

# The **libcoin** Package

Torsten Hothorn  
Universität Zürich

June 4, 2026

# 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                  | 17         |
| <b>3</b> | <b>C Code</b>                 | <b>20</b>  |
| 3.1      | Header and Source Files       | 20         |
| 3.2      | Variables                     | 23         |
| 3.2.1    | Example Data and Code         | 27         |
| 3.3      | Conventions                   | 29         |
| 3.4      | C User Interface              | 29         |
| 3.4.1    | One-Dimensional Case (“1d”)   | 29         |
| 3.4.2    | Two-Dimensional Case (“2d”)   | 38         |
| 3.5      | Tests                         | 49         |
| 3.6      | Test Statistics               | 56         |
| 3.7      | Linear Statistics             | 75         |
| 3.8      | Expectation and Covariance    | 76         |
| 3.8.1    | Linear Statistic              | 76         |
| 3.8.2    | Influence                     | 78         |
| 3.8.3    | X                             | 82         |
| 3.9      | Computing Sums                | 85         |
| 3.9.1    | Simple Sums                   | 86         |
| 3.9.2    | Kronecker Sums                | 90         |
| 3.9.3    | Column Sums                   | 102        |
| 3.9.4    | Tables                        | 106        |
| 3.10     | Utilities                     | 118        |
| 3.10.1   | Blocks                        | 118        |
| 3.10.2   | Permutation Helpers           | 123        |
| 3.10.3   | Other Utils                   | 125        |
| 3.11     | Memory                        | 137        |
| <b>4</b> | <b>Package Infrastructure</b> | <b>147</b> |

# Licence

Copyright (C) 2016-2026 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/>.

# 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  $\boldsymbol{\mu}(\mathcal{A}) \in \mathbb{R}^{PQ}$  and covariance  $\boldsymbol{\Sigma}(\mathcal{A}) \in \mathbb{R}^{PQ \times PQ}$  of  $\mathbf{T}$  under  $H_0$  given all permutations  $\sigma \in S(\mathcal{A})$  of the responses are derived by [Strasser and Weber \(1999\)](#):

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

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

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

with corresponding  $Q \times Q$  covariance matrix

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

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

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

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

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

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

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

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

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

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

```
"R/libcoin.R" 3a≡  
  ⟨ R Header 151a ⟩  
  ⟨ LinStatExpCov 4 ⟩  
  ⟨ LinStatExpCov1d 6 ⟩  
  ⟨ LinStatExpCov2d 8 ⟩  
  ⟨ vcov LinStatExpCov 10 ⟩  
  ⟨ doTest 12 ⟩  
  ⟨ Contrasts 14 ⟩  
  ◇
```

The **libcoin** package implements two R 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, 17.

Uses: block 26bd, subset 25cf, 26a, weights 24ef, 25a.

```

⟨ LinStatExpCov 4 ⟩ ≡
  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))
      .LinStatExpCov1d(X = X, Y = Y,
        weights = weights, subset = subset,
        block = block, checkNAs = checkNAs,
        varonly = varonly, nresample = nresample,
        standardise = standardise, tol = tol)
    else if (!is.null(ix) && !is.null(iy))
      .LinStatExpCov2d(X = X, Y = Y, ix = ix, iy = iy,
        weights = weights, subset = subset,
        block = block, checkNAs = checkNAs,
        varonly = varonly, nresample = nresample,
        standardise = standardise, tol = tol)
    else
      stop("incorrect call to ", sQuote("LinStatExpCov()"))
  }
  ◇

```

Fragment referenced in 3a.

Uses: block 26bd, subset 25cf, 26a, weights 24ef, 25a.

### 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 **X** and **Y** with  $N$  rows as well as  $P$  and  $Q$  columns, respectively. The special case of a dummy matrix **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](#), [15b](#).

Uses: `block` [26bd](#), `N` [23ab](#), `subset` [25cf](#), [26a](#), `weights` [24ef](#), [25a](#).

Missing values are only allowed in `X` and `Y`, all other vectors must not contain `NA`s. 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 <- seq_len(N)[-which(ms)]
    }
  }
  ◇

```

Fragment referenced in [6](#).

Uses: `N` [23ab](#), `subset` [25cf](#), [26a](#).

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.



```

⟨ LinStatExpCov1d 6 ⟩ ≡
.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") <- seq_len(rg[2])
    }
  }

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

  ⟨ Check weights, subset, block 5a ⟩

  if (checkNAs) {
    ⟨ Handle Missing Values 5b ⟩
  }

  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 26bd, N 23ab, NROW 126a, R\_ExpectationCovarianceStatistic 30b, R\_PermutedLinearStatistic 36, subset 25cf, 26a, weights 24ef, 25a.

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  $X$  is now of dimension  $(L_x + 1) \times P$  and the matrix  $Y$  of dimension  $(L_y + 1) \times Q$ . The combination of  $X$  and  $\mathbf{ix}$  means that the  $i$ th observation corresponds to the row  $X[\mathbf{ix}[i] + 1, ]$ . 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  $X$  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 26bd](#), [N 23ab](#), [NROW 126a](#), [R\\_ExpectationCovarianceStatistic\\_2d 40a](#), [R\\_PermutedLinearStatistic\\_2d 47](#),  
[subset 25cf](#), [26a](#), [weights 24ef](#), [25a](#), [x 23cef](#).

`ix` and `iy` can be factors but without any missing values

```

< Check ix 9a > ≡
  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") <- seq_len(rg[2])
  } else {
    ## lev can be data.frame (see inum::inum)
    lev <- attr(ix, "levels")
    if (!is.vector(lev)) lev <- seq_len(NROW(lev))
    attr(ix, "levels") <- lev
    if (checkNAs) stopifnot(!anyNA(ix))
  }
  ◇

```

Fragment referenced in [8](#), [15b](#).

Uses: [NROW 126a](#).

```

< Check iy 9b > ≡
  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") <- seq_len(rg[2])
  } else {
    ## lev can be data.frame (see inum::inum)
    lev <- attr(iy, "levels")
    if (!is.vector(lev)) lev <- seq_len(NROW(lev))
    attr(iy, "levels") <- lev
    if (checkNAs) stopifnot(!anyNA(iy))
  }
  ◇

```

Fragment referenced in [8](#), [15b](#).

Uses: [NROW 126a](#).

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.

```
<vcov LinStatExpCov 10> ≡
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` [135](#).

```
> 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](#), [18](#).

```

⟨ 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 [126b](#).

```

> ### 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"`.



$\langle \text{Contrasts 14} \rangle \equiv$

```
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.int(seq_len(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 [126a](#), P [23d](#), Q [24b](#), x [23cef](#), y [24acd](#).

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

```

Fragment referenced in 15b, 19.

Uses: block 26bd, subset 25cf, 26a, weights 24ef, 25a.

```

"R/ctabs.R" 15b≡
  ⟨ R Header 151a ⟩
  ctab <-
  function⟨ ctabs Prototype 15a ⟩
  {
    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)
      if (length(block) == 0)
        .Call(R_OneTableSums, ix, weights, subset)
      else
        .Call(R_TwoTableSums, ix, block, weights, subset)[, -1, drop = FALSE]
    else if (length(block) == 0)
      .Call(R_TwoTableSums, ix, iy, weights, subset)
    else
      .Call(R_ThreeTableSums, ix, iy, block, weights, subset)
  }
  ◇

```

Uses: block 26bd, N 23ab, R\_OneTableSums 107b, R\_ThreeTableSums 115a, R\_TwoTableSums 111a, subset 25cf, 26a, weights 24ef, 25a.

```

> t1 <- ctab(ix = ix, iy = iy)
> t2 <- xtab(~ ix + iy)
> max(abs(t1[-1, -1] - t2))

```

```
[1] 0
```



## 2.2 Manual Pages

```
"man/LinStatExpCov.Rd" 17≡
< Rd Header 151b>
\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{

  This function implements the permutation test framework by
  \bibcitet{libcoin::Strasser+Weber:1999}, see also
  \bibcitet{libcoin::Hothorn+Hornik+vandeWiel+Zeileis:2006} and
  \bibcitet{libcoin::Hothorn+Hornik+vandeWiel+Zeileis:2008}.

  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, i.e., 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 or \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{\bibshow{*}}
\examples{
  wilcox.test(Ozone ~ Month, data = airquality, subset = Month \%in\% c(5, 8),
    exact = FALSE, correct = FALSE)
}
```

```

"man/doTest.Rd" 18≡
< Rd Header 151b>
\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 \code{\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 (\code{lower} is \code{FALSE})
    or 1 - p-value (\code{lower} is \code{TRUE}) shall be returned.}
  \item{log}{a logical, if \code{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 \code{TRUE} maximally selected statistics assume
    that the cutpoints are ordered.}
  \item{maxselect}{a logical, if \code{TRUE} maximally selected statistics are
    computed. This requires that \code{X} was an implicitly defined design
    matrix in \code{\link{LinStatExpCov}}.}
  \item{pargs}{arguments as in \code{\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}
◇

```

```

"man/ctabs.Rd" 19≡
< Rd Header 151b>
\name{ctabs}
\alias{ctabs}
\title{
  Cross Tabulation
}
\description{
  Efficient weighted cross tabulation of two factors and a block
}
\usage{
ctabs< ctabs Prototype 15a>
}
\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 case 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 26bd, subset 25cf, 26a, weights 24ef, 25a.

# 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

```
"src/libcoin_internal.h" 20a≡  
  ⟨ C Header 152a ⟩  
  ⟨ R Includes 20b ⟩  
  ⟨ C Macros 20c ⟩  
  ⟨ C Global Variables 21a ⟩  
  ◇
```

These includes provide some R infrastructure at C level.

```
⟨ R Includes 20b ⟩ ≡  
  #define USE_FC_LEN_T  
  #include <float.h>          /* for DBL_MIN */  
  #include <R.h>  
  #include <Rinternals.h>  
  #include <Rversion.h>      /* for R_VERSION */  
  #include <R_ext/Lapack.h> /* for dspev */  
  #ifndef FCONE  
  # define FCONE  
  #endif  
  ◇
```

Fragment referenced in 20a.

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  $\geq$ .

```
⟨ C Macros 20c ⟩ ≡  
  
  #define S(i, j, n) ((i) >= (j) ? (n) * (j) + (i) - (j) * ((j) + 1) / 2 : (n) * (i) + (j) - (i) * ((i) + 1)  
  #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 20a.

Defines: **GE** 50, 53, **LE** 53, **S** 34b, 35a, 43, 44, 56b, 57b, 58b, 61a, 62a, 66, 67a, 70b, 74b, 85b, 96, 130ab, 133, 139a.

Uses: **x** 23cef, **y** 24acd.

```

< C Global Variables 21a > ≡
#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 20a.

Defines: ALTERNATIVE\_greater 61b, 64, 67a, ALTERNATIVE\_less 53, 61b, 64, 67a, ALTERNATIVE\_twosided 61b, 64, 67a, 71b, CovarianceInfluence\_SLOT 140b, 143, 144, Covariance\_SLOT 139ab, 143, 144, dim\_SLOT 137cd, 143, 144, DoCenter 75d, 79c, 82a, 83b, 85b, 91b, 103b, DoSymmetric 75d, 82a, 85b, DoVarOnly 34bcd, 43, ExpectationInfluence\_SLOT 140a, 143, 144, ExpectationX\_SLOT 139c, 143, 144, Expectation\_SLOT 138d, 143, 144, LinearStatistic\_SLOT 138c, 143, 144, Offset0 32b, 33a, 36, 40a, 42b, 43, 79a, 81a, 82c, 84b, 87b, 91b, 99b, 103b, 107b, 111a, 115a, 119b, 123a, PermutedLinearStatistic\_SLOT 142ab, 143, 144, Power1 79c, 83b, 103b, Power2 82a, 85b, StandardisedPermutedLinearStatistic\_SLOT 143, 144, Sumweights\_SLOT 141ad, 143, 144, 145b, TableBlock\_SLOT 33a, 140d, 141d, 143, 144, 145b, Table\_SLOT 141bc, 143, 144, 146, TESTSTAT\_maximum 68, 70ab, 71ab, 72, 74b, 75a, TESTSTAT\_quadratic 70a, tol\_SLOT 142c, 143, 144, VarianceInfluence\_SLOT 140c, 143, 144, Variance\_SLOT 139a, 143, 144, varonly\_SLOT 138a, 143, 144, Xfactor\_SLOT 138b, 143, 144.

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!).

```

"src/libcoin.h" 21b≡
< C Header 152a >
#include "libcoin_internal.h"
< Function Prototypes 22a >
◇

```



⟨ *Function Prototypes 22a* ⟩ ≡

```
extern ⟨ R_ExpectationCovarianceStatistic Prototype 29c ⟩;
extern ⟨ R_PermutedLinearStatistic Prototype 35b ⟩;
extern ⟨ R_StandardisePermutedLinearStatistic Prototype 37b ⟩;
extern ⟨ R_ExpectationCovarianceStatistic_2d Prototype 39a ⟩;
extern ⟨ R_PermutedLinearStatistic_2d Prototype 46a ⟩;
extern ⟨ R_QuadraticTest Prototype 49b ⟩;
extern ⟨ R_MaximumTest Prototype 51b ⟩;
extern ⟨ R_MaximallySelectedTest Prototype 54a ⟩;
extern ⟨ R_ExpectationInfluence Prototype 78b ⟩;
extern ⟨ R_CovarianceInfluence Prototype 80 ⟩;
extern ⟨ R_ExpectationX Prototype 82b ⟩;
extern ⟨ R_CovarianceX Prototype 84a ⟩;
extern ⟨ R_Sums Prototype 87a ⟩;
extern ⟨ R_KronSums Prototype 91a ⟩;
extern ⟨ R_KronSums_Permutation Prototype 99a ⟩;
extern ⟨ R_colSums Prototype 103a ⟩;
extern ⟨ R_OneTableSums Prototype 107a ⟩;
extern ⟨ R_TwoTableSums Prototype 110b ⟩;
extern ⟨ R_ThreeTableSums Prototype 114b ⟩;
extern ⟨ R_order_subset_wrt_block Prototype 119a ⟩;
extern ⟨ R_quadform Prototype 59b ⟩;
extern ⟨ R_kronecker Prototype 128b ⟩;
extern ⟨ R_MPinv_sym Prototype 131a ⟩;
extern ⟨ R_unpack_sym Prototype 134a ⟩;
extern ⟨ R_pack_sym Prototype 136a ⟩;
◇
```

Fragment referenced in [21b](#).

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

"src/libcoin.c" 22b≡

```
⟨ C Header 152a ⟩
#include "libcoin_internal.h"
#include <R_ext/stats_stubs.h> /* for S_rcont2 */
#include <mvtnormAPI.h> /* for calling mvtnorm */
⟨ Function Definitions 22c ⟩
◇
```

⟨ *Function Definitions 22c* ⟩ ≡

```
⟨ MoreUtils 125b ⟩
⟨ Memory 137a ⟩
⟨ P-Values 62b ⟩
⟨ KronSums 90b ⟩
⟨ colSums 102d ⟩
⟨ SimpleSums 86d ⟩
⟨ Tables 106b ⟩
⟨ Utils 118b ⟩
⟨ LinearStatistics 75b ⟩
⟨ Permutations 123b ⟩
⟨ ExpectationCovariances 76a ⟩
⟨ Test Statistics 56a ⟩
⟨ User Interface 29a ⟩
⟨ 2d User Interface 38b ⟩
⟨ Tests 49a ⟩
◇
```

Fragment referenced in [22b](#).

## 3.2 Variables

$N$  is the number of observations

```

< R N Input 23a > ≡
    SEXP N,
    ◇

```

Fragment referenced in 87a.

Defines: N 5ab, 6, 8, 15b, 23b, 32ab, 33ab, 34abcd, 36, 40a, 65, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86ab, 87b, 88a, 90a, 91b, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 103b, 104a, 106a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 120ab, 121, 122a, 123a, 130b.

which at C level is represented as `R_xlen_t` to allow for  $N > \text{INT\_MAX}$

```

< C integer N Input 23b > ≡
    R_xlen_t N
    ◇

```

Fragment referenced in 23ef, 31, 36, 40a, 75c, 79ab, 81ab, 82c, 83a, 84b, 85a, 87c, 88b, 89abc, 91b, 92b, 99b, 100a, 103b, 107b, 111a, 115a, 119bc, 120b, 121, 122b.

Defines: N 5ab, 6, 8, 15b, 23a, 32ab, 33ab, 34abcd, 36, 40a, 65, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86ab, 87b, 88a, 90a, 91b, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 103b, 104a, 106a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 120ab, 121, 122a, 123a, 130b.

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

```

< R x Input 23c > ≡
    SEXP x,
    ◇

```

Fragment referenced in 29b, 38c, 46a, 75c, 82b, 83a, 84a, 85a, 91a, 92b, 99a, 100a, 103a, 107a, 110b, 114b.

Defines: x 8, 14, 17, 20c, 23ef, 30abc, 32ab, 34acd, 35c, 36, 39b, 40ab, 41a, 42b, 43, 46b, 47, 75d, 82c, 83b, 84b, 85b, 91b, 92a, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 103b, 104a, 106a, 107b, 108a, 110a, 111a, 112a, 114a, 115a, 116a, 118a, 126ab, 127a, 130b, 131ab, 132, 133, 134ab, 135, 136abc, 154a, 159ab, 160bd.

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

```

< C integer P Input 23d > ≡
    int P
    ◇

```

Fragment referenced in 23ef, 31, 75c, 76b, 77, 78a, 83a, 85a, 92b, 100a, 145b, 146.

Defines: P 14, 30bc, 32ab, 33a, 34acd, 35a, 36, 40ab, 41a, 42b, 43, 44, 45, 47, 50, 51a, 53, 55, 68, 69, 70ab, 72, 73ab, 74ab, 75d, 76b, 77, 78a, 82bc, 83b, 84ab, 85b, 91ab, 93ab, 96, 98b, 99ab, 100b, 101c, 102c, 103b, 104a, 106a, 107b, 108a, 110a, 111a, 112a, 114a, 115a, 116a, 118a, 127b, 128a, 130b, 142d, 144.

```

< C real x Input 23e > ≡
    double *x,
    < C integer N Input 23b >,
    < C integer P Input 23d >,
    ◇

```

Fragment referenced in 92c, 101ab, 104b, 130b.

Defines: x 8, 14, 17, 20c, 23cf, 30abc, 32ab, 34acd, 35c, 36, 39b, 40ab, 41a, 42b, 43, 46b, 47, 75d, 82c, 83b, 84b, 85b, 91b, 92a, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 103b, 104a, 106a, 107b, 108a, 110a, 111a, 112a, 114a, 115a, 116a, 118a, 126ab, 127a, 130b, 131ab, 132, 133, 134ab, 135, 136abc, 154a, 159ab, 160bd.

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

$\langle C \text{ integer } x \text{ Input 23f} \rangle \equiv$   
`int *x,`  
 $\langle C \text{ integer } N \text{ Input 23b} \rangle,$   
 $\langle C \text{ integer } P \text{ Input 23d} \rangle,$   
 $\diamond$

Fragment referenced in 97a, 102ab, 108b, 112b, 116b.

Defines:  $x$  8, 14, 17, 20c, 23ce, 30abc, 32ab, 34acd, 35c, 36, 39b, 40ab, 41a, 42b, 43, 46b, 47, 75d, 82c, 83b, 84b, 85b, 91b, 92a, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 103b, 104a, 106a, 107b, 108a, 110a, 111a, 112a, 114a, 115a, 116a, 118a, 126ab, 127a, 130b, 131ab, 132, 133, 134ab, 135, 136abc, 154a, 159ab, 160bd.

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

$\langle R \text{ y Input 24a} \rangle \equiv$   
`SEXP y,`  
 $\diamond$

Fragment referenced in 29b, 38c, 46a, 78b, 79b, 80, 81b, 91a, 99a, 110b, 114b, 119a.

Defines:  $y$  14, 20c, 24cd, 30abc, 32b, 34ab, 35c, 36, 39b, 40ab, 41a, 42b, 43, 46b, 75d, 79ac, 81a, 82a, 91b, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 111a, 112a, 114a, 115a, 116a, 118a, 119b, 129b, 130a, 159ab, 160bd.

$\langle C \text{ integer } Q \text{ Input 24b} \rangle \equiv$   
`int Q`  
 $\diamond$

Fragment referenced in 24cd, 31, 76b, 77, 78a, 79ab, 81ab, 91b, 99b, 145b, 146.

Defines:  $Q$  14, 30bc, 32ab, 34abcd, 35a, 36, 40ab, 41a, 42b, 43, 44, 45, 47, 50, 51a, 53, 68, 69, 70ab, 71ab, 72, 74ab, 75ad, 76b, 77, 78a, 79ac, 81a, 82a, 91b, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 111a, 112a, 114a, 115a, 116a, 118a, 128a, 142d, 144, 145a.

$\langle C \text{ real } y \text{ Input 24c} \rangle \equiv$   
`double *y,`  
 $\langle C \text{ integer } Q \text{ Input 24b} \rangle,$   
 $\diamond$

Fragment referenced in 75c, 92bc, 97a, 100a, 101ab, 102ab.

Defines:  $y$  14, 20c, 24ad, 30abc, 32b, 34ab, 35c, 36, 39b, 40ab, 41a, 42b, 43, 46b, 75d, 79ac, 81a, 82a, 91b, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 111a, 112a, 114a, 115a, 116a, 118a, 119b, 129b, 130a, 159ab, 160bd.

or a factor at  $Q$  levels

$\langle C \text{ integer } y \text{ Input 24d} \rangle \equiv$   
`int *y,`  
 $\langle C \text{ integer } Q \text{ Input 24b} \rangle,$   
 $\diamond$

Fragment referenced in 112b, 116b.

Defines:  $y$  14, 20c, 24ac, 30abc, 32b, 34ab, 35c, 36, 39b, 40ab, 41a, 42b, 43, 46b, 75d, 79ac, 81a, 82a, 91b, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 111a, 112a, 114a, 115a, 116a, 118a, 119b, 129b, 130a, 159ab, 160bd.

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

$\langle R \text{ weights Input 24e} \rangle \equiv$   
`SEXP weights`  
 $\diamond$

Fragment referenced in 29b, 38c, 75c, 78b, 79b, 80, 81b, 82b, 83a, 84a, 85a, 87ac, 91ac, 103ac, 107ac, 110b, 111b, 114b, 115b, 119a, 122b.

Defines: **weights** 3b, 4, 5a, 6, 8, 15ab, 17, 19, 24f, 25a, 30ab, 32b, 33b, 34abcd, 35c, 36, 39b, 40a, 48a, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86a, 87b, 88a, 91b, 93ab, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 123a, 160bd.

can be constant one ( $\text{XLENGTH}(\text{weights}) == 0$  or  $\text{weights} = \text{integer}(0)$ ) or integer-valued, with  $\text{HAS\_WEIGHTS} == 0$  in the former case

```

⟨ C integer weights Input 24f ⟩ ≡
    int *weights,
    int HAS_WEIGHTS,
    ◇

```

Fragment referenced in 89ab, 94bc, 97cd, 105ab, 109ab, 113ab, 117ab.

Defines: **HAS\_WEIGHTS** 25a, 90a, 96, 98b, 106a, 110a, 114a, 118a, **weights** 3b, 4, 5a, 6, 8, 15ab, 17, 19, 24e, 25a, 30ab, 32b, 33b, 34abcd, 35c, 36, 39b, 40a, 48a, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86a, 87b, 88a, 91b, 93ab, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 123a, 160bd.

Case weights larger than  $\text{INT\_MAX}$  are stored as double

```

⟨ C real weights Input 25a ⟩ ≡
    double *weights,
    int HAS_WEIGHTS,
    ◇

```

Fragment referenced in 88b, 89c, 94a, 95, 97b, 98a, 104d, 105c, 108d, 109c, 112d, 113c, 116d, 117c.

Defines: **HAS\_WEIGHTS** 24f, 90a, 96, 98b, 106a, 110a, 114a, 118a, **weights** 3b, 4, 5a, 6, 8, 15ab, 17, 19, 24ef, 30ab, 32b, 33b, 34abcd, 35c, 36, 39b, 40a, 48a, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86a, 87b, 88a, 91b, 93ab, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 123a, 160bd.

The sum of all case weights is a double

```

⟨ C sumweights Input 25b ⟩ ≡
    double sumweights
    ◇

```

Fragment referenced in 77, 78a, 79b, 81b.

Defines: **sumweights** 31, 33ab, 34abcd, 42ab, 43, 45, 47, 48bd, 69, 70a, 71a, 75a, 77, 78a, 79ac, 81a, 82a, 123a, 141a.

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

```

⟨ R subset Input 25c ⟩ ≡
    SEXP subset
    ◇

```

Fragment referenced in 29b, 38c, 75c, 78b, 79b, 80, 81b, 82b, 83a, 84a, 85a, 87ac, 91ac, 99a, 100a, 103ac, 107ac, 110b, 111b, 114b, 115b, 119ac, 122ab.

Defines: **subset** 3b, 4, 5ab, 6, 8, 15ab, 17, 19, 25f, 26a, 30ab, 31, 32b, 33ab, 35c, 36, 39b, 40a, 42b, 43, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86ac, 87b, 88a, 91b, 93ab, 99b, 100b, 101c, 102c, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 120a, 122a, 123a, 124abc, 125a, 160bd.

are either not existent ( $\text{XLENGTH}(\text{subset}) == 0$ ) or of length

```

⟨ C integer Nsubset Input 25d ⟩ ≡
    R_xlen_t Nsubset
    ◇

```

Fragment referenced in 25e, 36, 40a, 79a, 81a, 82c, 84b, 87b, 91b, 99b, 103b, 107b, 111a, 115a, 124ab, 125a.

Defines: **Nsubset** 33b, 36, 40a, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86abc, 87b, 88a, 90a, 91b, 93ab, 99b, 100b, 101c, 102c, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 124ab, 125a.

Optionally, one can specify a subset of the subset via

```

⟨ C subset range Input 25e ⟩ ≡
    R_xlen_t offset,
    ⟨ C integer Nsubset Input 25d ⟩
    ◇

```

Fragment referenced in 25f, 26a, 75c, 79b, 81b, 83a, 85a, 87c, 91c, 100a, 103c, 107c, 111b, 115b.

Defines: **offset** 31, 33b, 34abcd, 75d, 79c, 82a, 83b, 85b, 86a, 88a, 93ab, 100b, 101c, 102c, 104a, 108a, 112a, 116a.

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

Subsets are stored either as integer

$\langle C \text{ integer subset Input 25f} \rangle \equiv$   
`int *subset,`  
 $\langle C \text{ subset range Input 25e} \rangle$   
 $\diamond$

Fragment referenced in 89bc, 94c, 95, 97d, 98a, 101b, 102b, 105bc, 109bc, 113bc, 117bc.

Defines: **subset** 3b, 4, 5ab, 6, 8, 15ab, 17, 19, 25c, 26a, 30ab, 31, 32b, 33ab, 35c, 36, 39b, 40a, 42b, 43, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86ac, 87b, 88a, 91b, 93ab, 99b, 100b, 101c, 102c, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 120a, 122a, 123a, 124abc, 125a, 160bd.

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

$\langle C \text{ real subset Input 26a} \rangle \equiv$   
`double *subset,`  
 $\langle C \text{ subset range Input 25e} \rangle$   
 $\diamond$

Fragment referenced in 88b, 89a, 94ab, 97bc, 101a, 102a, 104d, 105a, 108d, 109a, 112d, 113a, 116d, 117a.

Defines: **subset** 3b, 4, 5ab, 6, 8, 15ab, 17, 19, 25cf, 30ab, 31, 32b, 33ab, 35c, 36, 39b, 40a, 42b, 43, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86ac, 87b, 88a, 91b, 93ab, 99b, 100b, 101c, 102c, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 120a, 122a, 123a, 124abc, 125a, 160bd.

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

$\langle R \text{ block Input 26b} \rangle \equiv$   
`SEXP block`  
 $\diamond$

Fragment referenced in 29b, 38c, 46a, 114b, 119ac, 121, 122a.

Defines: **block** 3b, 4, 5a, 6, 8, 15ab, 17, 19, 26d, 30abc, 33ab, 35c, 36, 39b, 40ab, 46b, 115a, 116a, 118a, 119b, 120a, 121, 122a, 140d, 160bd.

at  $B$  levels

$\langle C \text{ integer } B \text{ Input 26c} \rangle \equiv$   
`int B`  
 $\diamond$

Fragment referenced in 26d, 31, 145b, 146.

Defines: **B** 30bc, 31, 32a, 33a, 36, 40ab, 41b, 44, 45, 47, 48b, 68, 69, 72, 115a, 116a, 118a, 128abc, 129ab, 130a, 142d, 144, 145b, 146.

are stored as a factor

$\langle C \text{ integer block Input 26d} \rangle \equiv$   
`int *block,`  
 $\langle C \text{ integer } B \text{ Input 26c} \rangle,$   
 $\diamond$

Fragment referenced in 116b.

Defines: **block** 3b, 4, 5a, 6, 8, 15ab, 17, 19, 26b, 30abc, 33ab, 35c, 36, 39b, 40ab, 46b, 115a, 116a, 118a, 119b, 120a, 121, 122a, 140d, 160bd.

The tabulation of block (potentially in subsets) is

$\langle R \text{ blockTable Input 26e} \rangle \equiv$   
`SEXP blockTable`  
 $\diamond$

Fragment referenced in 119c, 121, 122a.

Defines: **blockTable** 36, 119b, 120a, 121, 122a.

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 case 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 case 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.

