

The **libcoin** Package

Torsten Hothorn
Universität Zürich

December 13, 2017

Contents

| | | |
|----------|-------------------------------|------------|
| 1 | Introduction | 1 |
| 2 | R Code | 3 |
| 2.1 | R User Interface | 3 |
| 2.1.1 | One-Dimensional Case (“1d”) | 4 |
| 2.1.2 | Two-Dimensional Case (“2d”) | 7 |
| 2.1.3 | Methods and Tests | 10 |
| 2.1.4 | Tabulations | 15 |
| 2.2 | Manual Pages | 18 |
| 3 | C Code | 21 |
| 3.1 | Header and Source Files | 21 |
| 3.2 | Variables | 24 |
| 3.2.1 | Example Data and Code | 29 |
| 3.3 | Conventions | 31 |
| 3.4 | C User Interface | 31 |
| 3.4.1 | One-Dimensional Case (“1d”) | 31 |
| 3.4.2 | Two-Dimensional Case (“2d”) | 42 |
| 3.5 | Tests | 53 |
| 3.6 | Test Statistics | 60 |
| 3.7 | Linear Statistics | 79 |
| 3.8 | Expectation and Covariance | 80 |
| 3.8.1 | Linear Statistic | 80 |
| 3.8.2 | Influence | 82 |
| 3.8.3 | X | 86 |
| 3.9 | Computing Sums | 91 |
| 3.9.1 | Simple Sums | 92 |
| 3.9.2 | Kronecker Sums | 96 |
| 3.9.3 | Column Sums | 111 |
| 3.9.4 | Tables | 116 |
| 3.10 | Utilities | 130 |
| 3.10.1 | Blocks | 130 |
| 3.10.2 | Permutation Helpers | 135 |
| 3.10.3 | Other Utils | 138 |
| 3.11 | Memory | 145 |
| 4 | Package Infrastructure | 157 |

Licence

Copyright 2017 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 $\mu(\mathcal{A}) \in \mathbb{R}^{pq}$ and covariance $\Sigma(\mathcal{A}) \in \mathbb{R}^{pq \times pq}$ of \mathbf{T} under H_0 given all permutations $\sigma \in S(\mathcal{A})$ of the responses are derived by [Strasser](#)

and Weber (1999):

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

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

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

with corresponding $Q \times Q$ covariance matrix

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

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

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

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

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

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

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

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

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

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

Chapter 2

R Code

2.1 R User Interface

```
"libcoin.R" 3a≡
```

```
  < R Header 161a >  
  < LinStatExpCov 4 >  
  < LinStatExpCov1d 6 >  
  < LinStatExpCov2d 8 >  
  < vcov LinStatExpCov 10 >  
  < doTest 12 >  
  < Contrasts 14 >  
  ◊
```

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

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

Fragment referenced in 4, 18.

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

`<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))
    return(.LinStatExpCov1d(X = X, Y = Y, weights = weights,
                           subset = subset, block = block,
                           checkNAs = checkNAs,
                           varonly = varonly, nresample = nresample,
                           standardise = standardise, tol = tol))

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

  stop("incorrect call to LinStatExpCov")
}
◇
```

Fragment referenced in 3a.

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

2.1.1 One-Dimensional Case (“1d”)

We assume that \mathbf{x}_i and \mathbf{y}_i for $i = 1, \dots, N$ are available as numeric matrices `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 28bd, N 24bc, subset 27be, 28a, weights 26c.

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

⟨ *Handle Missing Values 5b* ⟩ ≡

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

Fragment referenced in 6.

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

The logical argument `varonly` triggers the computation of the diagonal elements of the covariance matrix Σ 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") <- 1: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 28bd, N 24bc, NROW 138b, R_ExpectationCovarianceStatistic 32c, R_PermutedLinearStatistic 40, subset 27be, 28a, weights 26c, weights, 26de.

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

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

```

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

[1] 8.8 9.5 10.3 9.8 10.5

> tapply(y, x, sum)

  1  2  3  4  5
8.8 9.5 10.3 9.8 10.5

```

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

```

> ls2 <- LinStatExpCov(X = x, Y = matrix(y, ncol = 1))
> all.equal(ls1, ls2)

[1] "Component Xfactor: 1 element mismatch"

```

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

2.1.2 Two-Dimensional Case (“2d”)

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

< LinStatExpCov2d 8 > ≡

```
.LinStatExpCov2d <-  
function(X = numeric(0), Y, ix, iy, weights = integer(0), subset = integer(0),  
        block = integer(0), checkNAs = TRUE, varonly = FALSE, nresample = 0,  
        standardise = FALSE,  
        tol = sqrt(.Machine$double.eps))  
{  
  
  IF <- function(x) is.integer(x) || is.factor(x)  
  
  if (!(length(ix) == length(iy)) && IF(ix) && IF(iy))  
    stop("incorrect ix and/or iy")  
  N <- length(ix)  
  
  < Check ix 9a >  
  
  < Check iy 9b >  
  
  if (length(X) > 0) {  
    if (!(NROW(X) == (length(attr(ix, "levels")) + 1) &&  
        all(complete.cases(X))))  
      stop("incorrect X")  
  }  
  
  if (!(NROW(Y) == (length(attr(iy, "levels")) + 1) &&  
      all(complete.cases(Y))))  
    stop("incorrect Y")  
  
  < Check weights, subset, block 5a >  
  
  ret <- .Call(R_ExpectationCovarianceStatistic_2d, X, ix, Y, iy,  
             weights, subset, block, as.integer(varonly), as.double(tol))  
  ret$varonly <- as.logical(ret$varonly)  
  ret$Xfactor <- as.logical(ret$Xfactor)  
  if (nresample > 0) {  
    ret$PermutedLinearStatistic <-  
      .Call(R_PermutedLinearStatistic_2d, X, ix, Y, iy, block, nresample, ret$Table)  
    if (standardise)  
      ret$StandardisedPermutedLinearStatistic <-  
        .Call(R_StandardisePermutedLinearStatistic, ret)  
  }  
  class(ret) <- c("LinStatExpCov2d", "LinStatExpCov")  
  ret  
}  
◇
```

Fragment referenced in [3a](#).

Uses: [block 28bd](#), [N 24bc](#), [NROW 138b](#), [R_ExpectationCovarianceStatistic_2d 44](#), [R_PermutedLinearStatistic_2d 51](#),
[subset 27be, 28a](#), [weights 26c](#), [weights, 26de](#), [x 24d, 25bc](#).

ix can be a factor but without any missing values

⟨ 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") <- 1:rg[2]
} else {
  ### lev can be data.frame (see inum::inum)
  lev <- attr(ix, "levels")
  if (!is.vector(lev)) lev <- 1:NROW(lev)
  attr(ix, "levels") <- lev
  if (checkNAs) stopifnot(!anyNA(ix))
}
◇
```

Fragment referenced in 8, 15b.

Uses: NROW 138b.

⟨ 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") <- 1:rg[2]
} else {
  ### lev can be data.frame (see inum::inum)
  lev <- attr(iy, "levels")
  if (!is.vector(lev)) lev <- 1:NROW(lev)
  attr(iy, "levels") <- lev
  if (checkNAs) stopifnot(!anyNA(iy))
}
◇
```

Fragment referenced in 8, 15b.

Uses: NROW 138b.

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, ls3)

[1] "Attributes: < Component ■class■: 1 string mismatch >"
[2] "Component ■TableBlock■: Mean relative difference: 1"
[3] "Component ■Table■: target is NULL, current is array"

> ### works also with factors
> ls3 <- LinStatExpCov(X = X, ix = factor(ix), Y = Y, iy = factor(iy))
> all.equal(ls1, ls3)
```

```
[1] "Attributes: < Component ■class■: 1 string mismatch >"
[2] "Component ■TableBlock■: Mean relative difference: 1"
[3] "Component ■Table■: target is NULL, current is array"
```

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, ls4)
```

```
[1] "Component ■Xfactor■: 1 element mismatch"
```

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$varonly)
    stop("cannot extract covariance matrix")
  PQ <- prod(object$dim)
  ret <- matrix(0, nrow = PQ, ncol = PQ)
  ret[lower.tri(ret, diag = TRUE)] <- object$Covariance
  ret <- ret + t(ret)
  diag(ret) <- diag(ret) / 2
  ret
}
```

Fragment referenced in [3a](#).

```
> 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_{quad} along with the corresponding p -values. If `nresample = 0` was used in the call to `LinStatExpCov()`, p -values are obtained from the limiting asymptotic distribution whenever such a thing is available at reasonable costs. Otherwise, the permutation p -value is returned (along with the permuted test statistics when `PermutedStatistics` is `TRUE`). The p -values (`lower = FALSE`) or $(1 - p)$ -values (`lower = TRUE`) can be computed on the log-scale.

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

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

< doTest 12 > ≡

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

  if (maxselect)
    stopifnot(object$Xfactor)

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

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

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

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

Fragment referenced in [3a](#).
Uses: [NCOL 138c](#).

```

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

$TestStatistic
[1] 0.8411982

$p.value
[1] NA

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

$TestStatistic
[1] 0.8411982

$p.value
[1] 0.8852087

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

$TestStatistic
[1] 0.8411982

$p.value
[1] 0.108822

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

$TestStatistic
[1] 0.8411982

$p.value
[1] 0.1150168

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

$TestStatistic
[1] 0.8411982

$p.value
[1] -2.164164

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

$TestStatistic
[1] 1.077484

$p.value
[1] 0.897828

```


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

`< Contrasts 14 > ≡`

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

Fragment referenced in 3a.

Uses: `NROW` 138b, `P` 25a, `Q` 25e, `x` 24d, 25bc, `y` 25d, 26ab.

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

```
> set.seed(29)
> ls1d <- LinStatExpCov(X = model.matrix(~ x - 1), Y = matrix(y, ncol = 1),
+                       nresample = 10, standardise = TRUE)
```

```

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

```

2.1.4 Tabulations

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

```

<ctabs Prototype 15a> ≡
  (ix, iy = integer(0), block = integer(0), weights = integer(0),
   subset = integer(0), checkNAs = TRUE)◊

```

Fragment referenced in [15b](#), [20](#).

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

"ctabs.R" 15b≡

```

<R Header 161a>
ctabs <- 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 && length(block) == 0)
    return(.Call(R_OneTableSums, ix, weights, subset))
  if (length(block) == 0)
    return(.Call(R_TwoTableSums, ix, iy, weights, subset))
  if (length(iy) == 0)
    return(.Call(R_TwoTableSums, ix, block, weights, subset)[,-1,drop = FALSE])
  return(.Call(R_ThreeTableSums, ix, iy, block, weights, subset))
}
◊

```

Uses: [block 28bd](#), [N 24bc](#), [R_OneTableSums 117a](#), [R_ThreeTableSums 126b](#), [R_TwoTableSums 121b](#), [subset 27be, 28a](#), [weights 26c](#), [weights, 26de](#).

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

[1] 0
```


2.2 Manual Pages

"LinStatExpCov.Rd" 18≡

```
\name{LinStatExpCov}
\alias{LinStatExpCov}
\alias{lmult}
\title{
  Linear Statistics with Expectation and Covariance
}
\description{
  Strasser-Weber type linear statistics and their expectation
  and covariance under the independence hypothesis
}
\usage{
LinStatExpCov(LinStatExpCov Prototype 3b)
lmult(x, object)
}
\arguments{
  \item{X}{numeric matrix of transformations.}
  \item{Y}{numeric matrix of influence functions.}
  \item{ix}{an optional integer vector expanding {X}.}
  \item{iy}{an optional integer vector expanding {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 {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 {X} was a factor.}
  \item{object}{an object of class {LinStatExpCov}.}
}
\details{
  The function, after minimal preprocessing, calls the underlying C code
  and computes the linear statistic, its expectation and covariance and,
  optionally, {nresample} samples from its permutation distribution.

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

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

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

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

aq <- subset(airquality, Month \%in\% c(5, 8))
}
```

"doTest.Rd" 19≡

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

"ctabs.Rd" 20≡

```
\name{ctabs}
\alias{ctabs}
\title{
  Cross Tabulation
}
\description{
  Efficient weighted cross tabulation of two factors and a block
}
\usage{
ctabs(ctabs Prototype 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 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 xtabs(weights ~ ix + iy + block, subset).
}
\value{
  If block is present, a three-way table. Otherwise,
  a one- or two-dimensional table.
}
\examples{

  ctabs(ix = 1:5, iy = 1:5, weights = 1:5 / 5)

}
\keyword{univar}
◇
```

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

Chapter 3

C Code

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

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

3.1 Header and Source Files

```
"libcoin_internal.h" 21a≡  
  
    < C Header 161b >  
    < R Includes 21b >  
    < C Macros 22a >  
    < C Global Variables 22b >  
    ◇
```

These includes provide some R infrastructure at C level.

```
< R Includes 21b > ≡  
  
    #include <R.h>  
    #include <Rinternals.h>  
    #include <Rmath.h>  
    #include <Rdefines.h>  
    #include <R_ext/stats_package.h> /* for S_rcont2 */  
    #include <R_ext/Applic.h> /* for dgemm */  
    #include <R_ext/Lapack.h> /* for dgesdd */  
    ◇
```

Fragment referenced in 21a.

We need three macros: **S** computes the element Σ_{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 22a > ≡

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

Fragment referenced in 21a.

Defines: GE 55, 57, LE 57, S 37b, 38b, 47, 48, 60b, 61b, 62b, 63b, 65a, 69, 70a, 74a, 78b, 91a, 103, 142, 143, 144, 147c.

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

< C Global Variables 22b > ≡

```
#define ALTERNATIVE_twosided 1  
#define ALTERNATIVE_less 2  
#define ALTERNATIVE_greater 3  
  
#define TESTSTAT_maximum 1  
#define TESTSTAT_quadratic 2  
  
#define LinearStatistic_SLOT 0  
#define Expectation_SLOT 1  
#define Covariance_SLOT 2  
#define Variance_SLOT 3  
#define ExpectationX_SLOT 4  
#define varonly_SLOT 5  
#define dim_SLOT 6  
#define ExpectationInfluence_SLOT 7  
#define CovarianceInfluence_SLOT 8  
#define VarianceInfluence_SLOT 9  
#define Xfactor_SLOT 10  
#define tol_SLOT 11  
#define PermutedLinearStatistic_SLOT 12  
#define StandardisedPermutedLinearStatistic_SLOT 13  
#define TableBlock_SLOT 14  
#define Sumweights_SLOT 15  
#define Table_SLOT 16  
  
#define DoSymmetric 1  
#define DoCenter 1  
#define DoVarOnly 1  
#define Power1 1  
#define Power2 2  
#define Offset0 0  
◇
```

Fragment referenced in 21a.

Defines: CovarianceInfluence_SLOT 149a, 153, 154, Covariance_SLOT 147c, 148a, 153, 154, dim_SLOT 145c, 146a, 153, 154, DoCenter 79d, 84b, 86b, 88, 91a, 98a, 112a, DoSymmetric 79d, 86b, 91a, DoVarOnly 37bc, 38a, 47, ExpectationInfluence_SLOT 148c, 153, 154, ExpectationX_SLOT 148b, 153, 154, Expectation_SLOT 147b, 153, 154, LinearStatistic_SLOT 147a, 153, 154, Offset0 35b, 36a, 40, 44, 46c, 47, 83b, 85b, 87a, 90a, 93b, 98a, 107b, 112a, 117a, 121b, 126b, 131b, 135a, PermutedLinearStatistic_SLOT 151bc, 153, 154, Power1 84b, 88, 112a, Power2 86b, 91a, StandardisedPermutedLinearStatistic_SLOT 153, 154, Sumweights_SLOT 150a, 151a, 153, 154, 155b, TableBlock_SLOT 36a, 149c, 151a, 153, 154, 155b, Table_SLOT 150bc, 153, 154, 156, tol_SLOT 152a, 153, 154, VarianceInfluence_SLOT 149b, 153, 154, Variance_SLOT 147c, 153, 154, varonly_SLOT 146b, 153, 154, Xfactor_SLOT 146c, 153, 154.

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

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

"libcoin.h" 23a≡

```
⟨ C Header 161b ⟩
#include "libcoin_internal.h"
⟨ Function Prototypes 23b ⟩
◇
```

⟨ Function Prototypes 23b ⟩ ≡

```
extern ⟨ R_ExpectationCovarianceStatistic Prototype 32b ⟩;
extern ⟨ R_PermutedLinearStatistic Prototype 38c ⟩;
extern ⟨ R_StandardisePermutedLinearStatistic Prototype 41c ⟩;
extern ⟨ R_ExpectationCovarianceStatistic_2d Prototype 43a ⟩;
extern ⟨ R_PermutedLinearStatistic_2d Prototype 50a ⟩;
extern ⟨ R_QuadraticTest Prototype 54 ⟩;
extern ⟨ R_MaximumTest Prototype 56b ⟩;
extern ⟨ R_MaximallySelectedTest Prototype 58 ⟩;
extern ⟨ R_ExpectationInfluence Prototype 83a ⟩;
extern ⟨ R_CovarianceInfluence Prototype 85a ⟩;
extern ⟨ R_ExpectationX Prototype 86c ⟩;
extern ⟨ R_CovarianceX Prototype 89 ⟩;
extern ⟨ R_Sums Prototype 93a ⟩;
extern ⟨ R_KronSums Prototype 97 ⟩;
extern ⟨ R_KronSums_Permutation Prototype 107a ⟩;
extern ⟨ R_colSums Prototype 111b ⟩;
extern ⟨ R_OneTableSums Prototype 116b ⟩;
extern ⟨ R_TwoTableSums Prototype 121a ⟩;
extern ⟨ R_ThreeTableSums Prototype 126a ⟩;
extern ⟨ R_order_subset_wrt_block Prototype 131a ⟩;
extern ⟨ R_kronecker Prototype 140a ⟩;
◇
```

Fragment referenced in 23a.

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

"libcoin.c" 23c≡

```
⟨ C Header 161b ⟩
#include "libcoin_internal.h"
#include <R_ext/stats_stubs.h> /* for S_rcont2 */
#include <mvtnormAPI.h> /* for calling mvtnorm */
⟨ Function Definitions 24a ⟩
◇
```

< Function Definitions 24a > ≡

< MoreUtils 138a >
< Memory 145a >
< P-Values 65b >
< KronSums 96b >
< colSums 111a >
< SimpleSums 92c >
< Tables 116a >
< Utils 130b >
< LinearStatistics 79b >
< Permutations 135b >
< ExpectationCovariances 80a >
< Test Statistics 60a >
< User Interface 31a >
< 2d User Interface 42b >
< Tests 53a >

◇

Fragment referenced in 23c.

3.2 Variables

N is the number of observations

< R N Input 24b > ≡

SEXP N ,

◇

Fragment referenced in 93a.

Defines: N 5ab, 6, 8, 15b, 24c, 35ab, 36ab, 37abc, 38a, 40, 44, 68, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92a, 93b, 94a, 96a, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 132b, 133ab, 134a, 135a, 143.

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

< C integer N Input 24c > ≡

`R_xlen_t` N

◇

Fragment referenced in 25bc, 34, 40, 44, 79c, 83b, 84a, 85b, 86a, 87ab, 90ab, 93c, 94b, 95abc, 98a, 99b, 107bc, 112a, 117a, 121b, 126b, 131b, 132a, 133ab, 134b.

Defines: N 5ab, 6, 8, 15b, 24b, 35ab, 36ab, 37abc, 38a, 40, 44, 68, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92a, 93b, 94a, 96a, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 132b, 133ab, 134a, 135a, 143.

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

< R x Input 24d > ≡

SEXP \mathbf{x} ,

◇

Fragment referenced in 31b, 42c, 50a, 79c, 86c, 87b, 89, 90b, 97, 99b, 107ac, 111b, 116b, 121a, 126a.

Defines: \mathbf{x} 8, 14, 18, 22a, 25bc, 32ac, 33, 35ab, 37ac, 38ad, 40, 43b, 44, 45ab, 46c, 47, 50b, 51, 79d, 87a, 88, 90a, 91a, 98a, 99a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 120b, 121b, 122b, 125, 126b, 127b, 130a, 138bc, 139a, 143, 144.

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

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

```
int P
◇
```

Fragment referenced in 25bc, 34, 79c, 80b, 81, 82, 87b, 90b, 99b, 107c, 155b, 156.

Defines: P 14, 32c, 33, 35ab, 36a, 37ac, 38ab, 40, 44, 45ab, 46c, 47, 48, 49, 51, 55, 56a, 57, 59, 71, 72, 73, 74a, 76, 77ab, 78ab, 79d, 80b, 81, 82, 86c, 87a, 88, 89, 90a, 91a, 97, 98a, 100, 101a, 103, 106, 107ab, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 120b, 121b, 122b, 125, 126b, 127b, 130a, 143, 152b, 154.

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

```
double *x,
⟨ C integer N Input 24c ⟩,
⟨ C integer P Input 25a ⟩,
◇
```

Fragment referenced in 99c, 108b, 109a, 113b, 143.

Defines: x 8, 14, 18, 22a, 24d, 25c, 32ac, 33, 35ab, 37ac, 38ad, 40, 43b, 44, 45ab, 46c, 47, 50b, 51, 79d, 87a, 88, 90a, 91a, 98a, 99a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 120b, 121b, 122b, 125, 126b, 127b, 130a, 138bc, 139a, 143, 144.

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

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

```
int *x,
⟨ C integer N Input 24c ⟩,
⟨ C integer P Input 25a ⟩,
◇
```

Fragment referenced in 104a, 110ab, 118b, 122c, 127c.

Defines: x 8, 14, 18, 22a, 24d, 25b, 32ac, 33, 35ab, 37ac, 38ad, 40, 43b, 44, 45ab, 46c, 47, 50b, 51, 79d, 87a, 88, 90a, 91a, 98a, 99a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 120b, 121b, 122b, 125, 126b, 127b, 130a, 138bc, 139a, 143, 144.

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

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

```
SEXP y,
◇
```

Fragment referenced in 31b, 42c, 50a, 83a, 84a, 85a, 86a, 97, 107a, 121a, 126a, 131a.

Defines: y 14, 22a, 26ab, 32ac, 33, 35b, 37ab, 38d, 40, 43b, 44, 45ab, 46c, 47, 50b, 79d, 83b, 84b, 85b, 86b, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 121b, 122b, 125, 126b, 127b, 130a, 131b, 141, 142.

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

```
int Q
◇
```

Fragment referenced in 26ab, 34, 80b, 81, 82, 83b, 84a, 85b, 86a, 98a, 107b, 155b, 156.

Defines: Q 14, 32c, 33, 35ab, 37abc, 38ab, 40, 44, 45ab, 46c, 47, 48, 49, 51, 55, 56a, 57, 71, 72, 73, 74abc, 76, 78ab, 79ad, 80b, 81, 82, 83b, 84b, 85b, 86b, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 121b, 122b, 125, 126b, 127b, 130a, 152b, 154, 155a.

< C real y Input 26a > ≡

```
    double *y,  
    < C integer Q Input 25e >,  
    ◇
```

Fragment referenced in 79c, 99bc, 104a, 107c, 108b, 109a, 110ab.

Defines: **y** 14, 22a, 25d, 26b, 32ac, 33, 35b, 37ab, 38d, 40, 43b, 44, 45ab, 46c, 47, 50b, 79d, 83b, 84b, 85b, 86b, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 121b, 122b, 125, 126b, 127b, 130a, 131b, 141, 142.

or a factor at Q levels

< C integer y Input 26b > ≡

```
    int *y,  
    < C integer Q Input 25e >,  
    ◇
```

Fragment referenced in 122c, 127c.

Defines: **y** 14, 22a, 25d, 26a, 32ac, 33, 35b, 37ab, 38d, 40, 43b, 44, 45ab, 46c, 47, 50b, 79d, 83b, 84b, 85b, 86b, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 121b, 122b, 125, 126b, 127b, 130a, 131b, 141, 142.

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

< R weights Input 26c > ≡

```
    SEXP weights  
    ◇
```

Fragment referenced in 31b, 42c, 79c, 83a, 84a, 85a, 86ac, 87b, 89, 90b, 93ac, 97, 98b, 111b, 112b, 116b, 117b, 121a, 122a, 126a, 127a, 131a, 134b.

Defines: **weights** 3b, 4, 5a, 6, 8, 15ab, 18, 20, 26de, 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 52a, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 93b, 94a, 98a, 100, 101a, 112a, 113a, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 135a.

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 26d > ≡

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

Fragment referenced in 95ab, 102ab, 104c, 105a, 114bc, 119bc, 123c, 124a, 128c, 129a.

Defines: **HAS_WEIGHTS** 26e, 96a, 103, 106, 115b, 120b, 125, 130a, **weights**, 4, 6, 8, 15b, 20, 26e, 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91a, 93b, 98a, 112a, 117a, 121b, 126b, 131b, 135a.

Uses: **weights** 26c.

