Documentation of the `C` functions
Weighted BACON algorithms

Tobias Schoch

University of Applied Sciences Northwestern Switzerland FHNW
School of Business, Riggenbachstrasse 16, CH-4600 Olten
tobias.schoch@fhnw.ch

April 22, 2021, version 0.3

Contents

1 Introduction

In this report, we document the `C` functions underlying the `wbacon` `R` package. Only the following methods are exported:

- `wbacon` (BACON algorithm for multivariate outlier detection)
- `wbacon_reg` (BACON algorithm for robust linear regression)
- `wquantile` (weighted quantile)

All other functions are not exported, hence, they are not callable from `R`. The methodological details of the functions are discussed in the document "methods.pdf" (see package folder `doc`).

For ease of referencing, we use the following abbreviations.

LAPACK: Anderson, E., Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenhaum, S. Hammerling, A. McKenney, and D. Sorensen (1999). *LAPACK Users' Guide*, 3rd ed., Philadelphia: Society for Industrial and Applied Mathematics (SIAM).

BLAS: Blackford, L. S., A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammerling, G. Henry, M. Heroux, L. Kaufman, and A. Lumsdaine (2002). An updated set of basic linear algebra subprograms (BLAS), *ACM Transactions on Mathematical Software*, 28, 135–151.

OpenMP: OpenMP Architecture Review Board (2018). *OpenMP Application Program Interface Version 5.0*, URL https://https://www.openmp.org.

2 Exported functions

`wbacon` *Weighted BACON algorithm for multivariate outlier detection*

Description

The function implements a weighted variant of Algorithm 3 of Billor et al. (2000). It calls a weighted variant of Algorithm 2 of Billor et al. (2000) to initialize the subset (see `initial_subset`).

Usage

```
void wbacon(double *x, double *w, double *center, double *scatter, double *dist,
    int *n, int *p, double *alpha, int *subset, double *cutoff, int *maxiter,
    int *verbose, int *version2, int *collect, int *success, int *threads)
```

Arguments

`x`	data, `double array[n, p]`.
`w`	sampling weights, `double array[n]`.
`center`	center, `double array[p]`.
`scatter`	scatter matrix, `double array[p, p]`.
`dist`	distances, `double array[n]`.
`n, p`	dimensions, `[int]`.
`alpha`	tuning constant, `[double]`, it defines the $1 - \alpha$ quantile of the chi-squared distribution.
`subset`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`cutoff`	cutoff threshold, `[double]`, i.e. $1 - \alpha$ quantile of the chi-squared distribution.
`maxiter`	maximum number of iterations, `[int]`.
`verbose`	toggle, `[int]`, `1`: verbose (i.e., the function prints detailed information to the console), `0`: quiet.
`version2`	toggle, `[int]`, defines the method to construct the initial subset: `1`: "Version 2" of Billor et al. (2000) is used; `0`: "Version 1" is used.
`collect`	size of the initial basic subset, `[int]`.
`success`	indicator, `[int]`, `1`: algorithm converged, `0`: failure of convergence.
`threads`	requested number of threads (OpenMP), `[int]`.

Details

The `subset` is implemented as an `int array[n]`. Elements in the subset are coded `1`; otherwise `0`. The function makes a copy, `w_cpy`, of the array `w` with sampling weights. This copy is used in the computations (e.g., `weightedmean`) and is modified such that `w_cpy[i] = 0.0` if `subset[i] == 0`.

See `methods.pdf` for more details.

Dependencies

internal: `initial_location`, `initial_subset`, `mahalanobis`, `cutoffval`, and `wbacon_error`

external: `Rmath.h:qchisq`

Value

On return, the following slots are overwritten:

`center`	estimated weighted coordinate-wise center
`scatter`	estimated lower triangular matrix of the weighted scatter matrix
`dist`	Mahalanobis distance
`subset0`	subset of outlier-free observations
`cutoff`	$1 - \alpha$ quantile of the chi-squared distribution
`maxiter`	number of iteration required
`success`	convergence or failure of convergence

References

Billor N., Hadi A.S., Vellemann P.F. (2000). BACON: Blocked Adaptive Computationally efficient Outlier Nominators. *Computational Statistics and Data Analysis* 34, pp. 279-298.

Béguin C., Hulliger B. (2008). The BACON-EEM Algorithm for Multivariate Outlier Detection in Incomplete Survey Data. *Survey Methodology* 34, pp. 91-103.

`wbacon_reg` *Weighted BACON algorithm for robust linear regression*

Description

The function implements a weighted variant of the Algorithms 4 and 5 of Billor et al. (2000).

