R package: blupADC

Documents in Chinese language can be found in this ([Chinese documents](#)).

blupADC-OVERVIEW

`blupADC` is an useful and powerful tool for handling genomic data and pedigree data in animal and plant breeding(**traditional blup and genomic selection**). In the design of this package, most of data analysis problems in breeding have been considered, and the speed of calculation is also the key point. In terms of the speed, the core functions of this package are coded by c++ (`Rcpp` and `RcppArmadillo`) , and it also supports parallel calculation by applying `openMP` programming.

`blupADC` provides many useful functions for the whole steps for animal and plant breeding, including pedigree analysis(**trace pedigree, rename pedigree, and correct pedigree errors**), genotype data format conversion(supports **Hapmap, Plink, Blupf90, Numeric, and VCF** format), genotype data quality control and imputation, construction of kinship matrix(**pedigree, genomic and single-step**),and genetic evaluation(by interfacing with two famous breeding softwares, **DMU** and **BLUPF90** in an easy way).

Finally, we kindly provides an easier way of applying `blupADC`, which is a free website([see more details](#)). Most functions of package `blupADC` can be found in this website. Thus, for user who has little code experience, we recommend to use this website(**only need to click and type, that's enough**). But the pitfall of this website is that it can't handle big data.

😊 Good Luck Charlie !

blupADC-Features

- Feature 1. Genomic data format conversion
- Feature 2. Genomic data quality control and genotype imputation
- Feature 3. Breed composition analysis and duplication detection of genomic data
- Feature 4. Pedigree tracing, and analysis
- Feature 5. Pedigree visualization
- Feature 6. Relationship matrix construction(A,G, and H)
- Feature 7. Genetic evaluation with DMU
- Feature 8. Genetic evaluation with BLUPF90

Feature 1.

Overview

`genotype_data_format_conversion` is the basic function of package: `blupADC`. By applying `genotype_data_format_conversion` , we can convert multiple genotype data formats in an easy way, including Hapmap, Plink, BLUPF90, Numeric and VCF.

Note: For analyzing VCF format data, `blupADC` has encapsulated software Plink `version-1.9` and VCFtools `version-0.1.17`. If you want change this version, you could set the related parameters in the part of Advanced parameter.

Example

```
library(blupADC)
sum_data=genotype_data_format_conversion(
         input_data_hmp=data_hmp,  #provided hapmap data object
         output_data_type=c("Plink","BLUPF90","Numeric"),# output data format
         return_result = TRUE,      # return result
         cpu_cores=1                # number of cpu
         )
```

```
library(blupADC)
colnames(data_hmp)[1]="rs#"
data_hmp[1:6,1]=paste0("SNP",1:6)
sum_data=genotype_data_format_conversion(
         input_data_hmp=data_hmp[1:6,],  #provided hapmap data object
         output_data_type=c("Plink","BLUPF90","Numeric"),# output data format
         return_result = TRUE,      # return result
         cpu_cores=1                # number of cpu
                  )
```

Output

```
str(sum_data)
```

According to the result of output, we find that the output contains 5 parts, including:

- **hmp** : `Hapmap` format genotype data

`The first column stands for the name of SNP, the thrid column stands for chromosome, the fourth column stands for the physical postion, and the twelth column and the after columns stand for the genotype data`

rs#	alleles	chrom	pos	strand	assembly	center	protLSID	assayLSID	panelLSID	QCcode	3098	3498	3297	2452
SNP1	NA	1	224488	NA	NA	NA	NA	NA	NA	NA	CC	AC	AC	CC
SNP2	NA	1	293696	NA	NA	NA	NA	NA	NA	NA	GG	TG	TG	GG
SNP3	NA	1	333333	NA	NA	NA	NA	NA	NA	NA	GG	TT	TT	GG
SNP4	NA	1	464830	NA	NA	NA	NA	NA	NA	NA	CC	CC	CC	CC
SNP5	NA	1	722623	NA	NA	NA	NA	NA	NA	NA	AA	GG	GG	AA
SNP6	NA	1	838596	NA	NA	NA	NA	NA	NA	NA	CC	TC	TT	CC

- **ped** : `Plink` 格式的ped数据

`The first column stands for family name,the second column stands for the individual name,the seventh column and the after columns stand for the genotype data`

3098	**3098**	**0**	**0**	**0**	**0**	**C**	**C**	**G**	**G**
3498	3498	0	0	0	0	A	C	T	G
3297	3297	0	0	0	0	A	C	T	G
2452	2452	0	0	0	0	C	C	G	G
4255	4255	0	0	0	0	A	C	G	G
2946	2946	0	0	0	0	C	C	T	G

- **map**：`Plink`格式的map数据

`The first column stands for chromosome, the second column stands for the name of SNP, the thrid column stands for the genetic positon(CM), and the fourth column stands for the physical position`

1	**SNP1**	**0.224488**	**224488**
1	SNP2	0.293696	293696
1	SNP3	0.333333	333333
1	SNP4	0.464830	464830
1	SNP5	0.722623	722623
1	SNP6	0.838596	838596

- **blupf90**：`BLUPF90`格式的基因型数据

`The first column stands for individual name, the second column stands for the genotype data(numeric)`

3098	**200000**
3498	112021
3297	112022
2452	200000
4255	102011
2946	212000

- **numeric**：`Numeric` 格式的基因型数据

`rownames stands for the individual name, colnames stands for the name of SNP, 0,1,2 stand for the numeric genotype`

	SNP1	SNP2	SNP3	SNP4	SNP5	SNP6
3098	2	0	0	0	0	0
3498	1	1	2	0	2	1
3297	1	1	2	0	2	2
2452	2	0	0	0	0	0
4255	1	0	2	0	1	1
2946	2	1	2	0	0	0

Parameter

🐼Basic

- **1: input_data_plink_ped**

User-provided `Plink`-ped format genotype data， `data.frame` class.

- **2:input_data_plink_map**

User-provided `Plink`-map format genotype data， `data.frame` class.

- **3:input_data_hmp**

User-provided `Hapmap` format genotype data， `data.frame` class.

- **4:input_data_BLUPF90**

User-provided `BLUPF90` format genotype data， `data.frame` class.

- **5:input_data_numeric**

User-provided `Numeric` format genotype data， `matrix(numeric)` class.

Note: `input_data_numeric` should contain both rownames and colnames.

