152b, LECV 151b, mPQB 141a, P 25a, Q 25e, sumweights 27a.

⟨ *Setup maxstat Memory 75* ⟩ ≡

```

mlinstat = Calloc(Q, double);
mexpect = Calloc(Q, double);
if (teststat == TESTSTAT_maximum) {
    mvar = Calloc(Q, double);
    /* not needed, but allocate anyway to make -Wmaybe-uninitialized happy */
    mcovar = Calloc(1, double);
    mMPinv = Calloc(1, double);
} else {
    mcovar = Calloc(Q * (Q + 1) / 2, double);
    mMPinv = Calloc(Q * (Q + 1) / 2, double);
    /* not needed, but allocate anyway to make -Wmaybe-uninitialized happy */
    mvar = Calloc(1, double);
}
if (nresample > 0) {
    mblinstat = Calloc(Q * nresample, double);
} else { /* not needed, but allocate anyway to make -Wmaybe-uninitialized happy */
    mblinstat = Calloc(1, double);
    blinstat = Calloc(1, double);
}

maxstat[0] = 0.0;

for (int q = 0; q < Q; q++) {
    mlinstat[q] = 0.0;
    mexpect[q] = 0.0;
    if (teststat == TESTSTAT_maximum)
        mvar[q] = 0.0;
    for (R_xlen_t np = 0; np < nresample; np++) {
        mblinstat[q + np * Q] = 0.0;
        bmaxstat[np] = 0.0;
    }
}
if (teststat == TESTSTAT_quadratic) {
    for (int q = 0; q < Q * (Q + 1) / 2; q++)
        mcovar[q] = 0.0;
}

sumleft = 0.0;
sumright = 0.0;
for (int p = 0; p < P; p++)
    sumright += ExpX[p];
sumweights = sumright;
◇

```

Fragment referenced in [73](#), [78](#).

Uses: [P 25a](#), [Q 25e](#), [sumweights 27a](#).

*⟨ Compute maxstat Variance / Covariance from Total Covariance 76a ⟩ ≡*

```

if (teststat == TESTSTAT_maximum) {
    for (int pp = 0; pp < p; pp++)
        mvar[q] += 2 * covar[S(pp + q * P, p + P * q, mPQB(P, Q, 1))];
    mvar[q] += covar[S(p + q * P, p + P * q, mPQB(P, Q, 1))];
} else {
    for (int qq = 0; qq <= q; qq++) {
        for (int pp = 0; pp < p; pp++)
            mcovar[S(q, qq, Q)] += 2 * covar[S(pp + q * P, p + P * qq, mPQB(P, Q, 1))];
        mcovar[S(q, qq, Q)] += covar[S(p + q * P, p + P * qq, mPQB(P, Q, 1))];
    }
}
}
◇

```

Fragment referenced in 73.

Uses: mPQB 141a, P 25a, Q 25e, S 22a.

*⟨ Compute maxstat Variance / Covariance Directly 76b ⟩ ≡*

```

/* does not work with blocks! */
if (teststat == TESTSTAT_maximum) {
    C_VarianceLinearStatistic(1, Q, varinf, &sumleft, &sumleft,
                             sumweights, 0, mvar);
} else {
    C_CovarianceLinearStatistic(1, Q, covinf, &sumleft, &sumleft,
                               sumweights, 0, mcovar);
}
}
◇

```

Fragment referenced in 73.

Uses: C\_CovarianceLinearStatistic 83, C\_VarianceLinearStatistic 84, Q 25e, sumweights 27a.

*⟨ Compute maxstat Test Statistic 76c ⟩ ≡*

```

if (teststat == TESTSTAT_maximum) {
    tmp = C_maxtype(Q, ls, mexpect, mvar, 1, tol,
                    ALTERNATIVE_twosided);
} else {
    tmp = C_quadform(Q, ls, mexpect, mMPinv);
}
}
◇

```

Fragment referenced in 73, 78.

Uses: C\_maxtype 66, C\_quadform 65, Q 25e.

$\langle$  *Compute maxstat Permutation P-Value 77*  $\rangle \equiv$

```
if (nresample > 0) {  
  greater = 0;  
  for (R_xlen_t np = 0; np < nresample; np++) {  
    if (bmaxstat[np] > maxstat[0]) greater++;  
  }  
  pval[0] = C_perm_pvalue(greater, nresample, lower, give_log);  
}  
◇
```

Fragment referenced in [73](#), [78](#).

Uses: `C_perm_pvalue` [68](#).



$\langle C\_unordered\_Xfactor\ 78 \rangle \equiv$

```

void C_unordered_Xfactor
(
     $\langle maxstat\ Xfactor\ Variables\ 72b \rangle$ 
) {
    double *mtmp;
    int qPp, nc, *levels, Pnonzero, *indl, *contrast;

     $\langle Setup\ maxstat\ Variables\ 74 \rangle$ 

     $\langle Setup\ maxstat\ Memory\ 75 \rangle$ 
    mtmp = Calloc(P, double);

    for (int p = 0; p < P; p++) wmax[p] = NA_INTEGER;

     $\langle Count\ Levels\ 79a \rangle$ 

    for (int j = 1; j < mi; j++) { /* go though all splits */

         $\langle Setup\ unordered\ maxstat\ Contrasts\ 79b \rangle$ 

         $\langle Compute\ unordered\ maxstat\ Linear\ Statistic\ and\ Expectation\ 80a \rangle$ 

        if (B == 1) {
             $\langle Compute\ unordered\ maxstat\ Variance\ /\ Covariance\ Directly\ 81a \rangle$ 
        } else {
             $\langle Compute\ unordered\ maxstat\ Variance\ /\ Covariance\ from\ Total\ Covariance\ 80b \rangle$ 
        }

        if ((sumleft >= minbucket) && (sumright >= minbucket)) {
            ls = mlinstat;
            /* compute MPinv only once */
            if (teststat != TESTSTAT_maximum)
                C_MPinv_sym(mcovar, Q, tol, mMPinv, &rank);
             $\langle Compute\ maxstat\ Test\ Statistic\ 76c \rangle$ 
            if (tmp > maxstat[0]) {
                for (int p = 0; p < Pnonzero; p++)
                    wmax[levels[p]] = contrast[levels[p]];
                maxstat[0] = tmp;
            }

            for (R_xlen_t np = 0; np < nresample; np++) {
                ls = mblinstat + np * Q;
                 $\langle Compute\ maxstat\ Test\ Statistic\ 76c \rangle$ 
                if (tmp > bmaxstat[np])
                    bmaxstat[np] = tmp;
            }
        }
    }

     $\langle Compute\ maxstat\ Permutation\ P-Value\ 77 \rangle$ 

    Free(mlinstat); Free(mexpect); Free(levels); Free(contrast); Free(indl); Free(mtmp);
    Free(mblinstat); Free(mvar); Free(mcovar); Free(mMPinv);
    if (nresample == 0) Free(blinstat);
}

```

Fragment referenced in 60a.

Defines: C\_unordered\_Xfactor 37b, 59.

Uses: B 28c, P 25a, Q 25e.

⟨ *Count Levels 79a* ⟩ ≡

```

    contrast = Calloc(P, int);
    Pnonzero = 0;
    for (int p = 0; p < P; p++) {
        if (ExpX[p] > 0) Pnonzero++;
    }
    levels = Calloc(Pnonzero, int);
    nc = 0;
    for (int p = 0; p < P; p++) {
        if (ExpX[p] > 0) {
            levels[nc] = p;
            nc++;
        }
    }

    if (Pnonzero >= 31)
        error("cannot search for unordered splits in >= 31 levels");

    int mi = 1;
    for (int l = 1; l < Pnonzero; l++) mi *= 2;
    indl = Calloc(Pnonzero, int);
    for (int p = 0; p < Pnonzero; p++) indl[p] = 0;
    ◇

```

Fragment referenced in [78](#).

Uses: P [25a](#).

⟨ *Setup unordered maxstat Contrasts 79b* ⟩ ≡

```

    /* indl determines if level p is left or right */
    int jj = j;
    for (int l = 1; l < Pnonzero; l++) {
        indl[l] = (jj%2);
        jj /= 2;
    }

    sumleft = 0.0;
    sumright = 0.0;
    for (int p = 0; p < P; p++) contrast[p] = 0;
    for (int p = 0; p < Pnonzero; p++) {
        sumleft += indl[p] * ExpX[levels[p]];
        sumright += (1 - indl[p]) * ExpX[levels[p]];
        contrast[levels[p]] = indl[p];
    }
    ◇

```

Fragment referenced in [78](#).

Uses: P [25a](#).

*⟨ Compute unordered maxstat Linear Statistic and Expectation 80a ⟩ ≡*

```

for (int q = 0; q < Q; q++) {
  mlinstat[q] = 0.0;
  mexpect[q] = 0.0;
  for (R_xlen_t np = 0; np < nresample; np++)
    mblinstat[q + np * Q] = 0.0;
  for (int p = 0; p < P; p++) {
    qPp = q * P + p;
    mlinstat[q] += contrast[p] * linstat[qPp];
    mexpect[q] += contrast[p] * expect[qPp];
    for (R_xlen_t np = 0; np < nresample; np++)
      mblinstat[q + np * Q] += contrast[p] * blinstat[q * P + p + np * PQ];
  }
}
◇

```

Fragment referenced in 78.

Uses: P 25a, Q 25e.

*⟨ Compute unordered maxstat Variance / Covariance from Total Covariance 80b ⟩ ≡*

```

if (teststat == TESTSTAT_maximum) {
  for (int q = 0; q < Q; q++) {
    mvar[q] = 0.0;
    for (int p = 0; p < P; p++) {
      qPp = q * P + p;
      mtmp[p] = 0.0;
      for (int pp = 0; pp < P; pp++)
        mtmp[p] += contrast[pp] * covar[S(pp + q * P, qPp, PQ)];
    }
    for (int p = 0; p < P; p++)
      mvar[q] += contrast[p] * mtmp[p];
  }
} else {
  for (int q = 0; q < Q; q++) {
    for (int qq = 0; qq <= q; qq++)
      mcovar[S(q, qq, Q)] = 0.0;
    for (int qq = 0; qq <= q; qq++) {
      for (int p = 0; p < P; p++) {
        mtmp[p] = 0.0;
        for (int pp = 0; pp < P; pp++)
          mtmp[p] += contrast[pp] * covar[S(pp + q * P, p + P * qq,
            mPQB(P, Q, 1))];
      }
      for (int p = 0; p < P; p++)
        mcovar[S(q, qq, Q)] += contrast[p] * mtmp[p];
    }
  }
}
◇

```

Fragment referenced in 78.

Uses: mPQB 141a, P 25a, Q 25e, S 22a.

$\langle \text{Compute unordered maxstat Variance / Covariance Directly 81a} \rangle \equiv$

```

if (teststat == TESTSTAT_maximum) {
    C_VarianceLinearStatistic(1, Q, varinf, &sumleft, &sumleft,
                             sumweights, 0, mvar);
} else {
    C_CovarianceLinearStatistic(1, Q, covinf, &sumleft, &sumleft,
                              sumweights, 0, mcovar);
}

```

Fragment referenced in 78.

Uses: C\_CovarianceLinearStatistic 83, C\_VarianceLinearStatistic 84, Q 25e, sumweights 27a.

### 3.7 Linear Statistics

$\langle \text{LinearStatistics 81b} \rangle \equiv$

```

     $\langle \text{RC\_LinearStatistic 81d} \rangle$ 

```

Fragment referenced in 24a.

$\langle \text{RC\_LinearStatistic Prototype 81c} \rangle \equiv$

```

void RC_LinearStatistic
(
     $\langle R \ x \ \text{Input 24d} \rangle$ 
     $\langle C \ \text{integer } N \ \text{Input 24c} \rangle$ ,
     $\langle C \ \text{integer } P \ \text{Input 25a} \rangle$ ,
     $\langle C \ \text{real } y \ \text{Input 26a} \rangle$ 
     $\langle R \ \text{weights Input 26c} \rangle$ ,
     $\langle R \ \text{subset Input 27b} \rangle$ ,
     $\langle C \ \text{subset range Input 27d} \rangle$ ,
     $\langle C \ \text{KronSums Answer 101d} \rangle$ 
)

```

Fragment referenced in 81d.

Uses: RC\_LinearStatistic 81d.

$\langle \text{RC\_LinearStatistic 81d} \rangle \equiv$

```

     $\langle \text{RC\_LinearStatistic Prototype 81c} \rangle$ 
{
    double center;

    RC_KronSums(x, N, P, y, Q, !DoSymmetric, &center, &center, !DoCenter, weights,
               subset, offset, Nsubset, PQ_ans);
}

```

Fragment referenced in 81b.

Defines: RC\_LinearStatistic 35b, 81c.

Uses: DoCenter 22b, DoSymmetric 22b, N 24bc, Nsubset 27c, offset 27d, P 25a, Q 25e, RC\_KronSums 101a, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

## 3.8 Expectation and Covariance

$\langle \textit{ExpectationCovariances}$  82a  $\rangle \equiv$

$\langle \textit{RC\_ExpectationInfluence}$  86a  $\rangle$   
 $\langle \textit{R\_ExpectationInfluence}$  85b  $\rangle$   
 $\langle \textit{RC\_CovarianceInfluence}$  88a  $\rangle$   
 $\langle \textit{R\_CovarianceInfluence}$  87a  $\rangle$   
 $\langle \textit{RC\_ExpectationX}$  90  $\rangle$   
 $\langle \textit{R\_ExpectationX}$  89a  $\rangle$   
 $\langle \textit{RC\_CovarianceX}$  93a  $\rangle$   
 $\langle \textit{R\_CovarianceX}$  92a  $\rangle$   
 $\langle \textit{C\_ExpectationLinearStatistic}$  82b  $\rangle$   
 $\langle \textit{C\_CovarianceLinearStatistic}$  83  $\rangle$   
 $\langle \textit{C\_VarianceLinearStatistic}$  84  $\rangle$   
 $\diamond$

Fragment referenced in 24a.

### 3.8.1 Linear Statistic

$\langle \textit{C\_ExpectationLinearStatistic}$  82b  $\rangle \equiv$

```
void C_ExpectationLinearStatistic
(
     $\langle \textit{C integer P Input}$  25a  $\rangle$ ,
     $\langle \textit{C integer Q Input}$  25e  $\rangle$ ,
    double *ExpInf,
    double *ExpX,
    const int add,
    double *PQ_ans
) {
    if (!add)
        for (int p = 0; p < mPQB(P, Q, 1); p++) PQ_ans[p] = 0.0;

    for (int p = 0; p < P; p++) {
        for (int q = 0; q < Q; q++)
            PQ_ans[q * P + p] += ExpX[p] * ExpInf[q];
    }
}
```

$\diamond$

Fragment referenced in 82a.

Defines: `C_ExpectationLinearStatistic` 37a, 46c.

Uses: `mPQB` 141a, `P` 25a, `Q` 25e.

$\langle C\_CovarianceLinearStatistic\ 83 \rangle \equiv$

```

void C_CovarianceLinearStatistic
(
     $\langle C\ integer\ P\ Input\ 25a \rangle$ ,
     $\langle C\ integer\ Q\ Input\ 25e \rangle$ ,
    double *CovInf,
    double *ExpX,
    double *CovX,
     $\langle C\ sumweights\ Input\ 27a \rangle$ ,
    const int add,
    double *PQPQ_sym_ans
) {
    double f1 = sumweights / (sumweights - 1);
    double f2 = 1.0 / (sumweights - 1);
    double tmp, *PP_sym_tmp;

    if (mPQB(P, Q, 1) == 1) {
        tmp = f1 * CovInf[0] * CovX[0];
        tmp -= f2 * CovInf[0] * ExpX[0] * ExpX[0];
        if (add) {
            PQPQ_sym_ans[0] += tmp;
        } else {
            PQPQ_sym_ans[0] = tmp;
        }
    } else {
        PP_sym_tmp = Calloc(PP12(P), double);
        C_KronSums_sym_(ExpX, 1, P,
            PP_sym_tmp);
        for (int p = 0; p < PP12(P); p++)
            PP_sym_tmp[p] = f1 * CovX[p] - f2 * PP_sym_tmp[p];
        C_kronecker_sym(CovInf, Q, PP_sym_tmp, P, 1 - (add >= 1),
            PQPQ_sym_ans);
        Free(PP_sym_tmp);
    }
}

```

Fragment referenced in 82a.

Defines: C\_CovarianceLinearStatistic 38a, 47, 76b, 81a, 84.

Uses: C\_kronecker\_sym 144, mPQB 141a, P 25a, PP12 140b, Q 25e, sumweights 27a.

$\langle C\_VarianceLinearStatistic\ 84 \rangle \equiv$

```

void C_VarianceLinearStatistic
(
   $\langle C\ integer\ P\ Input\ 25a \rangle$ ,
   $\langle C\ integer\ Q\ Input\ 25e \rangle$ ,
  double *VarInf,
  double *ExpX,
  double *VarX,
   $\langle C\ sumweights\ Input\ 27a \rangle$ ,
  const int add,
  double *PQ_ans
) {
  if (mPQB(P, Q, 1) == 1) {
    C_CovarianceLinearStatistic(P, Q, VarInf, ExpX, VarX,
                                sumweights, (add >= 1),
                                PQ_ans);
  } else {
    double *P_tmp;
    P_tmp = Calloc(P, double);
    double f1 = sumweights / (sumweights - 1);
    double f2 = 1.0 / (sumweights - 1);
    for (int p = 0; p < P; p++)
      P_tmp[p] = f1 * VarX[p] - f2 * ExpX[p] * ExpX[p];
    C_kronecker(VarInf, 1, Q, P_tmp, 1, P, 1 - (add >= 1),
                PQ_ans);
    Free(P_tmp);
  }
}

```

Fragment referenced in [82a](#).

Defines: `C_VarianceLinearStatistic` [37c](#), [47](#), [76b](#), [81a](#).

Uses: `C_CovarianceLinearStatistic` [83](#), `C_kronecker` [143](#), `mPQB` [141a](#), `P` [25a](#), `Q` [25e](#), `sumweights` [27a](#).

### 3.8.2 Influence

```

> sumweights <- sum(weights[subset])
> expecty <- a0 <- colSums(y[subset, ] * weights[subset]) / sumweights
> a1 <- .Call(libcoin:::R_ExpectationInfluence, y, weights, subset);
> a2 <- .Call(libcoin:::R_ExpectationInfluence, y, as.double(weights), as.double(subset));
> a3 <- .Call(libcoin:::R_ExpectationInfluence, y, weights, as.double(subset));
> a4 <- .Call(libcoin:::R_ExpectationInfluence, y, as.double(weights), subset);
> a5 <- LinStatExpCov(x, y, weights = weights, subset = subset)$ExpectationInfluence
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4) &&
+           isequal(a0, a5))

```

$\langle R\_ExpectationInfluence \text{ Prototype } 85a \rangle \equiv$

```
SEXP R_ExpectationInfluence
(
   $\langle R \text{ } y \text{ Input } 25d \rangle$ 
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ 
)
◇
```

Fragment referenced in 23b, 85b.

Uses: R\_ExpectationInfluence 85b.

$\langle R\_ExpectationInfluence \text{ } 85b \rangle \equiv$

```
 $\langle R\_ExpectationInfluence \text{ Prototype } 85a \rangle$ 
{
  SEXP ans;
   $\langle C \text{ integer } Q \text{ Input } 25e \rangle$ ;
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ;
   $\langle C \text{ integer } Nsubset \text{ Input } 27c \rangle$ ;
  double sumweights;

  Q = NCOL(y);
  N = XLENGTH(y) / Q;
  Nsubset = XLENGTH(subset);

  sumweights = RC_Sums(N, weights, subset, Offset0, Nsubset);

  PROTECT(ans = allocVector(REALSXP, Q));
  RC_ExpectationInfluence(N, y, Q, weights, subset, Offset0, Nsubset, sumweights, REAL(ans));
  UNPROTECT(1);
  return(ans);
}
◇
```

Fragment referenced in 82a.

Defines: R\_ExpectationInfluence 85a, 87a, 164, 165.

Uses: N 24bc, NCOL 139c, Nsubset 27c, Offset0 22b, Q 25e, RC\_ExpectationInfluence 86a, RC\_Sums 96a, subset 27be, 28a, sumweights 27a, weights 26c, weights, 26de, y 25d, 26ab.

$\langle RC\_ExpectationInfluence \text{ Prototype } 85c \rangle \equiv$

```
void RC_ExpectationInfluence
(
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ,
   $\langle R \text{ } y \text{ Input } 25d \rangle$ 
   $\langle C \text{ integer } Q \text{ Input } 25e \rangle$ ,
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ ,
   $\langle C \text{ subset range Input } 27d \rangle$ ,
   $\langle C \text{ sumweights Input } 27a \rangle$ ,
   $\langle C \text{ colSums Answer } 114c \rangle$ 
)
◇
```

Fragment referenced in 86a.

Uses: RC\_ExpectationInfluence 86a.



$\langle RC\_ExpectationInfluence$  86a  $\rangle \equiv$

```

 $\langle RC\_ExpectationInfluence$  Prototype 85c  $\rangle$ 
{
    double center;

    RC_colSums(REAL(y), N, Q, Power1, &center, !DoCenter, weights,
               subset, offset, Nsubset, P_ans);
    for (int q = 0; q < Q; q++)
        P_ans[q] = P_ans[q] / sumweights;
}

```

Fragment referenced in 82a.

Defines:  $RC\_ExpectationInfluence$  37a, 46c, 85bc.

Uses: DoCenter 22b, N 24bc, Nsubset 27c, offset 27d, Power1 22b, Q 25e, RC\_colSums 114a, subset 27be, 28a, sumweights 27a, weights 26c, weights, 26de, y 25d, 26ab.

```

> sumweights <- sum(weights[subset])
> yc <- t(t(y) - expecty)
> r1y <- rep(1:ncol(y), ncol(y))
> r2y <- rep(1:ncol(y), each = ncol(y))
> a0 <- colSums(yc[subset, r1y] * yc[subset, r2y] * weights[subset]) / sumweights
> a0 <- matrix(a0, ncol = ncol(y))
> vary <- diag(a0)
> a0 <- a0[lower.tri(a0, diag = TRUE)]
> a1 <- .Call(libcoin:::R_CovarianceInfluence, y, weights, subset, 0L);
> a2 <- .Call(libcoin:::R_CovarianceInfluence, y, as.double(weights), as.double(subset), 0L);
> a3 <- .Call(libcoin:::R_CovarianceInfluence, y, weights, as.double(subset), 0L);
> a4 <- .Call(libcoin:::R_CovarianceInfluence, y, as.double(weights), subset, 0L);
> a5 <- LinStatExpCov(x, y, weights = weights, subset = subset)$CovarianceInfluence
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4) &&
+           isequal(a0, a5))
> a1 <- .Call(libcoin:::R_CovarianceInfluence, y, weights, subset, 1L);
> a2 <- .Call(libcoin:::R_CovarianceInfluence, y, as.double(weights), as.double(subset), 1L);
> a3 <- .Call(libcoin:::R_CovarianceInfluence, y, weights, as.double(subset), 1L);
> a4 <- .Call(libcoin:::R_CovarianceInfluence, y, as.double(weights), subset, 1L);
> a5 <- LinStatExpCov(x, y, weights = weights, subset = subset, varonly = TRUE)$VarianceInfluence
> a0 <- vary
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4) &&
+           isequal(a0, a5))

```

$\langle R\_CovarianceInfluence$  Prototype 86b  $\rangle \equiv$

```

SEXP R_CovarianceInfluence
(
     $\langle R$  y Input 25d  $\rangle$ 
     $\langle R$  weights Input 26c  $\rangle$ ,
     $\langle R$  subset Input 27b  $\rangle$ ,
    SEXP varonly
)

```

Fragment referenced in 23b, 87a.

Uses:  $R\_CovarianceInfluence$  87a.

$\langle R\_CovarianceInfluence\ 87a \rangle \equiv$

```

 $\langle R\_CovarianceInfluence\ Prototype\ 86b \rangle$ 
{
    SEXP ans;
    SEXP ExpInf;
     $\langle C\ integer\ Q\ Input\ 25e \rangle$ ;
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;
    double sumweights;

    Q = NCOL(y);
    N = XLENGTH(y) / Q;
    Nsubset = XLENGTH(subset);

    PROTECT(ExpInf = R_ExpectationInfluence(y, weights, subset));

    sumweights = RC_Sums(N, weights, subset, Offset0, Nsubset);

    if (INTEGER(varonly)[0]) {
        PROTECT(ans = allocVector(REALSXP, Q));
    } else {
        PROTECT(ans = allocVector(REALSXP, Q * (Q + 1) / 2));
    }
    RC_CovarianceInfluence(N, y, Q, weights, subset, Offset0, Nsubset, REAL(ExpInf), sumweights,
                           INTEGER(varonly)[0], REAL(ans));
    UNPROTECT(2);
    return(ans);
}

```

Fragment referenced in 82a.

Defines: `R_CovarianceInfluence` 86b, 164, 165.

Uses: `N` 24bc, `NCOL` 139c, `Nsubset` 27c, `Offset0` 22b, `Q` 25e, `RC_CovarianceInfluence` 88a, `RC_Sums` 96a, `R_ExpectationInfluence` 85b, `subset` 27be, 28a, `sumweights` 27a, `weights` 26c, `weights`, 26de, `y` 25d, 26ab.

$\langle RC\_CovarianceInfluence\ Prototype\ 87b \rangle \equiv$

```

void RC_CovarianceInfluence
(
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
     $\langle R\ y\ Input\ 25d \rangle$ 
     $\langle C\ integer\ Q\ Input\ 25e \rangle$ ,
     $\langle R\ weights\ Input\ 26c \rangle$ ,
     $\langle R\ subset\ Input\ 27b \rangle$ ,
     $\langle C\ subset\ range\ Input\ 27d \rangle$ ,
    double *ExpInf,
     $\langle C\ sumweights\ Input\ 27a \rangle$ ,
    int VARONLY,
     $\langle C\ KronSums\ Answer\ 101d \rangle$ 
)

```

Fragment referenced in 88a.

Uses: `RC_CovarianceInfluence` 88a.

$\langle RC\_CovarianceInfluence\ 88a \rangle \equiv$

```

 $\langle RC\_CovarianceInfluence\ Prototype\ 87b \rangle$ 
{
  if (VARONLY) {
    RC_colSums(REAL(y), N, Q, Power2, ExpInf, DoCenter, weights,
              subset, offset, Nsubset, PQ_ans);
    for (int q = 0; q < Q; q++)
      PQ_ans[q] = PQ_ans[q] / sumweights;
  } else {
    RC_KronSums(y, N, Q, REAL(y), Q, DoSymmetric, ExpInf, ExpInf, DoCenter, weights,
              subset, offset, Nsubset, PQ_ans);
    for (int q = 0; q < Q * (Q + 1) / 2; q++)
      PQ_ans[q] = PQ_ans[q] / sumweights;
  }
}

```

Fragment referenced in 82a.

