Package 'QCGWAS'

November 6, 2013

Type Package

Title Quality Control of Genome Wide Association Study results

Version 1.0-7

Date 2013-10-28

Author Peter J. van der Most and Ilja M. Nolte

Maintainer Peter J. van der Most <p.j.van.der.most@umcg.nl>

Depends R (>= 3.0.1)

Description Tools for (automated and manual) quality control of the results of Genome Wide Association Studies

License GPL (>= 3)

R topics documented:

`QCGWAS-package` *Quality Control of Genome Wide Association Study results*

Description

Functions for automated and manual quality control of Genome Wide Association Study results.

Details

Package:	QCGWAS
Type:	Package
Version:	1.0-7
Date:	2013-10-28
License:	GPL (>= 3)

The core of the package is the function `QC_GWAS`. This function carries out an automated quality-control (QC) of a Genome Wide Association Study (GWAS) results file, reporting on the data-distribution, checking the SNPs for missing and invalid data, comparing the alleles and allele-frequency to a reference, and creating QQ and Manhattan plots.

Although the number of arguments in `QC_GWAS` may seem overwhelming, only three of them are required to run a basic QC. The name of the file to be QC'ed should be passed to the `filename` argument; the directory of said file to the `dir_data` argument; and a header-translation table to the `header_translations` argument. The results will be saved in a number of files and graphs in the data directory. For a quick introduction to QCGWAS, read the quick reference guide that can be found in "R\library\QCGWAS\doc".

Author(s)

Peter J. van der Most and Ilja M. Nolte

Maintainer: Peter J. van der Most <p.j.van.der.most@umcg.nl>

References

Van der Most, P.J., Vaez, A., Prins, B.P., Loretto Munoz, M., Snieder, H., Alizadeh, B.Z. & Nolte, I.M. (In preparation). QCGWAS: A flexible R package for automated quality control of Genome-Wide Association results.

See Also

`QC_GWAS`

calc_kurtosis *Skewness and Kurtosis*

Description

Functions for calculating skewness and kurtosis

Usage

```
calc_kurtosis(input,
              FRQ_val = NULL, HWE_val = NULL,
              cal_val = NULL, imp_val = NULL, ...)
calc_skewness(input,
              FRQ_val = NULL, HWE_val = NULL,
              cal_val = NULL, imp_val = NULL, ...)
```

Arguments

`input`	either a vector of effect sizes or a data frame using the standard column names.
`FRQ_val, HWE_val, cal_val, imp_val, ...`	arguments passed to `HQ_filter`.

Details

Kurtosis is a measure of how well a distribution matches a Gaussian distribution. A Gaussian distribution has a kurtosis of `0`. Negative kurtosis indicates a flatter distribution curve, while positive kurtosis indicates a sharper, thinner curve.

Skewness is a measure of distribution asymmetry. A symmetrical distribution has skewness `0`. A positive skewness indicates a long tail towards higher values, while a negative skewness indicates a long tail towards lower values.

Kurtosis is calculated as:

```
sum( (ES - mean(ES))^4) / ((length(ES)-1) * sd(ES)^4 )
```

Skewness is calculated as:

```
sum( (ES - mean(ES))^3) / ((length(ES)-1) * sd(ES)^3 )
```

Value

Respectively the kurtosis and skewness of the input effect-size distribution.

Note

Both functions accept vectors as `input`. If `input` is a data frame, the column names must match the standard names used by `QC_GWAS` (`"EFFECT"` for effect sizes, `"EFF_ALL_FREQ"` for allele frequency, etc.)

See Also

For plotting skewness and kurtosis: `plot_skewness`.

Examples

```
data("gwa_sample")

calc_kurtosis(gwa_sample$EFFECT)
calc_kurtosis(gwa_sample)
calc_kurtosis(gwa_sample$EFF_ALL_FREQ)
calc_kurtosis(gwa_sample,
              FRQ_val = 0.05, cal_val = 0.95,
              filter_NA = FALSE)
```

---

check_P *Checking GWAS p-values*

Description

A simple test to check if the reported p-values in a GWAS results file match the other statistics. This function calculates an expected p-value (from the effect size and standard error) and then correlates it with the actual, reported p-value.

Usage

```
check_P(dataset, HQ_subset,
        plot_correlation = FALSE, plot_if_threshold = FALSE,
        threshold_r = 0.99,
        save_name = "dataset", save_dir = getwd(),
        header_translations,
        use_log = FALSE, dataN = nrow(dataset), ...)
```

Arguments

`dataset`	table with at least three columns: p-value, effect size and standard error.
`HQ_subset`	an *optional* logical or numeric vector indicating the rows in `dataset` that contain high quality SNPs.
`plot_correlation`	logical; should a scatterplot of the reported vs. calculated p-values be made? If `TRUE`, the plot is saved as a .png file.
`plot_if_threshold`	logical; if `TRUE`, the scatterplot is only saved when the correlation between reported and calculated p-values is lower than `threshold_r`.
`threshold_r`	numeric; the correlation threshold for the scatterplot.
`save_name`	character string; the filename, *without* extension, for the scatterplot.
`save_dir`	character string; the directory where the output files are saved. Note that R uses *forward* slash (/) where Windows uses backslash (\).
`header_translations`	translation table for column names See `translate_header` for more information. If the argument is left empty, `dataset` is assumed to use the standard column names used by `QC_GWAS`.
`use_log, dataN`	arguments used by `QC_GWAS`; redundant when `check_P` is used separately.
`...`	arguments passed to `plot`.

Details

check_P calculates the expected p-value by taking the chi-square (1 degree of freedom) of the effect size divided by the standard error squared.

In a typical GWAS dataset, the expected and observed p-values should correlate perfectly. If this isn't the case, the problem either lies in a misidentified column, or the wrong values were used when generating the dataset.

Value

The correlation between expected and reported p-values.

Examples

```
data("gwa_sample")

selected_SNPs <- HQ_filter(data = gwa_sample,
                           FRQ_val = 0.05,
                           cal_val = 0.95,
                           filter_NA = FALSE)
check_P(gwa_sample, HQ_subset = selected_SNPs,
        plot_correlation = TRUE, plot_if_threshold = FALSE,
        save_name = "sample")
```

---

convert_impstatus *Convert imputation-status values to the QCGWAS standard*

Description

Converts imputation-status data to the standard values used by QC_GWAS

Usage

```
convert_impstatus(impstatus,
                  T_strings = c("1", "TRUE", "T"),
                  F_strings = c("0", "FALSE", "F"),
                  NA_strings = c(NA, "NA", ".", "-"),
                  use_log = FALSE, ...)
```

Arguments

impstatus	vector of imputation-status data.
T_strings	character-string(s) indicating imputed data.
F_strings	character-string(s) indicating genotyped data.
NA_strings	character-string(s) indicating missing data.
use_log, ...	arguments used by QC_GWAS; redundant when convert_impstatus is used separately.

Details

This function is used to convert the imputation-status column into the standard format, where 0 is genotyped and 1 is imputed. Untranslated values (i.e. strings that do not appear in any of the string arguments) will trigger a warning message and are set to NA.

Value

Numeric vector with values 0 for genotyped, 1 for imputed and NA for unknown data.

Note

The implementation of this function has changed. Previously, only character data was translated using the string arguments. Since version 1.0-4, the string arguments are used for all data types, so the user can determine the conversion of logical and numeric values as well.

However, this does mean that values 1 and 0 are no longer converted automatically. They must be specified in the string arguments, or else the function will report them as untranslated and converts them to NA. The same applies to TRUE, FALSE and NA. Note the difference between the character string "NA" and the value NA.

Finally, if the imputation-status column contains character strings, the main QC function QC_GWAS requires that all values are translated. If not, QC_GWAS will abort the QC.

Examples

```
status1 <- c(0,0,0,1,1,1,2,NA)
status2 <- c(TRUE, FALSE, TRUE, FALSE, NA)
status3 <- c("imputed", "genotyped", "NA", NA)
status4 <- c(status1, status2, status3)

( outcome1 <- convert_impstatus(status1,
                                T_strings = 1,
                                F_strings = 0,
                                NA_strings = NA) )
# status 1 contains an untranslated value "2", which is
# converted to NA. To avoid the warning message:

( outcome1 <- convert_impstatus(status1,
                                T_strings = 1,
                                F_strings = 0,
                                NA_strings = c(NA, 2)) )


( outcome2 <- convert_impstatus(status2,
                                T_strings = TRUE,
                                F_strings = FALSE,
                                NA_strings = NA) )

( outcome3 <- convert_impstatus(status3,
                                T_strings = "imputed",
                                F_strings = "genotyped",
                                NA_strings = c("NA", NA)) )
# Note that NA_strings includes both the character-string "NA"
# and the value NA. Otherwise, one of the two would go
# "untranslated" and trigger a warning message.


# And to check them all together
( outcome4 <- convert_impstatus(status4,
                        T_strings = c(1, TRUE, "imputed"),
                        F_strings = c(0, FALSE, "genotyped"),
                        NA_strings = c("NA", NA, 2)) )
```

`create_hapmap_reference` *Create an allele-reference file from HapMap data*

Description

This function creates the standard allele reference file, as used by `QC_GWAS` and `match_alleles`, from data publicly available at the website of the international HapMap project (see 'References').

Usage

```
create_hapmap_reference(dir = getwd(),
   download_hapmap = FALSE, download_subset,
   hapmap_files = list.files(path = dir, pattern = "freqs_chr"),
   filename = "allele_reference_HapMap",
   save_txt = TRUE, save_rdata = !save_txt,
   return_reference = FALSE)
```

Arguments

`dir`	character string; the directory of the input and output files. Note that R uses *forward* slash (/) where Windows uses the backslash (\).
`download_hapmap`	logical; if `TRUE`, the required allele-frequency files are downloaded from the HapMap website into `dir`, and then turned into a reference. If `FALSE`, the files specified in `hapmap_files` are used.
`download_subset`	character-string; indicates the population to download for creating the reference. Options are: ASW, CEU, CHB, CHD, GIH, JPT, LWK, MEX, MKK, TSI, YRI.
`hapmap_files`	character vector of the filenames of HapMap frequency-files to be included in the reference. The default option includes all files with the string "freqs_chr" in the filename. (This argument is only used when `download_hapmap` is `FALSE`.)
`filename`	character string; the name of the output file, *without* file-extension.
`save_txt, save_rdata`	logical; should the reference be saved as a tab-delimitated text file and/or an RData file? If saved as RData, the object name `allele_ref_std` is used for the reference table.
`return_reference`	logical; should the function return the reference as it output value?

Details

The function removes SNPs with invalid alleles and with allele frequencies that do not add up to 1. It also removes all instances of duplicate SNPids. If such entries are encountered, a warning is printed in the R console and the entries are saved in a .txt file in the output directory.

Like the `QC_GWAS`, `create_hapmap_reference` codes the X chromosome as `23`, Y as `24`, XY (not available on HapMap website) as `25` and M as `26`.

Both the .RData export and the function return store the alleles as factors rather than character strings.

Value

If `return_reference` is `TRUE`, the function returns the generated reference table. If `FALSE`, it returns an invisible `NULL`.

References

The required data is available at the Website of the International HapMap project, under bulk data downloads > bulk data > frequencies

http://hapmap.ncbi.nlm.nih.gov

The HapMap files downloaded by this function are subject to the HapMap terms and policies. See: http://hapmap.ncbi.nlm.nih.gov/datareleasepolicy.html

See Also

`match_alleles`

Examples

```
# This command will download the CEU HapMap dataset and use
# it to generate an allele-reference. Create a folder
# "new_hapmap" to store the data and make sure there is
# sufficient disk space and a reasonably fast internet
# connection.

#new_hapmap <- create_hapmap_reference(dir = "C:/new_hapmap",
# download_hapmap = TRUE, download_subset = "CEU",
# filename = "new_hapmap", save_txt = TRUE,
# return_reference = TRUE)
```

---

filter_GWAS *Automated filtering and reformatting of GWAS results files*

Description

This function was created as a convenient way to automate the removal of low-quality and non-autosomal SNPs. It includes the same formatting options as `QC_GWAS`.

Usage

```
filter_GWAS(ini_file,
            GWAS_files, output_names,
            gzip_output = TRUE,
            dir_GWAS = getwd(), dir_output = dir_GWAS,
            FRQ_HQ = NULL, HWE_HQ = NULL,
            cal_HQ = NULL, imp_HQ = NULL,
            FRQ_NA = TRUE, HWE_NA = TRUE,
            cal_NA = TRUE, imp_NA = TRUE,
            ignore_impstatus = FALSE,
            remove_X = FALSE, remove_Y = FALSE,
            remove_XY = FALSE, remove_M = FALSE,
            header_translations,
```

```
              check_impstatus = FALSE,
              imputed_T = c("1", "TRUE", "yes", "YES", "y", "Y"),
              imputed_F = c("0", "FALSE", "no", "NO", "n", "N"),
              imputed_NA = NULL,
              column_separators = c("\t", " ", "", ",", ";"),
              header = TRUE, nrows = -1, nrows_test = 1000,
              comment.char = "", na.strings = c("NA", "."),
              out_header = "original", out_quote = FALSE,
              out_sep = "\t", out_eol = "\n", out_na = "NA",
              out_dec = ".", out_qmethod = "escape",
              out_rownames = FALSE, out_colnames = TRUE, ...)
```

Arguments

`ini_file`	(the filename of) a table listing the files to be processed and the filters to be applied. See 'Details'.
`GWAS_files`	character vector: when no `ini_file` is provided, this identifies the files to be processed. See 'Details'.
`output_names`	character vector: the filenames for the output files. The default option is to use the input filenames. Note that, unlike with other `QCGWAS` functions, the file extensions should be included (However, the function will automatically add `".gz"` when the files are compressed.
`gzip_output`	logical; should the output files be compressed?
`dir_GWAS, dir_output`	character-strings specifying the directory address of the folders for the input files and the output, respectively. Note that R uses *forward* slash (/) where Windows uses backslash (\).
`FRQ_HQ, HWE_HQ, cal_HQ, imp_HQ`	Numeric vectors. When no `ini_file` is provided, these arguments specify the filter threshold-values for allele frequency, HWE p-value, callrate and imputation quality, respectively. Passed to `HQ_filter`.
`FRQ_NA, HWE_NA, cal_NA, imp_NA`	Logical vectors. When no `ini_file` is provided, these arguments specify whether missing values (of allele frequency, HWE p-value, callrates and imputation quality, respectively) are excluded (`TRUE`) or ignored (`FALSE`). Passed to `HQ_filter`.
`ignore_impstatus`	Logical vector. When no `ini_file` is provided, this argument specifies whether imputation status is taken into account when applying the filters. If `FALSE`, HWE p-value and callrate filters are applied only to genotyped SNPs, and imputation quality filters only to imputed SNPs. If `TRUE`, the filters are applied to all SNPs regardless of the imputation status.
`remove_X, remove_Y, remove_XY, remove_M`	logical; respectively whether X-chromosome, Y-chromosome, pseudo-autosomal and mitochondrial SNPs are removed. Note: these arguments accept only a single `TRUE` or `FALSE` value. Unlike the above settings, it's not possible to specify them independently for every dataset.
`header_translations`	translation table for column names. See `translate_header` for more information. If the argument is left empty, `dataset` is assumed to use the standard column-names used by `QC_GWAS`.

