# *User's Manual for ASRtriala v. 1.0.0*

## *An R package with complementary functions for single and multi-environment trial analyses*


**Prepared by**

**Salvador A. Gezan, Giovanni Galli and Darren Murray**

March 10, 2022

**User's Manual for ASRtriala v. 1.0.0**

### Bibliographical reference

Gezan, S.A., Galli, G., and Murray, D. 2022. ASRtriala: An R package with complementary functions for single and multi-environment trial analyses. Version 1.0.0 VSN International Ltd., Hemel Hempstead, HP1 1ES, UK.

### Authors' email addresses

Salvador A. Gezan salvador.gezan@vsni.co.uk
Giovanni Galli giovanni.galli@vsni.co.uk
Darren Murray darren.murray@vsni.co.uk

### Companion resources

This R library and associated documentation can be downloaded from:
https://vsni.co.uk/free-software/asrtriala

### Acknowledgements

The authors want to thank the team of testers from our group of close collaborators. In addition Amanda A. de Oliveira who contributed to the initial development of many of the functions. Also, we want to thank Johan Aparicio (Alliance Bioversity-CIAT) and Khaled Al-Sham'aa (ICARDA) for fruitful discussions on similar implementations of MET analyses.

## Preface

`ASRtriala` is a companion package for `ASReml-R` to help audit and prepare the data, and select a *best* model for the analysis of field trials. A set of spatial and non-spatial models can be pre-defined to be fitted, and then the *best* single-trial model selected can be used as final analysis, or as the first part of a two-stage multi-environment trial (MET) analysis. For the second step, `ASRtriala` has functions available to fit a range of MET model structures.

The main tasks included are:

- Auditing and preparing single-trial data and MET data.
- Fitting and selecting a single-trial model using `ASReml-R`.
- Fitting and selecting a MET model using `ASReml-R`.
- Enhancing output from MET models.

The main goal of this package is to assist with semi-automatic pipelines to perform spatial and/or non-spatial analyses of field trials and to use this output in the fitting of MET models with complex variance-covariance structures to model GxE as nested effects. The latter, focuses on the use of the two-stage analyses as proposed by Smith et al. (2001a).

To achieve the best use of the data and to increase the success on the implementation of this workflow, we have added functions that will help audit and prepare the data. This includes, for example, calculation of relevant summary statistics together with an assessment of the MET data in aspects such as connectivity and variability.

There are two functions that will fit user-specified single-trial and MET models using `ASReml-R` (Butler et al. 2017) in the background. These functions can be used to fit multiple spatial and non-spatial models for single-trial analyses or multiple GxE structures for MET analyses. They allow for the selection of the *best* model according to a choice of goodness-of-fit statistics. These routines also allow for detection of outliers, and provide several goodness-of-fit statistics that help with determining the quality of the fitted models.

Finally, we have included some routines to assist with the post-analysis of MET models. These include additional output for factor analytic (FA) structures with the generation of genotype-specific plots to assess GxE. In addition, for ranking and selection of genotypes across environments there are routines to calculate stability coefficients and to display biplots.

The routines considered here incorporate the latest developments and implementation of MET analyses based on extensive and proven statistical research and practical experience. Our intent is to facilitate the implementation and use of these technologies in a straightforward and efficient manner for plant breeding programs of any size. We aim to simplify the configuration of linear mixed models (LMMs) within `ASReml-R` to be more intuitive and to provide semi-automatic routines with a sound analytical workflow.

In this manual, we will present some of the structure of `ASRtriala` illustrating its use with examples using datasets provided in this package. `ASRtriala` is a tool that is provided 'as is,' but we have made all efforts to check our routines carefully and we have used real, publicly available datasets.

# Contents

# 1 Getting Started

ASRtriala is an R package available for Linux, Windows and Mac OS that can be obtained for free from https://vsni.co.uk/free-software/asrtriala

First, download the appropriate version of ASRtriala for your operating system.

- For Windows, the download will be a .zip file.
- For Linux, the download will be a .tgz file.
- For Mac, the download will be a .tar.gz file.

Then, before installing ASRtriala you will need to install the following packages:

- plyr
- ggplot2
- MASS
- mice

These are available for installation from the CRAN website https://cran.r-project.org/.

To install ASRtriala you can use one of the following commands as appropriate for your operating system.

For Windows:

```
install.packages(path, repos = NULL, type = "win.binary")
```

For Linux:

```
install.packages(path, repos = NULL)
```

For Mac:

```
install.packages(path, repos = NULL, type = "source")
```

where path is the location and name of the appropriate file for your operating system to install, for example: "/home/<username>/ASRtriala1.0.0.zip".

