{{}}

blupADC-简介

`blupADC` 是一个专注于分析动植物育种中的系谱数据、基因型数据及遗传评估的工具。在设计该工具时，我们对数据处理时可能遇到的各种问题均进行了详细的考量(**ps.如果您有好的建议，请积极联系作者!**)。此外，为了提高分析效率， `blupADC` 可支持并行计算(通过 `openMP`)，并且 `blupADC` 中的核心函数均通过c++(`Rcpp` and `RcppArmadillo`) 进行编写。

`blupADC` 提供了许多有用的功能在整个动植物育种的流程中，包括 系谱分析(系谱追溯、重命名及纠错)，基因型数据格式转换(支持**Hapmap, Plink, BLUPF90, Numeric 及VCF** 格式)，基因型数据的质控填充，亲缘关系矩阵的构建(**系谱，基因组及一步法亲缘关系矩阵**)以及遗传评估（仅需几行代码即可通过DMU和BLUPF90完成遗传评估）。

最后，为了方进一步方便用户的使用(尤其是编程基础弱的用户)，我们创建了一个免费的在线网站。绝大部分的 `blupADC` 的功能均可通过在网站上实现，用户只需上传、点击及下载即可完成整套的分析流程。但是，网站的一个缺点就是，不能处理大数据，请大家合理选择!

☺ 祝好运！

blupADC-安装

安装 `blupADC` 之前，用户首先需要安装如下3个包： `Rcpp`, `RcppArmadillo` and `data.table`.

```
install.packages(c("Rcpp", "RcppArmadillo","data.table"))
```

👉 **Note: 在 DMU 和 BLUPF90的分析中，我们通常需要提前下载好 DMU 软件 (DMU下载网站) 和 BLUPF90软件 (BLUPF90下载网站)。为了方便用户使用，我们已经将两款软件中基础模块封装进了 `blupADC`，请大家合理使用。**

如果您想将DMU和BLUPF90用作商业用途，请务必联系 DMU 和 BLUPF90的作者！！！

在 Linux 上 安装 blupADC

```
packageurl <- "https://github.com/TXiang-
lab/blupADC/raw/master/blupADC_1.0.2_R_x86_64-pc-linux-gnu.tar.gz"
install.packages(packageurl,repos=NULL,method="libcurl")
```

对于中国用户来说，我们推荐使用如下代码进行安装（**安装速度更快**）：

```
packageurl <- "https://gitee.com/qsmei/blup-
adc/attach_files/798138/download/blupADC_1.0.2_R_x86_64-pc-linux-gnu.tar.gz
"
install.packages(packageurl,repos=NULL,method="libcurl")
```

在 Windows 上 安装 blupADC

```
packageurl <- "https://github.com/TXiang-
lab/blupADC/raw/master/blupADC_1.0.2.zip"
install.packages(packageurl,repos=NULL)
```

对于中国用户来说，我们推荐使用如下代码进行安装（**安装速度更快**）：

```
packageurl <- "https://gitee.com/qsmei/blup-
adc/attach_files/798137/download/blupADC_1.0.2.zip"
install.packages(packageurl,repos=NULL)
```

安装成功后，我们输入如下代码即可加载R包:

```
library(blupADC)
```

功能

- 功能 1. 基因型数据间的格式转换
- 功能 2. 基因型数据的质控与填充
- 功能 3. 品种分析及基因型数据重复性检测
- 功能 4. 系谱追溯、重命名及纠错
- 功能 5. 系谱可视化
- 功能 6. 亲缘关系矩阵的构建(A,G, H)
- 功能 7. 利用DMU软件进行遗传评估
- 功能 8. 利用BLUPF90软件进行遗传评估

blupADC-小例子

为了方便用户使用，所有的文档均支持双语模式(中英文说明书)。

`blupADC` 内置了几个数据集对象, 包括 `data_hmp` 及 `origin_pedigree`.

此外， `blupADC` 提供一些示例文件， 这些文件存储在`~/blupADC/extdata` 路径下。通过输出下面的代码，我们就能得到 这些文件的绝对路径了:

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

功能 1. 基因型数据间的格式转换 (see more details)

```
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
                  )
```

功能 2. 基因型数据的质控与填充 (see more details)

```
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="VCF"                #output data format
            )
```

功能 3. 品种分析及基因型数据重复性检测 (see more details)

```
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
                  output_data_path="/root",       #output path
                  return_result=TRUE              #return result
                  )
```

功能 4. 系谱追溯、重命名及纠错 (see more details)

```
library(blupADC)
pedigree_result=trace_pedigree(
                input_pedigree=origin_pedigree,   #provided pedigree data object
                trace_generation=3,       # trace generation
                output_pedigree_tree=TRUE  # output pedigree tree
                )
```

功能 5. 系谱可视化 (see more details)

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

功能 6. 亲缘关系矩阵的构建(A,G, H) (see more details)

```
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
```

功能 7. 利用DMU软件进行遗传评估 (see more details)

```
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
        )
```

功能 8. 利用BLUPF90软件进行遗传评估 (see more details)

```
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
        )
```

功能1

简介

🐼大家好,通过前一章节的学习，相信大家已经对 `blupADC` 有了一个初步的了解了。从本节开始，我们将对 `blupADC` 中的几个重要的函数一一进行讲解。这一节主要给大家讲述的是如何使用 `blupADC` 中 `genotype_data_format_conversion` 函数来进行多种基因型格式数据的转换。

示例

首先,我们以一个简单的小例子来看看函数的用法

```
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
                  )
```