Weights larger than INT_MAX are stored as double

< C real weights Input 26e > ≡

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

Fragment referenced in 94b, 95c, 101b, 102c, 104b, 105b, 114a, 115a, 119a, 120a, 123b, 124b, 128b, 129b.

Defines: **HAS_WEIGHTS** 26d, 96a, 103, 106, 115b, 120b, 125, 130a, **weights**, 4, 6, 8, 15b, 20, 26d, 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91a, 93b, 98a, 112a, 117a, 121b, 126b, 131b, 135a.

Uses: **weights** 26c.

The sum of all weights is a double

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

```
double sumweights
◇
```

Fragment referenced in 81, 82, 84a, 86a.

Defines: **sumweights** 34, 36ab, 37abc, 38a, 46bc, 47, 49, 51, 52bd, 72, 73, 74b, 79a, 81, 82, 83b, 84b, 85b, 86b, 135a, 150a.

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

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

```
SEXP subset
◇
```

Fragment referenced in 31b, 42c, 79c, 83a, 84a, 85a, 86ac, 87b, 89, 90b, 93ac, 97, 98b, 107ac, 111b, 112b, 116b, 117b, 121a, 122a, 126a, 127a, 131a, 132a, 134ab.

Defines: **subset** 3b, 4, 5ab, 6, 8, 15ab, 18, 20, 27e, 28a, 32ac, 34, 35b, 36ab, 38d, 40, 43b, 44, 46c, 47, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92b, 93b, 94a, 98a, 100, 101a, 107b, 108a, 109b, 110c, 112a, 113a, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 132b, 134a, 135a, 136ab, 137ab.

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

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

```
R_xlen_t Nsubset
◇
```

Fragment referenced in 27d, 40, 44, 83b, 85b, 87a, 90a, 93b, 98a, 107b, 112a, 117a, 121b, 126b, 136ab, 137b.

Defines: **Nsubset** 36b, 40, 44, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92ab, 93b, 94a, 96a, 98a, 100, 101a, 107b, 108a, 109b, 110c, 112a, 113a, 117a, 118a, 121b, 122b, 126b, 127b, 136ab, 137b.

Optionally, one can specify a subset of the subset via

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

```
R_xlen_t offset,
   $\langle C \text{ integer Nsubset Input 27c} \rangle$ 
◇
```

Fragment referenced in 27e, 28a, 79c, 84a, 86a, 87b, 90b, 93c, 98b, 107c, 112b, 117b, 122a, 127a.

Defines: **offset** 34, 36b, 37abc, 38a, 79d, 84b, 86b, 88, 91ab, 94a, 100, 101a, 108a, 109b, 110c, 113a, 118a, 122b, 127b.

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

Subsets are stored either as integer

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

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

Fragment referenced in 95bc, 102bc, 105ab, 109a, 110b, 114c, 115a, 119c, 120a, 124ab, 129ab.

Defines: **subset** 3b, 4, 5ab, 6, 8, 15ab, 18, 20, 27b, 28a, 32ac, 34, 35b, 36ab, 38d, 40, 43b, 44, 46c, 47, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92b, 93b, 94a, 98a, 100, 101a, 107b, 108a, 109b, 110c, 112a, 113a, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 132b, 134a, 135a, 136ab, 137ab.

or double (to allow for indices larger than INT_MAX)

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

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

Fragment referenced in 94b, 95a, 101b, 102a, 104bc, 108b, 110a, 114ab, 119ab, 123bc, 128bc.

Defines: `subset` 3b, 4, 5ab, 6, 8, 15ab, 18, 20, 27be, 32ac, 34, 35b, 36ab, 38d, 40, 43b, 44, 46c, 47, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92b, 93b, 94a, 98a, 100, 101a, 107b, 108a, 109b, 110c, 112a, 113a, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 132b, 134a, 135a, 136ab, 137ab.

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

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

```
SEXP block  
◇
```

Fragment referenced in 31b, 42c, 50a, 126a, 131a, 132a, 133b, 134a.

Defines: `block` 3b, 4, 5a, 6, 8, 15ab, 18, 20, 28d, 32ac, 33, 36ab, 38d, 40, 43b, 44, 45a, 50b, 126b, 127b, 130a, 131b, 132b, 133b, 134a, 149c.

at B levels

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

```
int B  
◇
```

Fragment referenced in 28d, 34, 155b, 156.

Defines: `B` 32c, 33, 34, 35a, 36a, 40, 44, 45a, 46a, 48, 49, 51, 52b, 71, 72, 76, 126b, 127b, 130a, 139b, 140ab, 141, 142, 152b, 154, 155b, 156.

are stored as a factor

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

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

Fragment referenced in 127c.

Defines: `block` 3b, 4, 5a, 6, 8, 15ab, 18, 20, 28b, 32ac, 33, 36ab, 38d, 40, 43b, 44, 45a, 50b, 126b, 127b, 130a, 131b, 132b, 133b, 134a, 149c.

The tabulation of block (potentially in subsets) is

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

```
SEXP blockTable  
◇
```

Fragment referenced in 132a, 133b, 134a.

Defines: `blockTable` 40, 131b, 132b, 133b, 134a.

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

3.2.1 Example Data and Code

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

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

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

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



```

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

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

```

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

```

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

```

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

```

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

```

```

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

```

All three implementations give the same results.

3.3 Conventions

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

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

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

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

3.4 C User Interface

3.4.1 One-Dimensional Case (“1d”)

< User Interface 31a > ≡

```

  < RC_ExpectationCovarianceStatistic 34 >
  < R_ExpectationCovarianceStatistic 32c >
  < R_PermutatedLinearStatistic 40 >
  < R_StandardisePermutatedLinearStatistic 42a >
  ◇

```

Fragment referenced in [24a](#).

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

< User Interface Inputs 31b > ≡

```

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

```

Fragment referenced in [32b](#), [34](#), [38c](#).

This function can be called from other packages.

"libcoinAPI.h" 32a≡

```
⟨ C Header 161b ⟩
#include <R_ext/Rdynload.h>
#include <libcoin.h>

extern SEXP libcoin_R_ExpectationCovarianceStatistic(
    SEXP x, SEXP y, SEXP weights, SEXP subset, SEXP block, SEXP varonly,
    SEXP tol
) {

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

File defined by 32a, 38d, 41b, 43b, 50b, 53b, 139b.

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

⟨ R_ExpectationCovarianceStatistic Prototype 32b ⟩ ≡

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

Fragment referenced in 23b, 32c.

Uses: R_ExpectationCovarianceStatistic 32c.

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

⟨ R_ExpectationCovarianceStatistic 32c ⟩ ≡

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

    ⟨ Setup Dimensions 33 ⟩

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

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

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

Fragment referenced in 31a.

Defines: R_ExpectationCovarianceStatistic 6, 32ab, 159, 160.

Uses: B 28c, block 28bd, P 25a, Q 25e, RC_ExpectationCovarianceStatistic 34, 48, RC_init_LECV_1d 155b, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

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

⟨ Setup Dimensions 33 ⟩ ≡

```
int P, Q, B;

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

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

◇

Fragment referenced in 32c, 40.

Uses: B 28c, block 28bd, NCOL 138c, NLEVELS 139a, P 25a, Q 25e, x 24d, 25bc, y 25d, 26ab.

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

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

< RC_ExpectationCovarianceStatistic 34 > ≡

```
void RC_ExpectationCovarianceStatistic
(
  < User Interface Inputs 31b >
  SEXP ans
) {

  < C integer N Input 24c >;
  < C integer P Input 25a >;
  < C integer Q Input 25e >;
  < C integer B Input 28c >;
  double *sumweights, *table;
  double *ExpInf, *VarInf, *CovInf, *ExpX, *ExpXtotal, *VarX, *CovX;
  double *tmpV, *tmpCV;
  SEXP nullvec, subset_block;

  < Extract Dimensions 35a >

  < Compute Linear Statistic 35b >

  < Setup Memory and Subsets in Blocks 36a >

  /* 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 36b >

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

      < Compute Expectation Linear Statistic 37a >

      < Compute Covariance Influence 37b >

      if (C_get_varonly(ans)) {
        < Compute Variance Linear Statistic 37c >
      } else {
        < Compute Covariance Linear Statistic 38a >
      }
    }

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

  < Compute Variance from Covariance 38b >

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

Fragment referenced in [31a](#).

Defines: [RC_ExpectationCovarianceStatistic 32c](#).

Uses: [B 28c](#), [C_get_varonly 146b](#), [offset 27d](#), [subset 27be, 28a](#), [sumweights 27a](#).

The dimensions are available from the return object:

⟨ Extract Dimensions 35a ⟩ ≡

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

Fragment referenced in 34.

Uses: B 28c, C_get_B 151a, C_get_P 145c, C_get_Q 146a, N 24bc, NROW 138b, P 25a, Q 25e, x 24d, 25bc.

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

⟨ Compute Linear Statistic 35b ⟩ ≡

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

Fragment referenced in 34.

Uses: C_get_LinearStatistic 147a, N 24bc, Offset0 22b, P 25a, Q 25e, RC_LinearStatistic 79d, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

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

⟨ Setup Memory and Subsets in Blocks 36a ⟩ ≡

```

ExpInf = C_get_ExpectationInfluence(ans);
VarInf = C_get_VarianceInfluence(ans);
CovInf = C_get_CovarianceInfluence(ans);
ExpXtotal = C_get_ExpectationX(ans);
for (int p = 0; p < P; p++) ExpXtotal[p] = 0.0;
ExpX = Calloc(P, double);
VarX = Calloc(P, double);
CovX = Calloc(P * (P + 1) / 2, double);
table = C_get_TableBlock(ans);
sumweights = C_get_Sumweights(ans);
PROTECT(nullvec = allocVector(INTSXP, 0));

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

```

Fragment referenced in 34.

Uses: B 28c, block 28bd, C_get_CovarianceInfluence 149a, C_get_ExpectationInfluence 148c, C_get_ExpectationX 148b, C_get_Sumweights 150a, C_get_TableBlock 149c, C_get_VarianceInfluence 149b, N 24bc, Offset0 22b, P 25a, RC_OneTableSums 118a, RC_order_subset_wrt_block 132b, RC_Sums 94a, subset 27be, 28a, sumweights 27a, TableBlock_SLOT 22b.

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

⟨ Compute Sum of Weights in Block 36b ⟩ ≡

```

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

```

Fragment referenced in 34.

Uses: block 28bd, N 24bc, Nsubset 27c, offset 27d, RC_Sums 94a, subset 27be, 28a, sumweights 27a, weights 26c, weights, 26de.

⟨ Compute Expectation Linear Statistic 37a ⟩ ≡

```

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

```

Fragment referenced in 34.

Uses: [C_ExpectationLinearStatistic 80b](#), [C_get_Expectation 147b](#), [N 24bc](#), [offset 27d](#), [P 25a](#), [Q 25e](#),
[RC_ExpectationInfluence 84b](#), [RC_ExpectationX 88](#), [sumweights 27a](#), [weights 26c](#), [weights, 26de](#), [x 24d, 25bc](#), [y 25d, 26ab](#).

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

⟨ Compute Covariance Influence 37b ⟩ ≡

```

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

```

Fragment referenced in 34.

Uses: [C_ordered_Xfactor 71](#), [C_unordered_Xfactor 76](#), [DoVarOnly 22b](#), [N 24bc](#), [offset 27d](#), [Q 25e](#),
[RC_CovarianceInfluence 86b](#), [S 22a](#), [sumweights 27a](#), [weights 26c](#), [weights, 26de](#), [y 25d, 26ab](#).

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

⟨ Compute Variance Linear Statistic 37c ⟩ ≡

```

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

```

Fragment referenced in 34.

Uses: [C_get_Variance 147c](#), [C_VarianceLinearStatistic 82](#), [DoVarOnly 22b](#), [N 24bc](#), [offset 27d](#), [P 25a](#), [Q 25e](#),
[RC_CovarianceX 91a](#), [sumweights 27a](#), [weights 26c](#), [weights, 26de](#), [x 24d, 25bc](#).

⟨ Compute Covariance Linear Statistic 38a ⟩ ≡

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

◇

Fragment referenced in 34.

Uses: C_CovarianceLinearStatistic 81, C_get_Covariance 148a, DoVarOnly 22b, N 24bc, offset 27d, P 25a, Q 25e, RC_CovarianceX 91a, sumweights 27a, weights 26c, weights, 26de, x 24d, 25bc.

⟨ Compute Variance from Covariance 38b ⟩ ≡

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

◇

Fragment referenced in 34.

Uses: C_get_Covariance 148a, C_get_Variance 147c, C_get_varonly 146b, P 25a, Q 25e, S 22a.

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

⟨ R_PermutedLinearStatistic Prototype 38c ⟩ ≡

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

Fragment referenced in 23b, 40.

Uses: R_PermutedLinearStatistic 40.

"libcoinAPI.h" 38d≡

```
extern SEXP libcoin_R_PermutedLinearStatistic(
    SEXP x, SEXP y, SEXP weights, SEXP subset, SEXP block, SEXP nresample
) {

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

File defined by 32a, 38d, 41b, 43b, 50b, 53b, 139b.

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

The dimensions are extracted from the data in the same ways as above. The function differentiates between the absence and presence of blocks. Weights are removed by expanding subset accordingly. Once within-block permutations were set-up the Kronecker product of \mathbf{X} and \mathbf{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_PermutatedLinearStatistic 40 > ≡

```
< R_PermutatedLinearStatistic Prototype 38c >
{
  SEXP ans, expand_subset, block_subset, perm, tmp, blockTable;
  double *linstat;
  int PQ;
  < C integer N Input 24c >;
  < C integer Nsubset Input 27c >;
  R_xlen_t inresample;

  < Setup Dimensions 33 >
  PQ = P * Q;
  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 41a >
      C_doPermute(REAL(expand_subset), Nsubset, REAL(tmp), REAL(perm));

      RC_KronSums_Permutation(x, NROW(x), P, REAL(y), Q, expand_subset,
                             Offset0, Nsubset, perm, linstat);
    }
  } else {
    PROTECT(blockTable = allocVector(REALSXP, B + 1));
    /* same as RC_OneTableSums(block, noweights, expand_subset) */
    RC_OneTableSums(INTEGER(block), XLENGTH(block), B + 1, weights, subset, Offset0,
                    XLENGTH(subset), REAL(blockTable));
    PROTECT(block_subset = RC_order_subset_wrt_block(XLENGTH(block), expand_subset,
                                                      block, blockTable));

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

  UNPROTECT(4);
  return(ans);
}
◇
```

Fragment referenced in 31a.

Defines: R_PermutatedLinearStatistic 6, 38cd, 159, 160.

Uses: B 28c, block 28bd, blockTable 28e, C_doPermute 136b, C_doPermuteBlock 137b, N 24bc, NROW 138b, Nsubset 27c, Offset0 22b, P 25a, Q 25e, RC_KronSums_Permutation 108a, RC_OneTableSums 118a, RC_order_subset_wrt_block 132b, RC_setup_subset 135a, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

Setup Linear Statistic 41a ≡

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

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

"libcoinAPI.h" 41b≡

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

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [53b](#), [139b](#).
Uses: [LECV 145b](#).

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

R_StandardisePermutedLinearStatistic Prototype 41c ≡

```
SEXP R_StandardisePermutedLinearStatistic
(
    SEXP LECV
)
◇
```

Fragment referenced in [23b](#), [42a](#).
Uses: [LECV 145b](#).

< R_StandardisePermutedLinearStatistic 42a > ≡

```
< R_StandardisePermutedLinearStatistic Prototype 41c >
{
  SEXP ans;
  R_xlen_t nresample = C_get_nresample(LECV);
  double *ls;
  if (!nresample) return(R_NilValue);
  int PQ = C_get_P(LECV) * C_get_Q(LECV);

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

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

Fragment referenced in [31a](#).

Uses: [C_get_Covariance 148a](#), [C_get_Expectation 147b](#), [C_get_nresample 151b](#), [C_get_P 145c](#),
[C_get_PermutedLinearStatistic 151c](#), [C_get_Q 146a](#), [C_get_tol 152a](#), [C_get_Variance 147c](#), [C_get_varonly 146b](#),
[C_standardise 65a](#), [LECV 145b](#).

3.4.2 Two-Dimensional Case (“2d”)

< 2d User Interface 42b > ≡

```
< RC_ExpectationCovarianceStatistic_2d 48 >
< R_ExpectationCovarianceStatistic_2d 44 >
< R_PermutedLinearStatistic_2d 51 >
◇
```

Fragment referenced in [24a](#).

< 2d User Interface Inputs 42c > ≡

```
< R x Input 24d >
SEXP ix,
< R y Input 25d >
SEXP iy,
< R weights Input 26c >,
< R subset Input 27b >,
< R block Input 28b >,
◇
```

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

< R_ExpectationCovarianceStatistic_2d Prototype 43a > ≡

```
SEXP R_ExpectationCovarianceStatistic_2d
(
< 2d User Interface Inputs 42c >
SEXP varonly,
SEXP tol
)
◇
```

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

Uses: [R_ExpectationCovarianceStatistic_2d 44](#).

"libcoinAPI.h" [43b](#)≡

```
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 [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [53b](#), [139b](#).

Uses: [block 28bd](#), [R_ExpectationCovarianceStatistic_2d 44](#), [subset 27be](#), [28a](#), [weights 26c](#), [weights, 26de](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

$\langle R_ExpectationCovarianceStatistic_2d\ 44 \rangle \equiv$

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

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

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

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

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

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

Fragment referenced in 42b.

Defines: `R_ExpectationCovarianceStatistic_2d` 8, 43ab, 159, 160.

Uses: `B` 28c, `block` 28bd, `C_get_Table` 150b, `N` 24bc, `Nsubset` 27c, `Offset0` 22b, `P` 25a, `Q` 25e, `RC_init_LECV_2d` 156, `RC_ThreeTableSums` 127b, `RC_TwoTableSums` 122b, `subset` 27be, 28a, `weights` 26c, `weights`, 26de, `x` 24d, 25bc, `y` 25d, 26ab.

⟨ *Setup Dimensions 2d 45a* ⟩ ≡

```
int P, Q, B, Lx, Ly;

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

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

Lx = NLEVELS(ix);
Ly = NLEVELS(iy);
◇
```

Fragment referenced in 44, 51.

Uses: B 28c, block 28bd, NCOL 138c, NLEVELS 139a, P 25a, Q 25e, x 24d, 25bc, y 25d, 26ab.

⟨ *Linear Statistic 2d 45b* ⟩ ≡

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

Fragment referenced in 48, 52d.

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

⟨ 2d Total Table 46a ⟩ ≡

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

Fragment referenced in 48.

Uses: B 28c.

⟨ Col Row Total Sums 46b ⟩ ≡

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

Fragment referenced in 48, 51.

Uses: sumweights 27a.

⟨ 2d Expectation 46c ⟩ ≡

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

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

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

Fragment referenced in 48.

Uses: C_ExpectationLinearStatistic 80b, C_get_Expectation 147b, NROW 138b, Offset0 22b, P 25a, Q 25e,
RC_ExpectationInfluence 84b, RC_ExpectationX 88, subset 27be, 28a, sumweights 27a, x 24d, 25bc, y 25d, 26ab.

⟨ 2d Covariance 47 ⟩ ≡

```
/* 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 < P * (P + 1) / 2; p++) CovX[p] = 0.0;
        for (int p = 0; p < P; p++) CovX[S(p, p, P)] = ExpX[p];
    } else {
        RC_CovarianceX(x, NROW(x), P, Rrsum, subset, Offset0, 0, ExpX, !DoVarOnly, CovX);
    }
    C_CovarianceLinearStatistic(P, Q, C_get_CovarianceInfluence(ans),
        ExpX, CovX, sumweights[b], b,
        C_get_Covariance(ans));
}
◇
```

Fragment referenced in 48.

Uses: C_CovarianceLinearStatistic 81, C_get_Covariance 148a, C_get_CovarianceInfluence 149a, C_get_Variance 147c, C_get_VarianceInfluence 149b, C_get_varonly 146b, C_ordered_Xfactor 71, C_VarianceLinearStatistic 82, DoVarOnly 22b, NROW 138b, Offset0 22b, P 25a, Q 25e, RC_CovarianceInfluence 86b, RC_CovarianceX 91a, S 22a, subset 27be, 28a, sumweights 27a, x 24d, 25bc, y 25d, 26ab.

< RC_ExpectationCovarianceStatistic_2d 48 > ≡

```
void RC_ExpectationCovarianceStatistic_2d
(
  < 2d User Interface Inputs 42c >
  SEXP ans
) {

  < 2d Memory 49 >

  < 2d Total Table 46a >

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

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

    < Linear Statistic 2d 45b >

    < Col Row Total Sums 46b >

    < 2d Expectation 46c >

    < 2d Covariance 47 >

  }

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

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

Fragment referenced in [42b](#).

Defines: [RC_ExpectationCovarianceStatistic 32c](#), [34](#).

Uses: [B 28c](#), [C_get_Covariance 148a](#), [C_get_LinearStatistic 147a](#), [C_get_Variance 147c](#), [C_get_varonly 146b](#), [P 25a](#), [Q 25e](#), [S 22a](#).

⟨ 2d Memory 49 ⟩ ≡

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

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

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

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

if (C_get_varonly(ans)) {
    CovX = Calloc(P, double);
} else {
    CovX = Calloc(P * (P + 1) / 2, double);
}

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

Fragment referenced in 48.

Uses: B 28c, C_get_B 151a, C_get_dimTable 150c, C_get_ExpectationInfluence 148c, C_get_ExpectationX 148b, C_get_P 145c, C_get_Q 146a, C_get_Sumweights 150a, C_get_Table 150b, C_get_varonly 146b, C_get_Xfactor 146c, P 25a, Q 25e, sumweights 27a.

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

      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
[1,] 20.862132 19.435105 20.262426 19.214621 18.715794 19.585989 18.968036
[2,]  6.648524  5.862676  5.730850  5.527873  4.681689  6.842671  5.828585
[3,] 14.087811 13.705985 13.241406 12.608496 12.222386 11.278108 14.104790
[4,] 18.159181 16.898056 17.477228 16.716303 17.053664 17.438078 16.256430
[5,]  6.758675  5.383936  5.604400  4.801078  4.682510  5.525991  4.764740
[6,] 11.295184 13.525396 11.623801 10.729579 11.701565 10.202167 12.239779
[7,] 16.695185 15.869868 16.211875 16.250427 16.652174 15.526386 16.332053
[8,]  5.279886  4.788421  4.325420  5.185671  5.686899  4.685294  4.544657
[9,] 11.291651  9.783289 10.557466 10.754385  9.867325  9.148144 10.839265
[10,] 16.069725 17.485491 16.805419 17.657304 17.653889 17.725369 17.777694
[11,]  4.386114  5.434156  5.823254  5.368845  5.768359  4.717723  5.560541
[12,]  9.171665 10.450831 10.760147 11.086517 11.796552 13.095158  9.583131

      [,8]      [,9]      [,10]
[1,] 19.913398 19.424229 17.906139
[2,]  6.001509  6.354854  5.091668
```

```

[3,] 13.964219 12.253472 11.225575
[4,] 17.783448 17.883613 15.606345
[5,]  5.113195  6.394665  5.132931
[6,] 11.287367 10.019906 10.213060
[7,] 16.587834 15.898009 16.441387
[8,]  5.260119  4.948328  4.971712
[9,] 10.574732 10.380467 10.388247
[10,] 16.728009 17.583227 18.724520
[11,]  5.017115  4.809642  5.883780
[12,] 10.016834 11.542585 12.473410

```

`<R_PermutatedLinearStatistic_2d Prototype 50a> ≡`

```

SEXP R_PermutatedLinearStatistic_2d
(
  <R x Input 24d>
  SEXP ix,
  <R y Input 25d>
  SEXP iy,
  <R block Input 28b>,
  SEXP nresample,
  SEXP itable
)
◇

```

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

Uses: [R_PermutatedLinearStatistic_2d 51](#).

`"libcoinAPI.h" 50b≡`

```

extern SEXP libcoin_R_PermutatedLinearStatistic_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_PermutatedLinearStatistic_2d");
  return fun(x, ix, y, iy, block, nresample, itable);
}
◇

```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [53b](#), [139b](#).

Uses: [block 28bd](#), [R_PermutatedLinearStatistic_2d 51](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

< R_PermutatedLinearStatistic_2d 51 > ≡

< R_PermutatedLinearStatistic_2d Prototype 50a >

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

  < Setup Dimensions 2d 45a >

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

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

  < Setup Working Memory 52b >

  < Convert Table to Integer 52a >

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

  < Setup Log-Factorials 52c >

  GetRNGstate();

  for (R_xlen_t np = 0; np < inresample; np++) {

    < Setup Linear Statistic 41a >

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

    for (int b = 0; b < B; b++) {
      < Compute Permutated Linear Statistic 2d 52d >
    }
  }

  PutRNGstate();

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

Fragment referenced in [42b](#).

Defines: [R_PermutatedLinearStatistic_2d 8](#), [50ab](#), [52a](#), [159](#), [160](#).