Another option is to install ASRtriala directly from RStudio by first going to the menu: >Tools/Install Packages..., in the Install From field select

"Package Archive File (.zip; .tgz; tar.gz)"

and then you will have to search for the location of the file on your computer and select it.

Finally, you just need to click on Install, and once installed, load the library using the command:

```
library(ASRtriala)
```

Now you are ready to use it! A good way to get started is to request help directly from this library. For example, you can type:

```
help(ASRtriala)
```

This will show a complete description of the functions and the datasets used in this package by directly accessing the help pages available in R.

Once ASRtriala is loaded, you can also access the datasets by using the data() function, for example:

```
pheno.wheat <- ASRtriala::pheno.wheat
```

In the next section we will describe in detail the analytical pipeline or workflow to perform two-stage MET analyses. This will combine the use of the majority of the functions provided within ASRtriala. Later in the other sections, we will show how to prepare and run these analyses for a case based on real breeding data.

# 2 Analytical Flow with ASRtriala

Multi-environmental trial (MET) analyses are probably one of the most important aspects to evaluate in plant breeding. Here, groups of genotypes are evaluated in several environments (sites, years, and/or treatments) and their performance is assessed across these environments. Any lack of stability found in this performance is known as genotype-by-environment (GxE) interaction and it is due to differential responses of a genotype to the particular conditions found in a given environment.

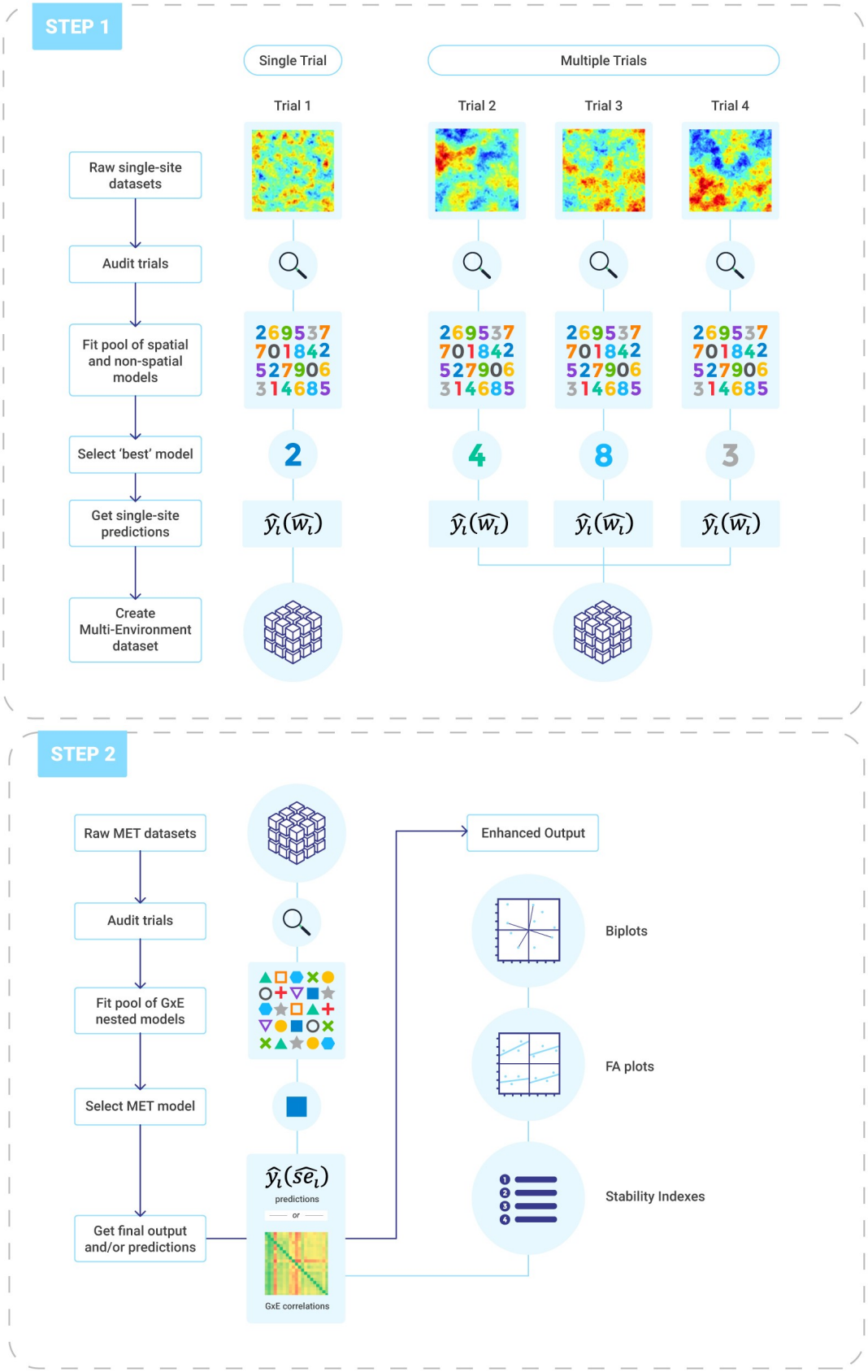
Analyses of MET studies can be complex as there are computational and modelling challenges. It is possible to perform a **single-stage analysis** combining all available sites/trials into a unique linear mixed model (LMM); however, this is often difficult as each individual trial might differ in its design structure, genotypes considered, and they might also have specific site conditions that require carefull modelling. In addition, when fitting a complex LMM, the large number of variance components to estimate simultaneously makes convergency difficult or sometimes impossible.

