DAPHnE-G

Francesco Pezzini

15 ottobre 2019

Contents

```
library(daphneg2)
```

Introduction

DAPHnE-G or Deep *Arabidopsis* PHenotypical and Environmental GWAS, is an R pipeline for GWAS analysis in *A.thaliana*. It handles a data repository with different phenotypic and environmental parameters and it takes care of the association study from the first data preprocessing to the downstream analysis of results.

The tool is quite modular and flexible and it will suite plant biologists who work with *Arabidopsis*, as DAPHnE-G is taylored specifically for this organism. On the other hand the modularity of the structure could allow the implementation of other data repositories for different organisms.

The pipeline has been developed in order to collect and automatize part of the analysis used in the *SNPstar* webserver ADD REFENRENCE?. This is a database/data repository that aims to characterize the mutations of different ecotypes of *Arabidopsis*. A part of this characterization involves association studies.

Requirements

The pipeline is implemented in R (version 3.6.1)("The Comprehensive R Archive Network" 2020). In particular the following packages are needed: filesstrings (version 3.1.5) (Nolan and Padilla-Parra 2019), dplyr (version 0.8.3) (Wickham, François, et al. 2019), MASS (version 7.3-51.4) (Ripley et al. 2019), ggplot2 (version 3.2.1) (Wickham, Chang, et al. 2019).

For development and testing the following packages have been exploited: devtools (version 2.2.1) (Wickham, Hester, et al. 2019), roxygen2 (version 7.0.0) (Wickham, Danenberg, et al. 2019), checkmate (version 1.9.4) ("CRAN - Package Checkmate," n.d.), testthat (version 2.3.0) (Wickham and RStudio 2019), rmarkdown (version 1.17) ("R Markdown" 2020), knitr (version 1.26) (Xie 2019), docopt (version 0.6.1) (Jonge 2018).

The GWAS analysis is carried out by the tool GEMMA (version 0.98.1) (Zhou and Stephens 2012).

System requirements and stats

The package has been developed on a Linux machine with Ubuntu-16.04 LTS, with 4 processors Intel® Core™ i5-3210M CPU @ 2.50GHz, 8Gb of RAM.

ADD INFO ABOUT TIME AND RESOURCES

Installation

To install DAPHnE-G it's enough to install the devtools package on your R and subsequently install the package from the corresponding Git repository:

Install and load devtools

- install.packages("devtools")
- library(devtools)

Install and load DAPHnE-G

- install_github("pezzcvd/daphneg2")
- library(daphneg2)

Data

DAPHnE-G data repository consists of 8 tables (5 included in the package plus 3 external): a description of the ecotypes; a collection of environmental parameters and corresponding metedata; the phenotypic traits also with metadata, the genotypic information (SNP occurrence matrix), the linkage disequilibrium SNP blocks and an additional summary of SNP positions. The latter three tables cancan be downloaded from the address: http://141.48.9.71/ (TO SUBSTITUTE WITH AN URL?)

Ecotypes

This table defines and gives information about the ecotypes that are supported. The collection comeS from the 1001genomes project (**???**), hosted by the corresponding repository.

The main feature of this collection is that the data provided for those ecotypes has been deeply genotyped. Therefore for each of them we have an extensive collection of SNPs, which is more precise and complete than the usual 250k array, that in turn needs a SNP imputation step to reach the same precision of our data technique.

The most important fields are *ecotypeID* and *name* because they univoquely identify our ecotypes and are used also by several functions to filter data.

```
head(ecotypes)
#>   ecotypeid      name CS_number country latitude longitude        collector
#> 1        88       CYR   CS76790     FRA  47.4000  0.683333 Valerie Le Corre
#> 2       108    LDV-18   CS77013     FRA  48.5167 -4.066670 Valerie Le Corre
#> 3       139    LDV-46   CS77014     FRA  48.5167 -4.066670 Valerie Le Corre
#> 4       159    MAR2-3   CS77070     FRA  47.3500  3.933330 Valerie Le Corre
#> 5       265     PYL-6   CS77198     FRA  44.6500 -1.166670 Valerie Le Corre
#> 6       350 TOU-A1-88   CS77382     FRA  46.6667  4.116670     Fabrice Roux
#>     seq_by
#> 1 Monsanto
#> 2 Monsanto
#> 3 Monsanto
#> 4 Monsanto
#> 5 Monsanto
#> 6 Monsanto
```