Usage

```
void wbacon_reg(double *x, double *y, double *w, double *resid, double *beta,
    int *subset0, double *dist, int *n, int *p, int *m, int *verbose,
    int *success, int *collect, double *alpha, int *maxiter, int *original,
    int *threads)
```

Arguments

`x`	design matrix, `double array[n, p]`.
`y`	response, `double array[n]`.
`w`	sampling weights, `double array[n]`.
`resid`	reiduals, `double array[n]`.
`subset0`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`dist`	distances/ tis, `double array[n]`.
`n, p`	dimensions, `[int]`.
`m`	size of subset, `[int]`.
`verbose`	toggle, `[int]`, `1`: verbose (i.e., the function prints detailed information to the console), `0`: quiet.
`success`	`1`: successful termination; `0`: error, did not converge, `[int]`.
`collect`	size of the initial basic subset, `[int]`.
`alpha`	cutoff threshold, `[double]`, i.e. $1 - \alpha$ quantile of the Student t-distribution.
`maxiter`	maximum number of iterations, `[int]`.
`original`	`1`: the subset of the `m = collect * p` smallest observations (small w.r.t. to the Mahalanobis distances) is taken from the subset generated by Algorithm 3 as the basic subset for regression [this is the original method of Billor et al. (2000)]; otherwise (i.e., when `0`) the subset that results from Algorithm 3 of Billor et al. (2000) is taken to be the basic subset for regression, `[int]`.
`threads`	requested number of threads (OpenMP), `[int]`.

Details

The regression is computed in two steps. First, we call the weighted BACON algorithm for multivariate outlier detection (Algorithm 3, see `wbacon`) on the design matrix `x` (Note: the regression intercept, if there is one, must be dropped). As a result, we obtain `subset` and `m`, which are then used as an input to `wbacon_reg`.

The function `wbacon_reg` calls `initial_reg` to initialize the regression. Then, it calls `algorithm_4` and `algorithm_5`.

See `methods.pdf` for more details.

Dependencies

`initial_reg`, `algorithm_4`, and `algorithm_5`

Value

On return, the following slots are overwritten:

`beta`	regression coefficients
`resid`	residuals
`dist`	distances/ tis

`subset0`	subset of outlier-free observations
`maxiter`	number of iteration required
`success`	convergence or failure of convergence
`x`	is overwritten with the QR factorization as returned by `LAPACK: dgels`, respectively, `LAPACK: dgeqrf`

References

Billor N., Hadi A.S., Vellemann P.F. (2000). BACON: Blocked Adaptive Computationally efficient Outlier Nominators. *Computational Statistics and Data Analysis* 34, pp. 279-298.

`wquantile` *Weighted quantile*

Description

Weighted quantile.

Usage

```
void wquantile(double *array, double *weights, int *n, double *prob,
    double *result)
```

Arguments

`array`	data, `double array[n]`.
`weights`	sampling weights, `double array[n]`.
`n`	dimension, `int`.
`prob`	probability that defines the quantile, `double`, such that $0 \leq$`prob`≤ 1.
`result`	quantile, `double`.

Details

- The function is based on a weighted version of the Select (FIND, quickselect) algorithm of C.A.R. Hoare with the Bentley and McIlroy (1993) 3-way partitioning scheme. For very small arrays, we use insertion sort.
- For equal weighting, i.e. when all elements in `weights` are equal, wquantile computes quantiles of type 2 in Hyndman and Fan (1996).
- (Weighted) Select (and Quicksort) is efficient for large arrays. But its overhead can be severe for small arrays; hence, we use insertion sort for small arrays; cf. Bentley and McIlroy (1993). The size threshold below which insertion sort is used can be specified by setting the macro `_n_quickselect` at compile time; see Sect. 7.

See `methods.pdf` for more details.

Dependency

`wquantile_noalloc`

Value

On return, `result` is overwritten with the weighted quantile.

References

Bentley, J.L. and D.M. McIlroy (1993). Engineering a Sort Function, *Software - Practice and Experience* 23, pp. 1249-1265.

Hyndman, R.J. and Y. Fan (1996). Sample Quantiles in Statistical Packages, *The American Statistician 50*, pp. 361-365.

3 Error handling [wbacon_error.c]

Error handling refers to the functions that operate on matrices, and which may fail (e.g., because of rank deficiency). These functions return a value of typedef enum wbacon_error_type. The function wbacon_error can be called to return a human readable error message.

---

wbacon_error_type   *Error type* [typedef enum]

WBACON_ERROR_OK
: no error.

WBACON_ERROR_RANK_DEFICIENT
: matrix is rank deficient.

WBACON_ERROR_NOT_POSITIVE_DEFINITE
: matrix is not positive definite.

WBACON_ERROR_TRIANG_MAT_SINGULAR
: triangular matrix is singular.

WBACON_ERROR_CONVERGENCE_FAILURE
: the algorithm did not converge

[WBACON_ERROR_COUNT]
: error count. This is not an actual error; it is used for internal purposes.