Alternatively, it is possible to perform a **two-stage analysis**. Here, the process is divided into two steps. During step 1, the focus is on each individual trial where all efforts are aimed at fitting the *best* possible model considering spatial and/or non-spatial adjustments as part of the modelling process. After the *best* model is fitted, predictions (or adjusted means) are obtained together with their statistical weights. For step 2, predictions from individual trial analyses are all combined to fit a MET model that considers complex variance-covariance structures to model GxE.

The separation of the analyses into these two steps facilitates the process by focusing on each of the parts individually. There is some loss of information between the single- and two-stage process, but this is minimized by the appropriate use of weights and by enabling complex models to be fitted, rather than a simplified model which is often required to make fitting a single-stage model possible.

One very important advantage of the implementation of a two-step analysis for breeding programs is the possibility of implementing a more operational analytical pipeline. Here, each individual trial is analyzed as their raw data is available, and complex statistical tools (namely spatial analyses) are considered the norm. Predictions (or adjusted means) and weights obtained during this process can be stored and managed in a database system for later use. For example, it might be required to use a portion of the trials, or even a portion of the genotypes, in a MET analysis for a given region. This task will be easier and quicker to do in this simplified set. Additionally, any of these predictions can be used, for example, to develop Genomic Prediction (GP) models or discovery of SNPs via Genome-Wide Association Studies (GWAS). This has the advantage of not requiring to re-visit (or re-fit) each trial again, only to retrieve their information from the database system.

The general workflow of the above two-step process is presented in the following figures. And in the next section, we will start by illustrating the fitting of single-trial models.


STEP 1
Single Trial
Multiple Trials
Trial 1
Trial 2
Trial 3
Trial 4
Raw single-site datasets
Audit trials
Fit pool of spatial and non-spatial models
Select 'best' model
Get single-site predictions
Create Multi-Environment dataset
269537
701842
527906
314685
2
4
8
3
$\hat{y}_i(\widehat{w}_i)$
STEP 2
Raw MET datasets
Audit trials
Fit pool of GxE nested models
Select MET model
Get final output and/or predictions
$\hat{y}_i(\widehat{se}_i)$
predictions
or
GxE correlations
Enhanced Output
Biplots
FA plots
Stability Indexes

# 3 Single-site Analyses

Statistical evaluation of a single-trial is relatively straightforward especially when good experimental design principles have been implemented requiring only the incorporation of all relevant design terms (*e.g,.* incomplete blocks) into the fitted model. However, with the availability of spatial analyses, it is possible to improve the statistical accuracy of the treatment (or genotype) mean estimates by explicitly modelling the spatial heterogeneity of a trial. Several approaches exists, including the definition of design model terms (such as random row and columns effects), modelling of trends (*e.g.*, polynomials or cubic smoothing splines), or by specifying autocorrelation among residuals, as with the use of the first order autorregressive (AR1) error structure. There are several manuscripts that describe some of these tools, a good introductory reference with several cases is Gezan et al. (2010).

`ASRtriala` incorporates a workflow to fit a single-trial analysis that can select the *best* possible model, therefore achieving good accuracies on the genotype mean predictions. We use the term *best* in italics as it is not possible to fit all possible models, and here we only fit a predefined set of models for which we select what we consider the best according to a goodness-of-fit statistic of choice.

The workflow for this section, that assumes we have a dataset from a unique trial, is described in the following tasks:

- Preparing and auditing data.
- Selecting the *best* model.
- Fitting final selected *best* model.
- Extracting and interpreting relevant output.

