1. Installing the NAM package

To install the package in R, you just have to open R and type:

- `install.packages("NAM")`

NAM can be installed in R 3.2.0 or more recent versions. Check the version typing `R.version`. To load the package, you have to type in R

- `library(NAM)`

Some quick demostrations of what the package can do are available through the R function `example`. Check it out!

- `example(plot.NAM)`
- `example(Fst)`
- `example(snpH2)`

2. Loading and formatting data

Our package does not require a specific input file, just objects in standard R classes, such as numeric matrices and vectors. In this vignette we are going to show some codes that would allow users to load and manipulate datasets in R. For example, `read` commands are commonly used to load data into R. It is possible to check how they work by typing `?` before the command. For example:

- `?read.table`
- `?read.csv`

Let the file “genotypes.csv” be a spreadsheet with the genotypic data, where the first row contains the marker names and each column represents a genotype, where first column contains the genotype identification. An example of loading genotypic data:

- `gen = read.csv( "~/Desktop/genotypes.csv", header = TRUE )`

It is impotant to keep the statement `header = TRUE` when the first row contains the name of the markers. Data is imported as a `data.frame` object. To convert to a numeric object you can try

- `gen = data.matrix(gen)`

And then check if it is numeric

- `is.numeric(gen)`

This step is not necessary if you are importing the phenotypes or other information. In this case, you can obtain your numeric vectors directly from the `data.frame`. Let the file “data.csv” be a spreadsheet with three columns called Phenotype1, Phenotype2 and Family, and we want to generate three R objects named $Phe1$, $Phe2$ and Fam. To get numeric vectors, you can try

- `data = read.csv("~/Desktop/data.csv")`
- `Phe1 = as.numeric( data$Phenotype1 )`
- `Phe2 = as.numeric( data$Phenotype2 )`

- `Fam = as.numeric( data$Family )`

Notice that in R, `NA` is used to represent missing values.

To import GBS data (CGTA text format), the following code can be used

GENOTYPE: *'gen'* matrix

- `G = Import_data("GBSdata.txt",type = "GBS")` *# Reading data into R*
- `gen = G$gen`
- `chr = G$chr`

And to import hapmap data, the following code can be used to provide two important inputs in the NAM format: genotype (`gen`) and chromosome(`chr`). Let “hapmap.txt” be a hapmap file.

- `G = Import_data("hapmap.txt",type = "HapMap")` *# Reading data into R*
- `gen = G$gen`
- `chr = G$chr`

The function “Import_data” also accepts a third type of data, “VCF”.

Some package, such as the function BLUP of the SoyNAM package, have datasets already compatible with the require inputs of NAM package for association analysis. It is also possible to load an example dataset that comes with the NAM package to see data format. Try:

- `data(tpod)`
- `head(y)`
- `gen[1:4,1:4]`
- `head(fam)`
- `head(chr)`

Analyses performed by the NAM package require inputs in numeric format. To check if the objects required for genome-wide association studies are numeric, use the logical command `is.numeric`.

- `is.numeric(phenotype)`
- `is.numeric(genotypes)`
- `is.numeric(population)`
- `is.numeric(chromosomes)`

To verify if the input is correct regarding the class of object, you may want to try:

- `is.vector(phenotype)`
- `is.matrix(genotypes)`
- `is.vector(population)`
- `is.vector(chromosomes)`

You can force an object to be numeric. Example:

- `phenotype = as.numeric(phenotype)`

It is recommended to check that the object is in the expected format after forcing it into a specific class.

3. Genome-wide association studies

The linear model upon which association analyses are performed is briefly described in the description of R function `?gwas`. More in-depth basis are provided in the supplementary file available with the following code:
`system(paste('open',system.file("doc","gwa_description.pdf",package="NAM")))`

To perform genome-wide association studies, at least two objects are required: A numeric matrix containing the *genotypic information* where columns represent markers and rows represent the genotypes, and a numeric vector containing the *phenotypes*. In addition, two other objects can be used for association mapping: a *stratification* term, a numeric vector with the same length as the phenotypes used to indicate the population that each individual comes from, and a numeric vector equal to the number of *chromosomes* that indicates how many markers belong to each chromosome. The sum of this object must be equal to number of columns of the genotypic matrix.