```
> LSEC <-
+ function(X, Y, weights = integer(0), subset = integer(0), block = integer(0))
+ {
+   if (length(weights) == 0) weights <- rep.int(1, NROW(X))
+   if (length(subset) == 0) subset <- seq_len(NROW(X))
+
+   X <- X[subset,, drop = FALSE]
+   Y <- Y[subset,, drop = FALSE]
+   weights <- weights[subset]
+
+   if (length(block) == 0) {
+     w. <- sum(weights)
+     wX <- weights * X
+     wY <- weights * Y
+     ExpX <- colSums(wX)
+     ExpY <- colSums(wY) / w.
+     CovX <- crossprod(X, wX)
+     Yc <- t(t(Y) - ExpY)
+     CovY <- crossprod(Yc, weights * Yc) / w.
+     T <- crossprod(X, wY)
+     Exp <- kronecker(ExpY, ExpX)
+     Cov <- w. / (w. - 1) * kronecker(CovY, CovX) -
```

```

+           1 / (w. - 1) * kronecker(CovY, tcrossprod(ExpX))
+
+       list(LinearStatistic = as.vector(T), Expectation = as.vector(Exp),
+           Covariance = Cov, Variance = diag(Cov))
+   } else {
+       block <- block[subset]
+       ret <- list(LinearStatistic = 0, Expectation = 0,
+           Covariance = 0, Variance = 0)
+       for (b in levels(block)) {
+           tmp <- LSEC(X = X, Y = Y, weights = weights, subset = which(block == b))
+           for (l in names(ret)) ret[[l]] <- ret[[l]] + tmp[[l]]
+       }
+       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 LSEC()) with the 1d and 2d C level implementations in the two situations with and without specification of X (i.e., the dummy matrix in the latter case).

```

> ### with X given
> testit <-
+ function(...)
+ {
+   a <- LinStatExpCov(x, y, ...)
+   b <- LSEC(x, y, ...)
+   d <- LinStatExpCov(X = iX2d, ix = ix, Y = iY2d, iy = iy, ...)
+   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 <- LSEC(Xfactor, y, ...)
+   d <- LinStatExpCov(X = integer(0), ix = ix, Y = iY2d, iy = iy, ...)
+   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)
+ )

```

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_` are C functions with pointer arguments only and return a scalar or 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 29a ⟩ ≡
  ⟨ RC_ExpectationCovarianceStatistic 31 ⟩
  ⟨ R_ExpectationCovarianceStatistic 30b ⟩
  ⟨ R_PermutedLinearStatistic 36 ⟩
  ⟨ R_StandardisePermutedLinearStatistic 38a ⟩
  ◇

```

Fragment referenced in [22c](#).

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

```

⟨ User Interface Input 29b ⟩ ≡
  ⟨ R x Input 23c ⟩
  ⟨ R y Input 24a ⟩
  ⟨ R weights Input 24e ⟩,
  ⟨ R subset Input 25c ⟩,
  ⟨ R block Input 26b ⟩,
  ◇

```

Fragment referenced in [29c](#), [31](#), [35b](#).



```

⟨ R_ExpectationCovarianceStatistic Prototype 29c ⟩ ≡
  SEXP R_ExpectationCovarianceStatistic
  (
    ⟨ User Interface Input 29b ⟩
    SEXP varonly,
    SEXP tol
  )
  ◇

```

Fragment referenced in 22a, 30b.

Uses: R\_ExpectationCovarianceStatistic 30b.

This function can be called from other packages.

```

"src/libcoinAPI.h" 30a≡
  ⟨ C Header 152a ⟩
  #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 30a, 35c, 37c, 39b, 46b, 49c, 52, 54b, 60a, 128c, 131b, 134b, 136b.

Uses: block 26bd, R\_ExpectationCovarianceStatistic 30b, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

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

```

⟨ R_ExpectationCovarianceStatistic 30b ⟩ ≡
  ⟨ R_ExpectationCovarianceStatistic Prototype 29c ⟩
  {
    SEXP ans;

    ⟨ Setup Dimensions 30c ⟩

    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 29a.

Defines: R\_ExpectationCovarianceStatistic 6, 29c, 30a, 149, 150, 159ab, 160bcd.

Uses: B 26c, block 26bd, P 23d, Q 24b, RC\_ExpectationCovarianceStatistic 31, 44, RC\_init\_LECV\_1d 145b, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

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

```

⟨ Setup Dimensions 30c ⟩ ≡
  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 [30b](#), [36](#).

Uses: B [26c](#), block [26bd](#), NCOL [126b](#), NLEVELS [127a](#), P [23d](#), Q [24b](#), x [23cef](#), y [24acd](#).

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: `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 `LinStatExpCov()`)

```

⟨ RC_ExpectationCovarianceStatistic 31 ⟩ ≡
void RC_ExpectationCovarianceStatistic
(
    ⟨ User Interface Input 29b ⟩
    SEXP ans
) {
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer P Input 23d ⟩;
    ⟨ C integer Q Input 24b ⟩;
    ⟨ C integer B Input 26c ⟩;
    double *sumweights, *table;
    double *ExpInf, *VarInf, *CovInf, *ExpX, *ExpXtotal, *VarX, *CovX;
    double *tmpV, *tmpCV;
    SEXP nullvec, subset_block;

    ⟨ Extract Dimensions 32a ⟩

    ⟨ Compute Linear Statistic 32b ⟩

    ⟨ Setup Memory and Subsets in Blocks 33a ⟩

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

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

        ⟨ Compute Sum of Weights in Block 33b ⟩

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

            ⟨ Compute Expectation Linear Statistic 34a ⟩

            ⟨ Compute Covariance Influence 34b ⟩

            if (C_get_varonly(ans)) {
                ⟨ Compute Variance Linear Statistic 34c ⟩
            } else {
                ⟨ Compute Covariance Linear Statistic 34d ⟩
            }
        }

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

    ⟨ Compute Variance from Covariance 35a ⟩

    R_Free(ExpX); R_Free(VarX); R_Free(CovX);
    UNPROTECT(2);
}
◇

```

Fragment referenced in [29a](#).

Defines: `RC_ExpectationCovarianceStatistic` [30b](#).

Uses: `B` [26c](#), `C_get_varonly` [138a](#), `offset` [25e](#), `subset` [25cf](#), [26a](#), `sumweights` [25b](#).

The dimensions are available from the return object:

*⟨ Extract Dimensions 32a ⟩*  $\equiv$

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

Fragment referenced in 31.

Uses: B 26c, C\_get\_B 141d, C\_get\_P 137c, C\_get\_Q 137d, N 23ab, NROW 126a, P 23d, Q 24b, x 23cef.

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

*⟨ Compute Linear Statistic 32b ⟩*  $\equiv$

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

Fragment referenced in 31.

Uses: C\_get\_LinearStatistic 138c, N 23ab, Offset0 21a, P 23d, Q 24b, RC\_LinearStatistic 75d, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

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 absence of block, this just returns subset.

*⟨ Setup Memory and Subsets in Blocks 33a ⟩*  $\equiv$

```
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 = R_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 = R_Calloc(P, double);
    CovX = R_Calloc(1, double);
} else {
    VarX = R_Calloc(1, double);
    CovX = R_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 31.

Uses: B 26c, block 26bd, C\_get\_CovarianceInfluence 140b, C\_get\_ExpectationInfluence 140a, C\_get\_ExpectationX 139c, C\_get\_Sumweights 141a, C\_get\_TableBlock 140d, C\_get\_VarianceInfluence 140c, C\_get\_varonly 138a, N 23ab, Offset0 21a, P 23d, PP12 127b, RC\_OneTableSums 108a, RC\_order\_subset\_wrt\_block 120a, RC\_Sums 88a, subset 25cf, 26a, sumweights 25b, TableBlock\_SLOT 21a.

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.

```

⟨ Compute Sum of Weights in Block 33b ⟩ ≡
/* compute sum of case 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 31.

Uses: block 26bd, N 23ab, Nsubset 25d, offset 25e, RC\_Sums 88a, subset 25cf, 26a, sumweights 25b, weights 24ef, 25a.

```

⟨ Compute Expectation Linear Statistic 34a ⟩ ≡
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 31.

Uses: C\_ExpectationLinearStatistic 76b, C\_get\_Expectation 138d, N 23ab, offset 25e, P 23d, Q 24b, RC\_ExpectationInfluence 79c, RC\_ExpectationX 83b, sumweights 25b, weights 24ef, 25a, x 23cef, y 24acd.

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).

```

⟨ Compute Covariance Influence 34b ⟩ ≡
/* 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 31.

Uses: C\_ordered\_Xfactor 68, C\_unordered\_Xfactor 72, DoVarOnly 21a, N 23ab, offset 25e, Q 24b, RC\_CovarianceInfluence 82a, S 20c, sumweights 25b, weights 24ef, 25a, y 24acd.

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

```

⟨ Compute Variance Linear Statistic 34c ⟩ ≡
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 31.

Uses: C\_get\_Variance 139a, C\_VarianceLinearStatistic 78a, DoVarOnly 21a, N 23ab, offset 25e, P 23d, Q 24b, RC\_CovarianceX 85b, sumweights 25b, weights 24ef, 25a, x 23cef.

```

⟨ Compute Covariance Linear Statistic 34d ⟩ ≡
    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 31.

Uses: C\_CovarianceLinearStatistic 77, C\_get\_Covariance 139b, DoVarOnly 21a, N 23ab, offset 25e, P 23d, Q 24b, RC\_CovarianceX 85b, sumweights 25b, weights 24ef, 25a, x 23cef.

```

⟨ Compute Variance from Covariance 35a ⟩ ≡
    /* 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 31.

Uses: C\_get\_Covariance 139b, C\_get\_Variance 139a, C\_get\_varonly 138a, mPQB 128a, P 23d, Q 24b, S 20c.

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 35b ⟩ ≡
    SEXP R_PermutedLinearStatistic
    (
        ⟨ User Interface Input 29b ⟩
        SEXP nresample
    )
    ◇

```

Fragment referenced in 22a, 36.

Uses: R\_PermutedLinearStatistic 36.

This function can be called from other packages.

```

"src/libcoinAPI.h" 35c≡
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 30a, 35c, 37c, 39b, 46b, 49c, 52, 54b, 60a, 128c, 131b, 134b, 136b.

Uses: block 26bd, R\_PermutedLinearStatistic 36, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

The dimensions are extracted from the data in the same ways as above. The function differentiates between the absence and presence of blocks. Case 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).

```

⟨ R_PermutedLinearStatistic 36 ⟩ ≡
  ⟨ R_PermutedLinearStatistic Prototype 35b ⟩
  {
    SEXP ans, expand_subset, block_subset, perm, tmp, blockTable;
    double *linstat;
    int PQ;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;
    R_xlen_t inresample;

    ⟨ Setup Dimensions 30c ⟩
    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++) {
        ⟨ Setup Linear Statistic 37a ⟩
        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, noweight, 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++) {
        ⟨ Setup Linear Statistic 37a ⟩
        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 29a.

Defines: R\_PermutedLinearStatistic 6, 35bc, 149, 150.

Uses: B 26c, block 26bd, blockTable 26e, C\_doPermute 124b, C\_doPermuteBlock 125a, mPQB 128a, N 23ab, NROW 126a, Nsubset 25d, Offset0 21a, P 23d, Q 24b, RC\_KronSums\_Permutation 100b, RC\_OneTableSums 108a, RC\_order\_subset\_wrt\_block 120a, RC\_setup\_subset 123a, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

```

⟨ Setup Linear Statistic 37a ⟩ ≡
    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 [36](#), [47](#).

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

```

⟨ R_StandardisePermutedLinearStatistic Prototype 37b ⟩ ≡
    SEXP R_StandardisePermutedLinearStatistic
    (
        SEXP LECV
    )
    ◇

```

Fragment referenced in [22a](#), [38a](#).

Uses: LECV [137b](#).

This function can be called from other packages.

```

"src/libcoinAPI.h" 37c≡
    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 [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).

Uses: LECV [137b](#).



```

⟨ R_StandardisePermutedLinearStatistic 38a ⟩ ≡
  ⟨ R_StandardisePermutedLinearStatistic Prototype 37b ⟩
  {
    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 [29a](#).

Uses: [C\\_get\\_Covariance 139b](#), [C\\_get\\_Expectation 138d](#), [C\\_get\\_nresample 142a](#), [C\\_get\\_P 137c](#),  
[C\\_get\\_PermutedLinearStatistic 142b](#), [C\\_get\\_Q 137d](#), [C\\_get\\_tol 142c](#), [C\\_get\\_Variance 139a](#), [C\\_get\\_varonly 138a](#),  
[C\\_standardise 62a](#), [LECV 137b](#).

### 3.4.2 Two-Dimensional Case (“2d”)

```

⟨ 2d User Interface 38b ⟩ ≡
  ⟨ RC_ExpectationCovarianceStatistic_2d 44 ⟩
  ⟨ R_ExpectationCovarianceStatistic_2d 40a ⟩
  ⟨ R_PermutedLinearStatistic_2d 47 ⟩
  ◇

```

Fragment referenced in [22c](#).

```

⟨ 2d User Interface Input 38c ⟩ ≡
  ⟨ R x Input 23c ⟩
  SEXP ix,
  ⟨ R y Input 24a ⟩
  SEXP iy,
  ⟨ R weights Input 24e ⟩,
  ⟨ R subset Input 25c ⟩,
  ⟨ R block Input 26b ⟩,
  ◇

```

Fragment referenced in [39a](#), [44](#).

```

⟨ R_ExpectationCovarianceStatistic_2d Prototype 39a ⟩ ≡
  SEXP R_ExpectationCovarianceStatistic_2d
  (
    ⟨ 2d User Interface Input 38c ⟩
    SEXP varonly,
    SEXP tol
  )
◇

```

Fragment referenced in [22a](#), [40a](#).

Uses: [R\\_ExpectationCovarianceStatistic\\_2d 40a](#).

This function can be called from other packages.

```

"src/libcoinAPI.h" 39b≡
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);
}
◇

```

File defined by [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).

Uses: [block 26bd](#), [R\\_ExpectationCovarianceStatistic\\_2d 40a](#), [subset 25cf](#), [26a](#), [weights 24ef](#), [25a](#), [x 23cef](#), [y 24acd](#).

```

⟨ R_ExpectationCovarianceStatistic_2d 40a ⟩ ≡
  ⟨ R_ExpectationCovarianceStatistic_2d Prototype 39a ⟩
  {
    SEXP ans;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;
    int Xfactor;

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

    ⟨ Setup Dimensions 2d 40b ⟩

    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 38b.

Defines: R\_ExpectationCovarianceStatistic\_2d 8, 39ab, 149, 150.

Uses: B 26c, block 26bd, C\_get\_Table 141b, N 23ab, Nsubset 25d, Offset0 21a, P 23d, Q 24b, RC\_init\_LECV\_2d 146, RC\_ThreeTableSums 116a, RC\_TwoTableSums 112a, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

```

⟨ Setup Dimensions 2d 40b ⟩ ≡
  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 40a, 47.

Uses: B 26c, block 26bd, NCOL 126b, NLEVELS 127a, P 23d, Q 24b, x 23cef, y 24acd.

$\langle \text{Linear Statistic 2d 41a} \rangle \equiv$

```

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 [44](#), [48d](#).

Uses: P [23d](#), Q [24b](#), x [23cef](#), y [24acd](#).

$\langle \text{2d Total Table 41b} \rangle \equiv$

```

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 [44](#).

Uses: B [26c](#).

```

⟨ Col Row Total Sums 42a ⟩ ≡
/* Remember: first row / column count NAs */
/* column sums */
for (int q = 1; q < Lxp1; 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 < Lxp1; 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 [44](#), [47](#).  
 Uses: [sumweights 25b](#).

```

⟨ 2d Expectation 42b ⟩ ≡
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 [44](#).  
 Uses: [C\\_ExpectationLinearStatistic 76b](#), [C\\_get\\_Expectation 138d](#), [NROW 126a](#), [Offset0 21a](#), [P 23d](#), [Q 24b](#),  
[RC\\_ExpectationInfluence 79c](#), [RC\\_ExpectationX 83b](#), [subset 25cf](#), [26a](#), [sumweights 25b](#), [x 23cef](#), [y 24acd](#).

( 2d Covariance 43 ) ≡

```

/* 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 44.

Uses: C\_CovarianceLinearStatistic 77, C\_get\_Covariance 139b, C\_get\_CovarianceInfluence 140b,  
 C\_get\_Variance 139a, C\_get\_VarianceInfluence 140c, C\_get\_varonly 138a, C\_ordered\_Xfactor 68,  
 C\_VarianceLinearStatistic 78a, DoVarOnly 21a, NROW 126a, Offset0 21a, P 23d, PP12 127b, Q 24b,  
 RC\_CovarianceInfluence 82a, RC\_CovarianceX 85b, S 20c, subset 25cf, 26a, sumweights 25b, x 23cef, y 24acd.

```

⟨ RC_ExpectationCovarianceStatistic_2d 44 ⟩ ≡
void RC_ExpectationCovarianceStatistic_2d
(
    ⟨ 2d User Interface Input 38c ⟩
    SEXP ans
) {
    ⟨ 2d Memory 45 ⟩

    ⟨ 2d Total Table 41b ⟩

    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;

        ⟨ Linear Statistic 2d 41a ⟩

        ⟨ Col Row Total Sums 42a ⟩

        ⟨ 2d Expectation 42b ⟩

        ⟨ 2d Covariance 43 ⟩
    }

    /* 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))];
    }

    R_Free(CovX);
    R_Free(table2d);
    UNPROTECT(2);
}
◇

```

Fragment referenced in 38b.

Defines: RC\_ExpectationCovarianceStatistic 30b, 31.

Uses: B 26c, C\_get\_Covariance 139b, C\_get\_LinearStatistic 138c, C\_get\_Variance 139a, C\_get\_varonly 138a, mPQB 128a, P 23d, Q 24b, S 20c.

( 2d Memory 45 ) ≡

```

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 = R_Calloc(P, double);
} else {
    CovX = R_Calloc(PP12(P), double);
}

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

```

◇

Fragment referenced in 44.

Uses: B 26c, C\_get\_B 141d, C\_get\_dimTable 141c, C\_get\_ExpectationInfluence 140a, C\_get\_ExpectationX 139c, C\_get\_P 137c, C\_get\_Q 137d, C\_get\_Sumweights 141a, C\_get\_Table 141b, C\_get\_varonly 138a, C\_get\_Xfactor 138b, P 23d, PP12 127b, Q 24b, sumweights 25b.

```

> 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
```

```
< R_PermutedLinearStatistic_2d Prototype 46a > ≡
  SEXP R_PermutedLinearStatistic_2d
  (
    < R x Input 23c >
    SEXP ix,
    < R y Input 24a >
    SEXP iy,
    < R block Input 26b >,
    SEXP nresample,
    SEXP itable
  )
◇
```

Fragment referenced in [22a](#), [47](#).

Uses: [R\\_PermutedLinearStatistic\\_2d 47](#).

This function can be called from other packages.

```
"src/libcoinAPI.h" 46b ≡
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 [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).

Uses: [block 26bd](#), [R\\_PermutedLinearStatistic\\_2d 47](#), [x 23cef](#), [y 24acd](#).

```

⟨ R_PermutedLinearStatistic_2d 47 ⟩ ≡
⟨ R_PermutedLinearStatistic_2d Prototype 46a ⟩
{
    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 40b ⟩

    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 48b ⟩

    ⟨ Convert Table to Integer 48a ⟩

    for (int b = 0; b < B; b++) {
        btab = INTEGER(Ritable) + Lxp1 * Lyp1 * b;
        ⟨ Col Row Total Sums 42a ⟩
        if (sumweights[b] > maxn) maxn = sumweights[b];
    }

    ⟨ Setup Log-Factorials 48c ⟩

    GetRNGstate();

    for (R_xlen_t np = 0; np < inresample; np++) {
        ⟨ Setup Linear Statistic 37a ⟩

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

        for (int b = 0; b < B; b++) {
            ⟨ Compute Permuted Linear Statistic 2d 48d ⟩
        }
    }

    PutRNGstate();

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

```

Fragment referenced in 38b.

Defines: R\_PermutedLinearStatistic\_2d 8, 46ab, 48a, 149, 150.

Uses: B 26c, mPQB 128a, P 23d, Q 24b, sumweights 25b, x 23cef.

*⟨ Convert Table to Integer 48a ⟩ ≡*

```
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 [47](#).

Uses: [R\\_PermutedLinearStatistic\\_2d 47](#), [weights 24ef, 25a](#).

*⟨ Setup Working Memory 48b ⟩ ≡*

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

Fragment referenced in [47](#).

Uses: [B 26c](#), [sumweights 25b](#).

*⟨ Setup Log-Factorials 48c ⟩ ≡*

```
fact = R_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 [47](#).

Note: the interface to `S_rcont2` changed in R-4.1.0.

*⟨ Compute Permuted Linear Statistic 2d 48d ⟩ ≡*

```
#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;
⟨ Linear Statistic 2d 41a ⟩
◇
```

Fragment referenced in [47](#).

Uses: [sumweights 25b](#).

## 3.5 Tests

```
⟨ Tests 49a ⟩ ≡  
  ⟨ R_QuadraticTest 50 ⟩  
  ⟨ R_MaximumTest 53 ⟩  
  ⟨ R_MaximallySelectedTest 55 ⟩  
  ◇
```

Fragment referenced in [22c](#).

```
⟨ R_QuadraticTest Prototype 49b ⟩ ≡  
  SEXP R_QuadraticTest  
  (  
    ⟨ R_LECV Input 137b ⟩,  
    SEXP pvalue,  
    SEXP lower,  
    SEXP give_log,  
    SEXP PermutedStatistics  
  )  
  ◇
```

Fragment referenced in [22a](#), [50](#).

This function can be called from other packages.

```
"src/libcoinAPI.h" 49c≡  
extern SEXP libcoin_R_QuadraticTest(  
  SEXP LECV, 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(LECV, pvalue, lower, give_log, PermutedStatistics);  
}  
◇
```

File defined by [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).  
Uses: LECV [137b](#).

```

⟨ R_QuadraticTest 50 ⟩ ≡
  ⟨ R_QuadraticTest Prototype 49b ⟩
  {
    SEXP ans, stat, pval, names, permstat;
    double *MPinv, *ls, st, pst, *ex;
    int rank, P, Q, PQ, greater = 0;
    R_xlen_t nresample;

    ⟨ Setup Test Memory 51a ⟩

    MPinv = R_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);
      R_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);
    R_Free(MPinv);
    return(ans);
  }
  ◇

```

Fragment referenced in [49a](#).

Uses: [C\\_chisq\\_pvalue](#) [62c](#), [C\\_get\\_Covariance](#) [139b](#), [C\\_get\\_Expectation](#) [138d](#), [C\\_get\\_LinearStatistic](#) [138c](#),  
[C\\_get\\_nresample](#) [142a](#), [C\\_get\\_PermutedLinearStatistic](#) [142b](#), [C\\_get\\_tol](#) [142c](#), [C\\_perm\\_pvalue](#) [63](#),  
[C\\_quadform](#) [61a](#), [GE](#) [20c](#), [LECV](#) [137b](#), [P](#) [23d](#), [PP12](#) [127b](#), [Q](#) [24b](#).

```

⟨ Setup Test Memory 51a ⟩ ≡
  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 50, 53.
Uses: C_get_nresample 142a, C_get_P 137c, C_get_Q 137d, C_get_varonly 138a, LECV 137b, mPQB 128a, P 23d, Q 24b.

```

```

⟨ R_MaximumTest Prototype 51b ⟩ ≡
  SEXP R_MaximumTest
  (
    ⟨ R LECV Input 137b ⟩,
    SEXP alternative,
    SEXP pvalue,
    SEXP lower,
    SEXP give_log,
    SEXP PermutedStatistics,
    SEXP maxpts,
    SEXP releps,
    SEXP abseps
  )
  ◇

```

Fragment referenced in 22a, 53.

This function can be called from other packages.

```

"src/libcoinAPI.h" 52≡
extern SEXP libcoin_R_MaximumTest(
    SEXP LECV, 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(LECV, alternative, pvalue, lower, give_log, PermutedStatistics, maxpts, releps,
        abseps);
}
◇

```

File defined by [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).  
 Uses: LECV [137b](#).

```

⟨ R_MaximumTest 53 ⟩ ≡
  ⟨ R_MaximumTest Prototype 51b ⟩
  {
    SEXP ans, stat, pval, names, permstat;
    double st, pst, *ex, *cv, *ls, tl;
    int P, Q, PQ, vo, alt, greater;
    R_xlen_t nresample;

    ⟨ Setup Test Memory 51a ⟩

    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 [49a](#).

Uses: ALTERNATIVE\_less [21a](#), C\_get\_Covariance [139b](#), C\_get\_Expectation [138d](#), C\_get\_LinearStatistic [138c](#), C\_get\_nresample [142a](#), C\_get\_PermutedLinearStatistic [142b](#), C\_get\_tol [142c](#), C\_get\_Variance [139a](#), C\_get\_varonly [138a](#), C\_maxtype [61b](#), C\_maxtype\_pvalue [65](#), C\_perm\_pvalue [63](#), GE [20c](#), LE [20c](#), LECV [137b](#), P [23d](#), Q [24b](#).



```

⟨ R_MaximallySelectedTest Prototype 54a ⟩ ≡
  SEXP R_MaximallySelectedTest
  (
    SEXP LECV,
    SEXP ordered,
    SEXP teststat,
    SEXP minbucket,
    SEXP lower,
    SEXP give_log
  )
  ◇

```

Fragment referenced in [22a](#), [55](#).  
 Uses: [LECV 137b](#).