`check_impstatus`
: logical; should `convert_impstatus` be called to convert the imputation-status column into standard values?

`imputed_T, imputed_F, imputed_NA`
: arguments passed to `convert_impstatus`.

`column_separators`
: character string or vector; specifies the values used as column delimitator in the GWAS file(s). The argument is passed to `load_test`; see the description of that function for more information.

`nrows_test`
: integer; the number of rows used for "trial-loading". Before loading the entire dataset, the function `load_test` is called to determine the dataset's file-format by reading the top `x` lines, where `x` is `nrows_test`. Setting `nrows_test` to a low number (e.g. `150`) means quick testing, but runs the risk of missing problems in lower rows. To test the entire dataset, set it to `-1`.

`header, nrows, comment.char, na.strings, ...`
: arguments passed to `read.table` when importing the dataset.

`out_header`
: Translation table for the column names of the *output* file. This argument is the opposite of `header_translations`: it translates the standard column-names of `QC_GWAS` to user-defined ones. `output_header` can be one of three things:
  - A user specified table similar to the one used by `translate_header`. However, as this translates standard names into non-standard ones, the standard names should be in the right column, and the desired ones in the left. There is also no requirement for the names in the *left* column to be uppercase.
  - The name of a file in `dir_GWAS` containing such a table.
  - Character string specifying a standard form. See `QC_GWAS`, section 'Output header' for the options.

`out_quote, out_sep, out_eol, out_na, out_dec, out_qmethod, out_rownames, out_colnames`
: arguments passed to `write.table` when saving the final dataset.

Details

The easiest way to use `filter_GWAS` is by passing an ini file to the `ini_file` argument. The ini file can be generated by running `QC_series` with the `save_filtersettings` argument set to `TRUE`. The output will include a file 'Check_filtersettings.txt', describing the (high-quality) filter settings used for each file (taking into account whether there was enough data, i.e. whether the `use_threshold` was met, to apply the filters).

The `ini_file` argument accepts both a table or the name of a file in `dir_GWAS` or the current R working directory.

If no `ini_file` is specified, the function will use the `GWAS_files`, x_HQ, x_NA and `ignore_impstatus` arguments to construct such a table. `GWAS_files` can either be a character vector or a single value. If a single string, all filenames containing the string will be processed. The other arguments can also be a vector or a single value; if the latter, they will be recycled to create a vector of the correct length.

If neither `ini_file` nor `GWAS_files` are specified, the function will look for a file `Check_filtersettings.txt` in `dir_GWAS` and the current R working directory.

Note that `ini_file` overrules the other filter settings, i.e. one cannot adjust `ini_file` through the other arguments.

Value

An invisible logical vector, indicating which files were successfully filtered.

Note

R is not the optimal platform for filtering GWAS files. This function was added at the request of a user, but an UNIX script is likely to be faster.

`gwa_sample` *Sample dataset for the QCGWAS package*

Description

A fake GWAS results dataset for use in the examples.

Usage

```
data(gwa_sample)
```

Format

A data frame with 10000 observations on the following 15 variables.

`MARKER` a character vector; the SNP IDs.

`STRAND` a character vector; the DNA strand of the listed alleles.

`CHR` a character vector; the chromosome of the listed SNP.

`POSITION` a numeric vector; the basepair position of the listed SNP.

`EFFECT_ALL` a character vector; the effect allele.

`OTHER_ALL` a character vector; the non-effect allele.

`N_TOTAL` a numeric vector; the sample size.

`EFF_ALL_FREQ` a numeric vector; frequency of the effect-allele in the dataset.

`HWE_PVAL` a numeric vector; Hardy-Weinberg (HWE) p-value of the listed alleles.

`CALLRATE` a numeric vector; SNP callrate.

`EFFECT` a numeric vector; effect size of the listed effect allele.

`STDERR` a numeric vector; standard error of the effect.

`PVALUE` a numeric vector; p-value of the effect.

`IMPUTED` a numeric vector; imputation status - i.e. whether the SNP is genotyped (`0`) or imputed (`1`).

`IMP_QUALITY` a numeric vector; imputation quality.

header_translations *Translation table for GWAS dataset headers*

Description

This is a sample translation table, as used by `translate_header` to translate dataset column names to the QCGWAS standard. An editable .txt version can be found in "R\library\QCGWAS\doc".

Usage

```
data(header_translations)
```

Format

A data frame with 104 observations on the following 2 variables.

`STANDARD` a character vector; standard column names as used by QCGWAS.

`ALTERNATIVE` a character vector; alternative column names, as used in the datasets.

HQ_filter *Select high-quality data in GWAS datasets*

Description

This function accepts a `QC_GWAS` dataset and returns a vector of logical values indicating which entries meet the quality criteria.

Usage

```
HQ_filter(data,
          ignore_impstatus = FALSE,
          FRQ_val = NULL, HWE_val = NULL,
          cal_val = NULL, imp_val = NULL,
          filter_NA = TRUE,
          FRQ_NA = filter_NA, HWE_NA = filter_NA,
          cal_NA = filter_NA, imp_NA = filter_NA)
```

Arguments

`data` table to be filtered. `HQ_filter` assumes the dataset uses the standard `QC_GWAS` column names.

`ignore_impstatus` logical; if `FALSE`, HWE p-value and callrate filters are applied only to genotyped SNPs, and imputation quality filters only to imputed SNPs. If `TRUE`, the filters are applied to all SNPs regardless of the imputation status.

`FRQ_val, HWE_val, cal_val, imp_val` numeric; the *minimal* required value for allele frequency, HWE p-value, call-rate and imputation quality respectively. Note that the allele-frequency filter is two-sided: for a filter-value of `x`, it will exclude entries with `freq < x` and `freq > 1 - x`.

`filter_NA` logical; if `TRUE`, then missing filter variables will be excluded; if `FALSE`, they will be ignored. `filter_NA` is the default setting for all variables. Variable-specific settings can be specified with the following arguments.

`FRQ_NA, HWE_NA, cal_NA, imp_NA`
logical; variable-specific settings for `filter_NA`.

Details

A SNP is considered high-quality if it meets all quality criteria. The thresholds are inclusive; i.e. SNPs that have a value equal or higher than the threshold will be considered high-quality.

To filter missing values only, set the filter argument to `NA`, and the corresponding NA-filter to `TRUE`.

To disable filtering entirely, set to `NULL`. This disables the filtering of missing values as well.

When imputation status is missing or invalid (and `ignore_impstatus` is `FALSE`), only the allele-frequency filter will be applied.

Value

A vector of logical values, indicating which values in `data` meet (`TRUE`) or fail (`FALSE`) the quality criteria.

Note

The table entered in the `data` argument must use the standard column names of `QC_GWAS`. Functions using `HQ_filter` usually allow the user to specify a translation table. If not, `translate_header` can be used to translate the header manually.

Examples

```
data("gwa_sample")

selected_SNPs <- HQ_filter(data = gwa_sample,
                           FRQ_val = 0.01,
                           cal_val = 0.95,
                           filter_NA = FALSE)
summary(gwa_sample[selected_SNPs, ])

selected_SNPs <- HQ_filter(data = gwa_sample,
                           FRQ_val = 0.01,
                           cal_val = 0.95,
                           filter_NA = FALSE,
                           ignore_impstatus = TRUE)
summary(gwa_sample[selected_SNPs, ])
```

---

identify_column *Identify non-standard column names*

Description

This function is a subroutine of `translate_header` and several other functions. It is used to translate non-standard column names into standard ones.

Usage

```
identify_column(std_name, alt_names, header)
```

Arguments

std_name	character string; the standard name for the data-column.
alt_names	translation table; with in the left column the standard name(s) and in the right column possible alternatives. See translate_header for more details.
header	the column names of the dataset. The names should be entirely in uppercase.

Details

The purpose of identify_column is essentially to look up in the translation table (alt_names) which of the names in header can be translated into std_name.

Value

An integer vector of the entry(s) in header (i.e. the column-numbers) that can be translated into std_name.

See Also

translate_header

Examples

```
sample_data <-
  data.frame(SNP = paste("rs", 1:10, sep = ""),
             chrom = 2,
             effect = 1:10/10,
             misc = NA)
sample_header <- toupper(names(sample_data))


alt_headers <-
  data.frame(
    standard = c("MARKER", "MARKER", "CHR", "CHR"),
    alternative = c("MARKER", "SNP", "CHR", "CHROM"),
    stringsAsFactors = FALSE)

identify_column(std_name = "EFFECT", alt_names = alt_headers,
                header = sample_header)
identify_column(std_name = "MARKER", alt_names = alt_headers,
                header = sample_header)
identify_column(std_name = "CHR", alt_names = alt_headers,
                header = sample_header)
identify_column(std_name = "MISC", alt_names = alt_headers,
                header = sample_header)
```

`intensity_plot` *Generates an intensity plot from x, y datasets*

Description

This function is used by `match_alleles` to generate an intensity plot. This function is currently only partially implemented, so we recommend that users do not bother with it.

Usage

```
intensity_plot(x, y, strata, nbin = 20,
               xmax = max(x), xmin = min(x),
               ymax = max(y), ymin = min(y),
               strata_colours = c("black", "red", "turquoise3"),
               verbose = TRUE, xlab = "x", ylab = "y", ...)
```

Arguments

`x, y`	numerical vectors; the x and y coordiantes of the datapoints.
`strata`	logical vector; indicates whether the datapoint belongs to strata 1 or 2. If missing, all datapoints are assumed to belong to strata 1.
`nbin`	integer: the number of bins (categories) on the x and y axis.
`xmax, xmin, ymax, ymin`	numeric; the range of x and y values shown in the plot.
`strata_colours`	character vector of length 3, indicating the colours to be used for entries of strata 1, 2 or mixed, respectively.
`verbose`	logical; determines whether a warning is printed in the console when datapoints are removed.
`xlab, ylab, ...`	arguments passed to `plot`.

Details

This function is intended as an alternative to the standard scatter plot for the allele-frequency correlation graph.

Value

An invisible object of type `list` with the following two components:

`NA_removed`	The number of entries removed due to missing values
`outliers_removed`	The number of entries removed because they exceeded the thresholds specified by the min/max arguments.

See Also

`match_alleles`

load_GWAS *Easy loading of GWAS results files*

Description

load_GWAS is wrapper-function of read.table that makes loading large GWAS results files less of a hassle. It automatically unpacks .zip and .gz files and uses load_test to determine which column separator the file uses.

Usage

```
load_GWAS(filename, dir = getwd(),
          column_separators = c("\t", " ", "", ",", ";"),
          test_nrows = 1000,
          header = TRUE, nrows = -1,
          comment.char = "", na.strings = c("NA", "."),
          stringsAsFactors = FALSE, ...)

load_test(filename, dir = getwd(),
          column_separators = c("\t", " ", "", ",", ";"),
          test_nrows = 1000, ...)
```

Arguments

filename	character string; the complete filename of the file to be loaded. Note that compressed files (.gz or .zip files) can only be unpacked if the filename of the archive contains the extension of the archived file. For example, if the archived file is named "data1.csv", the archive should be "data1.csv.zip".
dir	character string; the directory containing the file. Note that R uses *forward* slash (/) where Windows uses backslash (\).
column_separators	character string or vector of the column-separators to be tried by load_test. White-space can be specified by "", but it is recommended you try tab ("\t") and space (" ") first.
test_nrows	integer; the number of lines that load_test checks in the trail-load. A smaller number means faster loading, but also makes it more likely that errors slip through. To check the entire dataset, set to -1.
header, nrows, comment.char, na.strings, stringsAsFactors, ...	Arguments passed to read.table.

Details

load_test determines the correct column separator simply by trying them individually until it finds one that works (that is: one that results in a dataset with an equal number of cells in every row AND at least five or more columns). If none work, it reports the error-message generated by the last column separator tried.

The column separators are tried in the order specified by the column_separators argument.

By default, load_test only checks the first 1000 lines (adjustable by the test_nrows argument); if the problem lies further down in the dataset, it will not catch it. In such a case, load_GWAS and QC_GWAS will crash when attempting to load the dataset.

A common problem is employing white-space ("") as column separator for a file that uses empty fields to indicate missing values. The separators surrounding an empty field are adjacent, so R parses them as a single column separator. In this particular example, specifying a single space (" ") or tab ("\t") as column separator solves the problem (this is why the default setting of `column_separators` puts these values before white-space).

Value

`load_GWAS` returns the table imported from the specified file.

`load_test` returns a list with 4 components:

`success`	logical; whether `load_test` was able to load a dataset with five or more columns.
`error`	character string; if unable to load the file, this returns the error-message of the last column separator to be tried.
`file_type`	character string; the last three characters of `filename`.
`sep`	the first column-separator that succeeded in loading a dataset with five or more columns.

Note

`load_GWAS` uses the same default loading-settings as `QC_GWAS`. `load_test`, on the other hand, has no default values for `header`, `comment.char`, `na.strings` and `stringsAsFactors`, and uses the `read.table` defaults instead.