---

wbacon_error   *Human readable error string*

Description

Returns a human readable error string.

Usage

```
const char* wbacon_error(wbacon_error_type err)
```

Arguments

err   error of typedef enum [wbacon_error_type].

Value

Returns a string with a human readable error message.

4 wBACON [wbacon.c]

To offer functions with a clean interface, most of the functions use the typedef struct wbdata and workarray.

wbdata	*Data* [typedef struct]

n	dimension.
p	dimension.
x	pointer to data, double array[n,p].
w	pointer to weight, double array[n].
dist	pointer to distance, double array[n].

workarray	*Work arrays* [typedef struct]

iarray	pointer to work array, int array[n] .
work_n	pointer to work array, double array[n].
work_np	pointer to work array, double array[n, p].
work_pp	pointer to work array, double array[pp].
work_2n	pointer to work array, double array[2n].

Internal functions

initial_location	*Internal function*

Description

Computes the initial location: either version “v1” or “v2” or Billor et al. (2000).

Usage

```
static wbacon_error_type initial_location(wbdata *dat, workarray *work,
    double* restrict select_weight, double* restrict center,
    double* restrict scatter, int* version2)
```

Arguments

dat	data, typedef struct wbdata.
work	work array, typedef struct workarray.
select_weight	weight that indicates membership of an observation in the sample (=1.0), otherwise 0.0, array[n].
center	center, double array[p].
scatter	scatter matrix, double array[p, p].
version2	toggle, [int], defines the method to construct the initial subset: 1: “Version 2” of Billor et al. (2000) is used; 0: “Version 1” is used.

Dependency

wquantile_noalloc, euclidean_norm2, and mahalanobis

Value

The function returns a wbacon_error_type: the return value is either WBACON_ERROR_OK (i.e., no error) or the error handed over by mahalanobis.

See methods.pdf for the details.

On return, the following slots are overwritten:

center

scatter

initial_subset *Internal function*

Description

Computes the initial subset. This is a weighted variant of Algorithm 2 of Billor et al. (2000).

Usage

```
static wbacon_error_type initial_subset(wbdata *dat, workarray *work,
    double* restrict select_weight, double* restrict center,
    double* restrict scatter, int* restrict subset,
    int* restrict subsetsize, int *verbose, int *collect)
```

Arguments

`dat`	data, typedef struct `wbdata`.
`work`	work array, typedef struct `workarray`.
`select_weight`	weight that indicates membership of an observation in the sample (=1.0), otherwise 0.0, `array[n]`.
`center`	center, `double array[p]`.
`scatter`	scatter matrix, `double array[p, p]`.
`subset`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`subsetsize`	size of `subset`, `[int]`.
`verbose`	toggle, `[int]`, `1`: verbose (i.e., the function prints detailed information to the console), `0`: quiet.
`collect`	size of the initial basic subset, `[int]`.

Dependency

`scatter_w`

Value

The function returns a `wbacon_error_type`: the return value is either `WBACON_ERROR_OK` (i.e., no error) or the error handed over by `check_matrix_fullrank`.

On return, the following slots are overwritten:

`dat->w`	elements in the initial subset have $w_i = 1$, else $w_i = 0$
`subset`	subset
`subsetsize`	size of the subset

mahalanobis	*Internal function*

Description

Computes the Mahalanobis distance of the x_i's; see `methods.pdf` for the details.

Usage

```
static inline wbacon_error_type mahalanobis(wbdata *dat, workarray *work,
    double* restrict select_weight, double* restrict center,
    double* restrict scatter)
```

Arguments

`dat`	data, typedef struct `wbdata`.
`work`	work array, typedef struct `workarray`.
`select_weight`	weight that indicates membership of an observation in the sample (=1.0), otherwise 0.0, `array[n]`.
`center`	center, `double array[p]`.
`scatter`	scatter matrix, `double array[p, p]`.

Details

The function's loop over the columns of the data matrix is parallelized using the OpenMP preprocessor directive

```
#pragma omp parallel for if(n > OMP_MIN_SIZE)
```

where `OMP_MIN_SIZE` is of size 100 000 , and `n` and is the number of rows. The inner loop over the `n` rows is equipped with the directive `#pragma omp simd` to tell the compiler that we demand SIMD vectorization.

Dependencies

internal: `mean_scatter_w`

external: `LAPACK:dtrsm` and `LAPACK:dpotrf`

Value

The function returns a `wbacon_error_type`: the return value is either `WBACON_ERROR_OK` (i.e., no error) or `WBACON_ERROR_RANK_DEFICIENT`.

On return, `dat->dist` is overwritten with the Mahalanobis distance.

scatter_w	*Internal function*

Description

Computes the weighted scatter matrix.