This function can be called from other packages.

```

"src/libcoinAPI.h" 54b≡
extern SEXP libcoin_R_MaximallySelectedTest(
  SEXP LECV, 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(LECV, ordered, teststat, minbucket, lower, give_log);
}
  ◇

```

File defined by [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).  
 Uses: [LECV 137b](#).

```

< R_MaximallySelectedTest 55 > ≡
< R_MaximallySelectedTest Prototype 54a >
{
  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 [49a](#).

Uses: [C\\_get\\_nresample 142a](#), [C\\_get\\_P 137c](#), [C\\_ordered\\_Xfactor 68](#), [C\\_unordered\\_Xfactor 72](#), [LECV 137b](#), [P 23d](#).

### 3.6 Test Statistics

$\langle \text{Test Statistics } 56a \rangle \equiv$   
 $\langle C\_maxstand\_Covariance \ 56b \rangle$   
 $\langle C\_maxstand\_Variance \ 57a \rangle$   
 $\langle C\_minstand\_Covariance \ 57b \rangle$   
 $\langle C\_minstand\_Variance \ 58a \rangle$   
 $\langle C\_maxabsstand\_Covariance \ 58b \rangle$   
 $\langle C\_maxabsstand\_Variance \ 59a \rangle$   
 $\langle C\_quadform \ 61a \rangle$   
 $\langle R\_quadform \ 60b \rangle$   
 $\langle C\_maxtype \ 61b \rangle$   
 $\langle C\_standardise \ 62a \rangle$   
 $\langle C\_ordered\_Xfactor \ 68 \rangle$   
 $\langle C\_unordered\_Xfactor \ 72 \rangle$   
 $\diamond$

Fragment referenced in [22c](#).

$\langle C\_maxstand\_Covariance \ 56b \rangle \equiv$   

```
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);
}

```

 $\diamond$

Fragment referenced in [56a](#).

Defines: `C_maxstand_Covariance` [61b](#).

Uses: `S` [20c](#).

```

⟨ C_maxstand_Variance 57a ⟩ ≡
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 [56a](#).

Defines: C\_maxstand\_Variance [61b](#).

```

⟨ C_minstand_Covariance 57b ⟩ ≡
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 [56a](#).

Defines: C\_minstand\_Covariance [61b](#).

Uses: S [20c](#).

```

⟨ C_minstand_Variance 58a ⟩ ≡
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 [56a](#).

Defines: C\_minstand\_Variance [61b](#).

```

⟨ C_maxabsstand_Covariance 58b ⟩ ≡
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 [56a](#).

Defines: C\_maxabsstand\_Covariance [61b](#).

Uses: S [20c](#).

```

< C_maxabsstand_Variance 59a > ≡
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 [56a](#).

Defines: C\_maxabsstand\_Variance [61b](#).

```

> 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))

```

```

< R_quadform Prototype 59b > ≡
SEXP R_quadform
(
    SEXP linstat,
    SEXP expect,
    SEXP MPinv_sym
)
◇

```

Fragment referenced in [22a](#), [60b](#).

Uses: R\_quadform [60b](#).

This function can be called from other packages.

```

"src/libcoinAPI.h" 60a≡
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 [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).  
 Uses: [R\\_quadform 60b](#).

```

⟨ R_quadform 60b ⟩ ≡
  ⟨ R_quadform Prototype 59b ⟩
  {
    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 [56a](#).  
 Defines: [R\\_quadform 59b](#), [60a](#), [149](#), [150](#), [155a](#).  
 Uses: [C\\_quadform 61a](#), [NCOL 126b](#), [NROW 126a](#).

```

⟨ C_quadform 61a ⟩ ≡
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 [56a](#).

Defines: `C_quadform` [50](#), [60b](#), [71b](#), [155a](#).

Uses: `S` [20c](#).

```

⟨ C_maxtype 61b ⟩ ≡
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 [56a](#).

Defines: `C_maxtype` [53](#), [71b](#).

Uses: `ALTERNATIVE_greater` [21a](#), `ALTERNATIVE_less` [21a](#), `ALTERNATIVE_twosided` [21a](#), `C_maxabsstand_Covariance` [58b](#), `C_maxabsstand_Variance` [59a](#), `C_maxstand_Covariance` [56b](#), `C_maxstand_Variance` [57a](#), `C_minstand_Covariance` [57b](#), `C_minstand_Variance` [58a](#).



```

⟨ C_standardise 62a ⟩ ≡
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 [56a](#).  
 Defines: `C_standardise` [38a](#).  
 Uses: `S` [20c](#).

```

⟨ P-Values 62b ⟩ ≡
⟨ C_chisq_pvalue 62c ⟩
⟨ C_perm_pvalue 63 ⟩
⟨ C_norm_pvalue 64 ⟩
⟨ C_maxtype_pvalue 65 ⟩
◇

```

Fragment referenced in [22c](#).

```

⟨ C_chisq_pvalue 62c ⟩ ≡
/* 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 [62b](#).  
 Defines: `C_chisq_pvalue` [50](#).

```

⟨ C_perm_pvalue 63 ⟩ ≡
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 [62b](#).

Defines: C\_perm\_pvalue [50](#), [53](#), [71c](#).

```

⟨ C_norm_pvalue 64 ⟩ ≡
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 [62b](#).

Defines: `C_norm_pvalue` [65](#).

Uses: `ALTERNATIVE_greater` [21a](#), `ALTERNATIVE_less` [21a](#), `ALTERNATIVE_twosided` [21a](#).

```

⟨ C_maxtype_pvalue 65 ⟩ ≡
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));

    ⟨ Setup mvtnorm Memory 66 ⟩

    ⟨ Setup mvtnorm Correlation 67a ⟩

    /* call mvtnorm's mvtdst C function defined in mvtnorm/include/mvtnormAPI.h */
    mvtnorm_C_mvtdst(&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 MVTGST");
                ans = 0.0;
    }
    R_Free(corr); R_Free(sd); R_Free(lowerbnd); R_Free(upperbnd);
    R_Free(infin); R_Free(delta); R_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 [62b](#).

Defines: C\_maxtype\_pvalue [53](#).

Uses: C\_norm\_pvalue [64](#), N [23ab](#).

$\langle \text{Setup mvtnorm Memory 66} \rangle \equiv$

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

sd = R_Calloc(n, double);
lowerbnd = R_Calloc(n, double);
upperbnd = R_Calloc(n, double);
infin = R_Calloc(n, int);
delta = R_Calloc(n, double);
index = R_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 [65](#).

Uses: [S 20c](#).

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

*⟨ Setup mvtnorm Correlation 67a ⟩ ≡*

```

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 [65](#).

Uses: ALTERNATIVE\_greater [21a](#), ALTERNATIVE\_less [21a](#), ALTERNATIVE\_twosided [21a](#), [S 20c](#).

*⟨ maxstat Xfactor Variables 67b ⟩ ≡*

```

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 [68](#), [72](#).

Uses: LECV [137b](#).

```

⟨ C_ordered_Xfactor 68 ⟩ ≡
void C_ordered_Xfactor
(
    ⟨ maxstat Xfactor Variables 67b ⟩
) {
    ⟨ Setup maxstat Variables 69 ⟩

    ⟨ Setup maxstat Memory 70a ⟩

    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) {
                ⟨ Compute maxstat Variance / Covariance Directly 71a ⟩
            } else {
                ⟨ Compute maxstat Variance / Covariance from Total Covariance 70b ⟩
            }
        }

        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);
            ⟨ Compute maxstat Test Statistic 71b ⟩
            if (tmp > maxstat[0]) {
                wmax[0] = p;
                maxstat[0] = tmp;
            }

            for (R_xlen_t np = 0; np < nresample; np++) {
                ls = mblinstat + np * Q;
                ⟨ Compute maxstat Test Statistic 71b ⟩
                if (tmp > bmaxstat[np])
                    bmaxstat[np] = tmp;
            }
        }
    }
    ⟨ Compute maxstat Permutation P-Value 71c ⟩
    R_Free(mlinstat); R_Free(mexpect); R_Free(mblinstat);
    R_Free(mvar); R_Free(mcovar); R_Free(mMPinv);
    if (nresample == 0) R_Free(blinstat);
}
◇

```

Fragment referenced in 56a.

Defines: C\_ordered\_Xfactor 34b, 43, 55.

Uses: B 26c, P 23d, Q 24b, TESTSTAT\_maximum 21a.

*< Setup maxstat Variables 69 >*  $\equiv$

```
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 [68](#), [72](#).

Uses: [B 26c](#), [C\\_get\\_B 141d](#), [C\\_get\\_Covariance 139b](#), [C\\_get\\_CovarianceInfluence 140b](#), [C\\_get\\_Expectation 138d](#),  
[C\\_get\\_ExpectationX 139c](#), [C\\_get\\_LinearStatistic 138c](#), [C\\_get\\_nresample 142a](#), [C\\_get\\_P 137c](#),  
[C\\_get\\_PermutedLinearStatistic 142b](#), [C\\_get\\_Q 137d](#), [C\\_get\\_tol 142c](#), [C\\_get\\_Variance 139a](#),  
[C\\_get\\_VarianceInfluence 140c](#), [C\\_get\\_varonly 138a](#), [LECV 137b](#), [mPQB 128a](#), [P 23d](#), [Q 24b](#), [sumweights 25b](#).



*⟨ Setup maxstat Memory 70a ⟩ ≡*

```

mlinstat = R_Calloc(Q, double);
mexpect = R_Calloc(Q, double);
if (teststat == TESTSTAT_maximum) {
    mvar = R_Calloc(Q, double);
    /* not needed, but allocate anyway to make -Wmaybe-uninitialized happy */
    mcovar = R_Calloc(1, double);
    mMPinv = R_Calloc(1, double);
} else {
    mcovar = R_Calloc(Q * (Q + 1) / 2, double);
    mMPinv = R_Calloc(Q * (Q + 1) / 2, double);
    /* not needed, but allocate anyway to make -Wmaybe-uninitialized happy */
    mvar = R_Calloc(1, double);
}
if (nresample > 0) {
    mblinstat = R_Calloc(Q * nresample, double);
} else { /* not needed, but allocate anyway to make -Wmaybe-uninitialized happy */
    mblinstat = R_Calloc(1, double);
    blinstat = R_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 [68](#), [72](#).

Uses: P [23d](#), Q [24b](#), sumweights [25b](#), TESTSTAT\_maximum [21a](#), TESTSTAT\_quadratic [21a](#).

*⟨ Compute maxstat Variance / Covariance from Total Covariance 70b ⟩ ≡*

```

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 [68](#).

Uses: mPQB [128a](#), P [23d](#), Q [24b](#), S [20c](#), TESTSTAT\_maximum [21a](#).

```

⟨ Compute maxstat Variance / Covariance Directly 71a ⟩ ≡
/* 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 [68](#).

Uses: C\_CovarianceLinearStatistic [77](#), C\_VarianceLinearStatistic [78a](#), Q [24b](#), sumweights [25b](#), TESTSTAT\_maximum [21a](#).

```

⟨ Compute maxstat Test Statistic 71b ⟩ ≡
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 [68](#), [72](#).

Uses: ALTERNATIVE\_twosided [21a](#), C\_maxtype [61b](#), C\_quadform [61a](#), Q [24b](#), TESTSTAT\_maximum [21a](#).

```

⟨ Compute maxstat Permutation P-Value 71c ⟩ ≡
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 [68](#), [72](#).

Uses: C\_perm\_pvalue [63](#).

```

⟨ C_unordered_Xfactor 72 ⟩ ≡
void C_unordered_Xfactor
(
    ⟨ maxstat Xfactor Variables 67b ⟩
) {
    double *mtmp;
    int qPp, nc, *levels, Pnonzero, *indl, *contrast;

    ⟨ Setup maxstat Variables 69 ⟩

    ⟨ Setup maxstat Memory 70a ⟩
    mtmp = R_Calloc(P, double);

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

    ⟨ Count Levels 73a ⟩

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

        ⟨ Setup unordered maxstat Contrasts 73b ⟩

        ⟨ Compute unordered maxstat Linear Statistic and Expectation 74a ⟩

        if (B == 1) {
            ⟨ Compute unordered maxstat Variance / Covariance Directly 75a ⟩
        } else {
            ⟨ Compute unordered maxstat Variance / Covariance from Total Covariance 74b ⟩
        }

        if ((sumleft >= minbucket) && (sumright >= minbucket)) {
            ls = mlinstat;
            /* compute MPinv only once */
            if (teststat != TESTSTAT_maximum)
                C_MPinv_sym(mcovar, Q, tol, mMPinv, &rank);
            ⟨ Compute maxstat Test Statistic 71b ⟩
            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;
                ⟨ Compute maxstat Test Statistic 71b ⟩
                if (tmp > bmaxstat[np])
                    bmaxstat[np] = tmp;
            }
        }
    }

    ⟨ Compute maxstat Permutation P-Value 71c ⟩

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

```

Fragment referenced in 56a.

Defines: C\_unordered\_Xfactor 34b, 55.

Uses: B 26c, P 23d, Q 24b, TESTSTAT\_maximum 21a.

```

< Count Levels 73a > ≡
    contrast = R_Calloc(P, int);
    Pnonzero = 0;
    for (int p = 0; p < P; p++) {
        if (ExpX[p] > 0) Pnonzero++;
    }
    levels = R_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 = R_Calloc(Pnonzero, int);
    for (int p = 0; p < Pnonzero; p++) indl[p] = 0;
    ◇

```

Fragment referenced in [72](#).  
 Uses: P [23d](#).

```

< Setup unordered maxstat Contrasts 73b > ≡
    /* 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 [72](#).  
 Uses: P [23d](#).

*⟨ Compute unordered maxstat Linear Statistic and Expectation 74a ⟩ ≡*

```

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 [72](#).

Uses: [P 23d](#), [Q 24b](#).

*⟨ Compute unordered maxstat Variance / Covariance from Total Covariance 74b ⟩ ≡*

```

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 [72](#).

Uses: [mPQB 128a](#), [P 23d](#), [Q 24b](#), [S 20c](#), [TESTSTAT\\_maximum 21a](#).

```

⟨ Compute unordered maxstat Variance / Covariance Directly 75a ⟩ ≡
    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 72.

Uses: C\_CovarianceLinearStatistic 77, C\_VarianceLinearStatistic 78a, Q 24b, sumweights 25b, TESTSTAT\_maximum 21a.

### 3.7 Linear Statistics

```

⟨ LinearStatistics 75b ⟩ ≡
    ⟨ RC_LinearStatistic 75d ⟩
    ◇

```

Fragment referenced in 22c.

```

⟨ RC_LinearStatistic Prototype 75c ⟩ ≡
    void RC_LinearStatistic
    (
        ⟨ R x Input 23c ⟩
        ⟨ C integer N Input 23b ⟩,
        ⟨ C integer P Input 23d ⟩,
        ⟨ C real y Input 24c ⟩
        ⟨ R weights Input 24e ⟩,
        ⟨ R subset Input 25c ⟩,
        ⟨ C subset range Input 25e ⟩,
        ⟨ C KronSums Answer 92d ⟩
    )
    ◇

```

Fragment referenced in 75d.

Uses: RC\_LinearStatistic 75d.

```

⟨ RC_LinearStatistic 75d ⟩ ≡
    ⟨ RC_LinearStatistic Prototype 75c ⟩
    {
        double center;

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

```

Fragment referenced in 75b.

Defines: RC\_LinearStatistic 32b, 75c.

Uses: DoCenter 21a, DoSymmetric 21a, N 23ab, Nsubset 25d, offset 25e, P 23d, Q 24b, RC\_KronSums 92a, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

## 3.8 Expectation and Covariance

$\langle \text{ExpectationCovariances } 76a \rangle \equiv$   
 $\langle \text{RC\_ExpectationInfluence } 79c \rangle$   
 $\langle \text{R\_ExpectationInfluence } 79a \rangle$   
 $\langle \text{RC\_CovarianceInfluence } 82a \rangle$   
 $\langle \text{R\_CovarianceInfluence } 81a \rangle$   
 $\langle \text{RC\_ExpectationX } 83b \rangle$   
 $\langle \text{R\_ExpectationX } 82c \rangle$   
 $\langle \text{RC\_CovarianceX } 85b \rangle$   
 $\langle \text{R\_CovarianceX } 84b \rangle$   
 $\langle \text{C\_ExpectationLinearStatistic } 76b \rangle$   
 $\langle \text{C\_CovarianceLinearStatistic } 77 \rangle$   
 $\langle \text{C\_VarianceLinearStatistic } 78a \rangle$   
 $\diamond$

Fragment referenced in [22c](#).

### 3.8.1 Linear Statistic

$\langle \text{C\_ExpectationLinearStatistic } 76b \rangle \equiv$   

```
void C_ExpectationLinearStatistic
(
     $\langle \text{C integer } P \text{ Input } 23d \rangle$ ,
     $\langle \text{C integer } Q \text{ Input } 24b \rangle$ ,
    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 [76a](#).

Defines: `C_ExpectationLinearStatistic` [34a](#), [42b](#).

Uses: `mPQB` [128a](#), `P` [23d](#), `Q` [24b](#).

```

⟨ C_CovarianceLinearStatistic 77 ⟩ ≡
void C_CovarianceLinearStatistic
(
    ⟨ C integer P Input 23d ⟩,
    ⟨ C integer Q Input 24b ⟩,
    double *CovInf,
    double *ExpX,
    double *CovX,
    ⟨ C sumweights Input 25b ⟩,
    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 = R_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);
        R_Free(PP_sym_tmp);
    }
}
◇

```

Fragment referenced in 76a.

Defines: C\_CovarianceLinearStatistic 34d, 43, 71a, 75a, 78a.

Uses: C\_kronecker\_sym 130a, mPQB 128a, P 23d, PP12 127b, Q 24b, sumweights 25b.



```

⟨ C_VarianceLinearStatistic 78a ⟩ ≡
void C_VarianceLinearStatistic
(
  ⟨ C integer P Input 23d ⟩,
  ⟨ C integer Q Input 24b ⟩,
  double *VarInf,
  double *ExpX,
  double *VarX,
  ⟨ C sumweights Input 25b ⟩,
  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 = R_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);
    R_Free(P_tmp);
  }
}
◇

```

Fragment referenced in 76a.

Defines: C\_VarianceLinearStatistic 34c, 43, 71a, 75a.

Uses: C\_CovarianceLinearStatistic 77, C\_kronecker 129b, mPQB 128a, P 23d, Q 24b, sumweights 25b.

### 3.8.2 Influence

```

> sumweights <- sum(weights[subset])
> expecty <- colSums(y[subset, ] * weights[subset]) / sumweights
> a0 <- expecty
> 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))

```

```

⟨ R_ExpectationInfluence Prototype 78b ⟩ ≡
SEXP R_ExpectationInfluence
(
  ⟨ R y Input 24a ⟩
  ⟨ R weights Input 24e ⟩,
  ⟨ R subset Input 25c ⟩
)
◇

```

Fragment referenced in 22a, 79a.

Uses: R\_ExpectationInfluence 79a.

```

⟨ R_ExpectationInfluence 79a ⟩ ≡
  ⟨ R_ExpectationInfluence Prototype 78b ⟩
  {
    SEXP ans;
    ⟨ C integer Q Input 24b ⟩;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;
    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 76a.

Defines: R\_ExpectationInfluence 78b, 81a, 149, 150.

Uses: N 23ab, NCOL 126b, Nsubset 25d, Offset0 21a, Q 24b, RC\_ExpectationInfluence 79c, RC\_Sums 88a, subset 25cf, 26a, sumweights 25b, weights 24ef, 25a, y 24acd.

```

⟨ RC_ExpectationInfluence Prototype 79b ⟩ ≡
  void RC_ExpectationInfluence
  (
    ⟨ C integer N Input 23b ⟩,
    ⟨ R y Input 24a ⟩
    ⟨ C integer Q Input 24b ⟩,
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩,
    ⟨ C sumweights Input 25b ⟩,
    ⟨ C colSums Answer 104c ⟩
  )
  ◇

```

Fragment referenced in 79c.

Uses: RC\_ExpectationInfluence 79c.

```

⟨ RC_ExpectationInfluence 79c ⟩ ≡
  ⟨ RC_ExpectationInfluence Prototype 79b ⟩
  {
    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 76a.

Defines: RC\_ExpectationInfluence 34a, 42b, 79ab.

Uses: DoCenter 21a, N 23ab, Nsubset 25d, offset 25e, Power1 21a, Q 24b, RC\_colSums 104a, subset 25cf, 26a, sumweights 25b, weights 24ef, 25a, y 24acd.

```

> 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)$CovarianceInfluence
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4) &&
+           isequal(a0, a5))
> a0 <- vary
> a1 <- .Call(libcoin:::R_CovarianceInfluence, y, weights, subset, 1L)
> a2 <- .Call(libcoin:::R_CovarianceInfluence, y, as.double(weights), as.double(subset), 1L)
> a3 <- .Call(libcoin:::R_CovarianceInfluence, y, weights, as.double(subset), 1L)
> a4 <- .Call(libcoin:::R_CovarianceInfluence, y, as.double(weights), subset, 1L)
> a5 <- LinStatExpCov(x, y, weights = weights, subset = subset, varonly = TRUE)$VarianceInfluence
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4) &&
+           isequal(a0, a5))

```

```

⟨ R_CovarianceInfluence Prototype 80 ⟩ ≡
  SEXP R_CovarianceInfluence
  (
    ⟨ R y Input 24a ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    SEXP varonly
  )
◇

```

Fragment referenced in [22a](#), [81a](#).  
 Uses: [R\\_CovarianceInfluence 81a](#).

```

⟨ R_CovarianceInfluence 81a ⟩ ≡
  ⟨ R_CovarianceInfluence Prototype 80 ⟩
  {
    SEXP ans;
    SEXP ExpInf;
    ⟨ C integer Q Input 24b ⟩;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;
    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 [76a](#).

Defines: [R\\_CovarianceInfluence 80](#), [149](#), [150](#).

Uses: [N 23ab](#), [NCOL 126b](#), [Nsubset 25d](#), [Offset0 21a](#), [Q 24b](#), [RC\\_CovarianceInfluence 82a](#), [RC\\_Sums 88a](#),  
[R\\_ExpectationInfluence 79a](#), [subset 25cf](#), [26a](#), [sumweights 25b](#), [weights 24ef](#), [25a](#), [y 24acd](#).

```

⟨ RC_CovarianceInfluence Prototype 81b ⟩ ≡
  void RC_CovarianceInfluence
  (
    ⟨ C integer N Input 23b ⟩,
    ⟨ R y Input 24a ⟩
    ⟨ C integer Q Input 24b ⟩,
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩,
    double *ExpInf,
    ⟨ C sumweights Input 25b ⟩,
    int VARONLY,
    ⟨ C KronSums Answer 92d ⟩
  )
  ◇

```

Fragment referenced in [82a](#).

Uses: [RC\\_CovarianceInfluence 82a](#).

```

⟨ RC_CovarianceInfluence 82a ⟩ ≡
  ⟨ RC_CovarianceInfluence Prototype 81b ⟩
  {
    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 76a.

Defines: RC\_CovarianceInfluence 34b, 43, 81ab.

Uses: DoCenter 21a, DoSymmetric 21a, N 23ab, Nsubset 25d, offset 25e, Power2 21a, Q 24b, RC\_colSums 104a, RC\_KronSums 92a, subset 25cf, 26a, sumweights 25b, weights 24ef, 25a, y 24acd.

### 3.8.3 X

```

⟨ R_ExpectationX Prototype 82b ⟩ ≡
  SEXP R_ExpectationX
  (
    ⟨ R x Input 23c ⟩
    SEXP P,
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩
  )
  ◇

```

Fragment referenced in 22a, 82c.

Uses: P 23d, R\_ExpectationX 82c.

```

⟨ R_ExpectationX 82c ⟩ ≡
  ⟨ R_ExpectationX Prototype 82b ⟩
  {
    SEXP ans;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;

    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 76a.

Defines: R\_ExpectationX 82b, 84b, 149, 150.

Uses: N 23ab, Nsubset 25d, Offset0 21a, P 23d, RC\_ExpectationX 83b, subset 25cf, 26a, weights 24ef, 25a, x 23cef.

```

⟨ RC_ExpectationX Prototype 83a ⟩ ≡
void RC_ExpectationX
(
  ⟨ R x Input 23c ⟩
  ⟨ C integer N Input 23b ⟩,
  ⟨ C integer P Input 23d ⟩,
  ⟨ R weights Input 24e ⟩,
  ⟨ R subset Input 25c ⟩,
  ⟨ C subset range Input 25e ⟩,
  ⟨ C OneTableSums Answer 108c ⟩
)
◇

```

Fragment referenced in [83b](#).  
 Uses: [RC\\_ExpectationX 83b](#).

```

⟨ RC_ExpectationX 83b ⟩ ≡
⟨ RC_ExpectationX Prototype 83a ⟩
{
  double center;

  if (TYPEOF(x) == INTSXP) {
    double* Pp1tmp = R_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];
    R_Free(Pp1tmp);
  } else {
    RC_colSums(REAL(x), N, P, Power1, &center, !DoCenter, weights, subset, offset, Nsubset, P_ans);
  }
}
◇

```

Fragment referenced in [76a](#).  
 Defines: [RC\\_ExpectationX 34a](#), [42b](#), [82c](#), [83a](#).  
 Uses: [DoCenter 21a](#), [N 23ab](#), [Nsubset 25d](#), [offset 25e](#), [P 23d](#), [Power1 21a](#), [RC\\_colSums 104a](#), [RC\\_OneTableSums 108a](#),  
[subset 25cf](#), [26a](#), [weights 24ef](#), [25a](#), [x 23cef](#).

```

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

```

```

⟨ R_CovarianceX Prototype 84a ⟩ ≡
  SEXP R_CovarianceX
  (
    ⟨ R x Input 23c ⟩
    SEXP P,
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    SEXP varonly
  )
  ◇

```

Fragment referenced in [22a](#), [84b](#).  
 Uses: P [23d](#), R\_CovarianceX [84b](#).

```

⟨ R_CovarianceX 84b ⟩ ≡
  ⟨ R_CovarianceX Prototype 84a ⟩
  {
    SEXP ans;
    SEXP ExpX;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;

    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 [76a](#).  
 Defines: R\_CovarianceX [84a](#), [149](#), [150](#).  
 Uses: N [23ab](#), Nsubset [25d](#), Offset0 [21a](#), P [23d](#), RC\_CovarianceX [85b](#), R\_ExpectationX [82c](#), subset [25cf](#), [26a](#), weights [24ef](#), [25a](#), x [23cef](#).

```

⟨ RC_CovarianceX Prototype 85a ⟩ ≡
void RC_CovarianceX
(
    ⟨ R x Input 23c ⟩
    ⟨ C integer N Input 23b ⟩,
    ⟨ C integer P Input 23d ⟩,
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩,
    double *ExpX,
    int VARONLY,
    ⟨ C KronSums Answer 92d ⟩
)
◇

```

Fragment referenced in [85b](#).  
 Uses: [RC\\_CovarianceX 85b](#).

```

⟨ RC_CovarianceX 85b ⟩ ≡
⟨ RC_CovarianceX Prototype 85a ⟩
{
    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 [76a](#).  
 Defines: [RC\\_CovarianceX 34cd](#), [43](#), [84b](#), [85a](#).  
 Uses: [DoCenter 21a](#), [DoSymmetric 21a](#), [N 23ab](#), [Nsubset 25d](#), [offset 25e](#), [P 23d](#), [Power2 21a](#), [PP12 127b](#), [RC\\_colSums 104a](#),  
[RC\\_KronSums 92a](#), [S 20c](#), [subset 25cf](#), [26a](#), [weights 24ef](#), [25a](#), [x 23cef](#).

### 3.9 Computing Sums

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



```

⟨ init subset loop 86a ⟩ ≡
  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 [90a](#), [96](#), [98b](#), [106a](#), [110a](#), [114a](#), [118a](#).  
 Uses: [N 23ab](#), [Nsubset 25d](#), [offset 25e](#), [subset 25cf](#), [26a](#), [weights 24ef](#), [25a](#).