In addition, for convenience, users can provide the file name, file path, and file type of genotype data directly without reading them in R environment.

- **6:input_data_type**

  File type of provided genotype data， `character` class.

  - Hapmap
  - Plink
  - BLUPF90
  - Numeric
  - VCF
- **7:input_data_path**

File path of provided genotype data， `character` class.

- **8:input_data_name**

File name of provided genotype data， `character` class.

Note: if input_data_type is Plink or VCF, user don't need to include suffix in the file name of genotype data.

eg. for Plink type data, files name are test1.map and test1.ped， we should set `input_data_name="test1"`.

- **9:return_result**

Whether return result, `logical` class. Default is FALSE.

Additionally, for convenience, users can save output genotype data into local computer .

- **10:output_data_path**

File path of output genotype data, `character` class.

- **11:output_data_name**

File name of output genotype data, `character` class.

- **12:output_data_type**

  File type of output genotype data, `character` class.

  - Hapmap
  - Plink
  - BLUPF90
  - Numeric
  - VCF

Note: users can output multiple formats of genotype data simultaneously. e.g. `output_data_type=c("Hapmap","Plink","BLUPF90","Numeric")`, outout 4 types of genotype data simultaneously .

Advanced

- **13:cpu_cores**

Number of cpu in calculating, `numeric` class. Default is 1.

- **14:miss_base**

Missing genotype character, `character` class. Default is "NN".

- **15:miss_base_num**

Missing genotype number after numeric conversion, `numeric` class. Default is 5.

- **16:plink_software_path**

Path of plink software in user computer. `character` class.

- **17:plink_software_name**

Name of plink software in user computer. `character` class.

- **18:vcftools_software_path**

Path of vcftools software in user computer. `character` class.

- **19:vcftools_software_name**

Name of vcftools software in user computer. `character` class.

Feature 2.

Overview

🤘 Generally, most genotype data need to perform quality control and imputation before applying in animal and plant breeding. For convenience, package `blupADC` provides `genotype_data_QC_Imputation` function to perform quality control and imputation by interfacing with software **Plink** and software **Beagle** in an easy way (we only need to provide the software path and software name).

👉 **Note: For convenience, `blupADC` has encapsulated software Plink(for quality control) `version-1.9` and software Beagle(for imputation) `version-5.2`. If you want change this version, you should set the related parameters in the part of Advanced parameter.**

Example

```
library(blupADC)
genotype_data_QC_Imputation(
            input_data_hmp=data_hmp,    #provided hapmap data object
            data_analysis_method="QC_Imputation",   #analysis method type,QC +
imputatoin
            output_data_path="/root/result",        #output data path
            output_data_name="YY_data",             #output data name
            output_data_type="Plink"                #output data format
            )
```

In the process of quality control and imputation, we should provide genotype data, these parameters are the as in `genotype_data_format_conversion` function(see more details).

Parameter

💗Basic

- **1: data_analysis_method**

  Method of analyzing data, `character` class.

  - `"QC"` : only perform quality control
  - `"Imputation"` : only perform imputation
- `"QC_Imputation"` : first perform quality control, and then perform imputation

- **2: qc_snp_rate**

Threshold of SNPs call rate in quality control, `numeric` class. Default is 0.1

- **3: qc_ind_rate**

Threshold of individuals call rate in quality control, `numeric` class. Default is 0.1

- **4: qc_maf**

Threshold of minor allele frequency(MAF) in quality control, `numeric` class. Default is 0.05

- **5: qc_hwe**

Threshold of hardy weinberg equilibrium(HWE) in quality control, `numeric` class. Default is 1e-7

💗Advanced

- **6：plink_software_path**

Path of software **Plink** , `character` class.

- **7：plink_software_name**

Name of software **Plink** , `character` class.

- **8：beagle_software_path**

Path of software **Beagle** , `character` class.

- **9：beagle_software_name**

Name of software **Beagle** , `character` class.

- **10：beagle_ref_data_path**

File path of reference data in imputation, `character` class.

- **11：beagle_ref_data_name**

File name of reference data in imputation, `character` class.

- **12：beagle_ped_path**

File path of pedigree data in imputation, `character` class.

- **13：beagle_ped_name**

File name of pedigree data in imputation, `character` class.

Feature 3.

Overview

🧑 Breed composition analysis is usually a problem in data analysis. In package: `blupADC`, user can solve this problem by applying `genotype_data_check` function. In addition, user can detect the duplication of genomic data easily by applying `genotype_data_check` function.

Example

Breed composition analysis

```
library(blupADC)
check_result=genotype_data_check(
                  input_data_hmp=PCA_data_hmp,   #provided hapmap data object
                  duplication_check=FALSE,       #whether check the duplication
of genotype
                  breed_check=TRUE,              # whether check the record of
breed
                  breed_record=PCA_Breed,        # provided breed record
                  return_result=TRUE             #return result
                  )
```

Check duplication

```
library(blupADC)
check_result=genotype_data_check(
                  input_data_hmp=data_hmp,   #provided hapmap data object
                 duplication_threshold=0.95, #threshold of duplication
                  duplication_check=TRUE,    #whether check the duplication of
genotype
                  breed_check=FALSE,         # whether check the record of breed
                  return_result=TRUE         #return result
                  )
```

Output

The result of output mainly contains two parts, including:

- **duplicated_genotype**

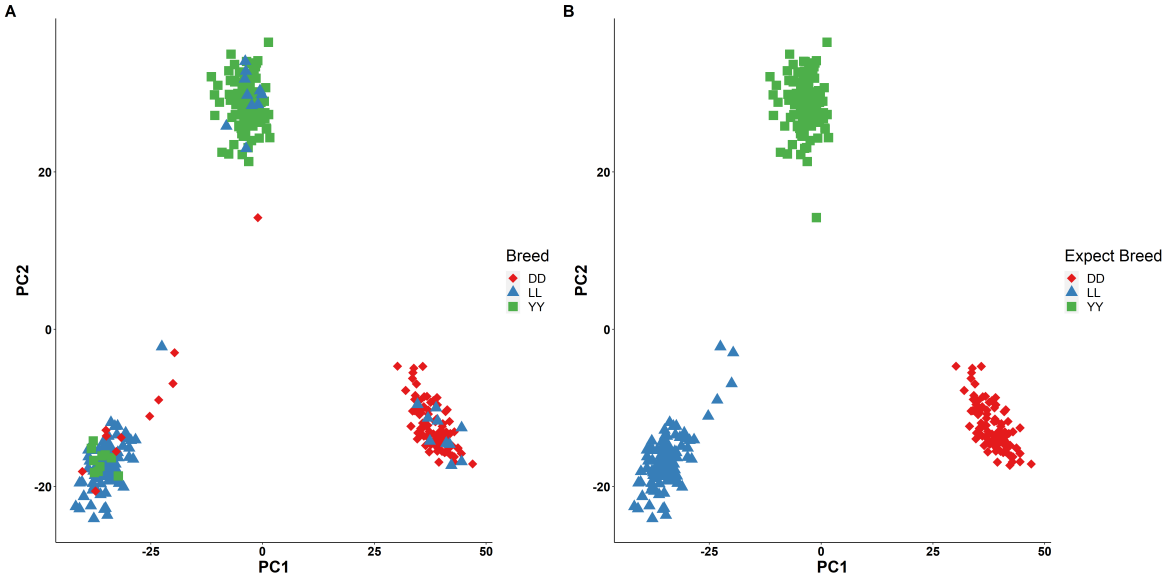
IND1	**IND1**	**1**
IND2	IND2	1
IND3	IND3	1
IND4	IND4	1

The first and the second column is the name of individual, the third column is the percentage of overlap.

- **pca_outlier**

Id	**Breed**	**Expeced_Breed**
IND1	YY	DD
IND100	LL	DD
IND12	YY	DD
IND14	YY	DD
IND16	YY	DD
IND18	YY	DD

Figure A is the PCA result before correcting breed record , Figure B is the PCA result after correcting breed correcting record



Parameter

Many parameters in `genotype_data_overlap` are the same as in `genotype_data_format_conversion` function (see more details).

Thus, we will introduce specific parameters in `genotype_data_overlap` function.

- **1: selected_snps**

Number of SNPs in detecting overlap, `numeric` class. Default is 1000.

- **2: overlap_threshold**

Threshold of duplicate genotype, `numeric` class. Default is 0.95.

- **3: duplication_check**

Whether check duplication of genotype, `logical` class. Default is TRUE.

- **4: breed_check**

Whether check breed record of genotype, `logical` class. Default is FALSE.

- **5: ind_breed**

Breed record of individuals, `data.frame` class.

The format of `ind_breed` is showing as follow:

Id	Breed
IND1	YY
IND2	YY
IND3	YY
IND4	YY
IND5	YY
IND6	YY

When the proportion of genotype data between two individuals is larger than this threshold, these two individuals will be regarded as the same individual.

Feature 4.

Overview

🐴 Pedigree is the important information in animal breeding. By applying `trace_pedigree` function in package: `blupADC`, user can trace, rename, correct pedigree errors in an easy way. In addition, user can visualize the pedigree structure by `ggped` function.

Example

💖Trace pedigree

```
library(blupADC)
pedigree_result=trace_pedigree(
                input_pedigree=origin_pedigree,   #provided pedigree data object
                output_pedigree_tree=TRUE  # output pedigree tree
                )
```

Output

By typing `str(pedigree_result)`, we can get the output result of this function:

```
str(pedigree_result)
## List of 5
##  $ ped          : chr [1:15945, 1:3] "DD19348310" "DD19386807" "DD19119705"
"DD16007415" ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "Offspring" "Sire" "Dam"
##  $ rename_ped   :'data.frame':   15945 obs. of  6 variables:
##   ..$ Offspring   : chr [1:15945] "DD19348310" "DD19386807" "DD19119705"
"DD16007415" ...
##   ..$ Generation  : num [1:15945] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ Offspring_Id: int [1:15945] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ Sire_Id     : num [1:15945] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ Dam_Id      : num [1:15945] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ Order       : int [1:15945] 1 2 3 4 5 6 7 8 9 10 ...
##  $ pedigree_tree: chr [1:15945, 1:15] "DD19348310" "DD19386807" "DD19119705"
"DD16007415" ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:15] "Offspring" "Sire" "Dam" "SireSire" ...
##  $ error_id_set :List of 4
##   ..$ error_duplicated_id: chr [1:24] "DD19119705" "DD20488904" "DD20153801"
"DD20376912" ...
##   ..$ error_sex_id: chr "DD13006182"
##   ..$ error_breed_id: NULL
##   ..$ error_birth_date_id: NULL
```

Output result includes several parts:

- **ped:** pedigree without rename

- **rename_ped:** renamed pedigree. The first column is original id, the second column is generation, columns 3-5 stand for the renamed pedigree.
- **pedigree_tree:** pedigree tree. Pedigree tree contains ancestry records information for each individual. For saving time, software doesn't output pedigree tree in default.
- **error_id_set:** dataset of pedigree errors .According to the type of pedigree errors, these datasets can be divided four parts:
  - `error_duplicated_id`: same individual but has different records of sire and dam
  - `error_sex_id`: same individual appears in the column of sire and dam simultaneously
  - `error_breed_id`: breed of parents and offspring is different (only for specify format of original id)
  - `error_birth_date_id`: offspring born before its parents (need to provide birth data information in the fourth column of pedigree)

Parameter

💥Basic

- **1: input_pedigree**

User-provided pedigree data, `data.frame` or `matrix` class.

📩The format of provided pedigree data should be one of the following format:

- 3 columns format:

Offspring	Sire	Dam
DD19575312	DD18768902	DD16376015
DD19513112	DD18768902	DD17111017
DD20348012	DD19132207	DD19234510
DD20361110	DD19331001	DD19293112
DD20471212	DD19331001	DD19320808
DD20564818	DD19331001	DD19311009

- 4 columns format:

Offspring	Sire	Dam	Birth_Date
DD19575312	DD18768902	DD16376015	20200101
DD19513112	DD18768902	DD17111017	20200102
DD20348012	DD19132207	DD19234510	20200103
DD20361110	DD19331001	DD19293112	20200104
DD20471212	DD19331001	DD19320808	20200105

Offspring	Sire	Dam	Birth_Date
DD20564818	DD19331001	DD19311009	20200106

- Multiple columns format:

Offspring	Sire	Dam	SireSire	DamSire	SireSireSire
DD20231905	DD19581002	DD18750810	DD16785512	DD15507717	DD14008512
DD20458701	DD19564302	DD18925809	DD15349017	DD15245411	DD16771212
DD20324707	DD19232903	DD18571012	DD16794714	DD16744412	DD16714516
DD19288609	DD18713408	DD18552609	DD15180015	DD15479214	DD15243711
DD16222012	DD15145005	DD15378812	DD14110014	DD15501518	DD15206217
DD17684713	DD16672107	DD15122311	DD15505715	DD15347415	DD16383111

Note: When the format of provided pedigree data is multiple columns， the colnames of pedigree data should be the specify format , e.g. `SireSire` stands for the father of offspring's father, `SirSireSire` stands for the father of `SireSire`

Missing record in pedigree could be set as **NA** or **0** .