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

⟨ *Convert Table to Integer 52a* ⟩ ≡

```
PROTECT(Ritable = allocVector(INTSXP, LENGTH(itable)));
for (int i = 0; i < LENGTH(itable); i++) {
  if (REAL(itable)[i] > INT_MAX)
    error("cannot deal with weights larger INT_MAX in R_PermutatedLinearStatistic_2d");
  INTEGER(Ritable)[i] = (int) REAL(itable)[i];
}
◇
```

Fragment referenced in 51.

Uses: R_PermutatedLinearStatistic_2d 51, weights 26c.

⟨ *Setup Working Memory 52b* ⟩ ≡

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

Fragment referenced in 51.

Uses: B 28c, sumweights 27a.

⟨ *Setup Log-Factorials 52c* ⟩ ≡

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

Fragment referenced in 51.

⟨ *Compute Permuted Linear Statistic 2d 52d* ⟩ ≡

```
S_rcont2(&Lx, &Ly, rsum + Lxp1 * b + 1,
        csum + Lyp1 * b + 1, sumweights + b, fact, jwork, rtable2);

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 45b ⟩
◇
```

Fragment referenced in 51.

Uses: sumweights 27a.

3.5 Tests

Tests 53a ≡

R_QuadraticTest 55
R_MaximumTest 57
R_MaximallySelectedTest 59
◇

Fragment referenced in [24a](#).

"libcoinAPI.h" 53b≡

```
extern SEXP libcoin_R_QuadraticTest(  
  SEXP LEV, SEXP pvalue, SEXP lower, SEXP give_log, SEXP PermutedStatistics  
  
) {  
  static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;  
  if(fun == NULL)  
    fun = (SEXP*)(SEXP, SEXP, SEXP, SEXP, SEXP)  
      R_GetCCallable("libcoin", "R_QuadraticTest");  
  return fun(LEV, pvalue, lower, give_log, PermutedStatistics);  
}  
  
extern SEXP libcoin_R_MaximumTest(  
  SEXP LEV, SEXP alternative, SEXP pvalue, SEXP lower, SEXP give_log,  
  SEXP PermutedStatistics, SEXP maxpts, SEXP releps, SEXP abseps  
  
) {  
  
  static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;  
  if(fun == NULL)  
    fun = (SEXP*)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP, SEXP)  
      R_GetCCallable("libcoin", "R_MaximumTest");  
  return fun(LEV, alternative, pvalue, lower, give_log, PermutedStatistics, maxpts, releps,  
    abseps);  
}  
  
extern SEXP libcoin_R_MaximallySelectedTest(  
  SEXP LEV, SEXP ordered, SEXP teststat, SEXP minbucket, SEXP lower, SEXP give_log  
  
) {  
  
  static SEXP(*fun)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP) = NULL;  
  if(fun == NULL)  
    fun = (SEXP*)(SEXP, SEXP, SEXP, SEXP, SEXP, SEXP)  
      R_GetCCallable("libcoin", "R_MaximallySelectedTest");  
  return fun(LEV, ordered, teststat, minbucket, lower, give_log);  
}  
◇
```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [53b](#), [139b](#).

⟨ R_QuadraticTest Prototype 54 ⟩ ≡

```
SEXP R_QuadraticTest
(
  ⟨ R LECV Input 145b ⟩,
  SEXP pvalue,
  SEXP lower,
  SEXP give_log,
  SEXP PermutedStatistics
)
◇
```

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

< R_QuadraticTest 55 > ≡

< R_QuadraticTest Prototype 54 >

```
{  
  
  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 56a >  
  
  MPinv = Calloc(PQ * (PQ + 1) / 2, double); /* was: C_get_MPinv(LECV); */  
  C_MPinv_sym(C_get_Covariance(LECV), PQ, C_get_tol(LECV), MPinv, &rank);  
  
  REAL(stat)[0] = C_quadform(PQ, C_get_LinearStatistic(LECV),  
                             C_get_Expectation(LECV), MPinv);  
  
  if (!PVALUE) {  
    UNPROTECT(2);  
    Free(MPinv);  
    return(ans);  
  }  
  
  if (C_get_nresample(LECV) == 0) {  
    REAL(pval)[0] = C_chisq_pvalue(REAL(stat)[0], rank, LOWER, GIVELOG);  
  } else {  
    nresample = C_get_nresample(LECV);  
    ls = C_get_PermutatedLinearStatistic(LECV);  
    st = REAL(stat)[0];  
    ex = C_get_Expectation(LECV);  
    greater = 0;  
    for (R_xlen_t np = 0; np < nresample; np++) {  
      pst = C_quadform(PQ, ls + PQ * np, ex, MPinv);  
      if (GE(pst, st, C_get_tol(LECV)))  
        greater++;  
      if (PSTAT) REAL(permstat)[np] = pst;  
    }  
    REAL(pval)[0] = C_perm_pvalue(greater, nresample, LOWER, GIVELOG);  
  }  
  
  UNPROTECT(2);  
  Free(MPinv);  
  return(ans);  
}  
◇
```

Fragment referenced in 53a.

Uses: C_chisq_pvalue 66a, C_get_Covariance 148a, C_get_Expectation 147b, C_get_LinearStatistic 147a,
C_get_nresample 151b, C_get_PermutatedLinearStatistic 151c, C_get_tol 152a, C_perm_pvalue 66b, C_quadform 63b,
GE 22a, LECV 145b, P 25a, Q 25e.

⟨ *Setup Test Memory 56a* ⟩ ≡

```
P = C_get_P(LECV);
Q = C_get_Q(LECV);
PQ = P * Q;

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

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

Uses: [C_get_nresample 151b](#), [C_get_P 145c](#), [C_get_Q 146a](#), [C_get_varonly 146b](#), [LECV 145b](#), [P 25a](#), [Q 25e](#).

⟨ *R_MaximumTest Prototype 56b* ⟩ ≡

```
SEXP R_MaximumTest
(
    ⟨ R LECV Input 145b ⟩,
    SEXP alternative,
    SEXP pvalue,
    SEXP lower,
    SEXP give_log,
    SEXP PermutedStatistics,
    SEXP maxpts,
    SEXP releps,
    SEXP abseps
)
◇
```

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

< R_MaximumTest 57 > ≡

```
< R_MaximumTest Prototype 56b >
{
  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 56a >

  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 53a.

Uses: C_get_Covariance 148a, C_get_Expectation 147b, C_get_LinearStatistic 147a, C_get_nresample 151b,
C_get_PermutedLinearStatistic 151c, C_get_tol 152a, C_get_Variance 147c, C_get_varonly 146b, C_maxtype 64,
C_maxtype_pvalue 68, C_perm_pvalue 66b, GE 22a, LE 22a, LECV 145b, P 25a, Q 25e.

< R_MaximallySelectedTest Prototype 58 > ≡

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

Fragment referenced in [23b](#), [59](#).
Uses: [LECV 145b](#).

< R_MaximallySelectedTest 59 > ≡

< R_MaximallySelectedTest Prototype 58 >

```
{  
  
  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 [53a](#).

Uses: [C_get_nresample 151b](#), [C_get_P 145c](#), [C_ordered_Xfactor 71](#), [C_unordered_Xfactor 76](#), [LECV 145b](#), [P 25a](#).

3.6 Test Statistics

⟨ *Test Statistics 60a* ⟩ ≡

⟨ *C_maxstand_Covariance 60b* ⟩
⟨ *C_maxstand_Variance 61a* ⟩
⟨ *C_minstand_Covariance 61b* ⟩
⟨ *C_minstand_Variance 62a* ⟩
⟨ *C_maxabsstand_Covariance 62b* ⟩
⟨ *C_maxabsstand_Variance 63a* ⟩
⟨ *C_quadform 63b* ⟩
⟨ *C_maxtype 64* ⟩
⟨ *C_standardise 65a* ⟩
⟨ *C_ordered_Xfactor 71* ⟩
⟨ *C_unordered_Xfactor 76* ⟩
◇

Fragment referenced in [24a](#).

⟨ *C_maxstand_Covariance 60b* ⟩ ≡

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

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

Fragment referenced in [60a](#).

Defines: [C_maxstand_Covariance 64](#).

Uses: [S 22a](#).

$\langle C_maxstand_Variance\ 61a \rangle \equiv$

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

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

Fragment referenced in [60a](#).

Defines: [C_maxstand_Variance 64](#).

$\langle C_minstand_Covariance\ 61b \rangle \equiv$

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

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

Fragment referenced in [60a](#).

Defines: [C_minstand_Covariance 64](#).

Uses: [S 22a](#).

< C_minstand_Variance 62a > ≡

```
double C_minstand_Variance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *var,
    const double tol
) {

    double ans = R_PosInf, tmp = 0.0;

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

Fragment referenced in [60a](#).

Defines: [C_minstand_Variance 64](#).

< C_maxabsstand_Covariance 62b > ≡

```
double C_maxabsstand_Covariance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *covar_sym,
    const double tol
) {

    double ans = R_NegInf, tmp = 0.0;

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

Fragment referenced in [60a](#).

Defines: [C_maxabsstand_Covariance 64](#).

Uses: [S 22a](#).

$\langle C_maxabsstand_Variance\ 63a \rangle \equiv$

```
double C_maxabsstand_Variance
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *var,
    const double tol
) {

    double ans = R_NegInf, tmp = 0.0;

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

Fragment referenced in [60a](#).

Defines: `C_maxabsstand_Variance` [64](#).

$\langle C_quadform\ 63b \rangle \equiv$

```
double C_quadform
(
    const int PQ,
    const double *linstat,
    const double *expect,
    const double *MPinv_sym
) {

    double ans = 0.0, tmp = 0.0;

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

    return(ans);
}
◇
```

Fragment referenced in [60a](#).

Defines: `C_quadform` [55](#), [74c](#).

Uses: [S 22a](#).

$\langle C_maxtype\ 64 \rangle \equiv$

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

    double ret = 0.0;

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

Fragment referenced in 60a.

Defines: C_maxtype 57, 74c.

Uses: C_maxabsstand_Covariance 62b, C_maxabsstand_Variance 63a, C_maxstand_Covariance 60b, C_maxstand_Variance 61a, C_minstand_Covariance 61b, C_minstand_Variance 62a.

< C_standardise 65a > ≡

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

    double var;

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

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

< P-Values 65b > ≡

```
< C_chisq_pvalue 66a >
< C_perm_pvalue 66b >
< C_norm_pvalue 67 >
< C_maxtype_pvalue 68 >
◇
```

Fragment referenced in [24a](#).

$\langle C_chisq_pvalue\ 66a \rangle \equiv$

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

Fragment referenced in [65b](#).
Defines: `C_chisq_pvalue` [55](#).

$\langle C_perm_pvalue\ 66b \rangle \equiv$

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

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

Fragment referenced in [65b](#).
Defines: `C_perm_pvalue` [55](#), [57](#), [75](#).

`< C_norm_pvalue 67 > ≡`

```
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 [65b](#).

< C_maxtype_pvalue 68 > ≡

```
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 69 >

    < Setup mvtnorm Correlation 70a >

    /* 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 MVT DST");
                ans = 0.0;
    }
    Free(corr); Free(sd); Free(lowerbnd); Free(upperbnd);
    Free(infin); Free(delta); Free(index);

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

Fragment referenced in [65b](#).
Defines: `C_maxtype_pvalue` [57](#).
Uses: `N` [24bc](#).

⟨ *Setup mvtnorm Memory* 69 ⟩ ≡

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

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

/* determine elements with non-zero variance */

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

Fragment referenced in [68](#).

Uses: [S 22a](#).

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

⟨ *Setup mvtnorm Correlation 70a* ⟩ ≡

```
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 [68](#).
Uses: [S 22a](#).

⟨ *maxstat Xfactor Variables 70b* ⟩ ≡

```
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 [71](#), [76](#).
Uses: [LECV 145b](#).

< C_ordered_Xfactor 71 > ≡

```
void C_ordered_Xfactor
(
  < maxstat Xfactor Variables 70b >
) {

  < Setup maxstat Variables 72 >

  < Setup maxstat Memory 73 >

  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 74b >
      } else {
        < Compute maxstat Variance / Covariance from Total Covariance 74a >
      }
    }

    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 74c >
      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 74c >
        if (tmp > bmaxstat[np])
          bmaxstat[np] = tmp;
      }
    }
  }
  < Compute maxstat Permutation P-Value 75 >
  Free(mlinstat); Free(mexpect); Free(mblinstat);
  Free(mvar); Free(mcovar); Free(mMPinv);
  if (nresample == 0) Free(blinstat);
}
◇
```

Fragment referenced in 60a.

Defines: C_ordered_Xfactor 37b, 47, 59.

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

⟨ Setup maxstat Variables 72 ⟩ ≡

```
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 = P * Q;
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 71, 76.

Uses: B 28c, C_get_B 151a, C_get_Covariance 148a, C_get_CovarianceInfluence 149a, C_get_Expectation 147b, C_get_ExpectationX 148b, C_get_LinearStatistic 147a, C_get_nresample 151b, C_get_P 145c, C_get_PermutedLinearStatistic 151c, C_get_Q 146a, C_get_tol 152a, C_get_Variance 147c, C_get_VarianceInfluence 149b, C_get_varonly 146b, LECV 145b, P 25a, Q 25e, sumweights 27a.

⟨ Setup maxstat Memory 73 ⟩ ≡

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

maxstat[0] = 0.0;

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

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

Fragment referenced in [71](#), [76](#).
Uses: [P 25a](#), [Q 25e](#), [sumweights 27a](#).

< Compute maxstat Variance / Covariance from Total Covariance 74a > ≡

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

Fragment referenced in 71.

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

< Compute maxstat Variance / Covariance Directly 74b > ≡

```
/* 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 71.

Uses: C_CovarianceLinearStatistic 81, C_VarianceLinearStatistic 82, Q 25e, sumweights 27a.

< Compute maxstat Test Statistic 74c > ≡

```
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 71, 76.

Uses: C_maxtype 64, C_quadform 63b, Q 25e.

⟨ *Compute maxstat Permutation P-Value 75* ⟩ ≡

```
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 [71](#), [76](#).

Uses: [C_perm_pvalue 66b](#).

< C_unordered_Xfactor 76 > ≡

```
void C_unordered_Xfactor
(
  < maxstat Xfactor Variables 70b >
) {

  double *mtmp;
  int qPp, nc, *levels, Pnonzero, *indl, *contrast;

  < Setup maxstat Variables 72 >

  < Setup maxstat Memory 73 >
  mtmp = Calloc(P, double);

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

  < Count Levels 77a >

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

    < Setup unordered maxstat Contrasts 77b >

    < Compute unordered maxstat Linear Statistic and Expectation 78a >

    if (B == 1) {
      < Compute unordered maxstat Variance / Covariance Directly 79a >
    } else {
      < Compute unordered maxstat Variance / Covariance from Total Covariance 78b >
    }

    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 74c >
      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 74c >
        if (tmp > bmaxstat[np])
          bmaxstat[np] = tmp;
      }
    }
  }

  < Compute maxstat Permutation P-Value 75 >

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

Fragment referenced in 60a.

Defines: C_unordered_Xfactor 37b, 59.

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

< Count Levels 77a > ≡

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

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

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

Fragment referenced in [76](#).

Uses: [P 25a](#).

< Setup unordered maxstat Contrasts 77b > ≡

```
/* 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 [76](#).

Uses: [P 25a](#).

< Compute unordered maxstat Linear Statistic and Expectation 78a > ≡

```
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 76.

Uses: P 25a, Q 25e.

< Compute unordered maxstat Variance / Covariance from Total Covariance 78b > ≡

```
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, P * Q)];
      }
      for (int p = 0; p < P; p++)
        mcovar[S(q, qq, Q)] += contrast[p] * mtmp[p];
    }
  }
}
◇
```

Fragment referenced in 76.

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

< Compute unordered maxstat Variance / Covariance Directly 79a > ≡

```
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 76.

Uses: C_CovarianceLinearStatistic 81, C_VarianceLinearStatistic 82, Q 25e, sumweights 27a.

3.7 Linear Statistics

< LinearStatistics 79b > ≡

```
< RC_LinearStatistic 79d >
◇
```

Fragment referenced in 24a.

< RC_LinearStatistic Prototype 79c > ≡

```
void RC_LinearStatistic
(
    < R x Input 24d >
    < C integer N Input 24c >,
    < C integer P Input 25a >,
    < C real y Input 26a >
    < R weights Input 26c >,
    < R subset Input 27b >,
    < C subset range Input 27d >,
    < C KronSums Answer 99d >
)
◇
```

Fragment referenced in 79d.

Uses: RC_LinearStatistic 79d.

< RC_LinearStatistic 79d > ≡

```
< RC_LinearStatistic Prototype 79c >
{
    double center;

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

Fragment referenced in 79b.

Defines: RC_LinearStatistic 35b, 79c.

Uses: DoCenter 22b, DoSymmetric 22b, N 24bc, Nsubset 27c, offset 27d, P 25a, Q 25e, RC_KronSums 99a, subset 27bc, 28a, weights 26c, weights, 26de, x 24d, 25bc, y 25d, 26ab.

3.8 Expectation and Covariance

< ExpectationCovariances 80a > ≡

< RC_ExpectationInfluence 84b >
< R_ExpectationInfluence 83b >
< RC_CovarianceInfluence 86b >
< R_CovarianceInfluence 85b >
< RC_ExpectationX 88 >
< R_ExpectationX 87a >
< RC_CovarianceX 91a >
< R_CovarianceX 90a >
< C_ExpectationLinearStatistic 80b >
< C_CovarianceLinearStatistic 81 >
< C_VarianceLinearStatistic 82 >
◇

Fragment referenced in [24a](#).

3.8.1 Linear Statistic

< C_ExpectationLinearStatistic 80b > ≡

```
void C_ExpectationLinearStatistic
(
    < C integer P Input 25a >,
    < C integer Q Input 25e >,
    double *ExpInf,
    double *ExpX,
    const int add,
    double *PQ_ans
) {
    if (!add)
        for (int p = 0; p < P * Q; 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];
    }
}
◇
```

Fragment referenced in [80a](#).

Defines: `C_ExpectationLinearStatistic` [37a](#), [46c](#).

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

< C_CovarianceLinearStatistic 81 > ≡

```
void C_CovarianceLinearStatistic
(
  < C integer P Input 25a >,
  < C integer Q Input 25e >,
  double *CovInf,
  double *ExpX,
  double *CovX,
  < C sumweights Input 27a >,
  const int add,
  double *PQPQ_sym_ans
) {

  double f1 = sumweights / (sumweights - 1);
  double f2 = 1.0 / (sumweights - 1);
  double tmp, *PP_sym_tmp;

  if (P * Q == 1) {
    tmp = f1 * CovInf[0] * CovX[0];
    tmp -= f2 * CovInf[0] * ExpX[0] * ExpX[0];
    if (add) {
      PQPQ_sym_ans[0] += tmp;
    } else {
      PQPQ_sym_ans[0] = tmp;
    }
  } else {
    PP_sym_tmp = Calloc(P * (P + 1) / 2, double);
    C_KronSums_sym(ExpX, 1, P,
                  PP_sym_tmp);
    for (int p = 0; p < P * (P + 1) / 2; p++)
      PP_sym_tmp[p] = f1 * CovX[p] - f2 * PP_sym_tmp[p];
    C_kronecker_sym(CovInf, Q, PP_sym_tmp, P, 1 - (add >= 1),
                  PQPQ_sym_ans);
    Free(PP_sym_tmp);
  }
}
◇
```

Fragment referenced in [80a](#).

Defines: [C_CovarianceLinearStatistic 38a](#), [47](#), [74b](#), [79a](#), [82](#).

Uses: [C_kronecker_sym 142](#), [P 25a](#), [Q 25e](#), [sumweights 27a](#).

< C_VarianceLinearStatistic 82 > ≡

```
void C_VarianceLinearStatistic
(
  < C integer P Input 25a >,
  < C integer Q Input 25e >,
  double *VarInf,
  double *ExpX,
  double *VarX,
  < C sumweights Input 27a >,
  const int add,
  double *PQ_ans
) {

  if (P * Q == 1) {
    C_CovarianceLinearStatistic(P, Q, VarInf, ExpX, VarX,
                               sumweights, (add >= 1),
                               PQ_ans);
  } else {
    double *P_tmp;
    P_tmp = Calloc(P, double);
    double f1 = sumweights / (sumweights - 1);
    double f2 = 1.0 / (sumweights - 1);
    for (int p = 0; p < P; p++)
      P_tmp[p] = f1 * VarX[p] - f2 * ExpX[p] * ExpX[p];
    C_kronecker(VarInf, 1, Q, P_tmp, 1, P, 1 - (add >= 1),
               PQ_ans);
    Free(P_tmp);
  }
}
◇
```

Fragment referenced in 80a.

Defines: C_VarianceLinearStatistic 37c, 47, 74b, 79a.

Uses: C_CovarianceLinearStatistic 81, C_kronecker 141, P 25a, Q 25e, sumweights 27a.

3.8.2 Influence

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

< R_ExpectationInfluence Prototype 83a > ≡

```
SEXP R_ExpectationInfluence
(
  < R y Input 25d >
  < R weights Input 26c >,
  < R subset Input 27b >
)
◇
```

Fragment referenced in 23b, 83b.

Uses: R_ExpectationInfluence 83b.

< R_ExpectationInfluence 83b > ≡

```
< R_ExpectationInfluence Prototype 83a >
{
  SEXP ans;
  < C integer Q Input 25e >;
  < C integer N Input 24c >;
  < C integer Nsubset Input 27c >;
  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 80a.

Defines: R_ExpectationInfluence 83a, 85b, 159, 160.

Uses: N 24bc, NCOL 138c, Nsubset 27c, Offset0 22b, Q 25e, RC_ExpectationInfluence 84b, RC_Sums 94a, subset 27be, 28a, sumweights 27a, weights 26c, weights, 26de, y 25d, 26ab.

< RC_ExpectationInfluence Prototype 84a > ≡

```
void RC_ExpectationInfluence
(
  < C integer N Input 24c >,
  < R y Input 25d >
  < C integer Q Input 25e >,
  < R weights Input 26c >,
  < R subset Input 27b >,
  < C subset range Input 27d >,
  < C sumweights Input 27a >,
  < C colSums Answer 113c >
)
◇
```

Fragment referenced in 84b.

Uses: RC_ExpectationInfluence 84b.

< RC_ExpectationInfluence 84b > ≡

```
< RC_ExpectationInfluence Prototype 84a >
{
  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 80a.

Defines: *RC_ExpectationInfluence* 37a, 46c, 83b, 84a.

Uses: *DoCenter* 22b, *N* 24bc, *Nsubset* 27c, *offset* 27d, *Power1* 22b, *Q* 25e, *RC_colSums* 113a, *subset* 27be, 28a, *sumweights* 27a, *weights* 26c, *weights*, 26de, *y* 25d, 26ab.

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

< R_CovarianceInfluence Prototype 85a > ≡

```
SEXP R_CovarianceInfluence
(
  < R y Input 25d >
  < R weights Input 26c >,
  < R subset Input 27b >,
  SEXP varonly
)
◇
```

Fragment referenced in 23b, 85b.

Uses: *R_CovarianceInfluence* 85b.

< R_CovarianceInfluence 85b > ≡

```
< R_CovarianceInfluence Prototype 85a >
{
  SEXP ans;
  SEXP ExpInf;
  < C integer Q Input 25e >;
  < C integer N Input 24c >;
  < C integer Nsubset Input 27c >;
  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 [80a](#).

Defines: [R_CovarianceInfluence 85a](#), [159](#), [160](#).

Uses: [N 24bc](#), [NCOL 138c](#), [Nsubset 27c](#), [Offset0 22b](#), [Q 25e](#), [RC_CovarianceInfluence 86b](#), [RC_Sums 94a](#),
[R_ExpectationInfluence 83b](#), [subset 27be, 28a](#), [sumweights 27a](#), [weights 26c](#), [weights, 26de, y 25d, 26ab](#).

< RC_CovarianceInfluence Prototype 86a > ≡

```
void RC_CovarianceInfluence
(
  < C integer N Input 24c >,
  < R y Input 25d >
  < C integer Q Input 25e >,
  < R weights Input 26c >,
  < R subset Input 27b >,
  < C subset range Input 27d >,
  double *ExpInf,
  < C sumweights Input 27a >,
  int VARONLY,
  < C KronSums Answer 99d >
)
◇
```

Fragment referenced in [86b](#).

Uses: [RC_CovarianceInfluence 86b](#).