## 3.1 Preparing and Auditing data

To illustrate the flow of functions and show some of the capabilities of `ASRtriala` we are going to use a dataset from wheat published by Belamkar et al. (2018). This corresponds to raw data on eight locations, where the response variable is yield (recorded in bushels/acre). Some of the trials have two replicates, but most have a single replication, but all have incomplete blocks. In relation to the genotypes there is a total of three checks and 270 test lines. Most of the lines are present in all locations.

Below we call this dataset, and then we filter it to only have a single location (in this case `Alliance`) for which we show the first six rows of data.

```
pheno.wheat <- ASRtriala::pheno.wheat
st.data <- pheno.wheat[pheno.wheat$location == "Alliance",
    ]
head(st.data)
```

```
##   location rep ibk check     gen col row yield
## 1 Alliance   1   1     1 Camelot   1  16  71.4
## 2 Alliance   1   1     0 NE16471   1  17  56.5
## 3 Alliance   1   1     0 NE16510   1  18  50.8
## 4 Alliance   1   1     0 NE16516   1  19  49.8
## 5 Alliance   1   1     0 NE16570   1  20  52.2
## 6 Alliance   1   1     0 NE16650   1  21  42.0
```

For `ASRtriala`, the names of columns are totally flexible; however, there are a couple of important considerations to follow:

- The column `check` should have only two levels (or values): 0 for test lines, and 1 for checks/controls. This column is only relevant if later it is of interest to separate the estimation of the effects of the checks (as fixed effects) from the test lines (as random effects). If both sets are assumed fixed or random, this column is not required.
- The columns `col` and `row` correspond to the coordinates of the experimental units (*e.g.*, plots). These positions should identify a unique unit, and are required for spatial analyses.
- For spatial analyses `ASReml-R` requires that the trial described by the positions is continuous, therefore data is assumed to come in a full and complete grid. If this is not the case then the function `fill.grid()` included within `ASRtriala` should be used. Further details are found in the help associated with this function.

The data frame `st.data` does comply with the previous considerations, and therefore we can proceed with the following step of **auditing** the trial. Here, we use the function `audit.single()` that will evaluate and verify the integrity and quality of the data for single field trials using simple exploratory data analyses tools; this is shown below:

```
audit <- audit.single(data = st.data, gen = "gen", check = "check",
    row = "row", col = "col", ibk = "ibk", rep = "rep",
    resp = "yield", type.label = "none")
```

Here we have specified all relevant columns from the data frame of interest, and now we can explore some of the output.

```
audit$trial.stats
```

```
##     n  min   mean  max    sd missing    CVp
## 1 597 21.1 58.955 94.5 11.08       3 18.794
```

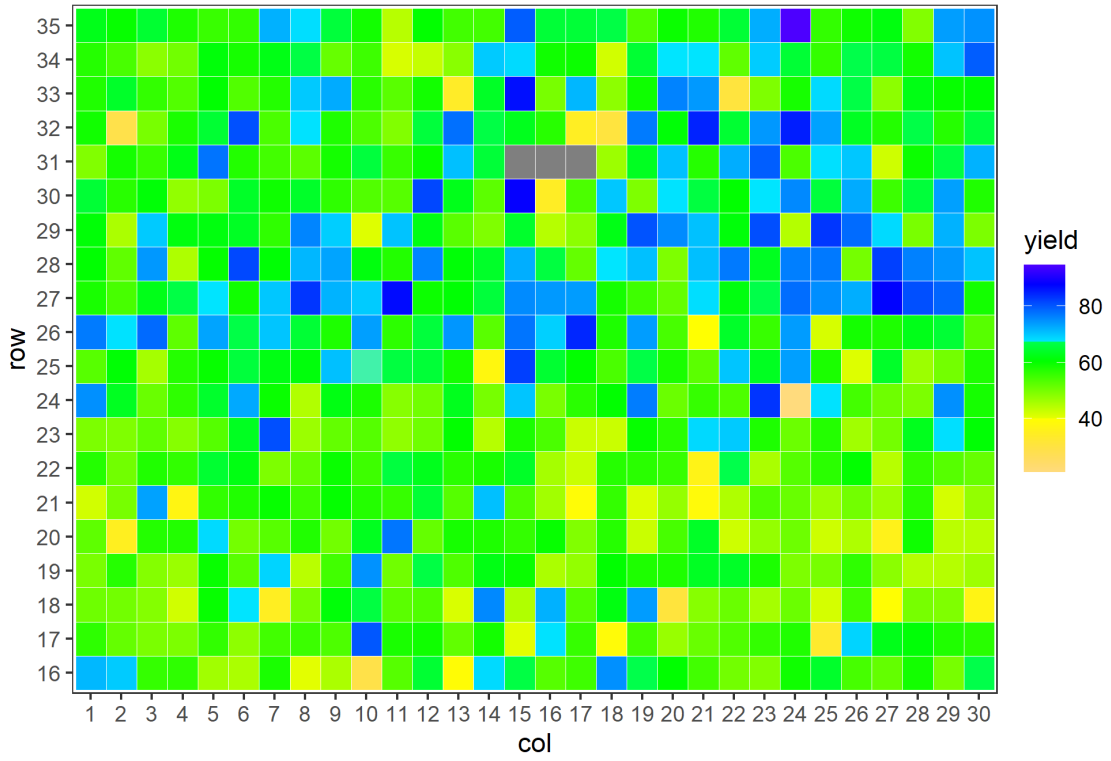
```
audit$rep.stats
```

```
##   rep   n  min   mean  max      sd missing    CVp
## 1   1 300 21.1 55.128 83.1  9.7266       0 17.644
## 2   2 297 28.8 62.822 94.5 11.0357       3 17.567
```

```
audit$check.inc
```

```
##       rep
## check   1   2
##     0 270 267
##     1  30  30
```

```
audit$trial.plot
```