- **2：input_pedigree_path**

File path of pedigree data, `character` class.

- **3：input_pedigree_name**

File name of pedigree data, `character` class.

- **4：pedigree_format_conversion**

Whether convert multiple columns pedigree into standard 3 columns pedigree, `logical` class. Whether the format of provided pedigree data is multiple columns, user need to set `pedigree_format_conversion=TRUE` .

- **5：output_pedigree_path**

File path of output pedigree data, `character` class.

- **6：output_pedigree_name**

File path of output pedigree name, `character` class.

Advanced

- **7：dup_error_check**

Whether check the pedigree error of error_duplicated， `logical` class. Default is TRUE.

- **8：sex_error_check**

Whether check the pedigree error of error_sex， `logical` class. Default is TRUE.

- **9：breed_error_check**

Feature 5

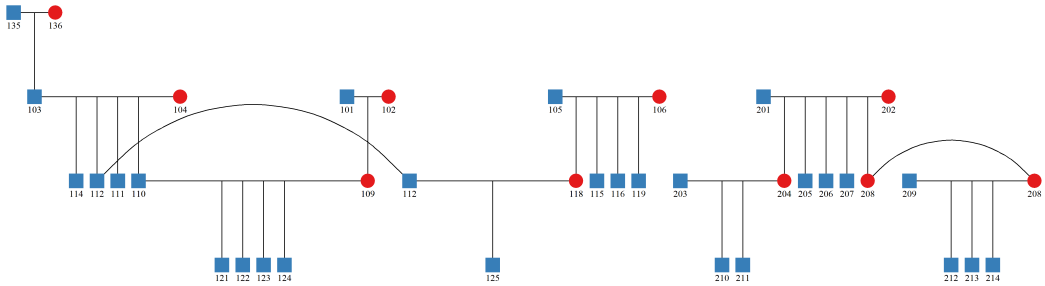
Overview

An intuitive and clear structure of pedigree could help breeders to make better decision in breeding plan. By applying `ggped` function, user can plot the structure of pedigree in an easy way.

Example

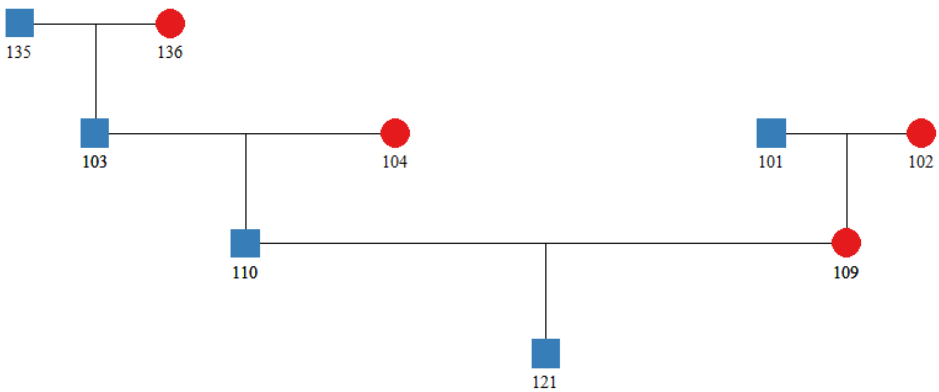
Plot whole pedigree

```
library(blupADC)
pedigree_result=ggped(
                input_pedigree=plot_pedigree
                )
```



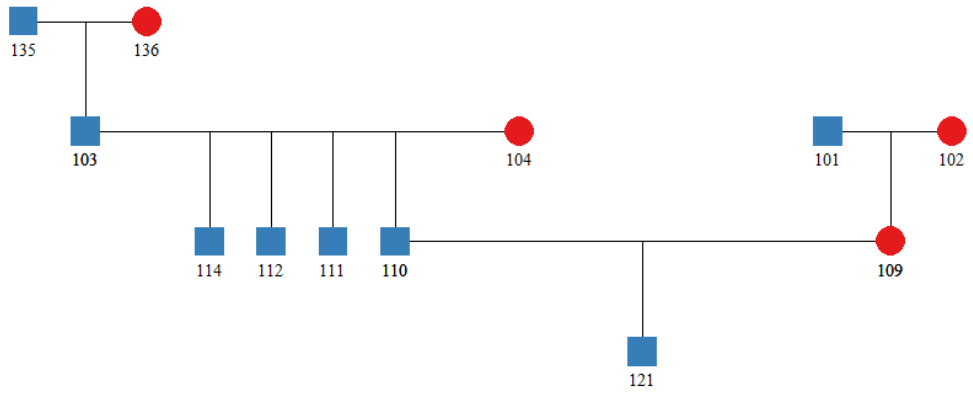
Plot subset of whole pedigree

```
library(blupADC)
pedigree_result=ggped(
                input_pedigree=plot_pedigree,
                trace_id=c("121")             #provided subset-id
                )
```