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`

```

⟨ start subset loop 86b ⟩ ≡
  for (R_xlen_t i = 0; i < (Nsubset == 0 ? N : Nsubset) - 1; i++)
  ◇

```

Fragment referenced in [90a](#), [96](#), [98b](#), [106a](#), [110a](#), [114a](#), [118a](#).  
 Uses: [N 23ab](#), [Nsubset 25d](#).

After computations in the loop, we compute the next element

```

⟨ continue subset loop 86c ⟩ ≡
  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 [90a](#), [96](#), [98b](#), [106a](#), [110a](#), [114a](#), [118a](#).  
 Uses: [Nsubset 25d](#), [subset 25cf](#), [26a](#).

### 3.9.1 Simple Sums

```

⟨ SimpleSums 86d ⟩ ≡
  ⟨ C_Sums_dweights_dsubset 88b ⟩
  ⟨ C_Sums_iweights_dsubset 89a ⟩
  ⟨ C_Sums_iweights_isset 89b ⟩
  ⟨ C_Sums_dweights_isset 89c ⟩
  ⟨ RC_Sums 88a ⟩
  ⟨ R_Sums 87b ⟩
  ◇

```

Fragment referenced in [22c](#).

```

> 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))

```

```

⟨ R_Sums Prototype 87a ⟩ ≡
  SEXP R_Sums
  (
    ⟨ R N Input 23a ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩
  )
  ◇

```

Fragment referenced in [22a](#), [87b](#).  
 Uses: [R\\_Sums 87b](#).

```

⟨ R_Sums 87b ⟩ ≡
  ⟨ R_Sums Prototype 87a ⟩
  {
    SEXP ans;
    ⟨ C integer Nsubset Input 25d ⟩;

    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 [86d](#).  
 Defines: [R\\_Sums 87a](#), [149](#), [150](#).  
 Uses: [N 23ab](#), [Nsubset 25d](#), [Offset0 21a](#), [RC\\_Sums 88a](#), [subset 25cf](#), [26a](#), [weights 24ef](#), [25a](#).

```

⟨ RC_Sums Prototype 87c ⟩ ≡
  double RC_Sums
  (
    ⟨ C integer N Input 23b ⟩,
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩
  )
  ◇

```

Fragment referenced in [88a](#).  
 Uses: [RC\\_Sums 88a](#).

```

⟨ RC_Sums 88a ⟩ ≡
  ⟨ RC_Sums Prototype 87c ⟩
  {
    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 86d.

Defines: RC\_Sums 33ab, 79a, 81a, 87bc, 119b, 123a.

Uses: C\_Sums\_dweights\_dsubset 88b, C\_Sums\_dweights\_isubset 89c, C\_Sums\_iweights\_dsubset 89a,  
C\_Sums\_iweights\_isubset 89b, N 23ab, Nsubset 25d, offset 25e, subset 25cf, 26a, weights 24ef, 25a.

```

⟨ C_Sums_dweights_dsubset 88b ⟩ ≡
  double C_Sums_dweights_dsubset
  (
    ⟨ C integer N Input 23b ⟩,
    ⟨ C real weights Input 25a ⟩
    ⟨ C real subset Input 26a ⟩
  ) {
    double *s, *w;
    ⟨ Sums Body 90a ⟩
  }
  ◇

```

Fragment referenced in 86d.

Defines: C\_Sums\_dweights\_dsubset 88a.

```

⟨ C_Sums_iweights_dsubset 89a ⟩ ≡
double C_Sums_iweights_dsubset
(
    ⟨ C integer N Input 23b ⟩,
    ⟨ C integer weights Input 24f ⟩
    ⟨ C real subset Input 26a ⟩
) {
    double *s;
    int *w;
    ⟨ Sums Body 90a ⟩
}
◇

```

Fragment referenced in [86d](#).  
Defines: C\_Sums\_iweights\_dsubset [88a](#).

```

⟨ C_Sums_iweights_isubset 89b ⟩ ≡
double C_Sums_iweights_isubset
(
    ⟨ C integer N Input 23b ⟩,
    ⟨ C integer weights Input 24f ⟩
    ⟨ C integer subset Input 25f ⟩
) {
    int *s, *w;
    ⟨ Sums Body 90a ⟩
}
◇

```

Fragment referenced in [86d](#).  
Defines: C\_Sums\_iweights\_isubset [88a](#).

```

⟨ C_Sums_dweights_isubset 89c ⟩ ≡
double C_Sums_dweights_isubset
(
    ⟨ C integer N Input 23b ⟩,
    ⟨ C real weights Input 25a ⟩
    ⟨ C integer subset Input 25f ⟩
) {
    int *s;
    double *w;
    ⟨ Sums Body 90a ⟩
}
◇

```

Fragment referenced in [86d](#).  
Defines: C\_Sums\_dweights\_isubset [88a](#).

```

⟨ Sums Body 90a ⟩ ≡
double ans = 0.0;

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

⟨ init subset loop 86a ⟩
⟨ start subset loop 86b ⟩
{
  w = w + diff;
  ans += w[0];
  ⟨ continue subset loop 86c ⟩
}
w = w + diff;
ans += w[0];

return(ans);
◇

```

Fragment referenced in [88b](#), [89abc](#).

Uses: HAS\_WEIGHTS [24f](#), [25a](#), N [23ab](#), Nsubset [25d](#).

### 3.9.2 Kronecker Sums

```

⟨ KronSums 90b ⟩ ≡
⟨ C_KronSums_dweights_dsubset 94a ⟩
⟨ C_KronSums_iweights_dsubset 94b ⟩
⟨ C_KronSums_iweights_isubset 94c ⟩
⟨ C_KronSums_dweights_isubset 95 ⟩
⟨ C_XfactorKronSums_dweights_dsubset 97b ⟩
⟨ C_XfactorKronSums_iweights_dsubset 97c ⟩
⟨ C_XfactorKronSums_iweights_isubset 97d ⟩
⟨ C_XfactorKronSums_dweights_isubset 98a ⟩
⟨ RC_KronSums 92a ⟩
⟨ R_KronSums 91b ⟩
⟨ C_KronSums_Permutation_isubset 101b ⟩
⟨ C_KronSums_Permutation_dsubset 101a ⟩
⟨ C_XfactorKronSums_Permutation_isubset 102b ⟩
⟨ C_XfactorKronSums_Permutation_dsubset 102a ⟩
⟨ RC_KronSums_Permutation 100b ⟩
⟨ R_KronSums_Permutation 99b ⟩
◇

```

Fragment referenced in [22c](#).

```

> 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))
```

```
 $\langle R\_KronSums \text{ Prototype } 91a \rangle \equiv$ 
  SEXP R_KronSums
  (
     $\langle R \text{ } x \text{ Input } 23c \rangle$ 
    SEXP P,
     $\langle R \text{ } y \text{ Input } 24a \rangle$ 
     $\langle R \text{ } weights \text{ Input } 24e \rangle$ ,
     $\langle R \text{ } subset \text{ Input } 25c \rangle$ ,
    SEXP symmetric
  )
   $\diamond$ 
```

Fragment referenced in [22a](#), [91b](#).

Uses: P [23d](#), R\_KronSums [91b](#).

```
 $\langle R\_KronSums \text{ } 91b \rangle \equiv$ 
 $\langle R\_KronSums \text{ Prototype } 91a \rangle$ 
{
  SEXP ans;
   $\langle C \text{ integer } Q \text{ Input } 24b \rangle$ ;
   $\langle C \text{ integer } N \text{ Input } 23b \rangle$ ;
   $\langle C \text{ integer } Nsubset \text{ Input } 25d \rangle$ ;

  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);
}
 $\diamond$ 
```

Fragment referenced in [90b](#).

Defines: R\_KronSums [91a](#), [149](#), [150](#).

Uses: DoCenter [21a](#), N [23ab](#), NCOL [126b](#), Nsubset [25d](#), Offset0 [21a](#), P [23d](#), Q [24b](#), RC\_KronSums [92a](#), subset [25cf](#), [26a](#), weights [24ef](#), [25a](#), x [23cef](#), y [24acd](#).

```
 $\langle RC\_KronSums \text{ Prototype } 91c \rangle \equiv$ 
  void RC_KronSums
  (
     $\langle RC \text{ KronSums Input } 92b \rangle$ 
     $\langle R \text{ } weights \text{ Input } 24e \rangle$ ,
     $\langle R \text{ } subset \text{ Input } 25c \rangle$ ,
     $\langle C \text{ subset range Input } 25e \rangle$ ,
     $\langle C \text{ KronSums Answer } 92d \rangle$ 
  )
   $\diamond$ 
```

Fragment referenced in [92a](#).

Uses: RC\_KronSums [92a](#).

```

⟨ RC_KronSums 92a ⟩ ≡
  ⟨ RC_KronSums Prototype 91c ⟩
  {
    if (typeof(x) == INTSXP) {
      ⟨ KronSums Integer x 93a ⟩
    } else {
      ⟨ KronSums Double x 93b ⟩
    }
  }
  ◇

```

Fragment referenced in 90b.

Defines: *RC\_KronSums* 75d, 82a, 85b, 91bc.

Uses: *x* 23cef.

```

⟨ RC KronSums Input 92b ⟩ ≡
  ⟨ R x Input 23c ⟩
  ⟨ C integer N Input 23b ⟩,
  ⟨ C integer P Input 23d ⟩,
  ⟨ C real y Input 24c ⟩
  const int SYMMETRIC,
  double *centerx,
  double *centery,
  const int CENTER,
  ◇

```

Fragment referenced in 91c.

```

⟨ C KronSums Input 92c ⟩ ≡
  ⟨ C real x Input 23e ⟩
  ⟨ C real y Input 24c ⟩
  const int SYMMETRIC,
  double *centerx,
  double *centery,
  const int CENTER,
  ◇

```

Fragment referenced in 94abc, 95.

```

⟨ C KronSums Answer 92d ⟩ ≡
  double *PQ_ans
  ◇

```

Fragment referenced in 75c, 81b, 85a, 91c, 94abc, 95, 97bcd, 98a, 100a, 101ab, 102ab.

$\langle \text{KronSums Integer } x \text{ 93a} \rangle \equiv$

```

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 [92a](#).

Uses: C\_XfactorKronSums\_dweights\_dsubset [97b](#), C\_XfactorKronSums\_dweights\_isubset [98a](#),  
C\_XfactorKronSums\_iweights\_dsubset [97c](#), C\_XfactorKronSums\_iweights\_isubset [97d](#), N [23ab](#), Nsubset [25d](#),  
offset [25e](#), P [23d](#), Q [24b](#), subset [25cf](#), [26a](#), weights [24ef](#), [25a](#), x [23cef](#), y [24acd](#).

$\langle \text{KronSums Double } x \text{ 93b} \rangle \equiv$

```

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 [92a](#).

Uses: C\_KronSums\_dweights\_dsubset [94a](#), C\_KronSums\_dweights\_isubset [95](#), C\_KronSums\_iweights\_dsubset [94b](#),  
C\_KronSums\_iweights\_isubset [94c](#), N [23ab](#), Nsubset [25d](#), offset [25e](#), P [23d](#), Q [24b](#), subset [25cf](#), [26a](#), weights [24ef](#),  
[25a](#), x [23cef](#), y [24acd](#).



```

⟨ C_KronSums_dweights_dsubset 94a ⟩ ≡
void C_KronSums_dweights_dsubset
(
    ⟨ C_KronSums Input 92c ⟩
    ⟨ C real weights Input 25a ⟩
    ⟨ C real subset Input 26a ⟩,
    ⟨ C_KronSums Answer 92d ⟩
) {
    double *s, *w;
    ⟨ KronSums Body 96 ⟩
}
◇

```

Fragment referenced in 90b.

Defines: C\_KronSums\_dweights\_dsubset 93b.

```

⟨ C_KronSums_iweights_dsubset 94b ⟩ ≡
void C_KronSums_iweights_dsubset
(
    ⟨ C_KronSums Input 92c ⟩
    ⟨ C integer weights Input 24f ⟩
    ⟨ C real subset Input 26a ⟩,
    ⟨ C_KronSums Answer 92d ⟩
) {
    double *s;
    int *w;
    ⟨ KronSums Body 96 ⟩
}
◇

```

Fragment referenced in 90b.

Defines: C\_KronSums\_iweights\_dsubset 93b.

```

⟨ C_KronSums_iweights_isubset 94c ⟩ ≡
void C_KronSums_iweights_isubset
(
    ⟨ C_KronSums Input 92c ⟩
    ⟨ C integer weights Input 24f ⟩
    ⟨ C integer subset Input 25f ⟩,
    ⟨ C_KronSums Answer 92d ⟩
) {
    int *s, *w;
    ⟨ KronSums Body 96 ⟩
}
◇

```

Fragment referenced in 90b.

Defines: C\_KronSums\_iweights\_isubset 93b.

```

⟨ C_KronSums_dweights_isubset 95 ⟩ ≡
  void C_KronSums_dweights_isubset
  (
    ⟨ C_KronSums_Input 92c ⟩
    ⟨ C_real_weights_Input 25a ⟩
    ⟨ C_integer_subset_Input 25f ⟩,
    ⟨ C_KronSums_Answer 92d ⟩
  ) {
    int *s;
    double *w;
    ⟨ KronSums_Body 96 ⟩
  }
  ◇

```

Fragment referenced in [90b](#).

Defines: `C_KronSums_dweights_isubset` [93b](#).

```

⟨ KronSums Body 96 ⟩ ≡
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];
    }
    ⟨ init subset loop 86a ⟩
    ⟨ start subset loop 86b ⟩
    {
      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];
        }
      }
      ⟨ continue subset loop 86c ⟩
    }
    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 [94abc](#), [95](#).

Uses: HAS\_WEIGHTS [24f](#), [25a](#), N [23ab](#), P [23d](#), Q [24b](#), S [20c](#), x [23cef](#), y [24acd](#).

## Xfactor Kronecker Sums

$\langle C \text{ XfactorKronSums Input 97a} \rangle \equiv$   
     $\langle C \text{ integer } x \text{ Input 23f} \rangle$   
     $\langle C \text{ real } y \text{ Input 24c} \rangle$   
     $\diamond$

Fragment referenced in [97bcd](#), [98a](#).

$\langle C\_XfactorKronSums\_dweights\_dsubset \text{ 97b} \rangle \equiv$   
    void C\_XfactorKronSums\_dweights\_dsubset  
    (  
         $\langle C \text{ XfactorKronSums Input 97a} \rangle$   
         $\langle C \text{ real weights Input 25a} \rangle$   
         $\langle C \text{ real subset Input 26a} \rangle$ ,  
         $\langle C \text{ KronSums Answer 92d} \rangle$   
    ) {  
        double \*s, \*w;  
         $\langle XfactorKronSums Body \text{ 98b} \rangle$   
    }  
     $\diamond$

Fragment referenced in [90b](#).

Defines: C\_XfactorKronSums\_dweights\_dsubset [93a](#).

$\langle C\_XfactorKronSums\_iweights\_dsubset \text{ 97c} \rangle \equiv$   
    void C\_XfactorKronSums\_iweights\_dsubset  
    (  
         $\langle C \text{ XfactorKronSums Input 97a} \rangle$   
         $\langle C \text{ integer weights Input 24f} \rangle$   
         $\langle C \text{ real subset Input 26a} \rangle$ ,  
         $\langle C \text{ KronSums Answer 92d} \rangle$   
    ) {  
        double \*s;  
        int \*w;  
         $\langle XfactorKronSums Body \text{ 98b} \rangle$   
    }  
     $\diamond$

Fragment referenced in [90b](#).

Defines: C\_XfactorKronSums\_iweights\_dsubset [93a](#).

$\langle C\_XfactorKronSums\_iweights\_isubset \text{ 97d} \rangle \equiv$   
    void C\_XfactorKronSums\_iweights\_isubset  
    (  
         $\langle C \text{ XfactorKronSums Input 97a} \rangle$   
         $\langle C \text{ integer weights Input 24f} \rangle$   
         $\langle C \text{ integer subset Input 25f} \rangle$ ,  
         $\langle C \text{ KronSums Answer 92d} \rangle$   
    ) {  
        int \*s, \*w;  
         $\langle XfactorKronSums Body \text{ 98b} \rangle$   
    }  
     $\diamond$

Fragment referenced in [90b](#).

Defines: C\_XfactorKronSums\_iweights\_isubset [93a](#).

```

⟨ C_XfactorKronSums_dweights_isubset 98a ⟩ ≡
void C_XfactorKronSums_dweights_isubset
(
    ⟨ C_XfactorKronSums Input 97a ⟩
    ⟨ C_real weights Input 25a ⟩
    ⟨ C_integer subset Input 25f ⟩,
    ⟨ C_KronSums Answer 92d ⟩
) {
    int *s;
    double *w;
    ⟨ XfactorKronSums Body 98b ⟩
}
◇

```

Fragment referenced in [90b](#).

Defines: `C_XfactorKronSums_dweights_isubset` [93a](#).

```

⟨ XfactorKronSums Body 98b ⟩ ≡
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;
    ⟨ init_subset loop 86a ⟩
    ⟨ start_subset loop 86b ⟩
    {
        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];
        }
        ⟨ continue_subset loop 86c ⟩
    }
    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 [97bcd](#), [98a](#).

Uses: `HAS_WEIGHTS` [24f](#), [25a](#), `mPQB` [128a](#), `N` [23ab](#), `P` [23d](#), `Q` [24b](#), `x` [23cef](#), `y` [24acd](#).

## 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))
```

```
< R_KronSums_Permutation Prototype 99a > ≡
  SEXP R_KronSums_Permutation
  (
    < R x Input 23c >
    SEXP P,
    < R y Input 24a >
    < R subset Input 25c >,
    SEXP subsety
  )
  ◇
```

Fragment referenced in [22a](#), [99b](#).

Uses: P [23d](#), R\_KronSums\_Permutation [99b](#).

```
< R_KronSums_Permutation 99b > ≡
  < R_KronSums_Permutation Prototype 99a >
  {
    SEXP ans;
    < C integer Q Input 24b >;
    < C integer N Input 23b >;
    < C integer Nsubset Input 25d >;

    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 [90b](#).

Defines: R\_KronSums\_Permutation [99a](#), [149](#), [150](#).

Uses: N [23ab](#), NCOL [126b](#), Nsubset [25d](#), Offset0 [21a](#), P [23d](#), Q [24b](#), RC\_KronSums\_Permutation [100b](#), subset [25cf](#), [26a](#), x [23cef](#), y [24acd](#).

```

⟨ RC_KronSums_Permutation Prototype 100a ⟩ ≡
void RC_KronSums_Permutation
(
    ⟨ R x Input 23c ⟩
    ⟨ C integer N Input 23b ⟩,
    ⟨ C integer P Input 23d ⟩,
    ⟨ C real y Input 24c ⟩
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩,
    SEXP subsety,
    ⟨ C KronSums Answer 92d ⟩
)
◇

```

Fragment referenced in [100b](#).  
 Uses: RC\_KronSums\_Permutation [100b](#).

```

⟨ RC_KronSums_Permutation 100b ⟩ ≡
⟨ RC_KronSums_Permutation Prototype 100a ⟩
{
    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 [90b](#).  
 Defines: RC\_KronSums\_Permutation [36](#), [99b](#), [100a](#).  
 Uses: C\_KronSums\_Permutation\_dsubset [101a](#), C\_KronSums\_Permutation\_isubset [101b](#),  
 C\_XfactorKronSums\_Permutation\_dsubset [102a](#), C\_XfactorKronSums\_Permutation\_isubset [102b](#), N [23ab](#),  
 Nsubset [25d](#), offset [25e](#), P [23d](#), Q [24b](#), subset [25cf](#), [26a](#), x [23cef](#), y [24acd](#).

```

⟨ C_KronSums_Permutation_dsubset 101a ⟩ ≡
void C_KronSums_Permutation_dsubset
(
    ⟨ C real x Input 23e ⟩
    ⟨ C real y Input 24c ⟩
    ⟨ C real subset Input 26a ⟩,
    double *subsetsy,
    ⟨ C KronSums Answer 92d ⟩
) {
    ⟨ KronSums Permutation Body 101c ⟩
}
◇

```

Fragment referenced in [90b](#).

Defines: C\_KronSums\_Permutation\_dsubset [100b](#).

```

⟨ C_KronSums_Permutation_isubset 101b ⟩ ≡
void C_KronSums_Permutation_isubset
(
    ⟨ C real x Input 23e ⟩
    ⟨ C real y Input 24c ⟩
    ⟨ C integer subset Input 25f ⟩,
    int *subsetsy,
    ⟨ C KronSums Answer 92d ⟩
) {
    ⟨ KronSums Permutation Body 101c ⟩
}
◇

```

Fragment referenced in [90b](#).

Defines: C\_KronSums\_Permutation\_isubset [100b](#).

Because `subset` might not be ordered (in the presence of blocks) we have to go through all elements explicitly here.

```

⟨ KronSums Permutation Body 101c ⟩ ≡
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) subsetsy[i] - 1] *
                           x[pN + (R_xlen_t) subset[i] - 1];
    }
}
◇

```

Fragment referenced in [101ab](#).

Uses: N [23ab](#), Nsubset [25d](#), offset [25e](#), P [23d](#), Q [24b](#), subset [25cf](#), 26a, x [23cef](#), y [24acd](#).



## Xfactor Permuted Kronecker Sums

```

⟨ C_XfactorKronSums_Permutation_dsubset 102a ⟩ ≡
    void C_XfactorKronSums_Permutation_dsubset
    (
        ⟨ C integer x Input 23f ⟩
        ⟨ C real y Input 24c ⟩
        ⟨ C real subset Input 26a ⟩,
        double *subsets,
        ⟨ C KronSums Answer 92d ⟩
    ) {
        ⟨ XfactorKronSums Permutation Body 102c ⟩
    }
    ◇

```

Fragment referenced in [90b](#).

Defines: C\_XfactorKronSums\_Permutation\_dsubset [100b](#).

```

⟨ C_XfactorKronSums_Permutation_isubset 102b ⟩ ≡
    void C_XfactorKronSums_Permutation_isubset
    (
        ⟨ C integer x Input 23f ⟩
        ⟨ C real y Input 24c ⟩
        ⟨ C integer subset Input 25f ⟩,
        int *subsets,
        ⟨ C KronSums Answer 92d ⟩
    ) {
        ⟨ XfactorKronSums Permutation Body 102c ⟩
    }
    ◇

```

Fragment referenced in [90b](#).

Defines: C\_XfactorKronSums\_Permutation\_isubset [100b](#).

```

⟨ XfactorKronSums Permutation Body 102c ⟩ ≡
    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) subsets[i] - 1];
    }
    ◇

```

Fragment referenced in [102ab](#).

Uses: mPQB [128a](#), N [23ab](#), Nsubset [25d](#), offset [25e](#), P [23d](#), Q [24b](#), subset [25cf](#), [26a](#), x [23cef](#), y [24acd](#).

## 3.9.3 Column Sums

```

⟨ colSums 102d ⟩ ≡
    ⟨ C_colSums_dweights_dsubset 104d ⟩
    ⟨ C_colSums_weights_dsubset 105a ⟩
    ⟨ C_colSums_weights_isubset 105b ⟩
    ⟨ C_colSums_dweights_isubset 105c ⟩
    ⟨ RC_colSums 104a ⟩
    ⟨ R_colSums 103b ⟩
    ◇

```

Fragment referenced in [22c](#).

```

> 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))

```

```

⟨ R_colSums Prototype 103a ⟩ ≡
  SEXP R_colSums
  (
    ⟨ R x Input 23c ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩
  )
◇

```

Fragment referenced in [22a](#), [103b](#).  
 Uses: `R_colSums` [103b](#).

```

⟨ R_colSums 103b ⟩ ≡
  ⟨ R_colSums Prototype 103a ⟩
  {
    SEXP ans;
    int P;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;
    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 [102d](#).  
 Defines: `R_colSums` [103a](#), [149](#), [150](#).  
 Uses: `DoCenter` [21a](#), `N` [23ab](#), `NCOL` [126b](#), `Nsubset` [25d](#), `Offset0` [21a](#), `P` [23d](#), `Power1` [21a](#), `RC_colSums` [104a](#), `subset` [25cf](#), [26a](#),  
`weights` [24ef](#), [25a](#), `x` [23cef](#).

```

⟨ RC_colSums Prototype 103c ⟩ ≡
  void RC_colSums
  (
    ⟨ C colSums Input 104b ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩,
    ⟨ C colSums Answer 104c ⟩
  )
◇

```

Fragment referenced in [104a](#).  
 Uses: `RC_colSums` [104a](#).

```

⟨ RC_colSums 104a ⟩ ≡
  ⟨ RC_colSums Prototype 103c ⟩
  {
    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 102d.

Defines: RC\_colSums 79c, 82a, 83b, 85b, 103bc.

Uses: C\_colSums\_dweights\_dsubset 104d, C\_colSums\_dweights\_isubset 105c, C\_colSums\_iweights\_dsubset 105a, C\_colSums\_iweights\_isubset 105b, N 23ab, Nsubset 25d, offset 25e, P 23d, subset 25cf, 26a, weights 24ef, 25a, x 23cef.

```

⟨ C_colSums Input 104b ⟩ ≡
  ⟨ C real x Input 23e ⟩
  const int power,
  double *centerx,
  const int CENTER,
  ◇

```

Fragment referenced in 103c, 104d, 105abc.

```

⟨ C_colSums Answer 104c ⟩ ≡
  double *P_ans
  ◇

```

Fragment referenced in 79b, 103c, 104d, 105abc.

```

⟨ C_colSums_dweights_dsubset 104d ⟩ ≡
  void C_colSums_dweights_dsubset
  (
    ⟨ C_colSums Input 104b ⟩
    ⟨ C real weights Input 25a ⟩
    ⟨ C real subset Input 26a ⟩,
    ⟨ C_colSums Answer 104c ⟩
  ) {
    double *s, *w;
    ⟨ colSums Body 106a ⟩
  }
  ◇

```

Fragment referenced in 102d.

Defines: C\_colSums\_dweights\_dsubset 104a.

```

⟨ C_colSums_iweights_dsubset 105a ⟩ ≡
void C_colSums_iweights_dsubset
(
    ⟨ C_colSums Input 104b ⟩
    ⟨ C_integer weights Input 24f ⟩
    ⟨ C_real subset Input 26a ⟩,
    ⟨ C_colSums Answer 104c ⟩
) {
    double *s;
    int *w;
    ⟨ colSums Body 106a ⟩
}
◇

```

Fragment referenced in [102d](#).  
Defines: C\_colSums\_iweights\_dsubset [104a](#).

```

⟨ C_colSums_iweights_isubset 105b ⟩ ≡
void C_colSums_iweights_isubset
(
    ⟨ C_colSums Input 104b ⟩
    ⟨ C_integer weights Input 24f ⟩
    ⟨ C_integer subset Input 25f ⟩,
    ⟨ C_colSums Answer 104c ⟩
) {
    int *s, *w;
    ⟨ colSums Body 106a ⟩
}
◇

```

Fragment referenced in [102d](#).  
Defines: C\_colSums\_iweights\_isubset [104a](#).

```

⟨ C_colSums_dweights_isubset 105c ⟩ ≡
void C_colSums_dweights_isubset
(
    ⟨ C_colSums Input 104b ⟩
    ⟨ C_real weights Input 25a ⟩
    ⟨ C_integer subset Input 25f ⟩,
    ⟨ C_colSums Answer 104c ⟩
) {
    int *s;
    double *w;
    ⟨ colSums Body 106a ⟩
}
◇

```

Fragment referenced in [102d](#).  
Defines: C\_colSums\_dweights\_isubset [104a](#).

```

⟨ colSums Body 106a ⟩ ≡
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];
  }
  ⟨ init subset loop 86a ⟩
  ⟨ start subset loop 86b ⟩
  {
    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);
    }
    ⟨ continue subset loop 86c ⟩
  }
  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 [104d](#), [105abc](#).

Uses: HAS\_WEIGHTS [24f](#), [25a](#), N [23ab](#), P [23d](#), x [23cef](#).

### 3.9.4 Tables

#### OneTable Sums

```

⟨ Tables 106b ⟩ ≡
  ⟨ C_OneTableSums_dweights_dsubset 108d ⟩
  ⟨ C_OneTableSums_iweights_dsubset 109a ⟩
  ⟨ C_OneTableSums_iweights_isubset 109b ⟩
  ⟨ C_OneTableSums_dweights_isubset 109c ⟩
  ⟨ RC_OneTableSums 108a ⟩
  ⟨ R_OneTableSums 107b ⟩
  ⟨ C_TwoTableSums_dweights_dsubset 112d ⟩
  ⟨ C_TwoTableSums_iweights_dsubset 113a ⟩
  ⟨ C_TwoTableSums_iweights_isubset 113b ⟩
  ⟨ C_TwoTableSums_dweights_isubset 113c ⟩
  ⟨ RC_TwoTableSums 112a ⟩
  ⟨ R_TwoTableSums 111a ⟩
  ⟨ C_ThreeTableSums_dweights_dsubset 116d ⟩
  ⟨ C_ThreeTableSums_iweights_dsubset 117a ⟩
  ⟨ C_ThreeTableSums_iweights_isubset 117b ⟩
  ⟨ C_ThreeTableSums_dweights_isubset 117c ⟩
  ⟨ RC_ThreeTableSums 116a ⟩
  ⟨ R_ThreeTableSums 115a ⟩
◇

```

Fragment referenced in [22c](#).