Defines: RC\_CovarianceInfluence 37b, 47, 87ab.

Uses: DoCenter 22b, DoSymmetric 22b, N 24bc, Nsubset 27c, offset 27d, Power2 22b, Q 25e, RC\_colSums 114a, RC\_KronSums 101a, subset 27be, 28a, sumweights 27a, weights 26c, weights, 26de, y 25d, 26ab.

### 3.8.3 X

$\langle R\_ExpectationX\ Prototype\ 88b \rangle \equiv$

```

SEXP R_ExpectationX
(
   $\langle R\ x\ Input\ 24d \rangle$ 
  SEXP P,
   $\langle R\ weights\ Input\ 26c \rangle$ ,
   $\langle R\ subset\ Input\ 27b \rangle$ 
)

```

Fragment referenced in 23b, 89a.

Uses: P 25a, R\_ExpectationX 89a.

$\langle R\_ExpectationX \text{ 89a} \rangle \equiv$

```

 $\langle R\_ExpectationX \text{ Prototype 88b} \rangle$ 
{
    SEXP ans;
     $\langle C \text{ integer } N \text{ Input 24c} \rangle$ ;
     $\langle C \text{ integer } N_{\text{subset}} \text{ Input 27c} \rangle$ ;

    N = XLENGTH(x) / INTEGER(P)[0];
    Nsubset = XLENGTH(subset);

    PROTECT(ans = allocVector(REALSXP, INTEGER(P)[0]));
    RC_ExpectationX(x, N, INTEGER(P)[0], weights, subset,
                    Offset0, Nsubset, REAL(ans));
    UNPROTECT(1);
    return(ans);
}

```

Fragment referenced in 82a.

Defines: `R_ExpectationX` 88b, 92a, 164, 165.

Uses: `N` 24bc, `Nsubset` 27c, `Offset0` 22b, `P` 25a, `RC_ExpectationX` 90, `subset` 27be, 28a, `weights` 26c, `weights`, 26de, `x` 24d, 25bc.

$\langle RC\_ExpectationX \text{ Prototype 89b} \rangle \equiv$

```

void RC_ExpectationX
(
     $\langle R \text{ } x \text{ Input 24d} \rangle$ 
     $\langle C \text{ integer } N \text{ Input 24c} \rangle$ ,
     $\langle C \text{ integer } P \text{ Input 25a} \rangle$ ,
     $\langle R \text{ weights Input 26c} \rangle$ ,
     $\langle R \text{ subset Input 27b} \rangle$ ,
     $\langle C \text{ subset range Input 27d} \rangle$ ,
     $\langle C \text{ OneTableSums Answer 119c} \rangle$ 
)

```

Fragment referenced in 90.

Uses: `RC_ExpectationX` 90.

$\langle RC\_ExpectationX\ 90 \rangle \equiv$

```

 $\langle RC\_ExpectationX\ Prototype\ 89b \rangle$ 
{
    double center;

    if (TYPEOF(x) == INTXP) {
        double* Pp1tmp = Calloc(P + 1, double);
        RC_OneTableSums(INTEGER(x), N, P + 1, weights, subset, offset, Nsubset, Pp1tmp);
        for (int p = 0; p < P; p++) P_ans[p] = Pp1tmp[p + 1];
        Free(Pp1tmp);
    } else {
        RC_colSums(REAL(x), N, P, Power1, &center, !DoCenter, weights, subset, offset, Nsubset, P_ans);
    }
}
◇

```

Fragment referenced in [82a](#).

Defines: [RC\\_ExpectationX 37a](#), [46c](#), [89ab](#).

Uses: [DoCenter 22b](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#), [P 25a](#), [Power1 22b](#), [RC\\_colSums 114a](#), [RC\\_OneTableSums 119a](#), [subset 27be](#), [28a](#), [weights 26c](#), [weights, 26de](#), [x 24d](#), [25bc](#).

```

> a0 <- colSums(x[subset, ] * weights[subset])
> a0

```

```

[1] 59.67771 31.68129 47.29375

```

```

> a1 <- .Call(libcoin:::R_ExpectationX, x, P, weights, subset);
> a2 <- .Call(libcoin:::R_ExpectationX, x, P, as.double(weights), as.double(subset));
> a3 <- .Call(libcoin:::R_ExpectationX, x, P, weights, as.double(subset));
> a4 <- .Call(libcoin:::R_ExpectationX, x, P, as.double(weights), subset);
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4) &&
+           isequal(a0, LECVxyws$ExpectationX))
> a0 <- colSums(x[subset, ]^2 * weights[subset])
> a1 <- .Call(libcoin:::R_CovarianceX, x, P, weights, subset, 1L);
> a2 <- .Call(libcoin:::R_CovarianceX, x, P, as.double(weights), as.double(subset), 1L);
> a3 <- .Call(libcoin:::R_CovarianceX, x, P, weights, as.double(subset), 1L);
> a4 <- .Call(libcoin:::R_CovarianceX, x, P, as.double(weights), subset, 1L);
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
> a0 <- as.vector(colSums(Xfactor[subset, ] * weights[subset]))
> a0

```

```

[1] 0 15 1 4 9 2 20 6 0 15

```

```

> a1 <- .Call(libcoin:::R_ExpectationX, ix, Lx, weights, subset);
> a2 <- .Call(libcoin:::R_ExpectationX, ix, Lx, as.double(weights), as.double(subset));
> a3 <- .Call(libcoin:::R_ExpectationX, ix, Lx, weights, as.double(subset));
> a4 <- .Call(libcoin:::R_ExpectationX, ix, Lx, as.double(weights), subset);
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
> a1 <- .Call(libcoin:::R_CovarianceX, ix, Lx, weights, subset, 1L);
> a2 <- .Call(libcoin:::R_CovarianceX, ix, Lx, as.double(weights), as.double(subset), 1L);
> a3 <- .Call(libcoin:::R_CovarianceX, ix, Lx, weights, as.double(subset), 1L);
> a4 <- .Call(libcoin:::R_CovarianceX, ix, Lx, as.double(weights), subset, 1L);

```

```

> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
> r1x <- rep(1:ncol(Xfactor), ncol(Xfactor))
> r2x <- rep(1:ncol(Xfactor), each = ncol(Xfactor))
> a0 <- colSums(Xfactor[subset, r1x] * Xfactor[subset, r2x] * weights[subset])
> a0 <- matrix(a0, ncol = ncol(Xfactor))
> vary <- diag(a0)
> a0 <- a0[lower.tri(a0, diag = TRUE)]
> a1 <- .Call(libcoin:::R_CovarianceX, ix, Lx, weights, subset, 0L)
> a2 <- .Call(libcoin:::R_CovarianceX, ix, Lx, as.double(weights), as.double(subset), 0L)
> a3 <- .Call(libcoin:::R_CovarianceX, ix, Lx, weights, as.double(subset), 0L)
> a4 <- .Call(libcoin:::R_CovarianceX, ix, Lx, as.double(weights), subset, 0L)
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))

```

$\langle R\_CovarianceX \text{ Prototype } 91 \rangle \equiv$

```

SEXP R_CovarianceX
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
  SEXP P,
   $\langle R \text{ } weights \text{ Input } 26c \rangle$ ,
   $\langle R \text{ } subset \text{ Input } 27b \rangle$ ,
  SEXP varonly
)
◇

```

Fragment referenced in [23b](#), [92a](#).

Uses: P [25a](#), R\_CovarianceX [92a](#).

$\langle R\_CovarianceX\ 92a \rangle \equiv$

```

 $\langle R\_CovarianceX\ Prototype\ 91 \rangle$ 
{
    SEXP ans;
    SEXP ExpX;
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;

    N = XLENGTH(x) / INTEGER(P)[0];
    Nsubset = XLENGTH(subset);

    PROTECT(ExpX = R_ExpectationX(x, P, weights, subset));

    if (INTEGER(varonly)[0]) {
        PROTECT(ans = allocVector(REALSXP, INTEGER(P)[0]));
    } else {
        PROTECT(ans = allocVector(REALSXP, INTEGER(P)[0] * (INTEGER(P)[0] + 1) / 2));
    }
    RC_CovarianceX(x, N, INTEGER(P)[0], weights, subset, Offset0, Nsubset, REAL(ExpX),
        INTEGER(varonly)[0], REAL(ans));
    UNPROTECT(2);
    return(ans);
}

```

Fragment referenced in 82a.

Defines: R\_CovarianceX 91, 164, 165.

Uses: N 24bc, Nsubset 27c, Offset0 22b, P 25a, RC\_CovarianceX 93a, R\_ExpectationX 89a, subset 27bc, 28a, weights 26c, weights, 26de, x 24d, 25bc.

$\langle RC\_CovarianceX\ Prototype\ 92b \rangle \equiv$

```

void RC_CovarianceX
(
     $\langle R\ x\ Input\ 24d \rangle$ 
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
     $\langle C\ integer\ P\ Input\ 25a \rangle$ ,
     $\langle R\ weights\ Input\ 26c \rangle$ ,
     $\langle R\ subset\ Input\ 27b \rangle$ ,
     $\langle C\ subset\ range\ Input\ 27d \rangle$ ,
    double *ExpX,
    int VARONLY,
     $\langle C\ KronSums\ Answer\ 101d \rangle$ 
)

```

Fragment referenced in 93a.

Uses: RC\_CovarianceX 93a.

$\langle RC\_CovarianceX\ 93a \rangle \equiv$

```

 $\langle RC\_CovarianceX\ Prototype\ 92b \rangle$ 
{
    double center;

    if (TYPEOF(x) == INTSXP) {
        if (VARONLY) {
            for (int p = 0; p < P; p++) PQ_ans[p] = ExpX[p];
        } else {
            for (int p = 0; p < PP12(P); p++)
                PQ_ans[p] = 0.0;
            for (int p = 0; p < P; p++)
                PQ_ans[S(p, p, P)] = ExpX[p];
        }
    } else {
        if (VARONLY) {
            RC_colSums(REAL(x), N, P, Power2, &center, !DoCenter, weights,
                      subset, offset, Nsubset, PQ_ans);
        } else {
            RC_KronSums(x, N, P, REAL(x), P, DoSymmetric, &center, &center, !DoCenter, weights,
                       subset, offset, Nsubset, PQ_ans);
        }
    }
}

```

Fragment referenced in [82a](#).

Defines: [RC\\_CovarianceX 37c, 38a, 47, 92ab](#).

Uses: [DoCenter 22b](#), [DoSymmetric 22b](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#), [P 25a](#), [Power2 22b](#), [PP12 140b](#), [RC\\_colSums 114a](#), [RC\\_KronSums 101a](#), [S 22a](#), [subset 27be, 28a](#), [weights 26c](#), [weights, 26de](#), [x 24d, 25bc](#).

### 3.9 Computing Sums

The core concept of all functions in the section is the computation of various sums over observations, weights, or blocks. We start with an initialisation of the loop over all observations

$\langle init\ subset\ loop\ 93b \rangle \equiv$

```

R_xlen_t diff = 0;
s = subset + offset;
w = weights;
/* subset is R-style index in 1:N */
if (Nsubset > 0)
    diff = (R_xlen_t) s[0] - 1;

```

Fragment referenced in [98a, 105, 108, 116b, 121b, 126, 131a](#).

Uses: [N 24bc](#), [Nsubset 27c](#), [offset 27d](#), [subset 27be, 28a](#), [weights 26c](#).

and loop over  $i = 1, \dots, N$  when no subset was specified or over the subset of the subset given by [offset](#) and [Nsubset](#), allowing for number of observations larger than [INT\\_MAX](#)



$\langle \text{start subset loop 94a} \rangle \equiv$

```
for (R_xlen_t i = 0; i < (Nsubset == 0 ? N : Nsubset) - 1; i++)
  ◇
```

Fragment referenced in [98a](#), [105](#), [108](#), [116b](#), [121b](#), [126](#), [131a](#).

Uses: [N 24bc](#), [Nsubset 27c](#).

After computations in the loop, we compute the next element

$\langle \text{continue subset loop 94b} \rangle \equiv$

```
if (Nsubset > 0) {
  /* NB: diff also works with R style index */
  diff = (R_xlen_t) s[1] - s[0];
  if (diff < 0)
    error("subset not sorted");
  s++;
} else {
  diff = 1;
}
◇
```

Fragment referenced in [98a](#), [105](#), [108](#), [116b](#), [121b](#), [126](#), [131a](#).

Uses: [Nsubset 27c](#), [subset 27be](#), [28a](#).

### 3.9.1 Simple Sums

$\langle \text{SimpleSums 94c} \rangle \equiv$

```

  ◇ C_Sums_dweights_dsubset 96b
  ◇ C_Sums_iweights_dsubset 97a
  ◇ C_Sums_iweights_isubset 97b
  ◇ C_Sums_dweights_isubset 97c
  ◇ RC_Sums 96a
  ◇ R_Sums 95b
  ◇
```

Fragment referenced in [24a](#).

```

> a0 <- sum(weights[subset])
> a1 <- .Call(libcoin:::R_Sums, N, weights, subset)
> a2 <- .Call(libcoin:::R_Sums, N, as.double(weights), as.double(subset))
> a3 <- .Call(libcoin:::R_Sums, N, weights, as.double(subset))
> a4 <- .Call(libcoin:::R_Sums, N, as.double(weights), subset)
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
```

$\langle R\_Sums \text{ Prototype } 95a \rangle \equiv$

```
SEXP R_Sums
(
   $\langle R \text{ } N \text{ Input } 24b \rangle$ 
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ 
)
◇
```

Fragment referenced in 23b, 95b.

Uses: R\_Sums 95b.

$\langle R\_Sums \text{ } 95b \rangle \equiv$

```
 $\langle R\_Sums \text{ Prototype } 95a \rangle$ 
{
  SEXP ans;
   $\langle C \text{ integer Nsubset Input } 27c \rangle$ ;

  Nsubset = XLENGTH(subset);

  PROTECT(ans = allocVector(REALSXP, 1));
  REAL(ans)[0] = RC_Sums(INTEGER(N)[0], weights, subset, Offset0, Nsubset);
  UNPROTECT(1);

  return(ans);
}
◇
```

Fragment referenced in 94c.

Defines: R\_Sums 95a, 164, 165.

Uses: N 24bc, Nsubset 27c, Offset0 22b, RC\_Sums 96a, subset 27be, 28a, weights 26c, weights, 26de.

$\langle RC\_Sums \text{ Prototype } 95c \rangle \equiv$

```
double RC_Sums
(
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ,
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ ,
   $\langle C \text{ subset range Input } 27d \rangle$ 
)
◇
```

Fragment referenced in 96a.

Uses: RC\_Sums 96a.

$\langle RC\_Sums\ 96a \rangle \equiv$

```

 $\langle RC\_Sums\ Prototype\ 95c \rangle$ 
{
  if (XLENGTH(weights) == 0) {
    if (XLENGTH(subset) == 0) {
      return((double) N);
    } else {
      return((double) Nsubset);
    }
  }
  if (TYPEOF(weights) == INTSXP) {
    if (TYPEOF(subset) == INTSXP) {
      return(C_Sums_iweights_isubset(N, INTEGER(weights), XLENGTH(weights),
                                     INTEGER(subset), offset, Nsubset));
    } else {
      return(C_Sums_iweights_dsubset(N, INTEGER(weights), XLENGTH(weights),
                                     REAL(subset), offset, Nsubset));
    }
  } else {
    if (TYPEOF(subset) == INTSXP) {
      return(C_Sums_dweights_isubset(N, REAL(weights), XLENGTH(weights),
                                     INTEGER(subset), offset, Nsubset));
    } else {
      return(C_Sums_dweights_dsubset(N, REAL(weights), XLENGTH(weights),
                                     REAL(subset), offset, Nsubset));
    }
  }
}

```

Fragment referenced in [94c](#).

Defines: [RC\\_Sums 36ab](#), [85b](#), [87a](#), [95bc](#), [132b](#), [136a](#).

Uses: [C\\_Sums\\_dweights\\_dsubset 96b](#), [C\\_Sums\\_dweights\\_isubset 97c](#), [C\\_Sums\\_iweights\\_dsubset 97a](#),  
[C\\_Sums\\_iweights\\_isubset 97b](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#), [subset 27be](#), [28a](#), [weights 26c](#).

$\langle C\_Sums\_dweights\_dsubset\ 96b \rangle \equiv$

```

double C_Sums_dweights_dsubset
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ real\ subset\ Input\ 28a \rangle$ 
) {
  double *s, *w;
   $\langle Sums\ Body\ 98a \rangle$ 
}

```

Fragment referenced in [94c](#).

Defines: [C\\_Sums\\_dweights\\_dsubset 96a](#).

$\langle C\_Sums\_iweights\_dsubset\ 97a \rangle \equiv$

```
double C_Sums_iweights_dsubset
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ real\ subset\ Input\ 28a \rangle$ 
) {
  double *s;
  int *w;
   $\langle Sums\ Body\ 98a \rangle$ 
}
◇
```

Fragment referenced in 94c.

Defines: C\_Sums\_iweights\_dsubset 96a.

$\langle C\_Sums\_iweights\_isubset\ 97b \rangle \equiv$

```
double C_Sums_iweights_isubset
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ 
) {
  int *s, *w;
   $\langle Sums\ Body\ 98a \rangle$ 
}
◇
```

Fragment referenced in 94c.

Defines: C\_Sums\_iweights\_isubset 96a.

$\langle C\_Sums\_dweights\_isubset\ 97c \rangle \equiv$

```
double C_Sums_dweights_isubset
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ 
) {
  int *s;
  double *w;
   $\langle Sums\ Body\ 98a \rangle$ 
}
◇
```

Fragment referenced in 94c.

Defines: C\_Sums\_dweights\_isubset 96a.

$\langle \text{Sums Body } 98a \rangle \equiv$

```
double ans = 0.0;

if (Nsubset > 0) {
  if (!HAS_WEIGHTS) return((double) Nsubset);
} else {
  if (!HAS_WEIGHTS) return((double) N);
}

 $\langle \text{init subset loop } 93b \rangle$ 
 $\langle \text{start subset loop } 94a \rangle$ 
{
  w = w + diff;
  ans += w[0];
   $\langle \text{continue subset loop } 94b \rangle$ 
}
w = w + diff;
ans += w[0];

return(ans);
◇
```

Fragment referenced in [96b](#), [97abc](#).

Uses: HAS\_WEIGHTS [26de](#), N [24bc](#), Nsubset [27c](#).

### 3.9.2 Kronecker Sums

$\langle \text{KronSums } 98b \rangle \equiv$

```
 $\langle C\_KronSums\_dweights\_dsubset \ 103b \rangle$ 
 $\langle C\_KronSums\_iweights\_dsubset \ 104a \rangle$ 
 $\langle C\_KronSums\_iweights\_isubset \ 104b \rangle$ 
 $\langle C\_KronSums\_dweights\_isubset \ 104c \rangle$ 
 $\langle C\_XfactorKronSums\_dweights\_dsubset \ 106b \rangle$ 
 $\langle C\_XfactorKronSums\_iweights\_dsubset \ 106c \rangle$ 
 $\langle C\_XfactorKronSums\_iweights\_isubset \ 107a \rangle$ 
 $\langle C\_XfactorKronSums\_dweights\_isubset \ 107b \rangle$ 
 $\langle RC\_KronSums \ 101a \rangle$ 
 $\langle R\_KronSums \ 100a \rangle$ 
 $\langle C\_KronSums\_Permutation\_isubset \ 111a \rangle$ 
 $\langle C\_KronSums\_Permutation\_dsubset \ 110b \rangle$ 
 $\langle C\_XfactorKronSums\_Permutation\_isubset \ 112a \rangle$ 
 $\langle C\_XfactorKronSums\_Permutation\_dsubset \ 111c \rangle$ 
 $\langle RC\_KronSums\_Permutation \ 110a \rangle$ 
 $\langle R\_KronSums\_Permutation \ 109b \rangle$ 
◇
```

Fragment referenced in [24a](#).

```
> r1 <- rep(1:ncol(x), ncol(y))
> r2 <- rep(1:ncol(y), each = ncol(x))
> a0 <- colSums(x[subset,r1] * y[subset,r2] * weights[subset])
> a1 <- .Call(libcoin:::R_KronSums, x, P, y, weights, subset, 0L)
> a2 <- .Call(libcoin:::R_KronSums, x, P, y, as.double(weights), as.double(subset), 0L)
> a3 <- .Call(libcoin:::R_KronSums, x, P, y, weights, as.double(subset), 0L)
```

```

> a4 <- .Call(libcoin:::R_KronSums, x, P, y, as.double(weights), subset, 0L)
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
> a0 <- as.vector(colSums(Xfactor[subset,r1Xfactor] *
+                         y[subset,r2Xfactor] * weights[subset]))
> a1 <- .Call(libcoin:::R_KronSums, ix, Lx, y, weights, subset, 0L)
> a2 <- .Call(libcoin:::R_KronSums, ix, Lx, y, as.double(weights), as.double(subset), 0L)
> a3 <- .Call(libcoin:::R_KronSums, ix, Lx, y, weights, as.double(subset), 0L)
> a4 <- .Call(libcoin:::R_KronSums, ix, Lx, y, as.double(weights), subset, 0L)
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))

```

$\langle R\_KronSums \text{ Prototype } 99 \rangle \equiv$

```

SEXP R_KronSums
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
  SEXP P,
   $\langle R \text{ } y \text{ Input } 25d \rangle$ 
   $\langle R \text{ } weights \text{ Input } 26c \rangle$ ,
   $\langle R \text{ } subset \text{ Input } 27b \rangle$ ,
  SEXP symmetric
)
◇

```

Fragment referenced in 23b, 100a.

Uses: P 25a, R\_KronSums 100a.

$\langle R\_KronSums\ 100a \rangle \equiv$

```

 $\langle R\_KronSums\ Prototype\ 99 \rangle$ 
{
    SEXP ans;
     $\langle C\ integer\ Q\ Input\ 25e \rangle$ ;
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;

    double center;

    Q = NCOL(y);
    N = XLENGTH(y) / Q;
    Nsubset = XLENGTH(subset);

    if (INTEGER(symmetric)[0]) {
        PROTECT(ans = allocVector(REALSXP, INTEGER(P)[0] * (INTEGER(P)[0] + 1) / 2));
    } else {
        PROTECT(ans = allocVector(REALSXP, INTEGER(P)[0] * Q));
    }
    RC_KronSums(x, N, INTEGER(P)[0], REAL(y), Q, INTEGER(symmetric)[0], &center, &center,
        !DoCenter, weights, subset, Offset0, Nsubset, REAL(ans));
    UNPROTECT(1);
    return(ans);
}

```

Fragment referenced in 98b.

Defines:  $R\_KronSums\ 99, 164, 165$ .

Uses: DoCenter 22b, N 24bc, NCOL 139c, Nsubset 27c, Offset0 22b, P 25a, Q 25e,  $RC\_KronSums\ 101a$ , subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

$\langle RC\_KronSums\ Prototype\ 100b \rangle \equiv$

```

void RC_KronSums
(
     $\langle RC\ KronSums\ Input\ 101b \rangle$ 
     $\langle R\ weights\ Input\ 26c \rangle$ ,
     $\langle R\ subset\ Input\ 27b \rangle$ ,
     $\langle C\ subset\ range\ Input\ 27d \rangle$ ,
     $\langle C\ KronSums\ Answer\ 101d \rangle$ 
)

```

Fragment referenced in 101a.

Uses:  $RC\_KronSums\ 101a$ .

$\langle RC\_KronSums\ 101a \rangle \equiv$

```

     $\langle RC\_KronSums\ Prototype\ 100b \rangle$ 
    {
        if (typeof(x) == INTSXP) {
             $\langle KronSums\ Integer\ x\ 102 \rangle$ 
        } else {
             $\langle KronSums\ Double\ x\ 103a \rangle$ 
        }
    }
     $\diamond$ 

```

Fragment referenced in 98b.

Defines: RC\_KronSums 81d, 88a, 93a, 100ab.

Uses: x 24d, 25bc.

$\langle RC\ KronSums\ Input\ 101b \rangle \equiv$

```

     $\langle R\ x\ Input\ 24d \rangle$ 
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
     $\langle C\ integer\ P\ Input\ 25a \rangle$ ,
     $\langle C\ real\ y\ Input\ 26a \rangle$ 
    const int SYMMETRIC,
    double *centerx,
    double *centery,
    const int CENTER,
     $\diamond$ 

```

Fragment referenced in 100b.

$\langle C\ KronSums\ Input\ 101c \rangle \equiv$

```

     $\langle C\ real\ x\ Input\ 25b \rangle$ 
     $\langle C\ real\ y\ Input\ 26a \rangle$ 
    const int SYMMETRIC,
    double *centerx,
    double *centery,
    const int CENTER,
     $\diamond$ 

```

Fragment referenced in 103b, 104abc.

$\langle C\ KronSums\ Answer\ 101d \rangle \equiv$

```

    double *PQ_ans
     $\diamond$ 

```

Fragment referenced in 81c, 87b, 92b, 100b, 103b, 104abc, 106bc, 107ab, 109c, 110b, 111ac, 112a.



$\langle \text{KronSums Integer } x \text{ 102} \rangle \equiv$

```

if (SYMMETRIC) error("not implemented");
if (CENTER) error("not implemented");
if (TYPEOF(weights) == INTSXP) {
  if (TYPEOF(subset) == INTSXP) {
    C_XfactorKronSums_iweights_isubset(INTEGER(x), N, P, y, Q,
      INTEGER(weights), XLENGTH(weights) > 0, INTEGER(subset),
      offset, Nsubset, PQ_ans);
  } else {
    C_XfactorKronSums_iweights_dsubset(INTEGER(x), N, P, y, Q,
      INTEGER(weights), XLENGTH(weights) > 0, REAL(subset),
      offset, Nsubset, PQ_ans);
  }
} else {
  if (TYPEOF(subset) == INTSXP) {
    C_XfactorKronSums_dweights_isubset(INTEGER(x), N, P, y, Q,
      REAL(weights), XLENGTH(weights) > 0, INTEGER(subset),
      offset, Nsubset, PQ_ans);
  } else {
    C_XfactorKronSums_dweights_dsubset(INTEGER(x), N, P, y, Q,
      REAL(weights), XLENGTH(weights) > 0, REAL(subset),
      offset, Nsubset, PQ_ans);
  }
}
}
◇

```

Fragment referenced in [101a](#).