下面，我们将对`genotype_data_format_conversion`中的参数一一进行讲解

输出

```
str(sum_data)
```

我们可以清楚的看到输出结果分为5个部分(长度为5的列表)，分别为：

- **hmp**：`Hapmap`格式的基因型数据

`第1列为SNP，第3列为染色体，第4列为物理位置，第12列开始为基因型数据`

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

`第1列为家系，第2列为个体号，第7列开始为基因型数据。`

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

第1列为染色体，第2列为SNP，第3列为遗传距离(cM)，第4列为物理位置。

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

第1列为个体号，第2列为基因型。

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

- **numeric**：`Numeric`格式的基因型数据

行名为个体，列名为SNP，0,1,2表示的是个体在某个SNP位点的基因型数据

	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

参数

基础参数

- **参数1:input_data_plink_ped**

用户提供的 Plink-ped格式的数据，`data.frame`类型。具体格式见结果部分

- **参数2:input_data_plink_map**

用户提供的 Plink-map格式的数据，`data.frame`类型。具体格式见结果部分

- **参数3:input_data_hmp**

用户提供的 Hapmap格式的数据，`data.frame`类型。具体格式见结果部分

- **参数4:input_data_blupf90**

用户提供的 BLUPF90格式数据，`data.frame`类型。具体格式见结果部分

- **参数5:input_data_numeric**

用户提供的 Numeric格式的数据，`matrix(纯数字)`类型。具体格式见结果部分

此外，为了方便用户使用，用户还可以直接通过提供本地数据的路径、名称与数据类型即可完成数据提供这一步骤,而无需将数据读入到R里面。

- **参数6:input_data_path**

用户提供的本地数据的文件路径，`character`类型。

- **参数7:input_data_name**

用户提供的本地数据的文件名称，`character`类型。

Note: 如果提供的数据类型为Plink ,那么本地文件名称不需要带后缀，eg. 本地文件名为test1.map test1.ped，我们提供文件名称为：`input_data_name="test1"`。除了Plink格式的数据外，其他数据格式必须提供完整的名称(带后缀)。

- **参数8:input_data_type**

  用户提供的本地数据的格式，`character`类型。数据格式包括：

  - `Hapmap`
  - `Plink`
  - `BLUPF90`
  - `Numeric`
  - `VCF`
- **参数9:return_result**

是否在R中返回输出的结果，`logical`类型。默认为FLASE。

此外，为了便于输出数据的保存，用户可以将数据保存到本地

- **参数10:output_data_path**

输出的基因型数据保存到本地的路径，`character`类型。

- **参数11:output_data_name**

输出的基因型数据保存到本地的文件名称，`character`类型。

- **参数12:output_data_type**

  用户提供的本地数据的格式，`character`类型。数据格式包括：

  - `Hapmap`
  - `Plink`
  - `BLUPF90`
  - `Numeric`
  - `VCF`

可选择同时输出多种数据格式。
e.g.`output_data_type=c("Hapmap","Plink","BLUPF90","Numeric")` 可同时输出以上4种数据类型。

👊进阶参数

- **参数13:cpu_cores**

函数调用的cpu个数，`numeric`类型。默认调用1个

- **参数14:miss_base**

缺失值在原基因型数据中所表示的的字符，`character`类型。默认为"NN"。

- **参数15:miss_base_num**

数字化格式转换中缺失值转换成的数字，`numeric`类型。默认为 5。

- **参数16:plink_software_path**

Plink软件的路径，`character`类型。涉及到VCF格式转换时，需要使用 **Plink**软件。当**Plink**软件的路径已加入到环境变量后，此项参数不需要再设置。

- **参数17:plink_software_name**

Plink软件的名称，`character`类型。涉及到VCF格式转换时，需要使用 **Plink**软件。软件默认名称为：plink。

- **参数18:vcftools_software_path**

vcftools软件的路径，`character`类型。涉及到VCF格式转换时，需要使用**vcftools**软件。当**vcftools**软件的路径已加入到环境变量后，此项参数不需要再设置。

- **参数19:vcftools_software_name**

vcftools软件的名称，`character`类型。涉及到VCF格式转换时，需要使用 **vcftools**软件。软件默认名称为：vcftools。

功能2

简介

🐮通常来讲，我们公司拿到的原始芯片数据大都是包含缺失值且未经过质控的。而在实际的数据分析中，我们一般都要求数据是经过质控和填充。为此，`blupADC`中提供了`genotype_data_QC_Imputation`函数，可以方便我们在R中调用**Plink**(用于质控)和**Beagle**(用于填充)软件进行基因型数据的质控和填充。

👉 **Note: 为了方便用户使用，`blupADC` 已经事先封装好了 Plink(用于质控) `version-1.9` 和 Beagle(用于填充) `version-5.2` 软件，用户无需再重新下载. 如果用户想自行指定软件的版本，可以通过更改相关的参数(在进阶参数部分里).**

老规矩，我们还是用一个小例子来看下函数的用法:

示例

```
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="VCF"                 #output data format
            )
```

通过上述代码，我们即可对本地的**Hapmap**格式的基因型数据进行质控和填充，并将其以**Plink**格式并保存到本地。

进行质控和填充时，我们必须要事先提供基因型数据，这部分的参数与`genotype_data_format_conversion`函数中的参数用法一致。具体大家可参阅之前的介绍: 基因型数据间的格式转换。

完成了基因型数据的提供后，我们可以通过以下参数进行质控填充的相关分析。

参数详解

💗基础参数

- **参数1：data_analysis_method**

  指定基因型数据的处理方法，`character`类型。可选方法包括：

  - "QC" :进行质控
  - "Imputation" :进行填充
  - "QC_Imputation" :先质控，后填充
- **参数2：qc_snp_rate**

使用Plink进行质控时，SNP call rate 的阈值，`numeric`类型，默认为0.1。

- **参数3：qc_ind_rate**

使用Plink进行质控时，IND call rate 的阈值，`numeric`类型，默认为0.1。

- **参数4：qc_maf**

使用Plink进行质控时，MAF 的阈值，`numeric`类型，默认为0.05。

- **参数5：qc_hwe**

使用Plink进行质控时，哈代温伯格平衡的阈值,`numeric`类型，默认为1e-6。

💗进阶参数

- **参数6：plink_software_path**

Plink软件的路径,`character`类型。

- **参数7：plink_software_name**

Plink软件的名称,`character`类型。

- **参数8：beagle_software_path**

Beagle软件的路径,`character`类型。

- **参数9：beagle_software_name**

Beagle软件的名称,`character`类型。

- **参数10：beagle_ref_data_path**

使用beagle进行填充时，提供的reference data的路径，`character`类型。

- **参数11：beagle_ref_data_name**

使用beagle进行填充时，提供的reference data的名称，`character`类型。

- **参数12：beagle_ped_path**

使用beagle进行填充时，提供的系谱数据的路径，`character`类型。

- **参数13：beagle_ped_name**

功能3

简介

🧒 品种成分分析一直以来都是数据分析中的一个难题。`blupADC`为用户提供了`genotype_data_check`函数，使得用户能够方便的解决这个问题。此外，用户还可以用这个函数进行基因型数据的重复性检测。

示例

品种成分分析

```
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
                  )
```

重复性检测

```
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
                  )
```