< RC_CovarianceInfluence 86b > ≡

```
< RC_CovarianceInfluence Prototype 86a >
{
  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 [80a](#).

Defines: [RC_CovarianceInfluence 37b, 47, 85b, 86a](#).

Uses: [DoCenter 22b](#), [DoSymmetric 22b](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#), [Power2 22b](#), [Q 25e](#), [RC_colSums 113a](#),
[RC_KronSums 99a](#), [subset 27be, 28a](#), [sumweights 27a](#), [weights 26c](#), [weights, 26de](#), [y 25d, 26ab](#).

3.8.3 X

< R_ExpectationX Prototype 86c > ≡

```
SEXP R_ExpectationX
(
  < R x Input 24d >
  SEXP P,
  < R weights Input 26c >,
  < R subset Input 27b >
)
◇
```

Fragment referenced in [23b, 87a](#).

Uses: [P 25a](#), [R_ExpectationX 87a](#).

< R_ExpectationX 87a > ≡

```
< R_ExpectationX Prototype 86c >
{
  SEXP ans;
  < C integer N Input 24c >;
  < C integer Nsubset Input 27c >;

  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 80a.

Defines: *R_ExpectationX* 86c, 90a, 159, 160.

Uses: *N* 24bc, *Nsubset* 27c, *Offset0* 22b, *P* 25a, *RC_ExpectationX* 88, *subset* 27be, 28a, *weights* 26c, *weights*, 26de, *x* 24d, 25bc.

< RC_ExpectationX Prototype 87b > ≡

```
void RC_ExpectationX
(
  < R x Input 24d >
  < C integer N Input 24c >,
  < C integer P Input 25a >,
  < R weights Input 26c >,
  < R subset Input 27b >,
  < C subset range Input 27d >,
  < C OneTableSums Answer 118c >
)
◇
```

Fragment referenced in 88.

Uses: *RC_ExpectationX* 88.

$\langle RC_ExpectationX\ 88 \rangle \equiv$

```
 $\langle RC\_ExpectationX\ Prototype\ 87b \rangle$ 
{
  double center;

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

Fragment referenced in 80a.

Defines: RC_ExpectationX 37a, 46c, 87ab.

Uses: DoCenter 22b, N 24bc, Nsubset 27c, offset 27d, P 25a, Power1 22b, RC_colSums 113a, RC_OneTableSums 118a, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc.

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

```
[1] 41.61233 12.61379 26.76585
```

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

```
[1] 12 9 2 0 0 0 7 23 9 0
```

```
> 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 89> ≡

```

SEXP R_CovarianceX
(
  <R x Input 24d>
  SEXP P,
  <R weights Input 26c>,
  <R subset Input 27b>,
  SEXP varonly
)
◇

```

Fragment referenced in [23b](#), [90a](#).

Uses: [P 25a](#), [R_CovarianceX 90a](#).

$\langle R_CovarianceX\ 90a \rangle \equiv$

```
 $\langle R\_CovarianceX\ Prototype\ 89 \rangle$ 
{
  SEXP ans;
  SEXP ExpX;
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
   $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;

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

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

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

Fragment referenced in 80a.

Defines: R_CovarianceX 89, 159, 160.

Uses: N 24bc, Nsubset 27c, Offset0 22b, P 25a, RC_CovarianceX 91a, R_ExpectationX 87a, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc.

$\langle RC_CovarianceX\ Prototype\ 90b \rangle \equiv$

```
void RC_CovarianceX
(
   $\langle R\ x\ Input\ 24d \rangle$ 
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle C\ integer\ P\ Input\ 25a \rangle$ ,
   $\langle R\ weights\ Input\ 26c \rangle$ ,
   $\langle R\ subset\ Input\ 27b \rangle$ ,
   $\langle C\ subset\ range\ Input\ 27d \rangle$ ,
  double *ExpX,
  int VARONLY,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
)
◇
```

Fragment referenced in 91a.

Uses: RC_CovarianceX 91a.

$\langle RC_CovarianceX\ 91a \rangle \equiv$

```
 $\langle RC\_CovarianceX\ Prototype\ 90b \rangle$ 
{
    double center;

    if (typeof(x) == INTSXP) {
        if (VARONLY) {
            for (int p = 0; p < P; p++) PQ_ans[p] = ExpX[p];
        } else {
            for (int p = 0; p < P * (P + 1) / 2; 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 80a.

Defines: RC_CovarianceX 37c, 38a, 47, 90ab.

Uses: DoCenter 22b, DoSymmetric 22b, N 24bc, Nsubset 27c, offset 27d, P 25a, Power2 22b, RC_colSums 113a, RC_KronSums 99a, S 22a, subset 27be, 28a, weights 26c, weights, 26de, x 24d, 25bc.

3.9 Computing Sums

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

$\langle init\ subset\ loop\ 91b \rangle \equiv$

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

Fragment referenced in 96a, 103, 106, 115b, 120b, 125, 130a.

Uses: N 24bc, Nsubset 27c, offset 27d, subset 27be, 28a, weights 26c.

and loop over $i = 1, \dots, N$ when no subset was specified or over the subset of the subset given by `offset` and `Nsubset`, allowing for number of observations larger than `INT_MAX`

< start subset loop 92a > ≡

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

Fragment referenced in [96a](#), [103](#), [106](#), [115b](#), [120b](#), [125](#), [130a](#).
Uses: [N 24bc](#), [Nsubset 27c](#).

After computations in the loop, we compute the next element

< continue subset loop 92b > ≡

```
    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 [96a](#), [103](#), [106](#), [115b](#), [120b](#), [125](#), [130a](#).
Uses: [Nsubset 27c](#), [subset 27be](#), [28a](#).

3.9.1 Simple Sums

< SimpleSums 92c > ≡

```
    < C_Sums_dweights_dsubset 94b >  
    < C_Sums_iweights_dsubset 95a >  
    < C_Sums_iweights_isset 95b >  
    < C_Sums_dweights_isset 95c >  
    < RC_Sums 94a >  
    < R_Sums 93b >  
    ◇
```

Fragment referenced in [24a](#).

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

< R_Sums Prototype 93a > ≡

```
SEXP R_Sums
(
  < R N Input 24b >
  < R weights Input 26c >,
  < R subset Input 27b >
)
◇
```

Fragment referenced in 23b, 93b.
Uses: R_Sums 93b.

< R_Sums 93b > ≡

```
< R_Sums Prototype 93a >
{
  SEXP ans;
  < C integer Nsubset Input 27c >;

  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 92c.
Defines: R_Sums 93a, 159, 160.
Uses: N 24bc, Nsubset 27c, Offset0 22b, RC_Sums 94a, subset 27be, 28a, weights 26c, weights, 26de.

< RC_Sums Prototype 93c > ≡

```
double RC_Sums
(
  < C integer N Input 24c >,
  < R weights Input 26c >,
  < R subset Input 27b >,
  < C subset range Input 27d >
)
◇
```

Fragment referenced in 94a.
Uses: RC_Sums 94a.

< RC_Sums 94a > ≡

```
< RC_Sums Prototype 93c >
{
  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_iveights_isubset(N, INTEGER(weights), XLENGTH(weights),
                                     INTEGER(subset), offset, Nsubset));
    } else {
      return(C_Sums_iveights_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 [92c](#).

Defines: [RC_Sums 36ab](#), [83b](#), [85b](#), [93bc](#), [131b](#), [135a](#).

Uses: [C_Sums_dweights_dsubset 94b](#), [C_Sums_dweights_isubset 95c](#), [C_Sums_iveights_dsubset 95a](#),
[C_Sums_iveights_isubset 95b](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#), [subset 27be](#), [28a](#), [weights 26c](#).

< C_Sums_dweights_dsubset 94b > ≡

```
double C_Sums_dweights_dsubset
(
  < C integer N Input 24c >,
  < C real weights Input 26e >
  < C real subset Input 28a >
)
{
  double *s, *w;
  < Sums Body 96a >
}
◇
```

Fragment referenced in [92c](#).

Defines: [C_Sums_dweights_dsubset 94a](#).

$\langle C_Sums_iweights_dsubset\ 95a \rangle \equiv$

```
double C_Sums_iweights_dsubset
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ real\ subset\ Input\ 28a \rangle$ 
)
{
  double *s;
  int *w;
   $\langle Sums\ Body\ 96a \rangle$ 
}
◇
```

Fragment referenced in 92c.

Defines: C_Sums_iweights_dsubset 94a.

$\langle C_Sums_iweights_isubset\ 95b \rangle \equiv$

```
double C_Sums_iweights_isubset
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ 
)
{
  int *s, *w;
   $\langle Sums\ Body\ 96a \rangle$ 
}
◇
```

Fragment referenced in 92c.

Defines: C_Sums_iweights_isubset 94a.

$\langle C_Sums_dweights_isubset\ 95c \rangle \equiv$

```
double C_Sums_dweights_isubset
(
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ,
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ 
)
{
  int *s;
  double *w;
   $\langle Sums\ Body\ 96a \rangle$ 
}
◇
```

Fragment referenced in 92c.

Defines: C_Sums_dweights_isubset 94a.

< Sums Body 96a > ≡

```
double ans = 0.0;

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

< init subset loop 91b >
< start subset loop 92a >
{
  w = w + diff;
  ans += w[0];
  < continue subset loop 92b >
}

w = w + diff;
ans += w[0];

return(ans);
```

◇

Fragment referenced in [94b](#), [95abc](#).

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

3.9.2 Kronecker Sums

< KronSums 96b > ≡

```
< C_KronSums_dweights_dsubset 101b >
< C_KronSums_iweights_dsubset 102a >
< C_KronSums_iweights_isubset 102b >
< C_KronSums_dweights_isubset 102c >
< C_XfactorKronSums_dweights_dsubset 104b >
< C_XfactorKronSums_iweights_dsubset 104c >
< C_XfactorKronSums_iweights_isubset 105a >
< C_XfactorKronSums_dweights_isubset 105b >
< RC_KronSums 99a >
< R_KronSums 98a >
< C_KronSums_Permutation_isubset 109a >
< C_KronSums_Permutation_dsubset 108b >
< C_XfactorKronSums_Permutation_isubset 110b >
< C_XfactorKronSums_Permutation_dsubset 110a >
< RC_KronSums_Permutation 108a >
< R_KronSums_Permutation 107b >
```

◇

Fragment referenced in [24a](#).

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

```

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

```

< R_KronSums Prototype 97 > ≡

```

SEXP R_KronSums
(
  < R x Input 24d >
  SEXP P,
  < R y Input 25d >
  < R weights Input 26c >,
  < R subset Input 27b >,
  SEXP symmetric
)
◇

```

Fragment referenced in [23b](#), [98a](#).

Uses: P [25a](#), R_KronSums [98a](#).

$\langle R_KronSums\ 98a \rangle \equiv$

```
 $\langle R\_KronSums\ Prototype\ 97 \rangle$ 
{
  SEXP ans;
   $\langle C\ integer\ Q\ Input\ 25e \rangle$ ;
   $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
   $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;

  double center;

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

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

Fragment referenced in 96b.

Defines: $R_KronSums\ 97, 159, 160$.

Uses: $DoCenter\ 22b, N\ 24bc, NCOL\ 138c, Nsubset\ 27c, Offset0\ 22b, P\ 25a, Q\ 25e, RC_KronSums\ 99a, subset\ 27be, 28a,$
 $weights\ 26c, weights,\ 26de, x\ 24d, 25bc, y\ 25d, 26ab.$

$\langle RC_KronSums\ Prototype\ 98b \rangle \equiv$

```
void RC_KronSums
(
   $\langle RC\ KronSums\ Input\ 99b \rangle$ 
   $\langle R\ weights\ Input\ 26c \rangle$ ,
   $\langle R\ subset\ Input\ 27b \rangle$ ,
   $\langle C\ subset\ range\ Input\ 27d \rangle$ ,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
)
◇
```

Fragment referenced in 99a.

Uses: $RC_KronSums\ 99a.$

$\langle RC_KronSums\ 99a \rangle \equiv$

```
 $\langle RC\_KronSums\ Prototype\ 98b \rangle$ 
{
    if (TYPEOF(x) == INTSXP) {
         $\langle KronSums\ Integer\ x\ 100 \rangle$ 
    } else {
         $\langle KronSums\ Double\ x\ 101a \rangle$ 
    }
}
◇
```

Fragment referenced in [96b](#).

Defines: [RC_KronSums 79d](#), [86b](#), [91a](#), [98ab](#).

Uses: [x 24d](#), [25bc](#).

$\langle RC\ KronSums\ Input\ 99b \rangle \equiv$

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

Fragment referenced in [98b](#).

$\langle C\ KronSums\ Input\ 99c \rangle \equiv$

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

Fragment referenced in [101b](#), [102abc](#).

$\langle C\ KronSums\ Answer\ 99d \rangle \equiv$

```
double *PQ_ans
◇
```

Fragment referenced in [79c](#), [86a](#), [90b](#), [98b](#), [101b](#), [102abc](#), [104bc](#), [105ab](#), [107c](#), [108b](#), [109a](#), [110ab](#).

$\langle \text{KronSums Integer } x \text{ 100} \rangle \equiv$

```
if (SYMMETRIC) error("not implemented");
if (CENTER) error("not implemented");
if (TYPEOF(weights) == INTSXP) {
  if (TYPEOF(subset) == INTSXP) {
    C_XfactorKronSums_iveights_isubset(INTEGER(x), N, P, y, Q,
      INTEGER(weights), XLENGTH(weights) > 0, INTEGER(subset),
      offset, Nsubset, PQ_ans);
  } else {
    C_XfactorKronSums_iveights_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 99a.

Uses: C_XfactorKronSums_dweights_dsubset 104b, C_XfactorKronSums_dweights_isubset 105b,
C_XfactorKronSums_iveights_dsubset 104c, C_XfactorKronSums_iveights_isubset 105a, N 24bc, Nsubset 27c,
offset 27d, P 25a, Q 25e, subset 27be, 28a, weights 26c, x 24d, 25bc, y 25d, 26ab.

$\langle \text{KronSums Double } x \text{ 101a} \rangle \equiv$

```
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 99a.

Uses: C_KronSums_dweights_dsubset 101b, C_KronSums_dweights_isubset 102c, C_KronSums_iweights_dsubset 102a,
C_KronSums_iweights_isubset 102b, N 24bc, Nsubset 27c, offset 27d, P 25a, Q 25e, subset 27be, 28a, weights 26c,
x 24d, 25bc, y 25d, 26ab.

$\langle \text{C_KronSums_dweights_dsubset 101b} \rangle \equiv$

```
void C_KronSums_dweights_dsubset
(
   $\langle \text{C KronSums Input 99c} \rangle$ 
   $\langle \text{C real weights Input 26e} \rangle$ 
   $\langle \text{C real subset Input 28a} \rangle$ ,
   $\langle \text{C KronSums Answer 99d} \rangle$ 
)
{
  double *s, *w;
   $\langle \text{KronSums Body 103} \rangle$ 
}
}
```

Fragment referenced in 96b.

Defines: C_KronSums_dweights_dsubset 101a.

$\langle C_KronSums_iweights_dsubset\ 102a \rangle \equiv$

```
void C_KronSums_iweights_dsubset
(
   $\langle C\ KronSums\ Input\ 99c \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
)
{
  double *s;
  int *w;
   $\langle KronSums\ Body\ 103 \rangle$ 
}
◇
```

Fragment referenced in [96b](#).

Defines: `C_KronSums_iweights_dsubset` [101a](#).

$\langle C_KronSums_iweights_isubset\ 102b \rangle \equiv$

```
void C_KronSums_iweights_isubset
(
   $\langle C\ KronSums\ Input\ 99c \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
)
{
  int *s, *w;
   $\langle KronSums\ Body\ 103 \rangle$ 
}
◇
```

Fragment referenced in [96b](#).

Defines: `C_KronSums_iweights_isubset` [101a](#).

$\langle C_KronSums_dweights_isubset\ 102c \rangle \equiv$

```
void C_KronSums_dweights_isubset
(
   $\langle C\ KronSums\ Input\ 99c \rangle$ 
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
) {
  int *s;
  double *w;
   $\langle KronSums\ Body\ 103 \rangle$ 
}
◇
```

Fragment referenced in [96b](#).

Defines: `C_KronSums_dweights_isubset` [101a](#).

< KronSums Body 103 > ≡

```
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 91b >
    < start subset loop 92a >
    {
      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 92b >
    }
    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 [101b](#), [102abc](#).

Uses: HAS_WEIGHTS [26de](#), N [24bc](#), P [25a](#), Q [25e](#), S [22a](#), x [24d](#), [25bc](#), y [25d](#), [26ab](#).

Xfactor Kronecker Sums

$\langle C \text{ XfactorKronSums Input 104a} \rangle \equiv$

$\langle C \text{ integer } x \text{ Input 25c} \rangle$
 $\langle C \text{ real } y \text{ Input 26a} \rangle$

◇

Fragment referenced in [104bc](#), [105ab](#).

$\langle C_XfactorKronSums_dweights_dsubset \text{ 104b} \rangle \equiv$

```
void C_XfactorKronSums_dweights_dsubset
(
   $\langle C \text{ XfactorKronSums Input 104a} \rangle$ 
   $\langle C \text{ real weights Input 26e} \rangle$ 
   $\langle C \text{ real subset Input 28a} \rangle$ ,
   $\langle C \text{ KronSums Answer 99d} \rangle$ 
)
{
  double *s, *w;
   $\langle XfactorKronSums Body 106 \rangle$ 
}
```

◇

Fragment referenced in [96b](#).

Defines: `C_XfactorKronSums_dweights_dsubset 100`.

$\langle C_XfactorKronSums_iweights_dsubset \text{ 104c} \rangle \equiv$

```
void C_XfactorKronSums_iweights_dsubset
(
   $\langle C \text{ XfactorKronSums Input 104a} \rangle$ 
   $\langle C \text{ integer weights Input 26d} \rangle$ 
   $\langle C \text{ real subset Input 28a} \rangle$ ,
   $\langle C \text{ KronSums Answer 99d} \rangle$ 
)
{
  double *s;
  int *w;
   $\langle XfactorKronSums Body 106 \rangle$ 
}
```

◇

Fragment referenced in [96b](#).

Defines: `C_XfactorKronSums_iweights_dsubset 100`.

$\langle C_XfactorKronSums_iweights_isubset\ 105a \rangle \equiv$

```
void C_XfactorKronSums_iweights_isubset
(
   $\langle C\ XfactorKronSums\ Input\ 104a \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
)
{
  int *s, *w;
   $\langle XfactorKronSums\ Body\ 106 \rangle$ 
}
◇
```

Fragment referenced in 96b.

Defines: C_XfactorKronSums_iweights_isubset 100.

$\langle C_XfactorKronSums_dweights_isubset\ 105b \rangle \equiv$

```
void C_XfactorKronSums_dweights_isubset
(
   $\langle C\ XfactorKronSums\ Input\ 104a \rangle$ 
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
) {
  int *s;
  double *w;
   $\langle XfactorKronSums\ Body\ 106 \rangle$ 
}
◇
```

Fragment referenced in 96b.

Defines: C_XfactorKronSums_dweights_isubset 100.

< XfactorKronSums Body 106 > ≡

```
int *xx, ixi;
double *yy;

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

for (int q = 0; q < Q; q++) {
  yy = y + N * q;
  xx = x;
  < init subset loop 91b >
  < start subset loop 92a >
  {
    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 92b >
  }
  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 [104bc](#), [105ab](#).

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

Permuted Kronecker Sums

```
> a0 <- colSums(x[subset,r1] * y[subsety, r2])
> a1 <- .Call(libcoin:::R_KronSums_Permutation, x, P, y, subset, subsety)
> a2 <- .Call(libcoin:::R_KronSums_Permutation, x, P, y, as.double(subset), as.double(subsety))
> stopifnot(isequal(a0, a1) && isequal(a0, a1))
> a0 <- as.vector(colSums(Xfactor[subset,r1Xfactor] * y[subsety, r2Xfactor]))
> a1 <- .Call(libcoin:::R_KronSums_Permutation, ix, Lx, y, subset, subsety)
> a1 <- .Call(libcoin:::R_KronSums_Permutation, ix, Lx, y, as.double(subset), as.double(subsety))
> stopifnot(isequal(a0, a1))
```

$\langle R_KronSums_Permutation \text{ Prototype } 107a \rangle \equiv$

```
SEXP R_KronSums_Permutation
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
  SEXP P,
   $\langle R \text{ } y \text{ Input } 25d \rangle$ 
   $\langle R \text{ subset Input } 27b \rangle$ ,
  SEXP subsety
)
◇
```

Fragment referenced in [23b](#), [107b](#).

Uses: [P 25a](#), [R_KronSums_Permutation 107b](#).

$\langle R_KronSums_Permutation \text{ } 107b \rangle \equiv$

```
 $\langle R\_KronSums\_Permutation \text{ Prototype } 107a \rangle$ 
{
  SEXP ans;
   $\langle C \text{ integer } Q \text{ Input } 25e \rangle$ ;
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ;
   $\langle C \text{ integer } Nsubset \text{ Input } 27c \rangle$ ;

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

  PROTECT(ans = allocVector(REALSXP, INTEGER(P)[0] * Q));
  RC_KronSums_Permutation(x, N, INTEGER(P)[0], REAL(y), Q, subset, Offset0, Nsubset,
                          subsety, REAL(ans));
  UNPROTECT(1);
  return(ans);
}
◇
```

Fragment referenced in [96b](#).

Defines: [R_KronSums_Permutation 107a](#), [159](#), [160](#).

Uses: [N 24bc](#), [NCOL 138c](#), [Nsubset 27c](#), [Offset0 22b](#), [P 25a](#), [Q 25e](#), [RC_KronSums_Permutation 108a](#), [subset 27be](#), [28a](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

$\langle RC_KronSums_Permutation \text{ Prototype } 107c \rangle \equiv$

```
void RC_KronSums_Permutation
(
   $\langle R \text{ } x \text{ Input } 24d \rangle$ 
   $\langle C \text{ integer } N \text{ Input } 24c \rangle$ ,
   $\langle C \text{ integer } P \text{ Input } 25a \rangle$ ,
   $\langle C \text{ real } y \text{ Input } 26a \rangle$ 
   $\langle R \text{ subset Input } 27b \rangle$ ,
   $\langle C \text{ subset range Input } 27d \rangle$ ,
  SEXP subsety,
   $\langle C \text{ KronSums Answer } 99d \rangle$ 
)
◇
```

Fragment referenced in [108a](#).

Uses: [RC_KronSums_Permutation 108a](#).

$\langle RC_KronSums_Permutation\ 108a \rangle \equiv$

```
 $\langle RC\_KronSums\_Permutation\ Prototype\ 107c \rangle$ 
{
  if (TYPEOF(x) == INTSXP) {
    if (TYPEOF(subset) == INTSXP) {
      C_XfactorKronSums_Permutation_isubset(INTEGER(x), N, P, y, Q,
      INTEGER(subset), offset, Nsubset,
      INTEGER(subsety), PQ_ans);
    } else {
      C_XfactorKronSums_Permutation_dsubset(INTEGER(x), N, P, y, Q,
      REAL(subset), offset, Nsubset,
      REAL(subsety), PQ_ans);
    }
  } else {
    if (TYPEOF(subset) == INTSXP) {
      C_KronSums_Permutation_isubset(REAL(x), N, P, y, Q,
      INTEGER(subset), offset, Nsubset,
      INTEGER(subsety), PQ_ans);
    } else {
      C_KronSums_Permutation_dsubset(REAL(x), N, P, y, Q,
      REAL(subset), offset, Nsubset,
      REAL(subsety), PQ_ans);
    }
  }
}
}
◇
```

Fragment referenced in 96b.

Defines: RC_KronSums_Permutation 40, 107bc.

Uses: C_KronSums_Permutation_dsubset 108b, C_KronSums_Permutation_isubset 109a,
C_XfactorKronSums_Permutation_dsubset 110a, C_XfactorKronSums_Permutation_isubset 110b, N 24bc, Nsubset 27c,
offset 27d, P 25a, Q 25e, subset 27be, 28a, x 24d, 25bc, y 25d, 26ab.

$\langle C_KronSums_Permutation_dsubset\ 108b \rangle \equiv$

```
void C_KronSums_Permutation_dsubset
(
   $\langle C\ real\ x\ Input\ 25b \rangle$ 
   $\langle C\ real\ y\ Input\ 26a \rangle$ 
   $\langle C\ real\ subset\ Input\ 28a \rangle$ ,
  double *subsety,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
)
{
   $\langle KronSums\ Permutation\ Body\ 109b \rangle$ 
}
◇
```

Fragment referenced in 96b.

Defines: C_KronSums_Permutation_dsubset 108a.

$\langle C_KronSums_Permutation_isubset\ 109a \rangle \equiv$

```
void C_KronSums_Permutation_isubset
(
   $\langle C\ real\ x\ Input\ 25b \rangle$ 
   $\langle C\ real\ y\ Input\ 26a \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
  int *subset,
   $\langle C\ KronSums\ Answer\ 99d \rangle$ 
)
{
   $\langle KronSums\ Permutation\ Body\ 109b \rangle$ 
}
◇
```

Fragment referenced in [96b](#).

Defines: `C_KronSums_Permutation_isubset` [108a](#).