```
> a0 <- as.vector(xtabs(weights ~ ixf, subset = subset))
```

```

> a1 <- ctab(ix, weights = weights, subset = subset)[-1]
> a2 <- ctab(ix, weights = as.double(weights), subset = as.double(subset))[-1]
> a3 <- ctab(ix, weights = weights, subset = as.double(subset))[-1]
> a4 <- ctab(ix, weights = as.double(weights), subset = subset)[-1]
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))

```

```

⟨ R_OneTableSums Prototype 107a ⟩ ≡
  SEXP R_OneTableSums
  (
    ⟨ R x Input 23c ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩
  )
◇

```

Fragment referenced in [22a](#), [107b](#).  
 Uses: `R_OneTableSums` [107b](#).

```

⟨ R_OneTableSums 107b ⟩ ≡
  ⟨ R_OneTableSums Prototype 107a ⟩
  {
    SEXP ans;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;
    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 [106b](#).  
 Defines: `R_OneTableSums` [15b](#), [107a](#), [119b](#), [149](#), [150](#).  
 Uses: `N` [23ab](#), `NLEVELS` [127a](#), `Nsubset` [25d](#), `Offset0` [21a](#), `P` [23d](#), `RC_OneTableSums` [108a](#), `subset` [25cf](#), [26a](#), `weights` [24ef](#), [25a](#),  
`x` [23cef](#).

```

⟨ RC_OneTableSums Prototype 107c ⟩ ≡
  void RC_OneTableSums
  (
    ⟨ C OneTableSums Input 108b ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩,
    ⟨ C OneTableSums Answer 108c ⟩
  )
◇

```

Fragment referenced in [108a](#).  
 Uses: `RC_OneTableSums` [108a](#).

```

⟨ RC_OneTableSums 108a ⟩ ≡
  ⟨ RC_OneTableSums Prototype 107c ⟩
  {
    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 106b.

Defines: RC\_OneTableSums 33a, 36, 83b, 107bc.

Uses: C\_OneTableSums\_dweights\_dsubset 108d, C\_OneTableSums\_dweights\_isubset 109c,  
 C\_OneTableSums\_iweights\_dsubset 109a, C\_OneTableSums\_iweights\_isubset 109b, N 23ab, Nsubset 25d,  
 offset 25e, P 23d, subset 25cf, 26a, weights 24ef, 25a, x 23cef.

```

⟨ C_OneTableSums Input 108b ⟩ ≡
  ⟨ C integer x Input 23f ⟩
  ◇

```

Fragment referenced in 107c, 108d, 109abc.

```

⟨ C_OneTableSums Answer 108c ⟩ ≡
  double *P_ans
  ◇

```

Fragment referenced in 83a, 107c, 108d, 109abc.

```

⟨ C_OneTableSums_dweights_dsubset 108d ⟩ ≡
  void C_OneTableSums_dweights_dsubset
  (
    ⟨ C_OneTableSums Input 108b ⟩
    ⟨ C real weights Input 25a ⟩
    ⟨ C real subset Input 26a ⟩,
    ⟨ C_OneTableSums Answer 108c ⟩
  ) {
    double *s, *w;
    ⟨ OneTableSums Body 110a ⟩
  }
  ◇

```

Fragment referenced in 106b.

Defines: C\_OneTableSums\_dweights\_dsubset 108a.

```

⟨ C_OneTableSums_iweights_dsubset 109a ⟩ ≡
void C_OneTableSums_iweights_dsubset
(
    ⟨ C_OneTableSums Input 108b ⟩
    ⟨ C_integer weights Input 24f ⟩
    ⟨ C_real subset Input 26a ⟩,
    ⟨ C_OneTableSums Answer 108c ⟩
) {
    double *s;
    int *w;
    ⟨ OneTableSums Body 110a ⟩
}
◇

```

Fragment referenced in [106b](#).

Defines: C\_OneTableSums\_iweights\_dsubset [108a](#).

```

⟨ C_OneTableSums_iweights_isubset 109b ⟩ ≡
void C_OneTableSums_iweights_isubset
(
    ⟨ C_OneTableSums Input 108b ⟩
    ⟨ C_integer weights Input 24f ⟩
    ⟨ C_integer subset Input 25f ⟩,
    ⟨ C_OneTableSums Answer 108c ⟩
) {
    int *s, *w;
    ⟨ OneTableSums Body 110a ⟩
}
◇

```

Fragment referenced in [106b](#).

Defines: C\_OneTableSums\_iweights\_isubset [108a](#).

```

⟨ C_OneTableSums_dweights_isubset 109c ⟩ ≡
void C_OneTableSums_dweights_isubset
(
    ⟨ C_OneTableSums Input 108b ⟩
    ⟨ C_real weights Input 25a ⟩
    ⟨ C_integer subset Input 25f ⟩,
    ⟨ C_OneTableSums Answer 108c ⟩
) {
    int *s;
    double *w;
    ⟨ OneTableSums Body 110a ⟩
}
◇

```

Fragment referenced in [106b](#).

Defines: C\_OneTableSums\_dweights\_isubset [108a](#).



```

⟨ OneTableSums Body 110a ⟩ ≡
  int *xx;

  for (int p = 0; p < P; p++) P_ans[p] = 0.0;

  xx = x;
  ⟨ init subset loop 86a ⟩
  ⟨ start subset loop 86b ⟩
  {
    xx = xx + diff;
    if (HAS_WEIGHTS) {
      w = w + diff;
      P_ans[xx[0]] += (double) w[0];
    } else {
      P_ans[xx[0]]++;
    }
    ⟨ continue subset loop 86c ⟩
  }
  xx = xx + diff;
  if (HAS_WEIGHTS) {
    w = w + diff;
    P_ans[xx[0]] += w[0];
  } else {
    P_ans[xx[0]]++;
  }
  ◇

```

Fragment referenced in [108d](#), [109abc](#).  
 Uses: HAS\_WEIGHTS [24f](#), [25a](#), P [23d](#), x [23cef](#).

## 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))

```

```

⟨ R_TwoTableSums Prototype 110b ⟩ ≡
  SEXP R_TwoTableSums
  (
    ⟨ R x Input 23c ⟩
    ⟨ R y Input 24a ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩
  )
  ◇

```

Fragment referenced in [22a](#), [111a](#).  
 Uses: R\_TwoTableSums [111a](#).

```

⟨ R_TwoTableSums 111a ⟩ ≡
  ⟨ R_TwoTableSums Prototype 110b ⟩
  {
    SEXP ans, dim;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;
    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 106b.

Defines: *R\_TwoTableSums* 15b, 110b, 149, 150.

Uses: *mPQB* 128a, *N* 23ab, *NLEVELS* 127a, *Nsubset* 25d, *Offset0* 21a, *P* 23d, *Q* 24b, *RC\_TwoTableSums* 112a, *subset* 25cf, 26a, *weights* 24ef, 25a, *x* 23cef, *y* 24acd.

```

⟨ RC_TwoTableSums Prototype 111b ⟩ ≡
  void RC_TwoTableSums
  (
    ⟨ C TwoTableSums Input 112b ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩,
    ⟨ C TwoTableSums Answer 112c ⟩
  )
  ◇

```

Fragment referenced in 112a.

Uses: *RC\_TwoTableSums* 112a.

```

⟨ RC_TwoTableSums 112a ⟩ ≡
  ⟨ RC_TwoTableSums Prototype 111b ⟩
  {
    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 106b.

Defines: *RC\_TwoTableSums* 40a, 111ab.

Uses: *C\_TwoTableSums\_dweights\_dsubset* 112d, *C\_TwoTableSums\_dweights\_isubset* 113c,  
*C\_TwoTableSums\_iweights\_dsubset* 113a, *C\_TwoTableSums\_iweights\_isubset* 113b, *N* 23ab, *Nsubset* 25d,  
*offset* 25e, *P* 23d, *Q* 24b, *subset* 25cf, 26a, *weights* 24ef, 25a, *x* 23cef, *y* 24acd.

```

⟨ C_TwoTableSums Input 112b ⟩ ≡
  ⟨ C integer x Input 23f ⟩
  ⟨ C integer y Input 24d ⟩
  ◇

```

Fragment referenced in 111b, 112d, 113abc.

```

⟨ C_TwoTableSums Answer 112c ⟩ ≡
  double *PQ_ans
  ◇

```

Fragment referenced in 111b, 112d, 113abc.

```

⟨ C_TwoTableSums__dweights_dsubset 112d ⟩ ≡
  void C_TwoTableSums__dweights_dsubset
  (
    ⟨ C_TwoTableSums Input 112b ⟩
    ⟨ C real weights Input 25a ⟩
    ⟨ C real subset Input 26a ⟩,
    ⟨ C_TwoTableSums Answer 112c ⟩
  ) {
    double *s, *w;
    ⟨ TwoTableSums Body 114a ⟩
  }
  ◇

```

Fragment referenced in 106b.

Defines: *C\_TwoTableSums\_\_dweights\_dsubset* 112a.

```

⟨ C_TwoTableSums_iweights_dsubset 113a ⟩ ≡
void C_TwoTableSums_iweights_dsubset
(
    ⟨ C_TwoTableSums Input 112b ⟩
    ⟨ C_integer weights Input 24f ⟩
    ⟨ C_real subset Input 26a ⟩,
    ⟨ C_TwoTableSums Answer 112c ⟩
) {
    double *s;
    int *w;
    ⟨ TwoTableSums Body 114a ⟩
}
◇

```

Fragment referenced in 106b.

Defines: C\_TwoTableSums\_iweights\_dsubset 112a.

```

⟨ C_TwoTableSums_iweights_isubset 113b ⟩ ≡
void C_TwoTableSums_iweights_isubset
(
    ⟨ C_TwoTableSums Input 112b ⟩
    ⟨ C_integer weights Input 24f ⟩
    ⟨ C_integer subset Input 25f ⟩,
    ⟨ C_TwoTableSums Answer 112c ⟩
) {
    int *s, *w;
    ⟨ TwoTableSums Body 114a ⟩
}
◇

```

Fragment referenced in 106b.

Defines: C\_TwoTableSums\_iweights\_isubset 112a.

```

⟨ C_TwoTableSums_dweights_isubset 113c ⟩ ≡
void C_TwoTableSums_dweights_isubset
(
    ⟨ C_TwoTableSums Input 112b ⟩
    ⟨ C_real weights Input 25a ⟩
    ⟨ C_integer subset Input 25f ⟩,
    ⟨ C_TwoTableSums Answer 112c ⟩
) {
    int *s;
    double *w;
    ⟨ TwoTableSums Body 114a ⟩
}
◇

```

Fragment referenced in 106b.

Defines: C\_TwoTableSums\_dweights\_isubset 112a.

```

⟨ TwoTableSums Body 114a ⟩ ≡
    int *xx, *yy;

    for (int p = 0; p < Q * P; p++) PQ_ans[p] = 0.0;

    yy = y;
    xx = x;
    ⟨ init subset loop 86a ⟩
    ⟨ start subset loop 86b ⟩
    {
        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]]++;
        }
        ⟨ continue subset loop 86c ⟩
    }
    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 [112d](#), [113abc](#).

Uses: HAS\_WEIGHTS [24f](#), [25a](#), P [23d](#), Q [24b](#), x [23cef](#), y [24acd](#).

### 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))

```

```

⟨ R_ThreeTableSums Prototype 114b ⟩ ≡
    SEXP R_ThreeTableSums
    (
        ⟨ R x Input 23c ⟩
        ⟨ R y Input 24a ⟩
        ⟨ R block Input 26b ⟩,
        ⟨ R weights Input 24e ⟩,
        ⟨ R subset Input 25c ⟩
    )
◇

```

Fragment referenced in [22a](#), [115a](#).

Uses: R\_ThreeTableSums [115a](#).

```

⟨ R_ThreeTableSums 115a ⟩ ≡
  ⟨ R_ThreeTableSums Prototype 114b ⟩
  {
    SEXP ans, dim;
    ⟨ C integer N Input 23b ⟩;
    ⟨ C integer Nsubset Input 25d ⟩;
    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 106b.

Defines: R\_ThreeTableSums 15b, 114b, 149, 150.

Uses: B 26c, block 26bd, mPQB 128a, N 23ab, NLEVELS 127a, Nsubset 25d, Offset0 21a, P 23d, Q 24b, RC\_ThreeTableSums 116a, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

```

⟨ RC_ThreeTableSums Prototype 115b ⟩ ≡
  void RC_ThreeTableSums
  (
    ⟨ C ThreeTableSums Input 116b ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ C subset range Input 25e ⟩,
    ⟨ C ThreeTableSums Answer 116c ⟩
  )
  ◇

```

Fragment referenced in 116a.

Uses: RC\_ThreeTableSums 116a.

```

⟨ RC_ThreeTableSums 116a ⟩ ≡
  ⟨ RC_ThreeTableSums Prototype 115b ⟩
  {
    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 106b.

Defines: *RC\_ThreeTableSums* 40a, 115ab.

Uses: B 26c, block 26bd, *C\_ThreeTableSums\_dweights\_dsubset* 116d, *C\_ThreeTableSums\_dweights\_isubset* 117c, *C\_ThreeTableSums\_iweights\_dsubset* 117a, *C\_ThreeTableSums\_iweights\_isubset* 117b, N 23ab, Nsubset 25d, offset 25e, P 23d, Q 24b, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

```

⟨ C_ThreeTableSums Input 116b ⟩ ≡
  ⟨ C integer x Input 23f ⟩
  ⟨ C integer y Input 24d ⟩
  ⟨ C integer block Input 26d ⟩
  ◇

```

Fragment referenced in 115b, 116d, 117abc.

```

⟨ C_ThreeTableSums Answer 116c ⟩ ≡
  double *PQL_ans
  ◇

```

Fragment referenced in 115b, 116d, 117abc.

```

⟨ C_ThreeTableSums_dweights_dsubset 116d ⟩ ≡
  void C_ThreeTableSums_dweights_dsubset
  (
    ⟨ C_ThreeTableSums Input 116b ⟩
    ⟨ C real weights Input 25a ⟩
    ⟨ C real subset Input 26a ⟩,
    ⟨ C_ThreeTableSums Answer 116c ⟩
  ) {
    double *s, *w;
    ⟨ ThreeTableSums Body 118a ⟩
  }
  ◇

```

Fragment referenced in 106b.

Defines: *C\_ThreeTableSums\_dweights\_dsubset* 116a.

```

⟨ C_ThreeTableSums_iweights_dsubset 117a ⟩ ≡
void C_ThreeTableSums_iweights_dsubset
(
    ⟨ C ThreeTableSums Input 116b ⟩
    ⟨ C integer weights Input 24f ⟩
    ⟨ C real subset Input 26a ⟩,
    ⟨ C ThreeTableSums Answer 116c ⟩
) {
    double *s;
    int *w;
    ⟨ ThreeTableSums Body 118a ⟩
}
◇

```

Fragment referenced in 106b.

Defines: C\_ThreeTableSums\_iweights\_dsubset 116a.

```

⟨ C_ThreeTableSums_iweights_isubset 117b ⟩ ≡
void C_ThreeTableSums_iweights_isubset
(
    ⟨ C ThreeTableSums Input 116b ⟩
    ⟨ C integer weights Input 24f ⟩
    ⟨ C integer subset Input 25f ⟩,
    ⟨ C ThreeTableSums Answer 116c ⟩
) {
    int *s, *w;
    ⟨ ThreeTableSums Body 118a ⟩
}
◇

```

Fragment referenced in 106b.

Defines: C\_ThreeTableSums\_iweights\_isubset 116a.

```

⟨ C_ThreeTableSums_dweights_isubset 117c ⟩ ≡
void C_ThreeTableSums_dweights_isubset
(
    ⟨ C ThreeTableSums Input 116b ⟩
    ⟨ C real weights Input 25a ⟩
    ⟨ C integer subset Input 25f ⟩,
    ⟨ C ThreeTableSums Answer 116c ⟩
) {
    int *s;
    double *w;
    ⟨ ThreeTableSums Body 118a ⟩
}
◇

```

Fragment referenced in 106b.

Defines: C\_ThreeTableSums\_dweights\_isubset 116a.



```

⟨ ThreeTableSums Body 118a ⟩ ≡
    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;
    ⟨ init subset loop 86a ⟩
    ⟨ start subset loop 86b ⟩
    {
        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]]++;
        }
        ⟨ continue subset loop 86c ⟩
    }
    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 [116d](#), [117abc](#).

Uses: B [26c](#), block [26bd](#), HAS\_WEIGHTS [24f](#), [25a](#), mPQB [128a](#), P [23d](#), Q [24b](#), x [23cef](#), y [24acd](#).

## 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))

```

```

⟨ Utils 118b ⟩ ≡
    ⟨ C_setup_subset 120b ⟩
    ⟨ C_setup_subset_block 121 ⟩
    ⟨ C_order_subset_wrt_block 122a ⟩
    ⟨ RC_order_subset_wrt_block 120a ⟩
    ⟨ R_order_subset_wrt_block 119b ⟩
    ◇

```

Fragment referenced in [22c](#).

```

⟨ R_order_subset_wrt_block Prototype 119a ⟩ ≡
  SEXP R_order_subset_wrt_block
  (
    ⟨ R y Input 24a ⟩
    ⟨ R weights Input 24e ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ R block Input 26b ⟩
  )
  ◇

```

Fragment referenced in [22a](#), [119b](#).  
 Uses: [R\\_order\\_subset\\_wrt\\_block 119b](#).

```

⟨ R_order_subset_wrt_block 119b ⟩ ≡
  ⟨ R_order_subset_wrt_block Prototype 119a ⟩
  {
    ⟨ C integer N Input 23b ⟩;
    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 [118b](#).  
 Defines: [R\\_order\\_subset\\_wrt\\_block 119a](#), [149](#), [150](#).  
 Uses: [block 26bd](#), [blockTable 26e](#), [N 23ab](#), [NCOL 126b](#), [NLEVELS 127a](#), [Offset0 21a](#), [RC\\_order\\_subset\\_wrt\\_block 120a](#),  
[RC\\_Sums 88a](#), [R\\_OneTableSums 107b](#), [subset 25cf](#), [26a](#), [weights 24ef](#), [25a](#), [y 24acd](#).

```

⟨ RC_order_subset_wrt_block Prototype 119c ⟩ ≡
  SEXP RC_order_subset_wrt_block
  (
    ⟨ C integer N Input 23b ⟩,
    ⟨ R subset Input 25c ⟩,
    ⟨ R block Input 26b ⟩,
    ⟨ R blockTable Input 26e ⟩
  )
  ◇

```

Fragment referenced in [120a](#).  
 Uses: [RC\\_order\\_subset\\_wrt\\_block 120a](#).

```

⟨ RC_order_subset_wrt_block 120a ⟩ ≡
  ⟨ RC_order_subset_wrt_block Prototype 119c ⟩
  {
    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 118b.

Defines: RC\_order\_subset\_wrt\_block 33a, 36, 119bc.

Uses: block 26bd, blockTable 26e, C\_order\_subset\_wrt\_block 122a, C\_setup\_subset 120b, C\_setup\_subset\_block 121, N 23ab, subset 25cf, 26a.

```

⟨ C_setup_subset 120b ⟩ ≡
  void C_setup_subset
  (
    ⟨ C integer N Input 23b ⟩,
    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 118b.

Defines: C\_setup\_subset 120a, 123a.

Uses: N 23ab.

```

⟨ C_setup_subset_block 121 ⟩ ≡
void C_setup_subset_block
(
    ⟨ C integer N Input 23b ⟩,
    ⟨ R block Input 26b ⟩,
    ⟨ R blockTable Input 26e ⟩,
    SEXP ans
) {
    double *cumtable;
    int Nlevels = LENGTH(blockTable);

    cumtable = R_Calloc(Nlevels, double);
    for (int k = 0; k < Nlevels; k++) cumtable[k] = 0.0;

    /* table[0] are missings, i.e. 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;
        }
    }

    R_Free(cumtable);
}
◇

```

Fragment referenced in [118b](#).

Defines: `C_setup_subset_block` [120a](#).

Uses: block [26bd](#), blockTable [26e](#), N [23ab](#).

```

⟨ C_order_subset_wrt_block 122a ⟩ ≡
void C_order_subset_wrt_block
(
    ⟨ R_subset Input 25c ⟩,
    ⟨ R_block Input 26b ⟩,
    ⟨ R_blockTable Input 26e ⟩,
    SEXP ans
) {
    double *cumtable;
    int Nlevels = LENGTH(blockTable);

    cumtable = R_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];
    }

    R_Free(cumtable);
}
◇

```

Fragment referenced in 118b.

Defines: C\_order\_subset\_wrt\_block 120a.

Uses: block 26bd, blockTable 26e, N 23ab, subset 25cf, 26a.

```

⟨ RC_setup_subset Prototype 122b ⟩ ≡
SEXP RC_setup_subset
(
    ⟨ C_integer N Input 23b ⟩,
    ⟨ R_weights Input 24e ⟩,
    ⟨ R_subset Input 25c ⟩
)
◇

```

Fragment referenced in 123a.

Uses: RC\_setup\_subset 123a.

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.

```

⟨ RC_setup_subset 123a ⟩ ≡
  ⟨ RC_setup_subset Prototype 122b ⟩
  {
    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 123b.

Defines: RC\_setup\_subset 36, 122b.

Uses: C\_setup\_subset 120b, N 23ab, Offset0 21a, RC\_Sums 88a, subset 25cf, 26a, sumweights 25b, weights 24ef, 25a.

### 3.10.2 Permutation Helpers

```

⟨ Permutations 123b ⟩ ≡
  ⟨ RC_setup_subset 123a ⟩
  ⟨ C_Permute 124a ⟩
  ⟨ C_doPermute 124b ⟩
  ⟨ C_PermuteBlock 124c ⟩
  ⟨ C_doPermuteBlock 125a ⟩
  ◇

```

Fragment referenced in 22c.

```

⟨ C_Permute 124a ⟩ ≡
void C_Permute
(
    double *subset,
    ⟨ C integer Nsubset Input 25d ⟩,
    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 [123b](#).

Defines: C\_Permute 124bc.

Uses: Nsubset 25d, subset 25cf, 26a.

```

⟨ C_doPermute 124b ⟩ ≡
void C_doPermute
(
    double *subset,
    ⟨ C integer Nsubset Input 25d ⟩,
    double *Nsubset_tmp,
    double *perm
) {
    Malloc(Nsubset_tmp, subset, Nsubset);
    C_Permute(Nsubset_tmp, Nsubset, perm);
}
◇

```

Fragment referenced in [123b](#).

Defines: C\_doPermute 36.

Uses: C\_Permute 124a, Nsubset 25d, subset 25cf, 26a.

```

⟨ C_PermuteBlock 124c ⟩ ≡
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 [123b](#).

Defines: C\_PermuteBlock 125a.

Uses: C\_Permute 124a, subset 25cf, 26a.

```

⟨ C_doPermuteBlock 125a ⟩ ≡
    void C_doPermuteBlock
    (
        double *subset,
        ⟨ C_integer Nsubset Input 25d ⟩,
        double *table,
        int Nlevels,
        double *Nsubset_tmp,
        double *perm
    ) {
        Malloc(Nsubset_tmp, subset, Nsubset);
        C_PermuteBlock(Nsubset_tmp, table, Nlevels, perm);
    }
    ◇

```

Fragment referenced in [123b](#).

Defines: `C_doPermuteBlock` [36](#).

Uses: `C_PermuteBlock` [124c](#), `Nsubset` [25d](#), `subset` [25cf](#), [26a](#).

### 3.10.3 Other Utils

```

⟨ MoreUtils 125b ⟩ ≡
    ⟨ NROW 126a ⟩
    ⟨ NCOL 126b ⟩
    ⟨ NLEVELS 127a ⟩
    ⟨ C_kronecker 129b ⟩
    ⟨ R_kronecker 129a ⟩
    ⟨ C_kronecker_sym 130a ⟩
    ⟨ C_KronSums_sym 130b ⟩
    ⟨ C_MPinv_sym 133 ⟩
    ⟨ R_MPinv_sym 132 ⟩
    ⟨ R_unpack_sym 135 ⟩
    ⟨ R_pack_sym 136c ⟩
    ◇

```

Fragment referenced in [22c](#).



```

⟨ NROW 126a ⟩ ≡
  R_xlen_t NROW
  (
    SEXP x
  ) {
    SEXP a;
    R_xlen_t ret;

    PROTECT(a = getAttrib(x, R_DimSymbol)); // rchk warning
    if (a == R_NilValue) {
      UNPROTECT(1);
      return(XLENGTH(x));
    }
    if (TYPEOF(a) == REALSXP) {
      ret = (R_xlen_t) REAL(a)[0];
      UNPROTECT(1);
      return(ret);
    }
    ret = (R_xlen_t) INTEGER(a)[0];
    UNPROTECT(1);
    return(ret);
  }
  ◇

```

Fragment referenced in [125b](#).  
 Defines: *NROW* [6](#), [8](#), [9ab](#), [14](#), [32a](#), [36](#), [42b](#), [43](#), [60b](#), [127a](#), [129a](#), [136c](#).  
 Uses: *x* [23cef](#).

```

⟨ NCOL 126b ⟩ ≡
  int NCOL
  (
    SEXP x
  ) {
    SEXP a;
    int ret;

    PROTECT(a = getAttrib(x, R_DimSymbol)); // rchk warning
    if (a == R_NilValue) {
      UNPROTECT(1);
      return(1);
    }
    if (TYPEOF(a) == REALSXP) {
      ret = (int) REAL(a)[1];
      UNPROTECT(1);
      return(ret);
    }
    ret = INTEGER(a)[1];
    UNPROTECT(1);
    return(ret);
  }
  ◇

```

Fragment referenced in [125b](#).  
 Defines: *NCOL* [12](#), [30c](#), [40b](#), [60b](#), [79a](#), [81a](#), [91b](#), [99b](#), [103b](#), [119b](#), [129a](#).  
 Uses: *x* [23cef](#).

```

⟨ NLEVELS 127a ⟩ ≡
    int NLEVELS
    (
        SEXP x
    ) {
        SEXP a;
        int maxlev = 0;

        PROTECT(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];
            }
            UNPROTECT(1);
            return(maxlev);
        }
        maxlev = NROW(a);
        UNPROTECT(1);
        return(maxlev);
    }
    ◇

```

Fragment referenced in [125b](#).