Uses: C\_XfactorKronSums\_dweights\_dsubset [106b](#), C\_XfactorKronSums\_dweights\_isubset [107b](#),  
 C\_XfactorKronSums\_iweights\_dsubset [106c](#), C\_XfactorKronSums\_iweights\_isubset [107a](#), N [24bc](#), Nsubset [27c](#),  
 offset [27d](#), P [25a](#), Q [25e](#), subset [27be](#), [28a](#), weights [26c](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

$\langle \text{KronSums Double } x \text{ 103a} \rangle \equiv$

```

if (typeof(weights) == INTSXP) {
  if (typeof(subset) == INTSXP) {
    C_KronSums_iweights_isubset(REAL(x), N, P, y, Q, SYMMETRIC, centerx, centery, CENTER,
      INTEGER(weights), XLENGTH(weights) > 0, INTEGER(subset),
      offset, Nsubset, PQ_ans);
  } else {
    C_KronSums_iweights_dsubset(REAL(x), N, P, y, Q, SYMMETRIC, centerx, centery, CENTER,
      INTEGER(weights), XLENGTH(weights) > 0, REAL(subset),
      offset, Nsubset, PQ_ans);
  }
} else {
  if (typeof(subset) == INTSXP) {
    C_KronSums_dweights_isubset(REAL(x), N, P, y, Q, SYMMETRIC, centerx, centery, CENTER,
      REAL(weights), XLENGTH(weights) > 0, INTEGER(subset),
      offset, Nsubset, PQ_ans);
  } else {
    C_KronSums_dweights_dsubset(REAL(x), N, P, y, Q, SYMMETRIC, centerx, centery, CENTER,
      REAL(weights), XLENGTH(weights) > 0, REAL(subset),
      offset, Nsubset, PQ_ans);
  }
}

```

Fragment referenced in 101a.

Uses: C\_KronSums\_dweights\_dsubset 103b, C\_KronSums\_dweights\_isubset 104c, C\_KronSums\_iweights\_dsubset 104a,  
 C\_KronSums\_iweights\_isubset 104b, N 24bc, Nsubset 27c, offset 27d, P 25a, Q 25e, subset 27be, 28a, weights 26c,  
 x 24d, 25bc, y 25d, 26ab.

$\langle \text{C\_KronSums\_dweights\_dsubset 103b} \rangle \equiv$

```

void C_KronSums_dweights_dsubset
(
   $\langle \text{C KronSums Input 101c} \rangle$ 
   $\langle \text{C real weights Input 26e} \rangle$ 
   $\langle \text{C real subset Input 28a} \rangle$ ,
   $\langle \text{C KronSums Answer 101d} \rangle$ 
) {
  double *s, *w;
   $\langle \text{KronSums Body 105} \rangle$ 
}

```

Fragment referenced in 98b.

Defines: C\_KronSums\_dweights\_dsubset 103a.

$\langle C\_KronSums\_iweights\_dsubset\ 104a \rangle \equiv$

```
void C_KronSums_iweights_dsubset
(
   $\langle C\ KronSums\ Input\ 101c \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\ KronSums\ Answer\ 101d \rangle$ 
) {
  double *s;
  int *w;
   $\langle KronSums\ Body\ 105 \rangle$ 
}
◇
```

Fragment referenced in 98b.

Defines: C\_KronSums\_iweights\_dsubset 103a.

$\langle C\_KronSums\_iweights\_isubset\ 104b \rangle \equiv$

```
void C_KronSums_iweights_isubset
(
   $\langle C\ KronSums\ Input\ 101c \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ KronSums\ Answer\ 101d \rangle$ 
) {
  int *s, *w;
   $\langle KronSums\ Body\ 105 \rangle$ 
}
◇
```

Fragment referenced in 98b.

Defines: C\_KronSums\_iweights\_isubset 103a.

$\langle C\_KronSums\_dweights\_isubset\ 104c \rangle \equiv$

```
void C_KronSums_dweights_isubset
(
   $\langle C\ KronSums\ Input\ 101c \rangle$ 
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ KronSums\ Answer\ 101d \rangle$ 
) {
  int *s;
  double *w;
   $\langle KronSums\ Body\ 105 \rangle$ 
}
◇
```

Fragment referenced in 98b.

Defines: C\_KronSums\_dweights\_isubset 103a.

$\langle \text{KronSums Body 105} \rangle \equiv$

```

double *xx, *yy, cx = 0.0, cy = 0.0, *thisPQ_ans;
int idx;

for (int p = 0; p < P; p++) {
  for (int q = (SYMMETRIC ? p : 0); q < Q; q++) {
    /* SYMMETRIC is column-wise, default
       is row-wise (maybe need to change this) */
    if (SYMMETRIC) {
      idx = S(p, q, P);
    } else {
      idx = q * P + p;
    }
    PQ_ans[idx] = 0.0;
    thisPQ_ans = PQ_ans + idx;
    yy = y + N * q;
    xx = x + N * p;

    if (CENTER) {
      cx = centerx[p];
      cy = centery[q];
    }
     $\langle \text{init subset loop 93b} \rangle$ 
     $\langle \text{start subset loop 94a} \rangle$ 
    {
      xx = xx + diff;
      yy = yy + diff;
      if (HAS_WEIGHTS) {
        w = w + diff;
        if (CENTER) {
          thisPQ_ans[0] += (xx[0] - cx) * (yy[0] - cy) * w[0];
        } else {
          thisPQ_ans[0] += xx[0] * yy[0] * w[0];
        }
      } else {
        if (CENTER) {
          thisPQ_ans[0] += (xx[0] - cx) * (yy[0] - cy);
        } else {
          thisPQ_ans[0] += xx[0] * yy[0];
        }
      }
       $\langle \text{continue subset loop 94b} \rangle$ 
    }
    xx = xx + diff;
    yy = yy + diff;
    if (HAS_WEIGHTS) {
      w = w + diff;
      thisPQ_ans[0] += (xx[0] - cx) * (yy[0] - cy) * w[0];
    } else {
      thisPQ_ans[0] += (xx[0] - cx) * (yy[0] - cy);
    }
  }
}

```

Fragment referenced in [103b](#), [104abc](#).

Uses: HAS\_WEIGHTS [26de](#), N [24bc](#), P [25a](#), Q [25e](#), S [22a](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

## Xfactor Kronecker Sums

$\langle C\_XfactorKronSums\_Input\ 106a \rangle \equiv$

$\langle C\_integer\ x\ Input\ 25c \rangle$   
 $\langle C\_real\ y\ Input\ 26a \rangle$   
 $\diamond$

Fragment referenced in [106bc](#), [107ab](#).

$\langle C\_XfactorKronSums\_dweights\_dsubset\ 106b \rangle \equiv$

```
void C_XfactorKronSums_dweights_dsubset
(
   $\langle C\_XfactorKronSums\_Input\ 106a \rangle$ 
   $\langle C\_real\ weights\ Input\ 26e \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_KronSums\_Answer\ 101d \rangle$ 
) {
  double *s, *w;
   $\langle XfactorKronSums\_Body\ 108 \rangle$ 
}
```

$\diamond$

Fragment referenced in [98b](#).

Defines: `C_XfactorKronSums_dweights_dsubset` [102](#).

$\langle C\_XfactorKronSums\_iweights\_dsubset\ 106c \rangle \equiv$

```
void C_XfactorKronSums_iweights_dsubset
(
   $\langle C\_XfactorKronSums\_Input\ 106a \rangle$ 
   $\langle C\_integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_KronSums\_Answer\ 101d \rangle$ 
) {
  double *s;
  int *w;
   $\langle XfactorKronSums\_Body\ 108 \rangle$ 
}
```

$\diamond$

Fragment referenced in [98b](#).

Defines: `C_XfactorKronSums_iweights_dsubset` [102](#).

$\langle C\_XfactorKronSums\_iweights\_isubset \ 107a \rangle \equiv$

```
void C_XfactorKronSums_iweights_isubset
(
     $\langle C \ XfactorKronSums \ Input \ 106a \rangle$ 
     $\langle C \ integer \ weights \ Input \ 26d \rangle$ 
     $\langle C \ integer \ subset \ Input \ 27e \rangle$ ,
     $\langle C \ KronSums \ Answer \ 101d \rangle$ 
) {
    int *s, *w;
     $\langle XfactorKronSums \ Body \ 108 \rangle$ 
}
◇
```

Fragment referenced in [98b](#).

Defines: C\_XfactorKronSums\_iweights\_isubset [102](#).

$\langle C\_XfactorKronSums\_dweights\_isubset \ 107b \rangle \equiv$

```
void C_XfactorKronSums_dweights_isubset
(
     $\langle C \ XfactorKronSums \ Input \ 106a \rangle$ 
     $\langle C \ real \ weights \ Input \ 26e \rangle$ 
     $\langle C \ integer \ subset \ Input \ 27e \rangle$ ,
     $\langle C \ KronSums \ Answer \ 101d \rangle$ 
) {
    int *s;
    double *w;
     $\langle XfactorKronSums \ Body \ 108 \rangle$ 
}
◇
```

Fragment referenced in [98b](#).

Defines: C\_XfactorKronSums\_dweights\_isubset [102](#).

$\langle XfactorKronSums \text{ Body } 108 \rangle \equiv$

```

int *xx, ixi;
double *yy;

for (int p = 0; p < mPQB(P, Q, 1); p++) PQ_ans[p] = 0.0;

for (int q = 0; q < Q; q++) {
  yy = y + N * q;
  xx = x;
   $\langle \text{init subset loop } 93b \rangle$ 
   $\langle \text{start subset loop } 94a \rangle$ 
  {
    xx = xx + diff;
    yy = yy + diff;
    ixi = xx[0] - 1;
    if (HAS_WEIGHTS) {
      w = w + diff;
      if (ixi >= 0)
        PQ_ans[ixi + q * P] += yy[0] * w[0];
    } else {
      if (ixi >= 0)
        PQ_ans[ixi + q * P] += yy[0];
    }
     $\langle \text{continue subset loop } 94b \rangle$ 
  }
  xx = xx + diff;
  yy = yy + diff;
  ixi = xx[0] - 1;
  if (HAS_WEIGHTS) {
    w = w + diff;
    if (ixi >= 0)
      PQ_ans[ixi + q * P] += yy[0] * w[0];
  } else {
    if (ixi >= 0)
      PQ_ans[ixi + q * P] += yy[0];
  }
}

```

Fragment referenced in [106bc](#), [107ab](#).

Uses: HAS\_WEIGHTS [26de](#), mPQB [141a](#), N [24bc](#), P [25a](#), Q [25e](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

## Permuted Kronecker Sums

```

> a0 <- colSums(x[subset,r1] * y[subsety, r2])
> a1 <- .Call(libcoin:::R_KronSums_Permutation, x, P, y, subset, subsety)
> a2 <- .Call(libcoin:::R_KronSums_Permutation, x, P, y, as.double(subset), as.double(subsety))
> stopifnot(isequal(a0, a1) && isequal(a0, a2))
> a0 <- as.vector(colSums(Xfactor[subset,r1Xfactor] * y[subsety, r2Xfactor]))
> a1 <- .Call(libcoin:::R_KronSums_Permutation, ix, Lx, y, subset, subsety)
> a2 <- .Call(libcoin:::R_KronSums_Permutation, ix, Lx, y, as.double(subset), as.double(subsety))
> stopifnot(isequal(a0, a1) && isequal(a0, a2))

```

$\langle R\_KronSums\_Permutation \text{ Prototype } 109a \rangle \equiv$

```
SEXP R_KronSums_Permutation
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
  SEXP P,
   $\langle R \text{ } y \text{ Input } 25d \rangle$ 
   $\langle R \text{ } subset \text{ Input } 27b \rangle$ ,
  SEXP subsety
)
◇
```

Fragment referenced in [23b](#), [109b](#).

Uses: P [25a](#), R\_KronSums\_Permutation [109b](#).

$\langle R\_KronSums\_Permutation \text{ } 109b \rangle \equiv$

```
 $\langle R\_KronSums\_Permutation \text{ Prototype } 109a \rangle$ 
{
  SEXP ans;
   $\langle C \text{ integer } Q \text{ Input } 25e \rangle$ ;
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ;
   $\langle C \text{ integer } Nsubset \text{ Input } 27c \rangle$ ;

  Q = NCOL(y);
  N = XLENGTH(y) / Q;
  Nsubset = XLENGTH(subset);

  PROTECT(ans = allocVector(REALSXP, INTEGER(P)[0] * Q));
  RC_KronSums_Permutation(x, N, INTEGER(P)[0], REAL(y), Q, subset, Offset0, Nsubset,
                        subsety, REAL(ans));
  UNPROTECT(1);
  return(ans);
}
◇
```

Fragment referenced in [98b](#).

Defines: R\_KronSums\_Permutation [109a](#), [164](#), [165](#).

Uses: N [24bc](#), NCOL [139c](#), Nsubset [27c](#), Offset0 [22b](#), P [25a](#), Q [25e](#), RC\_KronSums\_Permutation [110a](#), subset [27be](#), [28a](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

$\langle RC\_KronSums\_Permutation \text{ Prototype } 109c \rangle \equiv$

```
void RC_KronSums_Permutation
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ,
   $\langle C \text{ integer } P \text{ Input } 25a \rangle$ ,
   $\langle C \text{ real } y \text{ Input } 26a \rangle$ 
   $\langle R \text{ } subset \text{ Input } 27b \rangle$ ,
   $\langle C \text{ subset range Input } 27d \rangle$ ,
  SEXP subsety,
   $\langle C \text{ KronSums Answer } 101d \rangle$ 
)
◇
```

Fragment referenced in [110a](#).

Uses: RC\_KronSums\_Permutation [110a](#).



$\langle RC\_KronSums\_Permutation\ 110a \rangle \equiv$

```

 $\langle RC\_KronSums\_Permutation\ Prototype\ 109c \rangle$ 
{
    if (TYPEOF(x) == INTSXP) {
        if (TYPEOF(subset) == INTSXP) {
            C_XfactorKronSums_Permutation_isubset(INTEGER(x), N, P, y, Q,
                                                    INTEGER(subset), offset, Nsubset,
                                                    INTEGER(subsety), PQ_ans);
        } else {
            C_XfactorKronSums_Permutation_dsubset(INTEGER(x), N, P, y, Q,
                                                    REAL(subset), offset, Nsubset,
                                                    REAL(subsety), PQ_ans);
        }
    } else {
        if (TYPEOF(subset) == INTSXP) {
            C_KronSums_Permutation_isubset(REAL(x), N, P, y, Q,
                                            INTEGER(subset), offset, Nsubset,
                                            INTEGER(subsety), PQ_ans);
        } else {
            C_KronSums_Permutation_dsubset(REAL(x), N, P, y, Q,
                                            REAL(subset), offset, Nsubset,
                                            REAL(subsety), PQ_ans);
        }
    }
}

```

Fragment referenced in 98b.

Defines: RC\_KronSums\_Permutation 40, 109bc.

Uses: C\_KronSums\_Permutation\_dsubset 110b, C\_KronSums\_Permutation\_isubset 111a,  
C\_XfactorKronSums\_Permutation\_dsubset 111c, C\_XfactorKronSums\_Permutation\_isubset 112a, N 24bc, Nsubset 27c,  
offset 27d, P 25a, Q 25e, subset 27be, 28a, x 24d, 25bc, y 25d, 26ab.

$\langle C\_KronSums\_Permutation\_dsubset\ 110b \rangle \equiv$

```

void C_KronSums_Permutation_dsubset
(
     $\langle C\ real\ x\ Input\ 25b \rangle$ 
     $\langle C\ real\ y\ Input\ 26a \rangle$ 
     $\langle C\ real\ subset\ Input\ 28a \rangle$ ,
    double *subsety,
     $\langle C\ KronSums\ Answer\ 101d \rangle$ 
) {
     $\langle KronSums\ Permutation\ Body\ 111b \rangle$ 
}

```

Fragment referenced in 98b.

Defines: C\_KronSums\_Permutation\_dsubset 110a.

$\langle C\_KronSums\_Permutation\_isubset \text{ 111a} \rangle \equiv$

```
void C_KronSums_Permutation_isubset
(
   $\langle C \text{ real } x \text{ Input 25b} \rangle$ 
   $\langle C \text{ real } y \text{ Input 26a} \rangle$ 
   $\langle C \text{ integer subset Input 27e} \rangle$ ,
  int *subsety,
   $\langle C \text{ KronSums Answer 101d} \rangle$ 
) {
   $\langle KronSums \text{ Permutation Body 111b} \rangle$ 
}
◇
```

Fragment referenced in [98b](#).

Defines: C\_KronSums\_Permutation\_isubset [110a](#).

Because **subset** might not be ordered (in the presence of blocks) we have to go through all elements explicitly here.

$\langle KronSums \text{ Permutation Body 111b} \rangle \equiv$

```
R_xlen_t qP, qN, pN, qPp;

for (int q = 0; q < Q; q++) {
  qN = q * N;
  qP = q * P;
  for (int p = 0; p < P; p++) {
    qPp = qP + p;
    PQ_ans[qPp] = 0.0;
    pN = p * N;
    for (R_xlen_t i = offset; i < Nsubset; i++)
      PQ_ans[qPp] += y[qN + (R_xlen_t) subsety[i] - 1] *
                    x[pN + (R_xlen_t) subset[i] - 1];
  }
}
◇
```

Fragment referenced in [110b](#), [111a](#).

Uses: N [24bc](#), Nsubset [27c](#), offset [27d](#), P [25a](#), Q [25e](#), subset [27be](#), [28a](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

## Xfactor Permuted Kronecker Sums

$\langle C\_XfactorKronSums\_Permutation\_dsubset \text{ 111c} \rangle \equiv$

```
void C_XfactorKronSums_Permutation_dsubset
(
   $\langle C \text{ integer } x \text{ Input 25c} \rangle$ 
   $\langle C \text{ real } y \text{ Input 26a} \rangle$ 
   $\langle C \text{ real subset Input 28a} \rangle$ ,
  double *subsety,
   $\langle C \text{ KronSums Answer 101d} \rangle$ 
) {
   $\langle XfactorKronSums \text{ Permutation Body 112b} \rangle$ 
}
◇
```

Fragment referenced in [98b](#).

Defines: C\_XfactorKronSums\_Permutation\_dsubset [110a](#).

$\langle C\_XfactorKronSums\_Permutation\_isubset\ 112a \rangle \equiv$

```
void C_XfactorKronSums_Permutation_isubset
(
   $\langle C\ integer\ x\ Input\ 25c \rangle$ 
   $\langle C\ real\ y\ Input\ 26a \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
  int *subsety,
   $\langle C\ KronSums\ Answer\ 101d \rangle$ 
) {
   $\langle XfactorKronSums\ Permutation\ Body\ 112b \rangle$ 
}
◇
```

Fragment referenced in 98b.

Defines: C\_XfactorKronSums\_Permutation\_isubset 110a.

$\langle XfactorKronSums\ Permutation\ Body\ 112b \rangle \equiv$

```
R_xlen_t qP, qN;

for (int p = 0; p < mPQB(P, Q, 1); p++) PQ_ans[p] = 0.0;

for (int q = 0; q < Q; q++) {
  qP = q * P;
  qN = q * N;
  for (R_xlen_t i = offset; i < Nsubset; i++)
    PQ_ans[x[(R_xlen_t) subset[i] - 1] - 1 + qP] += y[qN + (R_xlen_t) subsety[i] - 1];
}
◇
```

Fragment referenced in 111c, 112a.

Uses: mPQB 141a, N 24bc, Nsubset 27c, offset 27d, P 25a, Q 25e, subset 27be, 28a, x 24d, 25bc, y 25d, 26ab.

### 3.9.3 Column Sums

$\langle colSums\ 112c \rangle \equiv$

```
 $\langle C\_colSums\_dweights\_dsubset\ 115a \rangle$ 
 $\langle C\_colSums\_iweights\_dsubset\ 115b \rangle$ 
 $\langle C\_colSums\_iweights\_isubset\ 115c \rangle$ 
 $\langle C\_colSums\_dweights\_isubset\ 116a \rangle$ 
 $\langle RC\_colSums\ 114a \rangle$ 
 $\langle R\_colSums\ 113b \rangle$ 
◇
```

Fragment referenced in 24a.

```
> a0 <- colSums(x[subset,] * weights[subset])
> a1 <- .Call(libcoin:::R_colSums, x, weights, subset)
> a2 <- .Call(libcoin:::R_colSums, x, as.double(weights), as.double(subset))
> a3 <- .Call(libcoin:::R_colSums, x, weights, as.double(subset))
> a4 <- .Call(libcoin:::R_colSums, x, as.double(weights), subset)
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
```

$\langle R\_colSums \text{ Prototype } 113a \rangle \equiv$

```
SEXP R_colSums
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ 
)
◇
```

Fragment referenced in 23b, 113b.

Uses: R\_colSums 113b.

$\langle R\_colSums \text{ } 113b \rangle \equiv$

```
 $\langle R\_colSums \text{ Prototype } 113a \rangle$ 
{
  SEXP ans;
  int P;
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ;
   $\langle C \text{ integer } Nsubset \text{ Input } 27c \rangle$ ;
  double center;

  P = NCOL(x);
  N = XLENGTH(x) / P;
  Nsubset = XLENGTH(subset);

  PROTECT(ans = allocVector(REALSXP, P));
  RC_colSums(REAL(x), N, P, Power1, &center, !DoCenter, weights, subset, Offset0,
             Nsubset, REAL(ans));
  UNPROTECT(1);
  return(ans);
}
◇
```

Fragment referenced in 112c.

Defines: R\_colSums 113a, 164, 165.

Uses: DoCenter 22b, N 24bc, NCOL 139c, Nsubset 27c, Offset0 22b, P 25a, Power1 22b, RC\_colSums 114a, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc.

$\langle RC\_colSums \text{ Prototype } 113c \rangle \equiv$

```
void RC_colSums
(
   $\langle C \text{ colSums Input } 114b \rangle$ 
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ ,
   $\langle C \text{ subset range Input } 27d \rangle$ ,
   $\langle C \text{ colSums Answer } 114c \rangle$ 
)
◇
```

Fragment referenced in 114a.

Uses: RC\_colSums 114a.

$\langle RC\_colSums\ 114a \rangle \equiv$

```

 $\langle RC\_colSums\ Prototype\ 113c \rangle$ 
{
  if (TYPEOF(weights) == INTSXP) {
    if (TYPEOF(subset) == INTSXP) {
      C_colSums_iweights_isubset(x, N, P, power, centerx, CENTER,
                                INTEGER(weights), XLENGTH(weights) > 0, INTEGER(subset),
                                offset, Nsubset, P_ans);
    } else {
      C_colSums_iweights_dsubset(x, N, P, power, centerx, CENTER,
                                INTEGER(weights), XLENGTH(weights) > 0, REAL(subset),
                                offset, Nsubset, P_ans);
    }
  } else {
    if (TYPEOF(subset) == INTSXP) {
      C_colSums_dweights_isubset(x, N, P, power, centerx, CENTER,
                                REAL(weights), XLENGTH(weights) > 0, INTEGER(subset),
                                offset, Nsubset, P_ans);
    } else {
      C_colSums_dweights_dsubset(x, N, P, power, centerx, CENTER,
                                REAL(weights), XLENGTH(weights) > 0, REAL(subset),
                                offset, Nsubset, P_ans);
    }
  }
}

```

Fragment referenced in [112c](#).

Defines: [RC\\_colSums 86a, 88a, 90, 93a, 113bc](#).

Uses: [C\\_colSums\\_dweights\\_dsubset 115a](#), [C\\_colSums\\_dweights\\_isubset 116a](#), [C\\_colSums\\_iweights\\_dsubset 115b](#),  
[C\\_colSums\\_iweights\\_isubset 115c](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#), [P 25a](#), [subset 27be, 28a](#), [weights 26c](#), [x 24d, 25bc](#).

$\langle C\ colSums\ Input\ 114b \rangle \equiv$

```

 $\langle C\ real\ x\ Input\ 25b \rangle$ 
const int power,
double *centerx,
const int CENTER,

```

Fragment referenced in [113c, 115abc, 116a](#).

$\langle C\ colSums\ Answer\ 114c \rangle \equiv$

```

double *P_ans

```

Fragment referenced in [85c, 113c, 115abc, 116a](#).

$\langle C\_colSums\_dweights\_dsubset\ 115a \rangle \equiv$

```
void C_colSums_dweights_dsubset
(
     $\langle C\ colSums\ Input\ 114b \rangle$ 
     $\langle C\ real\ weights\ Input\ 26e \rangle$ 
     $\langle C\ real\ subset\ Input\ 28a \rangle$ ,
     $\langle C\ colSums\ Answer\ 114c \rangle$ 
) {
    double *s, *w;
     $\langle colSums\ Body\ 116b \rangle$ 
}
◇
```

Fragment referenced in [112c](#).

Defines: `C_colSums_dweights_dsubset` [114a](#).

$\langle C\_colSums\_iweights\_dsubset\ 115b \rangle \equiv$

```
void C_colSums_iweights_dsubset
(
     $\langle C\ colSums\ Input\ 114b \rangle$ 
     $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
     $\langle C\ real\ subset\ Input\ 28a \rangle$ ,
     $\langle C\ colSums\ Answer\ 114c \rangle$ 
) {
    double *s;
    int *w;
     $\langle colSums\ Body\ 116b \rangle$ 
}
◇
```

Fragment referenced in [112c](#).

Defines: `C_colSums_iweights_dsubset` [114a](#).

$\langle C\_colSums\_iweights\_isubset\ 115c \rangle \equiv$

```
void C_colSums_iweights_isubset
(
     $\langle C\ colSums\ Input\ 114b \rangle$ 
     $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
     $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
     $\langle C\ colSums\ Answer\ 114c \rangle$ 
) {
    int *s, *w;
     $\langle colSums\ Body\ 116b \rangle$ 
}
◇
```

Fragment referenced in [112c](#).

Defines: `C_colSums_iweights_isubset` [114a](#).

$\langle C\_colSums\_dweights\_isubset \ 116a \rangle \equiv$

```
void C_colSums_dweights_isubset
(
   $\langle C \ colSums \ Input \ 114b \rangle$ 
   $\langle C \ real \ weights \ Input \ 26e \rangle$ 
   $\langle C \ integer \ subset \ Input \ 27e \rangle$ ,
   $\langle C \ colSums \ Answer \ 114c \rangle$ 
) {
  int *s;
  double *w;
   $\langle colSums \ Body \ 116b \rangle$ 
}
◇
```

Fragment referenced in [112c](#).

Defines: C\_colSums\_dweights\_isubset [114a](#).

$\langle colSums \ Body \ 116b \rangle \equiv$

```
double *xx, cx = 0.0;

for (int p = 0; p < P; p++) {
  P_ans[0] = 0.0;
  xx = x + N * p;
  if (CENTER) {
    cx = centerx[p];
  }
   $\langle init \ subset \ loop \ 93b \rangle$ 
   $\langle start \ subset \ loop \ 94a \rangle$ 
  {
    xx = xx + diff;
    if (HAS_WEIGHTS) {
      w = w + diff;
      P_ans[0] += pow(xx[0] - cx, power) * w[0];
    } else {
      P_ans[0] += pow(xx[0] - cx, power);
    }
     $\langle continue \ subset \ loop \ 94b \rangle$ 
  }
  xx = xx + diff;
  if (HAS_WEIGHTS) {
    w = w + diff;
    P_ans[0] += pow(xx[0] - cx, power) * w[0];
  } else {
    P_ans[0] += pow(xx[0] - cx, power);
  }
  P_ans++;
}
◇
```

Fragment referenced in [115abc](#), [116a](#).