phenotypical and phn_explain

In the frame of the 1001genomes project, the Arapheno database is one of the implemented databases, which collects 461 phenotypes from already published studies. We collected all the available phenotypes, kept the values related to our 1131 ecotypes and in this way we established a phenotype collection. The phenotyic parameters are arranged according to ecotypes (accession id and accession name). Unfortunately the overlap with the ecotype collection is sometimes modest, leading to data sparsity problem.

```
head(arapheno_example)
#>   phenotype_name accession_id accession_name accession_cs_number
#> 1    stomatasize        10017        Petro-1             CS76370
#> 2    stomatasize        10005        Copac-1             CS76420
#> 3    stomatasize         9950          Vie-0             CS76418
#> 4    stomatasize         9949          Qui-0             CS76417
#> 5    stomatasize         9948          Pra-6             CS76416
#> 6    stomatasize         9942          Agu-1             CS76409
#>   accession_longitude accession_latitude accession_country phenotype_value
```

```
#> 1               21.46              44.34            Serbia        106.3447
#> 2               21.95              46.11           Romania        123.6541
#> 3                0.76              42.63             Spain        102.5497
#> 4               -6.93              42.69             Spain        105.6297
#> 5               -3.54              41.05             Spain        108.6393
#> 6               -1.34              41.32             Spain        120.1445
#>   obs_unit_id
#> 1       25679
#> 2       25678
#> 3       25677
#> 4       25676
#> 5       25675
#> 6       25674
```

Assuming that new data will/may be integrated in the pipeline data collection and that they might also come from different sources, a standard format for data insertion has been established. The data from a new study should be stored in a .csv file, named after its id (e.g. if the new phenotype is root length the file name will be root_length.csv) and consisting of at least three mandatory columns: "accession_id", "accession_name" and "phenotype_value". Each line represents an ecotype.

```
phenotypical[1:5,148:153]
#>      389_RGR 38_MT_GH 390_rosetteDM 391_GrowthRate 392_ScalingExponent
#> 88  100.4289       NA       238.430       4.768600           0.7806627
#> 108 107.9881       NA       290.955       4.769754           0.7570078
#> 139 141.7495       NA       292.970       4.965593           0.7561878
#> 159 121.8254       NA       416.055       7.634037           0.7145144
#> 265 118.3253       NA       123.505       2.714396           0.8588173
#>     39_After_Vern_Growth
#> 88                    NA
#> 108                   NA
#> 139                   NA
#> 159                   NA
#> 265                   NA
```

Usually the labels of our phenotypes are quite cryptic (the same holds also for environmental parameters). In fact the phenotype IDs are concise and it is not obvious for the user to understand what those IDs stand for and to which experiment they belong.

For this reason together with the "phenotypical" table comes a second more descriptive one: "phn_explain", which collects metadata. The "explain" tables are supposed to describe which kind of data corresponds to which paramentes and to provide the user with additional information on the experiment which originally generated the data.

```
colnames(phn_explain)
#>  [1] "species"           "phenotype_number"  "ID"
#>  [4] "doi"               "study"             "description"
#>  [7] "growth_conditions" "integration_date"  "number_replicates"
#> [10] "to_name"           "eo_name"           "unit"
```

The field "species" should always correspond to **Arabidopsis thaliana**; "phenotype_number" and "ID" univoquely identify the phenotypic parameter, although sometimes they are not really self explanatory. "doi" and "study" provide references to the arapheno link and pubblications the phenotypes belong to. "description" is a more extensive explanation of the type of data and measurement in comparison to "id". "integration_date" and "number_of_replicates" don't need further explanations and the last fields report several ontology nomenclatures terms: "to" stands for trait ontology, "eo" environment ontology, finally "unit" is the used measure unit.

environmental and env_explain

The environmental parameters we are using come from a previous study [ClimTools]. The authors selected different source databases (e.g. https://neo.sci.gsfc.nasa.gov/, https://webmap.ornl.gov/, http://worldclim.org/version2, etc.), collected data from them, interpolated them to get information useful for our coordinates and to store those preprocessed data ending up with a collection of 196 environmental parameters.