Defines: NLEVELS [30c](#), [40b](#), [107b](#), [111a](#), [115a](#), [119b](#).

Uses: NROW [126a](#), x [23cef](#).

Check for integer overflow when computing  $P(P+1)/2$  and  $PQ$ .

```

⟨ PP12 127b ⟩ ≡
    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 [137a](#).

Defines: PP12 [33a](#), [43](#), [45](#), [50](#), [77](#), [85b](#), [144](#), [145a](#).

Uses: P [23d](#).

```

⟨ mPQB 128a ⟩ ≡
  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 [137a](#).

Defines: mPQB [35a](#), [36](#), [44](#), [47](#), [51a](#), [69](#), [70b](#), [74b](#), [76b](#), [77](#), [78a](#), [98b](#), [102c](#), [111a](#), [115a](#), [118a](#), [144](#).

Uses: B [26c](#), P [23d](#), Q [24b](#).

```

> 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))

```

```

⟨ R_kronecker Prototype 128b ⟩ ≡
  SEXP R_kronecker
  (
    SEXP A,
    SEXP B
  )
  ◇

```

Fragment referenced in [22a](#), [129a](#).

Uses: B [26c](#).

This function can be called from other packages.

```

"src/libcoinAPI.h" 128c≡
  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 [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).

Uses: B [26c](#).

```

⟨ R_kronecker 129a ⟩ ≡
  ⟨ R_kronecker Prototype 128b ⟩
  {
    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 [125b](#).  
 Uses: B [26c](#), C\_kronecker [129b](#), NCOL [126b](#), NROW [126a](#).

```

⟨ C_kronecker 129b ⟩ ≡
  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 [125b](#).  
 Defines: C\_kronecker [78a](#), [129a](#).  
 Uses: B [26c](#), y [24acd](#).

```

⟨ C_kronecker_sym 130a ⟩ ≡
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 [125b](#).  
 Defines: C\_kronecker\_sym [77](#).  
 Uses: B [26c](#), S [20c](#), y [24acd](#).

```

⟨ C_KronSums_sym 130b ⟩ ≡
/* sum_i (t(x[i,]) %*% x[i,]) */
void C_KronSums_sym_
(
    ⟨ C_real x Input 23e ⟩
    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 [125b](#).  
 Defines: C\_KronSums\_sym Never used.  
 Uses: N [23ab](#), P [23d](#), S [20c](#), x [23cef](#).

```

> 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))

```

```

⟨ R_MPinv_sym Prototype 131a ⟩ ≡
  SEXP R_MPinv_sym
  (
    SEXP x,
    SEXP n,
    SEXP tol
  )
  ◇

```

Fragment referenced in [22a](#), [132](#).  
 Uses: R\_MPinv\_sym [132](#), x [23cef](#).

This function can be called from other packages.

```

"src/libcoinAPI.h" 131b≡
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 [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).  
 Uses: R\_MPinv\_sym [132](#), x [23cef](#).

```

⟨ R_MPinv_sym 132 ⟩ ≡
  ⟨ R_MPinv_sym Prototype 131a ⟩
  {
    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 [125b](#).

Defines: `R_MPinv_sym` [131ab](#), [149](#), [150](#), [154a](#), [155a](#).

Uses: `x` [23cef](#).

```

< C_MPinv_sym 133 > ≡
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 = R_Calloc(n * (n + 1) / 2, double);
        Memcpy(rx, x, n * (n + 1) / 2);
        work = R_Calloc(3 * n, double);
        val = R_Calloc(n, double);
        vec = R_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];
                }
            }
        }
        R_Free(rx); R_Free(work); R_Free(val); R_Free(vec);
    }
}
◇

```

Fragment referenced in [125b](#).  
 Uses: [S 20c](#), [x 23cef](#).

```

> 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))
```

$\langle R\_unpack\_sym \text{ Prototype 134a} \rangle \equiv$

```
SEXP R_unpack_sym
(
    SEXP x,
    SEXP names,
    SEXP diagonly
)
◇
```

Fragment referenced in [22a](#), [135](#).

Uses: [R\\_unpack\\_sym 135](#), [x 23cef](#).

This function can be called from other packages.

"src/libcoinAPI.h" 134b $\equiv$

```
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 [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).

Uses: [R\\_unpack\\_sym 135](#), [x 23cef](#).

```

⟨ R_unpack_sym 135 ⟩ ≡
  ⟨ R_unpack_sym Prototype 134a ⟩
  {
    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(diagonally)[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 [125b](#).  
 Defines: `R_unpack_sym` [10](#), [134ab](#), [149](#), [150](#), [155a](#).  
 Uses: `x` [23cef](#).

```

> 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))

```

```

⟨ R_pack_sym Prototype 136a ⟩ ≡
    SEXP R_pack_sym
    (
        SEXP x
    )
    ◇

```

Fragment referenced in [22a](#), [136c](#).

Uses: `R_pack_sym` [136c](#), `x` [23cef](#).

This function can be called from other packages.

```

"src/libcoinAPI.h" 136b≡
    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 [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).

Uses: `R_pack_sym` [136c](#), `x` [23cef](#).

```

⟨ R_pack_sym 136c ⟩ ≡
    ⟨ R_pack_sym Prototype 136a ⟩
    {
        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 [125b](#).

Defines: `R_pack_sym` [136ab](#), [149](#), [150](#), [154d](#).

Uses: `NROW` [126a](#), `x` [23cef](#).

### 3.11 Memory

```

⟨ Memory 137a ⟩ ≡
  ⟨ C_get_P 137c ⟩
  ⟨ C_get_Q 137d ⟩
  ⟨ PP12 127b ⟩
  ⟨ mPQB 128a ⟩
  ⟨ C_get_varonly 138a ⟩
  ⟨ C_get_Xfactor 138b ⟩
  ⟨ C_get_LinearStatistic 138c ⟩
  ⟨ C_get_Expectation 138d ⟩
  ⟨ C_get_Variance 139a ⟩
  ⟨ C_get_Covariance 139b ⟩
  ⟨ C_get_ExpectationX 139c ⟩
  ⟨ C_get_ExpectationInfluence 140a ⟩
  ⟨ C_get_CovarianceInfluence 140b ⟩
  ⟨ C_get_VarianceInfluence 140c ⟩
  ⟨ C_get_TableBlock 140d ⟩
  ⟨ C_get_Sumweights 141a ⟩
  ⟨ C_get_Table 141b ⟩
  ⟨ C_get_dimTable 141c ⟩
  ⟨ C_get_B 141d ⟩
  ⟨ C_get_nresample 142a ⟩
  ⟨ C_get_PermutedLinearStatistic 142b ⟩
  ⟨ C_get_tol 142c ⟩
  ⟨ RC_init_LECV_1d 145b ⟩
  ⟨ RC_init_LECV_2d 146 ⟩
  ◇

```

Fragment referenced in [22c](#).

```

⟨ R LECV Input 137b ⟩ ≡
  SEXP LECV
  ◇

```

Fragment referenced in [49b](#), [51b](#), [137cd](#), [138abcd](#), [139abc](#), [140abcd](#), [141abcd](#), [142abc](#).

Defines: [LECV 37bc](#), [38a](#), [49c](#), [50](#), [51a](#), [52](#), [53](#), [54ab](#), [55](#), [67b](#), [69](#), [137cd](#), [138abcd](#), [139abc](#), [140abcd](#), [141abcd](#), [142abc](#).

```

⟨ C_get_P 137c ⟩ ≡
  int C_get_P
  (
    ⟨ R LECV Input 137b ⟩
  ) {
    return(INTEGER(VECTOR_ELT(LECV, dim_SLOT))[0]);
  }
  ◇

```

Fragment referenced in [137a](#).

Defines: [C\\_get\\_P 32a](#), [38a](#), [45](#), [51a](#), [55](#), [69](#), [139ab](#), [142a](#).

Uses: [dim\\_SLOT 21a](#), [LECV 137b](#).

```

⟨ C_get_Q 137d ⟩ ≡
  int C_get_Q
  (
    ⟨ R LECV Input 137b ⟩
  ) {
    return(INTEGER(VECTOR_ELT(LECV, dim_SLOT))[1]);
  }
  ◇

```

Fragment referenced in [137a](#).

Defines: [C\\_get\\_Q 32a](#), [38a](#), [45](#), [51a](#), [69](#), [139ab](#), [142a](#).

Uses: [dim\\_SLOT 21a](#), [LECV 137b](#).

```

⟨ C_get_varonly 138a ⟩ ≡
    int C_get_varonly
    (
        ⟨ R LECV Input 137b ⟩
    ) {
        return(VECTOR_ELT(LECV, varonly_SLOT)[0]);
    }
    ◇

```

Fragment referenced in 137a.  
 Defines: C\_get\_varonly 31, 33a, 35a, 38a, 43, 44, 45, 51a, 53, 69, 139b.  
 Uses: LECV 137b, varonly\_SLOT 21a.

```

⟨ C_get_Xfactor 138b ⟩ ≡
    int C_get_Xfactor
    (
        ⟨ R LECV Input 137b ⟩
    ) {
        return(VECTOR_ELT(LECV, Xfactor_SLOT)[0]);
    }
    ◇

```

Fragment referenced in 137a.  
 Defines: C\_get\_Xfactor 45.  
 Uses: LECV 137b, Xfactor\_SLOT 21a.

```

⟨ C_get_LinearStatistic 138c ⟩ ≡
    double* C_get_LinearStatistic
    (
        ⟨ R LECV Input 137b ⟩
    ) {
        return(REAL(VECTOR_ELT(LECV, LinearStatistic_SLOT)));
    }
    ◇

```

Fragment referenced in 137a.  
 Defines: C\_get\_LinearStatistic 32b, 44, 50, 53, 69, 145a.  
 Uses: LECV 137b, LinearStatistic\_SLOT 21a.

```

⟨ C_get_Expectation 138d ⟩ ≡
    double* C_get_Expectation
    (
        ⟨ R LECV Input 137b ⟩
    ) {
        return(REAL(VECTOR_ELT(LECV, Expectation_SLOT)));
    }
    ◇

```

Fragment referenced in 137a.  
 Defines: C\_get\_Expectation 34a, 38a, 42b, 50, 53, 69, 145a.  
 Uses: Expectation\_SLOT 21a, LECV 137b.

```

< C_get_Variance 139a > ≡
double* C_get_Variance
(
    < R LECV Input 137b >
) {
    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 [137a](#).

Defines: `C_get_Variance` [34c](#), [35a](#), [38a](#), [43](#), [44](#), [53](#), [69](#), [139b](#), [145a](#).

Uses: `Covariance_SLOT` [21a](#), `C_get_P` [137c](#), `C_get_Q` [137d](#), `LECV` [137b](#), `S` [20c](#), `Variance_SLOT` [21a](#).

```

< C_get_Covariance 139b > ≡
double* C_get_Covariance
(
    < R LECV Input 137b >
) {
    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 [137a](#).

Defines: `C_get_Covariance` [34d](#), [35a](#), [38a](#), [43](#), [44](#), [50](#), [53](#), [69](#), [145a](#).

Uses: `Covariance_SLOT` [21a](#), `C_get_P` [137c](#), `C_get_Q` [137d](#), `C_get_Variance` [139a](#), `C_get_varonly` [138a](#), `LECV` [137b](#).

```

< C_get_ExpectationX 139c > ≡
double* C_get_ExpectationX
(
    < R LECV Input 137b >
) {
    return(REAL(VECTOR_ELT(LECV, ExpectationX_SLOT)));
}
◇

```

Fragment referenced in [137a](#).

Defines: `C_get_ExpectationX` [33a](#), [45](#), [69](#).

Uses: `ExpectationX_SLOT` [21a](#), `LECV` [137b](#).

```

⟨ C_get_ExpectationInfluence 140a ⟩ ≡
double* C_get_ExpectationInfluence
(
    ⟨ R LECV Input 137b ⟩
) {
    return(REAL(VECTOR_ELT(LECV, ExpectationInfluence_SLOT)));
}
◇

```

Fragment referenced in [137a](#).

Defines: `C_get_ExpectationInfluence` [33a](#), [45](#), [145a](#).

Uses: `ExpectationInfluence_SLOT` [21a](#), `LECV` [137b](#).

```

⟨ C_get_CovarianceInfluence 140b ⟩ ≡
double* C_get_CovarianceInfluence
(
    ⟨ R LECV Input 137b ⟩
) {
    return(REAL(VECTOR_ELT(LECV, CovarianceInfluence_SLOT)));
}
◇

```

Fragment referenced in [137a](#).

Defines: `C_get_CovarianceInfluence` [33a](#), [43](#), [69](#), [145a](#).

Uses: `CovarianceInfluence_SLOT` [21a](#), `LECV` [137b](#).

```

⟨ C_get_VarianceInfluence 140c ⟩ ≡
double* C_get_VarianceInfluence
(
    ⟨ R LECV Input 137b ⟩
) {
    return(REAL(VECTOR_ELT(LECV, VarianceInfluence_SLOT)));
}
◇

```

Fragment referenced in [137a](#).

Defines: `C_get_VarianceInfluence` [33a](#), [43](#), [69](#), [145a](#).

Uses: `LECV` [137b](#), `VarianceInfluence_SLOT` [21a](#).

```

⟨ C_get_TableBlock 140d ⟩ ≡
double* C_get_TableBlock
(
    ⟨ R LECV Input 137b ⟩
) {
    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 [137a](#).

Defines: `C_get_TableBlock` [33a](#).

Uses: `block` [26bd](#), `LECV` [137b](#), `TableBlock_SLOT` [21a](#).

```

⟨ C_get_Sumweights 141a ⟩ ≡
double* C_get_Sumweights
(
  ⟨ R LECV Input 137b ⟩
) {
  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 [137a](#).  
 Defines: `C_get_Sumweights` [33a](#), [45](#).  
 Uses: `LECV` [137b](#), `sumweights` [25b](#), `Sumweights_SLOT` [21a](#).

```

⟨ C_get_Table 141b ⟩ ≡
double* C_get_Table
(
  ⟨ R LECV Input 137b ⟩
) {
  if (LENGTH(LECV) <= Table_SLOT)
    error("Cannot extract table from object");
  return(REAL(VECTOR_ELT(LECV, Table_SLOT)));
}
◇

```

Fragment referenced in [137a](#).  
 Defines: `C_get_Table` [40a](#), [45](#).  
 Uses: `LECV` [137b](#), `Table_SLOT` [21a](#).

```

⟨ C_get_dimTable 141c ⟩ ≡
int* C_get_dimTable
(
  ⟨ R LECV Input 137b ⟩
) {
  if (LENGTH(LECV) <= Table_SLOT)
    error("Cannot extract table from object");
  return(INTEGER(getAttrib(VECTOR_ELT(LECV, Table_SLOT),
    R_DimSymbol)));
}
◇

```

Fragment referenced in [137a](#).  
 Defines: `C_get_dimTable` [45](#), [141d](#).  
 Uses: `LECV` [137b](#), `Table_SLOT` [21a](#).

```

⟨ C_get_B 141d ⟩ ≡
int C_get_B
(
  ⟨ R LECV Input 137b ⟩
) {
  if (VECTOR_ELT(LECV, TableBlock_SLOT) != R_NilValue)
    return(LENGTH(VECTOR_ELT(LECV, Sumweights_SLOT)));
  return(C_get_dimTable(LECV)[2]);
}
◇

```

Fragment referenced in [137a](#).  
 Defines: `C_get_B` [32a](#), [45](#), [69](#).  
 Uses: `C_get_dimTable` [141c](#), `LECV` [137b](#), `Sumweights_SLOT` [21a](#), `TableBlock_SLOT` [21a](#).



```

⟨ C_get_nresample 142a ⟩ ≡
    R_xlen_t C_get_nresample
    (
        ⟨ R LECV Input 137b ⟩
    ) {
        int PQ = C_get_P(LECV) * C_get_Q(LECV);
        return(XLENGTH(VECTOR_ELT(LECV, PermutedLinearStatistic_SLOT)) / PQ);
    }
    ◇

```

Fragment referenced in [137a](#).

Defines: `C_get_nresample` [38a](#), [50](#), [51a](#), [53](#), [55](#), [69](#).

Uses: `C_get_P` [137c](#), `C_get_Q` [137d](#), `LECV` [137b](#), `PermutedLinearStatistic_SLOT` [21a](#).

```

⟨ C_get_PermutedLinearStatistic 142b ⟩ ≡
    double* C_get_PermutedLinearStatistic
    (
        ⟨ R LECV Input 137b ⟩
    ) {
        return(REAL(VECTOR_ELT(LECV, PermutedLinearStatistic_SLOT)));
    }
    ◇

```

Fragment referenced in [137a](#).

Defines: `C_get_PermutedLinearStatistic` [38a](#), [50](#), [53](#), [69](#).

Uses: `LECV` [137b](#), `PermutedLinearStatistic_SLOT` [21a](#).

```

⟨ C_get_tol 142c ⟩ ≡
    double C_get_tol
    (
        ⟨ R LECV Input 137b ⟩
    ) {
        return(REAL(VECTOR_ELT(LECV, tol_SLOT))[0]);
    }
    ◇

```

Fragment referenced in [137a](#).

Defines: `C_get_tol` [38a](#), [50](#), [53](#), [69](#).

Uses: `LECV` [137b](#), `tol_SLOT` [21a](#).

```

⟨ Memory Input Checks 142d ⟩ ≡
    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 [144](#).

Uses: `B` [26c](#), `P` [23d](#), `Q` [24b](#).

⟨ *Memory Names 143* ⟩ ≡

```

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 144.

Uses: CovarianceInfluence\_SLOT 21a, Covariance\_SLOT 21a, dim\_SLOT 21a, ExpectationInfluence\_SLOT 21a, ExpectationX\_SLOT 21a, Expectation\_SLOT 21a, LinearStatistic\_SLOT 21a, PermutedLinearStatistic\_SLOT 21a, StandardisedPermutedLinearStatistic\_SLOT 21a, Sumweights\_SLOT 21a, TableBlock\_SLOT 21a, Table\_SLOT 21a, tol\_SLOT 21a, VarianceInfluence\_SLOT 21a, Variance\_SLOT 21a, varonly\_SLOT 21a, Xfactor\_SLOT 21a.

```

⟨ R_init_LECV 144 ⟩ ≡
    SEXP vo, d, names, tolerance, tmp;
    int PQ;

    ⟨ Memory Input Checks 142d ⟩
    PQ = mPQB(P, Q, 1);
    ⟨ Memory Names 143 ⟩

    /* 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);

    ⟨ Initialise Zero 145a ⟩
    ◇

```

Fragment referenced in [145b](#), [146](#).

Uses: [B 26c](#), [CovarianceInfluence\\_SLOT 21a](#), [Covariance\\_SLOT 21a](#), [dim\\_SLOT 21a](#), [ExpectationInfluence\\_SLOT 21a](#), [ExpectationX\\_SLOT 21a](#), [Expectation\\_SLOT 21a](#), [LinearStatistic\\_SLOT 21a](#), [mPQB 128a](#), [P 23d](#), [PermutedLinearStatistic\\_SLOT 21a](#), [PP12 127b](#), [Q 24b](#), [StandardisedPermutedLinearStatistic\\_SLOT 21a](#), [Sumweights\\_SLOT 21a](#), [TableBlock\\_SLOT 21a](#), [Table\\_SLOT 21a](#), [tol\\_SLOT 21a](#), [VarianceInfluence\\_SLOT 21a](#), [Variance\\_SLOT 21a](#), [varonly\\_SLOT 21a](#), [Xfactor\\_SLOT 21a](#).

```

⟨ Initialise Zero 145a ⟩ ≡
/* 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 [144](#).

Uses: [C\\_get\\_Covariance 139b](#), [C\\_get\\_CovarianceInfluence 140b](#), [C\\_get\\_Expectation 138d](#),  
[C\\_get\\_ExpectationInfluence 140a](#), [C\\_get\\_LinearStatistic 138c](#), [C\\_get\\_Variance 139a](#),  
[C\\_get\\_VarianceInfluence 140c](#), [PP12 127b](#), [Q 24b](#).

```

⟨ RC_init_LECV_1d 145b ⟩ ≡
SEXP RC_init_LECV_1d
(
    ⟨ C integer P Input 23d ⟩,
    ⟨ C integer Q Input 24b ⟩,
    int varonly,
    ⟨ C integer B Input 26c ⟩,
    int Xfactor,
    double tol
) {
    SEXP ans;

    ⟨ R_init_LECV 144 ⟩

    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 [137a](#).

Defines: [RC\\_init\\_LECV\\_1d 30b](#).

Uses: [B 26c](#), [Sumweights\\_SLOT 21a](#), [TableBlock\\_SLOT 21a](#).

```

⟨ RC_init_LECV_2d 146 ⟩ ≡
SEXP RC_init_LECV_2d
(
  ⟨ C integer P Input 23d ⟩,
  ⟨ C integer Q Input 24b ⟩,
  int varonly,
  int Lx,
  int Ly,
  ⟨ C integer B Input 26c ⟩,
  int Xfactor,
  double tol
) {
  SEXP ans, tabdim, tab;

  if (Lx <= 0)
    error("Lx is not positive");

  if (Ly <= 0)
    error("Ly is not positive");

  ⟨ R_init_LECV 144 ⟩

  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 [137a](#).  
 Defines: `RC_init_LECV_2d` [40a](#).  
 Uses: `B` [26c](#), `Table_SLOT` [21a](#).

## Chapter 4

# Package Infrastructure

```
"R/AAA.R" 147a≡
  < R Header 151a >
  .onUnload <-
  function(libpath)
    library.dynam.unload("libcoin", libpath)
  ◇

"DESCRIPTION" 147b≡
  < R Header 151a >
  Package: libcoin
  Title: Linear Test Statistics for Permutation Inference
  Date: 2026-06-03
  Version: 1.0-13
  Authors@R: c(person("Torsten", "Hothorn", role = c("aut", "cre"),
    email = "Torsten.Hothorn@R-project.org",
    comment = c(ORCID = "0000-0001-8301-0471")),
    person("Henric", "Winell", role = "aut",
    comment = c(ORCID = "0000-0001-7995-3047")))
  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, bibtex
  Imports: stats, mvtnorm
  LinkingTo: mvtnorm
  NeedsCompilation: yes
  License: GPL-2
  URL: https://codeberg.org/thothorn/coin/
  ◇

"NAMESPACE" 147c≡
  < R Header 151a >
  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).

```
"src/Makevars" 148a≡  
  < R Header 151a >  
  PKG_CFLAGS=$(C_VISIBILITY)  
  PKG_LIBS = $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS)  
  ◇
```

```
"src/libcoin-win.def" 148b≡  
  < def Header 152b >  
  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`.

```

"src/libcoin-init.c" 149≡
  < C Header 152a >
  #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 [149](#), [150](#).

Uses: [R\\_colSums 103b](#), [R\\_CovarianceInfluence 81a](#), [R\\_CovarianceX 84b](#), [R\\_ExpectationCovarianceStatistic 30b](#), [R\\_ExpectationCovarianceStatistic\\_2d 40a](#), [R\\_ExpectationInfluence 79a](#), [R\\_ExpectationX 82c](#), [R\\_KronSums 91b](#), [R\\_KronSums\\_Permutation 99b](#), [R\\_MPinv\\_sym 132](#), [R\\_OneTableSums 107b](#), [R\\_order\\_subset\\_wrt\\_block 119b](#), [R\\_pack\\_sym 136c](#), [R\\_PermutedLinearStatistic 36](#), [R\\_PermutedLinearStatistic\\_2d 47](#), [R\\_quadform 60b](#), [R\\_Sums 87b](#), [R\\_ThreeTableSums 115a](#), [R\\_TwoTableSums 111a](#), [R\\_unpack\\_sym 135](#).



```

"src/libcoin-init.c" 150≡
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 [149](#), [150](#).

Uses: [R\\_colSums 103b](#), [R\\_CovarianceInfluence 81a](#), [R\\_CovarianceX 84b](#), [R\\_ExpectationCovarianceStatistic 30b](#),  
[R\\_ExpectationCovarianceStatistic\\_2d 40a](#), [R\\_ExpectationInfluence 79a](#), [R\\_ExpectationX 82c](#), [R\\_KronSums 91b](#),  
[R\\_KronSums\\_Permutation 99b](#), [R\\_MPinv\\_sym 132](#), [R\\_OneTableSums 107b](#), [R\\_order\\_subset\\_wrt\\_block 119b](#),  
[R\\_pack\\_sym 136c](#), [R\\_PermutedLinearStatistic 36](#), [R\\_PermutedLinearStatistic\\_2d 47](#), [R\\_quadform 60b](#),  
[R\\_Sums 87b](#), [R\\_ThreeTableSums 115a](#), [R\\_TwoTableSums 111a](#), [R\\_unpack\\_sym 135](#).

$\langle R \text{ Header } 151a \rangle \equiv$

```
### Copyright (C) 2016-2026 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](#), [15b](#), [147abc](#), [148a](#), [158ab](#), [159bc](#).

$\langle Rd \text{ Header } 151b \rangle \equiv$

```
%%% Copyright (C) 2016-2026 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 [17](#), [18](#), [19](#), [152c](#), [153a](#), [159a](#).

$\langle C \text{ Header } 152a \rangle \equiv$

```
/*
    Copyright (C) 2016-2026 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 [20a](#), [21b](#), [22b](#), [30a](#), [149](#), [160bcd](#).

$\langle \text{def Header } 152b \rangle \equiv$

```
;;; Copyright (C) 2016-2026 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 [148b](#), [160a](#).

$\langle BibTeX \text{ Header } 152c \rangle \equiv$

$\langle Rd \text{ Header } 151b \rangle$

◇

Fragment referenced in [157a](#).

```
"inst/NEWS.Rd" 153a≡
  < Rd Header 151b>
  \newcommand{\C}{\ifelse{latex}{\out{\textsf{C}}}{C}}
  \newcommand{\nuweb}{\ifelse{latex}{\out{\textsf{nuweb}}}{nuweb}}

  \name{NEWS}
  \title{NEWS file for the \pkg{libcoin} package}
  ◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 153b≡
  \section{Changes in Version 1.0-13 (2026-06-03)}{
    \itemize{
      \item \file{NEWS.Rd} is now generated by \file{libcoin.w}.
      \item Remove the no longer needed \code{STRICT_R_HEADERS}.
      \item Documentation updates.
      \item Makefile improvements.
    }
  }
  ◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 153c≡
  \section{Changes in Version 1.0-12 (2026-03-16)}{
    \itemize{
      \item \code{PROTECT} calls to \code{getAttrib} as advised by \code{rchk}.
    }
  }
  ◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 153d≡
  \section{Changes in Version 1.0-11 (2026-03-06)}{
    \itemize{
      \item Use \file{REFERENCES.bib}.
    }
  }
  ◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 153e≡
  \section{Changes in Version 1.0-10 (2023-09-26)}{
    \itemize{
      \item Add Henric Winell as author.
      \item Enable \code{STRICT_R_HEADERS}.
      \item The \nuweb source code is now included (see \file{../inst/nuweb/}).
      \item Documentation updates.
    }
  }
  ◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 154a≡
\section{Changes in Version 1.0-9 (2021-09-27)}{
  \itemize{
    \item \code{R_MPinv_sym} can now determine the full dimension of the packed
      input \code{x} by setting \code{n} to \code{0L}.
    \item Add \code{#define USE_FC_LEN_T} to header as requested by CRAN
      2021-09-25.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).  
 Uses: `R_MPinv_sym` 132, x 23cef.