Usage

```
static inline void scatter_w(wbdata *dat, double* restrict work,
    double* restrict select_weight, double* restrict center,
    double* restrict scatter)
```

Arguments

`dat`	data, typedef struct `wbdata`.
`work_np`	work array, `double array[n, p]`.
`select_weight`	weight that indicates membership of an observation in the sample (=1.0), otherwise 0.0, `array[n]`.
`center`	center, `double array[p]`.
`scatter`	scatter matrix, `double array[p, p]`.

Details

The weighted `scatter` matrix is computed without (re-) computing the `center`.

The function's loop over the columns of the data matrix is parallelized using the OpenMP preprocessor directive

```
#pragma omp parallel for if(n > OMP_MIN_SIZE)
```

where `OMP_MIN_SIZE` is of size 100 000 and `n` is the number of rows. The inner loop over the `n` rows is equipped with the directive `#pragma omp simd` to tell the compiler that we demand SIMD vectorization.

Dependency

`BLAS:dsyrk`

Value

On return, `scatter` is overwritten with the lower triangular matrix of the weighted scatter matrix.

mean_scatter_w *Internal function*

Description

Computes the weighted scatter matrix.

Usage

```
static inline void mean_scatter_w(wbdata *dat, double* restrict select_weight,
    double* restrict work_n, double* restrict work_np, double* restrict center,
    double* restrict scatter)
```

Arguments

dat	data, typedef struct wbdata.
select_weight	weight that indicates membership of an observation in the sample (=1.0), otherwise 0.0, array[n].
work_n	work array, double array[n].
work_np	work array, double array[n, p].
center	center, double array[p].
scatter	scatter matrix, double array[p, p].

Details

The function's loop over the columns of the data matrix is parallelized using the OpenMP preprocessor directive

```
#pragma omp parallel for if(n > OMP_MIN_SIZE)
```

where `OMP_MIN_SIZE` is of size 100 000 and `n` is the number of rows. The inner loop over the `n` rows is equipped with the directive `#pragma omp simd` to tell the compiler that we demand SIMD vectorization.

Dependency

BLAS:dsyrk

Value

On return, `scatter` and `mean` are overwritten with, respectively, the lower triangular matrix of the weighted scatter matrix and the weighted coordinate-wise mean.

`euclidean_norm2` *Internal function*

Description

Computes the squared Euclidean norm $\|\boldsymbol{x} - \boldsymbol{c}\|_2^2$, where $\boldsymbol{c}$ denotes the center.

Usage

```
static inline void euclidean_norm2(wbdata *dat, double* restrict work_np,
    double* restrict center)
```

Arguments

`dat`	data, typedef struct `wbdata`.
`work_np`	work array, `double array[n, p]`.
`center`	center, `double array[p]`.

Details

The implementation follows closely S. Hammarling's `dnrm2` function in LAPACK, which uses a onepass algorithm. The algorithm incorporates some form of scaling to prevent underflows. Higham (2002, p. 507 and 571) shows that the return value of the function can only overflow if $\|\boldsymbol{x}\|_2$ exceeds the largest storable double value. See also Hanson and Hopkins (2017).

Value

On return, `dat->dist` is overwritten with the Euclidean norm.

References

Hanson, R.J., and T. Hopkins (2017). Remark on Algorithm 539: A Modern Fortran Reference Implementation for Carefully Computing the Euclidean Norm, *ACM Transactions on Mathematical Software* 44, Article 24.

Higham, N.J. (2002). *Accuracy and Stability of Numerical Algorithms*, 2nd ed., Philadelphia: Society for Industrial and Applied Mathematics.

`check_matrix_fullrank`
Internal function

Description

Check whether the array/ matrix `x` has full rank.

Usage

```
static wbacon_error_type check_matrix_fullrank(double* restrict x, int p)
```

Arguments

`x`	data, `double array[p, p]`.
`p`	dimension, `[int]`.

Details

See `methods.pdf` for the details.

Dependency

`LAPACK:dpotrf`

Value

The function returns a instance of `wbacon_error_type`:

- `WBACON_ERROR_OK` (i.e., no error),
- `WBACON_ERROR_NOT_POSITIVE_DEFINITE` or
- `WBACON_ERROR_RANK_DEFICIENT`.

`cutoffval` *Internal function*

Description

Computes the correction factor used in the determination of the chi-squared quantile criterion; see `methods.pdf` for the details.

Usage

```
static inline double cutoffval(int n, int k, int p)
```

Arguments

`k`	subset size, `[int]`.
`n, p`	dimensions, `[int]`.

Value

Returns the correction factor.