Examples

```
## As the function requires a GWAS file to work,
## the following code should be adjusted before execution.
## Because this is a demonstration, the nrows argument is used
## to read only the first 100 rows.

# data_GWAS <-
#    load_GWAS("GWA_results1.txt.zip",
#              dir = "C:/GWAS_results",
#              nrows = 100)
```

---

match_alleles *Check and correct alleles in GWAS result files*

Description

This function checks the reported alleles and allele frequencies in GWAS results data by comparing them to a reference table. It will also uniformize the dataset by switching all SNPs to the positive strand and flipping the alleles so that the effect allele matches the reference minor allele.

Usage

```
match_alleles(dataset, ref_set, HQ_subset,
              dataname = "dataset", ref_name = "reference",
       unmatched_data = !all(dataset$MARKER %in% ref_set$SNP),
              check_strand = FALSE,
              save_mismatches = TRUE, delete_mismatches = FALSE,
              delete_diffEAF = FALSE, threshold_diffEAF = 0.15,
              check_FRQ = TRUE, check_ambiguous = FALSE,
              plot_FRQ = FALSE, plot_intensity = FALSE,
              plot_if_threshold = FALSE,
              threshold_r = 0.95,
              return_SNPs = FALSE, return_ref_values = FALSE,
              header_translations, header_reference,
              save_name = dataname, save_dir = getwd(),
              use_log = FALSE, log_SNPall = nrow(dataset))
```

Arguments

`dataset`	table containing the allele data. `dataset` should always contain columns for the SNPID, the effect allele and the other allele. Strand and allele frequency may be required, depending upon the settings, while effect size is optional. `match_alleles` accepts non-standard column names, provided a translation table is specified in `header_translations`. The order of columns does not matter; nor does the presence of other columns.
`ref_set`	table containing the reference data. `ref_set` should always contain columns for the SNPID, the minor allele and the major allele. Minor allele frequency is only required if `check_FRQ` or `check_ambiguous` are `TRUE`. The standard column-names are `"SNP"`, `"MINOR"`, `"MAJOR"` and `"MAF"`. Non-standard column names are accepted if a translation table is specified in `header_reference`. All SNPs *must* be aligned to the positive strand.
`HQ_subset`	an *optional* logical or numeric vector indicating the rows in `dataset` that contain high quality SNPs.
`dataname, ref_name`	character strings; the names of the dataset and reference, respectively. Used as identifiers in the output files.
`unmatched_data`	logical; are there SNPs in the dataset that do not appear in the reference? This argument is currently redundant: the function will determine it automatically.
`check_strand`	logical; should the function check for negative-strand SNPs? If `FALSE`, all SNPs are assumed to be on the positive strand.
`save_mismatches`	logical; should mismatching entries be exported to a .txt file before they are corrected?
`delete_mismatches`	logical; should mismatching SNPs (that could not be corrected by strand-switching) have their effect allele set to missing?
`delete_diffEAF`	logical; should SNPs that exceed the `threshold_diffEAF` have their effect allele set to missing?
`threshold_diffEAF`	numeric; the max. allowed difference between reported and reference allele frequency.

`check_FRQ` logical; should the function correlate the reported allele-frequency with that of the reference?

`check_ambiguous` logical; should the function do separate frequency correlations and create separate plots for SNPs with a strand-independent allele-pair (i.e. an A/T or C/G configuration)?

`plot_FRQ` logical; should a scatterplot of reported vs. reference allele-frequency be made?

`plot_intensity` logical; if `TRUE`, instead of a scatterplot an intensity plot is generated. This option is currently only partially implemented. Leave to `FALSE` for now.

`plot_if_threshold` logical; if `TRUE`, the scatterplot is only made when frequency correlation is below the threshold specified by `threshold_r`.

`threshold_r` numeric; the correlation threshold value.

`return_SNPs` logical; should the return value include the relevant columns of `dataset`?

`return_ref_values` logical; should the return-value include the matching entries in `ref_set`?

`header_translations, header_reference` translation tables for converting the column names of `dataset` and `ref_set` to standard names, respectively. See `translate_header` for more information.

`save_name` character string; the filename, *without* extension, for the various output files.

`save_dir` character string; the directory where the output files are saved. Note that R uses *forward* slash (/) where Windows uses backslash (\).

`use_log, log_SNPall` arguments used by `QC_GWAS`; redundant when `match_alleles` is used separately.

Details

`match_alleles` is one of the more complicated functions of `QCGWAS`. However, what it does is quite simple:

- Check for incorrect allele pairs
- Uniformize the output so that all SNPs are on the positive strand, and identical SNPs will have the same effect allele and other allele in all datasets
- Check the allele frequency

The complexity stems from the fact that these three tasks have to be carried out together and often overlap. So the actual function schematic looks like this:

- Switch negative-strand SNPs (i.e. SNPs with `"-"` in the strand-column) to the positive strand. This step can be disabled by setting `check-strand` to `FALSE`.
- Correct (if possible) mismatching alleles. A mismatch is when the reported allele-pair does not match that in the reference. `match_alleles` will attempt to fix the mismatch by "strand-switching" the alleles. The assumption is that dataset merely reported the wrong strand, so converting them to the opposing strand should solve the mismatch. If it does not, the SNPs are truly mismatches. If `delete_mismatches` is `TRUE`, the effect alleles are set to `NA`; if `FALSE`, they are restored to their original configuration and excluded from the allele-frequency test. If `save_mismatches` is `TRUE`, the entries are exported in a .txt before being changed. Note that `save_mismatches` only exports true mismatches (i.e. not those that were fixed after strand-switching).

- Align the allele-pairs with the reference. In order to have the same effect allele with the same SNP in every dataset, SNPs are "flipped" so that the effect allele matches the reference minor allele. Flipped alleles will also have their allele frequency and effect size inverted.
- Check for undetected strand-mismatch by counting the number of ambiguous and (if check_FRQ is TRUE) suspect SNPs. Ambiguous SNPs are SNPs with an allele-pair that is identical on both strands (i.e. A/T or C/G). Suspect SNPs are ambiguous SNPs whose allele-frequency differs strongly from that of the reference.
- Check allele-frequencies by correlating and/or plotting them against the reference. If check_ambiguous is TRUE, additional scatterplots will be made for the subsets of ambiguous and non-ambiguous SNPs. If delete_diffEAF is TRUE, SNPs whose allele-frequency differs from the reference by more than threshold_diffEAF have their effect alleles set to NA as well. This entire step can be disabled by setting check_FRQ to FALSE.

Value

An object of class 'list' with the following components:

FRQ_cor, FRQ_cor_ambiguous, FRQ_cor_nonambi
: Allele-frequency correlations for all, ambiguous, and non-ambiguous SNPs respectively.

n_SNPs
: Total number of SNPs in dataset

n_missing, n_missing_data, n_missing_ref
: Number of SNPs with missing allele-data in either dataset or reference, dataset only, and reference only, respectively.

n_negative_strand, n_negative_switch, n_negative_mismatch
: Number of negative-strand SNPs, the subset of negative-strand SNPs that were strand-switched twice because they did not match the reference, and the subset of double-switched SNPs that were still mismatching after the second strand-switch.

n_strandswitch, n_mismatch
: Number of SNPs that was strand-switched because they did not match the reference, and the subset of those that still did not match after the strand-switch.

n_flipped
: Number of SNPs whose alleles were flipped to align them with the reference.

n_ambiguous, n_suspect
: Number of ambiguous SNPs, and the subset of those that had a large allele-frequency aberration.

n_diffEAF
: Number of SNPs whose allele-frequency differs from the reference by more than threshold_diffEAF.

MARKER
: When return_SNPs and/or return_ref_values is TRUE, this returns the column of dataset containing the SNP IDs. If not, this returns NULL.

EFFECT_ALL, OTHER_ALL, STRAND, EFFECT, EFF_ALL_FREQ
: If return_SNPs is TRUE, these elements return the corrected columns of data-set. If FALSE, these return NULL. Note: match_alleles only returns those columns that were checked; if check_FRQ is FALSE, EFF_ALL_FREQ return NULL. The same goes for check_strand and STAND. EFFECT is only returned if present in dataset.

ref_MINOR, ref_MAJOR, ref_MAF
: If return_ref_values is TRUE, these elements return the reference minor and major alleles and allele frequency column for the SNPs in MARKER.If FALSE, these return NULL. ref_MAF is only returned when check_FRQ is TRUE.

Interpreting the output

The output of `match_alleles` may seem a bit overwhelming at first, so here is a short explanation of what it means and what you should pay attention to.

The columns included in the return value when `return_SNPs` is `TRUE` are the post-matching dataset. This is only relevant if you want to continue working with the corrected dataset. Similarly, the output of `return_ref_values` is only used for comparing the post-matching dataset to the reference.

`n_missing`, `n_missing_data` and `n_missing_ref` report the prevalence of missing allele data, but are otherwise irrelevant.

The majority of return values serve to check whether strand-switching was performed correctly. `n_strandswitch` indicates how many SNPs were converted to the other strand because of a mismatch with the reference. In our experience, many cohorts do not include stand data, or simply set all SNPs to `"+"`, so the presence of strand-switched SNPs isn't an indicator of problems by itself. However, if the strand-switching did not fix the mismatch, there may a problem. The subset of strand-switched SNPs that could not be fixed is reported as `n_mismatch`, and indicates incorrect allele data or, possibly, triallelic SNPs.

Depending on the argument `save_mismatches`, mismatching entries are exported as a .txt file, together with the reference data. This allows the user to see which SNPs are affected.

Another sign of trouble is when negative-strand SNPs (`n_negative_strand`) are present (i.e. the cohort included real strand data, rather just setting it to `"+"`), but strand-switching still occurred. Negative-strand SNPs are converted to the positive strand before their alleles are compared to the reference, so they should not appear here. If they do, it means that either the strand-column data is incorrect, or it is an ordinary mismatch (see above).

Negative-strand SNPs that are "strand-switched" will revert to their original allele configuration (but the strand column now reports them as being on positive strand). The output of `QC_GWAS` calls them double strand-switches, but here they are reported as `n_negative_switch`. The subset of those that could not be fixed is `n_negative_mismatch`.

Just to be clear: the relevant output is still `n_strandswitch` and `n_negative_strand`, not `n_negative_switch` and `n_negative_mismatch`. Whether it was the negative-strand SNPs that were switched or not is not important: the important thing is that there were negative-strand SNPs (i.e. the cohort included real strand data rather setting everything to `"+"`); yet strand-switches were still necessary and cannot be attributed to mismatch (i.e. the strand data is incorrect).

`n_flipped` counts how many SNPs had their alleles reversed to match the effect allele with the reference minor-allele. This is merely recorded for "administrative" purposes, and shouldn't concern the user.

`n_ambiguous` and `n_suspect` are another test of the strand information. Ambiguous SNPs are SNPs that have the same allele pair on the positive and negative strands (i.e. A/T or C/G). Matching them with the allele-reference therefor won't detected incorrect strand-information. In a normal-sized GWAS results file, about 15% of SNPs will be ambiguous.

Suspect SNPs are the subset of ambiguous SNPs whose allele frequency is significantly different from that in the reference (< 0.35. vs > 0.65 or visa versa). In our experience, a GWAS results file with 2.5M SNPs will have only a few dozen suspect SNPs. However, if it's a sizable proportion of all ambiguous SNPs, it indicates that the ambiguous SNPs are listed for the wrong strand. This will also have resulted in the wrong SNPs being flipped in the previous step, so it should be visible in the allele-frequency correlation test as well.

`n_diffEAF` counts SNPs with significantly different allele-frequencies. A large number here indicates either that the allele-frequencies are incorrect or listed for the wrong allele (see below), or that the population used in the dataset does not match that of the reference.

The FRQ_cor value is the correlation between the reported and reference allele-frequencies. If allele frequency is correct, the correlation should be near 1. If it's close to -1, the listed frequency is that of the other (i.e. non-effect) allele.

the FRQ_cor_ambiguous and FRQ_cor_nonambi values are the same test for the subsets of ambiguous and non-ambiguous SNPs. If ambiguous SNPs are listed on the wrong strand, then they will have been flipped incorrectly, so allele-frequency correlation should also move towards -1.

Note

The function does not delete SNPs, regardless of the delete_mismatches delete_diffEAF arguments. Setting these to TRUE means that any such SNPs are marked by having their effect allele set to NA. The actual deletion takes place inside QC_GWAS.

See Also

create_hapmap_reference for creating an allele reference from publicly-available HapMap data

switch_strand

Examples

```
# In order to keep the QCGWAS package small, no allele reference
# is included. Follow the example of the create_hapmap_reference
# function to make one.

#  data("gwa_sample")
#  hapmap_ref <- read.table("C:/new_hapmap/new_hapmap.txt",
#                  header = TRUE, stringsAsFactors = FALSE)

#  match_alleles(gwa_sample, hapmap_ref,
#    dataname = "sample data", ref_name = "HapMap",
#    save_name = "test_allele1", save_dir = "C:/new_hapmap",
#    check_strand = TRUE, plot_FRQ = TRUE)

# HQ_SNPs <- HQ_filter(data = gwa_sample, filter_NA = TRUE,
#                          filter_FRQ = 0.01, filter_cal = 0.95)
#  match_alleles(gwa_sample, hapmap_ref,
#    HQ_subset = HQ_SNPs,
#    dataname = "sample data", ref_name = "HapMap",
#    save_name = "test_allele2", save_dir = "C:/new_hapmap",
#    check_strand = TRUE, plot_FRQ = TRUE)

#  match_output <-
#    match_alleles(gwa_sample, hapmap_ref,
#      HQ_subset = HQ_SNPs,
#      delete_mismatches = TRUE, return_SNPs = TRUE,
#      delete_diffEAF = TRUE, threshold_diffEAF = 0.15,
#      dataname = "sample data", ref_name = "HapMap",
#      save_name = "test_allele3", save_dir = "C:/new_hapmap",
#      check_strand = TRUE, plot_FRQ = TRUE)

#  if(sum(match_output$n_negative_strand,
#         match_output$n_strandswitch, match_output$n_mismatch,
#         match_output$n_flipped, match_output$n_diffEAF) > 0){
#    gwa_sample$EFFECT_ALL   <- match_output$EFFECT_ALL
#    gwa_sample$OTHER_ALL    <- match_output$OTHER_ALL
```

```
#    gwa_sample$STRAND       <- match_output$STRAND
#    gwa_sample$EFFECT       <- match_output$EFFECT
#    gwa_sample$EFF_ALL_FREQ <- match_output$EFF_ALL_FREQ
#  }
```

`plot_distribution` *GWAS effect-Size distribution plot*

Description

This function generates the effect-size distribution boxplot created by `QC_series`.