Uses: HAS\_WEIGHTS [26de](#), N [24bc](#), P [25a](#), x [24d](#), [25bc](#).

### 3.9.4 Tables

#### OneTable Sums

$\langle \text{Tables 117a} \rangle \equiv$

```

     $\langle C\_OneTableSums\_dweights\_dsubset \text{ 120a} \rangle$ 
     $\langle C\_OneTableSums\_iweights\_dsubset \text{ 120b} \rangle$ 
     $\langle C\_OneTableSums\_iweights\_isubset \text{ 120c} \rangle$ 
     $\langle C\_OneTableSums\_dweights\_isubset \text{ 121a} \rangle$ 
     $\langle RC\_OneTableSums \text{ 119a} \rangle$ 
     $\langle R\_OneTableSums \text{ 118a} \rangle$ 
     $\langle C\_TwoTableSums\_dweights\_dsubset \text{ 124b} \rangle$ 
     $\langle C\_TwoTableSums\_iweights\_dsubset \text{ 124c} \rangle$ 
     $\langle C\_TwoTableSums\_iweights\_isubset \text{ 125a} \rangle$ 
     $\langle C\_TwoTableSums\_dweights\_isubset \text{ 125b} \rangle$ 
     $\langle RC\_TwoTableSums \text{ 123b} \rangle$ 
     $\langle R\_TwoTableSums \text{ 122b} \rangle$ 
     $\langle C\_ThreeTableSums\_dweights\_dsubset \text{ 129b} \rangle$ 
     $\langle C\_ThreeTableSums\_iweights\_dsubset \text{ 129c} \rangle$ 
     $\langle C\_ThreeTableSums\_iweights\_isubset \text{ 130a} \rangle$ 
     $\langle C\_ThreeTableSums\_dweights\_isubset \text{ 130b} \rangle$ 
     $\langle RC\_ThreeTableSums \text{ 128b} \rangle$ 
     $\langle R\_ThreeTableSums \text{ 127b} \rangle$ 

```

◇

Fragment referenced in [24a](#).

```

> a0 <- as.vector(xtabs(weights ~ ixf, subset = subset))
> a1 <- ctabx(ix, weights = weights, subset = subset)[-1]
> a2 <- ctabx(ix, weights = as.double(weights), subset = as.double(subset))[-1]
> a3 <- ctabx(ix, weights = weights, subset = as.double(subset))[-1]
> a4 <- ctabx(ix, weights = as.double(weights), subset = subset)[-1]
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))

```

$\langle R\_OneTableSums \text{ Prototype 117b} \rangle \equiv$

```

SEXP R_OneTableSums
(
   $\langle R \text{ x Input 24d} \rangle$ 
   $\langle R \text{ weights Input 26c} \rangle$ ,
   $\langle R \text{ subset Input 27b} \rangle$ 
)

```

◇

Fragment referenced in [23b](#), [118a](#).

Uses: [R\\_OneTableSums 118a](#).



$\langle R\_OneTableSums\ 118a \rangle \equiv$

```

 $\langle R\_OneTableSums\ Prototype\ 117b \rangle$ 
{
    SEXP ans;
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;
    int P;

    N = XLENGTH(x);
    Nsubset = XLENGTH(subset);
    P = NLEVELS(x) + 1;

    PROTECT(ans = allocVector(REALSXP, P));
    RC_OneTableSums(INTEGER(x), N, P, weights, subset,
                    Offset0, Nsubset, REAL(ans));
    UNPROTECT(1);
    return(ans);
}

```

Fragment referenced in 117a.

Defines: R\_OneTableSums 16, 117b, 132b, 164, 165.

Uses: N 24bc, NLEVELS 140a, Nsubset 27c, Offset0 22b, P 25a, RC\_OneTableSums 119a, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc.

$\langle RC\_OneTableSums\ Prototype\ 118b \rangle \equiv$

```

void RC_OneTableSums
(
     $\langle C\ OneTableSums\ Input\ 119b \rangle$ 
     $\langle R\ weights\ Input\ 26c \rangle$ ,
     $\langle R\ subset\ Input\ 27b \rangle$ ,
     $\langle C\ subset\ range\ Input\ 27d \rangle$ ,
     $\langle C\ OneTableSums\ Answer\ 119c \rangle$ 
)

```

Fragment referenced in 119a.

Uses: RC\_OneTableSums 119a.

$\langle RC\_OneTableSums\ 119a \rangle \equiv$

```

 $\langle RC\_OneTableSums\ Prototype\ 118b \rangle$ 
{
  if (typeof(weights) == INTSXP) {
    if (typeof(subset) == INTSXP) {
      C_OneTableSums_iweights_isubset(x, N, P,
                                     INTEGER(weights), XLENGTH(weights) > 0, INTEGER(subset),
                                     offset, Nsubset, P_ans);
    } else {
      C_OneTableSums_iweights_dsubset(x, N, P,
                                     INTEGER(weights), XLENGTH(weights) > 0, REAL(subset),
                                     offset, Nsubset, P_ans);
    }
  } else {
    if (typeof(subset) == INTSXP) {
      C_OneTableSums_dweights_isubset(x, N, P,
                                     REAL(weights), XLENGTH(weights) > 0, INTEGER(subset),
                                     offset, Nsubset, P_ans);
    } else {
      C_OneTableSums_dweights_dsubset(x, N, P,
                                     REAL(weights), XLENGTH(weights) > 0, REAL(subset),
                                     offset, Nsubset, P_ans);
    }
  }
}

```

Fragment referenced in 117a.

Defines: `RC_OneTableSums` 36a, 40, 90, 118ab.

Uses: `C_OneTableSums_dweights_dsubset` 120a, `C_OneTableSums_dweights_isubset` 121a,  
`C_OneTableSums_iweights_dsubset` 120b, `C_OneTableSums_iweights_isubset` 120c, `N` 24bc, `Nsubset` 27c, `offset` 27d,  
`P` 25a, `subset` 27be, 28a, `weights` 26c, `x` 24d, 25bc.

$\langle C\ OneTableSums\ Input\ 119b \rangle \equiv$

```

 $\langle C\ integer\ x\ Input\ 25c \rangle$ 

```

Fragment referenced in 118b, 120abc, 121a.

$\langle C\ OneTableSums\ Answer\ 119c \rangle \equiv$

```

double *P_ans

```

Fragment referenced in 89b, 118b, 120abc, 121a.

$\langle C\_OneTableSums\_dweights\_dsubset\ 120a \rangle \equiv$

```
void C_OneTableSums_dweights_dsubset
(
   $\langle C\_OneTableSums\_Input\ 119b \rangle$ 
   $\langle C\_real\ weights\ Input\ 26e \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_OneTableSums\_Answer\ 119c \rangle$ 
) {
  double *s, *w;
   $\langle OneTableSums\_Body\ 121b \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: C\_OneTableSums\_dweights\_dsubset [119a](#).

$\langle C\_OneTableSums\_iweights\_dsubset\ 120b \rangle \equiv$

```
void C_OneTableSums_iweights_dsubset
(
   $\langle C\_OneTableSums\_Input\ 119b \rangle$ 
   $\langle C\_integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_OneTableSums\_Answer\ 119c \rangle$ 
) {
  double *s;
  int *w;
   $\langle OneTableSums\_Body\ 121b \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: C\_OneTableSums\_iweights\_dsubset [119a](#).

$\langle C\_OneTableSums\_iweights\_isubset\ 120c \rangle \equiv$

```
void C_OneTableSums_iweights_isubset
(
   $\langle C\_OneTableSums\_Input\ 119b \rangle$ 
   $\langle C\_integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\_integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\_OneTableSums\_Answer\ 119c \rangle$ 
) {
  int *s, *w;
   $\langle OneTableSums\_Body\ 121b \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: C\_OneTableSums\_iweights\_isubset [119a](#).

$\langle C\_OneTableSums\_dweights\_isubset\ 121a \rangle \equiv$

```
void C_OneTableSums_dweights_isubset
(
   $\langle C\_OneTableSums\ Input\ 119b \rangle$ 
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\_OneTableSums\ Answer\ 119c \rangle$ 
) {
  int *s;
  double *w;
   $\langle OneTableSums\ Body\ 121b \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: C\_OneTableSums\_dweights\_isubset [119a](#).

$\langle OneTableSums\ Body\ 121b \rangle \equiv$

```
int *xx;

for (int p = 0; p < P; p++) P_ans[p] = 0.0;

xx = x;
 $\langle init\ subset\ loop\ 93b \rangle$ 
 $\langle start\ subset\ loop\ 94a \rangle$ 
{
  xx = xx + diff;
  if (HAS_WEIGHTS) {
    w = w + diff;
    P_ans[xx[0]] += (double) w[0];
  } else {
    P_ans[xx[0]]++;
  }
   $\langle continue\ subset\ loop\ 94b \rangle$ 
}
xx = xx + diff;
if (HAS_WEIGHTS) {
  w = w + diff;
  P_ans[xx[0]] += w[0];
} else {
  P_ans[xx[0]]++;
}
}
◇
```

Fragment referenced in [120abc](#), [121a](#).

Uses: HAS\_WEIGHTS [26de](#), P [25a](#), x [24d](#), [25bc](#).

## TwoTable Sums

```
> a0 <- xtabs(weights ~ ixf + iyf, subset = subset)
> class(a0) <- "matrix"
> dimnames(a0) <- NULL
> attributes(a0)$call <- NULL
> a1 <- ctabs(ix, iy, weights = weights, subset = subset)[-1, -1]
```

```

> a2 <- ctabs(ix, iy, weights = as.double(weights),
+             subset = as.double(subset))[-1, -1]
> a3 <- ctabs(ix, iy, weights = weights, subset = as.double(subset))[-1, -1]
> a4 <- ctabs(ix, iy, weights = as.double(weights), subset = subset)[-1, -1]
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))

```

$\langle R\_TwoTableSums \text{ Prototype } 122a \rangle \equiv$

```

SEXP R_TwoTableSums
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
   $\langle R \text{ } y \text{ Input } 25d \rangle$ 
   $\langle R \text{ } weights \text{ Input } 26c \rangle$ ,
   $\langle R \text{ } subset \text{ Input } 27b \rangle$ 
)
◇

```

Fragment referenced in [23b](#), [122b](#).

Uses: `R_TwoTableSums` [122b](#).

$\langle R\_TwoTableSums \text{ } 122b \rangle \equiv$

```

 $\langle R\_TwoTableSums \text{ Prototype } 122a \rangle$ 
{
  SEXP ans, dim;
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ;
   $\langle C \text{ integer } Nsubset \text{ Input } 27c \rangle$ ;
  int P, Q;

  N = XLENGTH(x);
  Nsubset = XLENGTH(subset);
  P = NLEVELS(x) + 1;
  Q = NLEVELS(y) + 1;

  PROTECT(ans = allocVector(REALSXP, mPQB(P, Q, 1)));
  PROTECT(dim = allocVector(INTSXP, 2));
  INTEGER(dim)[0] = P;
  INTEGER(dim)[1] = Q;
  dimgets(ans, dim);
  RC_TwoTableSums(INTEGER(x), N, P, INTEGER(y), Q,
                  weights, subset, Offset0, Nsubset, REAL(ans));
  UNPROTECT(2);
  return(ans);
}
◇

```

Fragment referenced in [117a](#).

Defines: `R_TwoTableSums` [16](#), [122a](#), [164](#), [165](#).

Uses: `mPQB` [141a](#), `N` [24bc](#), `NLEVELS` [140a](#), `Nsubset` [27c](#), `Offset0` [22b](#), `P` [25a](#), `Q` [25e](#), `RC_TwoTableSums` [123b](#), `subset` [27be](#), [28a](#), `weights` [26c](#), `weights`, [26de](#), `x` [24d](#), [25bc](#), `y` [25d](#), [26ab](#).

$\langle RC\_TwoTableSums \text{ Prototype } 123a \rangle \equiv$

```
void RC_TwoTableSums
(
   $\langle C \text{ TwoTableSums Input } 123c \rangle$ 
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ ,
   $\langle C \text{ subset range Input } 27d \rangle$ ,
   $\langle C \text{ TwoTableSums Answer } 124a \rangle$ 
)
◇
```

Fragment referenced in [123b](#).  
Uses: `RC_TwoTableSums` [123b](#).

$\langle RC\_TwoTableSums \text{ } 123b \rangle \equiv$

```
 $\langle RC\_TwoTableSums \text{ Prototype } 123a \rangle$ 
{
  if (typeof(weights) == INTSXP) {
    if (typeof(subset) == INTSXP) {
      C_TwoTableSums_iweights_isubset(x, N, P, y, Q,
                                     INTEGER(weights), XLENGTH(weights) > 0, INTEGER(subset),
                                     offset, Nsubset, PQ_ans);
    } else {
      C_TwoTableSums_iweights_dsubset(x, N, P, y, Q,
                                     INTEGER(weights), XLENGTH(weights) > 0, REAL(subset),
                                     offset, Nsubset, PQ_ans);
    }
  } else {
    if (typeof(subset) == INTSXP) {
      C_TwoTableSums_dweights_isubset(x, N, P, y, Q,
                                     REAL(weights), XLENGTH(weights) > 0, INTEGER(subset),
                                     offset, Nsubset, PQ_ans);
    } else {
      C_TwoTableSums_dweights_dsubset(x, N, P, y, Q,
                                     REAL(weights), XLENGTH(weights) > 0, REAL(subset),
                                     offset, Nsubset, PQ_ans);
    }
  }
}
◇
```

Fragment referenced in [117a](#).  
Defines: `RC_TwoTableSums` [44](#), [122b](#), [123a](#).  
Uses: `C_TwoTableSums_dweights_dsubset` [124b](#), `C_TwoTableSums_dweights_isubset` [125b](#),  
`C_TwoTableSums_iweights_dsubset` [124c](#), `C_TwoTableSums_iweights_isubset` [125a](#), `N` [24bc](#), `Nsubset` [27c](#), `offset` [27d](#),  
`P` [25a](#), `Q` [25e](#), `subset` [27be](#), [28a](#), `weights` [26c](#), `x` [24d](#), [25bc](#), `y` [25d](#), [26ab](#).

$\langle C \text{ TwoTableSums Input } 123c \rangle \equiv$

```
 $\langle C \text{ integer } x \text{ Input } 25c \rangle$ 
 $\langle C \text{ integer } y \text{ Input } 26b \rangle$ 
◇
```

Fragment referenced in [123a](#), [124bc](#), [125ab](#).

$\langle C\_TwoTableSums\_Answer\ 124a \rangle \equiv$

```
double *PQ_ans
◇
```

Fragment referenced in [123a](#), [124bc](#), [125ab](#).

$\langle C\_TwoTableSums\_dweights\_dsubset\ 124b \rangle \equiv$

```
void C_TwoTableSums_dweights_dsubset
(
   $\langle C\_TwoTableSums\_Input\ 123c \rangle$ 
   $\langle C\_real\ weights\ Input\ 26e \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_TwoTableSums\_Answer\ 124a \rangle$ 
) {
  double *s, *w;
   $\langle TwoTableSums\_Body\ 126 \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: `C_TwoTableSums_dweights_dsubset` [123b](#).

$\langle C\_TwoTableSums\_iweights\_dsubset\ 124c \rangle \equiv$

```
void C_TwoTableSums_iweights_dsubset
(
   $\langle C\_TwoTableSums\_Input\ 123c \rangle$ 
   $\langle C\_integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_TwoTableSums\_Answer\ 124a \rangle$ 
) {
  double *s;
  int *w;
   $\langle TwoTableSums\_Body\ 126 \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: `C_TwoTableSums_iweights_dsubset` [123b](#).

$\langle C\_TwoTableSums\_iweights\_isubset\ 125a \rangle \equiv$

```
void C_TwoTableSums_iweights_isubset
(
     $\langle C\ TwoTableSums\ Input\ 123c \rangle$ 
     $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
     $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
     $\langle C\ TwoTableSums\ Answer\ 124a \rangle$ 
) {
    int *s, *w;
     $\langle TwoTableSums\ Body\ 126 \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: C\_TwoTableSums\_iweights\_isubset [123b](#).

$\langle C\_TwoTableSums\_dweights\_isubset\ 125b \rangle \equiv$

```
void C_TwoTableSums_dweights_isubset
(
     $\langle C\ TwoTableSums\ Input\ 123c \rangle$ 
     $\langle C\ real\ weights\ Input\ 26e \rangle$ 
     $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
     $\langle C\ TwoTableSums\ Answer\ 124a \rangle$ 
) {
    int *s;
    double *w;
     $\langle TwoTableSums\ Body\ 126 \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: C\_TwoTableSums\_dweights\_isubset [123b](#).



$\langle \text{TwoTableSums Body 126} \rangle \equiv$

```

int *xx, *yy;

for (int p = 0; p < Q * P; p++) PQ_ans[p] = 0.0;

yy = y;
xx = x;
 $\langle \text{init subset loop 93b} \rangle$ 
 $\langle \text{start subset loop 94a} \rangle$ 
{
    xx = xx + diff;
    yy = yy + diff;
    if (HAS_WEIGHTS) {
        w = w + diff;
        PQ_ans[yy[0] * P + xx[0]] += (double) w[0];
    } else {
        PQ_ans[yy[0] * P + xx[0]]++;
    }
     $\langle \text{continue subset loop 94b} \rangle$ 
}
xx = xx + diff;
yy = yy + diff;
if (HAS_WEIGHTS) {
    w = w + diff;
    PQ_ans[yy[0] * P + xx[0]] += w[0];
} else {
    PQ_ans[yy[0] * P + xx[0]]++;
}
}

```

Fragment referenced in [124bc](#), [125ab](#).

Uses: HAS\_WEIGHTS [26de](#), P [25a](#), Q [25e](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

### ThreeTable Sums

```

> a0 <- xtabs(weights ~ ixf + iyf + block, subset = subset)
> class(a0) <- "array"
> dimnames(a0) <- NULL
> attributes(a0)$call <- NULL
> a1 <- ctabs(ix, iy, block, weights, subset)[-1, -1,]
> a2 <- ctabs(ix, iy, block, as.double(weights), as.double(subset))[-1, -1,]
> a3 <- ctabs(ix, iy, block, weights, as.double(subset))[-1, -1,]
> a4 <- ctabs(ix, iy, block, as.double(weights), subset)[-1, -1,]
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))

```

$\langle R\_ThreeTableSums \text{ Prototype } 127a \rangle \equiv$

```
SEXP R_ThreeTableSums
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
   $\langle R \text{ } y \text{ Input } 25d \rangle$ 
   $\langle R \text{ } block \text{ Input } 28b \rangle$ ,
   $\langle R \text{ } weights \text{ Input } 26c \rangle$ ,
   $\langle R \text{ } subset \text{ Input } 27b \rangle$ 
)
◇
```

Fragment referenced in [23b](#), [127b](#).

Uses: [R\\_ThreeTableSums](#) [127b](#).

$\langle R\_ThreeTableSums \text{ } 127b \rangle \equiv$

```
 $\langle R\_ThreeTableSums \text{ Prototype } 127a \rangle$ 
{
  SEXP ans, dim;
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ;
   $\langle C \text{ integer } Nsubset \text{ Input } 27c \rangle$ ;
  int P, Q, B;

  N = XLENGTH(x);
  Nsubset = XLENGTH(subset);
  P = NLEVELS(x) + 1;
  Q = NLEVELS(y) + 1;
  B = NLEVELS(block);

  PROTECT(ans = allocVector(REALSXP, mPQB(P, Q, B)));
  PROTECT(dim = allocVector(INTSXP, 3));
  INTEGER(dim)[0] = P;
  INTEGER(dim)[1] = Q;
  INTEGER(dim)[2] = B;
  dimgets(ans, dim);
  RC_ThreeTableSums(INTEGER(x), N, P, INTEGER(y), Q,
                    INTEGER(block), B,
                    weights, subset, Offset0, Nsubset, REAL(ans));
  UNPROTECT(2);
  return(ans);
}
◇
```

Fragment referenced in [117a](#).

Defines: [R\\_ThreeTableSums](#) [16](#), [127a](#), [164](#), [165](#).

Uses: [B](#) [28c](#), [block](#) [28bd](#), [mPQB](#) [141a](#), [N](#) [24bc](#), [NLEVELS](#) [140a](#), [Nsubset](#) [27c](#), [Offset0](#) [22b](#), [P](#) [25a](#), [Q](#) [25e](#), [RC\\_ThreeTableSums](#) [128b](#), [subset](#) [27be](#), [28a](#), [weights](#) [26c](#), [weights](#), [26de](#), [x](#) [24d](#), [25bc](#), [y](#) [25d](#), [26ab](#).

$\langle RC\_ThreeTableSums \text{ Prototype } 128a \rangle \equiv$

```
void RC_ThreeTableSums
(
   $\langle C \text{ ThreeTableSums Input } 128c \rangle$ 
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ ,
   $\langle C \text{ subset range Input } 27d \rangle$ ,
   $\langle C \text{ ThreeTableSums Answer } 129a \rangle$ 
)
◇
```

Fragment referenced in [128b](#).

Uses: [RC\\_ThreeTableSums 128b](#).

$\langle RC\_ThreeTableSums \text{ } 128b \rangle \equiv$

```
 $\langle RC\_ThreeTableSums \text{ Prototype } 128a \rangle$ 
{
  if (typeof(weights) == INTSXP) {
    if (typeof(subset) == INTSXP) {
      C_ThreeTableSums_iweights_isubset(x, N, P, y, Q, block, B,
                                         INTEGER(weights), XLENGTH(weights) > 0, INTEGER(subset),
                                         offset, Nsubset, PQL_ans);
    } else {
      C_ThreeTableSums_iweights_dsubset(x, N, P, y, Q, block, B,
                                         INTEGER(weights), XLENGTH(weights) > 0, REAL(subset),
                                         offset, Nsubset, PQL_ans);
    }
  } else {
    if (typeof(subset) == INTSXP) {
      C_ThreeTableSums_dweights_isubset(x, N, P, y, Q, block, B,
                                         REAL(weights), XLENGTH(weights) > 0, INTEGER(subset),
                                         offset, Nsubset, PQL_ans);
    } else {
      C_ThreeTableSums_dweights_dsubset(x, N, P, y, Q, block, B,
                                         REAL(weights), XLENGTH(weights) > 0, REAL(subset),
                                         offset, Nsubset, PQL_ans);
    }
  }
}
◇
```

Fragment referenced in [117a](#).

Defines: [RC\\_ThreeTableSums 44](#), [127b](#), [128a](#).

Uses: [B 28c](#), [block 28bd](#), [C\\_ThreeTableSums\\_dweights\\_dsubset 129b](#), [C\\_ThreeTableSums\\_dweights\\_isubset 130b](#),  
[C\\_ThreeTableSums\\_iweights\\_dsubset 129c](#), [C\\_ThreeTableSums\\_iweights\\_isubset 130a](#), [N 24bc](#), [Nsubset 27c](#),  
[offset 27d](#), [P 25a](#), [Q 25e](#), [subset 27be](#), [28a](#), [weights 26c](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

$\langle C \text{ ThreeTableSums Input } 128c \rangle \equiv$

```
 $\langle C \text{ integer } x \text{ Input } 25c \rangle$ 
 $\langle C \text{ integer } y \text{ Input } 26b \rangle$ 
 $\langle C \text{ integer block Input } 28d \rangle$ 
◇
```

Fragment referenced in [128a](#), [129bc](#), [130ab](#).

$\langle C\_ThreeTableSums\_Answer\ 129a \rangle \equiv$

```
double *PQL_ans
◇
```

Fragment referenced in [128a](#), [129bc](#), [130ab](#).

$\langle C\_ThreeTableSums\_dweights\_dsubset\ 129b \rangle \equiv$

```
void C_ThreeTableSums_dweights_dsubset
(
   $\langle C\_ThreeTableSums\_Input\ 128c \rangle$ 
   $\langle C\_real\ weights\ Input\ 26e \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_ThreeTableSums\_Answer\ 129a \rangle$ 
) {
  double *s, *w;
   $\langle ThreeTableSums\_Body\ 131a \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: `C_ThreeTableSums_dweights_dsubset` [128b](#).

$\langle C\_ThreeTableSums\_iweights\_dsubset\ 129c \rangle \equiv$

```
void C_ThreeTableSums_iweights_dsubset
(
   $\langle C\_ThreeTableSums\_Input\ 128c \rangle$ 
   $\langle C\_integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_ThreeTableSums\_Answer\ 129a \rangle$ 
) {
  double *s;
  int *w;
   $\langle ThreeTableSums\_Body\ 131a \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: `C_ThreeTableSums_iweights_dsubset` [128b](#).