The genotypic matrix must be coded using 0-1-2 (aa, aA, AA), and we strongly recommend to keep the column names with the marker names. If the stratification parameter is provided, we strongly recommend to use zeros to code alleles with minor frequency. The package provides a function called *reference* that does that (type `?reference` for more details). If stratification is provided, the algorithm used to compute associations will allow minor alleles to have different effect, increasing the power of associations by allowing different populations be in different linkage phases between the marker being evaluated and the causative mutation.

To run the association analysis, use the function `gwas`. The arguments `y` (phenotypes) and `gen` (genotypes) are necessary for the associations, the arguments `fam` (stratification) and `chr` (number of markers per chromosome) are complimentary. Thus:

- `my_gwas = gwas (y = phenotype, gen = genotypes)`
- `my_gwas = gwas (y = phenotype, gen = genotypes, fam = population, chr = chromosomes)`

For large datasets, the computer memory may become a limitation. A second function was designed to overcome this issue by not keeping the haplotype-based design matrix in the computer memory. Try:

- `my_gwas = gwas2 ( y = phenotype, gen = genotypes )`
- `my_gwas = gwas2 ( y = phenotype, gen = genotypes, fam = population, chr = chromosomes )`

When multiple independent traits will be analyzed, there exist the possibility avoiding the Eigendecomposition of the kinship matrix for every GWAS you run using a same population. The function `eigX` generates and decomposes the kinship, and the output is suitable for the argument `EIG` in the `gwas2` function.

- `eigNAM = eigX(gen=genotypes, fam=population)`
- `trait_1 = gwas2( y=phenotype1, gen=genotypes, fam=population, chr=chromosomes, EIG=eigNAM )`
- `trait_2 = gwas2( y=phenotype2, gen=genotypes, fam=population, chr=chromosomes, EIG=eigNAM )`

For large number of SNPs, assocation analysis may present a heavy computation burden, which can be overcome through the computation SNPs in parallel. For parallel computing, we recently added an extension of `gwas2` that works along with the R package snow. The functions `gwasPAR` is accessible through the following command: `source(system.file("add","gwasPAR.R",package="NAM"))`.

There are four simple steps to get a parallel computation of your association studies: 1) load the `gwasPAR` function; 2) open a cluster using the snow package; 2) run gwas with `gwasPAR`; 3) close the cluster. The exmple code follows:

- `source(system.file("add","gwasPAR.R",package="NAM")) # Step 1`
- `CLUSTER = makeSOCKcluster(3) # Number of CPUs in the clusters # Step 2`
- `my_gwas = gwasPAR(y=phenotype,gen=genotypes,fam=population,chr=chromosomes,cl=CLUSTER) # Step 3`
- `stopCluster(CLUSTER) # Step 4`

Once the associtation analysis was performed, to visualize the Manhattan plots can use the `plot` command on the output of the function `gwas`.

- `plot( my_gwas )`

To check other designs for your Manhattan plot, check the examples provided by the package (see `?plot.NAM`). To figure out which SNP(s) represent the picks of the analysis, we design the argument `find`. With this argument, you can click in the plot to find out which markers correspond to the peaks. For example, you want to find out the markers responsible for two picks, try:

- `plot( my_gwas, find = 2 )`

To adjust significance threshold for multiple testing, you can use the Bonferroni correction by lowering the value of alpha, which is 0.05 by default. For example, if you are analyzing 150 markers, you can obtain the *Bonferroni threshold* by:

- `number_of_markers = 150`
- `plot( my_gwas, alpha = 0.05/number_of_markers )`

To plot the Manhattan plot using an acceptable false discovery rate (FDR) by chromosome or Bonferroni threshold by chromosome, try:

False discovery rate of 25%

- `plot( my_gwas, FDR = 0.25)`

Bonferroni threshold by chromosome

- `plot( my_gwas, FDR = 0)`

If you want to disregard the markers that provide null LRT when building the FDR threshold as previously showed, you can use the 'greater-than-zero' (gtz) command. It works as follows:

False discovery rate of 25%

- `plot( my_gwas, FDR = 0.25, gtz=TRUE)`

Bonferroni threshold by chromosome

- `plot( my_gwas, FDR = 0, gtz=TRUE)`

Most output statistcs are available in the *PolyTest* object inside the list output from the *gwas* function. These output includes -log(*P-values*), LOD scores, variance attributed to markers, heritability of the full model, marker effect by family and its standard deviation. For example, to get the LRT score of each SNP, you can type

- `SCORE = my_gwas$PolyTest$lrt`

These scores are LRT (*likelihood ratio test statistics*), they represent the improvement that each SNP provides to a mixed model. To obtain the $-log(P-value)$:

- `PVal = my_gwas$PolyTest$pval`

The object **PVal** contains all the -log(p-values). P-value are obtained from LRT using the Chi-squared density function with 0.5 degrees of freedom. The value 0.5 is used because random effect markers generate a mixture of Chi-squared and Bernoulli distributions once many markers have zero contribution.

To find out the amount of variance explained by each marker, type

- `Genetic_Var_each_SNP = my_gwas$PolyTest$var.snp`
- `Var_Explained_by_SNP = Genetic_Var_each_SNP / var(phenotype)`

To export as CSV file with all SNP statistics:

- `write.csv( my_gwas$PolyTest, "my_file_with_snp_scores.csv" )`

To find out which markers are above a given significance threshold, use the following code

- `THR = 0.05/number_of_markers`
- `w = which(PVal > THR)`
- `w` *# Significant markers*

To find out the Bonferroni threshold in LRT scale, try

- `optim(1,fn=function(x)abs(-log(pchisq(lrt,0.5,lower.tail=FALSE),base = 10) + log(0.05/number_of_markers)),method="CG")$par`

The meaning of each column from PolyTest is summarized below:

- conv = convergence
- fn1 = Likelihood of the full model (marker included)
- fn0 = Likelihood of the reduced model (marker excluded)
- lod = lod score, Logarithm of Odds
- pval = -log10(p-value)
- lrt = likelihood ratio test statistic
- sigma2g = genetic variance estimated from the SNP
- sigma2 = residual variance of the full model
- h2 = narrow-sense heritability of the full model h2=Va/(Va+Ve), where Va=Var(SNP)+Var(Polygene)
- lam_k = Va(snp)/Ve
- var.snp = variance explained by SNP as Var(marker)/Var(Y)
- intercept = intercept of the model
- std.eff (gwas/gwas2) = allele effect from the reference line (standard parent for NAM)
- eff.1 = allele effect in sub-population 1 or family 1

- eff.2 = allele effect in sub-population 2 or family 2
- eff.X = allele effect in sub-population X or family X

The output of the GWAS function provides the allele effect into the GWAS of multiple populations context, testing one marker at a time. To find out the effect of each marker conditional to the genome (i.e. given all the other makers are in the model). This technique is known as whole-genome regression (WGR) method.

- `WGR = wgr( y = phenotype, gen = genotypes )`
- `Allele_effect = WGR$g`
- `plot(abs(Allele_effect))` *# Have a look*

The above example characterizes the BLUP method, also known as snpBLUP and ridge regression blup (RR-BLUP). Since the example above was solved in Bayesian framework, it is also referred as Bayesian ridge regression (BRR) coefficient.

4. Marker quality control

Two functions are dedicated to quality control of the markers used in genome-wide studies: `snpQC` and `snpH2`. The latter function evaluates the Mendelian behavior and ability of each marker to carry a gene by computing the marker heritability as the index of gene content.

The function `snpQC` is used to remove repeated markers and markers that have minor allele frequency below a given threshold. This function is also used to impute missing values by semi-parametric procedures (random forest).

Repeated markers are two markers side-by-side with identical information (i.e. full linkage disequilibrium), where the threshold that defines “identical” can be specified by the user through the argument `psy` (default is 1). The argument `MAF` controls the threshold of minor allele frequency (default is 0.05). The logical argument `remove` asks if the used want to remove repeated markers and markers below the MAF threshold (`remove = TRUE`) or just to be notified about it (`remove = FALSE`), by default it removes the low quality markers. The logical argument `impute` asks if the user wants to impute the missing values, the default is `impute = FALSE`.