Usage

```
plot_distribution(data_table,
                  names = 1:ncol(data_table),
                  include = TRUE,
                  plot_order = 1:ncol(data_table),
                  quantile_lines = FALSE,
                  save_name = "Graph_distribution",
                  save_dir = getwd(), ...)
```

Arguments

`data_table`	table with a column of effect sizes for every dataset.
`names`	vector; the names for the datasets, for use in the graph. Note: it's best to keep these very short, as long labels won't be plotted. The default is the column *numbers* of `data_table`.
`include`	logical vector indicating which columns of `data_table` are included in the plot. The default setting is to include all.
`plot_order`	numeric vector determining the left-to-right order of plotting the datasets (columns). `QC_series` uses the sample size for this.
`quantile_lines`	logical; should lines representing the median and quartile values be included?
`save_name`	character string; the filename, *without* extension, for the graph file.
`save_dir`	character string; the directory where the graph is saved. Note that R uses *forward* slash (/) where Windows uses backslash (\).
`...`	arguments passed to `boxplot`.

Details

When running a QC over multiple files, `QC_series` collects the values of the `effectsize_HQ` output of `QC_GWAS` in a table, which is then passed to this function. If there are significant differences in the distribution of effect sizes, it usually indicates that the datasets did not use the same model or unit.

Value

An invisible `NULL`.

Note

There is a known bug with this function when called by QC_series. As input for names, QC_series pastes together a shortened filename and a "N = x" string giving the dataset's sample size.

The filenames are truncated to the first unique element; e.g. files "cohortX_male_HB.txt" and "cohortX_female_HB.txt" become "cohortX_male; N = 608" and "cohortX_female; N = 643", respectively. However, if the unique element is longer than approx. 15 characters, the label is too long to be plotted. The only solution is to change the filenames prior to passing the files to QC_series.

See Also

For comparing reported to expected effect-size distribution: QC_histogram.

For other plots comparing GWAS results files: plot_precision and plot_skewness.

Examples

```
data("gwa_sample")

chunk1 <- gwa_sample$EFFECT[1:1000]
chunk2 <- gwa_sample$EFFECT[1001:2000]
chunk3 <- gwa_sample$EFFECT[2001:3000]

plot_distribution(
  data_table = data.frame(chunk1, chunk2, chunk3),
  names = c("chunk 1", "chunk 2", "chunk 3"),
  quantile_lines = TRUE,
  save_name = "sample_distribution")
```

---

plot_precision *GWAS Precision Plot*

Description

This function generates the precision plot created by QC_series. Precision is defined as:

1 / median standard-error.

Usage

```
plot_precision(SE, N,
               labels = NULL,
               save_name = "Graph_precision",
               save_dir = getwd(), ...)
```

Arguments

SE, N	numeric vectors containing the median standard-error and sample size of the datasets, respectively.
labels	vector containing names or other identifiers for the datapoints, to be plotted in the graph. Note: it's best to keep these very short. To disable labeling, set to NULL (default).

save_name	character string; the filename, *without* extension, for the graph file.
save_dir	character string; the directory where the graph is saved. Note that R uses *forward* slash (/) where Windows uses backslash (\).
...	arguments passed to plot.

Details

When running a QC over multiple files, QC_series collects the values of the SE_HQ_median and sample_size_HQ outputs of QC_GWAS in a table, which is then passed to this function to convert it to a plot.

The plot is to provide a visual estimate whether the standard errors are within the expected range. As sample size increases, the median standard error is expected to decrease, so the plot should show a linear relation.

Value

An invisible NULL.

See Also

plot_distribution and plot_skewness.

Examples

```
value_SE <- c(0.078, 0.189, 0.077, 0.040, 0.021, 0.072)
value_N <- c(870, 830, 970, 690, 2200, 870)
value_labels <- paste("cohort", 1:6)

plot_precision(SE = value_SE, N = value_N,
               labels = value_labels,
               save_name = "sample_precision")
```

---

plot_regional *Regional Association Plot*

Description

A regional association plot is essentially a zoomed-in Manhattan plot, allowing the researcher to look at associations in a small, pre-defined area of the genome.

Usage

```
plot_regional(dataset,
              chr, start_pos, end_pos,
              plot_cutoff_p = 1,
              name_cutoff_p,
              data_name = NULL,
              save_name = "regional_association_plot",
              save_dir = getwd(),
              header_translations,
              main = "Regional association plot", ...)
```

Arguments

dataset	data frame containing the SNPs chromosome number, position, p-value and (if a name_cutoff_p is specified) their SNP IDs.
chr	character or numeric value indicating the chromosome of interest.
start_pos, end_pos	Numeric; the position values of the beginning and end of the region of interest.
plot_cutoff_p	numeric - the threshold of p-values to be shown in the QQ & Manhattan plots. Higher (less significant) p-values are not included in the plot. Note that, unlike in QC_GWAS or QC_plots, the default value is 1 (i.e. includes everything), rather than 0.05. Setting it to 0.05 excludes 95% of data-points, which significantly reduces running time and memory usage in larger datasets.
name_cutoff_p	numeric; SNPs with p-values lower than or equal to this value will have their names plotted in the graph. If set to NULL (default), no names will be plotted.
data_name	character string; the subtitle for the plot.
save_name	character string; the filename, *without* extension, for the graph file.
save_dir	character string; the directory where the graph is saved. Note that R uses *forward* slash (/) where Windows uses backslash (\).
header_translations	translation table for column names. See translate_header for more information. If the argument is left empty, dataset is assumed to use the standard column-names of QC_GWAS.
main, ...	arguments passed to plot.

Details

By default, plot_regional expects dataset to use the standard column-names used by QC_GWAS. A translation table can be specified in header_translations to allow non-standard names. See translate_header for more information.

Value

An invisible NULL.

See Also

For creating a Manhattan plot: QC_plots.

Examples

```
data("gwa_sample")

plot_regional(dataset = gwa_sample,
        chr = 2, start_pos = 55000000, end_pos = 75000000,
        data_name = "QC GWAS sample data",
        save_name = "sample_regional_association")
```

plot_skewness *GWAS Skewness vs. Kurtosis Plot*

Description

This function generates the skewness vs. kurtosis plot created by QC_series.

Usage

```
plot_skewness(skewness,
              kurtosis,
              labels = paste("Study", 1:length(skewness)),
              plot_labels = "outliers",
              save_name = "Graph_skewness_kurtosis",
              save_dir = getwd(), ...)
```

Arguments

skewness, kurtosis	Vectors containing the skewness and kurtosis values of the datasets
labels	vector containing names or other identifiers for the datapoints, to be plotted in the graph. Note: it's best to keep these very short.
plot_labels	character string or logical determining whether the values in labels are plotted next to the data points. The possible settings are: "none" (or FALSE); "all" (or TRUE); and "outliers" for outliers only.
save_name	character string; the filename, *without* extension, for the graph file.
save_dir	character string; the directory where the graph is saved. Note that R uses *forward* slash (/) where Windows uses backslash (\).
...	arguments passed to plot.

Details

When running a QC over multiple files, QC_series collects the values of the skewness_HQ and kurtosis_HQ output of QC_GWAS in a table, which is then passed to this function to convert it into a plot. Note that this values are calculated over high-quality SNPs only.

Kurtosis is a measure of how well a distribution matches a Gaussian distribution. A Gaussian distribution has a kurtosis of 0. Negative kurtosis indicates a flatter distribution curve, while positive kurtosis indicates a sharper, thinner curve.

Skewness is a measure of distribution asymmetry. A symmetrical distribution has skewness 0. A positive skewness indicates a long tail towards higher values, while a negative skewness indicates a long tail towards lower values.

Ideally, one expects both the skewness and kurtosis of effect sizes to be close to 0. In practice, these statistics can be hugely variable. QC_series uses only high-quality effect sizes to calculate these values in order to reduce some of the more extreme values. Still, it is recommended that you compare the values to those of other GWAS with the same phenotype, rather than relying on on the label outliers command to identify problems.

Value

An invisible NULL.

See Also

For calculating skewness and kurtosis: calc_kurtosis.

For other plots comparing GWAS results files: plot_precision and plot_distribution.

Examples

```
value_S <- c(0.05, -0.27, 0.10, 0.11)
value_K <- c( 6.7,  10.0, 10.1,  6.6)
value_labels <- paste("cohort", 1:4)

plot_skewness(skewness = value_S,
              kurtosis = value_K,
              labels = value_labels,
              plot_labels = "outliers",
              save_name = "sample_skewness_kurtosis")
```

QC_GWAS *Automated Quality Control of GWAS Results files*

Description

QC_GWAS runs a full quality control (QC) over a single GWAS results file. It removes missing and invalid data, checks the alleles and allele frequency with a reference, tests the reported p-value against both calculated and expected values, creates QQ and Manhattan plots and reports the distribution of the quality-parameters within the dataset, as well as various QC statistics.

QC_series does the same thing for multiple GWAS results files. It's mainly a wrapper that passes individual files to QC_GWAS, but it has a few extra features, such as making a checklist of important QC stats and creating several graphs to compare the QC'ed files.

Although the number of arguments in QC_GWAS may seem overwhelming, only three of them are required to run a basic QC. The name of the file to be QC'ed should be passed to the filename argument; the directory of said file to the dir_data argument; and a header-translation table to the header_translations argument. For a quick introduction to QCGWAS, read the quick reference guide that can be found in "R\library\QCGWAS\doc".

Usage

```
QC_GWAS(filename,
        filename_output = paste0("QC_", filename),
        dir_data = getwd(),
        dir_output = paste(dir_data, "QCGWASed", sep = "/"),
        dir_references = dir_data,
        header_translations,
        column_separators = c("\t", " ", "", ",", ";"),
        nrows = -1, nrows_test = 1000,
        header = TRUE, comment.char = "",
        na.strings = c("NA", "nan", "NaN", "."),
        imputed_T = c("1", "TRUE", "T"),
        imputed_F = c("0", "FALSE", "F"),
        imputed_NA = c(NA, "-"),
        save_final_dataset = TRUE,
```

```
          gzip_final_dataset = TRUE, order_columns = FALSE,
          spreadsheet_friendly_log = FALSE,
          out_header = "standard",
          out_quote = FALSE, out_sep = "\t", out_eol = "\n",
          out_na = "NA", out_dec = ".", out_qmethod = "escape",
          out_rownames = FALSE, out_colnames = TRUE,
          return_HQ_effectsizes = FALSE,
          remove_X = FALSE, remove_Y = FALSE,
          remove_XY = remove_Y, remove_M = FALSE,
          calculate_missing_p = FALSE,
          make_plots = TRUE, only_plot_if_threshold = TRUE,
          threshold_allele_freq_correlation = 0.95,
          threshold_p_correlation = 0.99,
          plot_intensity = FALSE,
          plot_histograms = make_plots, plot_QQ = make_plots,
          plot_QQ_bands = TRUE, plot_Manhattan = make_plots,
          plot_cutoff_p = 0.05,
          allele_ref_std, allele_name_std,
          allele_ref_alt, allele_name_alt,
          update_alt = FALSE, update_savename,
          update_as_rdata = FALSE, backup_alt = FALSE,
          remove_mismatches = TRUE,
          remove_mismatches_std = remove_mismatches,
          remove_mismatches_alt = remove_mismatches,
          threshold_diffEAF = 0.15, remove_diffEAF = FALSE,
          remove_diffEAF_std = remove_diffEAF,
          remove_diffEAF_alt = remove_diffEAF,
          check_ambiguous_alleles = FALSE,
          use_threshold = 0.1,
          useFRQ_threshold = use_threshold,
          useHWE_threshold = use_threshold,
          useCal_threshold = use_threshold,
          useImp_threshold = use_threshold,
          useMan_threshold = use_threshold,
          HQfilter_FRQ = 0.01, HQfilter_HWE = 10^-6,
          HQfilter_cal = 0.95, HQfilter_imp = 0.3,
          QQfilter_FRQ = c(NA, 0.01, 0.05),
          QQfilter_HWE = c(NA, 10^-6, 10^-4),
          QQfilter_cal = c(NA, 0.95, 0.99),
          QQfilter_imp = c(NA, 0.3, 0.5, 0.8),
          NAfilter = TRUE,
          NAfilter_FRQ = NAfilter, NAfilter_HWE = NAfilter,
          NAfilter_cal = NAfilter, NAfilter_imp = NAfilter,
          ignore_impstatus = FALSE,
          minimal_impQ_value = -0.5, maximal_impQ_value = 1.5,
          logI = 1L, logN = 1L, ...)
QC_series(data_files, datafile_tag, output_filenames,
          dir_data = getwd(),
          dir_output = paste(dir_data, "QCGWASed", sep = "/"),
          dir_references = dir_data,
          header_translations, out_header = "standard",
          allele_ref_std, allele_name_std,
```

```
          allele_ref_alt, allele_name_alt,
          update_alt = FALSE, update_savename,
          update_as_rdata = FALSE, backup_alt = FALSE,
          plot_effectsizes = TRUE, lim_effectsizes = NULL,
          plot_SE = TRUE, label_SE = TRUE,
          plot_SK = TRUE, label_SK = "outliers",
          save_filtersettings = FALSE, ...)
```

Arguments

`filename, data_files, datafile_tag`
: `filename` and `data_files` are, respectively, the name and names of the GWAS results file(s) to be QC'ed. If no `data_files` are specified, `QC_series` will process all filed in `dir_data` containing the string passed to `datafiles_tag` in their filename. See below for more information on the input requirements.

`filename_output, output_filenames`
: respectively the filename or names of the output of the QC. This should not include an extension, since the QC will automatically add one. The default is to use the input filename with `"QC_"` prefixed.

`dir_data, dir_output, dir_references`
: character strings specifying the directory dress of the folders for, respectively, the input file(s), the output file(s) and the auxillary files (header-translation tables and allele references). Note that R uses *forward* slash (/) where Windows uses backslash (\). If `dir_output` does not exist, it will be created. If no `dir_output` is specified, a folder named `"QCGWASed"` will be created in `dir_data`.

`header_translations`
: Translation table for the column names of the *input* file. Alternatively, the name of a file in `dir_references` containing such a table. See `translate_header` for details.