5 wBACON_reg [wbacon_reg.c]

To offer functions with a clean interface, most of the functions use the typedef structs regdata (see regdata.h), estimate, and workarray.

wbdata	*Data* [typedef struct]

n	dimension.
p	dimension.
x	pointer to the design matrix, double array[n,p].
wx	pointer to a copy of the design matrix, double array[n,p].
y	pointer to the response, double array[n].
wy	pointer to a copy of the response, double array[n].
w	pointer to the sampling weights, double array[n].
w_sqrt	pointer to the square root of sampling weights, double array[n].

Note. All slots of the instances of the typedef struct regdata are considered immutable, with one exception: wx and wy will be modified.

estimate	*Estimates* [typedef struct]

sigma	regression scale, double.
weight	pointer to the weights, double array[n].
resid	pointer to the residuals, double array[n].
beta	pointer to the regression coefficient, double array[p].
dist	pointer to the distances, double array[n].
L	pointer to the Cholesky factor, double array[p,p].
xty	pointer to $X^T y$, double array[p].

Note. The slots of the typedef struct estimate reflect the data and parameters of the model fit at the current stage. The instance est of estimate is updated iteratively.

workarray	*Work arrays* [typedef struct]

lwork determines the size of the array dgles_work, [int];

iarray pointer to work array, int array[n].

work_n pointer to work array, double array[n].

work_np pointer to work array, double array[np].

work_pp pointer to work array, double array[pp].

degels_work pointer to double array[lwork]; this array is required by LAPACK:dgels.

Note. The slots of the typedef struct workarray are not (and should not be) used to reference data over different function calls.

Internal functions

initial_reg	*Internal function*

Description

Initializes the least squares estimate.

Usage

```
static wbacon_error_type initial_reg(regdata *dat, workarray *work,
    estimate *est, int* restrict subset, int *m, int *verbose)
```

Arguments

dat regression data, typedef struct regdata.

work work array, typedef struct workarray.

est estimates, typedef struct estimate.

subset subset, int array[n]; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.

m size of the subset, [int].

verbose toggle, [int], 1: verbose (i.e., the function prints detailed information to the console), 0: quiet.

Details

The function's loop over the columns of the data matrix is parallelized using the OpenMP preprocessor directive

```
#pragma omp parallel for if(n > REG_OMP_MIN_SIZE)
```

where `REG_OMP_MIN_SIZE` is of size 1 000 000 and `n` is the number of rows and columns.

See `methods.pdf` for more details.

Dependencies

`fitwls`, `psort_array`, and `compute_ti`

Value

The function returns a `wbacon_error_type`: the return value is either `WBACON_ERROR_OK` (i.e., no error) or `WBACON_ERROR_RANK_DEFICIENT`.

On return, the following slots are overwritten:

`est->sigma`	regression scale
`est->resid`	residuals
`est->beta`	regression coefficients
`est->dist`	distances/ t_i's
`subset`	initial subset
`m`	size of `subset1`

`algorithm_4` *Internal function*

Description

Computes a weighted variant of Algorithm 4 of Billor et al. (2000).

Usage

```
static wbacon_error_type algorithm_4(regdata *dat, workarray *work,
    estimate *est, int* restrict subset0, int* restrict subset1, int *m,
    int *verbose, int *collect)
```

Arguments

`dat`	regression data, typedef struct `regdata`.
`work`	work array, typedef struct `workarray`.
`est`	estimates, typedef struct `estimate`.
`subset0`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.

subset1	subset, int array[n]; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
m	size of the subset, [int].
verbose	toggle, [int], 1: verbose (i.e., the function prints detailed information to the console), 0: quiet.
collect	size of the initial basic subset, [int].

Details

See methods.pdf for more details.

Dependencies

internal: update_chol_xty, cholesky_reg, compute_ti, and select_subset

external: BLAS:dgemv

Value

The function returns a wbacon_error_type either WBACON_ERROR_OK (i.e., no error) or the error handed over by

- update_chol_xty or
- compute_ti.

On return, the following slots are overwritten:

est->sigma	regression scale
est->resid	residuals
est->beta	regression coefficients
est->dist	distances/ t_i's
subset1	final subset of Algorithm 4
m	size of subset1

---

algorithm_5 *Internal function*

Description

Computes a weighted variant of Algorithm 5 of Billor et al. (2000).

Usage

```
static wbacon_error_type algorithm_5(regdata *dat, workarray *work,
    estimate *est, int* restrict subset0, int* restrict subset1,
    double *alpha, int *m, int *maxiter, int *verbose)
```

Arguments

`dat`	regression data, typedef struct `regdata`.
`work`	work array, typedef struct `workarray`.
`est`	estimates, typedef struct `estimate`.
`subset0`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`subset1`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`alpha`	defines the $1-\alpha$ quantile of the Student t-distribution.
`m`	size of the subset, `[int]`.
`maxiter`	maximum number of iterations, `[int]`.
`verbose`	toggle, `[int]`, `1`: verbose (i.e., the function prints detailed information to the console), `0`: quiet.