An example of how to use the function `snpQC` to impute missing loci and remove markers with MAF lower than 10% is:

- `adjusted_genotypes = snpQC( gen=genotypes, MAF=0.10, impute=TRUE )`

Then, you can try to verify the gene content by:

- `forneris_index = snpH2 ( adjusted_genotypes )`
- `plot ( forneris_index )`

To speed up imputations, it is recommend to impute one chromosome at a time. For example, to impute the first a hundred markers and then the following hundred, you can try:

- `genotypes[,001:100] = snpQC(gen=genotypes[,001:100], impute=TRUE, remove=FALSE )`
- `genotypes[,101:200] = snpQC(gen=genotypes[,101:200], impute=TRUE, remove=FALSE )`

An additional QC that can be performed is the removal of repeat genotypes. The NAM package provides a function for this task. The arguments are: a matrix of phenotypes (`y`), a family vector (`fam`) and the genotypic matrix (`gen`). If you are using a version >1.3.2, an additional argument can be specified, `thr`, the threshold above which genotypes are considered identical. In the NAM version 1.3.2 it is pre-specified as 0.95, which is also the default setting of newer versions.

- `cleanREP( y, fam, gen)`

It returns a list with the inputs (y, fam and gen) without the redundant genotypes. Thus, it is possible to clean phenotype matrix, genotypic matrix and family vector, all at once. An example with two phenotypes (phe1 and phe2) would look like:

- `PHENOS = cbind(phe1,phe2)`
- `CLEAN = cleanREP( y = PHENOS, fam = Family, gen = Genotypes)`
- `phe1_new = CLEAN$y[,1]`
- `phe2_new = CLEAN$y[,2]`
- `Family_new = CLEAN$fam`
- `Genotypes_new = CLEAN$gen`

5. Signatures of selection

It may be of interest to evaluate which genomic regions are responsible for the stratification of populations and to check if there is further structure among and within populations through the `Fst` function. F-statistics are used to calculate the variation distributed among sub populations (Fst), the heterozygousity of individuals compared to its populations (Fit) and the mean reduction in heterozygosity due to non-random mating (Fis). The `Fst` function implemented in NAM calculates Fst, Fit and Fis.

Two arguments are necessary for this function: the genotypic matrix (`gen`) and a stratification factor (`fam`).

- `my_FST = Fst ( gen = genotypes, fam = stratification )`
- `plot(my_FST)`

6. BLUPs and GEBVs

Considering that phenotypes are often replaced by BLUP values for mapping and selection, the NAM package provide functions that allow users to solve mixed models to compute BLUPs and variance components: `reml` and `gibbs`.

To obtain BLUPs using REML the user needs an object for each term of the model: numeric vector for each covariate and for the response variable, and a factors for categorical variables such as environment and genotype.

To check if a given `object` (eg. matrix, vector or factor) belongs to the correct class you expect, you can use the commands `is.vector(object)`, `is.numeric(object)`, `is.matrix(object)` and `is.factor(object)`. To force an object to change class, you can try `object = as.factor(object)` or `object = as.vector(object)`.

Let `trait` be a numeric vector representing your response variable, `env` be a factor representing a different environments, `block` be a factor that indicates some experimental constrain, and `lines` be a factor that represent your lines. To fit a model, try:
Fit the model

- `FIT = reml ( y = trait, X = ~ block + env, Z = ~ lines )`

Variance components

- `FIT$VC`

BLUP values (genetic values)

- `FIT$EBV`

Another possibility is to fit a GBLUP, useful to obtain breeding values using molecular data. Let `gen` be the genotypic matrix, `env` be a factor representing a different environments, and `lines` be a factor that represent your lines. The GBLUP model would be fitted as:

Genomic relationship matrix

- `G = GRM(gen)`

Fit the model

- `FIT = reml ( y = trait, X = ~ env, Z = ~ lines, K = G )`

GBLUP values (breeding values)

- `FIT$EBV`

The function `gibbs` is also unbiased and works with arguments similar to `reml`, with few important differences: (1) the `gibbs` function enable users to fit models with multiple random variables; (2) the kinship argument requires the inverse kernel to save computation time; (3) aside from the point estimates, `gibbs` also provides the posterior distribution for Bayesian inferences.