`column_separators`
: character string or vector; specifies the values used as column delimitator in the GWAS file. The argument is passed to `load_test`; see the description of that function for more information.

`nrows_test`
: integer; the number of rows used for "trial-loading". Before loading the entire dataset, the function `load_test` is called to determine the dataset's file-format by reading the top `x` lines, where `x` is `nrows_test`. Setting `nrows_test` to a low number (e.g. `150`) means quick testing, but runs the risk of missing problems in lower rows. To test the entire dataset, set it to `-1`.

`nrows, header, comment.char`
: arguments passed to `read.table` when importing the dataset.

`na.strings`
: character vector describing the values that represent missing data in the dataset. Passed to `read.table`.

`imputed_T, imputed_F, imputed_NA`
: character vectors; passed to `convert_impstatus` (as `T_strings`, `F_strings` and `NA_strings`, respectively) to translate the imputation-status column. Note that the current version of `QC_GWAS` *always* translates the imputation status. Even when the dataset already has the correct format, the user still needs to specify `1`, `0` and `NA` for these arguments, respectively. Also note that R distinguishes between the value `NA` and the character string `"NA"`.

`save_final_dataset`
: logical; should the post-QC dataset be saved?

`gzip_final_dataset`
: logical; should the post-QC dataset be compressed?

`order_columns`
: logical; should the post-QC dataset use the default column order?

`spreadsheet_friendly_log`
: logical; if `TRUE`, the final log file will be tab-separated, for easy viewing in a spreadsheet program. If `FALSE` (default), it will be formatted for pretty viewing in a text-processing program.

`out_header`
: Translation table for the column names of the *output* file. This argument is the opposite of `header_translations`: it translates the standard column-names of `QC_GWAS` to user-defined ones. `output_header` can be one of three things:

  - A user specified table similar to the one used by `translate_header`. However, as this translates standard names into non-standard ones, the standard names should be in the right column, and the desired ones in the left. There is also no requirement for the names in the *left* column to be uppercase.
  - The name of a file in `dir_references` containing such a column.
  - Character string specifying a standard form. See the section 'Output header' below for the options.

`out_quote, out_sep, out_eol, out_na, out_dec, out_qmethod, out_rownames, out_colnames`
: arguments passed to `write.table` when saving the final dataset.

`return_HQ_effectsizes`
: logical; return a vector of (max. 1000) high-quality effect-sizes? (In `QC_series`, this is set by the `plot_effectsizes` argument.)

`remove_X, remove_Y, remove_XY, remove_M`
: logical; respectively whether X-chromosome, Y-chromosome, pseudo-autosomal and mitochondrial SNPs are removed from the dataset.

`calculate_missing_p`
: logical; should the QC calculate missing/invalid p-values in the dataset?

`make_plots`
: logical; should the QC generate and save QQ plots, a Manhattan plot, histograms of data distribution and scatter plots of correlation in `dir_reference`?

`only_plot_if_threshold`
: logical; should the scatterplots only be made if the correlation is below a threshold value?

`threshold_allele_freq_correlation, threshold_p_correlation`
: numeric; thresholds for reporting and plotting the correlation between respectively the allele frequency of the dataset and the reference, and the calculated and reported p-values.

`plot_intensity`
: logical; if `TRUE`, instead of a scatterplot of allele correlations, an intensity plot is generated. This option is currently only partially implemented. Leave to `FALSE` for now.

`plot_histograms`
: logical; should histograms of the effect sizes, standard errors, allele frequencies, HWE p-values, callrates and imputation quality be made?

`plot_QQ, plot_Manhattan`
: logical; should QQ and Manhattan plots be made?

`plot_QQ_bands`
: logical; include probability bands in the QQ plot?

`plot_cutoff_p`
: numeric; significance threshold for inclusion in the QQ and Manhattan plots. The default value (`0.05`) excludes 95% of SNPs, significantly reducing running-time and memory usage. For this reason it is not recommend to set a higher value when QC'ing a normal-sized GWAS dataset.

`allele_ref_std, allele_ref_alt`
: the standard and alternative allele-reference tables. Alternatively, the name of a file in `dir_references` containing said table. Files in `.RData` format are accepted, but the table's object name must match the argument name. See `match_alleles` for more information on the input requirements.

`allele_name_std, allele_name_alt`
: character strings; these name the standard and alternative allele reference in the output. If no values are given, the function will use the reference's filename (if specified) or a default name.

`update_alt`
: logical; if the function encounters SNPs not included in either the standard or alternative reference, should these be added to the alternative reference file? If no alternative reference was specified, this creates one.

`update_savename`
: character string; the filename for saving the updated alternative reference, *without* extension. If `allele_ref_alt` is a filename, it is not necessary to specify this argument.

`update_as_rdata`
: logical; should the updated alternative allele reference be saved as `.RData` (`TRUE`) or a tab-delimitated .txt file (`FALSE`).

`backup_alt`
: logical; if the alternative allele reference is updated, should a back-up be made of the original reference file?

`remove_mismatches, remove_mismatches_std, remove_mismatches_alt`
: logical; should SNPs with mismatching alleles be removed from the dataset? `remove_mismatches` serves as the default value; the other two arguments determine this setting for the standard and alternative references, respectively.

`threshold_diffEAF`
: Numeric; the threshold for the difference between reported and reference allele-frequency. SNPs for which the difference exceeds the threshold are counted and (optionally) removed.

`remove_diffEAF, remove_diffEAF_std, remove_diffEAF_alt`
: Logical; should SNPs that exceed the `threshold_diffEAF` be removed from the dataset? `remove_diffEAF` serves as the default value; the other two arguments determine this setting for the standard and alternative references, respectively.

`check_ambiguous_alleles`
: logical; check for SNPs with strand-independent allele-configurations (i.e. A/T and C/G SNPs)?

`use_threshold`
: numeric; threshold value. The relative or absolute number of usable values required for a variable to be used in the QC. These arguments prevent the QC from applying filters to variables with no data. If a variable has less non-missing, non-invalid values than specified in the threshold, it will be ignored; i.e. no filter will be applied to it and no plots will be made. Values `> 1` specify the absolute threshold, while values of `1` or lower specify the fraction of SNPs remaining in the dataset. This argument is the default threshold for all variables; variable-specific thresholds can be set with the following arguments.

`useFRQ_threshold, useHWE_threshold, useCal_threshold, useImp_threshold, useMan_threshold`
: numeric; variable-specific thresholds for allele frequency, HWE p-value, call-rate, imputation quality and Manhattan plot (i.e. chromosome & position values) respectively.

`HQfilter_FRQ, HQfilter_HWE, HQfilter_cal, HQfilter_imp`
: numeric; threshold values for the high-quality SNP filter. SNPs that do not meet or exceed all four thresholds will be excluded from several QC tests. The filters

are for allele frequency, HWE p-value, callrate & imputation quality, respectively, and are processed by HQ_filter. See 'Filter arguments' for more information. Note: the high-quality filter does not remove SNPs; it merely excludes them from several QC tests.

QQfilter_FRQ, QQfilter_HWE, QQfilter_cal, QQfilter_imp
: numeric vector; threshold values for the QQ plot filters. SNPs that do not meet or exceed the value will be excluded from the QQ plot. Up to five values can be specified per filter. The filters are for allele-frequency, HWE p-value, callrate & imputation quality respectively, and are processed by QC_plots. See 'Filter arguments' for more information.

NAfilter, NAfilter_FRQ, NAfilter_HWE, NAfilter_cal, NAfilter_imp
: logical; should the high-quality and QQ filters exclude (TRUE) or ignore (FALSE) missing values? NAfilter is the default setting; the others allow allow variable specific settings.

ignore_impstatus
: logical; if FALSE, HWE p-value and callrate filters are applied only to genotyped SNPs, and imputation quality filters only to imputed SNPs. If TRUE, the filters are applied to all SNPs regardless of the imputation status.

minimal_impQ_value, maximal_impQ_value
: numeric; the minimal and maximal possible (i.e. non-invalid) imputation quality values.

logI, logN
: progress indicators used by QC_series: irrelevant for users.

plot_effectsizes, plot_SE, plot_SK
: logical; additional plot options for QC_series. The arguments toggle the creation of plots comparing the effect-size distribution, precision and skewness vs. kurtosis of all successfully QC'ed datasets, respectively. See plot_distribution, plot_precision and plot_skewness for more information.

lim_effectsizes
: specifies the y-axis range of the effect-size distribution plot.

label_SE
: logical; should the datapoints in the precision plot be labeled?

label_SK
: character string; determines whether the datapoints in the skewness vs. kurtosis plot are labeled. Options are "none", "all", or "outliers" (outliers only).

save_filtersettings
: logical; saves the filtersettings used by the high-quality filter to a file 'Check_filtersettings.txt' in the output directory. If a file of that name already exists, the settings are added to the end (i.e. it updates rather than overwrites existing files). The file can be used as ini file by filter_GWAS.

...
: in QC_series: arguments passed to QC_GWAS; in QC_GWAS: arguments passed to read.table when importing the dataset.

Details

The full quality-control carried out by QC_GWAS consists of 5 phases. The function takes a single dataset (or, rather, the location and filename of a single dataset) and runs it through the following phases:

- 1: Importing the dataset
- 2: Checking data integrity
- 3: Checking alleles
- 4: Generating QC statistics and graphs

- 5: Saving the output

Phase 1: importing the dataset

GWAS results files come in a variety of formats, so QC_GWAS is flexible about loading data. It uses an autoloader function (load_GWAS) to unpack .zip or .gz files and determine the column-separator used in the file. See the section 'Requirement for the input dataset' for more information.

Next, the function attempts to translate the dataset's column names (the header) to standard names, so that it knows what type of data a column contains. This is done by comparing the column names to a translation table (specified in the header_translations argument). See translate_header for more information.

Note that only the SNP ID, alleles, effect-size and standard error columns are required. The absence of other standard columns (chromosome, position, strand, allele frequency, HWE p-value, callrate, imputation quality, imputation status and used for imputation) will not cause the QC to abort. Instead, a warning is printed on screen and in the log file, and a dummy column filled with NA values is added to the dataset.

It is therefor important to check the log file: if a standard column is present but not identified (because it is missing or misspelled in the translation table) the QC will continue, but is unable to check/use the data inside. The unidentified column will be reported in the columns_unidentified value of the QC_GWAS return or in the "QC_checklist.txt" file generated by QC_series.

Phase 2: checking data integrity

The purpose of phase 2 is to ensure that the dataset *can* be QC'ed: that that all SNPs have the required data and that all columns contain only valid (or missing) values.

The first step is to remove SNPs that won't be used: monomorphic SNPs and (if specified by the arguments) allosomal, pseudo-autosomal and mitochondrial SNPs. The function considers SNPs monomorphic if they have a missing or invalid other (non-effect) allele, identical alleles or an allele-frequency of 1 or 0.

The second step is to check the imputation status column with the function convert_impstatus. See the section 'Requirement for the input dataset' for more information. Imputation status is one of the most important variables in the dataset: if unknown, the HWE p-value, callrate and imputation quality won't be used (unless ignore_impstatus is TRUE), as the function cannot determine which SNPs are imputed and which are not. For this reason, if convert_impstatus is unable to translate any character string in the column, the QC will abort.

The third step carries out three tests for all other standard variables:

- Does the column contain the correct type of data (integer, numeric or character)?
- How many values are missing (NA)?
- How many values are invalid (= impossible)?

The exact nature of the three tests differs per variable: see the documentation file in "R\library\QCGWAS\doc" for more detail.

The presence of the wrong data-type will cause the QC to abort. Wrong data-type indicates either a problem in the file itself, or with the way it was imported (in which case it is most likely due to a mistranslated header).

The final step is the removal of the invalid values and of unusable SNPs. The variables MARKER, EFFECT_ALLELE, OTHER_ALLELE, EFFECT and STDERR are considered crucial. SNPs with missing or invalid values in any of these variables are removed the dataset. Missing values in the other variables are ignored, while invalid values are set to NA.

Phase 3: checking alleles

This phase has three functions:

- To check if the correct alleles are reported for each SNP
- To check if the allele-frequency is reported for the correct (effect) allele
- To ensure that SNPs are aligned to the positive strand and use the same effect-allele in all post-QC datasets

This is achieved by comparing the data to a reference, using the function `match_alleles`. First, all SNPs are switched to the positive strand (the alleles are converted to their opposing base and the strand-value is set to `"+"`). If there are SNPs whose allele pair doesn't match the reference, `match_alleles` assumes the information in the strand column is absent or incorrect, and will also switch those SNPs to the other (presumably positive) strand. This step is referred to as strand-switching in QC output, and is independent from the negative-strand SNP conversion. It is therefor possible that a SNP is switched twice: once because the strand-column indicates it is on the negative strand, and twice because of a mismatch. This is referred to as double strand-switch in the output, and indicates either the wrong value in the strand column, or a mismatch with the reference. In the latter case, it will most likely be picked up in the next step.

If the strand-switch does not fix the mismatch, the SNPs are counted in the QC output as mismatches. Depending on the `remove_mismatches` arguments, the SNPs will either be removed from the dataset, or left in but excluded from the further tests of the allele-matching.

Next, `match_alleles` "flips" SNPs so that their effect allele matches the reference minor allele. This ensures that a SNP will have the same effect allele in all post-QC datasets.

`match_alleles` also counts the number of SNPs with a strand-independent allele configuration (A/T or C/G; these are designated as "ambiguous SNPs"), and the subset of those with an allele-frequency that is substantially different from the reference ("suspect SNPs"). If a substantial proportion of ambiguous SNPs is suspect, it indicates that the strand information is incorrect. In our experience, a regular, 2.5M SNP dataset usually consists of 15% ambiguous SNPs, of which a few dozen will be suspect.

The function also counts the number of SNPs whose allele frequency differs from the reference by more than a set amount (`threshold_difEAF`). If the relevant `remove_diffEAF` argument is `TRUE`, these SNP will be excluded from the dataset after the allele-matching is finished.

The final step is to correlate the reported allele-frequency against the reference. If allele-frequency is reported for the correct (effect) allele, the correlation should be close to 1. If the outcome is close to `-1`, the reported frequency is that of the other allele. Depending on the plot settings, a scatter plot of reported vs. reference frequency is made for all SNPs, and for the subsets of ambiguous and non-ambiguous SNPs.