Details

See `methods.pdf` for more details.

Dependencies

internal: `fitwls` and `compute_ti`

external: `Rmath.h:qt`

Value

The function returns a `wbacon_error_type`: the return value is either the error handed over by `compute_ti` or

- `WBACON_ERROR_OK` (i.e., no error) or
- `WBACON_ERROR_CONVERGENCE_FAILURE` if it does not converge in `maxiter` iterations.

On return, the following slots are overwritten:

`est->sigma`	regression scale
`est->resid`	residuals
`est->beta`	regression coefficients
`est->dist`	distances/ t_i's
`subset1`	final subset of outlier-free data
`m`	size of `subset1`
`maxiter`	number of iterations required

`select_subset` *Internal function*

Description

Selects the smallest `1..m` observations in `x` into the `subset`.

Usage

```
static void select_subset(double* restrict x, int* restrict iarray,
    int* restrict subset, int *m, int *n)
```

Arguments

`x`	data, `double array[n]`.
`iarray`	work array, `int array[n]`.
`subset`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`m`	size of the subset, `[int]`.

Details

The function calls `psort_array` to (partially) sort the elements of `x` in ascending order. Then, the smallest m observations are selected into `subset`.

Value

On return, `subset` is overwritten with the generated subset.

`compute_ti` *Internal function*

Description

Compute the t_i's (`tis`) of Billor et al. (2000, p. 288).

Usage

```
static wbacon_error_type compute_ti(regdata *dat, workarray *work,
    estimate *est, int* restrict subset, int *m, double* restrict tis)
```

Arguments

dat	regression data, typedef struct regdata.
work	work array, typedef struct workarray.
est	estimates, typedef struct estimate.
subset	subset, int array[n]; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
m	size of the subset, [int].
tis	double array[n].

Details

The function calls hat_matrix to compute the diagonal elements of the "hat" matrix and computes the regression scale. Then, it computes the t_i's.

Dependency

hat_matrix

Value

The function's loop over the columns of the hat matrix is parallelized using the OpenMP preprocessor directive

```
#pragma omp parallel for if(n > REG_OMP_MIN_SIZE)
```

where REG_OMP_MIN_SIZE is of size 1 000 000 and n is the number of rows and columns.

The function returns a wbacon_error_type: the return value is either WBACON_ERROR_OK (i.e., no error) or the error handed over by hat_matrix.

On return, tis is overwritten with the computed t_i's.

---

cholesky_reg *Internal function*

Description

Compute the least squares estimate using the Cholesky factor L and the matrix $X^T y$.

Usage

```
static inline void cholesky_reg(double *L, double *x, double *xty,
    double *beta, int *n, int *p)
```

Arguments

L	Cholesky factor, `double array[p,p]`.
x	data, `double array[n]`.
xty	$X^T y$ `double array[p]`.
beta	regression coefficients `double array[p]`.
n	dimension.
p	dimension.

Value

On return, `beta` is overwritten with the updated least squares estimate.

`hat_matrix` *Internal function*

Description

Computes the diagonal elements of the extended “hat” matrix.

Usage

```
static inline wbacon_error_type hat_matrix(regdata *dat, workarray *work,
    double* restrict L, double* restrict hat)
```

Arguments

dat	regression data, typedef struct `regdata`.
work	work array, typedef struct `workarray`.
L	Cholesky factor, `double array[p,p]`.
hat	hat matrx, `double array[n]`.

Details

The diagonal elements of the “hat” matrix are computed for the observations in the subset. For the elements not in the subset, an “extended hat” matrix is computed.

Value

The function returns a `wbacon_error_type`: the return value is either `WBACON_ERROR_OK` (i.e., no error) or `WBACON_ERROR_TRIANG_MAT_SINGULAR` when the triangular matrix is singular.

On return, `hat` is overwritten with the diagonal elements of the “hat” matrix.

Description

The function up- and downdates the Cholesky factor L and the matrix product by comparing the two sets `subset0` and `subset1`.

Usage

```
static wbacon_error_type update_chol_xty(regdata *dat, workarray *work,
    estimate *est, int* restrict subset0, int* restrict subset1, int *verbose)
```

Arguments

`dat`	regression data, typedef struct `regdata`.
`work`	work array, typedef struct `workarray`.
`est`	estimates, typedef struct `estimate`.
`subset0`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`subset1`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`m`	size of the `subset1`, `[int]`.
`verbose`	toggle, `[int]`, `1`: verbose (i.e., the function prints detailed information to the console), `0`: quiet.

Details

The function `update_chol_xty` compares the sets `subset0` and `subset1`. For all elements that are in `subset0` but not in `subset1`, it calls `chol_downdate`. For all elements that are not in `subset0` but in `subset1`, it calls `chol_update`.