```
"inst/NEWS.Rd" 154b≡
\section{Changes in Version 1.0-8 (2021-02-08)}{
  \itemize{
    \item Fix \LaTeX problem.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 154c≡
\section{Changes in Version 1.0-7 (2021-01-15)}{
  \itemize{
    \item Correct spelling in error message.
    \item Regression tests updated for \pkg{coin} 1.4-0.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 154d≡
\section{Changes in Version 1.0-6 (2020-08-13)}{
  \itemize{
    \item Interface to \code{S_rcont2} changed in \pkg{stats}, thanks
      to Martin Maechler for a fix.
    \item New \C function \code{R_pack_sym} for transforming a symmetric matrix
      to lower-packed storage mode.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).  
 Uses: `R_pack_sym` 136c.

```
"inst/NEWS.Rd" 155a≡
\section{Changes in Version 1.0-5 (2019-08-22)}{
  \itemize{
    \item New \C function \code{R_unpack_sym} for unpacking a symmetric matrix
      stored in lower-packed storage mode.
    \item New \C functions \code{R_quadform} and \code{R_MPinv_sym} providing
      interfaces to the internal functions \code{C_quadform} and
      \code{C_MPinv_sym}.
    \item Function \code{libcoin_StandardisePermutedLinearStatistic} was renamed
      to \code{libcoin_R_StandardisePermutedLinearStatistic}.
    \item Documentation updates.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

Uses: `C_quadform` [61a](#), `R_MPinv_sym` [132](#), `R_quadform` [60b](#), `R_unpack_sym` [135](#).

```
"inst/NEWS.Rd" 155b≡
\section{Changes in Version 1.0-4 (2019-02-28)}{
  \itemize{
    \item Bugfix in regression tests.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 155c≡
\section{Changes in Version 1.0-3 (2019-02-18)}{
  \itemize{
    \item Parts of the covariance matrix were not set to zero initially.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 155d≡
\section{Changes in Version 1.0-2 (2018-12-13)}{
  \itemize{
    \item Maximally selected statistics failed for large sample sizes because of
      unnecessary memory allocation. Report and fix by Joanidis
      Kristoforos.
    \item Some additional checks for integer overflow.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 155e≡
\section{Changes in Version 1.0-1 (2017-12-13)}{
  \itemize{
    \item Make valgrind happy.
  }
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 156a≡
\section{Changes in Version 1.0-0 (2017-12-12)}{
\itemize{
\item Make the package truly literate; there is one single \nuweb file
called \file{libcoin.w} which generates the complete package.
\item The package can now deal with long vectors.
}
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 156b≡
\section{Changes in Version 0.9-3 (2017-07-02)}{
\itemize{
\item Remove copy of \code{rcont2} from the \pkg{stats} package as it is
exported now.
}
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 156c≡
\section{Changes in Version 0.9-2 (2017-04-04)}{
\itemize{
\item Fix protect problem in \code{R_MaximumTest} reported by Tomas
Kalibera.
}
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 156d≡
\section{Changes in Version 0.9-1 (2017-02-06)}{
\itemize{
\item Calling \pkg{libcoin}'s \C routines (using \code{.Call()}) now
requires the entry points to be specified as \R objects, i.e., the use
of character strings is no longer allowed.
\item Stop when all observations are missing.
\item 2d case with missings in at least one variable used incorrect sample
sizes and thus produced wrong results.
}
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

```
"inst/NEWS.Rd" 156e≡
\section{Changes in Version 0.9-0 (2016-12-09)}{
\itemize{
\item \pkg{libcoin} published on CRAN.
}
}
◇
```

File defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).

"inst/REFERENCES.bib" 157a≡  
 ⟨*BibTeX Header 152c*⟩  
 ◇

File defined by 157abcd.

"inst/REFERENCES.bib" 157b≡  
 @article{Hothorn+Hornik+vandeWiel+Zeileis:2006,  
   author  = {Torsten Hothorn and Kurt Hornik and Mark A van de Wiel and Achim  
             Zeileis},  
   title   = {A {L}ego System for Conditional Inference},  
   journal = {The American Statistician},  
   year    = {2006},  
   volume  = {60},  
   number  = {3},  
   pages   = {257--263},  
   doi     = {10.1198/000313006X118430},  
 }  
 ◇

File defined by 157abcd.

"inst/REFERENCES.bib" 157c≡  
 @article{Hothorn+Hornik+vandeWiel+Zeileis:2008,  
   author  = {Torsten Hothorn and Kurt Hornik and Mark A van de Wiel and Achim  
             Zeileis},  
   title   = {Implementing a Class of Permutation Tests: The coin Package},  
   journal = {Journal of Statistical Software},  
   year    = {2008},  
   volume  = {28},  
   number  = {8},  
   pages   = {1--23},  
   doi     = {10.18637/jss.v028.i08},  
 }  
 ◇

File defined by 157abcd.

"inst/REFERENCES.bib" 157d≡  
 @article{Strasser+Weber:1999,  
   author  = {Helmut Strasser and Christian Weber},  
   title   = {The Asymptotic Theory of Permutation Statistics},  
   journal = {Mathematical Methods of Statistics},  
   year    = {1999},  
   volume  = {8},  
   number  = {2},  
   pages   = {220--250},  
   note    = {Preprint available from \url{https://epub.wu.ac.at/102}}  
 }  
 ◇

File defined by 157abcd.



```

"inst/C_API_example/DESCRIPTION" 158a≡
< R Header 151a >
Package: libcoinEx
Title: A 'libcoin' C API Usage Example
Version: 1.0-0
Authors@R: c(person("Torsten", "Hothorn", role = c("aut", "cre"),
                  email = "Torsten.Hothorn@R-project.org",
                  comment = c(ORCID = "0000-0001-8301-0471")),
             person("Henric", "Winell", role = "aut",
                  comment = c(ORCID = "0000-0001-7995-3047")))
Description: Example use of the 'libcoin' C API.
Depends: R (>= 3.4.0)
Imports: libcoin
LinkingTo: libcoin
NeedsCompilation: yes
License: GPL-2
◇

```

```

"inst/C_API_example/NAMESPACE" 158b≡
< R Header 151a >
useDynLib(libcoinEx, .registration = TRUE)

import("libcoin")
export(ExpectationCovarianceStatistic)
◇

```

```
"inst/C_API_example/man/ExpectationCovarianceStatistic.Rd" 159a≡
< Rd Header 151b>
\name{ExpectationCovarianceStatistic}
\alias{ExpectationCovarianceStatistic}
\title{
  \pkg{libcoin} C API Example
}
\description{
  A working example of calling compiled code from the \pkg{libcoin} package.
}
\usage{
ExpectationCovarianceStatistic(x, y)
}
\arguments{
  \item{x}{A numeric matrix.}
  \item{y}{A numeric matrix.}
}
\details{
  This function calls the C function \code{R_ExpectationCovarianceStatistic}, a
  simple wrapper for the C function
  \code{libcoin_R_ExpectationCovarianceStatistic} defined in the \pkg{libcoin}
  package.

  For more information on this approach, consult section 5.4.2 of the Writing R
  Extensions manual.
}
\value{
  A list.
}
\examples{
n <- 100
p <- 4
q <- 2
X <- matrix(runif(p * n), nc = p)
Y <- matrix(runif(q * n), nc = q)
ExpectationCovarianceStatistic(X, Y)
}
◇
```

Uses: `R_ExpectationCovarianceStatistic` 30b, x 23cef, y 24acd.

```
"inst/C_API_example/R/ExpectationCovarianceStatistic.R" 159b≡
< R Header 151a>
ExpectationCovarianceStatistic <- function(x, y) {
  .Call(R_ExpectationCovarianceStatistic,
        x, y, integer(0), integer(0), integer(0), 0L,
        sqrt(.Machine$double.eps))
}
◇
```

Uses: `R_ExpectationCovarianceStatistic` 30b, x 23cef, y 24acd.

```
"inst/C_API_example/src/Makevars" 159c≡
< R Header 151a>
PKG_CFLAGS=$(C_VISIBILITY)
PKG_LIBS = $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS)
◇
```

```
"inst/C_API_example/src/libcoinEx-win.def" 160a≡
< def Header 152b >
LIBRARY libcoinEx.dll
EXPORTS
    R_init_libcoinEx
◇
```

```
"inst/C_API_example/src/libcoinEx.h" 160b≡
< C Header 152a >
#include <Rinternals.h>

extern SEXP R_ExpectationCovarianceStatistic
(
    const SEXP x,
    const SEXP y,
    const SEXP weights,
    const SEXP subset,
    const SEXP block,
    const SEXP varonly,
    const SEXP tol
);
◇
```

Uses: block 26bd, R\_ExpectationCovarianceStatistic 30b, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

```
"inst/C_API_example/src/libcoinEx-init.c" 160c≡
< C Header 152a >
#include "libcoinEx.h"
#include <R_ext/Rdynload.h>
#include <R_ext/Visibility.h>

#define CALLDEF(name, n) {#name, (DL_FUNC) &name, n}

static const R_CallMethodDef callMethods[] = {
    CALLDEF(R_ExpectationCovarianceStatistic, 7),
    {NULL, NULL, 0}
};

void attribute_visible R_init_libcoinEx(DllInfo *dll) {
    R_registerRoutines(dll, NULL, callMethods, NULL, NULL);
    R_useDynamicSymbols(dll, FALSE);
    R_forceSymbols(dll, TRUE);
}
◇
```

Uses: R\_ExpectationCovarianceStatistic 30b.

```
"inst/C_API_example/src/R_ExpectationCovarianceStatistic.c" 160d≡
< C Header 152a >
#include <R_ext/Rdynload.h> /* required by R */
#include <libcoinAPI.h>

SEXP R_ExpectationCovarianceStatistic(SEXP x, SEXP y, SEXP weights, SEXP subset,
                                       SEXP block, SEXP varonly, SEXP tol)
{
    return(libcoin_R_ExpectationCovarianceStatistic(x, y, weights, subset,
                                                    block, varonly, tol));
}
◇
```

Uses: block 26bd, R\_ExpectationCovarianceStatistic 30b, subset 25cf, 26a, weights 24ef, 25a, x 23cef, y 24acd.

# Index

## Files

"DESCRIPTION" Defined by [147b](#).  
"inst/C\_API\_example/DESCRIPTION" Defined by [158a](#).  
"inst/C\_API\_example/man/ExpectationCovarianceStatistic.Rd" Defined by [159a](#).  
"inst/C\_API\_example/NAMESPACE" Defined by [158b](#).  
"inst/C\_API\_example/R/ExpectationCovarianceStatistic.R" Defined by [159b](#).  
"inst/C\_API\_example/src/libcoinEx-init.c" Defined by [160c](#).  
"inst/C\_API\_example/src/libcoinEx-win.def" Defined by [160a](#).  
"inst/C\_API\_example/src/libcoinEx.h" Defined by [160b](#).  
"inst/C\_API\_example/src/Makevars" Defined by [159c](#).  
"inst/C\_API\_example/src/R\_ExpectationCovarianceStatistic.c" Defined by [160d](#).  
"inst/NEWS.Rd" Defined by [153abcde](#), [154abcd](#), [155abcde](#), [156abcde](#).  
"inst/REFERENCES.bib" Defined by [157abcd](#).  
"man/ctabs.Rd" Defined by [19](#).  
"man/doTest.Rd" Defined by [18](#).  
"man/LinStatExpCov.Rd" Defined by [17](#).  
"NAMESPACE" Defined by [147c](#).  
"R/AAA.R" Defined by [147a](#).  
"R/ctabs.R" Defined by [15b](#).  
"R/libcoin.R" Defined by [3a](#).  
"src/libcoin-init.c" Defined by [149](#), [150](#).  
"src/libcoin-win.def" Defined by [148b](#).  
"src/libcoin.c" Defined by [22b](#).  
"src/libcoin.h" Defined by [21b](#).  
"src/libcoinAPI.h" Defined by [30a](#), [35c](#), [37c](#), [39b](#), [46b](#), [49c](#), [52](#), [54b](#), [60a](#), [128c](#), [131b](#), [134b](#), [136b](#).  
"src/libcoin\_internal.h" Defined by [20a](#).  
"src/Makevars" Defined by [148a](#).

## Fragments

<2d Covariance [43](#)> Referenced in [44](#).  
<2d Expectation [42b](#)> Referenced in [44](#).  
<2d Memory [45](#)> Referenced in [44](#).  
<2d Total Table [41b](#)> Referenced in [44](#).  
<2d User Interface [38b](#)> Referenced in [22c](#).  
<2d User Interface Input [38c](#)> Referenced in [39a](#), [44](#).  
<BibTeX Header [152c](#)> Referenced in [157a](#).  
<C colSums Answer [104c](#)> Referenced in [79b](#), [103c](#), [104d](#), [105abc](#).  
<C colSums Input [104b](#)> Referenced in [103c](#), [104d](#), [105abc](#).  
<C Global Variables [21a](#)> Referenced in [20a](#).  
<C Header [152a](#)> Referenced in [20a](#), [21b](#), [22b](#), [30a](#), [149](#), [160bcd](#).  
<C integer B Input [26c](#)> Referenced in [26d](#), [31](#), [145b](#), [146](#).  
<C integer block Input [26d](#)> Referenced in [116b](#).  
<C integer N Input [23b](#)> Referenced in [23ef](#), [31](#), [36](#), [40a](#), [75c](#), [79ab](#), [81ab](#), [82c](#), [83a](#), [84b](#), [85a](#), [87c](#), [88b](#), [89abc](#), [91b](#), [92b](#), [99b](#), [100a](#), [103b](#), [107b](#), [111a](#), [115a](#), [119bc](#), [120b](#), [121](#), [122b](#).  
<C integer Nsubset Input [25d](#)> Referenced in [25e](#), [36](#), [40a](#), [79a](#), [81a](#), [82c](#), [84b](#), [87b](#), [91b](#), [99b](#), [103b](#), [107b](#), [111a](#), [115a](#), [124ab](#), [125a](#).  
<C integer P Input [23d](#)> Referenced in [23ef](#), [31](#), [75c](#), [76b](#), [77](#), [78a](#), [83a](#), [85a](#), [92b](#), [100a](#), [145b](#), [146](#).  
<C integer Q Input [24b](#)> Referenced in [24cd](#), [31](#), [76b](#), [77](#), [78a](#), [79ab](#), [81ab](#), [91b](#), [99b](#), [145b](#), [146](#).

< C integer subset Input 25f > Referenced in 89bc, 94c, 95, 97d, 98a, 101b, 102b, 105bc, 109bc, 113bc, 117bc.  
 < C integer weights Input 24f > Referenced in 89ab, 94bc, 97cd, 105ab, 109ab, 113ab, 117ab.  
 < C integer x Input 23f > Referenced in 97a, 102ab, 108b, 112b, 116b.  
 < C integer y Input 24d > Referenced in 112b, 116b.  
 < C KronSums Answer 92d > Referenced in 75c, 81b, 85a, 91c, 94abc, 95, 97bcd, 98a, 100a, 101ab, 102ab.  
 < C KronSums Input 92c > Referenced in 94abc, 95.  
 < C Macros 20c > Referenced in 20a.  
 < C OneTableSums Answer 108c > Referenced in 83a, 107c, 108d, 109abc.  
 < C OneTableSums Input 108b > Referenced in 107c, 108d, 109abc.  
 < C real subset Input 26a > Referenced in 88b, 89a, 94ab, 97bc, 101a, 102a, 104d, 105a, 108d, 109a, 112d, 113a, 116d, 117a.  
 < C real weights Input 25a > Referenced in 88b, 89c, 94a, 95, 97b, 98a, 104d, 105c, 108d, 109c, 112d, 113c, 116d, 117c.  
 < C real x Input 23e > Referenced in 92c, 101ab, 104b, 130b.  
 < C real y Input 24c > Referenced in 75c, 92bc, 97a, 100a, 101ab, 102ab.  
 < C subset range Input 25e > Referenced in 25f, 26a, 75c, 79b, 81b, 83a, 85a, 87c, 91c, 100a, 103c, 107c, 111b, 115b.  
 < C sumweights Input 25b > Referenced in 77, 78a, 79b, 81b.  
 < C ThreeTableSums Answer 116c > Referenced in 115b, 116d, 117abc.  
 < C ThreeTableSums Input 116b > Referenced in 115b, 116d, 117abc.  
 < C TwoTableSums Answer 112c > Referenced in 111b, 112d, 113abc.  
 < C TwoTableSums Input 112b > Referenced in 111b, 112d, 113abc.  
 < C XfactorKronSums Input 97a > Referenced in 97bcd, 98a.  
 < Check ix 9a > Referenced in 8, 15b.  
 < Check iy 9b > Referenced in 8, 15b.  
 < Check weights, subset, block 5a > Referenced in 6, 8, 15b.  
 < Col Row Total Sums 42a > Referenced in 44, 47.  
 < colSums 102d > Referenced in 22c.  
 < colSums Body 106a > Referenced in 104d, 105abc.  
 < Compute Covariance Influence 34b > Referenced in 31.  
 < Compute Covariance Linear Statistic 34d > Referenced in 31.  
 < Compute Expectation Linear Statistic 34a > Referenced in 31.  
 < Compute Linear Statistic 32b > Referenced in 31.  
 < Compute maxstat Permutation P-Value 71c > Referenced in 68, 72.  
 < Compute maxstat Test Statistic 71b > Referenced in 68, 72.  
 < Compute maxstat Variance / Covariance Directly 71a > Referenced in 68.  
 < Compute maxstat Variance / Covariance from Total Covariance 70b > Referenced in 68.  
 < Compute Permuted Linear Statistic 2d 48d > Referenced in 47.  
 < Compute Sum of Weights in Block 33b > Referenced in 31.  
 < Compute unordered maxstat Linear Statistic and Expectation 74a > Referenced in 72.  
 < Compute unordered maxstat Variance / Covariance Directly 75a > Referenced in 72.  
 < Compute unordered maxstat Variance / Covariance from Total Covariance 74b > Referenced in 72.  
 < Compute Variance from Covariance 35a > Referenced in 31.  
 < Compute Variance Linear Statistic 34c > Referenced in 31.  
 < continue subset loop 86c > Referenced in 90a, 96, 98b, 106a, 110a, 114a, 118a.  
 < Contrasts 14 > Referenced in 3a.  
 < Convert Table to Integer 48a > Referenced in 47.  
 < Count Levels 73a > Referenced in 72.  
 < ctab Prototype 15a > Referenced in 15b, 19.  
 < C\_chisq\_pvalue 62c > Referenced in 62b.  
 < C\_colSums\_dweights\_dsubset 104d > Referenced in 102d.  
 < C\_colSums\_dweights\_isubset 105c > Referenced in 102d.  
 < C\_colSums\_iweights\_dsubset 105a > Referenced in 102d.  
 < C\_colSums\_iweights\_isubset 105b > Referenced in 102d.  
 < C\_CovarianceLinearStatistic 77 > Referenced in 76a.  
 < C\_doPermute 124b > Referenced in 123b.  
 < C\_doPermuteBlock 125a > Referenced in 123b.  
 < C\_ExpectationLinearStatistic 76b > Referenced in 76a.  
 < C\_get\_B 141d > Referenced in 137a.  
 < C\_get\_Covariance 139b > Referenced in 137a.  
 < C\_get\_CovarianceInfluence 140b > Referenced in 137a.  
 < C\_get\_dimTable 141c > Referenced in 137a.  
 < C\_get\_Expectation 138d > Referenced in 137a.  
 < C\_get\_ExpectationInfluence 140a > Referenced in 137a.  
 < C\_get\_ExpectationX 139c > Referenced in 137a.  
 < C\_get\_LinearStatistic 138c > Referenced in 137a.

(C\_get\_nresample 142a) Referenced in 137a.  
 (C\_get\_P 137c) Referenced in 137a.  
 (C\_get\_PermutedLinearStatistic 142b) Referenced in 137a.  
 (C\_get\_Q 137d) Referenced in 137a.  
 (C\_get\_Sumweights 141a) Referenced in 137a.  
 (C\_get\_Table 141b) Referenced in 137a.  
 (C\_get\_TableBlock 140d) Referenced in 137a.  
 (C\_get\_tol 142c) Referenced in 137a.  
 (C\_get\_Variance 139a) Referenced in 137a.  
 (C\_get\_VarianceInfluence 140c) Referenced in 137a.  
 (C\_get\_varonly 138a) Referenced in 137a.  
 (C\_get\_Xfactor 138b) Referenced in 137a.  
 (C\_kronecker 129b) Referenced in 125b.  
 (C\_kronecker\_sym 130a) Referenced in 125b.  
 (C\_KronSums\_dweights\_dsubset 94a) Referenced in 90b.  
 (C\_KronSums\_dweights\_isubset 95) Referenced in 90b.  
 (C\_KronSums\_iweights\_dsubset 94b) Referenced in 90b.  
 (C\_KronSums\_iweights\_isubset 94c) Referenced in 90b.  
 (C\_KronSums\_Permutation\_dsubset 101a) Referenced in 90b.  
 (C\_KronSums\_Permutation\_isubset 101b) Referenced in 90b.  
 (C\_KronSums\_sym 130b) Referenced in 125b.  
 (C\_maxabsstand\_Covariance 58b) Referenced in 56a.  
 (C\_maxabsstand\_Variance 59a) Referenced in 56a.  
 (C\_maxstand\_Covariance 56b) Referenced in 56a.  
 (C\_maxstand\_Variance 57a) Referenced in 56a.  
 (C\_maxtype 61b) Referenced in 56a.  
 (C\_maxtype\_pvalue 65) Referenced in 62b.  
 (C\_minstand\_Covariance 57b) Referenced in 56a.  
 (C\_minstand\_Variance 58a) Referenced in 56a.  
 (C\_MPinv\_sym 133) Referenced in 125b.  
 (C\_norm\_pvalue 64) Referenced in 62b.  
 (C\_OneTableSums\_dweights\_dsubset 108d) Referenced in 106b.  
 (C\_OneTableSums\_dweights\_isubset 109c) Referenced in 106b.  
 (C\_OneTableSums\_iweights\_dsubset 109a) Referenced in 106b.  
 (C\_OneTableSums\_iweights\_isubset 109b) Referenced in 106b.  
 (C\_ordered\_Xfactor 68) Referenced in 56a.  
 (C\_order\_subset\_wrt\_block 122a) Referenced in 118b.  
 (C\_Permute 124a) Referenced in 123b.  
 (C\_PermuteBlock 124c) Referenced in 123b.  
 (C\_perm\_pvalue 63) Referenced in 62b.  
 (C\_quadform 61a) Referenced in 56a.  
 (C\_setup\_subset 120b) Referenced in 118b.  
 (C\_setup\_subset\_block 121) Referenced in 118b.  
 (C\_standardise 62a) Referenced in 56a.  
 (C\_Sums\_dweights\_dsubset 88b) Referenced in 86d.  
 (C\_Sums\_dweights\_isubset 89c) Referenced in 86d.  
 (C\_Sums\_iweights\_dsubset 89a) Referenced in 86d.  
 (C\_Sums\_iweights\_isubset 89b) Referenced in 86d.  
 (C\_ThreeTableSums\_dweights\_dsubset 116d) Referenced in 106b.  
 (C\_ThreeTableSums\_dweights\_isubset 117c) Referenced in 106b.  
 (C\_ThreeTableSums\_iweights\_dsubset 117a) Referenced in 106b.  
 (C\_ThreeTableSums\_iweights\_isubset 117b) Referenced in 106b.  
 (C\_TwoTableSums\_dweights\_dsubset 112d) Referenced in 106b.  
 (C\_TwoTableSums\_dweights\_isubset 113c) Referenced in 106b.  
 (C\_TwoTableSums\_iweights\_dsubset 113a) Referenced in 106b.  
 (C\_TwoTableSums\_iweights\_isubset 113b) Referenced in 106b.  
 (C\_unordered\_Xfactor 72) Referenced in 56a.  
 (C\_VarianceLinearStatistic 78a) Referenced in 76a.  
 (C\_XfactorKronSums\_dweights\_dsubset 97b) Referenced in 90b.  
 (C\_XfactorKronSums\_dweights\_isubset 98a) Referenced in 90b.  
 (C\_XfactorKronSums\_iweights\_dsubset 97c) Referenced in 90b.  
 (C\_XfactorKronSums\_iweights\_isubset 97d) Referenced in 90b.  
 (C\_XfactorKronSums\_Permutation\_dsubset 102a) Referenced in 90b.

<C\_XfactorKronSums\_Permutation\_isubset 102b> Referenced in 90b.  
 <def Header 152b> Referenced in 148b, 160a.  
 <doTest 12> Referenced in 3a.  
 <doTest Prototype 11> Referenced in 12, 18.  
 <ExpectationCovariances 76a> Referenced in 22c.  
 <Extract Dimensions 32a> Referenced in 31.  
 <Function Definitions 22c> Referenced in 22b.  
 <Function Prototypes 22a> Referenced in 21b.  
 <Handle Missing Values 5b> Referenced in 6.  
 <init subset loop 86a> Referenced in 90a, 96, 98b, 106a, 110a, 114a, 118a.  
 <Initialise Zero 145a> Referenced in 144.  
 <KronSums 90b> Referenced in 22c.  
 <KronSums Body 96> Referenced in 94abc, 95.  
 <KronSums Double x 93b> Referenced in 92a.  
 <KronSums Integer x 93a> Referenced in 92a.  
 <KronSums Permutation Body 101c> Referenced in 101ab.  
 <Linear Statistic 2d 41a> Referenced in 44, 48d.  
 <LinearStatistics 75b> Referenced in 22c.  
 <LinStatExpCov 4> Referenced in 3a.  
 <LinStatExpCov Prototype 3b> Referenced in 4, 17.  
 <LinStatExpCov1d 6> Referenced in 3a.  
 <LinStatExpCov2d 8> Referenced in 3a.  
 <maxstat Xfactor Variables 67b> Referenced in 68, 72.  
 <Memory 137a> Referenced in 22c.  
 <Memory Input Checks 142d> Referenced in 144.  
 <Memory Names 143> Referenced in 144.  
 <MoreUtils 125b> Referenced in 22c.  
 <mPQB 128a> Referenced in 137a.  
 <NCOL 126b> Referenced in 125b.  
 <NLEVELS 127a> Referenced in 125b.  
 <NROW 126a> Referenced in 125b.  
 <OneTableSums Body 110a> Referenced in 108d, 109abc.  
 <P-Values 62b> Referenced in 22c.  
 <Permutations 123b> Referenced in 22c.  
 <PP12 127b> Referenced in 137a.  
 <R block Input 26b> Referenced in 29b, 38c, 46a, 114b, 119ac, 121, 122a.  
 <R blockTable Input 26e> Referenced in 119c, 121, 122a.  
 <R Header 151a> Referenced in 3a, 15b, 147abc, 148a, 158ab, 159bc.  
 <R Includes 20b> Referenced in 20a.  
 <R LECV Input 137b> Referenced in 49b, 51b, 137cd, 138abcd, 139abc, 140abcd, 141abcd, 142abc.  
 <R N Input 23a> Referenced in 87a.  
 <R subset Input 25c> Referenced in 29b, 38c, 75c, 78b, 79b, 80, 81b, 82b, 83a, 84a, 85a, 87ac, 91ac, 99a, 100a, 103ac, 107ac, 110b, 111b, 114b, 115b, 119ac, 122ab.  
 <R weights Input 24e> Referenced in 29b, 38c, 75c, 78b, 79b, 80, 81b, 82b, 83a, 84a, 85a, 87ac, 91ac, 103ac, 107ac, 110b, 111b, 114b, 115b, 119a, 122b.  
 <R x Input 23c> Referenced in 29b, 38c, 46a, 75c, 82b, 83a, 84a, 85a, 91a, 92b, 99a, 100a, 103a, 107a, 110b, 114b.  
 <R y Input 24a> Referenced in 29b, 38c, 46a, 78b, 79b, 80, 81b, 91a, 99a, 110b, 114b, 119a.  
 <RC KronSums Input 92b> Referenced in 91c.  
 <RC\_colSums 104a> Referenced in 102d.  
 <RC\_colSums Prototype 103c> Referenced in 104a.  
 <RC\_CovarianceInfluence 82a> Referenced in 76a.  
 <RC\_CovarianceInfluence Prototype 81b> Referenced in 82a.  
 <RC\_CovarianceX 85b> Referenced in 76a.  
 <RC\_CovarianceX Prototype 85a> Referenced in 85b.  
 <RC\_ExpectationCovarianceStatistic 31> Referenced in 29a.  
 <RC\_ExpectationCovarianceStatistic\_2d 44> Referenced in 38b.  
 <RC\_ExpectationInfluence 79c> Referenced in 76a.  
 <RC\_ExpectationInfluence Prototype 79b> Referenced in 79c.  
 <RC\_ExpectationX 83b> Referenced in 76a.  
 <RC\_ExpectationX Prototype 83a> Referenced in 83b.  
 <RC\_init\_LECV\_1d 145b> Referenced in 137a.  
 <RC\_init\_LECV\_2d 146> Referenced in 137a.  
 <RC\_KronSums 92a> Referenced in 90b.