Also in this case we needed to have an uniform format. Similarly to what we did for phenotypic data we are using two tables to describe the environmental dataset: "environmental", which collects the values of 196 environmental parameters for 1131 ecotypes; and "env explain" with the corresponding metadata. In this case we have an extensive description of the majority of ecotypes for each of the environmental parameters.

Also these two datasets are suitable for the integration of new data, with the same requirements as the previous case - the new parameter data should be stored in a .csv file, named after the parameter id. Inside the file the program expects three fields: "accession id", "accession name" and "phenotype value". Only the field requirements for the metadata file are different and less in number.

It follows an example of a subset of environmental parameters table, where we see ecotypes on the rows and climatic factors in the columns.

```
environmental[1:5,1:5]
#>      Latitude Longitude CO_Spring CO_Summer NO2_Spring
#> 88   0.683333   47.4000  130.2587  96.07813   302.1654
#> 108 -4.066670   48.5167  129.8865  96.07813   201.7717
#> 139 -4.066670   48.5167  129.8865  96.07813   201.7717
#> 159  3.933330   47.3500  129.9213  92.59843   229.0026
#> 265 -1.166670   44.6500  128.5497  94.88189   186.6798
```

We also provide an example of corresponding metadata.

```
colnames(env_explain)
#> [1] "Source"             "Description"        "ID"
#> [4] "Units"              "Analyzed.timeframe" "Link"
```

"Source" reports the source database, "ID" is the identification name, which could be not very self explanatory, "Description" is a more detailed explaination of the parameters, "Units" and "Analyzed.timeframe" fields don't need further explainations and "Link" reports the database link to the source data.

Genotype

From the same repository (1001genomes) it comes the genotypic information and it can be retrieved at the web address "https://1001genomes.org/data/GMI-MPI/releases/v3.1/". We extrapolated a concise SNP occurrence table from VCF files describing the set of mutations for each ecotype. The genotypes table in *DAPHnE-G* is the collection of bi-allelic point mutations, whose Minor Allele Frenquency (MAF) is greater than 5% and we end up with a very big table with 200000 SNPs and 1131 ecotypes (In the format accepted by GEMMA the number of columns is 1134 because we need to add the snpID, reference and alternative allele).

11000000 SNPs ~ 1%

1700000 SNPs ~ 5%

200000 SNPs ~ 5% and pruned list

```
head(geno_example)
#>            snpID ref alt 88 108 139 159 265 350 351
#> 239   chr1pos956   C   T  0   0   0   0   0   0   0
#> 250  chr1pos1019   G   T  0   0   0   0   0   0   0
#> 658  chr1pos7464   A   T  1   0   0   0   0   1   0
```

```
#> 834 chr1pos10585   C   A  0   0   0   0   0   0   0
#> 849 chr1pos10904   A   T  0   0   0   0   0   0   0
#> 955 chr1pos13538   T   A  0   0   0   0   1   0   1
```

Tags

With so many SNPs it might occur problems. For instance when many SNPs are inherited together (are in the same linkage equilibrium with each other) and they contribute to an even stronger multiple testing burdain, without contributing with additional sensible information.

To face with problem a strategy is to "prune" the list of SNPs to feed the sotfware that calculates associations. The pruning is performed calculating LD scores (linkage disequilibirum) and identifying subsets of SNPs with a high probability of being inherited together.

We took care of this step beforehand, using the software plink (version 1.9) (Rentería, Cortes, and Medland 2013). First we recoded the genotype information using .ped/.map format, then we pruned the SNPs according to their linkage disequilibrium, keeping only SNPs whose probability to be inherited together is very low and finally we tagged the variants in the same LD units with those that were kept after the pruning.

GEMMA and gemma_annotation

GEMMA is a software used to calculate associations for genomewide association studies. It implements linear mixed models, which help in the identification of data hierarchies and structures.