Value

The function returns a `wbacon_error_type`: the return value is either `WBACON_ERROR_OK` (i.e., no error) or the error handed over by `chol_downdate`.

On return, `L` and `xty` are overwritten with their updated values.

`chol_update` *Internal function*

Description

Rank-one update of the Cholesky factor.

Usage

```
static inline void chol_update(double* restrict L, double* restrict u, int p)
```

Arguments

`L`	Cholesky factor, `double array[p,p]`.
`u`	rank-one update for `L`, `double array[p]`.
`p`	dimension.

Details

This function computes a one rank-one update of the Cholesky factor.

Value

On return, `L` is overwritten by its updated value.

`chol_downdate` *Internal function*

Description

Rank-one downdate of the Cholesky factor.

Usage

```
static inline wbacon_error_type chol_downdate(double* restrict L,
    double* restrict u, int p)
```

Arguments

`L`	Cholesky factor, `double array[p,p]`.
`u`	rank-one downdate for `L`, `double array[p]`.
`p`	dimension.

Details

This function computes a one rank-one downdate of the Cholesky factor. The attempt to downdate may break down if the Cholesky factor becomes/is not positive definite. In this case, an error is returned.

Value

The function returns a wbacon_error_type: the return value is either WBACON_ERROR_OK (i.e., no error) or WBACON_ERROR_RANK_DEFICIENT.

On return, L is overwritten by its downdated value.

6 Weighted least squares [fitwls.c]

`fitwls`	*Weighted least squares*

Description

Returns the least squares estimate, the matrices Q and R of the QR factorization, the estimate of regression scale, and the residuals of a weighted linear regression.

Usage

```
int fitwls(regdata *dat, estimate* est, int* restrict subset,
    double* restrict work_dgels, int lwork)
```

Arguments

`dat`	regression data, typedef struct `regdata`.
`est`	estimates, typedef struct `estimate`.
`subset`	subset, `int array[n]`; with elements in the set $\{0, 1\}$, where 1 signifies that the element is in the subset.
`work_dgels`	work array, `double array[lwork]`.
`lwork`	dimension of array `work`, `[int]`; if `lwork<1`, the function determines and returns the optimal

Details

The regression coefficients are computed with the `LAPACK:dgels` subroutine using a QR factorization of the weighted design matrix.

The function's loop over the columns of the hat matrix is parallelized using the OpenMP preprocessor directive

```
#pragma omp parallel for if(n > FITLS_OMP_MIN_SIZE)
```

where `FITWLS_OMP_MIN_SIZE` is of size 1 000 000 and `n` is the number of rows.

Dependencies

`LAPACK:dgels` and `BLAS:dgemv`

Value

The function `fitwls` returns its status `info`; if successful, `info=0`; otherwise the computation failed. On return, the following slots of `struct estimate est` are overwritten:

`beta`	regression coefficients
`sigma`	regression scale
`resid`	residuals

and, the following slots of `struct regdata dat` are overwritten:

`wx` the QR factorization as returned by the subroutine `LAPACK:dgeqrf`

7 Weighted quantile [wquantile.c]

The following functions are documented in this section:

- wquantile_noalloc
- wselect0
- some internal functions

The source file wquantile.c defines two macros:

_n_quickselect
: threshold to switch from insertion sort to a weighted variant of the Select (FIND, quickselect) algorithm, default: 40 (i.e., for samples smaller than 40, insertion sort is used).

_n_ninther
: threshold for choosing the pivotal element, default: 50; for samples smaller than 50, the pivot is chosen by the median-of-three; for larger samples, Tukey's ninther is used.

(Weighted) quicksort/ Select(FIND, quickselect) method is efficient for large arrays. But its overhead can be severe for small arrays; hence, we use insertion sort for small arrays; cf. Bentley and McIlroy (1993). We have determined the numerical values by a series of benchmark tests with Google benchmark on an ordinary laptop computer (Intel i7 8th generation).

---

wquantile_noalloc *Weighted quantile without memory allocation*

Description

The same as wquantile but without memory allocation.

Usage

```
void wquantile_noalloc(double *array, double *weights, double *work, int *n,
    double *prob, double *result)
```

Arguments

array	data, double array[n].
weights	sampling weights, double array[n].
workwork	work array, double array[2*n].
n	dimension, [int].
prob	probability that defines the quantile, such that $0 \leq$prob≤ 1, [double].
result	quantile, [double].

Details

See wquantile.

Dependencies

wselect0 and wquant0

Value

On return, `result` is overwritten with the weighted quantile.

`wselect0` *Selection of the k-th largest element (k-th order statistic)*

Description

Returns the k-th largest element (k-th order statistic); sampling weights allowed.