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

$\langle KronSums\ Permutation\ Body\ 109b \rangle \equiv$

```
R_xlen_t qP, qN, pN, qPp;

for (int q = 0; q < Q; q++) {
  qN = q * N;
  qP = q * P;
  for (int p = 0; p < P; p++) {
    qPp = qP + p;
    PQ_ans[qPp] = 0.0;
    pN = p * N;
    for (R_xlen_t i = offset; i < Nsubset; i++)
      PQ_ans[qPp] += y[qN + (R_xlen_t) subset[i] - 1] *
                    x[pN + (R_xlen_t) subset[i] - 1];
  }
}
◇
```

Fragment referenced in [108b](#), [109a](#).

Uses: `N` [24bc](#), `Nsubset` [27c](#), `offset` [27d](#), `P` [25a](#), `Q` [25e](#), `subset` [27be](#), [28a](#), `x` [24d](#), [25bc](#), `y` [25d](#), [26ab](#).

Xfactor Permuted Kronecker Sums

$\langle C_XfactorKronSums_Permutation_dsubset \ 110a \rangle \equiv$

```
void C_XfactorKronSums_Permutation_dsubset
(
   $\langle C \ integer \ x \ Input \ 25c \rangle$ 
   $\langle C \ real \ y \ Input \ 26a \rangle$ 
   $\langle C \ real \ subset \ Input \ 28a \rangle$ ,
  double *subsety,
   $\langle C \ KronSums \ Answer \ 99d \rangle$ 
)
{
   $\langle XfactorKronSums \ Permutation \ Body \ 110c \rangle$ 
}
◇
```

Fragment referenced in [96b](#).

Defines: `C_XfactorKronSums_Permutation_dsubset` [108a](#).

$\langle C_XfactorKronSums_Permutation_isubset \ 110b \rangle \equiv$

```
void C_XfactorKronSums_Permutation_isubset
(
   $\langle C \ integer \ x \ Input \ 25c \rangle$ 
   $\langle C \ real \ y \ Input \ 26a \rangle$ 
   $\langle C \ integer \ subset \ Input \ 27e \rangle$ ,
  int *subsety,
   $\langle C \ KronSums \ Answer \ 99d \rangle$ 
)
{
   $\langle XfactorKronSums \ Permutation \ Body \ 110c \rangle$ 
}
◇
```

Fragment referenced in [96b](#).

Defines: `C_XfactorKronSums_Permutation_isubset` [108a](#).

$\langle XfactorKronSums \ Permutation \ Body \ 110c \rangle \equiv$

```
R_xlen_t qP, qN;

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

for (int q = 0; q < Q; q++) {
  qP = q * P;
  qN = q * N;
  for (R_xlen_t i = offset; i < Nsubset; i++)
    PQ_ans[x[(R_xlen_t) subset[i] - 1] - 1 + qP] += y[qN + (R_xlen_t) subsety[i] - 1];
}
◇
```

Fragment referenced in [110ab](#).

Uses: `N` [24bc](#), `Nsubset` [27c](#), `offset` [27d](#), `P` [25a](#), `Q` [25e](#), `subset` [27be](#), [28a](#), `x` [24d](#), [25bc](#), `y` [25d](#), [26ab](#).

3.9.3 Column Sums

colSums 111a) ≡

⟨ *C_colSums_dweights_dsubset* 114a ⟩
⟨ *C_colSums_iweights_dsubset* 114b ⟩
⟨ *C_colSums_iweights_isubset* 114c ⟩
⟨ *C_colSums_dweights_isubset* 115a ⟩
⟨ *RC_colSums* 113a ⟩
⟨ *R_colSums* 112a ⟩
◇

Fragment referenced in 24a.

```
> a0 <- colSums(x[subset,] * weights[subset])
> a1 <- .Call(libcoin:::R_colSums, x, weights, subset)
> a2 <- .Call(libcoin:::R_colSums, x, as.double(weights), as.double(subset))
> a3 <- .Call(libcoin:::R_colSums, x, weights, as.double(subset))
> a4 <- .Call(libcoin:::R_colSums, x, as.double(weights), subset)
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
```

R_colSums Prototype 111b) ≡

```
SEXP R_colSums
(
  ⟨ R x Input 24d ⟩
  ⟨ R weights Input 26c ⟩,
  ⟨ R subset Input 27b ⟩
)
◇
```

Fragment referenced in 23b, 112a.

Uses: *R_colSums* 112a.

$\langle R_colSums\ 112a \rangle \equiv$

```
 $\langle R\_colSums\ Prototype\ 111b \rangle$ 
{
    SEXP ans;
    int P;
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;
    double center;

    P = NCOL(x);
    N = XLENGTH(x) / P;
    Nsubset = XLENGTH(subset);

    PROTECT(ans = allocVector(REALSXP, P));
    RC_colSums(REAL(x), N, P, Power1, &center, !DoCenter, weights, subset, Offset0,
              Nsubset, REAL(ans));
    UNPROTECT(1);
    return(ans);
}
◇
```

Fragment referenced in [111a](#).

Defines: [R_colSums 111b](#), [159](#), [160](#).

Uses: [DoCenter 22b](#), [N 24bc](#), [NCOL 138c](#), [Nsubset 27c](#), [Offset0 22b](#), [P 25a](#), [Power1 22b](#), [RC_colSums 113a](#), [subset 27be](#), [28a](#),
[weights 26c](#), [weights, 26de](#), [x 24d](#), [25bc](#).

$\langle RC_colSums\ Prototype\ 112b \rangle \equiv$

```
void RC_colSums
(
     $\langle C\ colSums\ Input\ 113b \rangle$ 
     $\langle R\ weights\ Input\ 26c \rangle$ ,
     $\langle R\ subset\ Input\ 27b \rangle$ ,
     $\langle C\ subset\ range\ Input\ 27d \rangle$ ,
     $\langle C\ colSums\ Answer\ 113c \rangle$ 
)
◇
```

Fragment referenced in [113a](#).

Uses: [RC_colSums 113a](#).

< RC_colSums 113a > ≡

```
< RC_colSums Prototype 112b >
{
  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 [111a](#).

Defines: [RC_colSums 84b](#), [86b](#), [88](#), [91a](#), [112ab](#).

Uses: [C_colSums_dweights_dsubset 114a](#), [C_colSums_dweights_isubset 115a](#), [C_colSums_iweights_dsubset 114b](#),
[C_colSums_iweights_isubset 114c](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#), [P 25a](#), [subset 27be](#), [28a](#), [weights 26c](#), [x 24d](#), [25bc](#).

< C_colSums Input 113b > ≡

```
< C real x Input 25b >
const int power,
double *centerx,
const int CENTER,
◇
```

Fragment referenced in [112b](#), [114abc](#), [115a](#).

< C_colSums Answer 113c > ≡

```
double *P_ans
◇
```

Fragment referenced in [84a](#), [112b](#), [114abc](#), [115a](#).

$\langle C_colSums_dweights_dsubset\ 114a \rangle \equiv$

```
void C_colSums_dweights_dsubset
(
   $\langle C\ colSums\ Input\ 113b \rangle$ 
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\ colSums\ Answer\ 113c \rangle$ 
)
{
  double *s, *w;
   $\langle colSums\ Body\ 115b \rangle$ 
}
◇
```

Fragment referenced in 111a.

Defines: C_colSums_dweights_dsubset 113a.

$\langle C_colSums_iweights_dsubset\ 114b \rangle \equiv$

```
void C_colSums_iweights_dsubset
(
   $\langle C\ colSums\ Input\ 113b \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\ colSums\ Answer\ 113c \rangle$ 
)
{
  double *s;
  int *w;
   $\langle colSums\ Body\ 115b \rangle$ 
}
◇
```

Fragment referenced in 111a.

Defines: C_colSums_iweights_dsubset 113a.

$\langle C_colSums_iweights_isubset\ 114c \rangle \equiv$

```
void C_colSums_iweights_isubset
(
   $\langle C\ colSums\ Input\ 113b \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ colSums\ Answer\ 113c \rangle$ 
)
{
  int *s, *w;
   $\langle colSums\ Body\ 115b \rangle$ 
}
◇
```

Fragment referenced in 111a.

Defines: C_colSums_iweights_isubset 113a.

< C_colSums_dweights_isubset 115a > ≡

```
void C_colSums_dweights_isubset
(
  < C_colSums Input 113b >
  < C_real_weights Input 26e >
  < C_integer_subset Input 27e >,
  < C_colSums Answer 113c >
)
{
  int *s;
  double *w;
  < colSums Body 115b >
}
◇
```

Fragment referenced in [111a](#).

Defines: C_colSums_dweights_isubset [113a](#).

< colSums Body 115b > ≡

```
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 91b >
  < start_subset loop 92a >
  {
    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 92b >
  }
  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 [114abc](#), [115a](#).

Uses: HAS_WEIGHTS [26de](#), N [24bc](#), P [25a](#), x [24d](#), [25bc](#).

3.9.4 Tables

OneTable Sums

⟨ Tables 116a ⟩ ≡

```
⟨ C_OneTableSums_dweights_dsubset 119a ⟩
⟨ C_OneTableSums_iweights_dsubset 119b ⟩
⟨ C_OneTableSums_iweights_isubset 119c ⟩
⟨ C_OneTableSums_dweights_isubset 120a ⟩
⟨ RC_OneTableSums 118a ⟩
⟨ R_OneTableSums 117a ⟩
⟨ C_TwoTableSums_dweights_dsubset 123b ⟩
⟨ C_TwoTableSums_iweights_dsubset 123c ⟩
⟨ C_TwoTableSums_iweights_isubset 124a ⟩
⟨ C_TwoTableSums_dweights_isubset 124b ⟩
⟨ RC_TwoTableSums 122b ⟩
⟨ R_TwoTableSums 121b ⟩
⟨ C_ThreeTableSums_dweights_dsubset 128b ⟩
⟨ C_ThreeTableSums_iweights_dsubset 128c ⟩
⟨ C_ThreeTableSums_iweights_isubset 129a ⟩
⟨ C_ThreeTableSums_dweights_isubset 129b ⟩
⟨ RC_ThreeTableSums 127b ⟩
⟨ R_ThreeTableSums 126b ⟩
◇
```

Fragment referenced in 24a.

```
> a0 <- as.vector(xtabs(weights ~ ixf, subset = subset))
> a1 <- ctabs(ix, weights = weights, subset = subset)[-1]
> a2 <- ctabs(ix, weights = as.double(weights), subset = as.double(subset))[-1]
> a3 <- ctabs(ix, weights = weights, subset = as.double(subset))[-1]
> a4 <- ctabs(ix, weights = as.double(weights), subset = subset)[-1]
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
```

⟨ R_OneTableSums Prototype 116b ⟩ ≡

```
SEXP R_OneTableSums
(
  ⟨ R x Input 24d ⟩
  ⟨ R weights Input 26c ⟩,
  ⟨ R subset Input 27b ⟩
)
◇
```

Fragment referenced in 23b, 117a.

Uses: R_OneTableSums 117a.

$\langle R_OneTableSums\ 117a \rangle \equiv$

```
 $\langle R\_OneTableSums\ Prototype\ 116b \rangle$ 
{
    SEXP ans;
     $\langle C\ integer\ N\ Input\ 24c \rangle$ ;
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ;
    int P;

    N = XLENGTH(x);
    Nsubset = XLENGTH(subset);
    P = NLEVELS(x) + 1;

    PROTECT(ans = allocVector(REALSXP, P));
    RC_OneTableSums(INTEGER(x), N, P, weights, subset,
                   Offset0, Nsubset, REAL(ans));
    UNPROTECT(1);
    return(ans);
}
◇
```

Fragment referenced in [116a](#).

Defines: [R_OneTableSums 15b](#), [116b](#), [131b](#), [159](#), [160](#).

Uses: [N 24bc](#), [NLEVELS 139a](#), [Nsubset 27c](#), [Offset0 22b](#), [P 25a](#), [RC_OneTableSums 118a](#), [subset 27be](#), [28a](#), [weights 26c](#), [weights, 26de](#), [x 24d](#), [25bc](#).

$\langle RC_OneTableSums\ Prototype\ 117b \rangle \equiv$

```
void RC_OneTableSums
(
     $\langle C\ OneTableSums\ Input\ 118b \rangle$ 
     $\langle R\ weights\ Input\ 26c \rangle$ ,
     $\langle R\ subset\ Input\ 27b \rangle$ ,
     $\langle C\ subset\ range\ Input\ 27d \rangle$ ,
     $\langle C\ OneTableSums\ Answer\ 118c \rangle$ 
)
◇
```

Fragment referenced in [118a](#).

Uses: [RC_OneTableSums 118a](#).

< RC_OneTableSums 118a > ≡

```
< RC_OneTableSums Prototype 117b >
{
    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 [116a](#).

Defines: [RC_OneTableSums 36a](#), [40](#), [88](#), [117ab](#).

Uses: [C_OneTableSums_dweights_dsubset 119a](#), [C_OneTableSums_dweights_isubset 120a](#),
[C_OneTableSums_iweights_dsubset 119b](#), [C_OneTableSums_iweights_isubset 119c](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#),
[P 25a](#), [subset 27be](#), [28a](#), [weights 26c](#), [x 24d](#), [25bc](#).

< C_OneTableSums Input 118b > ≡

```
    < C integer x Input 25c >
    ◇
```

Fragment referenced in [117b](#), [119abc](#), [120a](#).

< C_OneTableSums Answer 118c > ≡

```
    double *P_ans
    ◇
```

Fragment referenced in [87b](#), [117b](#), [119abc](#), [120a](#).

$\langle C_OneTableSums_dweights_dsubset\ 119a \rangle \equiv$

```
void C_OneTableSums_dweights_dsubset
(
   $\langle C\_OneTableSums\_Input\ 118b \rangle$ 
   $\langle C\_real\ weights\ Input\ 26e \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_OneTableSums\_Answer\ 118c \rangle$ 
)
{
  double *s, *w;
   $\langle OneTableSums\ Body\ 120b \rangle$ 
}
◇
```

Fragment referenced in 116a.

Defines: C_OneTableSums_dweights_dsubset 118a.

$\langle C_OneTableSums_iweights_dsubset\ 119b \rangle \equiv$

```
void C_OneTableSums_iweights_dsubset
(
   $\langle C\_OneTableSums\_Input\ 118b \rangle$ 
   $\langle C\_integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\_real\ subset\ Input\ 28a \rangle$ ,
   $\langle C\_OneTableSums\_Answer\ 118c \rangle$ 
)
{
  double *s;
  int *w;
   $\langle OneTableSums\ Body\ 120b \rangle$ 
}
◇
```

Fragment referenced in 116a.

Defines: C_OneTableSums_iweights_dsubset 118a.

$\langle C_OneTableSums_iweights_isubset\ 119c \rangle \equiv$

```
void C_OneTableSums_iweights_isubset
(
   $\langle C\_OneTableSums\_Input\ 118b \rangle$ 
   $\langle C\_integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\_integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\_OneTableSums\_Answer\ 118c \rangle$ 
)
{
  int *s, *w;
   $\langle OneTableSums\ Body\ 120b \rangle$ 
}
◇
```

Fragment referenced in 116a.

Defines: C_OneTableSums_iweights_isubset 118a.

< C_OneTableSums_dweights_isubset 120a > ≡

```
void C_OneTableSums_dweights_isubset
(
  < C_OneTableSums Input 118b >
  < C_real_weights Input 26e >
  < C_integer_subset Input 27e >,
  < C_OneTableSums Answer 118c >
)
{
  int *s;
  double *w;
  < OneTableSums Body 120b >
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_OneTableSums_dweights_isubset` [118a](#).

< OneTableSums Body 120b > ≡

```
int *xx;

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

xx = x;
< init_subset_loop 91b >
< start_subset_loop 92a >
{
  xx = xx + diff;
  if (HAS_WEIGHTS) {
    w = w + diff;
    P_ans[xx[0]] += (double) w[0];
  } else {
    P_ans[xx[0]]++;
  }
  < continue_subset_loop 92b >
}
xx = xx + diff;
if (HAS_WEIGHTS) {
  w = w + diff;
  P_ans[xx[0]] += w[0];
} else {
  P_ans[xx[0]]++;
}
}
◇
```

Fragment referenced in [119abc](#), [120a](#).

Uses: `HAS_WEIGHTS` [26de](#), `P` [25a](#), `x` [24d](#), [25bc](#).

TwoTable Sums

```
> a0 <- xtabs(weights ~ ixf + iyf, subset = subset)
> class(a0) <- "matrix"
> dimnames(a0) <- NULL
> attributes(a0)$call <- NULL
```

```

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

```

< R_TwoTableSums Prototype 121a > ≡

```

SEXP R_TwoTableSums
(
  < R x Input 24d >
  < R y Input 25d >
  < R weights Input 26c >,
  < R subset Input 27b >
)
◇

```

Fragment referenced in [23b](#), [121b](#).

Uses: [R_TwoTableSums 121b](#).

< R_TwoTableSums 121b > ≡

```

< R_TwoTableSums Prototype 121a >
{
  SEXP ans, dim;
  < C integer N Input 24c >;
  < C integer Nsubset Input 27c >;
  int P, Q;

  N = XLENGTH(x);
  Nsubset = XLENGTH(subset);
  P = NLEVELS(x) + 1;
  Q = NLEVELS(y) + 1;

  PROTECT(ans = allocVector(REALSXP, P * Q));
  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 [116a](#).

Defines: [R_TwoTableSums 15b](#), [121a](#), [159](#), [160](#).

Uses: [N 24bc](#), [NLEVELS 139a](#), [Nsubset 27c](#), [Offset0 22b](#), [P 25a](#), [Q 25e](#), [RC_TwoTableSums 122b](#), [subset 27be](#), [28a](#), [weights 26c](#), [weights, 26de](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

< RC_TwoTableSums Prototype 122a > ≡

```
void RC_TwoTableSums
(
  < C TwoTableSums Input 122c >
  < R weights Input 26c >,
  < R subset Input 27b >,
  < C subset range Input 27d >,
  < C TwoTableSums Answer 123a >
)
◇
```

Fragment referenced in [122b](#).
Uses: [RC_TwoTableSums 122b](#).

< RC_TwoTableSums 122b > ≡

```
< RC_TwoTableSums Prototype 122a >
{
  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 [116a](#).

Defines: [RC_TwoTableSums 44](#), [121b](#), [122a](#).

Uses: [C_TwoTableSums_dweights_dsubset 123b](#), [C_TwoTableSums_dweights_isubset 124b](#),
[C_TwoTableSums_iweights_dsubset 123c](#), [C_TwoTableSums_iweights_isubset 124a](#), [N 24bc](#), [Nsubset 27c](#), [offset 27d](#),
[P 25a](#), [Q 25e](#), [subset 27be](#), [28a](#), [weights 26c](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

< C TwoTableSums Input 122c > ≡

```
< C integer x Input 25c >
< C integer y Input 26b >
◇
```

Fragment referenced in [122a](#), [123bc](#), [124ab](#).

< C_TwoTableSums Answer 123a > ≡

```
double *PQ_ans
◇
```

Fragment referenced in [122a](#), [123bc](#), [124ab](#).

< C_TwoTableSums_dweights_dsubset 123b > ≡

```
void C_TwoTableSums_dweights_dsubset
(
  < C_TwoTableSums Input 122c >
  < C real weights Input 26e >
  < C real subset Input 28a >,
  < C_TwoTableSums Answer 123a >
)
{
  double *s, *w;
  < TwoTableSums Body 125 >
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_TwoTableSums_dweights_dsubset` [122b](#).

< C_TwoTableSums_iweights_dsubset 123c > ≡

```
void C_TwoTableSums_iweights_dsubset
(
  < C_TwoTableSums Input 122c >
  < C integer weights Input 26d >
  < C real subset Input 28a >,
  < C_TwoTableSums Answer 123a >
)
{
  double *s;
  int *w;
  < TwoTableSums Body 125 >
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_TwoTableSums_iweights_dsubset` [122b](#).

$\langle C_TwoTableSums_iweights_isubset\ 124a \rangle \equiv$

```
void C_TwoTableSums_iweights_isubset
(
   $\langle C\ TwoTableSums\ Input\ 122c \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ TwoTableSums\ Answer\ 123a \rangle$ 
)
{
  int *s, *w;
   $\langle TwoTableSums\ Body\ 125 \rangle$ 
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_TwoTableSums_iweights_isubset` [122b](#).

$\langle C_TwoTableSums_dweights_isubset\ 124b \rangle \equiv$

```
void C_TwoTableSums_dweights_isubset
(
   $\langle C\ TwoTableSums\ Input\ 122c \rangle$ 
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ TwoTableSums\ Answer\ 123a \rangle$ 
)
{
  int *s;
  double *w;
   $\langle TwoTableSums\ Body\ 125 \rangle$ 
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_TwoTableSums_dweights_isubset` [122b](#).

⟨ *TwoTableSums Body 125* ⟩ ≡

```
int *xx, *yy;

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

yy = y;
xx = x;
⟨ init subset loop 91b ⟩
⟨ start subset loop 92a ⟩
{
    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 92b ⟩
}
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 [123bc](#), [124ab](#).

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

ThreeTable Sums

```
> a0 <- xtabs(weights ~ ixf + iyf + block, subset = subset)
> class(a0) <- "array"
> dimnames(a0) <- NULL
> attributes(a0)$call <- NULL
> a1 <- ctabs(ix, iy, block, weights, subset)[-1, -1,]
> a2 <- ctabs(ix, iy, block, as.double(weights), as.double(subset))[-1,-1,]
> a3 <- ctabs(ix, iy, block, weights, as.double(subset))[-1,-1,]
> a4 <- ctabs(ix, iy, block, as.double(weights), subset)[-1,-1,]
> stopifnot(isequal(a0, a1) && isequal(a0, a2) &&
+           isequal(a0, a3) && isequal(a0, a4))
```


< R_ThreeTableSums Prototype 126a > ≡

```
SEXP R_ThreeTableSums
(
  < R x Input 24d >
  < R y Input 25d >
  < R block Input 28b >,
  < R weights Input 26c >,
  < R subset Input 27b >
)
◇
```

Fragment referenced in [23b](#), [126b](#).

Uses: [R_ThreeTableSums 126b](#).

< R_ThreeTableSums 126b > ≡

```
< R_ThreeTableSums Prototype 126a >
{
  SEXP ans, dim;
  < C integer N Input 24c >;
  < C integer Nsubset Input 27c >;
  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, 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 [116a](#).

Defines: [R_ThreeTableSums 15b](#), [126a](#), [159](#), [160](#).

Uses: [B 28c](#), [block 28bd](#), [N 24bc](#), [NLEVELS 139a](#), [Nsubset 27c](#), [Offset0 22b](#), [P 25a](#), [Q 25e](#), [RC_ThreeTableSums 127b](#),
[subset 27be, 28a](#), [weights 26c](#), [weights, 26de](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

< RC_ThreeTableSums Prototype 127a > ≡

```
void RC_ThreeTableSums
(
  < C ThreeTableSums Input 127c >
  < R weights Input 26c >,
  < R subset Input 27b >,
  < C subset range Input 27d >,
  < C ThreeTableSums Answer 128a >
)
◇
```

Fragment referenced in [127b](#).
Uses: [RC_ThreeTableSums 127b](#).

< RC_ThreeTableSums 127b > ≡

```
< RC_ThreeTableSums Prototype 127a >
{
  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 [116a](#).
Defines: [RC_ThreeTableSums 44](#), [126b](#), [127a](#).
Uses: [B 28c](#), [block 28bd](#), [C_ThreeTableSums_dweights_dsubset 128b](#), [C_ThreeTableSums_dweights_isubset 129b](#),
[C_ThreeTableSums_iweights_dsubset 128c](#), [C_ThreeTableSums_iweights_isubset 129a](#), [N 24bc](#), [Nsubset 27c](#),
[offset 27d](#), [P 25a](#), [Q 25e](#), [subset 27be](#), [28a](#), [weights 26c](#), [x 24d](#), [25bc](#), [y 25d](#), [26ab](#).

< C ThreeTableSums Input 127c > ≡

```
< C integer x Input 25c >
< C integer y Input 26b >
< C integer block Input 28d >
◇
```

Fragment referenced in [127a](#), [128bc](#), [129ab](#).

< C ThreeTableSums Answer 128a > ≡

```
double *PQL_ans
◇
```

Fragment referenced in [127a](#), [128bc](#), [129ab](#).

< C_ThreeTableSums_dweights_dsubset 128b > ≡

```
void C_ThreeTableSums_dweights_dsubset
(
  < C ThreeTableSums Input 127c >
  < C real weights Input 26e >
  < C real subset Input 28a >,
  < C ThreeTableSums Answer 128a >
)
{
  double *s, *w;
  < ThreeTableSums Body 130a >
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_ThreeTableSums_dweights_dsubset` [127b](#).