输出

```
library(blupADC)
library(ggplot2)
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
                    )
pca_outlier=check_result$breed$outlier
rownames(pca_outlier)=NULL
check_result=genotype_data_check(
                    input_data_hmp=cbind(PCA_data_hmp,PCA_data_hmp[,12:15]),
#provided hapmap data object
                    duplication_check=TRUE,         #whether check the duplication
of genotype
                    breed_check=FALSE,               # whether check the record of
breed
                    breed_record=PCA_Breed,          # provided breed record
                    return_result=TRUE               #return result
                    )
duplicated_genotype=check_result$duplicate_genotype
```

输出的结果主要包括以下两个部分，如下：

- **duplicated_genotype**

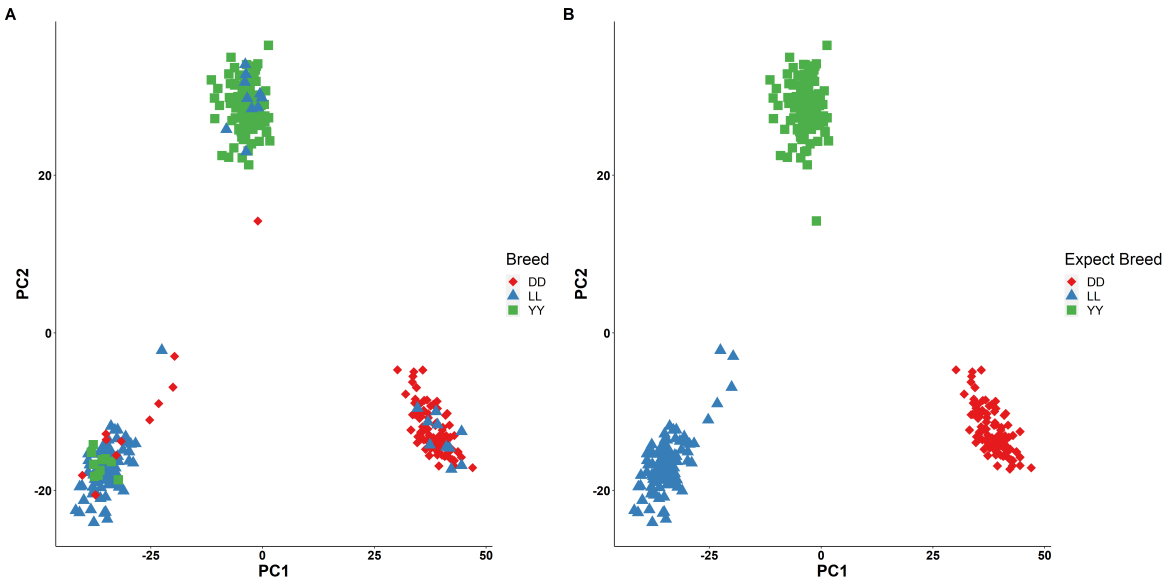
```
knitr::kable(head(duplicated_genotype))
```

第一列和第二列为个体名称，第三列为重复的比例

- **pca_outlier**

```
knitr::kable(head(pca_outlier))
```

图A是进行品种分析前的品种记录，图B是进行品种分析后(可以理解为对错误的品种记录数据进行纠正)的品种记录



参数

`genotype_data_check` 函数中的许多参数均与 `genotype_data_format_conversion` 函数中一致。故此，接下来我们将主要介绍 `genotype_data_check` 函数中独有的参数see more details).

- **1：selected_snps**

进行基因型数据重复性检测时，所选用的SNP数目， `numeric` 类型. 默认 为 1000.

- **2：overlap_threshold**

判定两个个体为重复的阈值, `numeric` 类型. 默认 为 0.95.

- **3：duplication_check**

是否进行基因型数据重复性检测, `logical` 类型. 默认为 TRUE.

- **4：breed_check**

是否进行品种分析, `logical` 类型. 默认为 FALSE.

- **5：ind_breed**

个体的品种记录数据， `data.frame` 类型.

`ind_breed` 数据格式如下所示:

```
knitr::kable(head(blupADC::PCA_Breed))
```

功能4

简介

大家好,这一节主要给大家讲述的是如何使用`blupADC`中的函数来进行系谱数据处理。`blupADC`提供的`trace_pedigree`函数，可以帮助我们非常方便的对系谱数据进行多种处理:包括系谱重命名、系谱追溯及系谱纠错等。

示例

同样的，我们还是用一个小例子来简单的看下该函数的用法

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

我们可以通过`str`查看函数的输出结果，如下所示:

```
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
```

可以很明显的看到,输出结果包括以下几个部分:

- **ped:** 经过处理后(纠错、追溯等)的原始系谱数据且未进行重命名
- **rename_ped:** 经过处理(纠错、追溯等)且重命名的系谱数据。第1列为原始系谱ID,第2列为个体在系谱中的代数，第3-5列为重命名后的系谱数据
- **pedigree_tree**:个体的系谱树矩阵。可以通过设置 pedigree_tree_depth指定系谱树包含的代数，默认不输出系谱树(节省时间)
- **error_id_set**: 系谱记录错误个体数据集。根据错误的种类可以分为以下4个子集
  - `error_duplicated_id`: 相同的个体，父母却不相同
  - `error_sex_id`: 个体同时出现在父亲列和母亲列
  - `error_breed_id`: 个体和父母的品种不相同(仅针对特殊格式)
  - `error_birth_date_id`: 个体的出生日期要早于父母的出生日期(需在系谱的第四列提供个体的出生日期)

下面，我们将具体讲解`trace_pedigree`函数中各种参数的含义:

参数详解

🕸基础参数

- **参数1：input_pedigree**

用户提供的系谱数据，`data.frame`或`matrix`类型。

📩用户提供系谱数据需为以下几种格式中的一种，包括:

- 3列系谱格式:

Offspring	Sire	Dam
DD19575312	DD18768902	DD16376015
DD19513112	DD18768902	DD17111017
DD20348012	DD19132207	DD19234510
DD20361110	DD19331001	DD19293112

Offspring	Sire	Dam
DD20471212	DD19331001	DD19320808
DD20564818	DD19331001	DD19311009

- 4列系谱格式:

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
DD20564818	DD19331001	DD19311009	20200106

- 多列系谱格式:

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:需要注意的是，当系谱为多列时，系谱的列名需要标注为特殊形式,e.g. SireSire:父亲的父亲, SirSireSire:父亲的父亲的父亲

系谱数据中缺失值可以设置为: **NA或0。**

同样的，为了便于用户操作，用户还可以直接提供本地系谱数据的路径和名称

- **参数2：input_pedigree_path**

本地系谱数据的路径,`character`类型。

- **参数3：input_pedigree_name**

本地系谱数据的名称，`character`类型。

- **参数4：pedigree_format_conversion**

是否将提供的多列系谱转换到3列，logical类型。如果用户提供的系谱数据包含多列，那么用户必须设置`pedigree_format_conversion=TRUE`。