Usage

```
void wselect0(double *array, double *weights, int lo, int hi, int k)
```

Arguments

`array`	data, `double array[lo..hi]`.
`weights`	sampling weights, `double array[n]`.
`lo`	lower boundary of arrays, `[int]`.
`hi`	upper boundary of arrays, `[int]`.
`k`	k-th largest element, such that `lo`≤ `k`≤`hi`, `[int]`.

Details

See wquantile.

Dependency

partition_3way

Value

On return, element `array[k]` is in its final sorted position; `weights` is sorted along with `array`.

insertionselect *Internal function*

Description

Computes the weighted quantile by sorting all elements in `array` in ascending order (using insertion sort). For small arrays, this can be considerably faster than quicksort.

Usage

```
double insertionselect(double *array, double *weights, int lo, int hi,
    double prob)
```

Arguments

`array`	data, `double array[n]`.
`weights`	sampling weights, `double array[n]`.
`lo`	lower boundary of arrays, `[int]`.
`hi`	upper boundary of arrays, `[int]`.
`prob`	probability that defines the quantile, `double`, such that $0 \leq$`prob`≤ 1.

Dependency

swap2

Value

On return, element `array[k]` is in its final sorted position; `weights` is sorted along with `array`.

Internal functions

`wquant0` *Internal function*

Description

Workhorse function that computes the weighted quantile recursively; see wquantile.

Usage

```
void wquant0(double *array, double *weights, double sum_w, int lo, int hi,
    double prob, double *result)
```

Dependencies

`insertionselect` and `partition_3way`

`partition_3way` *Internal function*

Description

3-way partitioning scheme of Bentley and McIlroy's (1993) with weights.

Usage

```
void partition_3way(double *array, double *weights, int lo, int hi, int *i,
    int *j)
```

Dependency

`swap2`

References

Bentley, J.L. and D.M. McIlroy (1993). Engineering a Sort Function, *Software - Practice and Experience* 23, pp. 1249-1265.

choose_pivot	*Internal function*

Description

Choose pivotal element: for arrays of size < `_n_ninther`, the median of three is taken as pivotal element, otherwise Tukey's ninther is used; see e.g. Bentley and McIlroy (1993).

Usage

```
static inline int choose_pivot(double *array, int lo, int hi)
```

Dependency

med3

References

Bentley, J.L. and D.M. McIlroy (1993). Engineering a Sort Function, *Software - Practice and Experience* 23, pp. 1249-1265.

swap2	*Internal function*

Description

Two elements in `array` are swapped (and the corresponding elements in array `weights` are also swapped).

Usage

```
static inline void swap2(double *array, double *weights, int i, int j)
```

med3	*Internal function*

Description

Median-of-three (but without swaps); see e.g. Sedgewick (1997, Chap. 7.5).

Usage

```
static inline double med3(double *array, int i, int j, int k)
```

References

Sedgewick, R. (1997). *Algorithms in C, Parts 1-4, Fundamentals, Data Structures, Sorting, and Searching*, Addison-Wesley Longman Publishing Co., Inc., 3rd ed.

8 Partial sorting [partial_sort.c]

psort_array	*Partially sort an array with index*

Description

Partially sorts array x in ascending order; the accompanying int array (called index) is sorted along with the array.

Usage

```
void psort_array(double *x, int *index, int n, int k)
```

Arguments

x	data, double array[n].
index	index, int array[n]; the array will be overwritten.
n	dimension, [int].
k	value that determines the upper array boundary of x[0..k], where $k \leq n$, [int].

Details

This function is a wrapper for the function partial_sort_with_index.

The function takes care of generating the array index. The elements of this array will set up to be 0..(n -1).

Dependency

partial_sort_with_index

Value

On return, the array x[0..k] is partially sorted in ascending order; the array index[0..k] is sorted along with x[0..k].

Internal functions

Most of the internal functions which are called from psort_array are identical with the internal functions of wselect0. Therefore, we do not document separately.

`partial_sort_with_index`

Internal function

Description

Partially sorts a array `x` in ascending order; the accompanying `int` array (called `index`) is sorted along with the array.

Usage

```
void partial_sort_with_index(double *x, int *index, int *lo, int *hi, int *k)
```

Arguments

`x`	data, `double array[lo..hi]`.
`index`	index, `int array[lo..hi]`; the array will be overwritten.
`lo, hi`	indices, `[int]`, usually `lo = 0` and `hi = n -1`.
`k`	an `[int]` in `lo..hi`; determines the k-th largest element up to which `x` is to be sorted.

Details

The array `index` must be generated by the caller.

Value

On return, the elements `lo..k` in the array `x[lo..hi]` are partially sorted in ascending order; the array `index[lo..k]` is sorted along with `x[lo..k]`.