The standard and alternative allele reference

The above steps describe what happens when the dataset is compared to a single reference. However, we found that many GWAS datasets contain SNPs not present in our standard HapMap reference, so we added a second, flexible reference that can be updated with any unknown SNPs the QC encounters.

SNPs that are not found in either reference are converted to the positive strand, and "flipped" if their allele frequency is > 0.50. If `update_alt` is `TRUE`, these SNPs are then added to the alternative reference and saved under the name `update_savename`. There are a few caveats to this system: see the section 'Updating the alternative reference' for details.

Phase 4: generating QC statistics and graphs

At this stage, no further changes will be made to the dataset (except, optionally, to recalculate missing p-values). The function will now start to calculate the QC statistics and generate the important graphs. These are:

- Create histograms of variable distribution (optional)

- Check p-values by correlating them to a p calculated from the effect-size and standard-error (via the check_P function).
- Recalculate missing/invalid p-values (optional)
- Calculate QC statistics
- Create QQ and Manhattan plots (optional, see QC_plots function for more information).

Phase 5: saving the output

A series of tables is added to the bottom of the log file, reporting the QC statistics and the data distribution. If save_final_dataset is TRUE, the post-QC data will be exported as a .txt file. The column names and format of that file can be specified by the out arguments.

Value

The most important output of the QC is the log file. See the section 'QC output files' for more details. This section only describes the function return within R.

QC_series returns a single, invisible, logical value, indicating whether the alternative allele-reference has been updated.

QC_GWAS returns an object of class 'list'. If the QC was not successful, this list contains only five of the following components (QC_successful, filename_input, filename_output, all_ref_changed, effectsize_return). If it was, it will contain all of them:

QC_successful
: logical; indicates whether the QC was completed. If FALSE, the function was either unable to load the dataset, encountered an unexpected datatype, or removed all SNPs during the QC. The log file and screen output will indicate what triggered the abort.

filename_input, filename_output
: the filenames of the dataset pre- and post-QC respectively.

sample_size, sample_size_HQ
: the highest reported sample size for all SNPs and high-quality SNPs only, respectively.

lambda, lambda_geno, lambda_imp
: the lambda values of all, genotyped and imputed SNPs, respectively.

SNP_N_input
: the number of SNPs in the original dataset.

SNP_N_input_monomorphic
: the number of SNPs removed because they are monomorphic.

SNP_N_input_monomorphic_identic_alleles
: the subset of above that had identical alleles, but allele-frequencies that were not 0 or 1.

SNP_N_input_chr
: the number of SNPs removed because they were X-chromosomal, Y-chromosomal, pseudo-autosomal or mitochondrial (depends on the remove-arguments). If all remove arguments were set to FALSE, this returns NA.

SNP_N_preQC
: the number of SNPs that entered phase 2b (i.e. after removal of the monomorphic and excluded-chromosome SNPs).

SNP_N_preQC_unusable
: the number of SNPs removed in phase 2d, due to missing or invalid crucial variables.

SNP_N_preQC_invalid
: the number of SNPs with invalid, non-crucial values in phase 2d.

SNP_N_preQC_min
: the number of negative-strand SNPs in phase 2d.

SNP_N_midQC
: the number of SNPs in the dataset during allele-matching (phase 3).

SNP_N_midQC_min
: the number of negative strand SNPs in phase 3.

SNP_N_midQC_min_std, SNP_N_midQC_min_alt,SNP_N_midQC_min_new
: the number of negative strand SNPs matched against, respectively, the standard allele reference, the alternative allele reference or neither.

SNP_N_midQC_strandswitch_std, SNP_N_midQC_strandswitch_alt
: the number of SNPs that were strand-switched because of a mismatch with the standard and alternative allele reference, respectively.

SNP_N_midQC_strandswitch_std_min, SNP_N_midQC_strandswitch_alt_min
: the subset of previous that were negative-strand SNPs. NOTE: at this point in the QC, negative-strand SNPs have already been converted to the positive strand, i.e. they should *not* appear in this category. If they do, there is a problem with the reported strand, or with the reference table.

SNP_N_midQC_mismatch
: the number of SNPs that were still mismatching after the strand-switch.

SNP_N_midQC_mismatch_std, SNP_N_midQC_mismatch_alt
: subset of previous that were matched with the standard and alternative allele reference, respectively.

SNP_N_midQC_mismatch_std_min, SNP_N_midQC_mismatch_alt_min
: subset of previous that are negative-strand SNPs.

SNP_N_midQC_flip_std, SNP_N_midQC_flip_alt, SNP_N_midQC_flip_new
: Number of SNPs that were flipped (had their alleles reversed) when matched against, respectively, the standard allele reference, the alternative allele reference or neither.

SNP_N_midQC_ambiguous
: the number of ambiguous SNPs

SNP_N_midQC_ambiguous_std, SNP_N_midQC_ambiguous_alt, SNP_N_midQC_ambiguous_new
: subset of ambiguous SNPs matched against, respectively, the standard allele reference, the alternative allele reference or neither.

SNP_N_midQC_suspect
: the subset of ambiguous SNPs whose allele frequencies differ strongly from those in the reference.

SNP_N_midQC_suspect_std, SNP_N_midQC_suspect_alt
: the subsets of previous matched against the standard and alternative allele reference, respectively.

SNP_N_midQC_diffEAF
: the number of SNPs whose allele frequency differs strongly from the reference.

SNP_N_midQC_diffEAF_std, SNP_N_midQC_diffEAF_alt
: subset of previous that were matched with the standard and alternative allele reference, respectively.

SNP_N_postQC
: the number of SNPs in the final dataset.

SNP_N_postQC_geno, SNP_N_postQC_imp
: the number of genotyped and imputed SNPs in the final dataset.

SNP_N_postQC_invalid
: the number of SNPs with invalid values remaining in the final dataset. Note: any invalid values have already been changed to NA at this point. This merely counts how many of those SNPs are still in the dataset.

`SNP_N_postQC_min`
: the number of negative-strand SNPs in the final dataset. Note: all SNPs have been switched to the positive strand at this point. This merely counts how many of those SNPs are still in the dataset.

`SNP_N_postQC_HQ`
: the number of high-quality SNPs in the final dataset.

`fixed_HWE, fixed_callrate, fixed_sampleN, fixed_impQ`
: logical or character string; are HWE p-values, callrates, sample-size and imputation quality values identical for all relevant SNPs? If `TRUE`, it indicates that the parameters have not been calculated and are dummy values. If a parameter fails the threshold test, this returns `"insuf. data"` (`"no data"` for sample size).

`effect_25, effect_median, effect_75`
: the quartile values of the effect-size distribution.

`effect_mean`
: the mean of the effect-size distribution.

`SE_median, SE_median_HQ`
: the median standard error of all SNPs and high-quality SNPs only, respectively.

`skewness, kurtosis`
: the skewness and kurtosis value of the dataset.

`skewness_HQ, kurtosis_HQ`
: the skewness and kurtosis value for high-quality SNPs only.

`all_ref_std_name, all_ref_alt_name`
: the names used for the standard and alternative allele-references in the output.

`all_MAF_std_r, all_MAF_alt_r`
: allele-frequency correlation with the standard and alternative allele references.

`all_ambiguous_MAF_std_r, all_ambiguous_MAF_alt_r`
: allele-frequency correlations for the subset of ambiguous SNPs in the standard and alternative allele references, respectively.

`all_non_ambig_MAF_std_r, all_non_ambig_MAF_alt_r`
: allele-frequency correlations for the subset of non-ambiguous SNPs in the standard and alternative allele references, respectively.

`all_ref_changed`
: logical; has an updated alternative allele reference been saved?

`effectsize_return`
: logical; is a vector of high-quality effect-sizes returned in `effectsizes_HQ`?

`effectsizes_HQ`
: if `effectsize_return` is `TRUE`, a vector of 1000 high-quality effect-sizes; if not, `NULL`. If a dataset contains less than 1000 high-quality SNPs, the vector is padded with NA's to bring it to 1000 values.

`pvalue_r`
: the correlation between reported and calculated p-values.

`visschers_stat, visschers_stat_HQ`
: the Visscher's statistic for all SNPs and high-quality SNPs only, respectively.

`columns_std_missing`
: the names of any missing, standard columns: if no columns are missing, this returns `0`.

`columns_std_empty`
: the names of any empty, standard columns: if no columns are empty, this returns `0`.

`columns_unidentified`
: the names of any unidentified columns in the input dataset. If none, this returns `0`.

`outcome_useFRQ, outcome_useHWE, outcome_useCal, outcome_useImp, outcome_useMan` logical; indicates whether the variable passed the threshold test.

`...` the remaining 'setting' components return the the actual filter settings used in the QC (i.e. taking into account whether the variables passed the threshold test).

QC output files

The results of the QC are reported in a variety of .txt and .png files saved in `dir_output`. The files use the same output name as the dataset, with an extension to indicate their contents (i.e. '_log.txt', '_graph_QQ.png'). The .txt files are tab-delimited and are best viewed in a spreadsheet program. The most important one is the log file. This file summarizes the results of the QC and the data inside the file.

The log file

The top of the file is table of log entries, reporting any (potential) problems encountered during the QC. Some of these are just routine updates; the removal of SNPs with missing data, for example. However, do check the other entries. These report important but non-fatal problems, relating to crucial missing data or invalid data. In such a case, and provided the QC did not abort, the affected SNPs will have been exported to a .txt file before being excluded, so the user can inspect them without having to reload the entire dataset. The .txt's are:

`[filename_output]_SNPs_invalid_allele2.txt`

`[filename_output]_SNPs_duplicates.txt`

`[filename_output]_SNPs_removed.txt`

`[filename_output]_SNPs_improbable_values.txt`

(Note: the names of the files are slightly confusing: the "SNPs_removed" file contains all SNPs removed in phase 2d. This does not include monomorphic SNPs, or SNPs from excluded chromosomes, as these are removed in phase 2a. Also, the "SNPs_improbable_values" file does not include SNPs with invalid values for crucial variables, as these are already in the "SNPs_removed" file.)

Another important but non-fatal problem is missing columns. `QC_GWAS` uses a translation table to determine the contents of a column. If the translation table is incomplete, or contains a typo, the function will be unable to translate (and therefor use) a column. If this involves, say, callrate, it merely means the function cannot apply the callrate filters, but the absence of p-values or imputation status will disable many features of the QC. If you know that a data column is present, yet the log reports it missing, check the translation table. The `QC_series` checklist output and the `columns_unidentified` value of the `QC_GWAS` return report the names of any unidentified columns in the dataset.

If the QC aborts, the log file should give some indication why this occurred. However, if it was successful, there will be several other tables in the log file.

The second table reports the number of SNPs in the dataset at various stages of the QC; as well as how many (and for what reason) SNPs were removed.

The third table gives an overview of the data itself. It reports how many values were missing and invalid per variable in both the pre- and post-QC datasets, as well as the quartile and mean values of the post-QC data. A few notes on the nomenclature: invalid values will have been removed (for crucial variables) or set to missing (for non-crucial variables) in stage 2d. The post-QC 'invalid' column merely records how many of these SNPs remain in the dataset. 'Unusable' values are the missing and invalid values combined (shown as percentage of the total data). Finally, pre-QC refers to the dataset during stage 2b-c, but after monomorphic SNPs and SNPs from excluded chromosomes (stage 2a) have been removed.

The fourth table reports on the allele-matching in phase 3. The concepts are explained in 'details' and the `match_alleles` function; here we just mention what the user needs to pay attention to.

Strand-switching counts SNPs whose strand was switched because it mismatched with the reference. As many cohorts do not add strand data (or set every SNP to "+"), the presence of such SNPs is not a red flag by itself. However, if there are mismatching SNPs (the subset of strand-switched SNPs that could not be fixed), there is a problem with the allele data (or, possibly, a triallelic SNP). Check the [filename_output]_SNPs_mismatches-[ref-name].txt file to see the affected SNPs.

Another red flag is if there are strand-switched SNPs in a dataset that *also* contains negative-strand SNPs (i.e. the cohort included real strand data, rather just setting it to "+" for all SNPs). Negative-strand SNPs are converted to the positive strand beforehand, so they should not appear in this step (if they do, they are counted in the "double strand-switch" entry, but that is of minor importance). The real problem is that if a cohort includes negative-strand SNPs (i.e. real strand data), and there are still strand-switches, the strand data must be incorrect. Whether the strand-switches and the negative-strand SNPs overlap is unimportant.

The third possible problem is when a large proportion of ambiguous SNPs is suspect: it indicates that they are on the wrong strand.

Finally, a large number of SNPs with a deviating allele frequency indicates either that the frequency is reported for the wrong allele (see below) or that the dataset population does not match that of the reference.

The fifth table reports the QC outcome statistics. The p-value and allele-frequency correlations should be close to 1. An allele-frequency correlation of -1 means that the frequency was reported for the wrong (non-effect) allele. As for the p-value correlaction: in a typical GWAS dataset, the expected and observed p-values should correlate perfectly. If this isn't the case, it means either that a column was misidentified when loading the dataset or that the wrong values were used when generating the data.

The sixth table reports how many SNPs were removed by the various QQ plot filters.

The seventh table reports the chromosomes and alleles present in the final dataset.

The eighth table counts invalid values in the pre- and post-QC files for several variables. 'Extreme p' is a value that is only used when calculate_missing_p is TRUE. Any newly-calculated p-values that are < 1E-300 will be set to 1E-300, in order to exclude any values of 0 (1E-300 is close to the smallest numeric value that R can handle safely).

The final four tables contain the settings of the QC.

The QC_series output

QC_series saves a checklist, showing the most important QC stats of the various files side by side, and (depending on the plot-arguments) several graphs comparing effect-size distribution, precision and skewness vs. kurtosis of the QC'ed files.

Output header

The standard-format values used by out_header are:

- "standard" retains the default column names used by QC_GWAS.
- "original" restores the column names used in the input file.
- "old" uses the default column names of pre-v1.0b versions of QCGWAS.
- "GWAMA", "PLINK", "META" and "GenABEL" set the column names to those use by the respective programs. Note that META's alleleB corresponds to EFFECT_ALL in QC_GWAS.