- **参数5：output_pedigree_type**

系谱输出的格式，`character`类型。可选格式包括: BLUPF90, DMU 及 Normal(未重命名)。

- **参数6：output_pedigree_path**

系谱输出到本地的路径，`character`类型。

- **参数7：output_pedigree_name**

系谱输出到本地的名称，`character`类型。

👈进阶参数

- **参数8：dup_error_check**

检查相同个体的父母却不相同的错误，`logical`类型，默认为TRUE。

- **参数9：sex_error_check**

检查个体同时出现在父亲列和母亲列的错误，`logical`类型，默认为TRUE。

- **参数10：breed_error_check**

检查个体品种与父母品种不同的错误，`logical`类型，默认为FALSE。

- **参数11：birth_date_error_check**

检查个体出生日期早于父母的错误，`logical`类型，默认为FALSE。

- **参数12：trace_id**

追溯系谱记录的个体号，`character`类型，默认为`NULL`(追溯系谱中所有的个体)。

- **参数13：trace_offspring**

是否追溯子代，`logical`类型，默认为FALSE。

- **参数14：trace_generation**

追溯的代数，`numeric`类型，默认为5。

- **参数15：trace_birth_date**

追溯出生日期晚于指定日期的个体，`character`类型。

- **参数16：output_pedigree_tree**

是否输出系谱树，`logical`类型，默认为FALSE。

- **参数17：pedigree_tree_depth**

系谱树的深度(系谱代数)，`numeric`类型，默认为3。

功能5

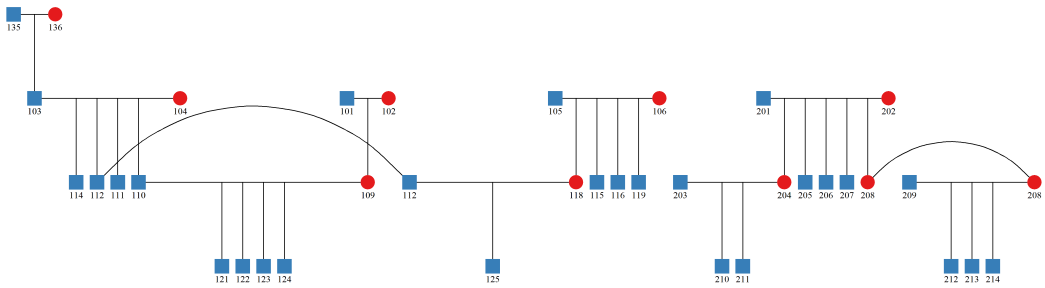
简介

 一个直观和清楚的系谱结构图能够帮助育种家和研究者做出更好的育种规划。通过使用 `ggped` 函数, 用户即能非常简单的绘制出所需的系谱图。

示例

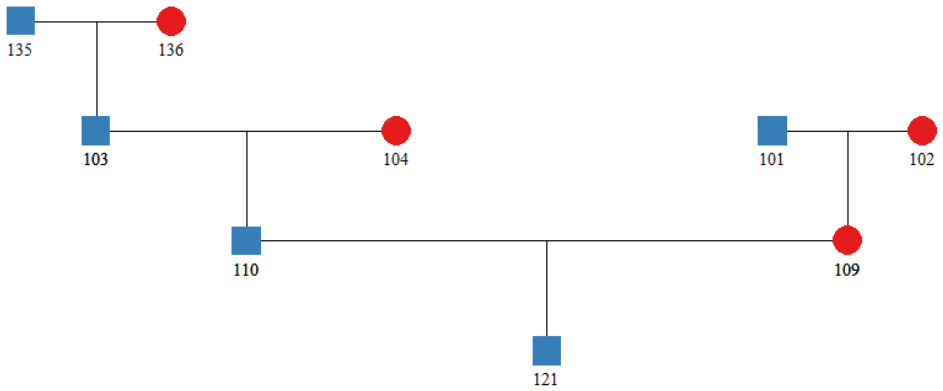
绘制系谱中所有个体

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



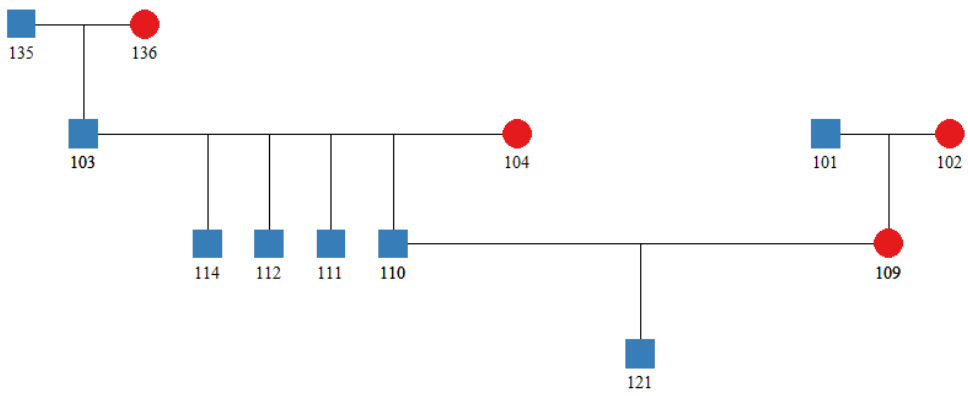
绘制系谱中的子集

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



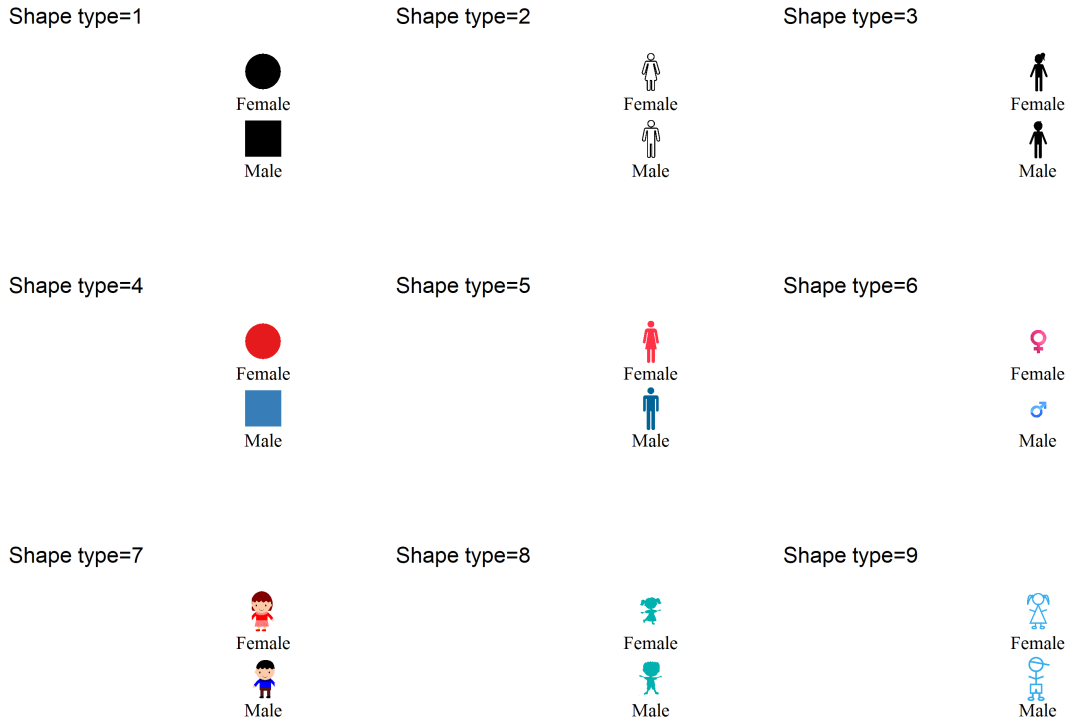
绘制系谱中的子集 (考虑子集的同胞)

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



系谱图的样式

通过修改`shape_type`参数， 用户即可改变系谱图的样式。默认的 `shape_type` 为 4。



参数详解

🕸基础参数

- **参数1：input_pedigree**

用户提供的系谱数据, `data.frame` 或者 `matrix` 类型.

📩提供的系谱数据类型应为3列如下的3列结构：

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

- **参数2：trace_id**

追溯子集的系谱，`character` 类型. 默认为 `NULL` (绘制系谱中所有个体)

- **参数3：trace_sibs**

追溯子集的系谱过程中，是否追溯子集的同胞，`logical` 类型. 默认为 `FALSE` .

- **参数4：ind_sex**

个体的性别记录，`data.frame` 类型.

数据格式如下:

Individual	Sex
101	Male