$\langle C\_ThreeTableSums\_iweights\_isubset \ 130a \rangle \equiv$

```
void C_ThreeTableSums_iweights_isubset
(
   $\langle C \ ThreeTableSums \ Input \ 128c \rangle$ 
   $\langle C \ integer \ weights \ Input \ 26d \rangle$ 
   $\langle C \ integer \ subset \ Input \ 27e \rangle$ ,
   $\langle C \ ThreeTableSums \ Answer \ 129a \rangle$ 
) {
  int *s, *w;
   $\langle ThreeTableSums \ Body \ 131a \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: C\_ThreeTableSums\_iweights\_isubset [128b](#).

$\langle C\_ThreeTableSums\_dweights\_isubset \ 130b \rangle \equiv$

```
void C_ThreeTableSums_dweights_isubset
(
   $\langle C \ ThreeTableSums \ Input \ 128c \rangle$ 
   $\langle C \ real \ weights \ Input \ 26e \rangle$ 
   $\langle C \ integer \ subset \ Input \ 27e \rangle$ ,
   $\langle C \ ThreeTableSums \ Answer \ 129a \rangle$ 
) {
  int *s;
  double *w;
   $\langle ThreeTableSums \ Body \ 131a \rangle$ 
}
◇
```

Fragment referenced in [117a](#).

Defines: C\_ThreeTableSums\_dweights\_isubset [128b](#).

$\langle \text{ThreeTableSums Body 131a} \rangle \equiv$

```

int *xx, *yy, *bb, PQ = mPQB(P, Q, 1);

for (int p = 0; p < PQ * B; p++) PQL_ans[p] = 0.0;

yy = y;
xx = x;
bb = block;
 $\langle \text{init subset loop 93b} \rangle$ 
 $\langle \text{start subset loop 94a} \rangle$ 
{
    xx = xx + diff;
    yy = yy + diff;
    bb = bb + diff;
    if (HAS_WEIGHTS) {
        w = w + diff;
        PQL_ans[(bb[0] - 1) * PQ + yy[0] * P + xx[0]] += (double) w[0];
    } else {
        PQL_ans[(bb[0] - 1) * PQ + yy[0] * P + xx[0]]++;
    }
     $\langle \text{continue subset loop 94b} \rangle$ 
}
xx = xx + diff;
yy = yy + diff;
bb = bb + diff;
if (HAS_WEIGHTS) {
    w = w + diff;
    PQL_ans[(bb[0] - 1) * PQ + yy[0] * P + xx[0]] += w[0];
} else {
    PQL_ans[(bb[0] - 1) * PQ + yy[0] * P + xx[0]]++;
}
}

```

Fragment referenced in [129bc](#), [130ab](#).

Uses: B [28c](#), block [28bd](#), HAS\_WEIGHTS [26de](#), mPQB [141a](#), P [25a](#), Q [25e](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

## 3.10 Utilities

### 3.10.1 Blocks

```

> sb <- sample(block)
> ns1 <- do.call("c", tapply(subset, sb[subset], function(i) i))
> ns2 <- .Call(libcoin:::R_order_subset_wrt_block, y, integer(0), subset, sb)
> stopifnot(isequal(ns1, ns2))

```

$\langle \text{Utils 131b} \rangle \equiv$

```

 $\langle \text{C_setup_subset 134a} \rangle$ 
 $\langle \text{C_setup_subset_block 134b} \rangle$ 
 $\langle \text{C_order_subset_wrt_block 135a} \rangle$ 
 $\langle \text{RC_order_subset_wrt_block 133b} \rangle$ 
 $\langle \text{R_order_subset_wrt_block 132b} \rangle$ 

```

Fragment referenced in [24a](#).

$\langle R\_order\_subset\_wrt\_block \text{ Prototype } 132a \rangle \equiv$

```
SEXP R_order_subset_wrt_block
(
   $\langle R \text{ y Input } 25d \rangle$ 
   $\langle R \text{ weights Input } 26c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ ,
   $\langle R \text{ block Input } 28b \rangle$ 
)
◇
```

Fragment referenced in [23b](#), [132b](#).

Uses: [R\\_order\\_subset\\_wrt\\_block 132b](#).

$\langle R\_order\_subset\_wrt\_block \text{ } 132b \rangle \equiv$

```
 $\langle R\_order\_subset\_wrt\_block \text{ Prototype } 132a \rangle$ 
{
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ;
  SEXP blockTable, ans;

  N = XLENGTH(y) / NCOL(y);

  if (XLENGTH(weights) > 0)
    error("cannot deal with weights here");

  if (NLEVELS(block) > 1) {
    PROTECT(blockTable = R_OneTableSums(block, weights, subset));
  } else {
    PROTECT(blockTable = allocVector(REALSXP, 2));
    REAL(blockTable)[0] = 0.0;
    REAL(blockTable)[1] = RC_Sums(N, weights, subset, Offset0, XLENGTH(subset));
  }

  PROTECT(ans = RC_order_subset_wrt_block(N, subset, block, blockTable));

  UNPROTECT(2);
  return(ans);
}
◇
```

Fragment referenced in [131b](#).

Defines: [R\\_order\\_subset\\_wrt\\_block 132a](#), [164](#), [165](#).

Uses: [block 28bd](#), [blockTable 28e](#), [N 24bc](#), [NCOL 139c](#), [NLEVELS 140a](#), [Offset0 22b](#), [RC\\_order\\_subset\\_wrt\\_block 133b](#), [RC\\_Sums 96a](#), [R\\_OneTableSums 118a](#), [subset 27be](#), [28a](#), [weights 26c](#), [weights, 26de](#), [y 25d](#), [26ab](#).

$\langle RC\_order\_subset\_wrt\_block \text{ Prototype } 133a \rangle \equiv$

```
SEXP RC_order_subset_wrt_block
(
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ,
   $\langle R \text{ subset Input } 27b \rangle$ ,
   $\langle R \text{ block Input } 28b \rangle$ ,
   $\langle R \text{ blockTable Input } 28e \rangle$ 
)
◇
```

Fragment referenced in [133b](#).

Uses: `RC_order_subset_wrt_block` [133b](#).

$\langle RC\_order\_subset\_wrt\_block \text{ } 133b \rangle \equiv$

```
 $\langle RC\_order\_subset\_wrt\_block \text{ Prototype } 133a \rangle$ 
{
  SEXP ans;
  int NOBLOCK = (XLENGTH(block) == 0 || XLENGTH(blockTable) == 2);

  if (XLENGTH(subset) > 0) {
    if (NOBLOCK) {
      return(subset);
    } else {
      PROTECT(ans = allocVector(TYPEOF(subset), XLENGTH(subset)));
      C_order_subset_wrt_block(subset, block, blockTable, ans);
      UNPROTECT(1);
      return(ans);
    }
  } else {
    PROTECT(ans = allocVector(TYPEOF(subset), N));
    if (NOBLOCK) {
      C_setup_subset(N, ans);
    } else {
      C_setup_subset_block(N, block, blockTable, ans);
    }
    UNPROTECT(1);
    return(ans);
  }
}
◇
```

Fragment referenced in [131b](#).

Defines: `RC_order_subset_wrt_block` [36a](#), [40](#), [132b](#), [133a](#).

Uses: `block` [28bd](#), `blockTable` [28e](#), `C_order_subset_wrt_block` [135a](#), `C_setup_subset` [134a](#), `C_setup_subset_block` [134b](#), `N` [24bc](#), `subset` [27be](#), [28a](#).



$\langle C\_setup\_subset\ 134a \rangle \equiv$

```
void C_setup_subset
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
  SEXP ans
) {
  for (R_xlen_t i = 0; i < N; i++) {
    /* ans is R style index in 1:N */
    if (TYPEOF(ans) == INTSXP) {
      INTEGER(ans)[i] = i + 1;
    } else {
      REAL(ans)[i] = (double) i + 1;
    }
  }
}
◇
```

Fragment referenced in [131b](#).

Defines: `C_setup_subset` [133b](#), [136a](#).

Uses: `N` [24bc](#).

$\langle C\_setup\_subset\_block\ 134b \rangle \equiv$

```
void C_setup_subset_block
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle R\ block\ Input\ 28b \rangle$ ,
   $\langle R\ blockTable\ Input\ 28e \rangle$ ,
  SEXP ans
) {
  double *cumtable;
  int Nlevels = LENGTH(blockTable);

  cumtable = Calloc(Nlevels, double);
  for (int k = 0; k < Nlevels; k++) cumtable[k] = 0.0;

  /* table[0] are missings, ie block == 0 ! */
  for (int k = 1; k < Nlevels; k++)
    cumtable[k] = cumtable[k - 1] + REAL(blockTable)[k - 1];

  for (R_xlen_t i = 0; i < N; i++) {
    /* ans is R style index in 1:N */
    if (TYPEOF(ans) == INTSXP) {
      INTEGER(ans)[(int) cumtable[INTEGER(block)[i]]++] = i + 1;
    } else {
      REAL(ans)[(R_xlen_t) cumtable[INTEGER(block)[i]]++] = (double) i + 1;
    }
  }

  Free(cumtable);
}
◇
```

Fragment referenced in [131b](#).

Defines: `C_setup_subset_block` [133b](#).

Uses: `block` [28bd](#), `blockTable` [28e](#), `N` [24bc](#).

$\langle C\_order\_subset\_wrt\_block\ 135a \rangle \equiv$

```

void C_order_subset_wrt_block
(
     $\langle R\ subset\ Input\ 27b \rangle$ ,
     $\langle R\ block\ Input\ 28b \rangle$ ,
     $\langle R\ blockTable\ Input\ 28e \rangle$ ,
    SEXP ans
) {
    double *cumtable;
    int Nlevels = LENGTH(blockTable);

    cumtable = Calloc(Nlevels, double);
    for (int k = 0; k < Nlevels; k++) cumtable[k] = 0.0;

    /* table[0] are missings, ie block == 0 ! */
    for (int k = 1; k < Nlevels; k++)
        cumtable[k] = cumtable[k - 1] + REAL(blockTable)[k - 1];

    /* subset is R style index in 1:N */
    if (TYPEOF(subset) == INTSXP) {
        for (R_xlen_t i = 0; i < XLENGTH(subset); i++)
            INTEGER(ans)[(int) cumtable[INTEGER(block)[INTEGER(subset)[i] -
1]]++] = INTEGER(subset)[i];
    } else {
        for (R_xlen_t i = 0; i < XLENGTH(subset); i++)
            REAL(ans)[(R_xlen_t) cumtable[INTEGER(block)[(R_xlen_t) REAL(subset)[i] -
1]]++] = REAL(subset)[i];
    }

    Free(cumtable);
}

```

Fragment referenced in [131b](#).

Defines: `C_order_subset_wrt_block` [133b](#).

Uses: `block` [28bd](#), `blockTable` [28e](#), `N` [24bc](#), `subset` [27be](#), [28a](#).

$\langle RC\_setup\_subset\ Prototype\ 135b \rangle \equiv$

```

SEXP RC_setup_subset
(
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
     $\langle R\ weights\ Input\ 26c \rangle$ ,
     $\langle R\ subset\ Input\ 27b \rangle$ 
)

```

Fragment referenced in [136a](#).

Uses: `RC_setup_subset` [136a](#).

Because this will only be used when really needed (in Permutations) we can be a little bit more generous with memory here. The return value is always `REALSXP`.

$\langle RC\_setup\_subset\ 136a \rangle \equiv$

```

 $\langle RC\_setup\_subset\ Prototype\ 135b \rangle$ 
{
    SEXP ans, mysubset;
    R_xlen_t sumweights;

    if (XLENGTH(subset) == 0) {
        PROTECT(mysubset = allocVector(REALSXP, N));
        C_setup_subset(N, mysubset);
    } else {
        PROTECT(mysubset = coerceVector(subset, REALSXP));
    }

    if (XLENGTH(weights) == 0) {
        UNPROTECT(1);
        return(mysubset);
    }

    sumweights = (R_xlen_t) RC_Sums(N, weights, mysubset, Offset0, XLENGTH(subset));
    PROTECT(ans = allocVector(REALSXP, sumweights));

    R_xlen_t itmp = 0;
    for (R_xlen_t i = 0; i < XLENGTH(mysubset); i++) {
        if (TYPEOF(weights) == REALSXP) {
            for (R_xlen_t j = 0; j < REAL(weights)[(R_xlen_t) REAL(mysubset)[i] - 1]; j++)
                REAL(ans)[itmp++] = REAL(mysubset)[i];
        } else {
            for (R_xlen_t j = 0; j < INTEGER(weights)[(R_xlen_t) REAL(mysubset)[i] - 1]; j++)
                REAL(ans)[itmp++] = REAL(mysubset)[i];
        }
    }
    UNPROTECT(2);
    return(ans);
}

```

Fragment referenced in [136b](#).

Defines: `RC_setup_subset` [40](#), [135b](#).

Uses: `C_setup_subset` [134a](#), `N` [24bc](#), `Offset0` [22b](#), `RC_Sums` [96a](#), `subset` [27be](#), [28a](#), `sumweights` [27a](#), `weights` [26c](#), `weights`, [26de](#).

### 3.10.2 Permutation Helpers

$\langle Permutations\ 136b \rangle \equiv$

```

 $\langle RC\_setup\_subset\ 136a \rangle$ 
 $\langle C\_Permute\ 137a \rangle$ 
 $\langle C\_doPermute\ 137b \rangle$ 
 $\langle C\_PermuteBlock\ 138a \rangle$ 
 $\langle C\_doPermuteBlock\ 138b \rangle$ 

```

Fragment referenced in [24a](#).

$\langle C\_Permute\ 137a \rangle \equiv$

```

void C_Permute
(
    double *subset,
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ,
    double *ans
) {
    R_xlen_t n = Nsubset, j;

    for (R_xlen_t i = 0; i < Nsubset; i++) {
        j = n * unif_rand();
        ans[i] = subset[j];
        subset[j] = subset[--n];
    }
}

```

Fragment referenced in [136b](#).

Defines: `C_Permute` [137b](#), [138a](#).

Uses: `Nsubset` [27c](#), `subset` [27be](#), [28a](#).

$\langle C\_doPermute\ 137b \rangle \equiv$

```

void C_doPermute
(
    double *subset,
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ,
    double *Nsubset_tmp,
    double *perm
) {
    Malloc(Nsubset_tmp, subset, Nsubset);
    C_Permute(Nsubset_tmp, Nsubset, perm);
}

```

Fragment referenced in [136b](#).

Defines: `C_doPermute` [40](#).

Uses: `C_Permute` [137a](#), `Nsubset` [27c](#), `subset` [27be](#), [28a](#).

$\langle C\_PermuteBlock\ 138a \rangle \equiv$

```

void C_PermuteBlock
(
    double *subset,
    double *table,
    int Nlevels,
    double *ans
) {
    double *px, *pans;

    px = subset;
    pans = ans;

    for (R_xlen_t j = 0; j < Nlevels; j++) {
        if (table[j] > 0) {
            C_Permute(px, (R_xlen_t) table[j], pans);
            px += (R_xlen_t) table[j];
            pans += (R_xlen_t) table[j];
        }
    }
}
◇

```

Fragment referenced in [136b](#).

Defines: `C_PermuteBlock` [138b](#).

Uses: `C_Permute` [137a](#), `subset` [27be](#), [28a](#).

$\langle C\_doPermuteBlock\ 138b \rangle \equiv$

```

void C_doPermuteBlock
(
    double *subset,
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ,
    double *table,
    int Nlevels,
    double *Nsubset_tmp,
    double *perm
) {
    Memcpy(Nsubset_tmp, subset, Nsubset);
    C_PermuteBlock(Nsubset_tmp, table, Nlevels, perm);
}
◇

```

Fragment referenced in [136b](#).

Defines: `C_doPermuteBlock` [40](#).

Uses: `C_PermuteBlock` [138a](#), `Nsubset` [27c](#), `subset` [27be](#), [28a](#).

### 3.10.3 Other Utils

⟨ *MoreUtils* 139a ⟩ ≡

```
⟨ NROW 139b ⟩
⟨ NCOL 139c ⟩
⟨ NLEVELS 140a ⟩
⟨ C_kronecker 143 ⟩
⟨ R_kronecker 142 ⟩
⟨ C_kronecker_sym 144 ⟩
⟨ C_KronSums_sym 145a ⟩
⟨ C_MPinv_sym 147 ⟩
⟨ R_MPinv_sym 146b ⟩
⟨ R_unpack_sym 149 ⟩
⟨ R_pack_sym 150c ⟩
◇
```

Fragment referenced in 24a.

⟨ *NROW* 139b ⟩ ≡

```
int NROW
(
    SEXP x
) {
    SEXP a;
    a = getAttrib(x, R_DimSymbol);
    if (a == R_NilValue) return(XLENGTH(x));
    if (TYPEOF(a) == REALSXP)
        return(REAL(a)[0]);
    return(INTEGER(a)[0]);
}
◇
```

Fragment referenced in 139a.

Defines: *NROW* 6, 8, 9ab, 14, 35a, 40, 46c, 47, 64c, 140a, 142, 150c.

Uses: *x* 24d, 25bc.

⟨ *NCOL* 139c ⟩ ≡

```
int NCOL
(
    SEXP x
) {
    SEXP a;
    a = getAttrib(x, R_DimSymbol);
    if (a == R_NilValue) return(1);
    if (TYPEOF(a) == REALSXP)
        return(REAL(a)[1]);
    return(INTEGER(a)[1]);
}
◇
```

Fragment referenced in 139a.

Defines: *NCOL* 12, 33, 45a, 64c, 85b, 87a, 100a, 109b, 113b, 132b, 142.

Uses: *x* 24d, 25bc.

$\langle NLEVELS\ 140a \rangle \equiv$

```

int NLEVELS
(
    SEXP x
) {
    SEXP a;
    int maxlev = 0;

    a = getAttrib(x, R_LevelsSymbol);
    if (a == R_NilValue) {
        if (TYPEOF(x) != INTSXP)
            error("cannot determine number of levels");
        for (R_xlen_t i = 0; i < XLENGTH(x); i++) {
            if (INTEGER(x)[i] > maxlev)
                maxlev = INTEGER(x)[i];
        }
        return(maxlev);
    }
    return(NROW(a));
}

```

Fragment referenced in [139a](#).

Defines: [NLEVELS 33, 45a, 118a, 122b, 127b, 132b](#).

Uses: [NROW 139b](#), [x 24d, 25bc](#).

Check for integer overflow when computing  $P(P+1)/2$  and  $PQ$ .

$\langle PP12\ 140b \rangle \equiv$

```

int PP12
(
    int P
) {
    double dP = (double) P;
    double ans;

    ans = dP * (dP + 1) / 2;

    if (ans > INT_MAX)
        error("cannot allocate memory: number of levels too large");

    return((int) ans);
}

```

Fragment referenced in [151a](#).

Defines: [PP12 36a, 47, 49, 55, 83, 93a, 159, 160a](#).

Uses: [P 25a](#).

$\langle mPQB\ 141a \rangle \equiv$

```

int mPQB
(
    int P,
    int Q,
    int B
) {
    double ans = P * Q * B;

    if (ans > INT_MAX)
        error("cannot allocate memory: number of levels too large");

    return((int) ans);
}
◇

```

Fragment referenced in [151a](#).

Defines: [mPQB 38b](#), [40](#), [48](#), [51](#), [56a](#), [74](#), [76a](#), [80b](#), [82b](#), [83](#), [84](#), [108](#), [112b](#), [122b](#), [127b](#), [131a](#), [159](#).

Uses: [B 28c](#), [P 25a](#), [Q 25e](#).

```

> A <- matrix(runif(12), ncol = 3)
> B <- matrix(runif(10), ncol = 2)
> K1 <- kronecker(A, B)
> K2 <- .Call(libcoin::R_kronecker, A, B)
> stopifnot(isequal(K1, K2))

```

"libcoinAPI.h" 141b≡

```

extern SEXP libcoin_R_kronecker(
    SEXP A, SEXP B
) {
    static SEXP(*fun)(SEXP, SEXP) = NULL;
    if (fun == NULL)
        fun = (SEXP(*) (SEXP, SEXP))
            R_GetCCallable("libcoin", "R_kronecker");
    return fun(A, B);
}
◇

```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).

Uses: [B 28c](#).

$\langle R\_kronecker\ Prototype\ 141c \rangle \equiv$

```

SEXP R_kronecker
(
    SEXP A,
    SEXP B
)
◇

```

Fragment referenced in [23b](#), [142](#).

Uses: [B 28c](#).



$\langle R\_kronecker\ 142 \rangle \equiv$

```
 $\langle R\_kronecker\ Prototype\ 141c \rangle$ 
{
  int m, n, r, s;
  SEXP ans;

  if (!isReal(A) || !isReal(B))
    error("R_kronecker: A and / or B are not of type REALSXP");

  m = NROW(A);
  n = NCOL(A);
  r = NROW(B);
  s = NCOL(B);

  PROTECT(ans = allocMatrix(REALSXP, m * n, r * s));
  C_kronecker(REAL(A), m, n, REAL(B), r, s, 1, REAL(ans));
  UNPROTECT(1);
  return(ans);
}
◇
```

Fragment referenced in [139a](#).

Uses: B [28c](#), C\_kronecker [143](#), NCOL [139c](#), NROW [139b](#).

$\langle C\_kronecker\ 143 \rangle \equiv$

```
void C_kronecker
(
    const double *A,
    const int m,
    const int n,
    const double *B,
    const int r,
    const int s,
    const int overwrite,
    double *ans
) {
    int mr, js, ir;
    double y;

    if (overwrite) {
        for (int i = 0; i < m * r * n * s; i++) ans[i] = 0.0;
    }

    mr = m * r;
    for (int i = 0; i < m; i++) {
        ir = i * r;
        for (int j = 0; j < n; j++) {
            js = j * s;
            y = A[j*m + i];
            for (int k = 0; k < r; k++) {
                for (int l = 0; l < s; l++)
                    ans[(js + l) * mr + ir + k] += y * B[l * r + k];
            }
        }
    }
}
```

◇

Fragment referenced in [139a](#).  
Defines: `C_kronecker` [84](#), [142](#).  
Uses: `B` [28c](#), `y` [25d](#), [26ab](#).

$\langle C\_kronecker\_sym\ 144 \rangle \equiv$

```

void C_kronecker_sym
(
    const double *A,
    const int m,
    const double *B,
    const int r,
    const int overwrite,
    double *ans
) {
    int mr, js, ir, s;
    double y;

    mr = m * r;
    s = r;

    if (overwrite) {
        for (int i = 0; i < mr * (mr + 1) / 2; i++) ans[i] = 0.0;
    }

    for (int i = 0; i < m; i++) {
        ir = i * r;
        for (int j = 0; j <= i; j++) {
            js = j * s;
            y = A[S(i, j, m)];
            for (int k = 0; k < r; k++) {
                for (int l = 0; l < (j < i ? s : k + 1); l++) {
                    ans[S(ir + k, js + l, mr)] += y * B[S(k, l, r)];
                }
            }
        }
    }
}

```

◇

Fragment referenced in [139a](#).

Defines: `C_kronecker_sym` [83](#).

Uses: [B 28c](#), [S 22a](#), [y 25d](#), [26ab](#).

$\langle C\_KronSums\_sym \ 145a \rangle \equiv$

```

/* sum_i (t(x[i,]) %*% x[i,]) */
void C_KronSums_sym_
(
   $\langle C \text{ real } x \text{ Input } 25b \rangle$ 
  double *PP_sym_ans
) {
  int pN, qN, SpqP;

  for (int q = 0; q < P; q++) {
    qN = q * N;
    for (int p = 0; p <= q; p++) {
      PP_sym_ans[S(p, q, P)] = 0.0;
      pN = p * N;
      SpqP = S(p, q, P);
      for (int i = 0; i < N; i++)
        PP_sym_ans[SpqP] += x[qN + i] * x[pN + i];
    }
  }
}

```

Fragment referenced in [139a](#).

Defines: C\_KronSums\_sym Never used.

Uses: N [24bc](#), P [25a](#), S [22a](#), x [24d](#), [25bc](#).

```

> covar <- vcov(ls1)
> covar_sym <- ls1$Covariance
> n <- (sqrt(1 + 8 * length(covar_sym)) - 1) / 2
> tol <- sqrt(.Machine$double.eps)
> MP1 <- MPinverse(covar, tol)
> MP2 <- .Call(libcoin:::R_MPinv_sym, covar_sym, as.integer(n), tol)
> lt <- lower.tri(covar, diag = TRUE)
> stopifnot(isequal(MP1$MPinv[lt], MP2$MPinv) &&
+           isequal(MP1$rank, MP2$rank))

```

"libcoinAPI.h" 145b $\equiv$

```

extern SEXP libcoin_R_MPinv_sym(
  SEXP x, SEXP n, SEXP tol
) {
  static SEXP(*fun)(SEXP, SEXP, SEXP) = NULL;
  if (fun == NULL)
    fun = (SEXP(*) (SEXP, SEXP, SEXP))
      R_GetCCallable("libcoin", "R_MPinv_sym");
  return fun(x, n, tol);
}

```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).

Uses: R\_MPinv\_sym [146b](#), x [24d](#), [25bc](#).

$\langle R\_MPinv\_sym \text{ Prototype 146a} \rangle \equiv$

```
SEXP R_MPinv_sym
(
    SEXP x,
    SEXP n,
    SEXP tol
)
◇
```

Fragment referenced in [23b](#), [146b](#).  
 Uses: [R\\_MPinv\\_sym 146b](#), [x 24d](#), [25bc](#).

$\langle R\_MPinv\_sym \text{ 146b} \rangle \equiv$

```
 $\langle R\_MPinv\_sym \text{ Prototype 146a} \rangle$ 
{
    int m;
    SEXP ans, names, MPinv, rank;

    m = INTEGER(n)[0];
    if (m == 0)
        m = (int) (sqrt(0.25 + 2 * LENGTH(x)) - 0.5);

    PROTECT(ans = allocVector(VECSXP, 2));
    PROTECT(names = allocVector(STRSXP, 2));
    SET_VECTOR_ELT(ans, 0, MPinv = allocVector(REALSXP, LENGTH(x)));
    SET_STRING_ELT(names, 0, mkChar("MPinv"));
    SET_VECTOR_ELT(ans, 1, rank = allocVector(INTSXP, 1));
    SET_STRING_ELT(names, 1, mkChar("rank"));
    namesgets(ans, names);

    C_MPinv_sym(REAL(x), m, REAL(tol)[0], REAL(MPinv), INTEGER(rank));

    UNPROTECT(2);
    return(ans);
}
◇
```

Fragment referenced in [139a](#).  
 Defines: [R\\_MPinv\\_sym 145b](#), [146a](#), [164](#), [165](#).  
 Uses: [x 24d](#), [25bc](#).

$\langle C\_MPinv\_sym\ 147 \rangle \equiv$

```

void C_MPinv_sym
(
    const double *x,
    const int n,
    const double tol,
    double *dMP,
    int *rank
) {
    double *val, *vec, dtol, *rx, *work, valinv;
    int valzero = 0, info = 0, kn;

    if (n == 1) {
        if (x[0] > tol) {
            dMP[0] = 1 / x[0];
            rank[0] = 1;
        } else {
            dMP[0] = 0;
            rank[0] = 0;
        }
    } else {
        rx = Calloc(n * (n + 1) / 2, double);
        Memcpy(rx, x, n * (n + 1) / 2);
        work = Calloc(3 * n, double);
        val = Calloc(n, double);
        vec = Calloc(n * n, double);

        F77_CALL(dspev)("V", "L", &n, rx, val, vec, &n, work,
                        &info FCONE FCONE);

        dtol = val[n - 1] * tol;

        for (int k = 0; k < n; k++)
            valzero += (val[k] < dtol);
        rank[0] = n - valzero;

        for (int k = 0; k < n * (n + 1) / 2; k++) dMP[k] = 0.0;

        for (int k = valzero; k < n; k++) {
            valinv = 1 / val[k];
            kn = k * n;
            for (int i = 0; i < n; i++) {
                for (int j = 0; j <= i; j++) {
                    /* MP is symmetric */
                    dMP[S(i, j, n)] += valinv * vec[kn + i] * vec[kn + j];
                }
            }
        }
        Free(rx); Free(work); Free(val); Free(vec);
    }
}

```

Fragment referenced in [139a](#).

Uses: [S 22a](#), [x 24d](#), [25bc](#).