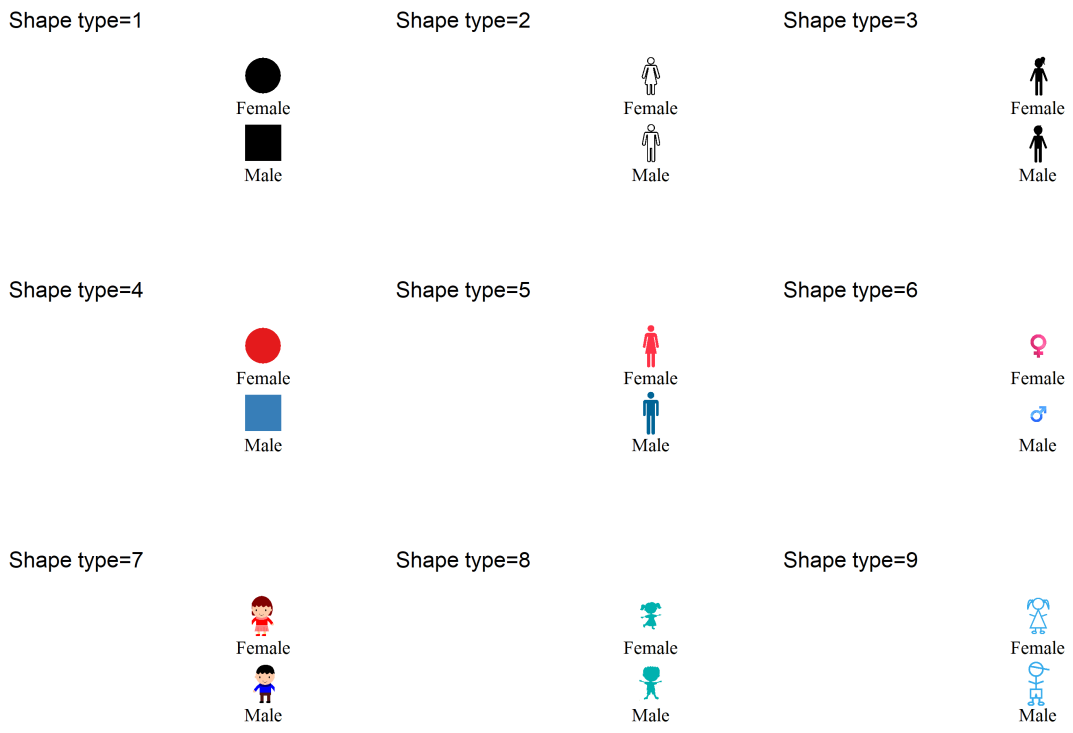
Plot subset of whole pedigree (with sibs)

```
library(blupADC)
pedigree_result=ggped(
                input_pedigree=plot_pedigree,
                trace_id=c("121"),
                trace_sibs=TRUE   #whether plot the sibs of subset-id
                )
```



Change the style of picture

User can change the style of pedigree by modifying the `shape_type` parameter. Default `shape_type` is 4.



Output

The output of `ggped` is the object of `ggplot`, user can plot the pedigree structure or save it directly.

Parameter

🎇Basic

- **1: input_pedigree**

User-provided pedigree data, `data.frame` or `matrix` class.

📩The format of provided pedigree data should be 3 columns format:

Offspring	Sire	Dam
DD19575312	DD18768902	DD16376015
DD19513112	DD18768902	DD17111017
DD20348012	DD19132207	DD19234510
DD20361110	DD19331001	DD19293112
DD20471212	DD19331001	DD19320808
DD20564818	DD19331001	DD19311009

- **2: trace_id**

Individuals set in tracing pedigree, `character` class. Default is `NULL` (plot whole individuals in pedigree)

- **3: trace_sibs**

Whether tracing the sibs of individuals set , `logical` class. Default is `FALSE` .

- **4: ind_sex**

Sex record of individuals , `data.frame` class.

The format of this data is showing as follow:

Individual	Sex
101	Male
102	Female
103	Male
104	Female

For individual who doesn't have the record of sex, the sex of this individual would be set as Male.

- **5: plot_family**

Whether showing family structure when plotting the pedigree, `logical` class. Default is `FALSE` .

- **6: shape_type**

The shape type when plotting pedigree , `numeric` class. Default is 4.

Advanced

- **7: shape_size**

The shape size of picture, `numeric` class. Default is `8` .

- **8: ind_text_size**

The size of individual name, `numeric` class. Default is `4` .

- **9: ind_text_vjust**

The vjust of individual name, `numeric` class. Default is `3` .

Feature 6

Overview

In the application of animal and plant breeding, the key step is the construction of kinship matrix. Package: `blupADC` provides `cal_kinship` function which can construct various type of relationship matrix directly, including **additive relationship matrix**(pedigree, genomic and single-step), and **dominance relationship matrix**(pedigree, genomic and single-step), and **the inverse matrix** of these kinship matrix.

In the construction of single-step relationship matrix, users can select metafounder algorithm or APY algorithm. In terms of the construction of dominance relationship matrix, users can select different coding manners for genomic dominance relationship matrix, gene dropping algorithm for pedigree dominance relationship matrix. In addition, `cal_kinship` can calculate several types of inbreeding coefficients (pedigree, genomic, and single-step) .

In the construction of single-step relationship matrix, users can select metafounder algorithm. In terms of the construction of dominance relationship matrix, users can select different coding manners for genomic dominance relationship matrix, gene dropping algorithm for pedigree dominance relationship matrix. In addition, `cal_kinship` can calculate several types of **inbreeding coefficients** (pedigree, genomic, and single-step) .

Example

Pedigree-based kinship matrix

```
library(blupADC)
kinship_result=cal_kinship(
                input_pedigree=origin_pedigree,          #provided hapmap data
object
                kinship_type=c("P_A"),      #type of  kinship matrix
                inbred_type=c("Pedigree"),      #type of inbreeding coefficients
                return_result=TRUE)               #return result
```

Note: In the construction of pedigree and single-step relationship matrix, people need to provide genotype data.

Genomic-based kinship matrix

```
library(blupADC)
kinship_result=cal_kinship(
                input_data_hmp=data_hmp,          #provided hapmap data object
                kinship_type=c("G_A","G_D"),      #type of  kinship matrix
                dominance_type=c("genotypic"),    #type of dominance effect
                inbred_type=c("Homozygous"),      #type of inbreeding
coefficients
                return_result=TRUE)               #return result
```

Note: In the construction of genomic and single-step relationship matrix, people need to provide genotype data. These parameters are the same as in `genotype_data_format_conversion` function((see more details).