To correctly use the program it would be useful to prepare a SNP annotation file in advance. This annotation consists of a simple localization: id, position and chromosome.

```
head(gemma_annot_example)
#>            V1   V2 V3
#> 1   chr1pos92   92  1
#> 2  chr1pos502  502  1
#> 3 chr1pos1016 1016  1
#> 4 chr1pos1645 1645  1
#> 5 chr1pos3102 3102  1
#> 6 chr1pos4648 4648  1
```

External files and programs

- geno_005_ld02.RData
- tags
- gemma (from https://github.com/genetics-statistics/GEMMA)
- gemma_annot.csv

Structure

The pipeline is divided into six parts:

- Repository handling
- Preprocessing
- Gwas analysis
- Quality checking

- Downstream report
- Command line wrapper

Each of them consists of several functions that are going to be explained in detail in the following sections.

PROBABLY TO COMMENT Moreover the function *DAPHnE-G.R* acts as a wrapper and provides a command line version of the tool.

Usage

DAPHnE-G has two modes of use, the first is the standard R package loading and function calling; the second is through the assstuffa.R function, which acts as a wrapper of all the other and is suited to work from command line.

In the next sections all the functionalities are described, starting with the dataset handling, then the association study, the quality checking and the a posteriori report. Eventually the command line options are listed.

Repository handling

This section describes how to work with the core dataset, how to set up an .RData file contaning all the previously described tables, supposed to be constantly updated to the most recent version and an archive of backups which keep track of all the changes. I will also show how those two systems mutually interact in order t oavoid loss of information.

METTI SCHEMA BELLO

infos

The phenotypical/environmental tables, containing the collection of values for the parameters, are identified through very concise names. This turns up sometimes in hard times for the user to properly understand what the chosen parameter is measuring. To this scope the tables *env_explain* and *phn_explain* are built - they contain a more detailed description of the parameters; what they measure, which study/publication they come from and possibly a more detailed description of the experimental setting. The function *infos* allows the user to print all these pieces of information for a selected parameter.

Let's suppose for example that we are interested in the environmental parameter **CO_Spring**, we would call the function with:

```
infos("CO_Spring")
#>                                                 Source
#> 3 Measurements of Pollution In The Troposphere (MOPITT)
#>                      Description        ID Units Analyzed.timeframe
#> 3 MOPITT Carbon monoxide spring  CO_Spring ppbv          2001-2010
#>                                                    Link
#> 3 https://neo.sci.gsfc.nasa.gov/view.php?datasetId=MOP_CO_M
```

and the program tracks the parameter to the IDs of either env_explain or phn_explain table, returning all the informations we could provide.

If the ID is not yet present in the dataset an error is reported, so we should first add the new parameter to the dataset, synchronize the system, and finally we will be able to print our information.

examples

infos(i.input = "CO_Spring")

set_backup

This is an internal use function. It is called by *add_param* and *synchro* to check the status of the backup archive. The function takes as an argument the path of the backup archive (by default set to home directory).

Then four tests are performed:

- Does the backups folder already exists at that path? If not one will be created.
- Is the directory populated? At least the five current versions of the tables are expected to be there. If not they are printed.
- Is there a RData folder? It will be created if needed.
- Is also this folder populated? Two RData files are expected. The genotypes are kept separately because the table is very big, computationally demanding and not always needed. Moreover it doesn't need to be updated.

examples

set_backup(sb.dest = "/home/user")

add_param

Add_param is the first of the two steps needed for adding a new parameter to the parameter list tables. The input files are the new parameter data file, the new parameter metadata file (both *.csv* format. see the **Data** section), an environmental-mode flag: False (default) stands for phenotypic analysis and True enables environmental analysis and the backup folder path.

The first operation is to set the correct tables according to the mode.

Then after checking that the files meet the format requirements the newparam file (data) is filtered out from non-supported ecotypes and duplicates, its values are fitted in the format of our data repository and finally added to it. The same is done with metadata file.

Finally the four backup files are written - two of them are tagged with the creation date (they will count as archive) and other two are tagged with "_current" and they are going to overwrite the previous ones. In this way the "_current.csv" files are always refering to the most up to date version of the core dataset.

examples

add_param(ap.newparam = "../inst/extdata/example_add_param.csv", ap.metadata = "../inst/extdata/example_add_param
= T)

synchro

synchro is the second step of the dataset updating procedure and coupled with *add_param* it provides a secure system to avoid loss of data and to handle human error.

The inputs are the environmental-mode flag described before in *add_param* section, the backup folder path and the backup version that will be saved in the RData file ("current" is default).