```
> m <- matrix(c(3, 2, 1,
```

```

+           2, 4, 2,
+           1, 2, 5),
+           ncol = 3)
> s <- m[lower.tri(m, diag = TRUE)]
> u1 <- .Call(libcoin:::R_unpack_sym, s, NULL, 0L)
> u2 <- .Call(libcoin:::R_unpack_sym, s, NULL, 1L)
> stopifnot(isequal(m, u1) && isequal(diag(m), u2))

```

"libcoinAPI.h" 148a≡

```

extern SEXP libcoin_R_unpack_sym(
    SEXP x, SEXP names, SEXP diagonly
) {
    static SEXP(*fun)(SEXP, SEXP, SEXP) = NULL;
    if (fun == NULL)
        fun = (SEXP*)(SEXP, SEXP, SEXP))
            R_GetCCallable("libcoin", "R_unpack_sym");
    return fun(x, names, diagonly);
}
◇

```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).  
 Uses: [R\\_unpack\\_sym 149](#), [x 24d](#), [25bc](#).

⟨ *R\_unpack\_sym Prototype 148b* ⟩ ≡

```

SEXP R_unpack_sym
(
    SEXP x,
    SEXP names,
    SEXP diagonly
)
◇

```

Fragment referenced in [23b](#), [149](#).  
 Uses: [R\\_unpack\\_sym 149](#), [x 24d](#), [25bc](#).

$\langle R\_unpack\_sym\ 149 \rangle \equiv$

```

 $\langle R\_unpack\_sym\ Prototype\ 148b \rangle$ 
{
  R_xlen_t n, k = 0;
  SEXP ans, dimnames;
  double *dx, *dans;

  // m = n * (n + 1)/2 <=> n^2 + n - 2 * m = 0
  n = sqrt(0.25 + 2 * XLENGTH(x)) - 0.5;

  dx = REAL(x);
  if (INTEGER(diagonly)[0]) {
    PROTECT(ans = allocVector(REALSXP, n));
    if (names != R_NilValue) {
      namesgets(ans, names);
    }
    dans = REAL(ans);
    for (R_xlen_t i = 0; i < n; i++) {
      dans[i] = dx[k];
      k += n - i;
    }
  } else {
    PROTECT(ans = allocMatrix(REALSXP, n, n));
    if (names != R_NilValue) {
      PROTECT(dimnames = allocVector(VECSXP, 2));
      SET_VECTOR_ELT(dimnames, 0, names);
      SET_VECTOR_ELT(dimnames, 1, names);
      dimnamesgets(ans, dimnames);
      UNPROTECT(1);
    }
    dans = REAL(ans);
    for (R_xlen_t i = 0; i < n; i++) {
      dans[i * n + i] = dx[k];      // diagonal
      k++;
      for (R_xlen_t j = i + 1; j < n; j++) {
        dans[i * n + j] = dx[k]; // lower triangular
        dans[j * n + i] = dx[k]; // upper triangular
        k++;
      }
    }
  }

  UNPROTECT(1);
  return ans;
}

```

Fragment referenced in [139a](#).

Defines: `R_unpack_sym` [10](#), [148ab](#), [164](#), [165](#).

Uses: `x` [24d](#), [25bc](#).

```

> m <- matrix(c(4, 3, 2, 1,
+              3, 5, 4, 2,
+              2, 4, 6, 5,
+              1, 2, 5, 7),
+            ncol = 4)
> s <- m[lower.tri(m, diag = TRUE)]

```



```
> p <- .Call(libcoin:::R_pack_sym, m)
> stopifnot(isequal(s, p))
```

"libcoinAPI.h" 150a≡

```
extern SEXP libcoin_R_pack_sym(
  SEXP x
) {
  static SEXP(*fun)(SEXP) = NULL;
  if (fun == NULL)
    fun = (SEXP(*) (SEXP))
      R_GetCCallable("libcoin", "R_pack_sym");
  return fun(x);
}
◇
```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [54a](#), [64a](#), [141b](#), [145b](#), [148a](#), [150a](#).

Uses: [R\\_pack\\_sym 150c](#), [x 24d](#), [25bc](#).

⟨ *R\_pack\_sym* Prototype 150b ⟩ ≡

```
SEXP R_pack_sym
(
  SEXP x
)
◇
```

Fragment referenced in [23b](#), [150c](#).

Uses: [R\\_pack\\_sym 150c](#), [x 24d](#), [25bc](#).

⟨ *R\_pack\_sym* 150c ⟩ ≡

```
⟨ R_pack_sym Prototype 150b ⟩
{
  R_xlen_t n, k = 0;
  SEXP ans;
  double *dx, *dans;

  n = NROW(x);
  dx = REAL(x);
  PROTECT(ans = allocVector(REALSXP, n * (n + 1) / 2));
  dans = REAL(ans);

  for (R_xlen_t i = 0; i < n; i++) {
    for (R_xlen_t j = i; j < n; j++) {
      dans[k] = dx[i * n + j];
      k++;
    }
  }

  UNPROTECT(1);
  return ans;
}
◇
```

Fragment referenced in [139a](#).

Defines: [R\\_pack\\_sym 150ab](#), [164](#), [165](#).

Uses: [NROW 139b](#), [x 24d](#), [25bc](#).

### 3.11 Memory

$\langle \text{Memory } 151a \rangle \equiv$

```

     $\langle C\_get\_P \text{ } 151c \rangle$ 
     $\langle C\_get\_Q \text{ } 152a \rangle$ 
     $\langle PP12 \text{ } 140b \rangle$ 
     $\langle mPQB \text{ } 141a \rangle$ 
     $\langle C\_get\_varonly \text{ } 152b \rangle$ 
     $\langle C\_get\_Xfactor \text{ } 152c \rangle$ 
     $\langle C\_get\_LinearStatistic \text{ } 152d \rangle$ 
     $\langle C\_get\_Expectation \text{ } 153a \rangle$ 
     $\langle C\_get\_Variance \text{ } 153b \rangle$ 
     $\langle C\_get\_Covariance \text{ } 154a \rangle$ 
     $\langle C\_get\_ExpectationX \text{ } 154b \rangle$ 
     $\langle C\_get\_ExpectationInfluence \text{ } 154c \rangle$ 
     $\langle C\_get\_CovarianceInfluence \text{ } 155a \rangle$ 
     $\langle C\_get\_VarianceInfluence \text{ } 155b \rangle$ 
     $\langle C\_get\_TableBlock \text{ } 155c \rangle$ 
     $\langle C\_get\_Sumweights \text{ } 156a \rangle$ 
     $\langle C\_get\_Table \text{ } 156b \rangle$ 
     $\langle C\_get\_dimTable \text{ } 156c \rangle$ 
     $\langle C\_get\_B \text{ } 157a \rangle$ 
     $\langle C\_get\_nresample \text{ } 157b \rangle$ 
     $\langle C\_get\_PermutedLinearStatistic \text{ } 157c \rangle$ 
     $\langle C\_get\_tol \text{ } 157d \rangle$ 
     $\langle RC\_init\_LECV\_1d \text{ } 160b \rangle$ 
     $\langle RC\_init\_LECV\_2d \text{ } 161 \rangle$ 
     $\diamond$ 

```

Fragment referenced in [24a](#).

$\langle R \text{ } LECV \text{ } Input \text{ } 151b \rangle \equiv$

```

    SEXP LECV
     $\diamond$ 

```

Fragment referenced in [54b](#), [56b](#), [151c](#), [152abcd](#), [153ab](#), [154abc](#), [155abc](#), [156abc](#), [157abcd](#).

Defines: [LECV 41bc](#), [42a](#), [55](#), [56a](#), [57](#), [58](#), [59](#), [72b](#), [74](#), [151c](#), [152abcd](#), [153ab](#), [154abc](#), [155abc](#), [156abc](#), [157abcd](#).

$\langle C\_get\_P \text{ } 151c \rangle \equiv$

```

    int C_get_P
    (
         $\langle R \text{ } LECV \text{ } Input \text{ } 151b \rangle$ 
    ) {
        return(VECTOR_ELT(LECV, dim_SLOT)[0]);
    }
     $\diamond$ 

```

Fragment referenced in [151a](#).

Defines: [C\\_get\\_P 35a](#), [42a](#), [49](#), [56a](#), [59](#), [74](#), [153b](#), [154a](#), [157b](#).

Uses: [dim\\_SLOT 22b](#), [LECV 151b](#).

$\langle C\_get\_Q\ 152a \rangle \equiv$

```
int C_get_Q
(
     $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
    return(VECTOR_ELT(LECV, dim_SLOT)[1]);
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_Q 35a, 42a, 49, 56a, 74, 153b, 154a, 157b.

Uses: dim\_SLOT 22b, LECV 151b.

$\langle C\_get\_varonly\ 152b \rangle \equiv$

```
int C_get_varonly
(
     $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
    return(VECTOR_ELT(LECV, varonly_SLOT)[0]);
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_varonly 34, 36a, 38b, 42a, 47, 48, 49, 56a, 57, 74, 154a.

Uses: LECV 151b, varonly\_SLOT 22b.

$\langle C\_get\_Xfactor\ 152c \rangle \equiv$

```
int C_get_Xfactor
(
     $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
    return(VECTOR_ELT(LECV, Xfactor_SLOT)[0]);
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_Xfactor 49.

Uses: LECV 151b, Xfactor\_SLOT 22b.

$\langle C\_get\_LinearStatistic\ 152d \rangle \equiv$

```
double* C_get_LinearStatistic
(
     $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
    return(REAL(VECTOR_ELT(LECV, LinearStatistic_SLOT)));
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_LinearStatistic 35b, 48, 55, 57, 74, 160a.

Uses: LECV 151b, LinearStatistic\_SLOT 22b.

$\langle C\_get\_Expectation\ 153a \rangle \equiv$

```
double* C_get_Expectation
(
     $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
    return(REAL(VECTOR_ELT(LECV, Expectation_SLOT)));
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_Expectation 37a, 42a, 46c, 55, 57, 74, 160a.

Uses: Expectation\_SLOT 22b, LECV 151b.

$\langle C\_get\_Variance\ 153b \rangle \equiv$

```
double* C_get_Variance
(
     $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
    int PQ = C_get_P(LECV) * C_get_Q(LECV);
    double *var, *covar;

    if (isNull(VECTOR_ELT(LECV, Variance_SLOT))) {
        SET_VECTOR_ELT(LECV, Variance_SLOT,
            allocVector(REALSXP, PQ));
        if (!isNull(VECTOR_ELT(LECV, Covariance_SLOT))) {
            covar = REAL(VECTOR_ELT(LECV, Covariance_SLOT));
            var = REAL(VECTOR_ELT(LECV, Variance_SLOT));
            for (int p = 0; p < PQ; p++)
                var[p] = covar[S(p, p, PQ)];
        }
    }
    return(REAL(VECTOR_ELT(LECV, Variance_SLOT)));
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_Variance 37c, 38b, 42a, 47, 48, 57, 74, 154a, 160a.

Uses: Covariance\_SLOT 22b, C\_get\_P 151c, C\_get\_Q 152a, LECV 151b, S 22a, Variance\_SLOT 22b.

$\langle C\_get\_Covariance\ 154a \rangle \equiv$

```
double* C_get_Covariance
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  int PQ = C_get_P(LECV) * C_get_Q(LECV);
  if (C_get_varonly(LECV) && PQ > 1)
    error("Cannot extract covariance from variance only object");
  if (C_get_varonly(LECV) && PQ == 1)
    return(C_get_Variance(LECV));
  return(REAL(VECTOR_ELT(LECV, Covariance_SLOT)));
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_Covariance 38ab, 42a, 47, 48, 55, 57, 74, 160a.

Uses: Covariance\_SLOT 22b, C\_get\_P 151c, C\_get\_Q 152a, C\_get\_Variance 153b, C\_get\_varonly 152b, LECV 151b.

$\langle C\_get\_ExpectationX\ 154b \rangle \equiv$

```
double* C_get_ExpectationX
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  return(REAL(VECTOR_ELT(LECV, ExpectationX_SLOT)));
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_ExpectationX 36a, 49, 74.

Uses: ExpectationX\_SLOT 22b, LECV 151b.

$\langle C\_get\_ExpectationInfluence\ 154c \rangle \equiv$

```
double* C_get_ExpectationInfluence
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  return(REAL(VECTOR_ELT(LECV, ExpectationInfluence_SLOT)));
}
◇
```

Fragment referenced in 151a.

Defines: C\_get\_ExpectationInfluence 36a, 49, 160a.

Uses: ExpectationInfluence\_SLOT 22b, LECV 151b.

$\langle C\_get\_CovarianceInfluence\ 155a \rangle \equiv$

```
double* C_get_CovarianceInfluence
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  return(REAL(VECTOR_ELT(LECV, CovarianceInfluence_SLOT)));
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_CovarianceInfluence` [36a](#), [47](#), [74](#), [160a](#).

Uses: `CovarianceInfluence_SLOT` [22b](#), `LECV` [151b](#).

$\langle C\_get\_VarianceInfluence\ 155b \rangle \equiv$

```
double* C_get_VarianceInfluence
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  return(REAL(VECTOR_ELT(LECV, VarianceInfluence_SLOT)));
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_VarianceInfluence` [36a](#), [47](#), [74](#), [160a](#).

Uses: `LECV` [151b](#), `VarianceInfluence_SLOT` [22b](#).

$\langle C\_get\_TableBlock\ 155c \rangle \equiv$

```
double* C_get_TableBlock
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  if (VECTOR_ELT(LECV, TableBlock_SLOT) == R_NilValue)
    error("object does not contain table block slot");
  return(REAL(VECTOR_ELT(LECV, TableBlock_SLOT)));
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_TableBlock` [36a](#).

Uses: `block` [28bd](#), `LECV` [151b](#), `TableBlock_SLOT` [22b](#).

$\langle C\_get\_Sumweights\ 156a \rangle \equiv$

```
double* C_get_Sumweights
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  if (VECTOR_ELT(LECV, Sumweights_SLOT) == R_NilValue)
    error("object does not contain sumweights slot");
  return(REAL(VECTOR_ELT(LECV, Sumweights_SLOT)));
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_Sumweights` [36a](#), [49](#).

Uses: `LECV` [151b](#), `sumweights` [27a](#), `Sumweights_SLOT` [22b](#).

$\langle C\_get\_Table\ 156b \rangle \equiv$

```
double* C_get_Table
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  if (LENGTH(LECV) <= Table_SLOT)
    error("Cannot extract table from object");
  return(REAL(VECTOR_ELT(LECV, Table_SLOT)));
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_Table` [44](#), [49](#).

Uses: `LECV` [151b](#), `Table_SLOT` [22b](#).

$\langle C\_get\_dimTable\ 156c \rangle \equiv$

```
int* C_get_dimTable
(
   $\langle R\ LECV\ Input\ 151b \rangle$ 
) {
  if (LENGTH(LECV) <= Table_SLOT)
    error("Cannot extract table from object");
  return(INTEGER(getAttrib(VECTOR_ELT(LECV, Table_SLOT),
    R_DimSymbol)));
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_dimTable` [49](#), [157a](#).

Uses: `LECV` [151b](#), `Table_SLOT` [22b](#).

$\langle C\_get\_B \text{ 157a} \rangle \equiv$

```
int C_get_B
(
   $\langle R \text{ LECV Input 151b} \rangle$ 
) {
  if (VECTOR_ELT(LECV, TableBlock_SLOT) != R_NilValue)
    return(LENGTH(VECTOR_ELT(LECV, Sumweights_SLOT)));
  return(C_get_dimTable(LECV)[2]);
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_B` [35a](#), [49](#), [74](#).

Uses: `C_get_dimTable` [156c](#), `LECV` [151b](#), `Sumweights_SLOT` [22b](#), `TableBlock_SLOT` [22b](#).

$\langle C\_get\_nresample \text{ 157b} \rangle \equiv$

```
R_xlen_t C_get_nresample
(
   $\langle R \text{ LECV Input 151b} \rangle$ 
) {
  int PQ = C_get_P(LECV) * C_get_Q(LECV);
  return(XLENGTH(VECTOR_ELT(LECV, PermutedLinearStatistic_SLOT)) / PQ);
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_nresample` [42a](#), [55](#), [56a](#), [57](#), [59](#), [74](#).

Uses: `C_get_P` [151c](#), `C_get_Q` [152a](#), `LECV` [151b](#), `PermutedLinearStatistic_SLOT` [22b](#).

$\langle C\_get\_PermutedLinearStatistic \text{ 157c} \rangle \equiv$

```
double* C_get_PermutedLinearStatistic
(
   $\langle R \text{ LECV Input 151b} \rangle$ 
) {
  return(REAL(VECTOR_ELT(LECV, PermutedLinearStatistic_SLOT)));
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_PermutedLinearStatistic` [42a](#), [55](#), [57](#), [74](#).

Uses: `LECV` [151b](#), `PermutedLinearStatistic_SLOT` [22b](#).

$\langle C\_get\_tol \text{ 157d} \rangle \equiv$

```
double C_get_tol
(
   $\langle R \text{ LECV Input 151b} \rangle$ 
) {
  return(REAL(VECTOR_ELT(LECV, tol_SLOT))[0]);
}
◇
```

Fragment referenced in [151a](#).

Defines: `C_get_tol` [42a](#), [55](#), [57](#), [74](#).

Uses: `LECV` [151b](#), `tol_SLOT` [22b](#).



⟨ *Memory Input Checks 158a* ⟩ ≡

```

if (P <= 0)
  error("P is not positive");

if (Q <= 0)
  error("Q is not positive");

if (B <= 0)
  error("B is not positive");

if (varonly < 0 || varonly > 1)
  error("varonly is not 0 or 1");

if (Xfactor < 0 || Xfactor > 1)
  error("Xfactor is not 0 or 1");

if (tol <= DBL_MIN)
  error("tol is not positive");

```

Fragment referenced in 159.  
 Uses: B 28c, P 25a, Q 25e.

⟨ *Memory Names 158b* ⟩ ≡

```

PROTECT(names = allocVector(STRSXP, Table_SLOT + 1));
SET_STRING_ELT(names, LinearStatistic_SLOT, mkChar("LinearStatistic"));
SET_STRING_ELT(names, Expectation_SLOT, mkChar("Expectation"));
SET_STRING_ELT(names, varonly_SLOT, mkChar("varonly"));
SET_STRING_ELT(names, Variance_SLOT, mkChar("Variance"));
SET_STRING_ELT(names, Covariance_SLOT, mkChar("Covariance"));
SET_STRING_ELT(names, ExpectationX_SLOT, mkChar("ExpectationX"));
SET_STRING_ELT(names, dim_SLOT, mkChar("dimension"));
SET_STRING_ELT(names, ExpectationInfluence_SLOT,
  mkChar("ExpectationInfluence"));
SET_STRING_ELT(names, Xfactor_SLOT, mkChar("Xfactor"));
SET_STRING_ELT(names, CovarianceInfluence_SLOT,
  mkChar("CovarianceInfluence"));
SET_STRING_ELT(names, VarianceInfluence_SLOT,
  mkChar("VarianceInfluence"));
SET_STRING_ELT(names, TableBlock_SLOT, mkChar("TableBlock"));
SET_STRING_ELT(names, Sumweights_SLOT, mkChar("Sumweights"));
SET_STRING_ELT(names, PermutedLinearStatistic_SLOT,
  mkChar("PermutedLinearStatistic"));
SET_STRING_ELT(names, StandardisedPermutedLinearStatistic_SLOT,
  mkChar("StandardisedPermutedLinearStatistic"));
SET_STRING_ELT(names, tol_SLOT, mkChar("tol"));
SET_STRING_ELT(names, Table_SLOT, mkChar("Table"));

```

Fragment referenced in 159.

Uses: CovarianceInfluence\_SLOT 22b, Covariance\_SLOT 22b, dim\_SLOT 22b, ExpectationInfluence\_SLOT 22b, ExpectationX\_SLOT 22b, Expectation\_SLOT 22b, LinearStatistic\_SLOT 22b, PermutedLinearStatistic\_SLOT 22b, StandardisedPermutedLinearStatistic\_SLOT 22b, Sumweights\_SLOT 22b, TableBlock\_SLOT 22b, Table\_SLOT 22b, tol\_SLOT 22b, VarianceInfluence\_SLOT 22b, Variance\_SLOT 22b, varonly\_SLOT 22b, Xfactor\_SLOT 22b.

$\langle R\_init\_LECV\ 159 \rangle \equiv$

```

SEXP vo, d, names, tolerance, tmp;
int PQ;

 $\langle$  Memory Input Checks 158a $\rangle$ 
PQ = mPQB(P, Q, 1);
 $\langle$  Memory Names 158b $\rangle$ 

/* Table_SLOT is always last and only used in 2d case, ie omitted here */
PROTECT(ans = allocVector(VECSXP, Table_SLOT + 1));
SET_VECTOR_ELT(ans, LinearStatistic_SLOT, allocVector(REALSXP, PQ));
SET_VECTOR_ELT(ans, Expectation_SLOT, allocVector(REALSXP, PQ));
SET_VECTOR_ELT(ans, varonly_SLOT, vo = allocVector(INTSXP, 1));
INTEGER(vo)[0] = varonly;
if (varonly) {
    SET_VECTOR_ELT(ans, Variance_SLOT, tmp = allocVector(REALSXP, PQ));
} else {
    /* always return variance */
    SET_VECTOR_ELT(ans, Variance_SLOT, tmp = allocVector(REALSXP, PQ));
    SET_VECTOR_ELT(ans, Covariance_SLOT,
        tmp = allocVector(REALSXP, PP12(PQ)));
}
SET_VECTOR_ELT(ans, ExpectationX_SLOT, allocVector(REALSXP, P));
SET_VECTOR_ELT(ans, dim_SLOT, d = allocVector(INTSXP, 2));
INTEGER(d)[0] = P;
INTEGER(d)[1] = Q;
SET_VECTOR_ELT(ans, ExpectationInfluence_SLOT,
    tmp = allocVector(REALSXP, B * Q));
for (int q = 0; q < B * Q; q++) REAL(tmp)[q] = 0.0;

/* should always _both_ be there */
SET_VECTOR_ELT(ans, VarianceInfluence_SLOT,
    tmp = allocVector(REALSXP, B * Q));
for (int q = 0; q < B * Q; q++) REAL(tmp)[q] = 0.0;

SET_VECTOR_ELT(ans, CovarianceInfluence_SLOT,
    tmp = allocVector(REALSXP, B * Q * (Q + 1) / 2));
for (int q = 0; q < B * Q * (Q + 1) / 2; q++) REAL(tmp)[q] = 0.0;

SET_VECTOR_ELT(ans, Xfactor_SLOT, allocVector(INTSXP, 1));
INTEGER(VECTOR_ELT(ans, Xfactor_SLOT))[0] = Xfactor;
SET_VECTOR_ELT(ans, TableBlock_SLOT, tmp = allocVector(REALSXP, B + 1));
for (int q = 0; q < B + 1; q++) REAL(tmp)[q] = 0.0;
SET_VECTOR_ELT(ans, Sumweights_SLOT, allocVector(REALSXP, B));
SET_VECTOR_ELT(ans, PermutedLinearStatistic_SLOT,
    allocMatrix(REALSXP, 0, 0));
SET_VECTOR_ELT(ans, StandardisedPermutedLinearStatistic_SLOT,
    allocMatrix(REALSXP, 0, 0));
SET_VECTOR_ELT(ans, tol_SLOT, tolerance = allocVector(REALSXP, 1));
REAL(tolerance)[0] = tol;
namesgets(ans, names);

 $\langle$  Initialise Zero 160a $\rangle$ 
 $\diamond$ 

```

Fragment referenced in 160b, 161.

Uses: B 28c, CovarianceInfluence\_SLOT 22b, Covariance\_SLOT 22b, dim\_SLOT 22b, ExpectationInfluence\_SLOT 22b, ExpectationX\_SLOT 22b, Expectation\_SLOT 22b, LinearStatistic\_SLOT 22b, mPQB 141a, P 25a, PermutedLinearStatistic\_SLOT 22b, PP12 140b, Q 25e, StandardisedPermutedLinearStatistic\_SLOT 22b, Sumweights\_SLOT 22b, TableBlock\_SLOT 22b, Table\_SLOT 22b, tol\_SLOT 22b, VarianceInfluence\_SLOT 22b, Variance\_SLOT 22b, varonly\_SLOT 22b, Xfactor\_SLOT 22b.

$\langle \text{Initialise Zero 160a} \rangle \equiv$

```

/* set initial zeros */
for (int p = 0; p < PQ; p++) {
    C_get_LinearStatistic(ans)[p] = 0.0;
    C_get_Expectation(ans)[p] = 0.0;
    if (varonly)
        C_get_Variance(ans)[p] = 0.0;
}
if (!varonly) {
    for (int p = 0; p < PP12(PQ); p++)
        C_get_Covariance(ans)[p] = 0.0;
}
for (int q = 0; q < Q; q++) {
    C_get_ExpectationInfluence(ans)[q] = 0.0;
    C_get_VarianceInfluence(ans)[q] = 0.0;
}
for (int q = 0; q < Q * (Q + 1) / 2; q++)
    C_get_CovarianceInfluence(ans)[q] = 0.0;

```

Fragment referenced in 159.

Uses: C\_get\_Covariance 154a, C\_get\_CovarianceInfluence 155a, C\_get\_Expectation 153a,  
 C\_get\_ExpectationInfluence 154c, C\_get\_LinearStatistic 152d, C\_get\_Variance 153b,  
 C\_get\_VarianceInfluence 155b, PP12 140b, Q 25e.

$\langle \text{RC\_init\_LECV\_1d 160b} \rangle \equiv$

```

SEXP RC_init_LECV_1d
(
     $\langle C \text{ integer } P \text{ Input 25a} \rangle$ ,
     $\langle C \text{ integer } Q \text{ Input 25e} \rangle$ ,
    int varonly,
     $\langle C \text{ integer } B \text{ Input 28c} \rangle$ ,
    int Xfactor,
    double tol
) {
    SEXP ans;

     $\langle R\_init\_LECV 159 \rangle$ 

    SET_VECTOR_ELT(ans, TableBlock_SLOT,
        allocVector(REALSXP, B + 1));

    SET_VECTOR_ELT(ans, Sumweights_SLOT,
        allocVector(REALSXP, B));

    UNPROTECT(2);
    return(ans);
}

```

Fragment referenced in 151a.

Defines: RC\_init\_LECV\_1d 32c.

Uses: B 28c, Sumweights\_SLOT 22b, TableBlock\_SLOT 22b.