Requirement for the input dataset

QC_GWAS will automatically unpack .gz and .zip files, provided the filename includes the extension of the packed file. For example, if the data file is named "data1.csv", the zip file should be "data1.csv.zip". If it is named "data1.zip", QC_GWAS won't be able to "call" the file inside.

QC_GWAS is flexible when it comes to file-format. By default, it can open datasets with a variety of column separators and NA values (the user can specify these via the column_separators and na.strings arguments). Read the documentation of the auto-loader function load_GWAS for more information.

Chromosome values can be coded as numeric or character: values of "X", "Y", "XY" and "M" will automatically be converted to 23, 24, 25 and 26, respectively.

By default, imputation status is coded as integers 0 (genotyped) and 1 (imputed). As of version 1.0-4, imputation status will always be translated using the imputed_T, imputed_F and imputed_NA arguments. This means that the user must specify values for these arguments, even when the dataset already uses the standard format. Because of the importance of imputation status, if the function is unable to translate character values, the QC will abort.

The minimal and maximal valid imputation quality values are determined by the minimal_impQ_value and maximal_impQ_value arguments.

Standard errors and p-values of 0 are considered invalid and removed in phase 2d, while values of -1 will be set to NA. Effect sizes of -1 are accepted, unless the standard error and/or the p value are also -1, in which case it is also set to NA.

Filter arguments

QC_GWAS has three sets arguments relating to filters: the arguments for the HQ (high-quality), QQ (plot) and NA (missing values) filters. The HQ and QQ arguments work mostly in the same way, but there are a few key differences.

The high-quality arguments accept single, numeric values that determine the minimal values of allele-frequency (FRQ), HWE p-value (HWE), callrate (cal) and imputation quality (imp) for a SNP to be of high-quality. The high-quality filter is used for the effect-size boxplot and the Manhattan plot, as well as several QC stats.

The QQ arguments accept a vector of max. 5 numeric values that are applied sequentially as filters in the QQ plots.

Both of these use the NA filter argument(s) to determine whether to exclude or ignore missing values.

Neither filter is used to *remove* SNPs; they merely exclude them from several QC tests. Both HQ and QQ filter criteria are only applied if the variable passed the threshold test, i.e. if there are sufficient non-missing, non-invalid values for the filter to be applied (see the use_threshold argument for details). It's pointless to filter an empty column.

If ignore_impstatus is FALSE (default), the imputation-quality criterion is only applied to imputed SNPs, and the HWE p-value and callrate criteria only to genotyped SNPs. If TRUE, the filters are applied to all SNPs, regardless of the imputation status.

The allele-frequency filters are two-sided: when set to value x, SNPs with frequency < x or > 1 - x are excluded.

To filter missing values only, set the filter to NA and the corresponding NA filter argument to TRUE. To disable entirely, set to NULL (this means the NA-filter setting is ignored as well).

The differences between the HQ and QQ filters are:

The HQ filter arguments accept a single value, the QQ filters can accept up to 5.

The HQ filter is a single filter: a SNP needs to meet all relevant criteria to be considered high-quality. The QQ filter values are applied separately.

The QQ filter has an additional feature: if passed a value x > 1, it will calculate a filter value of x / sample size. This is to allow size-based filtering of allele frequencies. Note that this filter uses the sample size listed for that specific SNP. If the sample size is missing, the relevant NA-filter setting is used to determine whether it should be excluded.

Updating the alternative reference

There are two drawbacks to the way the function updates the alternative reference file. One is a technical issue, but the other can affect the QC of subsequent files.

Firstly, the argument `update_alt` has a slightly misleading name: the alternative-reference *file* is updated, but the reference inside the R memory is not. If the user wants to do further QCs with the updated reference, (s)he will have to manually reload the updated file into R.

This is caused by the way R handles data-alterations that occur inside a function. Any changes made to data last only for the duration of the function. Once the function terminates, the memory reverts to its original state. In other words: the allele reference is updated inside `QC_GWAS`, but goes back to the pre-QC state once the QC is finished. `QC_series` deals with this by automatically reloading the reference file whenever it is updated, but, again, once the function terminates it will revert to its original state.

The second drawback is that the content of the alternative reference is arbitrary, depending on which file an unknown SNP is encountered in first. For example, suppose that SNP rs31 has alleles A and G, an allele frequency that varies around 0.5, and does not appear in the standard reference. When it is added to the alternative reference, the allele listed as the minor one depends entirely on the allele frequency in the first file it is encountered in.

More seriously, if the information in the first file is incorrect, the SNP may be strand-switched or excluded in subsequent files because it does not match the (incorrect) reference. This is another reason why it is important to check the log files: if there is a problem with a datafile's strand, alleles or allele-frequency, and the alternative reference was updated, the incorrect data may have been added to the reference. If so, one should go back to a previous reference file. The argument `backup_alt` is useful for this, though note that `QC_series` only does this the first time the reference is updated.

Also, if one wants to QC a large number of files for a meta-GWAS, one should use the same alternative allele reference file (and let `QC_GWAS` update it) for every file, otherwise it is possible that rs13 may have a different effect alleles in some post-QC files.

See Also

For the plots created by `QC_series`: `plot_distribution`, `plot_precision` and `plot_skewness`.

For loading and preparing a GWAS dataset: `load_GWAS`, `translate_header`, `convert_impstatus`.

For carrying out separate steps of `QC_GWAS`: `match_alleles`, `check_P`, `QC_plots`.

Examples

```
## For instructions on how to run QC_GWAS and QC_series
## check the quick start guide in /R/library/QCGWAS/doc
```

`QC_histogram` *Histogram(s) of expected and observed data distribution*

Description

`QC_histogram` creates two histograms: one showing the observed data distribution of a numeric variable, and one showing the expected distribution. It includes the option to filter the data with the high-quality filter. `histogram_series` generates a series of such histograms for multiple filter settings.

Usage

```
QC_histogram(dataset, data_col = 1,
             save_name = "dataset", save_dir = getwd(),
             export_outliers = FALSE,
             filter_FRQ = NULL, filter_cal = NULL,
             filter_HWE = NULL, filter_imp = NULL,
             filter_NA = TRUE,
             filter_NA_FRQ = filter_NA, filter_NA_cal = filter_NA,
             filter_NA_HWE = filter_NA, filter_NA_imp = filter_NA,
             breaks = "Sturges",
             graph_name = colnames(dataset)[data_col],
             header_translations, check_impstatus = FALSE,
             ignore_impstatus = FALSE,
             T_strings = c("1", "TRUE", "yes", "YES", "y", "Y"),
             F_strings = c("0", "FALSE", "no", "NO", "n", "N"),
             NA_strings = c(NA, "NA", ".", "-"), ...)
histogram_series(dataset, data_col = 1,
   save_name = paste0("dataset_F", 1:nrow(plot_table)),
   save_dir = getwd(), export_outliers = FALSE,
   filter_FRQ = NULL, filter_cal = NULL,
   filter_HWE = NULL, filter_imp = NULL,
   filter_NA = TRUE,
   filter_NA_FRQ = filter_NA, filter_NA_cal = filter_NA,
   filter_NA_HWE = filter_NA, filter_NA_imp = filter_NA,
   breaks = "Sturges",
   header_translations, ignore_impstatus = FALSE,
   check_impstatus = FALSE,
   T_strings = c("1", "TRUE", "yes", "YES", "y", "Y"),
   F_strings = c("0", "FALSE", "no", "NO", "n", "N"),
   NA_strings = c(NA, "NA", ".", "-"),
   ...)
```

Arguments

dataset	vector or table containing the variable of interest.
data_col	name or number of the column of dataset containing the variable of interest.
save_name	for QC_histogram, a character string; for histogram_series, a vector of character strings; specifying the filename(s) of the graph, *without* extension.
save_dir	character string; the directory where the output files are saved. Note that R uses *forward* slash (/) where Windows uses the backslash (\).
export_outliers	logical or numeric value; should outlying entries (which are excluded from the plot) be exported to an output file? If numeric, the number specifies the max. number of entries that is exported.
filter_FRQ, filter_cal, filter_HWE, filter_imp	Filter threshold-values for allele-frequency, callrate, HWE p-value and imputation quality, respectively. Passed to HQ_filter. QC_histogram takes only single values, but histogram_series accepts vectors as well (see 'details').
filter_NA	logical; if TRUE, then missing filter variables will be excluded; if FALSE, they will be ignored. QC_histogram takes only single values, but histogram_series accepts vectors as well (see 'Details'). filter_NA is the default setting for all

variables; variable-specific settings can be specified with the following arguments.

`filter_NA_FRQ, filter_NA_cal, filter_NA_HWE, filter_NA_imp`
: logical; variable-specific settings for `filter_NA`. These arguments are passed to `HQ_filter`.

`breaks`
: argument passed to `hist`; determines the cell-borders in the histogram.

`graph_name`
: character string; used in the title of the plot.

`header_translations`
: translation table for column names. See `translate_header` for more information. If the argument is left empty, `dataset` is assumed to use the standard column-names used by `QC_GWAS`.

`check_impstatus`
: logical; should `convert_impstatus` be called to convert the imputation-status column into standard values?

`ignore_impstatus`
: logical; if `FALSE`, HWE p-value and callrate filters are applied only to genotyped SNPs, and imputation quality filters only to imputed SNPs. If `TRUE`, the filters are applied to all SNPs regardless of the imputation status.

`T_strings, F_strings, NA_strings`
: arguments passed to `convert_impstatus`.

`...`
: in `histogram_series`: arguments passed to `QC_histogram`; in `QC_histogram`, arguments passed to `hist`.

Details

`histogram_series` accepts multiple filter-values, and passes these one by one to `QC_histogram` to generate a series of histograms. For example, specifying:

`filter_FRQ = c(0.05, 0.10), filter_cal = c(0.90, 0.95)`

will generate two histograms. The first excludes SNPs with allele frequency < 0.05 or callrate < 0.90; the second allele frequency < 0.10 or callrate < 0.95. The same principle applies to the `NA_filter` settings. If the vectors submitted to the filter arguments are of unequal length, the shorter vector will be recycled until it equals the length of the longer (if possible). To filter missing values only, set the filter to `NA` and the corresponding NA-filter argument to `TRUE`. Setting the filter argument to `NULL` will disable the filter entirely, regardless of the NA filter setting.

Value

Both functions return an invisible value `NULL`.

See Also

For creating QQ plots: `QQ_plot`.

Examples

```
data("gwa_sample")

QC_histogram(dataset = gwa_sample, data_col = "EFFECT",
             save_name = "sample_histogram",
             filter_FRQ = 0.01, filter_cal = 0.95,
             filter_NA = FALSE,
             graph_name = "Effect size histogram")
```

```
histogram_series(dataset = gwa_sample, data_col = "EFFECT",
           save_name = "sample_histogram",
           filter_FRQ = c(NA, 0.01, 0.01),
           filter_cal = c(NA, 0.95, 0.95),
           filter_NA = c(FALSE, FALSE, TRUE))
```

QC_plots *QQ and Manhattan plots*

Description

This function creates the most important graphs of the QC: the QQ plots and the Manhattan plot. It also calculates lambda, and determines the effect of the filters.

Usage

```
QC_plots(dataset,
         plot_QQ = TRUE, plot_Man = TRUE,
         FRQfilter_values = NULL, FRQfilter_NA = filter_NA,
         HWEfilter_values = NULL, HWEfilter_NA = filter_NA,
         calfilter_values = NULL, calfilter_NA = filter_NA,
         impfilter_values = NULL, impfilter_NA = filter_NA,
         impfilter_min = min(dataset$IMP_QUALITY, na.rm = TRUE),
         manfilter_FRQ = NULL, manfilter_HWE = NULL,
         manfilter_cal = NULL, manfilter_imp = NULL,
         filter_NA = TRUE,
         plot_cutoff_p = 0.05, plot_names = FALSE,
         QQ_colors = c("red", "blue", "orange", "green3", "yellow"),
         plot_QQ_bands = FALSE,
         save_name = "dataset", save_dir = getwd(),
         header_translations, use_log = FALSE,
         check_impstatus = FALSE, ignore_impstatus = FALSE,
         T_strings = c("1", "TRUE", "yes", "YES", "y", "Y"),
         F_strings = c("0", "FALSE", "no", "NO", "n", "N"),
         NA_strings = c(NA, "NA", ".", "-"))
```

Arguments

`dataset` vector of p-values or a data frame containing the p-value column and (depending on the settings) columns for chromosome number, position, the quality parameters, sample size and imputation status.

`plot_QQ, plot_Man` logical; should QQ and Manhattan plots be saved?

`FRQfilter_values, HWEfilter_values, calfilter_values, impfilter_values` numeric vectors; the threshold values for the QQ plot filters. The filters are for allele-frequency, HWE p-values, callrate and imputation-quality parameters, respectively. A maximum of five values can be specified per parameter.

- Set to `NULL` to disable the QQ filter for that parameter.
- To filter missing values only, set the filter value to `NA` and the corresponding `filter_NA` argument to `TRUE`.

- The allele-frequency filter is two-sided: for a filter-value of x, it will exclude entries with freq < x and freq > 1 - x.
- Values >= 1 will be divided by the SNP's sample size. This allows sample-size dependent filtering of allele frequencies. Note that this uses the sample size reported in the sample-size column for that specific SNP. SNPs without sample size will be excluded if the corresponding filter_NA argument is TRUE and ignored if it is FALSE.

FRQfilter_NA, HWEfilter_NA, calfilter_NA, impfilter_NA, filter_NA
: logical; should the filters exclude (TRUE) or ignore (FALSE) missing values? filter_NA is the default setting, the others allow variable-specific settings.

impfilter_min
: numeric; the lowest possible value for imputation-quality. This argument is currently redundant, as it is calculated automatically.

manfilter_FRQ, manfilter_HWE, manfilter_cal, manfilter_imp
: single, numeric values; the filter-settings for allele-frequency, HWE p-values, callrate and imputation quality respectively, for the Manhattan plot. The arguments are passed to HQ_filter. To filter missing values only, set to NA and the corresponding filter_NA argument to TRUE. To disable filtering entirely, set to NULL.

plot_cutoff_p
: numeric; the threshold of p-values to be shown in the QQ & Manhattan plots. Higher (less significant) p-values are excluded from the plot. The default setting is 0.05, which excludes 95% of data-points. It's not recommended to increase the value above 0.05, as this may dramatically increase running time and memory usage.

plot_names
: argument currently redundant.