102	Female
103	Male
104	Female

- **参数5：plot_family**

绘制系谱过程中，是否划分家系结构，`logical` 类型. 默认为 `FALSE` .

- **参数6：shape_type**

系谱图的样式，`numeric` 类型. 默认为 4.

👈进阶参数

- **参数7：shape_size**

系谱图的大小，`numeric` 类型. 默认为 `8` .

- **参数8：ind_text_size**

个体名称的文本大小，`numeric` 类型. 默认为 4` .

- **参数9：ind_text_vjust**

个体名称的文本垂直距离，`numeric` 类型. 默认为 `3` .

功能6

简述

🐮在动植物育种中，亲缘关系矩阵的构建是其中的关键步骤。在本章，我们将主要介绍如何利用`blupADC`中的`cal_kinship`函数完成各种亲缘关系矩阵的构建，包括：**加性亲缘关系矩阵**(系谱，基因组，一步法)及**显性亲缘关系矩阵**(系谱，基因组，一步法)等。此外，cal_kinship函数还能方便的计算各种类型的**近交系数**，包括：系谱近交系数，基因组近交系数(Homozygous,Digonal)及一步法近交系数(Digonal)。

示例

🐮首先，我们还是通过一个小例子来看该函数的用法:

```
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
```

在构建基因组亲缘关系矩阵及一步法亲缘关系矩阵的时候，我们必须要事先提供基因型数据，这部分的参数与`genotype_data_format_conversion`函数中的参数用法一致。具体大家可参阅之前的介绍:基因型数据间的格式转换。

完成了基因型数据的提供后，我们可以通过以下几个参数来指定构建亲缘关系矩阵类型及近交系数的类型。

参数详解

💪基础参数

- **参数1：kinship_type**

  指定构建亲缘关系矩阵的类型，`character`类型。可选关系矩阵类型：

  - "G_A" :基因组加性亲缘关系矩阵"
  - "G_Ainv" :基因组加性亲缘关系逆矩阵"
  - "G_D" :基因组显性亲缘关系矩阵
  - "G_Dinv" :基因组显性亲缘关系逆矩阵"
  - "P_A" :系谱加性亲缘关系矩阵
  - "P_Ainv" :系谱加性亲缘关系逆矩阵
  - "P_D" :系谱显性亲缘关系矩阵
  - "P_Dinv" :系谱显性亲缘关系逆矩阵
  - "H_A" :一步法加性亲缘关系矩阵
  - "H_Ainv" :一步法加性亲缘关系逆矩阵
  - "H_D" :一步法显性亲缘关系矩阵
  - "H_Dinv" :一步法显性亲缘关系逆矩阵

Note:如果计算系谱及一步法亲缘关系矩阵，必须要提供系谱数据。关于如何提供系谱数据，我们会在后面进行讲解。

- **参数2：dominance_type**

  指定计算的显性亲缘关系矩阵的类型，`character`类型，可选类型包括:

  - "genotypic" : 按照$(0-2pq, 1-2pq, 0-2pq)$方式编码方式构建显性亲缘关系矩阵
  - "classical" : 按照$(-2q^2, 2pq, -2p^2)$方式编码方式构建显性亲缘关系矩阵

> 关于二者的区别，具体可阅读该文献：On the Additive and Dominant Variance and Covariance of Individuals Within the Genomic Selection Scope

- **参数3：inbred_type**

  指定计算的近交系数的类型，`character`类型。可选近交系数类型包括:

  - "Homozygous" :根据纯合子位点计算
  - "G_Diag" :G矩阵对角线-1
  - "H_diag" :H矩阵对角线-1
  - "Pedigree" :A矩阵对角线-1
- **参数4：input_pedigree**