Single-step based kinship matrix

```
library(blupADC)
kinship_result=cal_kinship(
                input_data_hmp=data_hmp,          #provided hapmap data object
                input_pedigree=origin_pedigree,
                kinship_type=c("H_A"),      #type of  kinship matrix
                inbred_type=c("H_diag"),      #type of inbreeding coefficients
                return_result=TRUE)               #return result
```

Parameter

Basic

- **1: kinship_type**

  Type of kinship matrix, `character` class. User can select multiple types simultaneously, including:

  - "G_A" : genomic additive kinship matrix
  - "G_Ainv" :inverse of genomic additive kinship matrix
  - "G_D" :genomic dominance kinship matrix
  - "G_Dinv" :inverse of genomic dominance kinship matrix
  - "P_A" :pedigree additive kinship matrix
  - "P_Ainv" :inverse of pedigree additive kinship matrix
  - "P_D" : pedigree dominance kinship matrix
  - "P_Dinv" :inverse of pedigree dominance kinship matrix
  - "H_A" :single-step additive kinship matrix
  - "H_Ainv" :inverse of single-step additive kinship matrix
  - "H_D" :single-step dominance kinship matrix
  - "H_Dinv" :inverse of single-step dominance kinship matrix

Note: In the construction of pedigree and single-step relationship matrix, user need to provide pedigree data. In the construction of genomic and single-step relationship matrix, user need to provide genotype data.

- **2: dominance_type**

  Type of dominance effect in the construction of dominance relationship matrix , `character` class.

  - "genotypic" : coded by $0-2pq$, $1-2pq$, and $0-2pq$ for AA, Aa, and aa, respectively.
  - "classical" : coded by $-2q^2$, $2pq$, and $-2p^2$ for AA, Aa, and aa, respectively.

> More details about these two types dominance effects could be seen in this reference： [On the Additive and Dominant Variance and Covariance of Individuals Within the Genomic Selection Scope]

- **3：inbred_type**

  Type of inbreeding coefficients, `character` class.

  - "Homozygous" : proportion of homozygous sites
  - "G_Diag" : diagonal of genomic additive relationship matrix minus 1
  - "H_diag" :diagonal of single-step additive relationship matrix minus 1
  - "Pedigree" :diagonal of pedigree additive relationship matrix minus 1
- **4：input_pedigree**

User-provided pedigree data， `data.frame` or `matrix` class. (see more details about the format of pedigree data)

- **5：pedigree_rename**

Whether pedigree need to be renamed, `logical` class. Default is TRUE.

- **6：IND_geno_rename**

Whether genotype individuals need to be renamed according to the renamed pedigree, `logical` class. Default is FALSE.

- **7：rename_to_origin**

Whether renamed individuals should keep original name, `logical` class. Default is FALSE.

- **8：output_matrix_type**

  Type of output kinship matrix type, `character` class. Default is "col_three".

  - "`col_all`" : n*n format
  - "`col_three`" : 3 columns format. The first and the second column are the name of individuals, the third column is the relationship coefficients 。

1001	1001	0.989
1001	1002	0.421
1001	1003	0.567

- **9：output_matrix_path**

File path of output relationship matrix, `character` class.

- **10：output_matrix_name**

File name of output relationship matrix, `character` class.

Advanced

- **11：cpu_cores**

Number of cpu in calculating, `numeric` class. Default is 1.

- **12：kinship_base**

Whether take $p = q = 0.5$ in the construction of relationship matrix, `logical` class. Default is FALSE.

- **13：kinship_trace**

Whether take the trace of kinship matrix to scale relationship matrix, `logical` class. Default is FALSE.

- **14：Metafounder_algorithm**

Whether take the metafounder algorithm to construct single-step relationship matrix, `logical` class. Default is FALSE.

- **15：APY_algorithm**

Whether take the APY algorithm to construct inverse relationship matrix, `logical` class. Default is FALSE.

- **16：APY_eigen_threshold**

Threshold of variation explained by eigenvalues , `numeric` class. Default is 0.95.

- **17：APY_n_core**

Number of core animals , `numeric` class. Default is NULL.

- **18：SSBLUP_omega**

The value of omega in the construction of single-step additive relationship matrix, `numeric` class. Default is 0.05.

- **19：gene_dropping**

Whether take the gene dropping algorithm to construct pedigree dominance relationship matrix, `logical` class. Default is FALSE.

- **20：gene_dropping_iteration**

The number of iteration for gene dropping algorithm , `numeric` class. Default is 1000.

- **21：memory_save**

Whether take memory efficient way to construct kinship matrix, `logical` class. Default is FALSE. (This way need more storage space)

Feature 7

Overview

😇 In the previous section, we have given detailed description about data preparation. In the following section, we will introduce genetic evaluation software in animal and plant breeding. Nowadays, in the filed of animal and plant breeding, two of the most famous breeding software are **DMU** and **BLUPF90** (cited over than one thousand).

Although these two softwares have many advantages, these two softwares have one common pitfall: it is a little difficult to use for freshman(need to prepare parameter file). Thus, in order to overcome this pitfall, package `blupADC` provides `run_DMU` and `run_BLUPF90` for interfacing **DMU** and **BLUPF90** in an easy way.

In this section, we will give detail description about `run_DMU` function.

👉 **Note: Package `blupADC` has encapsulated the basic module of DMU(`dmu1`, `dmuai`, and `dmu5`), more modules could be download from website**(DMU download website).

For commercial use of DMU, user must contact the author of DMU !!!

Example

Single trait model - Pedigree BLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  path of provided files

run_DMU(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1"),                          #trait name
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),     #fixed effect
name
        random_effect_name=list(c("Id","Litter")),               #random effect
name
        covariate_effect_name=NULL,                              #covariate
effect name
        phe_path=data_path,                          #path of phenotype file
        phe_name="phenotype.txt",                    #name of phenotype file
        integer_n=5,                                 #number of integer variable
        analysis_model="PBLUP_A",                    #model of genetic
evaluation
        dmu_module="dmuai",                          #modeule of estimating
variance components
        relationship_path=data_path,                 #path of relationship file
        relationship_name="pedigree.txt",            #name of relationship file
        output_result_path="/root"                   # output path
        )
```

Single trait model - GBLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  path of provided files
run_DMU(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1"),                          #trait name
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),     #fixed effect
name
        random_effect_name=list(c("Id","Litter")),               #random effect
name
        covariate_effect_name=NULL,                              #covariate
effect name
        phe_path=data_path,                          #path of phenotype file
        phe_name="phenotype.txt",                    #name of phenotype file
        integer_n=5,                                 #number of integer variable
        analysis_model="GBLUP_A",                    #model of genetic
evaluation
        dmu_module="dmuai",                          #modeule of estimating
variance components
```

```
        relationship_path=data_path,                #path of relationship file
        relationship_name="G_Ainv_col_three.txt",           #name of
relationship file
        output_result_path="/root"                  # output path
        )
```