{RC\_KronSums Prototype 91c} Referenced in 92a.  
 {RC\_KronSums\_Permutation 100b} Referenced in 90b.  
 {RC\_KronSums\_Permutation Prototype 100a} Referenced in 100b.  
 {RC\_LinearStatistic 75d} Referenced in 75b.  
 {RC\_LinearStatistic Prototype 75c} Referenced in 75d.  
 {RC\_OneTableSums 108a} Referenced in 106b.  
 {RC\_OneTableSums Prototype 107c} Referenced in 108a.  
 {RC\_order\_subset\_wrt\_block 120a} Referenced in 118b.  
 {RC\_order\_subset\_wrt\_block Prototype 119c} Referenced in 120a.  
 {RC\_setup\_subset 123a} Referenced in 123b.  
 {RC\_setup\_subset Prototype 122b} Referenced in 123a.  
 {RC\_Sums 88a} Referenced in 86d.  
 {RC\_Sums Prototype 87c} Referenced in 88a.  
 {RC\_ThreeTableSums 116a} Referenced in 106b.  
 {RC\_ThreeTableSums Prototype 115b} Referenced in 116a.  
 {RC\_TwoTableSums 112a} Referenced in 106b.  
 {RC\_TwoTableSums Prototype 111b} Referenced in 112a.  
 {Rd Header 151b} Referenced in 17, 18, 19, 152c, 153a, 159a.  
 {R\_colSums 103b} Referenced in 102d.  
 {R\_colSums Prototype 103a} Referenced in 22a, 103b.  
 {R\_CovarianceInfluence 81a} Referenced in 76a.  
 {R\_CovarianceInfluence Prototype 80} Referenced in 22a, 81a.  
 {R\_CovarianceX 84b} Referenced in 76a.  
 {R\_CovarianceX Prototype 84a} Referenced in 22a, 84b.  
 {R\_ExpectationCovarianceStatistic 30b} Referenced in 29a.  
 {R\_ExpectationCovarianceStatistic Prototype 29c} Referenced in 22a, 30b.  
 {R\_ExpectationCovarianceStatistic\_2d 40a} Referenced in 38b.  
 {R\_ExpectationCovarianceStatistic\_2d Prototype 39a} Referenced in 22a, 40a.  
 {R\_ExpectationInfluence 79a} Referenced in 76a.  
 {R\_ExpectationInfluence Prototype 78b} Referenced in 22a, 79a.  
 {R\_ExpectationX 82c} Referenced in 76a.  
 {R\_ExpectationX Prototype 82b} Referenced in 22a, 82c.  
 {R\_init\_LECV 144} Referenced in 145b, 146.  
 {R\_kronecker 129a} Referenced in 125b.  
 {R\_kronecker Prototype 128b} Referenced in 22a, 129a.  
 {R\_KronSums 91b} Referenced in 90b.  
 {R\_KronSums Prototype 91a} Referenced in 22a, 91b.  
 {R\_KronSums\_Permutation 99b} Referenced in 90b.  
 {R\_KronSums\_Permutation Prototype 99a} Referenced in 22a, 99b.  
 {R\_MaximallySelectedTest 55} Referenced in 49a.  
 {R\_MaximallySelectedTest Prototype 54a} Referenced in 22a, 55.  
 {R\_MaximumTest 53} Referenced in 49a.  
 {R\_MaximumTest Prototype 51b} Referenced in 22a, 53.  
 {R\_MPinv\_sym 132} Referenced in 125b.  
 {R\_MPinv\_sym Prototype 131a} Referenced in 22a, 132.  
 {R\_OneTableSums 107b} Referenced in 106b.  
 {R\_OneTableSums Prototype 107a} Referenced in 22a, 107b.  
 {R\_order\_subset\_wrt\_block 119b} Referenced in 118b.  
 {R\_order\_subset\_wrt\_block Prototype 119a} Referenced in 22a, 119b.  
 {R\_pack\_sym 136c} Referenced in 125b.  
 {R\_pack\_sym Prototype 136a} Referenced in 22a, 136c.  
 {R\_PermutedLinearStatistic 36} Referenced in 29a.  
 {R\_PermutedLinearStatistic Prototype 35b} Referenced in 22a, 36.  
 {R\_PermutedLinearStatistic\_2d 47} Referenced in 38b.  
 {R\_PermutedLinearStatistic\_2d Prototype 46a} Referenced in 22a, 47.  
 {R\_quadform 60b} Referenced in 56a.  
 {R\_quadform Prototype 59b} Referenced in 22a, 60b.  
 {R\_QuadraticTest 50} Referenced in 49a.  
 {R\_QuadraticTest Prototype 49b} Referenced in 22a, 50.  
 {R\_StandardisePermutedLinearStatistic 38a} Referenced in 29a.  
 {R\_StandardisePermutedLinearStatistic Prototype 37b} Referenced in 22a, 38a.  
 {R\_Sums 87b} Referenced in 86d.  
 {R\_Sums Prototype 87a} Referenced in 22a, 87b.



{R\_ThreeTableSums 115a} Referenced in 106b.  
 {R\_ThreeTableSums Prototype 114b} Referenced in 22a, 115a.  
 {R\_TwoTableSums 111a} Referenced in 106b.  
 {R\_TwoTableSums Prototype 110b} Referenced in 22a, 111a.  
 {R\_unpack\_sym 135} Referenced in 125b.  
 {R\_unpack\_sym Prototype 134a} Referenced in 22a, 135.  
 {Setup Dimensions 30c} Referenced in 30b, 36.  
 {Setup Dimensions 2d 40b} Referenced in 40a, 47.  
 {Setup Linear Statistic 37a} Referenced in 36, 47.  
 {Setup Log-Factorials 48c} Referenced in 47.  
 {Setup maxstat Memory 70a} Referenced in 68, 72.  
 {Setup maxstat Variables 69} Referenced in 68, 72.  
 {Setup Memory and Subsets in Blocks 33a} Referenced in 31.  
 {Setup mvtnorm Correlation 67a} Referenced in 65.  
 {Setup mvtnorm Memory 66} Referenced in 65.  
 {Setup Test Memory 51a} Referenced in 50, 53.  
 {Setup unordered maxstat Contrasts 73b} Referenced in 72.  
 {Setup Working Memory 48b} Referenced in 47.  
 {SimpleSums 86d} Referenced in 22c.  
 {start subset loop 86b} Referenced in 90a, 96, 98b, 106a, 110a, 114a, 118a.  
 {Sums Body 90a} Referenced in 88b, 89abc.  
 {Tables 106b} Referenced in 22c.  
 {Test Statistics 56a} Referenced in 22c.  
 {Tests 49a} Referenced in 22c.  
 {ThreeTableSums Body 118a} Referenced in 116d, 117abc.  
 {TwoTableSums Body 114a} Referenced in 112d, 113abc.  
 {User Interface 29a} Referenced in 22c.  
 {User Interface Input 29b} Referenced in 29c, 31, 35b.  
 {Utils 118b} Referenced in 22c.  
 {vcov LinStatExpCov 10} Referenced in 3a.  
 {XfactorKronSums Body 98b} Referenced in 97bcd, 98a.  
 {XfactorKronSums Permutation Body 102c} Referenced in 102ab.

## Identifiers

ALTERNATIVE\_greater: 21a, 61b, 64, 67a.  
 ALTERNATIVE\_less: 21a, 53, 61b, 64, 67a.  
 ALTERNATIVE\_twosided: 21a, 61b, 64, 67a, 71b.  
 B: 26c, 30bc, 31, 32a, 33a, 36, 40ab, 41b, 44, 45, 47, 48b, 68, 69, 72, 115a, 116a, 118a, 128abc, 129ab, 130a, 142d, 144, 145b, 146.  
 block: 3b, 4, 5a, 6, 8, 15ab, 17, 19, 26b, 26d, 30abc, 33ab, 35c, 36, 39b, 40ab, 46b, 115a, 116a, 118a, 119b, 120a, 121, 122a, 140d, 160bd.  
 blockTable: 26e, 36, 119b, 120a, 121, 122a.  
 CovarianceInfluence\_SLOT: 21a, 140b, 143, 144.  
 Covariance\_SLOT: 21a, 139ab, 143, 144.  
 C\_chisq\_pvalue: 50, 62c.  
 C\_colSums\_dweights\_dsubset: 104a, 104d.  
 C\_colSums\_dweights\_isubset: 104a, 105c.  
 C\_colSums\_iweights\_dsubset: 104a, 105a.  
 C\_colSums\_iweights\_isubset: 104a, 105b.  
 C\_CovarianceLinearStatistic: 34d, 43, 71a, 75a, 77, 78a.  
 C\_doPermute: 36, 124b.  
 C\_doPermuteBlock: 36, 125a.  
 C\_ExpectationLinearStatistic: 34a, 42b, 76b.  
 C\_get\_B: 32a, 45, 69, 141d.  
 C\_get\_Covariance: 34d, 35a, 38a, 43, 44, 50, 53, 69, 139b, 145a.  
 C\_get\_CovarianceInfluence: 33a, 43, 69, 140b, 145a.  
 C\_get\_dimTable: 45, 141c, 141d.  
 C\_get\_Expectation: 34a, 38a, 42b, 50, 53, 69, 138d, 145a.  
 C\_get\_ExpectationInfluence: 33a, 45, 140a, 145a.  
 C\_get\_ExpectationX: 33a, 45, 69, 139c.  
 C\_get\_LinearStatistic: 32b, 44, 50, 53, 69, 138c, 145a.  
 C\_get\_nresample: 38a, 50, 51a, 53, 55, 69, 142a.

C\_get\_P: 32a, 38a, 45, 51a, 55, 69, [137c](#), 139ab, 142a.  
C\_get\_PermutedLinearStatistic: 38a, 50, 53, 69, [142b](#).  
C\_get\_Q: 32a, 38a, 45, 51a, 69, [137d](#), 139ab, 142a.  
C\_get\_Sumweights: 33a, 45, [141a](#).  
C\_get\_Table: 40a, 45, [141b](#).  
C\_get\_TableBlock: 33a, [140d](#).  
C\_get\_tol: 38a, 50, 53, 69, [142c](#).  
C\_get\_Variance: 34c, 35a, 38a, 43, 44, 53, 69, [139a](#), 139b, 145a.  
C\_get\_VarianceInfluence: 33a, 43, 69, [140c](#), 145a.  
C\_get\_varonly: 31, 33a, 35a, 38a, 43, 44, 45, 51a, 53, 69, [138a](#), 139b.  
C\_get\_Xfactor: 45, [138b](#).  
C\_kronecker: 78a, 129a, [129b](#).  
C\_kronecker\_sym: 77, [130a](#).  
C\_KronSums\_dweights\_dsubset: 93b, [94a](#).  
C\_KronSums\_dweights\_isubset: 93b, [95](#).  
C\_KronSums\_ieweights\_dsubset: 93b, [94b](#).  
C\_KronSums\_ieweights\_isubset: 93b, [94c](#).  
C\_KronSums\_Permutation\_dsubset: 100b, [101a](#).  
C\_KronSums\_Permutation\_isubset: 100b, [101b](#).  
C\_maxabsstand\_Covariance: [58b](#), 61b.  
C\_maxabsstand\_Variance: [59a](#), 61b.  
C\_maxstand\_Covariance: [56b](#), 61b.  
C\_maxstand\_Variance: [57a](#), 61b.  
C\_maxtype: 53, [61b](#), 71b.  
C\_maxtype\_pvalue: 53, [65](#).  
C\_minstand\_Covariance: [57b](#), 61b.  
C\_minstand\_Variance: [58a](#), 61b.  
C\_norm\_pvalue: [64](#), 65.  
C\_OneTableSums\_dweights\_dsubset: 108a, [108d](#).  
C\_OneTableSums\_dweights\_isubset: 108a, [109c](#).  
C\_OneTableSums\_ieweights\_dsubset: 108a, [109a](#).  
C\_OneTableSums\_ieweights\_isubset: 108a, [109b](#).  
C\_ordered\_Xfactor: 34b, 43, 55, [68](#).  
C\_order\_subset\_wrt\_block: 120a, [122a](#).  
C\_Permute: [124a](#), [124bc](#).  
C\_PermuteBlock: [124c](#), 125a.  
C\_perm\_pvalue: 50, 53, [63](#), 71c.  
C\_quadform: 50, 60b, [61a](#), 71b, 155a.  
C\_setup\_subset: 120a, [120b](#), 123a.  
C\_setup\_subset\_block: 120a, [121](#).  
C\_standardise: 38a, [62a](#).  
C\_Sums\_dweights\_dsubset: 88a, [88b](#).  
C\_Sums\_dweights\_isubset: 88a, [89c](#).  
C\_Sums\_ieweights\_dsubset: 88a, [89a](#).  
C\_Sums\_ieweights\_isubset: 88a, [89b](#).  
C\_ThreeTableSums\_dweights\_dsubset: 116a, [116d](#).  
C\_ThreeTableSums\_dweights\_isubset: 116a, [117c](#).  
C\_ThreeTableSums\_ieweights\_dsubset: 116a, [117a](#).  
C\_ThreeTableSums\_ieweights\_isubset: 116a, [117b](#).  
C\_TwoTableSums\_dweights\_dsubset: 112a, [112d](#).  
C\_TwoTableSums\_dweights\_isubset: 112a, [113c](#).  
C\_TwoTableSums\_ieweights\_dsubset: 112a, [113a](#).  
C\_TwoTableSums\_ieweights\_isubset: 112a, [113b](#).  
C\_unordered\_Xfactor: 34b, 55, [72](#).  
C\_VarianceLinearStatistic: 34c, 43, 71a, 75a, [78a](#).  
C\_XfactorKronSums\_dweights\_dsubset: 93a, [97b](#).  
C\_XfactorKronSums\_dweights\_isubset: 93a, [98a](#).  
C\_XfactorKronSums\_ieweights\_dsubset: 93a, [97c](#).  
C\_XfactorKronSums\_ieweights\_isubset: 93a, [97d](#).  
C\_XfactorKronSums\_Permutation\_dsubset: 100b, [102a](#).  
C\_XfactorKronSums\_Permutation\_isubset: 100b, [102b](#).  
dim\_SLOT: [21a](#), 137cd, 143, 144.  
DoCenter: [21a](#), 75d, 79c, 82a, 83b, 85b, 91b, 103b.

DoSymmetric: [21a](#), [75d](#), [82a](#), [85b](#).  
 DoVarOnly: [21a](#), [34bcd](#), [43](#).  
 ExpectationInfluence\_SLOT: [21a](#), [140a](#), [143](#), [144](#).  
 ExpectationX\_SLOT: [21a](#), [139c](#), [143](#), [144](#).  
 Expectation\_SLOT: [21a](#), [138d](#), [143](#), [144](#).  
 GE: [20c](#), [50](#), [53](#).  
 HAS\_WEIGHTS: [24f](#), [25a](#), [90a](#), [96](#), [98b](#), [106a](#), [110a](#), [114a](#), [118a](#).  
 LE: [20c](#), [53](#).  
 LECV: [37bc](#), [38a](#), [49c](#), [50](#), [51a](#), [52](#), [53](#), [54ab](#), [55](#), [67b](#), [69](#), [137b](#), [137cd](#), [138abcd](#), [139abc](#), [140abcd](#), [141abcd](#), [142abc](#).  
 LinearStatistic\_SLOT: [21a](#), [138c](#), [143](#), [144](#).  
 mPQB: [35a](#), [36](#), [44](#), [47](#), [51a](#), [69](#), [70b](#), [74b](#), [76b](#), [77](#), [78a](#), [98b](#), [102c](#), [111a](#), [115a](#), [118a](#), [128a](#), [144](#).  
 N: [5ab](#), [6](#), [8](#), [15b](#), [23a](#), [23b](#), [32ab](#), [33ab](#), [34abcd](#), [36](#), [40a](#), [65](#), [75d](#), [79ac](#), [81a](#), [82ac](#), [83b](#), [84b](#), [85b](#), [86ab](#), [87b](#), [88a](#), [90a](#), [91b](#), [93ab](#), [96](#), [98b](#), [99b](#), [100b](#), [101c](#), [102c](#), [103b](#), [104a](#), [106a](#), [107b](#), [108a](#), [111a](#), [112a](#), [115a](#), [116a](#), [119b](#), [120ab](#), [121](#), [122a](#), [123a](#), [130b](#).  
 NCOL: [12](#), [30c](#), [40b](#), [60b](#), [79a](#), [81a](#), [91b](#), [99b](#), [103b](#), [119b](#), [126b](#), [129a](#).  
 NLEVELS: [30c](#), [40b](#), [107b](#), [111a](#), [115a](#), [119b](#), [127a](#).  
 NROW: [6](#), [8](#), [9ab](#), [14](#), [32a](#), [36](#), [42b](#), [43](#), [60b](#), [126a](#), [127a](#), [129a](#), [136c](#).  
 Nsubset: [25d](#), [33b](#), [36](#), [40a](#), [75d](#), [79ac](#), [81a](#), [82ac](#), [83b](#), [84b](#), [85b](#), [86abc](#), [87b](#), [88a](#), [90a](#), [91b](#), [93ab](#), [99b](#), [100b](#), [101c](#), [102c](#), [103b](#), [104a](#), [107b](#), [108a](#), [111a](#), [112a](#), [115a](#), [116a](#), [124ab](#), [125a](#).  
 offset: [25e](#), [31](#), [33b](#), [34abcd](#), [75d](#), [79c](#), [82a](#), [83b](#), [85b](#), [86a](#), [88a](#), [93ab](#), [100b](#), [101c](#), [102c](#), [104a](#), [108a](#), [112a](#), [116a](#).  
 Offset: [21a](#), [32b](#), [33a](#), [36](#), [40a](#), [42b](#), [43](#), [79a](#), [81a](#), [82c](#), [84b](#), [87b](#), [91b](#), [99b](#), [103b](#), [107b](#), [111a](#), [115a](#), [119b](#), [123a](#).  
 P: [14](#), [23d](#), [30bc](#), [32ab](#), [33a](#), [34acd](#), [35a](#), [36](#), [40ab](#), [41a](#), [42b](#), [43](#), [44](#), [45](#), [47](#), [50](#), [51a](#), [53](#), [55](#), [68](#), [69](#), [70ab](#), [72](#), [73ab](#), [74ab](#), [75d](#), [76b](#), [77](#), [78a](#), [82bc](#), [83b](#), [84ab](#), [85b](#), [91ab](#), [93ab](#), [96](#), [98b](#), [99ab](#), [100b](#), [101c](#), [102c](#), [103b](#), [104a](#), [106a](#), [107b](#), [108a](#), [110a](#), [111a](#), [112a](#), [114a](#), [115a](#), [116a](#), [118a](#), [127b](#), [128a](#), [130b](#), [142d](#), [144](#).  
 PermutedLinearStatistic\_SLOT: [21a](#), [142ab](#), [143](#), [144](#).  
 Power1: [21a](#), [79c](#), [83b](#), [103b](#).  
 Power2: [21a](#), [82a](#), [85b](#).  
 PP12: [33a](#), [43](#), [45](#), [50](#), [77](#), [85b](#), [127b](#), [144](#), [145a](#).  
 Q: [14](#), [24b](#), [30bc](#), [32ab](#), [34abcd](#), [35a](#), [36](#), [40ab](#), [41a](#), [42b](#), [43](#), [44](#), [45](#), [47](#), [50](#), [51a](#), [53](#), [68](#), [69](#), [70ab](#), [71ab](#), [72](#), [74ab](#), [75ad](#), [76b](#), [77](#), [78a](#), [79ac](#), [81a](#), [82a](#), [91b](#), [93ab](#), [96](#), [98b](#), [99b](#), [100b](#), [101c](#), [102c](#), [111a](#), [112a](#), [114a](#), [115a](#), [116a](#), [118a](#), [128a](#), [142d](#), [144](#), [145a](#).  
 RC\_colSums: [79c](#), [82a](#), [83b](#), [85b](#), [103bc](#), [104a](#).  
 RC\_CovarianceInfluence: [34b](#), [43](#), [81ab](#), [82a](#).  
 RC\_CovarianceX: [34cd](#), [43](#), [84b](#), [85a](#), [85b](#).  
 RC\_ExpectationCovarianceStatistic: [30b](#), [31](#), [44](#).  
 RC\_ExpectationInfluence: [34a](#), [42b](#), [79ab](#), [79c](#).  
 RC\_ExpectationX: [34a](#), [42b](#), [82c](#), [83a](#), [83b](#).  
 RC\_init\_LECV\_1d: [30b](#), [145b](#).  
 RC\_init\_LECV\_2d: [40a](#), [146](#).  
 RC\_KronSums: [75d](#), [82a](#), [85b](#), [91bc](#), [92a](#).  
 RC\_KronSums\_Permutation: [36](#), [99b](#), [100a](#), [100b](#).  
 RC\_LinearStatistic: [32b](#), [75c](#), [75d](#).  
 RC\_OneTableSums: [33a](#), [36](#), [83b](#), [107bc](#), [108a](#).  
 RC\_order\_subset\_wrt\_block: [33a](#), [36](#), [119bc](#), [120a](#).  
 RC\_setup\_subset: [36](#), [122b](#), [123a](#).  
 RC\_Sums: [33ab](#), [79a](#), [81a](#), [87bc](#), [88a](#), [119b](#), [123a](#).  
 RC\_ThreeTableSums: [40a](#), [115ab](#), [116a](#).  
 RC\_TwoTableSums: [40a](#), [111ab](#), [112a](#).  
 R\_colSums: [103a](#), [103b](#), [149](#), [150](#).  
 R\_CovarianceInfluence: [80](#), [81a](#), [149](#), [150](#).  
 R\_CovarianceX: [84a](#), [84b](#), [149](#), [150](#).  
 R\_ExpectationCovarianceStatistic: [6](#), [29c](#), [30a](#), [30b](#), [149](#), [150](#), [159ab](#), [160bcd](#).  
 R\_ExpectationCovarianceStatistic\_2d: [8](#), [39ab](#), [40a](#), [149](#), [150](#).  
 R\_ExpectationInfluence: [78b](#), [79a](#), [81a](#), [149](#), [150](#).  
 R\_ExpectationX: [82b](#), [82c](#), [84b](#), [149](#), [150](#).  
 R\_KronSums: [91a](#), [91b](#), [149](#), [150](#).  
 R\_KronSums\_Permutation: [99a](#), [99b](#), [149](#), [150](#).  
 R\_MPinv\_sym: [131ab](#), [132](#), [149](#), [150](#), [154a](#), [155a](#).  
 R\_OneTableSums: [15b](#), [107a](#), [107b](#), [119b](#), [149](#), [150](#).  
 R\_order\_subset\_wrt\_block: [119a](#), [119b](#), [149](#), [150](#).  
 R\_pack\_sym: [136ab](#), [136c](#), [149](#), [150](#), [154d](#).  
 R\_PermutedLinearStatistic: [6](#), [35bc](#), [36](#), [149](#), [150](#).  
 R\_PermutedLinearStatistic\_2d: [8](#), [46ab](#), [47](#), [48a](#), [149](#), [150](#).

R\_quadform: 59b, 60a, [60b](#), 149, 150, 155a.  
R\_Sums: [87a](#), [87b](#), 149, 150.  
R\_ThreeTableSums: 15b, 114b, [115a](#), 149, 150.  
R\_TwoTableSums: 15b, 110b, [111a](#), 149, 150.  
R\_unpack\_sym: 10, 134ab, [135](#), 149, 150, 155a.  
S: [20c](#), 34b, 35a, 43, 44, 56b, 57b, 58b, 61a, 62a, 66, 67a, 70b, 74b, 85b, 96, 130ab, 133, 139a.  
StandardisedPermutedLinearStatistic\_SLOT: [21a](#), 143, 144.  
subset: 3b, 4, 5ab, 6, 8, 15ab, 17, 19, [25c](#), [25f](#), [26a](#), 30ab, 31, 32b, 33ab, 35c, 36, 39b, 40a, 42b, 43, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86ac, 87b, 88a, 91b, 93ab, 99b, 100b, 101c, 102c, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 120a, 122a, 123a, 124abc, 125a, 160bd.  
sumweights: [25b](#), 31, 33ab, 34abcd, 42ab, 43, 45, 47, 48bd, 69, 70a, 71a, 75a, 77, 78a, 79ac, 81a, 82a, 123a, 141a.  
Sumweights\_SLOT: [21a](#), 141ad, 143, 144, 145b.  
TableBlock\_SLOT: [21a](#), 33a, 140d, 141d, 143, 144, 145b.  
Table\_SLOT: [21a](#), 141bc, 143, 144, 146.  
TESTSTAT\_maximum: [21a](#), 68, 70ab, 71ab, 72, 74b, 75a.  
TESTSTAT\_quadratic: [21a](#), 70a.  
tol\_SLOT: [21a](#), 142c, 143, 144.  
VarianceInfluence\_SLOT: [21a](#), 140c, 143, 144.  
Variance\_SLOT: [21a](#), 139a, 143, 144.  
varonly\_SLOT: [21a](#), 138a, 143, 144.  
weights: 3b, 4, 5a, 6, 8, 15ab, 17, 19, [24e](#), [24f](#), [25a](#), 30ab, 32b, 33b, 34abcd, 35c, 36, 39b, 40a, 48a, 75d, 79ac, 81a, 82ac, 83b, 84b, 85b, 86a, 87b, 88a, 91b, 93ab, 103b, 104a, 107b, 108a, 111a, 112a, 115a, 116a, 119b, 123a, 160bd.  
x: 8, 14, 17, 20c, [23c](#), [23e](#), [23f](#), 30abc, 32ab, 34acd, 35c, 36, 39b, 40ab, 41a, 42b, 43, 46b, 47, 75d, 82c, 83b, 84b, 85b, 91b, 92a, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 103b, 104a, 106a, 107b, 108a, 110a, 111a, 112a, 114a, 115a, 116a, 118a, 126ab, 127a, 130b, 131ab, 132, 133, 134ab, 135, 136abc, 154a, 159ab, 160bd.  
Xfactor\_SLOT: [21a](#), 138b, 143, 144.  
y: 14, 20c, [24a](#), [24c](#), [24d](#), 30abc, 32b, 34ab, 35c, 36, 39b, 40ab, 41a, 42b, 43, 46b, 75d, 79ac, 81a, 82a, 91b, 93ab, 96, 98b, 99b, 100b, 101c, 102c, 111a, 112a, 114a, 115a, 116a, 118a, 119b, 129b, 130a, 159ab, 160bd.

# Bibliography

Helmut Strasser and Christian Weber. The asymptotic theory of permutation statistics. *Mathematical Methods of Statistics*, 8(2):220–250, 1999. Preprint available from <https://epub.wu.ac.at/102>. 1