As in the previous step the mode helps selecting the right files to load, where tmp and tmpx are the sources and s.table and s.explain are the destinations.

Then sources and destination files are compared to assess whether the core dataset is up-to-date.

Finally all the tables of the newly updated core dataset are saved in the RData file.

To sum up the data() provided with the package are the original version of the dataset and they could be restored if needed. While all the files in the "backups" folder - RData and .csv files - form a double check procedure to ensure the system is always updated. The choiche to keep a backup archive was taken to have

the possibility to synchronize the dataset to a previous version, in case a user notices some mistakes in a putative intermediate to last step.

examples

- synchro()
- synchro(s.mode = F)

preprocessing

The second section takes care of the actual GWAS analysis. It consists of two commands: one for the specific association study preprocessing (which produces input files for the gwas) and the analysis itself.

METTI SCHEMA BELLO

preproc

The preprocessing functionalt, in the function *preproc*, discriminates among several possibilities for the analysis and the final task is to write the files that will be fed to GEMMA for the GWAS analysis - in particular phenotype, genotype files and, if needed, covariate and model files.

The analysis can be univariate, to look for associations between mutations and only one parameter (phenotype); or multivariate if it considers also covariates.

We have in total five options:

- E univariate environmental
- P univariate phenotypical
- PxP multivariate phenotypical with phenotypical covariate(s).
- PxE multivariate phenotypical with environmental covariate(s).
- ExE multivariate environmental with environmental covariate(s).

Moreover, in case of multivariate analysis, the user can choose to specify one covariate of interest (*oneCov* case, it has to be present in the core dataset), or ask the system to evaluate the best subset of covariates among the whole collection, *nCov* case.

Once one of the five options is identified, the corresponding tables are selected and the process for *univariate* or *multivariate* is initiated through teh corresponding function.

examples

- preproc(pp.par = “CO_Spring”, pp.option = “E”, pp.geno =“inst/extdata/example_geno.RData”)
- preproc(pp.par = “100_4W”, pp.option = “P”, pp.geno =“inst/extdata/example_geno.RData”)
- preproc(pp.par = “CO_Spring”, pp.option = “ExE”, pp.geno =“inst/extdata/example_geno.RData”, pp.cvt = “ET”)
- preproc(pp.par = “100_4W”, pp.option = “PxP”, pp.geno =“inst/extdata/example_geno.RData”, pp.cvt = “101_Li7”)
- preproc(pp.par = “100_4W”, pp.option = “PxE”, pp.geno =“inst/extdata/example_geno.RData”, pp.cvt = “CO_Spring”)

Normal

This is an internal function called in both univariate and multivariate data preprocessing functions in order to ensure the normality condition.

It uses kolmogorov-smirnoff test to compare the submitted data vector with another distribution, in our case a normal distribution with same mean and variance as the input sample.

The test is performed for the data as they are, or iteratively in case the previous test fails, with log, logit, sqrt and inverse transformations.

Univariate

This is an internal function and it carries on the actual data preprocessing for an univariate analysis.

First of all data corresponding to the selected phenotype are selected and missing values are removed. Than normality is checked. At last phenotype and genotype files are written subselecting the complete dataset.

Multivariate

This internal function starts the data preprocessing for a multivariate analysis. In principle it works as in the univariate case. It starts retrieving phenotype values and removing missing values.

But the it needs to set up an additional covariate table, which will be the same as the main phenotypic table in case of PxP or ExE analysis (removing the column of the phenptype of interest), or different for PxE mode.

The last step consists of identifying how many covariates we are considering. In case of one covariate, it has to be one of the phenotypes of environmental factors already present in our list. On the contrary if the user wants to consider the best combination among all the possible covariates should set it to "all".

oneCov

This internal function finally writes the preprocessed phenotype, genotype and covariate files in case of a one covariate analysis. It should be used if the user would like to identify the associations between two variables present in the dataset, where the first will be the target and the second the covariate.

As first step the function retrieves the covariate data values from the dataset, removes NAs and subselects the ecotypes in order to keep only the ones with definite values for both phenotype and covariate.

Subsequently it assess normality for both phenotype and covariate (using *normal* function described before).