Now, lets see how to fit a GBLUP with the environment factor set as random effect. Let `gen` be the genotypic matrix, `env` be a factor representing a different environments, and `lines` be a factor that represent your lines. The GBLUP model would be fitted as:

Genomic relationship matrix

- `G = GRM(gen)`
- `iG = chol2inv(G)`

Fit the model

- `FIT = gibbs ( y = trait, Z = ~ lines + env, K = iG )`

GBLUP values (breeding values)

- `rowMeans(FIT$Posterior.Coef$Random1)`

Similarly, it is possible to fit other models for genomic selections, such as Bayesian ridge regression (BRR) and BayesA using one of these function two mixed model functions. To fit a simple model with environment as fixed effect:

Fit BRR using the gibbs function

- `FIT_BRR = gibbs ( y = trait, X = ~ env , Z = gen, S=NULL)`

Both functions `reml` and `gibbs` accept formulas and matrices as inputs. When multiple random effects are used in `gibbs`, the argument `Z` accepts formula or a list of matrices and the argument `iK` accepts matrix (if only the first random effect has known structure) or a list of matrices (if multiple random effects have known covariance structure). An additional argument in the `gibbs` function, `iR` allows users to include residual covariance structure. An example of `iR` could be the inverse kernel to informs the spatial layout of the observations, such as the outcome of the function `covar`, to account for heteroscedasticity due to spatial auto-correlation.

Although it is possible to use `reml` and `gibbs` to generate breeding values, the functions `wgr` (also implemented in the bWGR package) and `gmm` enables the use of more appropriated and optimized models for genomic prediction. Some faster algorithms not based on MCMC are also available, such as `emBB`, `emML`, `emDE`, `press` and others. Some popular methods that can be obtained from this function (Bayesian alphabet). The NAM package provides a wide variety of methods to estimate breeding values for observed genotypes or predict unphenotyped material, fit the model as follows:

a. BLUP

- `BLUP = wgr(y = phenotype, X = genotype, iv=FALSE, pi=0)`

b. BayesA

- `BA = wgr(y = phenotype, X = genotype, iv=TRUE, pi=0)`

c. BayesB

- `BB = wgr(y = phenotype, X = genotype, iv=TRUE, pi=0.8)`

d. BayesC

- `BC = wgr(y = phenotype, X = genotype, iv=FALSE, pi=0.8)`

e. Bayesian Elastic Net (under dev)

- `BEN = ben(y = phenotype, gen = genotype)`

f. Bayesian LASSO

- `BL = wgr(y = phenotype, X = genotype, de=TRUE)`

g. Extended Bayesian LASSO

- `EBL = wgr(y = phenotype, X = genotype, de=TRUE, pi=0.8)`

h. GBLUP

- `LK = gmm(y = phenotype, gen = genotype, model = "GBLUP")`

i. Reproducing Kernel Hilbert Spaces

- `GK = gmm(y = phenotype, gen = genotype, model = "RKHS")`

j. non-MCMC BayesDpi

- `eBD = emBD(y = phenotype, gen = genotype)`

k. non-MCMC BayesA

- `eBA = emBA(y = phenotype, gen = genotype)`

l. non-MCMC BayesB (variable selection not stable)

- `eBB = emBB(y = phenotype, gen = genotype)`

m. non-MCMC BayesC (variable selection not stable)

- `eBC = emBC(y = phenotype, gen = genotype)`

n. non-MCMC BRR

- `eRR = emRR(y = phenotype, gen = genotype)`

o. Fast Laplace Model

- `flm = emDE(y = phenotype, gen = genotype)`

p. Elastic net

- `eEN = emEN(y = phenotype, gen = genotype)`

q. Mixed L1-L2 (variation of Elastic net)

- `eMix = emBL(y = phenotype, gen = genotype)`

r. Maximum likelihood

- `eML = emML(y = phenotype, gen = genotype)`

r. PRESS-regularized gblup

- `ePRESS = press(y = phenotype, K = GRM(genotype))`

s. BayesCpi

- `bcpi = BCpi(y = phenotype, X = genotype)`

For a comparison of the ten methods, one can perform a cross-validation study. In cross-validation studies, a \code{k} part of the data is omitted and predicted back. The procedure is repeated various times. Prediction statistics such as mean-squared prediction error (MSPE) and prediction ability (PA) are computed by the comparison between observed and predicted values.