用户提供的系谱数据，`data.frame`或`matrix`类型。具体的系谱数据格式可以参阅之前的介绍：系谱追溯、重命名及纠错 。

- **参数5：pedigree_rename**

是否对系谱数据进行重命名,`logical`类型，默认为TRUE(对系谱进行重命名)。

- **参数6：IND_geno_rename**

是否根据系谱的重命名结果对基因型数据中的个体进行重命名，`logical`类型，默认为FALSE(不进行重命名)。

- **参数7：rename_to_origin**

将亲缘关系矩阵中重命名的个体号转换回原始的个体号，`logical`类型，默认为FALSE(不进行转换)。

- **参数8：output_matrix_type**

输出亲缘关系矩阵的格式，`character`类型。可选参数包括：

1. "col_all" :按照n*n的格式输出亲缘关系矩阵
2. "col_three" : 按照3列矩阵的格式输出亲缘关系矩阵，第1列和第2列为个体号，第3列为亲缘系数。DMU和BLUPf90软件均需提供这种格式的亲缘关系矩阵。形如：

1001	1001	0.989
1001	1002	0.421
1001	1003	0.567

默认参数为 "col_three"

- **参数9：output_matrix_path**

亲缘关系矩阵保存到本地的路径,`character`类型。

- **参数10：output_matrix_name**

亲缘关系矩阵保存到本地的名称，`character`类型。

进阶参数

- **参数11：cpu_cores**

调用的cpu个数，`numeric`类型，默认为1。

- **参数12：kinship_base**

是否按照基础群的方式构建基因组亲缘关系矩阵(p=q=0.5)，`logical`类型，默认为FALSE。

- **参数13：kinship_trace**

是否按照矩阵迹和的方式对基因组亲缘关系矩阵进行标准化，`logical`类型，默认为FALSE。

- **参数14：Metafounder_algorithm**

是否按照metafounder的方法计算一步法亲缘关系矩阵，`logical`类型，默认为FALSE。

- **参数15：APY_algorithm**

是否按照APY的方法计算亲缘关系矩阵的逆矩阵，`logical`类型，默认为FALSE。

- **参数16：APY_eigen_threshold**

特征值所能解释的遗传变异的比例的阈值，`numeric`类型. 默认为 0.95.

- **参数17：APY_n_core**

核心个体数，`numeric`类型. 默认为 NULL.

- **参数18：SSBLUP_omega**

构建一步法亲缘关系矩阵时G矩阵和A矩阵的比例，`numeric`类型，默认为0.05。

- **参数19：gene_dropping**

是否使用 gene dropping 的方法构建系谱显性亲缘关系矩阵，`logical`类型，默认为FALSE。

- **参数20：gene_dropping_iteration**

gene dropping方法的迭代次数，`numeric`类型，默认为1000。

- **参数21：memory_save**

是否使用省内存的方式进行上述所有的计算，`logical`类型，默认为FALSE。该方法非常节省内存，但是会极大的占据本地存储空间。

功能7

简述

在讲述完各种各样的数据预处理方法后，我们正式进入到育种数据的分析层面。在目前的动植物育种领域，主要的育种软件包括但不限于以下两种：**DMU**和**BLUPf90**。这两款软件均是于上世纪80-90年代开发的，并且一直处于维护中。截至目前，两款软件的引用次数均已接近千次(实际可能更多), 这也足见这两款软件的认可度。

但是，这两款软件均存在一个共同的缺点，就是使用起来较为麻烦(需要提供准备好的参数文件)。笔者当时学习参数文件的配置时，前前后后花费了近一个月的时间，足以见这两款软件的不友好性。

为此，我们在R中编写了相应的函数，使得用户可以轻松的完成两款软件参数文件的配置及对应的数据分析。本章我们主要讲述如何通过`BLUP_ADC`中的`run_DMU`函数，在R中调用**DMU**软件并完成数据分析。在下一章，我们将会讲述如何在R中调用**BLUPf90**软件，并完成数相应的据分析。

Note: 为了方便用户使用，`blupADC`已经封装了DMU中的几个基本模块(`dmu1`,`dmuai`, and `dmu5`), 更多的模块可以从DMU官网进行下载(DMU下载网站).

如果您想将DMU用作商业用途，请务必联系 DMU的作者！！！

示例

单性状模型-系谱

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  示例文件的路径

run_DMU(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1"),                              #性状名称
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),      #固定效应名称
        random_effect_name=list(c("Id","Litter")),                #随机效应名称
        covariate_effect_name=NULL,                               #协变量效应名称
        phe_path=data_path,                             #表型文件路径
        phe_name="phenotype.txt",                       #表型文件名
        integer_n=5,                                    #整型变量数
        analysis_model="PBLUP_A",                       #遗传评估模型
        dmu_module="dmuai",                             #方差组分估计使用的DMU模块
        relationship_path=data_path,                    #亲缘关系文件路径
        relationship_name="pedigree.txt",               #亲缘关系文件名
        output_result_path="/root"                      #结果输出路径
        )
```

单性状模型-GBLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  示例文件的路径

run_DMU(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1"),                              #性状名称
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),      #固定效应名称
        random_effect_name=list(c("Id","Litter")),                #随机效应名称
        covariate_effect_name=NULL,                               #协变量效应名称
        phe_path=data_path,                             #表型文件路径
        phe_name="phenotype.txt",                       #表型文件名
        integer_n=5,                                    #整型变量数
        analysis_model="GBLUP_A",                       #遗传评估模型
        dmu_module="dmuai",                             #方差组分估计使用的DMU模块
        relationship_path=data_path,                    #亲缘关系文件路径
        relationship_name="G_Ainv_col_three.txt",       #亲缘关系文件名
        output_result_path="/root"                      #结果输出路径
        )
```

单性状模型-Single-step(一步法)

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  示例文件的路径
```

```
run_DMU(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # colnames of phenotype
        target_trait_name=c("Trait1"),                              #性状名称
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),        #固定效应名称
        random_effect_name=list(c("Id","Litter")),                  #随机效应名称
        covariate_effect_name=NULL,                                 #协变量效应名称
        phe_path=data_path,                             #表型文件路径
        phe_name="phenotype.txt",                       #表型文件名
        integer_n=5,                                    #整型变量数
        analysis_model="SSBLUP_A",                      #遗传评估模型
        dmu_module="dmuai",                             #方差组分估计使用的DMU模块
        relationship_path=data_path,                    #亲缘关系文件路径
        relationship_name=c("pedigree.txt","G_A_col_three.txt"),     #亲缘关系文件
名
        output_result_path="/root"                      #结果输出路径
        )
```

细心的同学应该能发现，我们仅需改变 `analysis_model` 及 `relationship_name` 这两个参数即可完成 系谱、GBLUP及SSBLUP的分析，极大的简化了我们的分析步骤(PS: G_Ainv_col_three.txt 和 G_A_col_three.txt 文件 均可通过 `cal_kinship` 函数得到 了解更多)。

上面我们介绍的都是单性状模型(**只包括了一个目标性状**)，而在实际育种分析中，多性状模型也是非常常见。在上面代码的基础上稍作修改，我们就能够非常方便的完成多性状模型的运算，如下：

多性状模型-系谱

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")  #  示例文件的路径

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"),                          #性状名
称

fixed_effect_name=list(c("Sex","Herd_Year_Season"),c("Herd_Year_Season")),     #
固定效应名称
        random_effect_name=list(c("Id","Litter"),c("Id")),       #随机效应名称
        covariate_effect_name=NULL,                              #协变量效应名称
        phe_path=data_path,                             #表型文件路径
        phe_name="phenotype.txt",                       #表型文件名
        integer_n=5,                                    #整型变量数
        analysis_model="PBLUP_A",                       #遗传评估模型
        dmu_module="dmuai",                             #方差组分估计使用的DMU模块
        relationship_path=data_path,                    #亲缘关系文件路径
        relationship_name="pedigree.txt",               #亲缘关系文件名
        output_result_path="/root"                      #结果输出路径
        )
```