QQ_colors
: vector of R color-values; the color of the QQ filter-plots. The unfiltered data is black by default. This argument sets the colors of the least (first value) to most (last value) stringent filters. (For this setting, filter values >= 1 (i.e. sample-size based filtering) are considered less stringent than values < 1.)

plot_QQ_bands
: logical; should probability bands be added to the QQ plot?

save_name
: character string; the filename, *without* extension, for the graphs.

save_dir
: character string; the directory where the graphs are saved. Note that R uses *forward* slash (/) where Windows uses the backslash (\).

header_translations
: translation table for column names. See translate_header for more information. If the argument is left empty, dataset is assumed to use the standard column-names of QC_GWAS.

use_log
: argument used by QC_GWAS; redundant when QC_plots is used separately.

check_impstatus
: logical; should the imputation-status column be passed to convert_impstatus?

ignore_impstatus
: logical; if FALSE, HWE p-value and callrate filters are applied only to genotyped SNPs, and imputation quality filters only to imputed SNPs. If TRUE, the filters are applied to all SNPs regardless of the imputation status.

T_strings, F_strings, NA_strings
: arguments passed to convert_impstatus.

Details

The function QC_plots grew out of phase 4 of QC_GWAS. It carries out three functions, hence the vague name: it calculates lambda, it applies the QQ filters, and it creates the QQ and Manhattan plots (a separate function is available for creating regional-association plots: see below). The function schematic is as follows:

- Preparing the dataset: this step involves translating the dataset header to the standard column-names (by identify_column) and converting imputation status (by convert_impstatus). Both steps are optional, and are disabled by default. If the function cannot identify the imputation status column, it will generate a warning message and disable the imputation-status dependent filters.
- Calculating the QC stats: here it generates the filters an calculates how many SNPs are removed. Lambda is also calculated at this point.
- Creating a QQ graph of every variable for which filters have been specified. Every graph contains an unfiltered plot, plus plots for every effective filter. ("Effective" means "excludes more SNPs than the previous, less-stringent filter".)
- Creating the Manhattan plot. The default Manhattan plot covers chromosomes 1 to 23 (X). Fields for XY, Y and M are added when such SNPs are present.

Value

An object of class 'list' with the following components:

`lambda` vector of the lambda values of all SNPs, genotyped SNPs and imputed SNPs, respectively.

`ignore_impstatus` logical value indicating whether imputation status was used when applying the filters.

`FRQfilter_names, HWEfilter_names, calfilter_names, impfilter_names` character vectors naming the specified QQ filters.

`FRQfilter_N, HWEfilter_N, calfilter_N, impfilter_N` numeric vectors; the number of SNPs removed by the specified filters. Note that the filters are sorted before being applied, so the order may not match that of the input. Check the `filter_names` output to see the order that was used inside QC_plots.

`Manfilter_N` numeric; the number of SNPs removed by the Manhattan filter. This does not include those SNPs removed because they lacked p or chromosome/position-values, or failed the p-cutoff threshold.

Note

By default, QC_plots expects `dataset` to use the standard column-names used by QC_GWAS. A translation table can be specified in `header_translations` to allow non-standard names. See translate_header for more information.

The function accepts both integer and character chromosome values. Character values of `"X"`, `"Y"`, `"XY"` and `"M"` are automatically converted to integers. By default, the Manhattan plot shows all autosomal chromosomes and chromosome X. Fields for Y, XY and M are added only when such SNPs are present.

There must be more than 10 p-values at or below the `plot_cutoff_p` threshold for the QQ and Manhattan plots to be created.

See Also

plot_regional for creating a regional association plot.

check_P for comparing the reported p-values to the p expected from the effect size and standard error.

QQ_plot for generating simpler QQ plots.

Examples

```
data("gwa_sample")

QC_plots(dataset = gwa_sample,
  plot_QQ = TRUE, plot_QQ_bands = TRUE, plot_Man = TRUE,
  FRQfilter_values = c(NA, 0.01, 0.05, 3),
  calfilter_values = c(NA, 0.95, 0.99),
  manfilter_FRQ = 0.05, manfilter_cal = 0.95,
  filter_NA = TRUE, save_name = "sample_plots")
```

QQ_plot *QQ plot(s) of expected vs. reported p-values*

Description

QQ_plot generates a simple QQ plot of the expected and reported p-value distribution. It includes the option to filter the data with the high-quality filter. QQ_series generates a series of such QQ plots for multiple filter settings.

Usage

```
QQ_plot(dataset, save_name = "dataset", save_dir = getwd(),
        filter_FRQ = NULL, filter_cal = NULL,
        filter_HWE = NULL, filter_imp = NULL,
        filter_NA = TRUE,
        filter_NA_FRQ = filter_NA, filter_NA_cal = filter_NA,
        filter_NA_HWE = filter_NA, filter_NA_imp = filter_NA,
        p_cutoff = 0.05, plot_QQ_bands = FALSE,
        header_translations,
        check_impstatus = FALSE, ignore_impstatus = FALSE,
        T_strings = c("1", "TRUE", "yes", "YES", "y", "Y"),
        F_strings = c("0", "FALSE", "no", "NO", "n", "N"),
        NA_strings = c(NA, "NA", ".", "-"), ...)
QQ_series(dataset, save_name = "dataset", save_dir = getwd(),
          filter_FRQ = NULL, filter_cal = NULL,
          filter_HWE = NULL, filter_imp = NULL,
          filter_NA = TRUE,
          filter_NA_FRQ = filter_NA, filter_NA_cal = filter_NA,
          filter_NA_HWE = filter_NA, filter_NA_imp = filter_NA,
          p_cutoff = 0.05, plot_QQ_bands = FALSE,
          header_translations,
          check_impstatus = FALSE, ignore_impstatus = FALSE,
          T_strings = c("1", "TRUE", "yes", "YES", "y", "Y"),
          F_strings = c("0", "FALSE", "no", "NO", "n", "N"),
          NA_strings = c(NA, "NA", ".", "-"), ...)
```

Arguments

dataset	a data frame containing the p-value column and (depending on the settings) columns for chromosome number, position, the quality parameters, sample size and imputation status.
save_name	for QQ_plot, a character string; for QQ_series, a vector of character strings; specifying the filename(s) of the graph, *without* extension.
save_dir	character string; the directory where the output files are saved. Note that R uses *forward* slash (/) where Windows uses the backslash (\).
filter_FRQ, filter_cal, filter_HWE, filter_imp	Filter threshold-values for allele-frequency, callrate, HWE p-value and imputation quality, respectively. Passed to HQ_filter. QQ_plot takes only single values, but QQ_series accepts vectors as well (see 'details').
filter_NA	logical; if TRUE, then missing filter variables will be excluded; if FALSE, they will be ignored. QQ_plot takes only single values, but QQ_series accepts vectors as well (see 'Details'). filter_NA is the default setting for all variables; variable-specific settings can be specified with the following arguments.
filter_NA_FRQ, filter_NA_cal, filter_NA_HWE, filter_NA_imp	logical; variable-specific settings for filter_NA. These arguments are passed to HQ_filter.
p_cutoff	numeric; the threshold of p-values to be shown in the QQ plot(s). Higher (less significant) p-values are excluded from the plot. The default setting is 0.05, which excludes 95% of data-points. It's not recommended to increase the value above 0.05, as this may dramatically increase running time and memory usage.
plot_QQ_bands	logical; should probability bands be added to the QQ plot?
header_translations	translation table for column names. See translate_header for more information. If the argument is left empty, dataset is assumed to use the standard column-names of QC_GWAS.
check_impstatus	logical; should the imputation-status column be passed to convert_impstatus?
ignore_impstatus	logical; if FALSE, HWE p-value and callrate filters are applied only to genotyped SNPs, and imputation quality filters only to imputed SNPs. If TRUE, the filters are applied to all SNPs regardless of the imputation status.
T_strings, F_strings, NA_strings	arguments passed to convert_impstatus.
...	arguments passed to plot.

Details

QQ_series accepts multiple filter-values, and passes these one by one to QQ_plot to generate a series of plots. For example, specifying:

```
filter_FRQ = c(0.05, 0.10), filter_cal = c(0.90, 0.95)
```

will generate two plots. The first excludes SNPs with allele frequency < 0.05 or callrate < 0.90; the second allele frequency < 0.10 or callrate < 0.95. The same principle applies to the NA_filter settings. If the vectors submitted to the filter arguments are of unequal length, the shorter vector will be recycled until it equals the length of the longer (if possible). To filter missing values only, set the filter to NA and the corresponding NA-filter argument to TRUE. Setting the filter argument to NULL will disable the filter entirely, regardless of the NA-filter setting.

Value

Both functions return an invisible value NULL.

See Also

QC_plots for generating more complex QQ plots as well as Manhattan plots.

QC_histogram for creating histograms.

check_P for comparing the reported p-values to the p expected from the effect size and standard error.

Examples

```
data("gwa_sample")

QQ_plot(dataset = gwa_sample,
        save_name = "sample_QQ",
        filter_FRQ = 0.01, filter_cal = 0.95,
        filter_NA = FALSE)

QQ_series(dataset = gwa_sample,
          save_name = "sample_QQ",
          filter_FRQ = c(NA, 0.01, 0.01),
          filter_cal = c(NA, 0.95, 0.95),
          filter_NA = c(FALSE, FALSE, TRUE))
```

---

save_log *Log entries for QC_GWAS*

Description

A subroutine for reporting problems encountered while running QC_GWAS. It's not designed for use outside of this context, so we recommend that users do not bother with it. It is described here for the sake of completeness only.

Usage

```
save_log(phaseL, checkL, typeL, SNPL = allSNPs,
         allSNPs = 1L, actionL, noteL = "", fileL)
```

Arguments

phaseL, checkL, typeL	character-strings indicating (very briefly) what occurred to trigger a log entry.
SNPL	integer value indicating the number of SNPs affected.
allSNPs	integer value indicating the total number of SNPs in the dataset.
actionL	character-string describing very briefly the response to situation.
noteL	character-string for longer explanations of what happened.
fileL	character-string of the directory path and the file name, *without* file extension, of the log file.

Details

save_log is used to add entries to the log file the moment they are triggered (as opposed to waiting until the QC concludes and then saving the entire log). In case of a fatal crash, the log file should therefor give some indication of where in the QC it occurred.

save_log does not create the log file (this is done by QC_GWAS): it merely appends entries to the bottom of the specified file, regardless of what that file is. Hence it is not recommended to use the function outside of QC_GWAS.

Value

save_log returns an invisible NULL. The relevant output is the entry added to the log file.

---

switch_strand *Convert alleles to the opposing DNA strand*

Description

This function is a subroutine of QC_GWAS and match_alleles. It converts allele-pairs to the configuration on the opposing DNA strand.

Usage

```
switch_strand(input, strand_col = FALSE)
```

Arguments

input	table with the alleles in column 1 and 2, and (optionally) the strand-information (coded as "+" or "-") in column 3. Note: the alleles must be uppercase characters.
strand_col	logical; if strand-information is present, this switches the sign in column 3 as well.

Value

A table with two or three columns, depending on strand_col.

See Also

match_alleles

Examples

```
data("gwa_sample")

switched_data <- gwa_sample[ , c("MARKER", "EFFECT_ALL",
                                 "OTHER_ALL", "STRAND")]
switched_data[1:10, ]

switched_data[ , 2:4] <- switch_strand(input =
          switched_data[ , 2:4], strand_col = TRUE)
switched_data[1:10, ]
```

`translate_header` *Translate column names into standard names*

Description

This function is used to translate non-standard column names into the standard ones used by QC_GWAS and other functions. It can also translate the standard names into user-specified names (via the `out_header` argument of `QC_GWAS`).

Usage

```
translate_header(header,
    standard = c("MARKER", "CHR", "POSITION", "EFFECT_ALL",
                 "OTHER_ALL", "STRAND", "EFFECT", "STDERR",
                 "PVALUE", "EFF_ALL_FREQ", "HWE_PVAL",
                 "CALLRATE", "N_TOTAL", "IMPUTED",
                 "USED_FOR_IMP", "IMP_QUALITY"),
    alternative)
```

Arguments

`header`	character vector; the header to be translated.
`standard`	character vector; the names `header` should be translated into.
`alternative`	translation table; see 'Details' for more information.

Details

In a nutshell: the `header` argument contains the names you have; the `standard` argument contains the names you want; and `alternative` is the conversion table.

The table in `alternative` should have two columns. The left column contains the standard names; the right column possible alternatives. Only one alternative name should be listed per row. `translate_header` automatically changes the contents of `header` to uppercase, so `standard` and the right column of `alternative` should be uppercase as well.

A sample translation table is provided in the package data folder. It can be loaded via `data("header_translations")`. An editable .txt version can be found in the "R\library\QCGWAS\doc" folder.

Value

`translate_header` returns an object of class 'list' with 6 components:

`header_h`	character vector; the translated header. Unknown columns are included under their old names.
`missing_h`	character vector; the standard column names that were not found. If none, this returns `NULL`.
`unknown_h`	character vector; column names that could not be converted to a standard name. Note that these columns are also included in `header_h`. If none, this returns `NULL`.
`header_N, missing_N, unknown_N`	integer; the lengths of the above three vectors

See Also

header_translations for a sample translation table.

identify_column

Examples

```
sample_data <-
  data.frame(SNP = paste("rs", 1:10, sep = ""),
             chrom = 2,
             effect = 1:10/10,
             misc = NA,
             stringsAsFactors = FALSE)
# Creates a table with four columns:
#   SNP, chrom, effect and misc.

( alt_headers <-
    data.frame(
      standard = c("MARKER", "MARKER", "CHR", "CHR"),
      alternative = c("MARKER", "SNP", "CHR", "CHROM"),
      stringsAsFactors = FALSE) )
# Creates the translation table
#  with the standard names in column 1 and the alternatives
#  in column 2.

( header_info <-
    translate_header(header = names(sample_data),
      standard = c("MARKER", "CHR", "EFFECT"),
      alternative = alt_headers)  )

# Despite not being in the translation table, EFFECT is
#  changed to uppercase because it is present in standard.
#  misc is neither in standard or the translation table, so
#  it is marked as unknown and left unchanged.

names(sample_data) <- header_info$header_h
```

Index