The package includes the function \code{emCV} for cross-validating using Expectation-Maximization algorithms of whole-genome regression (also implemented in the package bWGR), and a slightly more comprehensive implementation shown below. To load the latter, enter the following script

- `source(system.file("add","cvNAM.R",package="NAM"))`

Which contains two functions: \code{CV_NAM} and \code{CV_Check}. The former function perform cross-validations, and the latter function summarizes the results. For example, load a small dataset (eg. `load(tpod)`), then check how different models perform:

- `TEST = CV_NAM(y,gen,IT=1500,BI=500)`
- `CV_Check(TEST)`

The function `gmm` used above provides some extra flexibility for replicatad trials. A data frame containing all relevant data can be provided to the argument `dta`, including a columns named “ID”, covariates and environmental information. For that, it is also important to have the rows of your genotypic matrix `gen` named with the same identification provided in the data fram `dta`. FOr that, use the R function `rownames` assign and verify the names of your genotypes in the genotypic matrix.

If spatial information is provided in the Block-Row-Column format, the function gmm will perform spatial adjustment, fitting at the same time the genetic and spatial terms. An example of how the input matrix of `dta` looks like is provided in the example:

- `example(gmm)`
- `head(DTA)`
- `Gen[1:3,1:3]`

In this particular example, the data frame contains information about the genotype identification (ID), the macro-environment (Year), and the spatial information in Block-Row-Column format. Additional columns could be included, such as other covariates to be included into the model. In this function, individuals without genotypic information will treated as a check (fixed effect). Check the example below:

- `demo( fittingMET )`

The output of this function will include prediction (breeding values) of all genotypes that are present in the genotypic matrix - including those without phenotypes. Check the example of how the model can extract field and genetic variation:

- `demo( fieldvar )`

7. Finding substructures

If there are unknown stratification factors in your population, such as heterotic groups, one can use R functions to perform the clusters analysis. Let `gen` be the genotypic matrix and suppose that you want to split the population into two groups. Some unsupervised machine learning approaches include:

a. Using hierarchical clustering

- `Clusters = hclust(dist(gen,method="man"),method="ward.D1")`
- `plot( Clusters )`
- `Stratification1 = cutree(Clusters, k = 2)`

b. Using k-means

- `Stratification2 = kmeans(gen, 2)$cluster`

c. Using multidimensional scaling and k-means

- `MDS = cmdscale(dist(gen),2)`
- `Stratification3 = kmeans(MDS, 2)$cluster`
- `plot(MDS, col = Stratification3)`

8. Other structured populations

Functions `gwas` and `gwas2` are very optimized for NAM populations or populations with a given reference haplote. Suppose that one does not have a reference and it is working with a random population instead, where the subgroups were either defined by unsupervised machine learning methods (section above) or they refer to other sources of structure - such as heterotic groups in maize or maturity zones in soybeans. For that, we implemented the function `gwas3` with the same arguments as previous counterparts. This functions has some interesting properties as well.

- 1. Markers are treated as the interaction between subpopulation and allele
- 2. It is compatible with any allele coding, which allows studies of dominance
- 3. Meta-analysis is facilitated (function `meta3`)

The function `meta3` takes as input a list of association analyis performed by `gwas3`. Only marker that overlap across association studies are evaluated. Nevertheless, the drawback of this function is the requirement for memory in comparison to `gwas2`. Some extra memory is necessary because `gwas3` stores the residual variances for meta-analysis purposes. A demonstration of meta-analysis through `gwas3` is provided by:

- `demo( metagwas )`

Whereas the meta-analysis that preserve the properties of `gwas2` through the function `gwasGE` is provided by:

- `demo( metaGxE )`

9. Further background

- EB-GWAS method:
  `system(paste('open',system.file("doc","gwa_description.pdf",package="NAM")))`
- GxE EB-GWAS:
  `system(paste('open',system.file("doc","gwa_ge_interactions.pdf",package="NAM")))`
- GWP/GWAS methods:
  `system(paste('open',system.file("doc","background_stat_gen.pdf",package="NAM")))`