接下来，我们将对 `run_DMU` 函数中的参数一一进行讲解。

参数详解

基础参数

- **参数1：phe_path**

本地表型数据文件的路径，`character`类型。

- **参数2：phe_name**

本地表型数据文件的名称，`character`类型。

- **参数3：phe_col_names**

表型数据的列名，`character`类型。

- **参数4：integer_n**

整型变量的数目，`numeric`类型。

- **参数5：genetic_effect_name**

遗传效应的名称(一般为个体号)，`character`类型。

- **参数6：target_trait_name**

目标性状的名称，`character`类型。

通过添加多个性状的名称，我们即可完成多性状模型的分析，e.g.
`target_trait_name=c("Trait1","Trait2")`即可完成`Trait1`和`Trait2`的双性状模型

- **参数7：fixed_effect_name**

目标性状的固定效应名称，`list`类型。在多性状模型中，`fixed_effect_name`为每个性状的固定效应名称向量组成的列表结构，性状的顺序需与`target_trait_name`一一对应。

e.g. 第一个性状的固定效应名称为：`c("Sex","Herd_Year_Season")`

第二个性状的固定效应名称为：`c("Sex")`

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

- **参数8：random_effect_name**

目标性状的随机效应名称，`list`类型。在多性状模型中，`random_effect_name`为每个性状的随机效应名称向量组成的列表结构，性状的顺序需与`target_trait_name`一一对应。

e.g. 第一个性状的随机效应名称为：`c("Id","Litter")`

第二个性状的随机效应名称为：`c("Id")`

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

- **参数9：covariate_effect_name**

目标性状的协变量效应名称，`list`类型。在多性状模型中，`random_effect_name`为每个性状的协变量效应名称向量组成的列表结构，性状的顺序需与`target_trait_name`一一对应。

e.g. 第一个性状的协变量效应名称为：`c("Age")`

第二个性状的协变量效应名称为：`NULL`(**意味着无协变量**)

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

- **参数10：analysis_model**

  遗传评估的分析模型，`character`类型。可选模型包括：

    - `"PBLUP_A"`: 系谱-加性效应模型
    - `"GBLUP_A"`:基因组-加性效应模型
    - `"GBLUP_AD"`:基因组-加性-显性效应模型
    - `"SSBLUP_A"`:一步法加性效应模型
    - `"User_define"`: 用户自定义模型
- **参数11：dmu_module**

  DMU分析时使用的模块，`character`类型。可选模块包括：

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

DMU软件在本地的路径，`character`类型。

- **参数13：relationship_path**

提供的亲缘关系文件的路径，`character`类型。

- **参数14：relationship_name**

提供的亲缘关系文件的名称，`character`类型。

针对不同的遗传评估分析模型，我们需要提供不同类型的亲缘关系文件。

针对 "PBLUP_A"模型，我们需要提供系谱文件,e.g. `relationship_name="pedigree.txt"`；

针对`"GBLUP_A"` 或 `"GBLUP_AD"`模型，我们需要提供3列格式的基因组亲缘关系矩阵的逆矩阵文件, e.g. `relationship_name=c("G_A_inv_matrix.txt","G_D_inv_matrix.txt")`;

针对 `"SSBLUP_A"`模型, 我们需要同时提供系谱文件及3列格式的基因组亲缘关系矩阵的文件, e.g. `relationship_name=c("pedigree.txt","G_A_matrix.txt")`。

- **参数15：output_result_path**

DMU运行结果的保存路径，`character`类型。

- **参数16：output_ebv_path**

输出的育种值、残差及校正表型文件的保存路径，`character`类型。

- **参数17：output_ebv_name**

输出的育种值、残差及校正表型文件的名称，`character`类型。

进阶参数

- **provided_effect_file_path**

性状效应记录文件的路径，`character`类型。为了方便用户输入固定效应、随机效应及协变量效应，用户可以在本地直接提供相应的文件，格式如下所示：

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

每类效应都用 *隔开，每一列的间隔均为制表符间隔。每个性状所在的行均有4个，第1-2个* 之间的效应代表的是固定效应，第2-3个* 之间的效应代表的是随机效应，第3-4个* 之间的效应代表的是协变量效应。*

- **参数19：provided_effect_file_name**

性状效应记录文件的名称，`character`类型。

- **参数20：provided_DIR_file_path**

用户自己提供的DIR文件的路径，`character`类型。

- **参数21：provided_DIR_file_name**

用户自己提供的DIR文件的名称，`character`类型。

- **参数22：included_permanent_effect**

是否进行永久环境效应分析，`logical`类型，默认为FASLE。

- **参数23：dmu_algorithm_code**

DMU模块内的算法代码，`numeric`类型。

- **参数24：provided_prior_file_path**

用户提供的方差组分-PRIOR文件的路径，`character`类型。

- **参数25：provided_prior_file_name**

用户提供的方差组分-PRIOR文件的名称，`character`类型。

- **参数26：missing_value**

表型数据的缺失值，`numeric`类型，默认为 -9999。

- **参数27：iteration_criteria**

方差组分迭代收敛的标准，`numeric`类型，默认为 1.0e-7。

- **参数28：genetic_effect_number**

SOL文件中，遗传效应所代表的数字，`numeric`类型，默认为4。

- **参数29：residual_cov_trait**

残差协方差约束为0的性状，`list`类型。e.g. 将Trait1和Trait2的残差协方差约束为0，`residual_cov_trait=list(c("Trait1","Trait2"))`