< C_ThreeTableSums_iweights_dsubset 128c > ≡

```
void C_ThreeTableSums_iweights_dsubset
(
  < C ThreeTableSums Input 127c >
  < C integer weights Input 26d >
  < C real subset Input 28a >,
  < C ThreeTableSums Answer 128a >
)
{
  double *s;
  int *w;
  < ThreeTableSums Body 130a >
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_ThreeTableSums_iweights_dsubset` [127b](#).

$\langle C_ThreeTableSums_iweights_isubset\ 129a \rangle \equiv$

```
void C_ThreeTableSums_iweights_isubset
(
   $\langle C\ ThreeTableSums\ Input\ 127c \rangle$ 
   $\langle C\ integer\ weights\ Input\ 26d \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ ThreeTableSums\ Answer\ 128a \rangle$ 
)
{
  int *s, *w;
   $\langle ThreeTableSums\ Body\ 130a \rangle$ 
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_ThreeTableSums_iweights_isubset` [127b](#).

$\langle C_ThreeTableSums_dweights_isubset\ 129b \rangle \equiv$

```
void C_ThreeTableSums_dweights_isubset
(
   $\langle C\ ThreeTableSums\ Input\ 127c \rangle$ 
   $\langle C\ real\ weights\ Input\ 26e \rangle$ 
   $\langle C\ integer\ subset\ Input\ 27e \rangle$ ,
   $\langle C\ ThreeTableSums\ Answer\ 128a \rangle$ 
)
{
  int *s;
  double *w;
   $\langle ThreeTableSums\ Body\ 130a \rangle$ 
}
◇
```

Fragment referenced in [116a](#).

Defines: `C_ThreeTableSums_dweights_isubset` [127b](#).

⟨ *ThreeTableSums Body 130a* ⟩ ≡

```
int *xx, *yy, *bb, PQ = P * Q;

for (int p = 0; p < PQ * B; p++) PQL_ans[p] = 0.0;

yy = y;
xx = x;
bb = block;
⟨ init subset loop 91b ⟩
⟨ start subset loop 92a ⟩
{
    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 92b ⟩
}
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 [128bc](#), [129ab](#).

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

3.10 Utilities

3.10.1 Blocks

```
> sb <- sample(block)
> ns1 <- do.call("c", tapply(subset, sb[subset], function(i) i))
> ns2 <- .Call(libcoin:::R_order_subset_wrt_block, y, integer(0), subset, sb)
> stopifnot(isequal(ns1, ns2))
```

⟨ *Utils 130b* ⟩ ≡

```
⟨ C_setup_subset 133a ⟩
⟨ C_setup_subset_block 133b ⟩
⟨ C_order_subset_wrt_block 134a ⟩
⟨ RC_order_subset_wrt_block 132b ⟩
⟨ R_order_subset_wrt_block 131b ⟩
}
```

Fragment referenced in [24a](#).

< R_order_subset_wrt_block Prototype 131a > ≡

```
SEXP R_order_subset_wrt_block
(
  < R y Input 25d >
  < R weights Input 26c >,
  < R subset Input 27b >,
  < R block Input 28b >
)
◇
```

Fragment referenced in [23b](#), [131b](#).
Uses: [R_order_subset_wrt_block 131b](#).

< R_order_subset_wrt_block 131b > ≡

```
< R_order_subset_wrt_block Prototype 131a >
{
  < C integer N Input 24c >;
  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 [130b](#).

Defines: [R_order_subset_wrt_block 131a](#), [159](#), [160](#).

Uses: [block 28bd](#), [blockTable 28e](#), [N 24bc](#), [NCOL 138c](#), [NLEVELS 139a](#), [Offset0 22b](#), [RC_order_subset_wrt_block 132b](#), [RC_Sums 94a](#), [R_OneTableSums 117a](#), [subset 27be, 28a](#), [weights 26c](#), [weights, 26de](#), [y 25d, 26ab](#).

< RC_order_subset_wrt_block Prototype 132a > ≡

```
SEXP RC_order_subset_wrt_block
(
  < C integer N Input 24c >,
  < R subset Input 27b >,
  < R block Input 28b >,
  < R blockTable Input 28e >
)
◇
```

Fragment referenced in [132b](#).

Uses: [RC_order_subset_wrt_block 132b](#).

< RC_order_subset_wrt_block 132b > ≡

```
< RC_order_subset_wrt_block Prototype 132a >
{
  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 [130b](#).

Defines: [RC_order_subset_wrt_block 36a, 40, 131b, 132a](#).

Uses: [block 28bd, blockTable 28e, C_order_subset_wrt_block 134a, C_setup_subset 133a, C_setup_subset_block 133b, N 24bc, subset 27be, 28a](#).

< C_setup_subset 133a > ≡

```
void C_setup_subset
(
  < C integer N Input 24c >,
  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 [130b](#).

Defines: `C_setup_subset` [132b](#), [135a](#).

Uses: `N` [24bc](#).

< C_setup_subset_block 133b > ≡

```
void C_setup_subset_block
(
  < C integer N Input 24c >,
  < R block Input 28b >,
  < R blockTable Input 28e >,
  SEXP ans
)
{
  double *cumtable;
  int Nlevels = LENGTH(blockTable);

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

  /* table[0] are missings, ie block == 0 ! */
  for (int k = 1; k < Nlevels; k++)
    cumtable[k] = cumtable[k - 1] + REAL(blockTable)[k - 1];

  for (R_xlen_t i = 0; i < N; i++) {
    /* ans is R style index in 1:N */
    if (TYPEOF(ans) == INTSXP) {
      INTEGER(ans)[(int) cumtable[INTEGER(block)[i]]++] = i + 1;
    } else {
      REAL(ans)[(R_xlen_t) cumtable[INTEGER(block)[i]]++] = (double) i + 1;
    }
  }

  Free(cumtable);
}
◇
```

Fragment referenced in [130b](#).

Defines: `C_setup_subset_block` [132b](#).

Uses: `block` [28bd](#), `blockTable` [28e](#), `N` [24bc](#).

< C_order_subset_wrt_block 134a > ≡

```
void C_order_subset_wrt_block
(
  < R subset Input 27b >,
  < R block Input 28b >,
  < R blockTable Input 28e >,
  SEXP ans
)
{
  double *cumtable;
  int Nlevels = LENGTH(blockTable);

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

  /* table[0] are missings, ie block == 0 ! */
  for (int k = 1; k < Nlevels; k++)
    cumtable[k] = cumtable[k - 1] + REAL(blockTable)[k - 1];

  /* subset is R style index in 1:N */
  if (TYPEOF(subset) == INTSXP) {
    for (R_xlen_t i = 0; i < XLENGTH(subset); i++)
      INTEGER(ans)[(int) cumtable[INTEGER(block)[INTEGER(subset)[i] -
1]]++] = INTEGER(subset)[i];
  } else {
    for (R_xlen_t i = 0; i < XLENGTH(subset); i++)
      REAL(ans)[(R_xlen_t) cumtable[INTEGER(block)[(R_xlen_t) REAL(subset)[i] -
1]]++] = REAL(subset)[i];
  }

  Free(cumtable);
}
◇
```

Fragment referenced in [130b](#).

Defines: `C_order_subset_wrt_block` [132b](#).

Uses: `block` [28bd](#), `blockTable` [28e](#), `N` [24bc](#), `subset` [27be](#), [28a](#).

< RC_setup_subset Prototype 134b > ≡

```
SEXP RC_setup_subset
(
  < C integer N Input 24c >,
  < R weights Input 26c >,
  < R subset Input 27b >
)
◇
```

Fragment referenced in [135a](#).

Uses: `RC_setup_subset` [135a](#).

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 135a > ≡

```
< RC_setup_subset Prototype 134b >
{
  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 [135b](#).

Defines: [RC_setup_subset 40](#), [134b](#).

Uses: [C_setup_subset 133a](#), [N 24bc](#), [Offset0 22b](#), [RC_Sums 94a](#), [subset 27be, 28a](#), [sumweights 27a](#), [weights 26c](#), [weights, 26de](#).

3.10.2 Permutation Helpers

< Permutations 135b > ≡

```
< RC_setup_subset 135a >
< C_Permute 136a >
< C_doPermute 136b >
< C_PermuteBlock 137a >
< C_doPermuteBlock 137b >
◇
```

Fragment referenced in [24a](#).

< C_Permute 136a > ≡

```
void C_Permute
(
    double *subset,
    < C integer Nsubset Input 27c >,
    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 [135b](#).
Defines: [C_Permute 136b](#), [137a](#).
Uses: [Nsubset 27c](#), [subset 27be](#), [28a](#).

< C_doPermute 136b > ≡

```
void C_doPermute
(
    double *subset,
    < C integer Nsubset Input 27c >,
    double *Nsubset_tmp,
    double *perm
) {
    Memcpy(Nsubset_tmp, subset, Nsubset);
    C_Permute(Nsubset_tmp, Nsubset, perm);
}
◇
```

Fragment referenced in [135b](#).
Defines: [C_doPermute 40](#).
Uses: [C_Permute 136a](#), [Nsubset 27c](#), [subset 27be](#), [28a](#).

$\langle C_PermuteBlock\ 137a \rangle \equiv$

```
void C_PermuteBlock
(
    double *subset,
    double *table,
    int Nlevels,
    double *ans
) {

    double *px, *pans;

    px = subset;
    pans = ans;

    for (R_xlen_t j = 0; j < Nlevels; j++) {
        if (table[j] > 0) {
            C_Permute(px, (R_xlen_t) table[j], pans);
            px += (R_xlen_t) table[j];
            pans += (R_xlen_t) table[j];
        }
    }
}
◇
```

Fragment referenced in [135b](#).
Defines: [C_PermuteBlock 137b](#).
Uses: [C_Permute 136a](#), [subset 27be, 28a](#).

$\langle C_doPermuteBlock\ 137b \rangle \equiv$

```
void C_doPermuteBlock
(
    double *subset,
     $\langle C\ integer\ Nsubset\ Input\ 27c \rangle$ ,
    double *table,
    int Nlevels,
    double *Nsubset_tmp,
    double *perm
) {
    Memcpy(Nsubset_tmp, subset, Nsubset);
    C_PermuteBlock(Nsubset_tmp, table, Nlevels, perm);
}
◇
```

Fragment referenced in [135b](#).
Defines: [C_doPermuteBlock 40](#).
Uses: [C_PermuteBlock 137a](#), [Nsubset 27c](#), [subset 27be, 28a](#).

3.10.3 Other Utils

⟨ *MoreUtils* 138a ⟩ ≡

```
⟨ NROW 138b ⟩
⟨ NCOL 138c ⟩
⟨ NLEVELS 139a ⟩
⟨ C_kronecker 141 ⟩
⟨ C_kronecker_sym 142 ⟩
⟨ C_KronSums_sym 143 ⟩
⟨ C_MPinv_sym 144 ⟩
⟨ R_kronecker 140b ⟩
◇
```

Fragment referenced in 24a.

⟨ *NROW* 138b ⟩ ≡

```
int NROW
(
  SEXP x
) {
  SEXP a;
  a = getAttrib(x, R_DimSymbol);
  if (a == R_NilValue) return(XLENGTH(x));
  if (TYPEOF(a) == REALSXP)
    return(REAL(a)[0]);
  return(INTEGER(a)[0]);
}
◇
```

Fragment referenced in 138a.

Defines: *NROW* 6, 8, 9ab, 14, 35a, 40, 46c, 47, 139a, 140b.

Uses: *x* 24d, 25bc.

⟨ *NCOL* 138c ⟩ ≡

```
int NCOL
(
  SEXP x
) {
  SEXP a;
  a = getAttrib(x, R_DimSymbol);
  if (a == R_NilValue) return(1);
  if (TYPEOF(a) == REALSXP)
    return(REAL(a)[1]);
  return(INTEGER(a)[1]);
}
◇
```

Fragment referenced in 138a.

Defines: *NCOL* 12, 33, 45a, 83b, 85b, 98a, 107b, 112a, 131b, 140b.

Uses: *x* 24d, 25bc.

`< NLEVELS 139a > ≡`

```
int NLEVELS
(
  SEXP x
) {

  SEXP a;
  int maxlev = 0;

  a = getAttrib(x, R_LevelsSymbol);
  if (a == R_NilValue) {
    if (TYPEOF(x) != INTSXP)
      error("cannot determine number of levels");
    for (R_xlen_t i = 0; i < XLENGTH(x); i++) {
      if (INTEGER(x)[i] > maxlev)
        maxlev = INTEGER(x)[i];
    }
    return(maxlev);
  }
  return(NROW(a));
}
◇
```

Fragment referenced in [138a](#).

Defines: [NLEVELS 33](#), [45a](#), [117a](#), [121b](#), [126b](#), [131b](#).

Uses: [NROW 138b](#), [x 24d](#), [25bc](#).

```
> A <- matrix(runif(12), ncol = 3)
> B <- matrix(runif(10), ncol = 2)
> K1 <- kronecker(A, B)
> K2 <- .Call(libcoin:::R_kronecker, A, B)
> stopifnot(isequal(K1, K2))
```

`"libcoinAPI.h" 139b ≡`

```
extern SEXP libcoin_R_kronecker(
  SEXP A, SEXP B
) {

  static SEXP(*fun)(SEXP, SEXP) = NULL;
  if(fun == NULL)
    fun = (SEXP*)(SEXP, SEXP)
      R_GetCCallable("libcoin", "R_kronecker");
  return fun(A, B);
}
◇
```

File defined by [32a](#), [38d](#), [41b](#), [43b](#), [50b](#), [53b](#), [139b](#).

Uses: [B 28c](#).

< R_kronecker Prototype 140a > ≡

```
SEXP R_kronecker
(
  SEXP A,
  SEXP B
)
◇
```

Fragment referenced in [23b](#), [140b](#).
Uses: [B 28c](#).

< R_kronecker 140b > ≡

```
< R_kronecker Prototype 140a >
{
  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 [138a](#).
Uses: [B 28c](#), [C_kronecker 141](#), [NCOL 138c](#), [NROW 138b](#).

C_kronecker 141 ≡

```
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 i, j, k, l, mr, js, ir;
    double y;

    if (overwrite) {
        for (i = 0; i < m * r * n * s; i++) ans[i] = 0.0;
    }

    mr = m * r;
    for (i = 0; i < m; i++) {
        ir = i * r;
        for (j = 0; j < n; j++) {
            js = j * s;
            y = A[j*m + i];
            for (k = 0; k < r; k++) {
                for (l = 0; l < s; l++)
                    ans[(js + l) * mr + ir + k] += y * B[l * r + k];
            }
        }
    }
}
◇
```

Fragment referenced in [138a](#).
Defines: [C_kronecker](#) [82](#), [140b](#).
Uses: [B](#) [28c](#), [y](#) [25d](#), [26ab](#).

< C_kronecker_sym 142 > ≡

```
void C_kronecker_sym
(
    const double *A,
    const int m,
    const double *B,
    const int r,
    const int overwrite,
    double *ans
) {

    int i, j, k, l, mr, js, ir, s;
    double y;

    mr = m * r;
    s = r;

    if (overwrite) {
        for (i = 0; i < mr * (mr + 1) / 2; i++) ans[i] = 0.0;
    }

    for (i = 0; i < m; i++) {
        ir = i * r;
        for (j = 0; j <= i; j++) {
            js = j * s;
            y = A[S(i, j, m)];
            for (k = 0; k < r; k++) {
                for (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 [138a](#).

Defines: `C_kronecker_sym` [81](#).

Uses: [B 28c](#), [S 22a](#), [y 25d](#), [26ab](#).

$\langle C_KronSums_sym\ 143 \rangle \equiv$

```
/* sum_i (t(x[i,]) %*% x[i,]) */
void C_KronSums_sym_
(
   $\langle C\ real\ x\ Input\ 25b \rangle$ 
  double *PP_sym_ans
) {

  int pN, qN, SpqP;

  for (int q = 0; q < P; q++) {
    qN = q * N;
    for (int p = 0; p <= q; p++) {
      PP_sym_ans[S(p, q, P)] = 0.0;
      pN = p * N;
      SpqP = S(p, q, P);
      for (int i = 0; i < N; i++)
        PP_sym_ans[SpqP] += x[qN + i] * x[pN + i];
    }
  }
}
◇
```

Fragment referenced in [138a](#).

Defines: `C_KronSums_sym` Never used.

Uses: `N` [24bc](#), `P` [25a](#), `S` [22a](#), `x` [24d](#), [25bc](#).

< C_MPinv_sym 144 > ≡

```
void C_MPinv_sym
(
    const double *x,
    const int n,
    const double tol,
    double *dMP,
    int *rank
) {

    double *val, *vec, dtol, *rx, *work, valinv;
    int valzero = 0, info = 0, kn;

    if (n == 1) {
        if (x[0] > tol) {
            dMP[0] = 1 / x[0];
            rank[0] = 1;
        } else {
            dMP[0] = 0;
            rank[0] = 0;
        }
    } else {
        rx = Calloc(n * (n + 1) / 2, double);
        Memcpy(rx, x, n * (n + 1) / 2);
        work = Calloc(3 * n, double);
        val = Calloc(n, double);
        vec = Calloc(n * n, double);

        F77_CALL(dspev)("V", "L", &n, rx, val, vec, &n, work,
                       &info);

        dtol = val[n - 1] * tol;

        for (int k = 0; k < n; k++)
            valzero += (val[k] < dtol);
        rank[0] = n - valzero;

        for (int k = 0; k < n * (n + 1) / 2; k++) dMP[k] = 0.0;

        for (int k = valzero; k < n; k++) {
            valinv = 1 / val[k];
            kn = k * n;
            for (int i = 0; i < n; i++) {
                for (int j = 0; j <= i; j++) {
                    /* MP is symmetric */
                    dMP[S(i, j, n)] += valinv * vec[kn + i] * vec[kn + j];
                }
            }
        }
        Free(rx); Free(work); Free(val); Free(vec);
    }
}
◇
```

Fragment referenced in [138a](#).
Uses: [S 22a](#), [x 24d](#), [25bc](#).

3.11 Memory

⟨ *Memory* 145a ⟩ ≡

```
⟨ C_get_P 145c ⟩
⟨ C_get_Q 146a ⟩
⟨ C_get_varonly 146b ⟩
⟨ C_get_Xfactor 146c ⟩
⟨ C_get_LinearStatistic 147a ⟩
⟨ C_get_Expectation 147b ⟩
⟨ C_get_Variance 147c ⟩
⟨ C_get_Covariance 148a ⟩
⟨ C_get_ExpectationX 148b ⟩
⟨ C_get_ExpectationInfluence 148c ⟩
⟨ C_get_CovarianceInfluence 149a ⟩
⟨ C_get_VarianceInfluence 149b ⟩
⟨ C_get_TableBlock 149c ⟩
⟨ C_get_Sumweights 150a ⟩
⟨ C_get_Table 150b ⟩
⟨ C_get_dimTable 150c ⟩
⟨ C_get_B 151a ⟩
⟨ C_get_nresample 151b ⟩
⟨ C_get_PermutedLinearStatistic 151c ⟩
⟨ C_get_tol 152a ⟩
⟨ RC_init_LECV_1d 155b ⟩
⟨ RC_init_LECV_2d 156 ⟩
◇
```

Fragment referenced in 24a.

⟨ *R_LECV Input* 145b ⟩ ≡

```
SEXP LECV
◇
```

Fragment referenced in 54, 56b, 145c, 146abc, 147abc, 148abc, 149abc, 150abc, 151abc, 152a.

Defines: LECV 41bc, 42a, 55, 56a, 57, 58, 59, 70b, 72, 145c, 146abc, 147abc, 148abc, 149abc, 150abc, 151abc, 152a.

⟨ *C_get_P* 145c ⟩ ≡

```
int C_get_P
(
  ⟨ R_LECV Input 145b ⟩
) {

    return(INTEGER(VECTOR_ELT(LECV, dim_SLOT))[0]);
}
◇
```

Fragment referenced in 145a.

Defines: C_get_P 35a, 42a, 49, 56a, 59, 72, 147c, 148a, 151b.

Uses: dim_SLOT 22b, LECV 145b.

$\langle C_get_Q$ 146a $\rangle \equiv$

```
int C_get_Q
(
   $\langle R$  LECV Input 145b  $\rangle$ 
) {

    return(INTEGER(VECTOR_ELT(LECV, dim_SLOT))[1]);
}
◇
```

Fragment referenced in 145a.

Defines: C_get_Q 35a, 42a, 49, 56a, 72, 147c, 148a, 151b.

Uses: dim_SLOT 22b, LECV 145b.

$\langle C_get_varonly$ 146b $\rangle \equiv$

```
int C_get_varonly
(
   $\langle R$  LECV Input 145b  $\rangle$ 
) {

    return(INTEGER(VECTOR_ELT(LECV, varonly_SLOT))[0]);
}
◇
```

Fragment referenced in 145a.

Defines: C_get_varonly 34, 38b, 42a, 47, 48, 49, 56a, 57, 72, 148a.

Uses: LECV 145b, varonly_SLOT 22b.

$\langle C_get_Xfactor$ 146c $\rangle \equiv$

```
int C_get_Xfactor
(
   $\langle R$  LECV Input 145b  $\rangle$ 
) {

    return(INTEGER(VECTOR_ELT(LECV, Xfactor_SLOT))[0]);
}
◇
```

Fragment referenced in 145a.

Defines: C_get_Xfactor 49.

Uses: LECV 145b, Xfactor_SLOT 22b.

< C_get_LinearStatistic 147a > ≡

```
double* C_get_LinearStatistic
(
  < R LECV Input 145b >
) {

    return(REAL(VECTOR_ELT(LECV, LinearStatistic_SLOT)));
}
◇
```

Fragment referenced in 145a.

Defines: C_get_LinearStatistic 35b, 48, 55, 57, 72, 155a.

Uses: LECV 145b, LinearStatistic_SLOT 22b.

< C_get_Expectation 147b > ≡

```
double* C_get_Expectation
(
  < R LECV Input 145b >
) {

    return(REAL(VECTOR_ELT(LECV, Expectation_SLOT)));
}
◇
```

Fragment referenced in 145a.

Defines: C_get_Expectation 37a, 42a, 46c, 55, 57, 72, 155a.

Uses: Expectation_SLOT 22b, LECV 145b.

< C_get_Variance 147c > ≡

```
double* C_get_Variance
(
  < R LECV Input 145b >
) {

    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 145a.

Defines: C_get_Variance 37c, 38b, 42a, 47, 48, 57, 72, 148a, 155a.

Uses: Covariance_SLOT 22b, C_get_P 145c, C_get_Q 146a, LECV 145b, S 22a, Variance_SLOT 22b.

$\langle C_get_Covariance\ 148a \rangle \equiv$

```
double* C_get_Covariance
(
   $\langle R\ LECV\ Input\ 145b \rangle$ 
) {

  int PQ = C_get_P(LECV) * C_get_Q(LECV);
  if (C_get_varonly(LECV) && PQ > 1)
    error("Cannot extract covariance from variance only object");
  if (C_get_varonly(LECV) && PQ == 1)
    return(C_get_Variance(LECV));
  return(REAL(VECTOR_ELT(LECV, Covariance_SLOT)));
}
◇
```

Fragment referenced in [145a](#).

Defines: [C_get_Covariance 38ab, 42a, 47, 48, 55, 57, 72, 155a](#).

Uses: [Covariance_SLOT 22b](#), [C_get_P 145c](#), [C_get_Q 146a](#), [C_get_Variance 147c](#), [C_get_varonly 146b](#), [LECV 145b](#).

$\langle C_get_ExpectationX\ 148b \rangle \equiv$

```
double* C_get_ExpectationX
(
   $\langle R\ LECV\ Input\ 145b \rangle$ 
) {
  return(REAL(VECTOR_ELT(LECV, ExpectationX_SLOT)));
}
◇
```

Fragment referenced in [145a](#).

Defines: [C_get_ExpectationX 36a, 49, 72](#).

Uses: [ExpectationX_SLOT 22b](#), [LECV 145b](#).

$\langle C_get_ExpectationInfluence\ 148c \rangle \equiv$

```
double* C_get_ExpectationInfluence
(
   $\langle R\ LECV\ Input\ 145b \rangle$ 
) {

  return(REAL(VECTOR_ELT(LECV, ExpectationInfluence_SLOT)));
}
◇
```

Fragment referenced in [145a](#).

Defines: [C_get_ExpectationInfluence 36a, 49, 155a](#).

Uses: [ExpectationInfluence_SLOT 22b](#), [LECV 145b](#).

< C_get_CovarianceInfluence 149a > ≡

```
double* C_get_CovarianceInfluence
(
  < R LECV Input 145b >
) {
    return(REAL(VECTOR_ELT(LECV, CovarianceInfluence_SLOT)));
}
◇
```

Fragment referenced in [145a](#).

Defines: `C_get_CovarianceInfluence` [36a](#), [47](#), [72](#), [155a](#).

Uses: `CovarianceInfluence_SLOT` [22b](#), `LECV` [145b](#).

< C_get_VarianceInfluence 149b > ≡

```
double* C_get_VarianceInfluence
(
  < R LECV Input 145b >
) {
    return(REAL(VECTOR_ELT(LECV, VarianceInfluence_SLOT)));
}
◇
```

Fragment referenced in [145a](#).

Defines: `C_get_VarianceInfluence` [36a](#), [47](#), [72](#), [155a](#).

Uses: `LECV` [145b](#), `VarianceInfluence_SLOT` [22b](#).

< C_get_TableBlock 149c > ≡

```
double* C_get_TableBlock
(
  < R LECV Input 145b >
) {
    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 [145a](#).