The last step is to write the tables.

nCov

This internal function finally writes the preprocessed phenotype, genotype, covariate and model files in case of a n covariates analysis. It should be used if the user would like to identify the associations between a target variable and the best subset of covariates among those in the dataset.

The datasets count 461 phenotypical traits and 196 environmental factors and it would be impossible to include all of them in one analysis and get a meaningful result from the mathematical/statistical prespective. So as described in REF VAN ZANTEN we will fit linear models with subsets of 10 randomly chosen variables, compare the quality of the models and pick the best one.

We decided to fit 1000 linear models and to use the Akaike Information Criterion (AIC) to compare them.

The rest of the script is very similar to the *oneCov* function and the preprocessed files are created. The only differences are that a *model* file is created, to keep track of which variables are finally used and that the covariate file contains the fitted values from the linear model that has been chosen.

GWAS analysis

The GWAS mode performs the actual association analysis (through the software GEMMA) and returns a table with all associations and corresponding P-value.

METTI SCHEMA BELLO

gwas

The script first has to set some parameters. It checks the number of covariates (0,1 or 2 = as more than 1) and sets a suffix to the file name that is going to be used. (e.g. "CO_spring"" (with no suffix) for 0 covariates, "CO_Spring_ET" for 1 covariate (ET is the name of teh covariate) or "CO_Spring_all" for 2 covariates).

Then it retrieves the path of GEMMA on user's computer (user has to provide the file's name, for example "gemma-0.98.1-linux-static").

The next steps are kinship matrix and association table calculation, which are both performed by GEMMA. (see below). The resulting P-values are then adjusted providing False Discovery Rates (FDRs) with function and finally the information about pruned SNPs is reintegrated.

The user is asked to provide genotype, phenotype, kinship and in case covariate files, which are supposed to be produced with the preprocessing command of DAPHnE-G. The program needs also an annotation file: which contains snpID, chromosome and position in base pairs; and a tags file where are saved all the lists of SNPs in the same LD unit.

Plus some additional parameters are provided with a default value (that the user can change): missingness and minor allele frequency.

examples

- gwas(gemma.name = "gemma-0.98.1-linux-static", gw.input = "CO_Spring", gw.cv = 0, gw.annot = "inst/extdata/gemma_annot_example.csv")
- gwas(gemma.name = "gemma-0.98.1-linux-static", gw.input = "CO_Spring", gw.cv = 0, gw.annot = "inst/extdata/gemma_annot_example.csv", gw.kinship = "inst/extdata/precomputed_kinship.txt")
- gwas(gemma.name = "gemma-0.98.1-linux-static", gw.input = "CO_Spring", gw.cv = 1, gw.annot = "inst/extdata/gemma_annot_example.csv",gw.cov = "ET")
- gwas(gemma.name = "gemma-0.98.1-linux-static", gw.input = "CO_Spring", gw.cv = 2, gw.annot = "inst/extdata/gemma_annot_example.csv",gw.cov = "all")

kinship

Kinship is a relatedness matrix, which measures the degree of similarity among the different ecotypes. This calculation is also taken care of with the GEMMA software.

gemma

It calculates the associations table using the software GEMMA, which exploits linear mixed models. The script needs a phenotype and a genotype and in case of multivariate analysis covariate files (created during the preprocessing); a kinship matrix (see above); a SNP annotation file and finally missingness and minor allele frequency are set to 5% by defalut, but can be set to different values at need.

adjust

This internal function provides an adjustment of GWAS result p-values. It simply loads the result file, calculate che adjustments with "benjamini-hochberg" and "benjamini-yekutieli" methods and save the file again.

tagged_snps TO REMOVE

It's an internal function and it adds to the result table more entries, one for each of the previously pruned SNPs that are inherited together with one of the SNPs in the original result file.

First the tags file is loaded, then for each of the snps with a P-value less than 10e-5 it retrieves from the tags table the SNPs in the same LD unit and creates an entry in the extended result file with the tagged SNPs as id and the same P-value as the original tag SNP. Finally the extaned result table is saved, not updating the previous table but rather in a new file.

Quality checking

This section functionality is to assess the quality of the study. This is achieved through GWAS graphical methods: manhattan plots, qq-plots and p-value histogram distribution.