$\langle RC\_init\_LECV\_2d\ 161 \rangle \equiv$

```

SEXP RC_init_LECV_2d
(
   $\langle C\ integer\ P\ Input\ 25a \rangle$ ,
   $\langle C\ integer\ Q\ Input\ 25e \rangle$ ,
  int varonly,
  int Lx,
  int Ly,
   $\langle C\ integer\ B\ Input\ 28c \rangle$ ,
  int Xfactor,
  double tol
) {
  SEXP ans, tabdim, tab;

  if (Lx <= 0)
    error("Lx is not positive");

  if (Ly <= 0)
    error("Ly is not positive");

   $\langle R\_init\_LECV\ 159 \rangle$ 

  PROTECT(tabdim = allocVector(INTSXP, 3));
  INTEGER(tabdim)[0] = Lx + 1;
  INTEGER(tabdim)[1] = Ly + 1;
  INTEGER(tabdim)[2] = B;
  SET_VECTOR_ELT(ans, Table_SLOT,
    tab = allocVector(REALSXP,
      INTEGER(tabdim)[0] *
      INTEGER(tabdim)[1] *
      INTEGER(tabdim)[2]));
  dimgets(tab, tabdim);

  UNPROTECT(3);
  return(ans);
}

```

Fragment referenced in [151a](#).  
 Defines: RC\_init\_LECV\_2d [44](#).  
 Uses: B [28c](#), Table\_SLOT [22b](#).

## Chapter 4

# Package Infrastructure

"AAA.R" 162a≡

```
< R Header 166a >
.onUnload <- function(libpath)
  library.dynam.unload("libcoin", libpath)
◇
```

"DESCRIPTION" 162b≡

```
Package: libcoin
Title: Linear Test Statistics for Permutation Inference
Date: 2021-09-27
Version: 1.0-9
Authors@R: person("Torsten", "Hothorn", role = c("aut", "cre"),
                  email = "Torsten.Hothorn@R-project.org")
Description: Basic infrastructure for linear test statistics and permutation
             inference in the framework of Strasser and Weber (1999) <https://epub.wu.ac.at/102/>.
             This package must not be used by end-users. CRAN package 'coin' implements all
             user interfaces and is ready to be used by anyone.
Depends: R (>= 3.4.0)
Suggests: coin
Imports: stats, mvtnorm
LinkingTo: mvtnorm
NeedsCompilation: yes
License: GPL-2
◇
```

"NAMESPACE" 162c≡

```
useDynLib(libcoin, .registration = TRUE)

importFrom("stats", complete.cases, vcov)
importFrom("mvtnorm", GenzBretz)

export(LinStatExpCov, doTest, ctab, "lmult")
S3method("vcov", "LinStatExpCov")
◇
```

Add flag `-g` to `PKG\_CFLAGS` for `perf` profiling (this is not portable).

"Makevars" 163a≡

```
PKG_CFLAGS=$(C_VISIBILITY)
PKG_LIBS = $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS)
◇
```

"libcoin-win.def" 163b≡

```
LIBRARY libcoin.dll
EXPORTS
    R_init_libcoin
◇
```

Other packages can link against **libcoin**. A small example package is contained in `libcoin/inst/C_API_example`.

"libcoin-init.c" 164≡

```
< C Header 166b >
#include "libcoin.h"
#include <R_ext/Rdynload.h>
#include <R_ext/Visibility.h>

#define CALLDEF(name, n) {#name, (DL_FUNC) &name, n}
#define REGCALL(name) R_RegisterCCallable("libcoin", #name, (DL_FUNC) &name)

static const R_CallMethodDef callMethods[] = {
    CALLDEF(R_ExpectationCovarianceStatistic, 7),
    CALLDEF(R_PermutedLinearStatistic, 6),
    CALLDEF(R_StandardisePermutedLinearStatistic, 1),
    CALLDEF(R_ExpectationCovarianceStatistic_2d, 9),
    CALLDEF(R_PermutedLinearStatistic_2d, 7),
    CALLDEF(R_QuadraticTest, 5),
    CALLDEF(R_MaximumTest, 9),
    CALLDEF(R_MaximallySelectedTest, 6),
    CALLDEF(R_ExpectationInfluence, 3),
    CALLDEF(R_CovarianceInfluence, 4),
    CALLDEF(R_ExpectationX, 4),
    CALLDEF(R_CovarianceX, 5),
    CALLDEF(R_Sums, 3),
    CALLDEF(R_KronSums, 6),
    CALLDEF(R_KronSums_Permutation, 5),
    CALLDEF(R_colSums, 3),
    CALLDEF(R_OneTableSums, 3),
    CALLDEF(R_TwoTableSums, 4),
    CALLDEF(R_ThreeTableSums, 5),
    CALLDEF(R_order_subset_wrt_block, 4),
    CALLDEF(R_quadform, 3),
    CALLDEF(R_kronecker, 2),
    CALLDEF(R_MPinv_sym, 3),
    CALLDEF(R_unpack_sym, 3),
    CALLDEF(R_pack_sym, 1),
    {NULL, NULL, 0}
};
◇
```

File defined by 164, 165.

Uses: R\_colSums 113b, R\_CovarianceInfluence 87a, R\_CovarianceX 92a, R\_ExpectationCovarianceStatistic 32c, R\_ExpectationCovarianceStatistic\_2d 44, R\_ExpectationInfluence 85b, R\_ExpectationX 89a, R\_KronSums 100a, R\_KronSums\_Permutation 109b, R\_MPinv\_sym 146b, R\_OneTableSums 118a, R\_order\_subset\_wrt\_block 132b, R\_pack\_sym 150c, R\_PermutedLinearStatistic 40, R\_PermutedLinearStatistic\_2d 51, R\_quadform 64c, R\_Sums 95b, R\_ThreeTableSums 127b, R\_TwoTableSums 122b, R\_unpack\_sym 149.

"libcoin-init.c" 165≡

```

void attribute_visible R_init_libcoin
(
    DllInfo *dll
) {
    R_registerRoutines(dll, NULL, callMethods, NULL, NULL);
    R_useDynamicSymbols(dll, FALSE);
    R_forceSymbols(dll, TRUE);
    REGCALL(R_ExpectationCovarianceStatistic);
    REGCALL(R_PermutedLinearStatistic);
    REGCALL(R_StandardisePermutedLinearStatistic);
    REGCALL(R_ExpectationCovarianceStatistic_2d);
    REGCALL(R_PermutedLinearStatistic_2d);
    REGCALL(R_QuadraticTest);
    REGCALL(R_MaximumTest);
    REGCALL(R_MaximallySelectedTest);
    REGCALL(R_ExpectationInfluence);
    REGCALL(R_CovarianceInfluence);
    REGCALL(R_ExpectationX);
    REGCALL(R_CovarianceX);
    REGCALL(R_Sums);
    REGCALL(R_KronSums);
    REGCALL(R_KronSums_Permutation);
    REGCALL(R_colSums);
    REGCALL(R_OneTableSums);
    REGCALL(R_TwoTableSums);
    REGCALL(R_ThreeTableSums);
    REGCALL(R_order_subset_wrt_block);
    REGCALL(R_quadform);
    REGCALL(R_kronecker);
    REGCALL(R_MPinv_sym);
    REGCALL(R_unpack_sym);
    REGCALL(R_pack_sym);
}
◇

```

File defined by [164](#), [165](#).

Uses: [R\\_colSums 113b](#), [R\\_CovarianceInfluence 87a](#), [R\\_CovarianceX 92a](#), [R\\_ExpectationCovarianceStatistic 32c](#),  
[R\\_ExpectationCovarianceStatistic\\_2d 44](#), [R\\_ExpectationInfluence 85b](#), [R\\_ExpectationX 89a](#), [R\\_KronSums 100a](#),  
[R\\_KronSums\\_Permutation 109b](#), [R\\_MPinv\\_sym 146b](#), [R\\_OneTableSums 118a](#), [R\\_order\\_subset\\_wrt\\_block 132b](#),  
[R\\_pack\\_sym 150c](#), [R\\_PermutedLinearStatistic 40](#), [R\\_PermutedLinearStatistic\\_2d 51](#), [R\\_quadform 64c](#), [R\\_Sums 95b](#),  
[R\\_ThreeTableSums 127b](#), [R\\_TwoTableSums 122b](#), [R\\_unpack\\_sym 149](#).



⟨ *R Header 166a* ⟩ ≡

```
### Copyright (C) 2017-2021 Torsten Hothorn
###
### This file is part of the 'libcoin' R add-on package.
###
### 'libcoin' is free software: you can redistribute it and/or modify
### it under the terms of the GNU General Public License as published by
### the Free Software Foundation, version 2.
###
### 'libcoin' is distributed in the hope that it will be useful,
### but WITHOUT ANY WARRANTY; without even the implied warranty of
### MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
### GNU General Public License for more details.
###
### You should have received a copy of the GNU General Public License
### along with 'libcoin'. If not, see <http://www.gnu.org/licenses/>.
###
### DO NOT EDIT THIS FILE
###
### Edit 'libcoin.w' and run 'nuweb -r libcoin.w'
```

◇

Fragment referenced in [3a](#), [16](#), [162a](#).

⟨ *C Header 166b* ⟩ ≡

```
/*
Copyright (C) 2017-2021 Torsten Hothorn

This file is part of the 'libcoin' R add-on package.

'libcoin' is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, version 2.

'libcoin' is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with 'libcoin'. If not, see <http://www.gnu.org/licenses/>.

DO NOT EDIT THIS FILE

Edit 'libcoin.w' and run 'nuweb -r libcoin.w'
*/
```

◇

Fragment referenced in [21a](#), [23ac](#), [32a](#), [164](#).

# Index

## Files

"AAA.R" Defined by 162a.  
"ctabs.R" Defined by 16.  
"ctabs.Rd" Defined by 20.  
"DESCRIPTION" Defined by 162b.  
"doTest.Rd" Defined by 19.  
"libcoin-init.c" Defined by 164, 165.  
"libcoin-win.def" Defined by 163b.  
"libcoin.c" Defined by 23c.  
"libcoin.h" Defined by 23a.  
"libcoin.R" Defined by 3a.  
"libcoinAPI.h" Defined by 32a, 38d, 41b, 43b, 50b, 54a, 64a, 141b, 145b, 148a, 150a.  
"libcoin\_internal.h" Defined by 21a.  
"LinStatExpCov.Rd" Defined by 18.  
"Makevars" Defined by 163a.  
"NAMESPACE" Defined by 162c.

## Fragments

$\langle$  2d Covariance 47  $\rangle$  Referenced in 48.  
 $\langle$  2d Expectation 46c  $\rangle$  Referenced in 48.  
 $\langle$  2d Memory 49  $\rangle$  Referenced in 48.  
 $\langle$  2d Total Table 46a  $\rangle$  Referenced in 48.  
 $\langle$  2d User Interface 42b  $\rangle$  Referenced in 24a.  
 $\langle$  2d User Interface Input 42c  $\rangle$  Referenced in 43a, 48.  
 $\langle$  C colSums Answer 114c  $\rangle$  Referenced in 85c, 113c, 115abc, 116a.  
 $\langle$  C colSums Input 114b  $\rangle$  Referenced in 113c, 115abc, 116a.  
 $\langle$  C Global Variables 22b  $\rangle$  Referenced in 21a.  
 $\langle$  C Header 166b  $\rangle$  Referenced in 21a, 23ac, 32a, 164.  
 $\langle$  C integer B Input 28c  $\rangle$  Referenced in 28d, 34, 160b, 161.  
 $\langle$  C integer block Input 28d  $\rangle$  Referenced in 128c.  
 $\langle$  C integer N Input 24c  $\rangle$  Referenced in 25bc, 34, 40, 44, 81c, 85bc, 87ab, 89ab, 92ab, 95c, 96b, 97abc, 100a, 101b, 109bc, 113b, 118a, 122b, 127b, 132b, 133a, 134ab, 135b.  
 $\langle$  C integer Nsubset Input 27c  $\rangle$  Referenced in 27d, 40, 44, 85b, 87a, 89a, 92a, 95b, 100a, 109b, 113b, 118a, 122b, 127b, 137ab, 138b.  
 $\langle$  C integer P Input 25a  $\rangle$  Referenced in 25bc, 34, 81c, 82b, 83, 84, 89b, 92b, 101b, 109c, 160b, 161.  
 $\langle$  C integer Q Input 25e  $\rangle$  Referenced in 26ab, 34, 82b, 83, 84, 85bc, 87ab, 100a, 109b, 160b, 161.  
 $\langle$  C integer subset Input 27e  $\rangle$  Referenced in 97bc, 104bc, 107ab, 111a, 112a, 115c, 116a, 120c, 121a, 125ab, 130ab.  
 $\langle$  C integer weights Input 26d  $\rangle$  Referenced in 97ab, 104ab, 106c, 107a, 115bc, 120bc, 124c, 125a, 129c, 130a.  
 $\langle$  C integer x Input 25c  $\rangle$  Referenced in 106a, 111c, 112a, 119b, 123c, 128c.  
 $\langle$  C integer y Input 26b  $\rangle$  Referenced in 123c, 128c.  
 $\langle$  C KronSums Answer 101d  $\rangle$  Referenced in 81c, 87b, 92b, 100b, 103b, 104abc, 106bc, 107ab, 109c, 110b, 111ac, 112a.  
 $\langle$  C KronSums Input 101c  $\rangle$  Referenced in 103b, 104abc.  
 $\langle$  C Macros 22a  $\rangle$  Referenced in 21a.

< C OneTableSums Answer 119c > Referenced in 89b, 118b, 120abc, 121a.  
 < C OneTableSums Input 119b > Referenced in 118b, 120abc, 121a.  
 < C real subset Input 28a > Referenced in 96b, 97a, 103b, 104a, 106bc, 110b, 111c, 115ab, 120ab, 124bc, 129bc.  
 < C real weights Input 26e > Referenced in 96b, 97c, 103b, 104c, 106b, 107b, 115a, 116a, 120a, 121a, 124b, 125b, 129b, 130b.  
 < C real x Input 25b > Referenced in 101c, 110b, 111a, 114b, 145a.  
 < C real y Input 26a > Referenced in 81c, 101bc, 106a, 109c, 110b, 111ac, 112a.  
 < C subset range Input 27d > Referenced in 27e, 28a, 81c, 85c, 87b, 89b, 92b, 95c, 100b, 109c, 113c, 118b, 123a, 128a.  
 < C sumweights Input 27a > Referenced in 83, 84, 85c, 87b.  
 < C ThreeTableSums Answer 129a > Referenced in 128a, 129bc, 130ab.  
 < C ThreeTableSums Input 128c > Referenced in 128a, 129bc, 130ab.  
 < C TwoTableSums Answer 124a > Referenced in 123a, 124bc, 125ab.  
 < C TwoTableSums Input 123c > Referenced in 123a, 124bc, 125ab.  
 < C XfactorKronSums Input 106a > Referenced in 106bc, 107ab.  
 < Check ix 9a > Referenced in 8, 16.  
 < Check iy 9b > Referenced in 8, 16.  
 < Check weights, subset, block 5a > Referenced in 6, 8, 16.  
 < Col Row Total Sums 46b > Referenced in 48, 51.  
 < colSums 112c > Referenced in 24a.  
 < colSums Body 116b > Referenced in 115abc, 116a.  
 < Compute Covariance Influence 37b > Referenced in 34.  
 < Compute Covariance Linear Statistic 38a > Referenced in 34.  
 < Compute Expectation Linear Statistic 37a > Referenced in 34.  
 < Compute Linear Statistic 35b > Referenced in 34.  
 < Compute maxstat Permutation P-Value 77 > Referenced in 73, 78.  
 < Compute maxstat Test Statistic 76c > Referenced in 73, 78.  
 < Compute maxstat Variance / Covariance Directly 76b > Referenced in 73.  
 < Compute maxstat Variance / Covariance from Total Covariance 76a > Referenced in 73.  
 < Compute Permuted Linear Statistic 2d 53a > Referenced in 51.  
 < Compute Sum of Weights in Block 36b > Referenced in 34.  
 < Compute unordered maxstat Linear Statistic and Expectation 80a > Referenced in 78.  
 < Compute unordered maxstat Variance / Covariance Directly 81a > Referenced in 78.  
 < Compute unordered maxstat Variance / Covariance from Total Covariance 80b > Referenced in 78.  
 < Compute Variance from Covariance 38b > Referenced in 34.  
 < Compute Variance Linear Statistic 37c > Referenced in 34.  
 < continue subset loop 94b > Referenced in 98a, 105, 108, 116b, 121b, 126, 131a.  
 < Contrasts 14 > Referenced in 3a.  
 < Convert Table to Integer 52a > Referenced in 51.  
 < Count Levels 79a > Referenced in 78.  
 < ctab Prototype 15 > Referenced in 16, 20.  
 < C\_chisq\_pvalue 67c > Referenced in 67b.  
 < C\_colSums\_dweights\_dsubset 115a > Referenced in 112c.  
 < C\_colSums\_dweights\_isubset 116a > Referenced in 112c.  
 < C\_colSums\_iweights\_dsubset 115b > Referenced in 112c.  
 < C\_colSums\_iweights\_isubset 115c > Referenced in 112c.  
 < C\_CovarianceLinearStatistic 83 > Referenced in 82a.  
 < C\_doPermute 137b > Referenced in 136b.  
 < C\_doPermuteBlock 138b > Referenced in 136b.  
 < C\_ExpectationLinearStatistic 82b > Referenced in 82a.  
 < C\_get\_B 157a > Referenced in 151a.  
 < C\_get\_Covariance 154a > Referenced in 151a.  
 < C\_get\_CovarianceInfluence 155a > Referenced in 151a.  
 < C\_get\_dimTable 156c > Referenced in 151a.  
 < C\_get\_Expectation 153a > Referenced in 151a.  
 < C\_get\_ExpectationInfluence 154c > Referenced in 151a.  
 < C\_get\_ExpectationX 154b > Referenced in 151a.  
 < C\_get\_LinearStatistic 152d > Referenced in 151a.  
 < C\_get\_nresample 157b > Referenced in 151a.  
 < C\_get\_P 151c > Referenced in 151a.

<C\_get\_PermutedLinearStatistic 157c> Referenced in 151a.  
 <C\_get\_Q 152a> Referenced in 151a.  
 <C\_get\_Sumweights 156a> Referenced in 151a.  
 <C\_get\_Table 156b> Referenced in 151a.  
 <C\_get\_TableBlock 155c> Referenced in 151a.  
 <C\_get\_tol 157d> Referenced in 151a.  
 <C\_get\_Variance 153b> Referenced in 151a.  
 <C\_get\_VarianceInfluence 155b> Referenced in 151a.  
 <C\_get\_varonly 152b> Referenced in 151a.  
 <C\_get\_Xfactor 152c> Referenced in 151a.  
 <C\_kronecker 143> Referenced in 139a.  
 <C\_kronecker\_sym 144> Referenced in 139a.  
 <C\_KronSums\_dweights\_dsubset 103b> Referenced in 98b.  
 <C\_KronSums\_dweights\_isubset 104c> Referenced in 98b.  
 <C\_KronSums\_iweights\_dsubset 104a> Referenced in 98b.  
 <C\_KronSums\_iweights\_isubset 104b> Referenced in 98b.  
 <C\_KronSums\_Permutation\_dsubset 110b> Referenced in 98b.  
 <C\_KronSums\_Permutation\_isubset 111a> Referenced in 98b.  
 <C\_KronSums\_sym 145a> Referenced in 139a.  
 <C\_maxabsstand\_Covariance 62b> Referenced in 60a.  
 <C\_maxabsstand\_Variance 63> Referenced in 60a.  
 <C\_maxstand\_Covariance 60b> Referenced in 60a.  
 <C\_maxstand\_Variance 61a> Referenced in 60a.  
 <C\_maxtype 66> Referenced in 60a.  
 <C\_maxtype\_pvalue 70> Referenced in 67b.  
 <C\_minstand\_Covariance 61b> Referenced in 60a.  
 <C\_minstand\_Variance 62a> Referenced in 60a.  
 <C\_MPinv\_sym 147> Referenced in 139a.  
 <C\_norm\_pvalue 69> Referenced in 67b.  
 <C\_OneTableSums\_dweights\_dsubset 120a> Referenced in 117a.  
 <C\_OneTableSums\_dweights\_isubset 121a> Referenced in 117a.  
 <C\_OneTableSums\_iweights\_dsubset 120b> Referenced in 117a.  
 <C\_OneTableSums\_iweights\_isubset 120c> Referenced in 117a.  
 <C\_ordered\_Xfactor 73> Referenced in 60a.  
 <C\_order\_subset\_wrt\_block 135a> Referenced in 131b.  
 <C\_Permute 137a> Referenced in 136b.  
 <C\_PermuteBlock 138a> Referenced in 136b.  
 <C\_perm\_pvalue 68> Referenced in 67b.  
 <C\_quadform 65> Referenced in 60a.  
 <C\_setup\_subset 134a> Referenced in 131b.  
 <C\_setup\_subset\_block 134b> Referenced in 131b.  
 <C\_standardise 67a> Referenced in 60a.  
 <C\_Sums\_dweights\_dsubset 96b> Referenced in 94c.  
 <C\_Sums\_dweights\_isubset 97c> Referenced in 94c.  
 <C\_Sums\_iweights\_dsubset 97a> Referenced in 94c.  
 <C\_Sums\_iweights\_isubset 97b> Referenced in 94c.  
 <C\_ThreeTableSums\_dweights\_dsubset 129b> Referenced in 117a.  
 <C\_ThreeTableSums\_dweights\_isubset 130b> Referenced in 117a.  
 <C\_ThreeTableSums\_iweights\_dsubset 129c> Referenced in 117a.  
 <C\_ThreeTableSums\_iweights\_isubset 130a> Referenced in 117a.  
 <C\_TwoTableSums\_dweights\_dsubset 124b> Referenced in 117a.  
 <C\_TwoTableSums\_dweights\_isubset 125b> Referenced in 117a.  
 <C\_TwoTableSums\_iweights\_dsubset 124c> Referenced in 117a.  
 <C\_TwoTableSums\_iweights\_isubset 125a> Referenced in 117a.  
 <C\_unordered\_Xfactor 78> Referenced in 60a.  
 <C\_VarianceLinearStatistic 84> Referenced in 82a.  
 <C\_XfactorKronSums\_dweights\_dsubset 106b> Referenced in 98b.  
 <C\_XfactorKronSums\_dweights\_isubset 107b> Referenced in 98b.

< C\_XfactorKronSums\_iweights\_dsubset 106c > Referenced in 98b.  
 < C\_XfactorKronSums\_iweights\_isubset 107a > Referenced in 98b.  
 < C\_XfactorKronSums\_Permutation\_dsubset 111c > Referenced in 98b.  
 < C\_XfactorKronSums\_Permutation\_isubset 112a > Referenced in 98b.  
 < doTest 12 > Referenced in 3a.  
 < doTest Prototype 11 > Referenced in 12, 19.  
 < ExpectationCovariances 82a > Referenced in 24a.  
 < Extract Dimensions 35a > Referenced in 34.  
 < Function Definitions 24a > Referenced in 23c.  
 < Function Prototypes 23b > Referenced in 23a.  
 < Handle Missing Values 5b > Referenced in 6.  
 < init subset loop 93b > Referenced in 98a, 105, 108, 116b, 121b, 126, 131a.  
 < Initialise Zero 160a > Referenced in 159.  
 < KronSums 98b > Referenced in 24a.  
 < KronSums Body 105 > Referenced in 103b, 104abc.  
 < KronSums Double x 103a > Referenced in 101a.  
 < KronSums Integer x 102 > Referenced in 101a.  
 < KronSums Permutation Body 111b > Referenced in 110b, 111a.  
 < Linear Statistic 2d 45b > Referenced in 48, 53a.  
 < LinearStatistics 81b > Referenced in 24a.  
 < LinStatExpCov 4 > Referenced in 3a.  
 < LinStatExpCov Prototype 3b > Referenced in 4, 18.  
 < LinStatExpCov1d 6 > Referenced in 3a.  
 < LinStatExpCov2d 8 > Referenced in 3a.  
 < maxstat Xfactor Variables 72b > Referenced in 73, 78.  
 < Memory 151a > Referenced in 24a.  
 < Memory Input Checks 158a > Referenced in 159.  
 < Memory Names 158b > Referenced in 159.  
 < MoreUtils 139a > Referenced in 24a.  
 < mPQB 141a > Referenced in 151a.  
 < NCOL 139c > Referenced in 139a.  
 < NLEVELS 140a > Referenced in 139a.  
 < NROW 139b > Referenced in 139a.  
 < OneTableSums Body 121b > Referenced in 120abc, 121a.  
 < P-Values 67b > Referenced in 24a.  
 < Permutations 136b > Referenced in 24a.  
 < PP12 140b > Referenced in 151a.  
 < R block Input 28b > Referenced in 31b, 42c, 50a, 127a, 132a, 133a, 134b, 135a.  
 < R blockTable Input 28e > Referenced in 133a, 134b, 135a.  
 < R Header 166a > Referenced in 3a, 16, 162a.  
 < R Includes 21b > Referenced in 21a.  
 < R LECV Input 151b > Referenced in 54b, 56b, 151c, 152abcd, 153ab, 154abc, 155abc, 156abc, 157abcd.  
 < R N Input 24b > Referenced in 95a.  
 < R subset Input 27b > Referenced in 31b, 42c, 81c, 85ac, 86b, 87b, 88b, 89b, 91, 92b, 95ac, 99, 100b, 109ac, 113ac, 117b, 118b, 122a, 123a, 127a, 128a, 132a, 133a, 135ab.  
 < R weights Input 26c > Referenced in 31b, 42c, 81c, 85ac, 86b, 87b, 88b, 89b, 91, 92b, 95ac, 99, 100b, 113ac, 117b, 118b, 122a, 123a, 127a, 128a, 132a, 135b.  
 < R x Input 24d > Referenced in 31b, 42c, 50a, 81c, 88b, 89b, 91, 92b, 99, 101b, 109ac, 113a, 117b, 122a, 127a.  
 < R y Input 25d > Referenced in 31b, 42c, 50a, 85ac, 86b, 87b, 99, 109a, 122a, 127a, 132a.  
 < RC KronSums Input 101b > Referenced in 100b.  
 < RC\_colSums 114a > Referenced in 112c.  
 < RC\_colSums Prototype 113c > Referenced in 114a.  
 < RC\_CovarianceInfluence 88a > Referenced in 82a.  
 < RC\_CovarianceInfluence Prototype 87b > Referenced in 88a.  
 < RC\_CovarianceX 93a > Referenced in 82a.  
 < RC\_CovarianceX Prototype 92b > Referenced in 93a.  
 < RC\_ExpectationCovarianceStatistic 34 > Referenced in 31a.  
 < RC\_ExpectationCovarianceStatistic\_2d 48 > Referenced in 42b.