Defines: `C_get_TableBlock` [36a](#).

Uses: `block` [28bd](#), `LECV` [145b](#), `TableBlock_SLOT` [22b](#).

$\langle C_get_Sumweights\ 150a \rangle \equiv$

```
double* C_get_Sumweights
(
   $\langle R\ LECV\ Input\ 145b \rangle$ 
) {
  if (VECTOR_ELT(LECV, Sumweights_SLOT) == R_NilValue)
    error("object does not contain sumweights slot");
  return(REAL(VECTOR_ELT(LECV, Sumweights_SLOT)));
}
◇
```

Fragment referenced in 145a.

Defines: C_get_Sumweights 36a, 49.

Uses: LECV 145b, sumweights 27a, Sumweights_SLOT 22b.

$\langle C_get_Table\ 150b \rangle \equiv$

```
double* C_get_Table
(
   $\langle R\ LECV\ Input\ 145b \rangle$ 
) {

  if (LENGTH(LECV) <= Table_SLOT)
    error("Cannot extract table from object");
  return(REAL(VECTOR_ELT(LECV, Table_SLOT)));
}
◇
```

Fragment referenced in 145a.

Defines: C_get_Table 44, 49.

Uses: LECV 145b, Table_SLOT 22b.

$\langle C_get_dimTable\ 150c \rangle \equiv$

```
int* C_get_dimTable
(
   $\langle R\ LECV\ Input\ 145b \rangle$ 
) {

  if (LENGTH(LECV) <= Table_SLOT)
    error("Cannot extract table from object");
  return(INTEGER(getAttrib(VECTOR_ELT(LECV, Table_SLOT),
                           R_DimSymbol)));
}
◇
```

Fragment referenced in 145a.

Defines: C_get_dimTable 49, 151a.

Uses: LECV 145b, Table_SLOT 22b.

$\langle C_get_B \text{ 151a} \rangle \equiv$

```
int C_get_B
(
   $\langle R \text{ LECV Input 145b} \rangle$ 
) {
    if (VECTOR_ELT(LECV, TableBlock_SLOT) != R_NilValue)
        return(LENGTH(VECTOR_ELT(LECV, Sumweights_SLOT)));
    return(C_get_dimTable(LECV)[2]);
}
◇
```

Fragment referenced in [145a](#).

Defines: [C_get_B 35a](#), [49](#), [72](#).

Uses: [C_get_dimTable 150c](#), [LECV 145b](#), [Sumweights_SLOT 22b](#), [TableBlock_SLOT 22b](#).

$\langle C_get_nresample \text{ 151b} \rangle \equiv$

```
R_xlen_t C_get_nresample
(
   $\langle R \text{ LECV Input 145b} \rangle$ 
) {
    int PQ = C_get_P(LECV) * C_get_Q(LECV);
    return(XLENGTH(VECTOR_ELT(LECV, PermutedLinearStatistic_SLOT)) / PQ);
}
◇
```

Fragment referenced in [145a](#).

Defines: [C_get_nresample 42a](#), [55](#), [56a](#), [57](#), [59](#), [72](#).

Uses: [C_get_P 145c](#), [C_get_Q 146a](#), [LECV 145b](#), [PermutedLinearStatistic_SLOT 22b](#).

$\langle C_get_PermutedLinearStatistic \text{ 151c} \rangle \equiv$

```
double* C_get_PermutedLinearStatistic
(
   $\langle R \text{ LECV Input 145b} \rangle$ 
) {
    return(REAL(VECTOR_ELT(LECV, PermutedLinearStatistic_SLOT)));
}
◇
```

Fragment referenced in [145a](#).

Defines: [C_get_PermutedLinearStatistic 42a](#), [55](#), [57](#), [72](#).

Uses: [LECV 145b](#), [PermutedLinearStatistic_SLOT 22b](#).

< C_get_tol 152a > ≡

```
double C_get_tol
(
  < R LECV Input 145b >
) {
    return(REAL(VECTOR_ELT(LECV, tol_SLOT))[0]);
}
◇
```

Fragment referenced in [145a](#).
Defines: `C_get_tol` [42a](#), [55](#), [57](#), [72](#).
Uses: `LECV` [145b](#), `tol_SLOT` [22b](#).

< Memory Input Checks 152b > ≡

```
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 [154](#).
Uses: `B` [28c](#), `P` [25a](#), `Q` [25e](#).

⟨ *Memory Names 153* ⟩ ≡

```
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 154.

Uses: CovarianceInfluence_SLOT 22b, Covariance_SLOT 22b, dim_SLOT 22b, ExpectationInfluence_SLOT 22b, ExpectationX_SLOT 22b, Expectation_SLOT 22b, LinearStatistic_SLOT 22b, PermutedLinearStatistic_SLOT 22b, StandardisedPermutedLinearStatistic_SLOT 22b, Sumweights_SLOT 22b, TableBlock_SLOT 22b, Table_SLOT 22b, tol_SLOT 22b, VarianceInfluence_SLOT 22b, Variance_SLOT 22b, varonly_SLOT 22b, Xfactor_SLOT 22b.

< R_init_LECV 154 > ≡

```
SEXP vo, d, names, tolerance, tmp;
int PQ;

< Memory Input Checks 152b >
PQ = P * Q;
< Memory Names 153 >

/* 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, allocVector(REALSXP, PQ));
} else {
    /* always return variance */
    SET_VECTOR_ELT(ans, Variance_SLOT, allocVector(REALSXP, PQ));
    SET_VECTOR_ELT(ans, Covariance_SLOT,
        allocVector(REALSXP, PQ * (PQ + 1) / 2));
}
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 155a >
```

◇

Fragment referenced in [155b](#), [156](#).

Uses: [B 28c](#), [CovarianceInfluence_SLOT 22b](#), [Covariance_SLOT 22b](#), [dim_SLOT 22b](#), [ExpectationInfluence_SLOT 22b](#), [ExpectationX_SLOT 22b](#), [Expectation_SLOT 22b](#), [LinearStatistic_SLOT 22b](#), [P 25a](#), [PermutedLinearStatistic_SLOT 22b](#), [Q 25e](#), [StandardisedPermutedLinearStatistic_SLOT 22b](#), [Sumweights_SLOT 22b](#), [TableBlock_SLOT 22b](#), [Table_SLOT 22b](#), [tol_SLOT 22b](#), [VarianceInfluence_SLOT 22b](#), [Variance_SLOT 22b](#), [varonly_SLOT 22b](#), [Xfactor_SLOT 22b](#).

⟨ *Initialise Zero 155a* ⟩ ≡

```
/* 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 < PQ * (PQ + 1) / 2; 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 [154](#).

Uses: [C_get_Covariance 148a](#), [C_get_CovarianceInfluence 149a](#), [C_get_Expectation 147b](#),
[C_get_ExpectationInfluence 148c](#), [C_get_LinearStatistic 147a](#), [C_get_Variance 147c](#),
[C_get_VarianceInfluence 149b](#), [Q 25e](#).

⟨ *RC_init_LECV_1d 155b* ⟩ ≡

```
SEXP RC_init_LECV_1d
(
    ⟨ C integer P Input 25a ⟩,
    ⟨ C integer Q Input 25e ⟩,
    int varonly,
    ⟨ C integer B Input 28c ⟩,
    int Xfactor,
    double tol
) {
    SEXP ans;

    ⟨ R_init_LECV 154 ⟩

    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 [145a](#).

Defines: [RC_init_LECV_1d 32c](#).

Uses: [B 28c](#), [Sumweights_SLOT 22b](#), [TableBlock_SLOT 22b](#).

$\langle RC_init_LECV_2d\ 156 \rangle \equiv$

```
SEXP RC_init_LECV_2d
(
   $\langle C\ integer\ P\ Input\ 25a \rangle$ ,
   $\langle C\ integer\ Q\ Input\ 25e \rangle$ ,
  int varonly,
  int Lx,
  int Ly,
   $\langle C\ integer\ B\ Input\ 28c \rangle$ ,
  int Xfactor,
  double tol
) {
  SEXP ans, tabdim, tab;

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

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

   $\langle R\_init\_LECV\ 154 \rangle$ 

  PROTECT(tabdim = allocVector(INTSXP, 3));
  INTEGER(tabdim)[0] = Lx + 1;
  INTEGER(tabdim)[1] = Ly + 1;
  INTEGER(tabdim)[2] = B;
  SET_VECTOR_ELT(ans, Table_SLOT,
                 tab = allocVector(REALSXP,
                                   INTEGER(tabdim)[0] *
                                   INTEGER(tabdim)[1] *
                                   INTEGER(tabdim)[2]));
  dimgets(tab, tabdim);

  UNPROTECT(3);
  return(ans);
}
◇
```

Fragment referenced in [145a](#).
Defines: `RC_init_LECV_2d` [44](#).
Uses: `B` [28c](#), `Table_SLOT` [22b](#).

Chapter 4

Package Infrastructure

"AAA.R" 157a≡

```
< R Header 161a >
.onUnload <- function(libpath)
  library.dynam.unload("libcoin", libpath)
◇
```

"DESCRIPTION" 157b≡

```
Package: libcoin
Title: Linear Test Statistics for Permutation Inference
Date: 2017-12-13
Version: 1.0-1
Authors@R: person("Torsten", "Hothorn", role = c("aut", "cre"),
                  email = "Torsten.Hothorn@R-project.org")
Description: Basic infrastructure for linear test statistics and permutation
             inference in the framework of Strasser and Weber (1999) <http://epub.wu.ac.at/102/>.
             This package must not be used by end-users. CRAN package 'coin' implements all
             user interfaces and is ready to be used by anyone.
Depends: R (>= 3.4.0)
Suggests: coin
Imports: stats, mvtnorm
LinkingTo: mvtnorm
NeedsCompilation: yes
License: GPL-2
◇
```

"NAMESPACE" 157c≡

```
useDynLib(libcoin, .registration = TRUE)

importFrom("stats", complete.cases, vcov)
importFrom("mvtnorm", GenzBretz)

export(LinStatExpCov, doTest, ctabs, "lmult")
S3method("vcov", "LinStatExpCov")
◇
```


Add flag `-g` to `PKG_CFLAGS` for `perf` profiling (this is not portable).

```
"Makevars" 158a≡
```

```
PKG_CFLAGS=$(C_VISIBILITY)
PKG_LIBS = $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS)
◇
```

```
"libcoin-win.def" 158b≡
```

```
LIBRARY libcoin.dll
EXPORTS
  R_init_libcoin
◇
```

Other packages can link against **libcoin**. A small example package is contained in `libcoin/inst/C_API_example`.

"libcoin-init.c" 159≡

```
< C Header 161b >
#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_kronecker, 2),
    {NULL, NULL, 0}
};
◇
```

File defined by 159, 160.

Uses: R_colSums 112a, R_CovarianceInfluence 85b, R_CovarianceX 90a, R_ExpectationCovarianceStatistic 32c, R_ExpectationCovarianceStatistic_2d 44, R_ExpectationInfluence 83b, R_ExpectationX 87a, R_KronSums 98a, R_KronSums_Permutation 107b, R_OneTableSums 117a, R_order_subset_wrt_block 131b, R_PermutedLinearStatistic 40, R_PermutedLinearStatistic_2d 51, R_Sums 93b, R_ThreeTableSums 126b, R_TwoTableSums 121b.

"libcoin-init.c" 160≡

```
< C Header 161b >
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_kronecker);
}
◇
```

File defined by 159, 160.

Uses: R_colSums 112a, R_CovarianceInfluence 85b, R_CovarianceX 90a, R_ExpectationCovarianceStatistic 32c, R_ExpectationCovarianceStatistic_2d 44, R_ExpectationInfluence 83b, R_ExpectationX 87a, R_KronSums 98a, R_KronSums_Permutation 107b, R_OneTableSums 117a, R_order_subset_wrt_block 131b, R_PermutedLinearStatistic 40, R_PermutedLinearStatistic_2d 51, R_Sums 93b, R_ThreeTableSums 126b, R_TwoTableSums 121b.

< R Header 161a > ≡

```
### Copyright 2017 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](#), [157a](#).

< C Header 161b > ≡

```
/*

Copyright 2017 Torsten Hothorn

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

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

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

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

DO NOT EDIT THIS FILE

Edit `libcoin.w' and run `nuweb -r libcoin.w'