SCHEMA BELLO

manhattan

A manhattan plot depicts the -log(P-values) of all analyzed SNPs sorting them according to chromosome and genomic position. It's very useful to visually identify quantity and coordinates of the strongest associations, which result in the highest peaks.

For a target parameter are produced three Manhattan plots: one with non-adjusted P-values and two with adjusted ones. For these latter are used Benjamini-Hochberg (BH) and Benjamini-Yekutieli (BY) adjustments.

For each of the plots three significance thresholds are highlighted: 1e-5, 1e-6 and 1e-7 for the non-adjusted P-values; 0.2, 0.1 and 0.05 for BH case and 0.1, 0.05, 0.01 for BY case. In both adjusted P-value cases the thresholds represent the False Discovery Rate (FDR), e.g. threshold of 0.05 equals an FDR of 5%.

examples

manhattan(mh.par = "CO_Spring")

qq_plot

A QQ plot represents the deviation of the observed P-values (in negavite log scale) of a specific experiment from the expected ones which should follow an uniform distribution. Given that for a successful GWAS experiment we expect just an minority of the SNPs to show the association, this little set should be vizualized as a small divergind tail on the right part of the plot.

The expected values are simulated dividing the interval [0,1] into n qeually spaced bins, where n is the number of SNPs analyzed and transforming those values in negative logs; the observed values are the negative logs of the sorted list of P values.

Also with this functions three plots are produced: one for the non-adjusted P values and two for the two sets of adjusted ones.

examples

qq_plot(qq.par = “CO_Spring”)

pval_distribution

The P-values distribution is simply an histogram representing the P-values of a specific experiment. The x-axis is binned in intervals of width 0.1. Also this plot helps to visualize the result of the experiment.

examples

pval_distribution(h.par = “CO_Spring”)

Summary report

Bibliography

“CRAN - Package Checkmate.” n.d. https://cran.r-project.org/web/packages/checkmate/index.html.

Jonge, Edwin de. 2018. “Docopt: Command-Line Interface Specification Language.” https://CRAN.R-project.org/package=docopt.

Nolan, Rory, and Sergi Padilla-Parra. 2019. “Filesstrings: Handy File and String Manipulation.” https://CRAN.R-project.org/package=filesstrings.

“R Markdown.” 2020. Accessed January 13. https://rmarkdown.rstudio.com/.

Rentería, Miguel E., Adrian Cortes, and Sarah E. Medland. 2013. “Using PLINK for Genome-Wide Association Studies (GWAS) and Data Analysis.” *Methods in Molecular Biology (Clifton, N.j.)* 1019: 193–213. doi:10.1007/978-1-62703-447-0_8.

Ripley, Brian, Bill Venables, Kurt Hornik Bates Douglas M., and David Firth. 2019. “MASS: Support Functions and Datasets for Venables and Ripley’s MASS.” https://CRAN.R-project.org/package=MASS.

“The Comprehensive R Archive Network.” 2020. Accessed January 13. https://cran.r-project.org/.

Wickham, Hadley, and R. Core team RStudio. 2019. “Testthat: Unit Testing for R.” https://CRAN.R-project.org/package=testthat.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and RStudio. 2019. “Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics.” https://CRAN.R-project.org/package=ggplot2.

Wickham, Hadley, Peter Danenberg, Gábor Csárdi, Manuel Eugster, and RStudio. 2019. “Roxygen2: In-Line Documentation for R.” https://CRAN.R-project.org/package=roxygen2.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and RStudio. 2019. “Dplyr: A Grammar of Data Manipulation.” https://CRAN.R-project.org/package=dplyr.

Wickham, Hadley, Jim Hester, Winston Chang, RStudio, and R. Core team. 2019. “Devtools: Tools to Make Developing R Packages Easier.” https://CRAN.R-project.org/package=devtools.

Xie, Andrew, Vogt. 2019. “Knitr: A General-Purpose Package for Dynamic Report Generation in R.” https://CRAN.R-project.org/package=knitr.

Zhou, Xiang, and Matthew Stephens. 2012. “Genome-Wide Efficient Mixed Model Analysis for Association Studies.” *Nature Genetics* 44 (7): 821–24. doi:10.1038/ng.2310.