Single trait model - Single-step BLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  path of provided files
run_DMU(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2")
, # colnames of phenotype
        target_trait_name=c("Trait1"),                          #trait name
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),     #fixed effect
name
        random_effect_name=list(c("Id","Litter")),               #random effect
name
        covariate_effect_name=NULL,                             #covariate
effect name
        phe_path=data_path,                         #path of phenotype file
        phe_name="phenotype.txt",                   #name of phenotype file
        integer_n=5,                                #number of integer variable
        analysis_model="SSBLUP_A",                   #model of genetic
evaluation
        dmu_module="dmuai",                         #modeule of estimating
variance components
        relationship_path=data_path,                #path of relationship file
        relationship_name=c("pedigree.txt","G_A_col_three.txt"),
#name of relationship file
        output_result_path="/root"                  # output path
        )
```

Through modifying the two parameters: `analysis_model` and `relationship_name` , we can perform Pedigree-BLUP, GBLUP, and SSBLUP analysis (PS: we can get G_Ainv_col_three.txt and G_A_col_three.txt by `cal_kinship` function, [see more details]() about `cal_kinship` function).

The above example is single-trait model, while in actual breeding, multiple traits model is also common. Similarly, we only need to modify several parameters to perform multiple traits model:

Multiple traits model - Pedigree BLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  path of provided files

run_DMU(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1","Trait2"),                          #trait
name

fixed_effect_name=list(c("Sex","Herd_Year_Season"),c("Herd_Year_Season")),
#fixed effect name
        random_effect_name=list(c("Id","Litter"),c("Id")),               #random
effect name
```

```
        covariate_effect_name=list(NULL,"Age"),
#covariate effect name
        phe_path=data_path,                          #path of phenotype file
        phe_name="phenotype.txt",                    #name of phenotype file
        integer_n=5,                                 #number of integer variable
        analysis_model="PBLUP_A",                    #model of genetic
evaluation
        dmu_module="dmuai",                          #modeule of estimating
variance components
        relationship_path=data_path,                 #path of relationship file
        relationship_name="pedigree.txt",            #name of relationship file
        output_result_path="/root"                   # output path
        )
```

Parameter

🐼Basic

- **1：phe_path**

File path of phenotype data， `character` class。

- **2：phe_name**

File name of phenotype data， `character` class。

Note: User-provided phenotype doesn't have colnames (the same as the requirement of DMU)

- **3：phe_col_names**

Colnames of phenotype data， `character` class。

- **4：integer_n**

Number of integer variable, `numeric` class。

- **5：genetic_effect_name**

Genetic effect name (usually is the individual name), `character` class.

- **6：target_trait_name**

Target trait name, `character` class.

For multiple traits model, we should set target_trait_name as character vector, e.g. `target_trait_name=c("Trait1","Trait2")`

- **7：fixed_effect_name**

Fixed effects name, `list` class.

For multiple traits model, the order of fixed effects name should correspond to the target trait name.

eg. `target_trait_name=c("Trait1","Trait2")`

`fixed_effect_name=list(c("Sex","Herd_Year_Season"),c("Herd_Year_Season"))`

which means the fixed effects name of trait1 is : `c("Sex","Herd_Year_Season")`, the fixed effect name of trait2 is : `c("Herd_Year_Season")`

- **8：random_effect_name**

Random effects name, `list` class.

For multiple traits model, the order of random effects name should correspond to the target trait name.

eg. `target_trait_name=c("Trait1","Trait2")`

`random_effect_name=list(c("Id","Litter"),c("Id"))`

which means the random effects name of trait1 is : `c("Id","Litter")`, the random effects name of trait2 is : `c("Id")`

- **9：covariate_effect_name**

Covariate effects name, `list` class.

For multiple traits model, the order of covariate effects name should correspond to the target trait name.

eg. `target_trait_name=c("Trait1","Trait2")`

`covariate_effect_name=list(NULL,"Age")`

which means the covariate effects name of trait1 is : `NULL` (NULL means no this effect), the covariate effects name of trait2 is : `Age`

- **10：analysis_model**

  Model of genetic evaluation, `character` class.

  - `"PBLUP_A"` : Pedigree BLUP- additive model
  - `"GBLUP_A"` :GBLUP- additive model
  - `"GBLUP_AD"` :GBLUP- additive and dominance model
  - `"SSBLUP_A"` :SSBLUP- additive model
  - `"User_define"`: User define model
- **11：dmu_module**

  Module of estimating variance components, `character` class.

  - `"dmuai"`
  - `"dmu4"`
  - `"dmu5"`
- **12：DMU_software_path**

Path of DMU software, `character` class.

- **13：relationship_path**

File path of relationship data, `character` class.

- **14：relationship_name**

File name of relationship data, `character` class.

For different genetic evaluation model , we should provide different relationship file.

E.g. for "PBLUP_A" model, we need to provide pedigree file, then we should set `relationship_name="pedigree.txt"` ;

for "GBLUP_A" model, we need to provide inverse of additive relationship matrix file(3 columns format), then we should set `relationship_name="G_Ainv_col_three.txt"` ;

for "SSBLUP_A" model, we need to provide pedigree and additive relationship matrix file(3 columns format), then we should set `relationship_name=c("pedigree.txt","G_A_col_three.txt")` ;

- **15：output_result_path**

Path of output DMU result, `character` class.

- **16：output_ebv_path**

File path of output EBV, `character` class. Default is equal to output_result_path

- **17：output_ebv_name**

File name of output EBV, `character` class.

Advanced

- **18：provided_effect_file_path**

File path of trait's model effect data, `character` class.

File of trait's model effect include fixed effects name, random effects name, and covariate effects name. Once user provides this file, user don't need to set these three parameters: `fixed_effect_name` `random_effect_name` `covariate_effect_name`.

The format of this effect file is as following:

V1	V2	V3	V4	V5	V6	V7	V8	V9
Trait1	*	Sex	Herd_Year_Season	*	Id	Litter	*	*
Trait2	*	Sex	*	Id	*	Age	*	

The first column is the name of target trait. Each column stands for one effect name. In order to recognize three types of effect, we set * to distinguish each type.