*/
◇
```

Fragment referenced in [21a](#), [23ac](#), [32a](#), [159](#), [160](#).

Index

Files

"AAA.R" Defined by 157a.
"ctabs.R" Defined by 15b.
"ctabs.Rd" Defined by 20.
"DESCRIPTION" Defined by 157b.
"doTest.Rd" Defined by 19.
"libcoin-init.c" Defined by 159, 160.
"libcoin-win.def" Defined by 158b.
"libcoin.c" Defined by 23c.
"libcoin.h" Defined by 23a.
"libcoin.R" Defined by 3a.
"libcoinAPI.h" Defined by 32a, 38d, 41b, 43b, 50b, 53b, 139b.
"libcoin_internal.h" Defined by 21a.
"LinStatExpCov.Rd" Defined by 18.
"Makevars" Defined by 158a.
"NAMESPACE" Defined by 157c.

Fragments

<2d Covariance 47> Referenced in 48.
<2d Expectation 46c> Referenced in 48.
<2d Memory 49> Referenced in 48.
<2d Total Table 46a> Referenced in 48.
<2d User Interface 42b> Referenced in 24a.
<2d User Interface Inputs 42c> Referenced in 43a, 48.
<C colSums Answer 113c> Referenced in 84a, 112b, 114abc, 115a.
<C colSums Input 113b> Referenced in 112b, 114abc, 115a.
<C Global Variables 22b> Referenced in 21a.
<C Header 161b> Referenced in 21a, 23ac, 32a, 159, 160.
<C integer B Input 28c> Referenced in 28d, 34, 155b, 156.
<C integer block Input 28d> Referenced in 127c.
<C integer N Input 24c> Referenced in 25bc, 34, 40, 44, 79c, 83b, 84a, 85b, 86a, 87ab, 90ab, 93c, 94b, 95abc, 98a, 99b, 107bc, 112a, 117a, 121b, 126b, 131b, 132a, 133ab, 134b.
<C integer Nsubset Input 27c> Referenced in 27d, 40, 44, 83b, 85b, 87a, 90a, 93b, 98a, 107b, 112a, 117a, 121b, 126b, 136ab, 137b.
<C integer P Input 25a> Referenced in 25bc, 34, 79c, 80b, 81, 82, 87b, 90b, 99b, 107c, 155b, 156.
<C integer Q Input 25e> Referenced in 26ab, 34, 80b, 81, 82, 83b, 84a, 85b, 86a, 98a, 107b, 155b, 156.
<C integer subset Input 27e> Referenced in 95bc, 102bc, 105ab, 109a, 110b, 114c, 115a, 119c, 120a, 124ab, 129ab.
<C integer weights Input 26d> Referenced in 95ab, 102ab, 104c, 105a, 114bc, 119bc, 123c, 124a, 128c, 129a.
<C integer x Input 25c> Referenced in 104a, 110ab, 118b, 122c, 127c.
<C integer y Input 26b> Referenced in 122c, 127c.
<C KronSums Answer 99d> Referenced in 79c, 86a, 90b, 98b, 101b, 102abc, 104bc, 105ab, 107c, 108b, 109a, 110ab.
<C KronSums Input 99c> Referenced in 101b, 102abc.
<C Macros 22a> Referenced in 21a.

< C OneTableSums Answer 118c > Referenced in 87b, 117b, 119abc, 120a.
 < C OneTableSums Input 118b > Referenced in 117b, 119abc, 120a.
 < C real subset Input 28a > Referenced in 94b, 95a, 101b, 102a, 104bc, 108b, 110a, 114ab, 119ab, 123bc, 128bc.
 < C real weights Input 26e > Referenced in 94b, 95c, 101b, 102c, 104b, 105b, 114a, 115a, 119a, 120a, 123b, 124b, 128b, 129b.
 < C real x Input 25b > Referenced in 99c, 108b, 109a, 113b, 143.
 < C real y Input 26a > Referenced in 79c, 99bc, 104a, 107c, 108b, 109a, 110ab.
 < C subset range Input 27d > Referenced in 27e, 28a, 79c, 84a, 86a, 87b, 90b, 93c, 98b, 107c, 112b, 117b, 122a, 127a.
 < C sumweights Input 27a > Referenced in 81, 82, 84a, 86a.
 < C ThreeTableSums Answer 128a > Referenced in 127a, 128bc, 129ab.
 < C ThreeTableSums Input 127c > Referenced in 127a, 128bc, 129ab.
 < C TwoTableSums Answer 123a > Referenced in 122a, 123bc, 124ab.
 < C TwoTableSums Input 122c > Referenced in 122a, 123bc, 124ab.
 < C XfactorKronSums Input 104a > Referenced in 104bc, 105ab.
 < 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 46b > Referenced in 48, 51.
 < colSums 111a > Referenced in 24a.
 < colSums Body 115b > Referenced in 114abc, 115a.
 < Compute Covariance Influence 37b > Referenced in 34.
 < Compute Covariance Linear Statistic 38a > Referenced in 34.
 < Compute Expectation Linear Statistic 37a > Referenced in 34.
 < Compute Linear Statistic 35b > Referenced in 34.
 < Compute maxstat Permutation P-Value 75 > Referenced in 71, 76.
 < Compute maxstat Test Statistic 74c > Referenced in 71, 76.
 < Compute maxstat Variance / Covariance Directly 74b > Referenced in 71.
 < Compute maxstat Variance / Covariance from Total Covariance 74a > Referenced in 71.
 < Compute Permuted Linear Statistic 2d 52d > Referenced in 51.
 < Compute Sum of Weights in Block 36b > Referenced in 34.
 < Compute unordered maxstat Linear Statistic and Expectation 78a > Referenced in 76.
 < Compute unordered maxstat Variance / Covariance Directly 79a > Referenced in 76.
 < Compute unordered maxstat Variance / Covariance from Total Covariance 78b > Referenced in 76.
 < Compute Variance from Covariance 38b > Referenced in 34.
 < Compute Variance Linear Statistic 37c > Referenced in 34.
 < continue subset loop 92b > Referenced in 96a, 103, 106, 115b, 120b, 125, 130a.
 < Contrasts 14 > Referenced in 3a.
 < Convert Table to Integer 52a > Referenced in 51.
 < Count Levels 77a > Referenced in 76.
 < ctabs Prototype 15a > Referenced in 15b, 20.
 < C_chisq_pvalue 66a > Referenced in 65b.
 < C_colSums_dweights_dsubset 114a > Referenced in 111a.
 < C_colSums_dweights_isset 115a > Referenced in 111a.
 < C_colSums_iweights_dsubset 114b > Referenced in 111a.
 < C_colSums_iweights_isset 114c > Referenced in 111a.
 < C_CovarianceLinearStatistic 81 > Referenced in 80a.
 < C_doPermute 136b > Referenced in 135b.
 < C_doPermuteBlock 137b > Referenced in 135b.
 < C_ExpectationLinearStatistic 80b > Referenced in 80a.
 < C_get_B 151a > Referenced in 145a.
 < C_get_Covariance 148a > Referenced in 145a.
 < C_get_CovarianceInfluence 149a > Referenced in 145a.
 < C_get_dimTable 150c > Referenced in 145a.
 < C_get_Expectation 147b > Referenced in 145a.
 < C_get_ExpectationInfluence 148c > Referenced in 145a.
 < C_get_ExpectationX 148b > Referenced in 145a.
 < C_get_LinearStatistic 147a > Referenced in 145a.
 < C_get_nresample 151b > Referenced in 145a.
 < C_get_P 145c > Referenced in 145a.

<C_get_PermutatedLinearStatistic 151c> Referenced in 145a.
 <C_get_Q 146a> Referenced in 145a.
 <C_get_Sumweights 150a> Referenced in 145a.
 <C_get_Table 150b> Referenced in 145a.
 <C_get_TableBlock 149c> Referenced in 145a.
 <C_get_tol 152a> Referenced in 145a.
 <C_get_Variance 147c> Referenced in 145a.
 <C_get_VarianceInfluence 149b> Referenced in 145a.
 <C_get_varonly 146b> Referenced in 145a.
 <C_get_Xfactor 146c> Referenced in 145a.
 <C_kronecker 141> Referenced in 138a.
 <C_kronecker_sym 142> Referenced in 138a.
 <C_KronSums_dweights_dsubset 101b> Referenced in 96b.
 <C_KronSums_dweights_isubset 102c> Referenced in 96b.
 <C_KronSums_iweights_dsubset 102a> Referenced in 96b.
 <C_KronSums_iweights_isubset 102b> Referenced in 96b.
 <C_KronSums_Permutation_dsubset 108b> Referenced in 96b.
 <C_KronSums_Permutation_isubset 109a> Referenced in 96b.
 <C_KronSums_sym 143> Referenced in 138a.
 <C_maxabsstand_Covariance 62b> Referenced in 60a.
 <C_maxabsstand_Variance 63a> Referenced in 60a.
 <C_maxstand_Covariance 60b> Referenced in 60a.
 <C_maxstand_Variance 61a> Referenced in 60a.
 <C_maxtype 64> Referenced in 60a.
 <C_maxtype_pvalue 68> Referenced in 65b.
 <C_minstand_Covariance 61b> Referenced in 60a.
 <C_minstand_Variance 62a> Referenced in 60a.
 <C_MPinv_sym 144> Referenced in 138a.
 <C_norm_pvalue 67> Referenced in 65b.
 <C_OneTableSums_dweights_dsubset 119a> Referenced in 116a.
 <C_OneTableSums_dweights_isubset 120a> Referenced in 116a.
 <C_OneTableSums_iweights_dsubset 119b> Referenced in 116a.
 <C_OneTableSums_iweights_isubset 119c> Referenced in 116a.
 <C_ordered_Xfactor 71> Referenced in 60a.
 <C_order_subset_wrt_block 134a> Referenced in 130b.
 <C_Permute 136a> Referenced in 135b.
 <C_PermuteBlock 137a> Referenced in 135b.
 <C_perm_pvalue 66b> Referenced in 65b.
 <C_quadform 63b> Referenced in 60a.
 <C_setup_subset 133a> Referenced in 130b.
 <C_setup_subset_block 133b> Referenced in 130b.
 <C_standardise 65a> Referenced in 60a.
 <C_Sums_dweights_dsubset 94b> Referenced in 92c.
 <C_Sums_dweights_isubset 95c> Referenced in 92c.
 <C_Sums_iweights_dsubset 95a> Referenced in 92c.
 <C_Sums_iweights_isubset 95b> Referenced in 92c.
 <C_ThreeTableSums_dweights_dsubset 128b> Referenced in 116a.
 <C_ThreeTableSums_dweights_isubset 129b> Referenced in 116a.
 <C_ThreeTableSums_iweights_dsubset 128c> Referenced in 116a.
 <C_ThreeTableSums_iweights_isubset 129a> Referenced in 116a.
 <C_TwoTableSums_dweights_dsubset 123b> Referenced in 116a.
 <C_TwoTableSums_dweights_isubset 124b> Referenced in 116a.
 <C_TwoTableSums_iweights_dsubset 123c> Referenced in 116a.
 <C_TwoTableSums_iweights_isubset 124a> Referenced in 116a.
 <C_unordered_Xfactor 76> Referenced in 60a.
 <C_VarianceLinearStatistic 82> Referenced in 80a.
 <C_XfactorKronSums_dweights_dsubset 104b> Referenced in 96b.
 <C_XfactorKronSums_dweights_isubset 105b> Referenced in 96b.

<C_XfactorKronSums_iweights_dsubset 104c> Referenced in 96b.
 <C_XfactorKronSums_iweights_isubset 105a> Referenced in 96b.
 <C_XfactorKronSums_Permutation_dsubset 110a> Referenced in 96b.
 <C_XfactorKronSums_Permutation_isubset 110b> Referenced in 96b.
 <doTest 12> Referenced in 3a.
 <doTest Prototype 11> Referenced in 12, 19.
 <ExpectationCovariances 80a> Referenced in 24a.
 <Extract Dimensions 35a> Referenced in 34.
 <Function Definitions 24a> Referenced in 23c.
 <Function Prototypes 23b> Referenced in 23a.
 <Handle Missing Values 5b> Referenced in 6.
 <init subset loop 91b> Referenced in 96a, 103, 106, 115b, 120b, 125, 130a.
 <Initialise Zero 155a> Referenced in 154.
 <KronSums 96b> Referenced in 24a.
 <KronSums Body 103> Referenced in 101b, 102abc.
 <KronSums Double x 101a> Referenced in 99a.
 <KronSums Integer x 100> Referenced in 99a.
 <KronSums Permutation Body 109b> Referenced in 108b, 109a.
 <Linear Statistic 2d 45b> Referenced in 48, 52d.
 <LinearStatistics 79b> Referenced in 24a.
 <LinStatExpCov 4> Referenced in 3a.
 <LinStatExpCov Prototype 3b> Referenced in 4, 18.
 <LinStatExpCov1d 6> Referenced in 3a.
 <LinStatExpCov2d 8> Referenced in 3a.
 <maxstat Xfactor Variables 70b> Referenced in 71, 76.
 <Memory 145a> Referenced in 24a.
 <Memory Input Checks 152b> Referenced in 154.
 <Memory Names 153> Referenced in 154.
 <MoreUtils 138a> Referenced in 24a.
 <NCOL 138c> Referenced in 138a.
 <NLEVELS 139a> Referenced in 138a.
 <NROW 138b> Referenced in 138a.
 <OneTableSums Body 120b> Referenced in 119abc, 120a.
 <P-Values 65b> Referenced in 24a.
 <Permutations 135b> Referenced in 24a.
 <R block Input 28b> Referenced in 31b, 42c, 50a, 126a, 131a, 132a, 133b, 134a.
 <R blockTable Input 28e> Referenced in 132a, 133b, 134a.
 <R Header 161a> Referenced in 3a, 15b, 157a.
 <R Includes 21b> Referenced in 21a.
 <R LECV Input 145b> Referenced in 54, 56b, 145c, 146abc, 147abc, 148abc, 149abc, 150abc, 151abc, 152a.
 <R N Input 24b> Referenced in 93a.
 <R subset Input 27b> Referenced in 31b, 42c, 79c, 83a, 84a, 85a, 86ac, 87b, 89, 90b, 93ac, 97, 98b, 107ac, 111b, 112b, 116b, 117b, 121a, 122a, 126a, 127a, 131a, 132a, 134ab.
 <R weights Input 26c> Referenced in 31b, 42c, 79c, 83a, 84a, 85a, 86ac, 87b, 89, 90b, 93ac, 97, 98b, 111b, 112b, 116b, 117b, 121a, 122a, 126a, 127a, 131a, 134b.
 <R x Input 24d> Referenced in 31b, 42c, 50a, 79c, 86c, 87b, 89, 90b, 97, 99b, 107ac, 111b, 116b, 121a, 126a.
 <R y Input 25d> Referenced in 31b, 42c, 50a, 83a, 84a, 85a, 86a, 97, 107a, 121a, 126a, 131a.
 <RC KronSums Input 99b> Referenced in 98b.
 <RC_colSums 113a> Referenced in 111a.
 <RC_colSums Prototype 112b> Referenced in 113a.
 <RC_CovarianceInfluence 86b> Referenced in 80a.
 <RC_CovarianceInfluence Prototype 86a> Referenced in 86b.
 <RC_CovarianceX 91a> Referenced in 80a.
 <RC_CovarianceX Prototype 90b> Referenced in 91a.
 <RC_ExpectationCovarianceStatistic 34> Referenced in 31a.
 <RC_ExpectationCovarianceStatistic_2d 48> Referenced in 42b.
 <RC_ExpectationInfluence 84b> Referenced in 80a.
 <RC_ExpectationInfluence Prototype 84a> Referenced in 84b.

<RC_ExpectationX 88> Referenced in 80a.
 <RC_ExpectationX Prototype 87b> Referenced in 88.
 <RC_init_LECV_1d 155b> Referenced in 145a.
 <RC_init_LECV_2d 156> Referenced in 145a.
 <RC_KronSums 99a> Referenced in 96b.
 <RC_KronSums Prototype 98b> Referenced in 99a.
 <RC_KronSums_Permutation 108a> Referenced in 96b.
 <RC_KronSums_Permutation Prototype 107c> Referenced in 108a.
 <RC_LinearStatistic 79d> Referenced in 79b.
 <RC_LinearStatistic Prototype 79c> Referenced in 79d.
 <RC_OneTableSums 118a> Referenced in 116a.
 <RC_OneTableSums Prototype 117b> Referenced in 118a.
 <RC_order_subset_wrt_block 132b> Referenced in 130b.
 <RC_order_subset_wrt_block Prototype 132a> Referenced in 132b.
 <RC_setup_subset 135a> Referenced in 135b.
 <RC_setup_subset Prototype 134b> Referenced in 135a.
 <RC_Sums 94a> Referenced in 92c.
 <RC_Sums Prototype 93c> Referenced in 94a.
 <RC_ThreeTableSums 127b> Referenced in 116a.
 <RC_ThreeTableSums Prototype 127a> Referenced in 127b.
 <RC_TwoTableSums 122b> Referenced in 116a.
 <RC_TwoTableSums Prototype 122a> Referenced in 122b.
 <R_colSums 112a> Referenced in 111a.
 <R_colSums Prototype 111b> Referenced in 23b, 112a.
 <R_CovarianceInfluence 85b> Referenced in 80a.
 <R_CovarianceInfluence Prototype 85a> Referenced in 23b, 85b.
 <R_CovarianceX 90a> Referenced in 80a.
 <R_CovarianceX Prototype 89> Referenced in 23b, 90a.
 <R_ExpectationCovarianceStatistic 32c> Referenced in 31a.
 <R_ExpectationCovarianceStatistic Prototype 32b> Referenced in 23b, 32c.
 <R_ExpectationCovarianceStatistic_2d 44> Referenced in 42b.
 <R_ExpectationCovarianceStatistic_2d Prototype 43a> Referenced in 23b, 44.
 <R_ExpectationInfluence 83b> Referenced in 80a.
 <R_ExpectationInfluence Prototype 83a> Referenced in 23b, 83b.
 <R_ExpectationX 87a> Referenced in 80a.
 <R_ExpectationX Prototype 86c> Referenced in 23b, 87a.
 <R_init_LECV 154> Referenced in 155b, 156.
 <R_kronecker 140b> Referenced in 138a.
 <R_kronecker Prototype 140a> Referenced in 23b, 140b.
 <R_KronSums 98a> Referenced in 96b.
 <R_KronSums Prototype 97> Referenced in 23b, 98a.
 <R_KronSums_Permutation 107b> Referenced in 96b.
 <R_KronSums_Permutation Prototype 107a> Referenced in 23b, 107b.
 <R_MaximallySelectedTest 59> Referenced in 53a.
 <R_MaximallySelectedTest Prototype 58> Referenced in 23b, 59.
 <R_MaximumTest 57> Referenced in 53a.
 <R_MaximumTest Prototype 56b> Referenced in 23b, 57.
 <R_OneTableSums 117a> Referenced in 116a.
 <R_OneTableSums Prototype 116b> Referenced in 23b, 117a.
 <R_order_subset_wrt_block 131b> Referenced in 130b.
 <R_order_subset_wrt_block Prototype 131a> Referenced in 23b, 131b.
 <R_PermutedLinearStatistic 40> Referenced in 31a.
 <R_PermutedLinearStatistic Prototype 38c> Referenced in 23b, 40.
 <R_PermutedLinearStatistic_2d 51> Referenced in 42b.
 <R_PermutedLinearStatistic_2d Prototype 50a> Referenced in 23b, 51.
 <R_QuadraticTest 55> Referenced in 53a.
 <R_QuadraticTest Prototype 54> Referenced in 23b, 55.
 <R_StandardisePermutedLinearStatistic 42a> Referenced in 31a.

<R_StandardisePermutedLinearStatistic Prototype 41c> Referenced in 23b, 42a.
 <R_Sums 93b> Referenced in 92c.
 <R_Sums Prototype 93a> Referenced in 23b, 93b.
 <R_ThreeTableSums 126b> Referenced in 116a.
 <R_ThreeTableSums Prototype 126a> Referenced in 23b, 126b.
 <R_TwoTableSums 121b> Referenced in 116a.
 <R_TwoTableSums Prototype 121a> Referenced in 23b, 121b.
 <Setup Dimensions 33> Referenced in 32c, 40.
 <Setup Dimensions 2d 45a> Referenced in 44, 51.
 <Setup Linear Statistic 41a> Referenced in 40, 51.
 <Setup Log-Factorials 52c> Referenced in 51.
 <Setup maxstat Memory 73> Referenced in 71, 76.
 <Setup maxstat Variables 72> Referenced in 71, 76.
 <Setup Memory and Subsets in Blocks 36a> Referenced in 34.
 <Setup mvtnorm Correlation 70a> Referenced in 68.
 <Setup mvtnorm Memory 69> Referenced in 68.
 <Setup Test Memory 56a> Referenced in 55, 57.
 <Setup unordered maxstat Contrasts 77b> Referenced in 76.
 <Setup Working Memory 52b> Referenced in 51.
 <SimpleSums 92c> Referenced in 24a.
 <start subset loop 92a> Referenced in 96a, 103, 106, 115b, 120b, 125, 130a.
 <Sums Body 96a> Referenced in 94b, 95abc.
 <Tables 116a> Referenced in 24a.
 <Test Statistics 60a> Referenced in 24a.
 <Tests 53a> Referenced in 24a.
 <ThreeTableSums Body 130a> Referenced in 128bc, 129ab.
 <TwoTableSums Body 125> Referenced in 123bc, 124ab.
 <User Interface 31a> Referenced in 24a.
 <User Interface Inputs 31b> Referenced in 32b, 34, 38c.
 <Utils 130b> Referenced in 24a.
 <vcov LinStatExpCov 10> Referenced in 3a.
 <XfactorKronSums Body 106> Referenced in 104bc, 105ab.
 <XfactorKronSums Permutation Body 110c> Referenced in 110ab.

Identifiers

B: [28c](#), [32c](#), [33](#), [34](#), [35a](#), [36a](#), [40](#), [44](#), [45a](#), [46a](#), [48](#), [49](#), [51](#), [52b](#), [71](#), [72](#), [76](#), [126b](#), [127b](#), [130a](#), [139b](#), [140ab](#), [141](#), [142](#), [152b](#), [154](#), [155b](#), [156](#).
 block: [3b](#), [4](#), [5a](#), [6](#), [8](#), [15ab](#), [18](#), [20](#), [28b](#), [28d](#), [32ac](#), [33](#), [36ab](#), [38d](#), [40](#), [43b](#), [44](#), [45a](#), [50b](#), [126b](#), [127b](#), [130a](#), [131b](#), [132b](#), [133b](#), [134a](#), [149c](#).
 blockTable: [28e](#), [40](#), [131b](#), [132b](#), [133b](#), [134a](#).
 CovarianceInfluence_SLOt: [22b](#), [149a](#), [153](#), [154](#).
 Covariance_SLOt: [22b](#), [147c](#), [148a](#), [153](#), [154](#).
 C_chisq_pvalue: [55](#), [66a](#).
 C_colSums_dweights_dsubset: [113a](#), [114a](#).
 C_colSums_dweights_isubset: [113a](#), [115a](#).
 C_colSums_ieweights_dsubset: [113a](#), [114b](#).
 C_colSums_ieweights_isubset: [113a](#), [114c](#).
 C_CovarianceLinearStatistic: [38a](#), [47](#), [74b](#), [79a](#), [81](#), [82](#).
 C_doPermute: [40](#), [136b](#).
 C_doPermuteBlock: [40](#), [137b](#).
 C_ExpectationLinearStatistic: [37a](#), [46c](#), [80b](#).
 C_get_B: [35a](#), [49](#), [72](#), [151a](#).
 C_get_Covariance: [38ab](#), [42a](#), [47](#), [48](#), [55](#), [57](#), [72](#), [148a](#), [155a](#).
 C_get_CovarianceInfluence: [36a](#), [47](#), [72](#), [149a](#), [155a](#).
 C_get_dimTable: [49](#), [150c](#), [151a](#).
 C_get_Expectation: [37a](#), [42a](#), [46c](#), [55](#), [57](#), [72](#), [147b](#), [155a](#).

C_get_ExpectationInfluence: 36a, 49, [148c](#), 155a.
C_get_ExpectationX: 36a, 49, 72, [148b](#).
C_get_LinearStatistic: 35b, 48, 55, 57, 72, [147a](#), 155a.
C_get_nresample: 42a, 55, 56a, 57, 59, 72, [151b](#).
C_get_P: 35a, 42a, 49, 56a, 59, 72, [145c](#), 147c, 148a, 151b.
C_get_PermutedLinearStatistic: 42a, 55, 57, 72, [151c](#).
C_get_Q: 35a, 42a, 49, 56a, 72, [146a](#), 147c, 148a, 151b.
C_get_Sumweights: 36a, 49, [150a](#).
C_get_Table: 44, 49, [150b](#).
C_get_TableBlock: 36a, [149c](#).
C_get_tol: 42a, 55, 57, 72, [152a](#).
C_get_Variance: 37c, 38b, 42a, 47, 48, 57, 72, [147c](#), 148a, 155a.
C_get_VarianceInfluence: 36a, 47, 72, [149b](#), 155a.
C_get_varonly: 34, 38b, 42a, 47, 48, 49, 56a, 57, 72, [146b](#), 148a.
C_get_Xfactor: 49, [146c](#).
C_kronecker: 82, 140b, [141](#).
C_kronecker_sym: 81, [142](#).
C_KronSums_dweights_dsubset: 101a, [101b](#).
C_KronSums_dweights_isubset: 101a, [102c](#).
C_KronSums_ieweights_dsubset: 101a, [102a](#).
C_KronSums_ieweights_isubset: 101a, [102b](#).
C_KronSums_Permutation_dsubset: 108a, [108b](#).
C_KronSums_Permutation_isubset: 108a, [109a](#).
C_maxabsstand_Covariance: [62b](#), 64.
C_maxabsstand_Variance: [63a](#), 64.
C_maxstand_Covariance: [60b](#), 64.
C_maxstand_Variance: [61a](#), 64.
C_maxtype: 57, [64](#), [74c](#).
C_maxtype_pvalue: 57, [68](#).
C_minstand_Covariance: [61b](#), 64.
C_minstand_Variance: [62a](#), 64.
C_OneTableSums_dweights_dsubset: 118a, [119a](#).
C_OneTableSums_dweights_isubset: 118a, [120a](#).
C_OneTableSums_ieweights_dsubset: 118a, [119b](#).
C_OneTableSums_ieweights_isubset: 118a, [119c](#).
C_ordered_Xfactor: 37b, 47, 59, [71](#).
C_order_subset_wrt_block: 132b, [134a](#).
C_Permute: [136a](#), 136b, 137a.
C_PermuteBlock: [137a](#), 137b.
C_perm_pvalue: 55, 57, [66b](#), 75.
C_quadform: 55, [63b](#), [74c](#).
C_setup_subset: 132b, [133a](#), 135a.
C_setup_subset_block: 132b, [133b](#).
C_standardise: 42a, [65a](#).
C_Sums_dweights_dsubset: 94a, [94b](#).
C_Sums_dweights_isubset: 94a, [95c](#).
C_Sums_ieweights_dsubset: 94a, [95a](#).
C_Sums_ieweights_isubset: 94a, [95b](#).
C_ThreeTableSums_dweights_dsubset: 127b, [128b](#).
C_ThreeTableSums_dweights_isubset: 127b, [129b](#).
C_ThreeTableSums_ieweights_dsubset: 127b, [128c](#).
C_ThreeTableSums_ieweights_isubset: 127b, [129a](#).
C_TwoTableSums_dweights_dsubset: 122b, [123b](#).
C_TwoTableSums_dweights_isubset: 122b, [124b](#).
C_TwoTableSums_ieweights_dsubset: 122b, [123c](#).
C_TwoTableSums_ieweights_isubset: 122b, [124a](#).
C_unordered_Xfactor: 37b, 59, [76](#).
C_VarianceLinearStatistic: 37c, 47, 74b, 79a, [82](#).

C_XfactorKronSums_dweights_dsubset: 100, [104b](#).
C_XfactorKronSums_dweights_isubset: 100, [105b](#).
C_XfactorKronSums_iweights_dsubset: 100, [104c](#).
C_XfactorKronSums_iweights_isubset: 100, [105a](#).
C_XfactorKronSums_Permutation_dsubset: 108a, [110a](#).
C_XfactorKronSums_Permutation_isubset: 108a, [110b](#).
dim_SLOT: [22b](#), [145c](#), [146a](#), 153, 154.
DoCenter: [22b](#), 79d, 84b, 86b, 88, 91a, 98a, 112a.
DoSymmetric: [22b](#), 79d, 86b, 91a.
DoVarOnly: [22b](#), 37bc, 38a, 47.
ExpectationInfluence_SLOT: [22b](#), [148c](#), 153, 154.
ExpectationX_SLOT: [22b](#), [148b](#), 153, 154.
Expectation_SLOT: [22b](#), [147b](#), 153, 154.
GE: [22a](#), 55, 57.
HAS_WEIGHTS: [26d](#), [26e](#), 96a, 103, 106, 115b, 120b, 125, 130a.
LE: [22a](#), 57.
LECV: 41bc, 42a, 55, 56a, 57, 58, 59, 70b, 72, [145b](#), 145c, 146abc, 147abc, 148abc, 149abc, 150abc, 151abc, 152a.
LinearStatistic_SLOT: [22b](#), [147a](#), 153, 154.
N: 5ab, 6, 8, 15b, [24b](#), [24c](#), 35ab, 36ab, 37abc, 38a, 40, 44, 68, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92a, 93b, 94a, 96a, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 132b, 133ab, 134a, 135a, 143.
NCOL: 12, 33, 45a, 83b, 85b, 98a, 107b, 112a, 131b, [138c](#), 140b.
NLEVELS: 33, 45a, 117a, 121b, 126b, 131b, [139a](#).
NROW: 6, 8, 9ab, 14, 35a, 40, 46c, 47, [138b](#), 139a, 140b.
Nsubset: [27c](#), 36b, 40, 44, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92ab, 93b, 94a, 96a, 98a, 100, 101a, 107b, 108a, 109b, 110c, 112a, 113a, 117a, 118a, 121b, 122b, 126b, 127b, 136ab, 137b.
offset: [27d](#), 34, 36b, 37abc, 38a, 79d, 84b, 86b, 88, 91ab, 94a, 100, 101a, 108a, 109b, 110c, 113a, 118a, 122b, 127b.
Offset0: [22b](#), 35b, 36a, 40, 44, 46c, 47, 83b, 85b, 87a, 90a, 93b, 98a, 107b, 112a, 117a, 121b, 126b, 131b, 135a.
P: 14, [25a](#), 32c, 33, 35ab, 36a, 37ac, 38ab, 40, 44, 45ab, 46c, 47, 48, 49, 51, 55, 56a, 57, 59, 71, 72, 73, 74a, 76, 77ab, 78ab, 79d, 80b, 81, 82, 86c, 87a, 88, 89, 90a, 91a, 97, 98a, 100, 101a, 103, 106, 107ab, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 120b, 121b, 122b, 125, 126b, 127b, 130a, 143, 152b, 154.
PermutedLinearStatistic_SLOT: [22b](#), 151bc, 153, 154.
Power1: [22b](#), 84b, 88, 112a.
Power2: [22b](#), 86b, 91a.
Q: 14, [25e](#), 32c, 33, 35ab, 37abc, 38ab, 40, 44, 45ab, 46c, 47, 48, 49, 51, 55, 56a, 57, 71, 72, 73, 74abc, 76, 78ab, 79ad, 80b, 81, 82, 83b, 84b, 85b, 86b, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 121b, 122b, 125, 126b, 127b, 130a, 152b, 154, 155a.
RC_colSums: 84b, 86b, 88, 91a, 112ab, [113a](#).
RC_CovarianceInfluence: 37b, 47, 85b, 86a, [86b](#).
RC_CovarianceX: 37c, 38a, 47, 90ab, [91a](#).
RC_ExpectationCovarianceStatistic: 32c, [34](#), [48](#).
RC_ExpectationInfluence: 37a, 46c, 83b, 84a, [84b](#).
RC_ExpectationX: 37a, 46c, 87ab, [88](#).
RC_init_LECV_1d: 32c, [155b](#).
RC_init_LECV_2d: 44, [156](#).
RC_KronSums: 79d, 86b, 91a, 98ab, [99a](#).
RC_KronSums_Permutation: 40, 107bc, [108a](#).
RC_LinearStatistic: 35b, 79c, [79d](#).
RC_OneTableSums: 36a, 40, 88, 117ab, [118a](#).
RC_order_subset_wrt_block: 36a, 40, 131b, 132a, [132b](#).
RC_setup_subset: 40, 134b, [135a](#).
RC_Sums: 36ab, 83b, 85b, 93bc, [94a](#), 131b, 135a.
RC_ThreeTableSums: 44, 126b, 127a, [127b](#).
RC_TwoTableSums: 44, 121b, 122a, [122b](#).
R_colSums: 111b, [112a](#), 159, 160.
R_CovarianceInfluence: 85a, [85b](#), 159, 160.
R_CovarianceX: 89, [90a](#), 159, 160.
R_ExpectationCovarianceStatistic: 6, 32ab, [32c](#), 159, 160.

R_ExpectationCovarianceStatistic_2d: 8, 43ab, [44](#), 159, 160.
R_ExpectationInfluence: 83a, [83b](#), 85b, 159, 160.
R_ExpectationX: 86c, [87a](#), 90a, 159, 160.
R_KronSums: 97, [98a](#), 159, 160.
R_KronSums_Permutation: 107a, [107b](#), 159, 160.
R_OneTableSums: 15b, 116b, [117a](#), 131b, 159, 160.
R_order_subset_wrt_block: 131a, [131b](#), 159, 160.
R_PermutatedLinearStatistic: 6, 38cd, [40](#), 159, 160.
R_PermutatedLinearStatistic_2d: 8, 50ab, [51](#), 52a, 159, 160.
R_Sums: 93a, [93b](#), 159, 160.
R_ThreeTableSums: 15b, 126a, [126b](#), 159, 160.
R_TwoTableSums: 15b, 121a, [121b](#), 159, 160.
S: [22a](#), 37b, 38b, 47, 48, 60b, 61b, 62b, 63b, 65a, 69, 70a, 74a, 78b, 91a, 103, 142, 143, 144, 147c.
StandardisedPermutatedLinearStatistic_SLOT: [22b](#), 153, 154.
subset: 3b, 4, 5ab, 6, 8, 15ab, 18, 20, [27b](#), [27e](#), [28a](#), 32ac, 34, 35b, 36ab, 38d, 40, 43b, 44, 46c, 47, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 92b, 93b, 94a, 98a, 100, 101a, 107b, 108a, 109b, 110c, 112a, 113a, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 132b, 134a, 135a, 136ab, 137ab.
sumweights: [27a](#), 34, 36ab, 37abc, 38a, 46bc, 47, 49, 51, 52bd, 72, 73, 74b, 79a, 81, 82, 83b, 84b, 85b, 86b, 135a, 150a.
Sumweights_SLOT: [22b](#), 150a, 151a, 153, 154, 155b.
TableBlock_SLOT: [22b](#), 36a, 149c, 151a, 153, 154, 155b.
Table_SLOT: [22b](#), 150bc, 153, 154, 156.
tol_SLOT: [22b](#), 152a, 153, 154.
VarianceInfluence_SLOT: [22b](#), 149b, 153, 154.
Variance_SLOT: [22b](#), 147c, 153, 154.
varonly_SLOT: [22b](#), 146b, 153, 154.
weights: 3b, 4, 5a, 6, 8, 15ab, 18, 20, [26c](#), 26de, 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 52a, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91ab, 93b, 94a, 98a, 100, 101a, 112a, 113a, 117a, 118a, 121b, 122b, 126b, 127b, 131b, 135a.
weights,: 4, 6, 8, 15b, 20, [26d](#), [26e](#), 32ac, 35b, 36b, 37abc, 38ad, 40, 43b, 44, 79d, 83b, 84b, 85b, 86b, 87a, 88, 90a, 91a, 93b, 98a, 112a, 117a, 121b, 126b, 131b, 135a.
x: 8, 14, 18, 22a, [24d](#), [25b](#), [25c](#), 32ac, 33, 35ab, 37ac, 38ad, 40, 43b, 44, 45ab, 46c, 47, 50b, 51, 79d, 87a, 88, 90a, 91a, 98a, 99a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 112a, 113a, 115b, 117a, 118a, 120b, 121b, 122b, 125, 126b, 127b, 130a, 138bc, 139a, 143, 144.
Xfactor_SLOT: [22b](#), 146c, 153, 154.
y: 14, 22a, [25d](#), [26a](#), [26b](#), 32ac, 33, 35b, 37ab, 38d, 40, 43b, 44, 45ab, 46c, 47, 50b, 79d, 83b, 84b, 85b, 86b, 98a, 100, 101a, 103, 106, 107b, 108a, 109b, 110c, 121b, 122b, 125, 126b, 127b, 130a, 131b, 141, 142.

Bibliography

Helmut Strasser and Christian Weber. On the asymptotic theory of permutation statistics. *Mathematical Methods of Statistics*, 8:220–250, 1999. preprint available from http://epub.wu-wien.ac.at/dyn/openURL?id=oai:epub.wu-wien.ac.at:epub-wu-01_94c. 1