<RC\_ExpectationInfluence 86a> Referenced in 82a.  
 <RC\_ExpectationInfluence Prototype 85c> Referenced in 86a.  
 <RC\_ExpectationX 90> Referenced in 82a.  
 <RC\_ExpectationX Prototype 89b> Referenced in 90.  
 <RC\_init\_LECV\_1d 160b> Referenced in 151a.  
 <RC\_init\_LECV\_2d 161> Referenced in 151a.  
 <RC\_KronSums 101a> Referenced in 98b.  
 <RC\_KronSums Prototype 100b> Referenced in 101a.  
 <RC\_KronSums.Permutation 110a> Referenced in 98b.  
 <RC\_KronSums.Permutation Prototype 109c> Referenced in 110a.  
 <RC\_LinearStatistic 81d> Referenced in 81b.  
 <RC\_LinearStatistic Prototype 81c> Referenced in 81d.  
 <RC\_OneTableSums 119a> Referenced in 117a.  
 <RC\_OneTableSums Prototype 118b> Referenced in 119a.  
 <RC\_order\_subset\_wrt\_block 133b> Referenced in 131b.  
 <RC\_order\_subset\_wrt\_block Prototype 133a> Referenced in 133b.  
 <RC\_setup\_subset 136a> Referenced in 136b.  
 <RC\_setup\_subset Prototype 135b> Referenced in 136a.  
 <RC\_Sums 96a> Referenced in 94c.  
 <RC\_Sums Prototype 95c> Referenced in 96a.  
 <RC\_ThreeTableSums 128b> Referenced in 117a.  
 <RC\_ThreeTableSums Prototype 128a> Referenced in 128b.  
 <RC\_TwoTableSums 123b> Referenced in 117a.  
 <RC\_TwoTableSums Prototype 123a> Referenced in 123b.  
 <R\_colSums 113b> Referenced in 112c.  
 <R\_colSums Prototype 113a> Referenced in 23b, 113b.  
 <R\_CovarianceInfluence 87a> Referenced in 82a.  
 <R\_CovarianceInfluence Prototype 86b> Referenced in 23b, 87a.  
 <R\_CovarianceX 92a> Referenced in 82a.  
 <R\_CovarianceX Prototype 91> Referenced in 23b, 92a.  
 <R\_ExpectationCovarianceStatistic 32c> Referenced in 31a.  
 <R\_ExpectationCovarianceStatistic Prototype 32b> Referenced in 23b, 32c.  
 <R\_ExpectationCovarianceStatistic\_2d 44> Referenced in 42b.  
 <R\_ExpectationCovarianceStatistic\_2d Prototype 43a> Referenced in 23b, 44.  
 <R\_ExpectationInfluence 85b> Referenced in 82a.  
 <R\_ExpectationInfluence Prototype 85a> Referenced in 23b, 85b.  
 <R\_ExpectationX 89a> Referenced in 82a.  
 <R\_ExpectationX Prototype 88b> Referenced in 23b, 89a.  
 <R\_init\_LECV 159> Referenced in 160b, 161.  
 <R\_kronecker 142> Referenced in 139a.  
 <R\_kronecker Prototype 141c> Referenced in 23b, 142.  
 <R\_KronSums 100a> Referenced in 98b.  
 <R\_KronSums Prototype 99> Referenced in 23b, 100a.  
 <R\_KronSums.Permutation 109b> Referenced in 98b.  
 <R\_KronSums.Permutation Prototype 109a> Referenced in 23b, 109b.  
 <R\_MaximallySelectedTest 59> Referenced in 53b.  
 <R\_MaximallySelectedTest Prototype 58> Referenced in 23b, 59.  
 <R\_MaximumTest 57> Referenced in 53b.  
 <R\_MaximumTest Prototype 56b> Referenced in 23b, 57.  
 <R\_MPinv\_sym 146b> Referenced in 139a.  
 <R\_MPinv\_sym Prototype 146a> Referenced in 23b, 146b.  
 <R\_OneTableSums 118a> Referenced in 117a.  
 <R\_OneTableSums Prototype 117b> Referenced in 23b, 118a.  
 <R\_order\_subset\_wrt\_block 132b> Referenced in 131b.  
 <R\_order\_subset\_wrt\_block Prototype 132a> Referenced in 23b, 132b.  
 <R\_pack\_sym 150c> Referenced in 139a.  
 <R\_pack\_sym Prototype 150b> Referenced in 23b, 150c.  
 <R\_PermutedLinearStatistic 40> Referenced in 31a.

<R\_PermutedLinearStatistic Prototype 38c> Referenced in 23b, 40.  
 <R\_PermutedLinearStatistic\_2d 51> Referenced in 42b.  
 <R\_PermutedLinearStatistic\_2d Prototype 50a> Referenced in 23b, 51.  
 <R\_quadform 64c> Referenced in 60a.  
 <R\_quadform Prototype 64b> Referenced in 23b, 64c.  
 <R\_QuadraticTest 55> Referenced in 53b.  
 <R\_QuadraticTest Prototype 54b> Referenced in 23b, 55.  
 <R\_StandardisePermutedLinearStatistic 42a> Referenced in 31a.  
 <R\_StandardisePermutedLinearStatistic Prototype 41c> Referenced in 23b, 42a.  
 <R\_Sums 95b> Referenced in 94c.  
 <R\_Sums Prototype 95a> Referenced in 23b, 95b.  
 <R\_ThreeTableSums 127b> Referenced in 117a.  
 <R\_ThreeTableSums Prototype 127a> Referenced in 23b, 127b.  
 <R\_TwoTableSums 122b> Referenced in 117a.  
 <R\_TwoTableSums Prototype 122a> Referenced in 23b, 122b.  
 <R\_unpack\_sym 149> Referenced in 139a.  
 <R\_unpack\_sym Prototype 148b> Referenced in 23b, 149.  
 <Setup Dimensions 33> Referenced in 32c, 40.  
 <Setup Dimensions 2d 45a> Referenced in 44, 51.  
 <Setup Linear Statistic 41a> Referenced in 40, 51.  
 <Setup Log-Factorials 52c> Referenced in 51.  
 <Setup maxstat Memory 75> Referenced in 73, 78.  
 <Setup maxstat Variables 74> Referenced in 73, 78.  
 <Setup Memory and Subsets in Blocks 36a> Referenced in 34.  
 <Setup mvtnorm Correlation 72a> Referenced in 70.  
 <Setup mvtnorm Memory 71> Referenced in 70.  
 <Setup Test Memory 56a> Referenced in 55, 57.  
 <Setup unordered maxstat Contrasts 79b> Referenced in 78.  
 <Setup Working Memory 52b> Referenced in 51.  
 <SimpleSums 94c> Referenced in 24a.  
 <start subset loop 94a> Referenced in 98a, 105, 108, 116b, 121b, 126, 131a.  
 <Sums Body 98a> Referenced in 96b, 97abc.  
 <Tables 117a> Referenced in 24a.  
 <Test Statistics 60a> Referenced in 24a.  
 <Tests 53b> Referenced in 24a.  
 <ThreeTableSums Body 131a> Referenced in 129bc, 130ab.  
 <TwoTableSums Body 126> Referenced in 124bc, 125ab.  
 <User Interface 31a> Referenced in 24a.  
 <User Interface Input 31b> Referenced in 32b, 34, 38c.  
 <Utils 131b> Referenced in 24a.  
 <vcov LinStatExpCov 10> Referenced in 3a.  
 <XfactorKronSums Body 108> Referenced in 106bc, 107ab.  
 <XfactorKronSums Permutation Body 112b> Referenced in 111c, 112a.

## Identifiers

B: [28c](#), 32c, 33, 34, 35a, 36a, 40, 44, 45a, 46a, 48, 49, 51, 52b, 73, 74, 78, 127b, 128b, 131a, 141abc, 142, 143, 144, 158a, 159, 160b, 161.  
 block: 3b, 4, 5a, 6, 8, 15, 16, 18, 20, [28b](#), [28d](#), 32ac, 33, 36ab, 38d, 40, 43b, 44, 45a, 50b, 127b, 128b, 131a, 132b, 133b, 134b, 135a, 155c.  
 blockTable: [28e](#), 40, 132b, 133b, 134b, 135a.  
 CovarianceInfluence\_SLOT: [22b](#), 155a, 158b, 159.  
 Covariance\_SLOT: [22b](#), 153b, 154a, 158b, 159.  
 C\_chisq\_pvalue: 55, [67c](#).  
 C\_colSums\_dweights\_dsubset: 114a, [115a](#).  
 C\_colSums\_dweights\_isubset: 114a, [116a](#).  
 C\_colSums\_ieweights\_dsubset: 114a, [115b](#).



C\_colSums\_iweights\_isubset: [114a](#), [115c](#).  
C\_CovarianceLinearStatistic: [38a](#), [47](#), [76b](#), [81a](#), [83](#), [84](#).  
C\_doPermute: [40](#), [137b](#).  
C\_doPermuteBlock: [40](#), [138b](#).  
C\_ExpectationLinearStatistic: [37a](#), [46c](#), [82b](#).  
C\_get\_B: [35a](#), [49](#), [74](#), [157a](#).  
C\_get\_Covariance: [38ab](#), [42a](#), [47](#), [48](#), [55](#), [57](#), [74](#), [154a](#), [160a](#).  
C\_get\_CovarianceInfluence: [36a](#), [47](#), [74](#), [155a](#), [160a](#).  
C\_get\_dimTable: [49](#), [156c](#), [157a](#).  
C\_get\_Expectation: [37a](#), [42a](#), [46c](#), [55](#), [57](#), [74](#), [153a](#), [160a](#).  
C\_get\_ExpectationInfluence: [36a](#), [49](#), [154c](#), [160a](#).  
C\_get\_ExpectationX: [36a](#), [49](#), [74](#), [154b](#).  
C\_get\_LinearStatistic: [35b](#), [48](#), [55](#), [57](#), [74](#), [152d](#), [160a](#).  
C\_get\_nresample: [42a](#), [55](#), [56a](#), [57](#), [59](#), [74](#), [157b](#).  
C\_get\_P: [35a](#), [42a](#), [49](#), [56a](#), [59](#), [74](#), [151c](#), [153b](#), [154a](#), [157b](#).  
C\_get\_PermutedLinearStatistic: [42a](#), [55](#), [57](#), [74](#), [157c](#).  
C\_get\_Q: [35a](#), [42a](#), [49](#), [56a](#), [74](#), [152a](#), [153b](#), [154a](#), [157b](#).  
C\_get\_Sumweights: [36a](#), [49](#), [156a](#).  
C\_get\_Table: [44](#), [49](#), [156b](#).  
C\_get\_TableBlock: [36a](#), [155c](#).  
C\_get\_tol: [42a](#), [55](#), [57](#), [74](#), [157d](#).  
C\_get\_Variance: [37c](#), [38b](#), [42a](#), [47](#), [48](#), [57](#), [74](#), [153b](#), [154a](#), [160a](#).  
C\_get\_VarianceInfluence: [36a](#), [47](#), [74](#), [155b](#), [160a](#).  
C\_get\_varonly: [34](#), [36a](#), [38b](#), [42a](#), [47](#), [48](#), [49](#), [56a](#), [57](#), [74](#), [152b](#), [154a](#).  
C\_get\_Xfactor: [49](#), [152c](#).  
C\_kronecker: [84](#), [142](#), [143](#).  
C\_kronecker\_sym: [83](#), [144](#).  
C\_KronSums\_dweights\_dsubset: [103a](#), [103b](#).  
C\_KronSums\_dweights\_isubset: [103a](#), [104c](#).  
C\_KronSums\_iweights\_dsubset: [103a](#), [104a](#).  
C\_KronSums\_iweights\_isubset: [103a](#), [104b](#).  
C\_KronSums\_Permutation\_dsubset: [110a](#), [110b](#).  
C\_KronSums\_Permutation\_isubset: [110a](#), [111a](#).  
C\_maxabsstand\_Covariance: [62b](#), [66](#).  
C\_maxabsstand\_Variance: [63](#), [66](#).  
C\_maxstand\_Covariance: [60b](#), [66](#).  
C\_maxstand\_Variance: [61a](#), [66](#).  
C\_maxtype: [57](#), [66](#), [76c](#).  
C\_maxtype\_pvalue: [57](#), [70](#).  
C\_minstand\_Covariance: [61b](#), [66](#).  
C\_minstand\_Variance: [62a](#), [66](#).  
C\_OneTableSums\_dweights\_dsubset: [119a](#), [120a](#).  
C\_OneTableSums\_dweights\_isubset: [119a](#), [121a](#).  
C\_OneTableSums\_iweights\_dsubset: [119a](#), [120b](#).  
C\_OneTableSums\_iweights\_isubset: [119a](#), [120c](#).  
C\_ordered\_Xfactor: [37b](#), [47](#), [59](#), [73](#).  
C\_order\_subset\_wrt\_block: [133b](#), [135a](#).  
C\_Permute: [137a](#), [137b](#), [138a](#).  
C\_PermuteBlock: [138a](#), [138b](#).  
C\_perm\_pvalue: [55](#), [57](#), [68](#), [77](#).  
C\_quadform: [55](#), [64c](#), [65](#), [76c](#).  
C\_setup\_subset: [133b](#), [134a](#), [136a](#).  
C\_setup\_subset\_block: [133b](#), [134b](#).  
C\_standardise: [42a](#), [67a](#).  
C\_Sums\_dweights\_dsubset: [96a](#), [96b](#).  
C\_Sums\_dweights\_isubset: [96a](#), [97c](#).  
C\_Sums\_iweights\_dsubset: [96a](#), [97a](#).  
C\_Sums\_iweights\_isubset: [96a](#), [97b](#).



C\_ThreeTableSums\_dweights\_dsubset: [128b](#), [129b](#).  
C\_ThreeTableSums\_dweights\_isubset: [128b](#), [130b](#).  
C\_ThreeTableSums\_iweights\_dsubset: [128b](#), [129c](#).  
C\_ThreeTableSums\_iweights\_isubset: [128b](#), [130a](#).  
C\_TwoTableSums\_dweights\_dsubset: [123b](#), [124b](#).  
C\_TwoTableSums\_dweights\_isubset: [123b](#), [125b](#).  
C\_TwoTableSums\_iweights\_dsubset: [123b](#), [124c](#).  
C\_TwoTableSums\_iweights\_isubset: [123b](#), [125a](#).  
C\_unordered\_Xfactor: [37b](#), [59](#), [78](#).  
C\_VarianceLinearStatistic: [37c](#), [47](#), [76b](#), [81a](#), [84](#).  
C\_XfactorKronSums\_dweights\_dsubset: [102](#), [106b](#).  
C\_XfactorKronSums\_dweights\_isubset: [102](#), [107b](#).  
C\_XfactorKronSums\_iweights\_dsubset: [102](#), [106c](#).  
C\_XfactorKronSums\_iweights\_isubset: [102](#), [107a](#).  
C\_XfactorKronSums\_Permutation\_dsubset: [110a](#), [111c](#).  
C\_XfactorKronSums\_Permutation\_isubset: [110a](#), [112a](#).  
dim\_SLOT: [22b](#), [151c](#), [152a](#), [158b](#), [159](#).  
DoCenter: [22b](#), [81d](#), [86a](#), [88a](#), [90](#), [93a](#), [100a](#), [113b](#).  
DoSymmetric: [22b](#), [81d](#), [88a](#), [93a](#).  
DoVarOnly: [22b](#), [37bc](#), [38a](#), [47](#).  
ExpectationInfluence\_SLOT: [22b](#), [154c](#), [158b](#), [159](#).  
ExpectationX\_SLOT: [22b](#), [154b](#), [158b](#), [159](#).  
Expectation\_SLOT: [22b](#), [153a](#), [158b](#), [159](#).  
GE: [22a](#), [55](#), [57](#).  
HAS\_WEIGHTS: [26d](#), [26e](#), [98a](#), [105](#), [108](#), [116b](#), [121b](#), [126](#), [131a](#).  
LE: [22a](#), [57](#).  
LECV: [41bc](#), [42a](#), [55](#), [56a](#), [57](#), [58](#), [59](#), [72b](#), [74](#), [151b](#), [151c](#), [152abcd](#), [153ab](#), [154abc](#), [155abc](#), [156abc](#), [157abcd](#).  
LinearStatistic\_SLOT: [22b](#), [152d](#), [158b](#), [159](#).  
mPQB: [38b](#), [40](#), [48](#), [51](#), [56a](#), [74](#), [76a](#), [80b](#), [82b](#), [83](#), [84](#), [108](#), [112b](#), [122b](#), [127b](#), [131a](#), [141a](#), [159](#).  
N: [5ab](#), [6](#), [8](#), [16](#), [24b](#), [24c](#), [35ab](#), [36ab](#), [37abc](#), [38a](#), [40](#), [44](#), [70](#), [81d](#), [85b](#), [86a](#), [87a](#), [88a](#), [89a](#), [90](#), [92a](#), [93ab](#), [94a](#), [95b](#), [96a](#), [98a](#), [100a](#), [102](#), [103a](#), [105](#), [108](#), [109b](#), [110a](#), [111b](#), [112b](#), [113b](#), [114a](#), [116b](#), [118a](#), [119a](#), [122b](#), [123b](#), [127b](#), [128b](#), [132b](#), [133b](#), [134ab](#), [135a](#), [136a](#), [145a](#).  
NCOL: [12](#), [33](#), [45a](#), [64c](#), [85b](#), [87a](#), [100a](#), [109b](#), [113b](#), [132b](#), [139c](#), [142](#).  
NLEVELS: [33](#), [45a](#), [118a](#), [122b](#), [127b](#), [132b](#), [140a](#).  
NROW: [6](#), [8](#), [9ab](#), [14](#), [35a](#), [40](#), [46c](#), [47](#), [64c](#), [139b](#), [140a](#), [142](#), [150c](#).  
Nsubset: [27c](#), [36b](#), [40](#), [44](#), [81d](#), [85b](#), [86a](#), [87a](#), [88a](#), [89a](#), [90](#), [92a](#), [93ab](#), [94ab](#), [95b](#), [96a](#), [98a](#), [100a](#), [102](#), [103a](#), [109b](#), [110a](#), [111b](#), [112b](#), [113b](#), [114a](#), [118a](#), [119a](#), [122b](#), [123b](#), [127b](#), [128b](#), [137ab](#), [138b](#).  
offset: [27d](#), [34](#), [36b](#), [37abc](#), [38a](#), [81d](#), [86a](#), [88a](#), [90](#), [93ab](#), [96a](#), [102](#), [103a](#), [110a](#), [111b](#), [112b](#), [114a](#), [119a](#), [123b](#), [128b](#).  
Offset0: [22b](#), [35b](#), [36a](#), [40](#), [44](#), [46c](#), [47](#), [85b](#), [87a](#), [89a](#), [92a](#), [95b](#), [100a](#), [109b](#), [113b](#), [118a](#), [122b](#), [127b](#), [132b](#), [136a](#).  
P: [14](#), [25a](#), [32c](#), [33](#), [35ab](#), [36a](#), [37ac](#), [38ab](#), [40](#), [44](#), [45ab](#), [46c](#), [47](#), [48](#), [49](#), [51](#), [55](#), [56a](#), [57](#), [59](#), [73](#), [74](#), [75](#), [76a](#), [78](#), [79ab](#), [80ab](#), [81d](#), [82b](#), [83](#), [84](#), [88b](#), [89a](#), [90](#), [91](#), [92a](#), [93a](#), [99](#), [100a](#), [102](#), [103a](#), [105](#), [108](#), [109ab](#), [110a](#), [111b](#), [112b](#), [113b](#), [114a](#), [116b](#), [118a](#), [119a](#), [121b](#), [122b](#), [123b](#), [126](#), [127b](#), [128b](#), [131a](#), [140b](#), [141a](#), [145a](#), [158a](#), [159](#).  
PermutedLinearStatistic\_SLOT: [22b](#), [157bc](#), [158b](#), [159](#).  
Power1: [22b](#), [86a](#), [90](#), [113b](#).  
Power2: [22b](#), [88a](#), [93a](#).  
PP12: [36a](#), [47](#), [49](#), [55](#), [83](#), [93a](#), [140b](#), [159](#), [160a](#).  
Q: [14](#), [25e](#), [32c](#), [33](#), [35ab](#), [37abc](#), [38ab](#), [40](#), [44](#), [45ab](#), [46c](#), [47](#), [48](#), [49](#), [51](#), [55](#), [56a](#), [57](#), [73](#), [74](#), [75](#), [76abc](#), [78](#), [80ab](#), [81ad](#), [82b](#), [83](#), [84](#), [85b](#), [86a](#), [87a](#), [88a](#), [100a](#), [102](#), [103a](#), [105](#), [108](#), [109b](#), [110a](#), [111b](#), [112b](#), [122b](#), [123b](#), [126](#), [127b](#), [128b](#), [131a](#), [141a](#), [158a](#), [159](#), [160a](#).  
RC\_colSums: [86a](#), [88a](#), [90](#), [93a](#), [113bc](#), [114a](#).  
RC\_CovarianceInfluence: [37b](#), [47](#), [87ab](#), [88a](#).  
RC\_CovarianceX: [37c](#), [38a](#), [47](#), [92ab](#), [93a](#).  
RC\_ExpectationCovarianceStatistic: [32c](#), [34](#), [48](#).  
RC\_ExpectationInfluence: [37a](#), [46c](#), [85bc](#), [86a](#).  
RC\_ExpectationX: [37a](#), [46c](#), [89ab](#), [90](#).  
RC\_init\_LECV\_1d: [32c](#), [160b](#).  
RC\_init\_LECV\_2d: [44](#), [161](#).  
RC\_KronSums: [81d](#), [88a](#), [93a](#), [100ab](#), [101a](#).

RC\_KronSums\_Permutation: 40, 109bc, [110a](#).  
 RC\_LinearStatistic: 35b, 81c, [81d](#).  
 RC\_OneTableSums: 36a, 40, 90, 118ab, [119a](#).  
 RC\_order\_subset\_wrt\_block: 36a, 40, 132b, 133a, [133b](#).  
 RC\_setup\_subset: 40, 135b, [136a](#).  
 RC\_Sums: 36ab, 85b, 87a, 95bc, [96a](#), 132b, 136a.  
 RC\_ThreeTableSums: 44, 127b, 128a, [128b](#).  
 RC\_TwoTableSums: 44, 122b, 123a, [123b](#).  
 R\_colSums: 113a, [113b](#), 164, 165.  
 R\_CovarianceInfluence: 86b, [87a](#), 164, 165.  
 R\_CovarianceX: 91, [92a](#), 164, 165.  
 R\_ExpectationCovarianceStatistic: 6, 32ab, [32c](#), 164, 165.  
 R\_ExpectationCovarianceStatistic\_2d: 8, 43ab, [44](#), 164, 165.  
 R\_ExpectationInfluence: 85a, [85b](#), 87a, 164, 165.  
 R\_ExpectationX: 88b, [89a](#), 92a, 164, 165.  
 R\_KronSums: 99, [100a](#), 164, 165.  
 R\_KronSums\_Permutation: 109a, [109b](#), 164, 165.  
 R\_MPinv\_sym: 145b, 146a, [146b](#), 164, 165.  
 R\_OneTableSums: 16, 117b, [118a](#), 132b, 164, 165.  
 R\_order\_subset\_wrt\_block: 132a, [132b](#), 164, 165.  
 R\_pack\_sym: 150ab, [150c](#), 164, 165.  
 R\_PermutedLinearStatistic: 6, 38cd, [40](#), 164, 165.  
 R\_PermutedLinearStatistic\_2d: 8, 50ab, [51](#), 52a, 164, 165.  
 R\_quadform: 64ab, [64c](#), 164, 165.  
 R\_Sums: 95a, [95b](#), 164, 165.  
 R\_ThreeTableSums: 16, 127a, [127b](#), 164, 165.  
 R\_TwoTableSums: 16, 122a, [122b](#), 164, 165.  
 R\_unpack\_sym: 10, 148ab, [149](#), 164, 165.  
 S: [22a](#), 37b, 38b, 47, 48, 60b, 61b, 62b, 65, 67a, 71, 72a, 76a, 80b, 93a, 105, 144, 145a, 147, 153b.  
 StandardisedPermutedLinearStatistic\_SLOT: [22b](#), 158b, 159.  
 subset: 3b, 4, 5ab, 6, 8, 15, 16, 18, 20, [27b](#), [27e](#), [28a](#), 32ac, 34, 35b, 36ab, 38d, 40, 43b, 44, 46c, 47, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 94b, 95b, 96a, 100a, 102, 103a, 109b, 110a, 111b, 112b, 113b, 114a, 118a, 119a, 122b, 123b, 127b, 128b, 132b, 133b, 135a, 136a, 137ab, 138ab.  
 sumweights: [27a](#), 34, 36ab, 37abc, 38a, 46bc, 47, 49, 51, 52b, 53a, 74, 75, 76b, 81a, 83, 84, 85b, 86a, 87a, 88a, 136a, 156a.  
 Sumweights\_SLOT: [22b](#), 156a, 157a, 158b, 159, 160b.  
 TableBlock\_SLOT: [22b](#), 36a, 155c, 157a, 158b, 159, 160b.  
 Table\_SLOT: [22b](#), 156bc, 158b, 159, 161.  
 tol\_SLOT: [22b](#), 157d, 158b, 159.  
 VarianceInfluence\_SLOT: [22b](#), 155b, 158b, 159.  
 Variance\_SLOT: [22b](#), 153b, 158b, 159.  
 varonly\_SLOT: [22b](#), 152b, 158b, 159.  
 weights: 3b, 4, 5a, 6, 8, 15, 16, 18, 20, [26c](#), 26de, 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 52a, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93ab, 95b, 96a, 100a, 102, 103a, 113b, 114a, 118a, 119a, 122b, 123b, 127b, 128b, 132b, 136a.  
 weights,: 4, 6, 8, 16, 20, [26d](#), [26e](#), 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 81d, 85b, 86a, 87a, 88a, 89a, 90, 92a, 93a, 95b, 100a, 113b, 118a, 122b, 127b, 132b, 136a.  
 x: 8, 14, 18, 22a, [24d](#), [25b](#), [25c](#), 32ac, 33, 35ab, 37ac, 38ad, 40, 43b, 44, 45ab, 46c, 47, 50b, 51, 81d, 89a, 90, 92a, 93a, 100a, 101a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 113b, 114a, 116b, 118a, 119a, 121b, 122b, 123b, 126, 127b, 128b, 131a, 139bc, 140a, 145ab, 146ab, 147, 148ab, 149, 150abc.  
 Xfactor\_SLOT: [22b](#), 152c, 158b, 159.  
 y: 14, 22a, [25d](#), [26a](#), [26b](#), 32ac, 33, 35b, 37ab, 38d, 40, 43b, 44, 45ab, 46c, 47, 50b, 81d, 85b, 86a, 87a, 88a, 100a, 102, 103a, 105, 108, 109b, 110a, 111b, 112b, 122b, 123b, 126, 127b, 128b, 131a, 132b, 143, 144.

# Bibliography

Helmut Strasser and Christian Weber. On the asymptotic theory of permutation statistics. *Mathematical Methods of Statistics*, 8:220–250, 1999. preprint available from [http://epub.wu-wien.ac.at/dyn/openURL?id=oai:epub.wu-wien.ac.at:epub-wu-01\\_94c](http://epub.wu-wien.ac.at/dyn/openURL?id=oai:epub.wu-wien.ac.at:epub-wu-01_94c). 1