- **参数30：selected_id**

只输出这部分个体的育种值、残差及校正表型，`character`类型。

- **参数31：cal_debv**

是否计算DEBV，`logical`类型，默认为FALSE。

- **参数32：debv_pedigree_path**

计算DEBV时，提供的系谱文件的路径，`character`类型。

- **参数33：debv_pedigree_name**

计算DEBV时，提供的系谱文件的名称，`character`类型。

功能8

简述

在前面的章节，我们已经讲述了如何在**R**中调用**DMU**软件并完成相应的分析。本章，我们将讲述如何通过`BLUP_ADC`中的`run_BLUPF90`函数，在**R**中调用**BLUPF90**软件并完成数据分析。为了方便用户的使用，我们已经尽可能地将`run_BLUPF90`函数中的参数 和 `run_DMU`函数中的参数进行了统一。

Note: 为了方便用户使用，`blupADC`已经封装了BLUPF90中的几个基本模块(`renumf90`，`remlf90`，`airemlf90`，和`blupf90`), 更多的模块可以从BLUPF90官网进行下载(BLUPF90下载网站).

如果您想将BLUPF90用作商业用途，请务必联系 BLUPF90的作者！！！

接下来，我们还是用几个简单的例子看看该函数的用法:

示例

单性状模型-系谱

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")    #  示例文件的路径

run_BLUPF90(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # 表型数据的列名(ps.表型文件无列名)
        target_trait_name=c("Trait1"),                              #性状名称
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),        #固定效应名称
        random_effect_name=list(c("Id","Litter")),                  #随机效应名称
        covariate_effect_name=NULL,                                 #协变量效应名称
        phe_path=data_path,                             #表型文件路径
        phe_name="phenotype.txt",                       #表型文件名
        analysis_model="PBLUP_A",                       #遗传评估模型
        relationship_path=data_path,                    #亲缘关系文件路径
        relationship_name="pedigree.txt",               #亲缘关系文件名
        output_result_path="/root"                      #结果输出路径
        )
```

单性状模型-GBLUP

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")    #  示例文件的路径

run_BLUPF90(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # 表型数据的列名(ps.表型文件无列名)
        target_trait_name=c("Trait1"),                              #性状名称
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),        #固定效应名称
        random_effect_name=list(c("Id","Litter")),                  #随机效应名称
        covariate_effect_name=NULL,                                 #协变量效应名称
        phe_path=data_path,                             #表型文件路径
        phe_name="phenotype.txt",                       #表型文件名
        analysis_model="GBLUP_A",                       #遗传评估模型
        relationship_path=data_path,                    #亲缘关系文件路径
        relationship_name="blupf90_genumeric",          #亲缘关系文件名
```

```
        output_result_path="/root"                          #结果输出路径
        )
```

单性状模型-Single-step(一步法)

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")   #  示例文件的路径

run_BLUPF90(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # 表型数据的列名(ps.表型文件无列名)
        target_trait_name=c("Trait1"),                              #性状名称
        fixed_effect_name=list(c("Sex","Herd_Year_Season")),        #固定效应名称
        random_effect_name=list(c("Id","Litter")),                  #随机效应名称
        covariate_effect_name=NULL,                                 #协变量效应名称
        phe_path=data_path,                             #表型文件路径
        phe_name="phenotype.txt",                       #表型文件名
        analysis_model="SSBLUP_A",                      #遗传评估模型
        relationship_path=data_path,                    #亲缘关系文件路径
        relationship_name=c("pedigree.txt","blupf90_genumeric"),        #亲缘关系文
件名
        output_result_path="/root"                      #结果输出路径
        )
```

同样的，与DMU使用类似，我们仅需改变 `analysis_model` 及 `relationship_name` 这两个参数即可完成 系谱、GBLUP及SSBLUP的分析(PS: blupf90_genumeric 文件 均可通过 `genotype_data_format_conversion` 函数得到 了解更多)。

多性状模型-系谱

上面我们介绍的都是单性状模型(**只包括了一个目标性状**)。与上一节的介绍的 `run_DMU` 函数类似，如果我们想完成双性状模型的计算，只需要在上面的脚本的基础上稍作修改就能实现目的，具体代码如下：

```
library(blupADC)
data_path=system.file("extdata", package = "blupADC")   #  示例文件的路径

run_BLUPF90(

phe_col_names=c("Id","Mean","Sex","Herd_Year_Season","Litter","Trait1","Trait2",
"Age"), # 表型数据的列名(ps.表型文件无列名)
        target_trait_name=c("Trait1","Trait2"),                     #性状名称

fixed_effect_name=list(c("Sex","Herd_Year_Season"),c("Herd_Year_Season")),     #
固定效应名称
        random_effect_name=list(c("Id","Litter"),c("Id")),                  #随机效
应名称
        covariate_effect_name=list(NULL,"Age"),                                 #协
变量效应名称
        phe_path=data_path,                             #表型文件路径
        phe_name="phenotype.txt",                       #表型文件名
        analysis_model="PBLUP_A",                       #遗传评估模型
        relationship_path=data_path,                    #亲缘关系文件路径
        relationship_name="pedigree.txt",               #亲缘关系文件名
        output_result_path="/root"                      #结果输出路径
        )
```

参数详解

接下来，我们将对`run_BLUPF90`中特有的参数进行讲解，剩余的参数大家可移步DMU软件的交互使用进行查看，相同参数的用法和含义均是一模一样的。

特有参数

- **参数1：blupf90_algorithm**

  BLPUF90进行方差组分估计时选用的算法，`character`类型。可选算法包括：

  - `"AI_REML"`
  - `"EM_REML"`
  - `"BLUP"`：无需估计方差组分，根据提供的先验直接求解混合线性模型方程组。

默认算法为："AI_REML"

- **参数2：provided_blupf90_prior_file_path**

用户提供的BLUPF90格式的方差组分-PRIOR文件的路径，`character`类型。

Note:需要注意的一点是，**BLUPF90**格式的PRIOR文件和**DMU**格式的PRIOR文件是不相同的。关于二者的差异，以后有时间会再出一篇文章进行讲解，这里就不再赘述了。

- **参数3：provided_blupf90_prior_file_name**

用户提供的BLUPF90格式的方差组分-PRIOR文件的名称，`character`类型。

- **参数4：provided_blupf90_prior_effect_name**

用户提供的PRIOR文件中，与方差组分对应的各个随机效应名称，`character`类型。

- **参数5：BLUPf90_software_path**

BLUPF90软件在本地的路径，`character`类型。