Effects name between the first * and the second * stand for fixed effects name;

effects name between the second * and the third * stand for random effects name;

effects name between the third * and the fourth * stand for covariate effects name.

- **19：provided_effect_file_name**

File name of trait's model effect data, `character` class.

- **20：provided_DIR_file_path**

File path of user-provided DIR data, `character` class.

- **21：provided_DIR_file_name**

File name of user-provided DIR data, `character` class.

- **22：included_permanent_effect**

Whether perform permanent-environment analysis, `logical` class. Default is FALSE.

- **23：dmu_algorithm_code**

Number of dmu-module algorithm, `numeric` class.

- **24：provided_prior_file_path**

File path of user-provided prior file, `character` class.

- **25: provided_prior_file_name**

File name of user-provided prior file, `character` class.

- **26: missing_value**

Missing value in phenotype file, `numeric` class. Default is -9999.

- **27: iteration_criteria**

Value of iteration convergence, `numeric` class. Default is 1.0e-7.

- **28: genetic_effect_number**

Number of genetic effect in SOL file, `numeric` class. Default is 4.

- **29: residual_cov_trait**

Traits combination of assuming residual-covariance equals to 0. e.g `residual_cov_trait=list(c("Trait1","Trait2"))`

- **30: selected_id**

Individuals set of output EBV, `character` class.

- **31: cal_debv**

Whether calculate de-regressed EBV(DEBV), `logical` class. Default is FALSE.

- **32: debv_pedigree_path**

File path of pedigree data for calculating DEBV, `character` class.

- **33: debv_pedigree_name**

File name of pedigree data for calculating DEBV, `character` class.

Feature 8

Oveview

In the previous section, we have given detailed description about the interface with **DMU** by `run_DMU` function. In this chapter, we will introduce the usage of `run_BLUPF90` function.

Note: the usage of `run_BLUPF90` and `run_DMU` is similar. Thus, we recommend to have a look at the usage of `run_DMU` function.

Note: Package `blupADC` has encapsulated the basic module of BLUPF90 (`renumf90`, `remlf90`, `airemlf90`, and `blupf90`), if you want to use more modules, please download from websit (BLUPF90 download website).

For commercial use of BLUPF90, user must contact the author of BLUPF90 !!!

Example

Single trait model - Pedigree BLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  path of provided files
```

```
run_BLUPF90(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1"),                          #trait name
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),     #fixed effect
name
        random_effect_name=list(c("Id","Litter")),               #random effect
name
        covariate_effect_name=NULL,                              #covariate
effect name
        phe_path=data_path,                          #path of phenotype file
        phe_name="phenotype.txt",                    #name of phenotype file
        analysis_model="PBLUP_A",                    #model of genetic
evaluation
        relationship_path=data_path,                 #path of relationship file
        relationship_name="pedigree.txt",            #name of relationship file
        output_result_path="/root"                   # output path
        )
```

Single trait model - GBLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  path of provided files

run_BLUPF90(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1"),                          #trait name
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),     #fixed effect
name
        random_effect_name=list(c("Id","Litter")),               #random effect
name
        covariate_effect_name=NULL,                              #covariate
effect name
        phe_path=data_path,                          #path of phenotype file
        phe_name="phenotype.txt",                    #name of phenotype file
        analysis_model="GBLUP_A",                    #model of genetic
evaluation
        relationship_path=data_path,                 #path of relationship file
        relationship_name="blupf90_genumeric",            #name of relationship
file
        output_result_path="/root"                   # output path
        )
```

Single trait model - Single-step BLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  path of provided files

run_BLUPF90(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
```

```
        target_trait_name=c("Trait1"),                          #trait name
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),     #fixed effect
name
        random_effect_name=list(c("Id","Litter")),               #random effect
name
        covariate_effect_name=NULL,                              #covariate
effect name
        phe_path=data_path,                          #path of phenotype file
        phe_name="phenotype.txt",                    #name of phenotype file
        analysis_model="SSBLUP_A",                    #model of genetic
evaluation
        relationship_path=data_path,                 #path of relationship file
        relationship_name=c("pedigree.txt","blupf90_genumeric"),
#name of relationship file
        output_result_path="/root"                  # output path
        )
```

Similar to `run_DMU` function, through modifying the two parameters: `analysis_model` and `relationship_name` , we can perform Pedigree-BLUP, GBLUP, and SSBLUP analysis(PS: blupf90_genumeric is generated through `cal_kinship` function, [see more details](#) about `cal_kinship` function).

Multiple traits model - Pedigree BLUP

The following code is about the usage of multiple traits model through BLUPF90:

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  path of provided files

run_BLUPF90(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1","Trait2"),                          #trait
name

fixed_effect_name=list(c("Sex","Herd_Year_Season"),c("Herd_Year_Season")),
#fixed effect name
        random_effect_name=list(c("Id","Litter"),c("Id")),               #random
effect name
        covariate_effect_name=list(NULL,"Age"),
#covariate effect name
        phe_path=data_path,                          #path of phenotype file
        phe_name="phenotype.txt",                    #name of phenotype file
        analysis_model="PBLUP_A",                    #model of genetic
evaluation
        relationship_path=data_path,                 #path of relationship file
        relationship_name=c("pedigree.txt"),            #name of relationship
file
        output_result_path="/root"                   # output path
        )
```

Parameter

Many parameters in `run_BLUPF90` are the same as in `run_DMU` function([see more details](#)).

🐲Thus, we will introduce specific parameters in `run_BLUPF90` function.

- **1：BLUPF90_algorithm**

  Algorithm of estimating variance components, `character` class. Default is "EM_REML".

  - `"AI_REML"`
  - `"EM_REML"`
  - `"BLUP"` : no need to estimate variance components, solve mixed linear model directly.
- **2：provided_blupf90_prior_file_path**

File path of user-provided prior file, `character` class.

Note: The format of BLUPF90 prior file is different to the format of DMU prior file. In the next section, i will give a detailed introduction.

- **3：provided_blupf90_prior_file_name**

File name of user-provided prior file, `character` class.

- **4：provided_blupf90_prior_effect_name**

Effect name of user-provided prior file, `character` class.

- **5：BLUPf90_software_path**

Path of software BLUPF90 , `character` class.