This function generates several tables that are relevant, but of special interest are the tables `$trial.stats` and `$rep.stats` that shows some summary statistics for the complete trial and by replicate, respectively. This is useful, for example, to identify replicates with an unusual level of variability.

Also important is the matrix with the incidence of checks and test treatments by replication `check.inc` which is obtained only considering non-NA observations. This matrix helps to assess the potential unbalanced representation of genotypes across replicates found on the trial. We can see that only three genotypes are missing on replicate 2, a fact that is not of much concern.

The statistics of the current trial have reasonable variability, and we can, as a final step, display the field layout. Here, missing values on the response variable are represented in grey. This plot can also help to visually identify some spatial trends or unexpected patterns. In this case, it seems that the top half of the trial has better yield than the bottom half.

## 3.2 Fitting a user-defined single-trial model

Before we proceed to find the *best* single-trial model from an array of pre-defined models we will fit one simple model for the current data from the location `Alliance`. Recall that this site has two replicates and several incomplete blocks within replicate, and, for this particular example, we are interested in separating the checks (or controls) as fixed effects from the test lines that will be treated as random effects. In order to do this we will use the function `fit.single()`.

```
mT1 <- fit.single(data = st.data, gen = "gen", check = "check",
    rep = "rep", ibk = "ibk", row = "row", col = "col",
    resp = "yield", type.gen = "random", type.rep = "fixed",
    add.rep = TRUE, add.ibk = TRUE, type.residual = "ar1.rowcol",
    add.nugget = FALSE, threshold = 3.5)
```

There are several things happening in this function. It will internally call `ASReml-R` and it will extract and generate all relevant output. In addition, it will provide goodness-of-fit statistics together with suggested outliers, both of which can be used to assess the quality of the fitted model.

The first argument specifies the dataset, and the arguments following this identify the different elements of the model. The column names for the genotypes and response must be supplied. Other optional column names that can be supplied are for checks, replicates and incomplete blocks. If spatial coordinates are available, these are provided as the numerical values in row and columns. In the above code, we have specified genotype as random with `type.gen = "random"` and replicate as fixed with `type.rep = "fixed"`. Note that checks, if included, will be always assumed to be fixed for this function. The arguments `add.rep = TRUE` and `add.ibk = TRUE` are critical to indicate that these terms, if defined, should be included in the model, otherwise they are ignored.

The function `fit.single()` allows for fitting of different error structures, these are: `indep`, `ar1.row`, `ar1.col`, and `ar1.rowcol`, corresponding to independent residuals, and first order autocorrelation in row, columns and both, respectively. In this example we have selected the most complex option `ar1.rowcol`, which was specified without a nugget (or microsite error) random effect. Finally, we have the argument `threshold` that will flag for *suggested* outliers based on standardized residuals; in this case anything larger than 3.5 will be identified and presented in an output data frame. These residuals are approximations estimated internally by `ASReml-R` given that we are dealing with LMMs and should be used with care.

Let's look at some of the output from the fitting of the above model. The first thing is to check if the model has fitted successfully. For this, we check the output in `mT1$warnings` that in this case is `NA`; hence, we can move to explore the model call using:

```
mT1$call
```

```
## asreml::asreml(fixed = yield ~ 1 + rep + at(check, "1"):gen,
##     random = ~rep:ibk + at(check, "0"):gen, residual = ~ar1(row):ar1v(col),
##     data = data, na.action = list(x = "include", y = "include"))
```

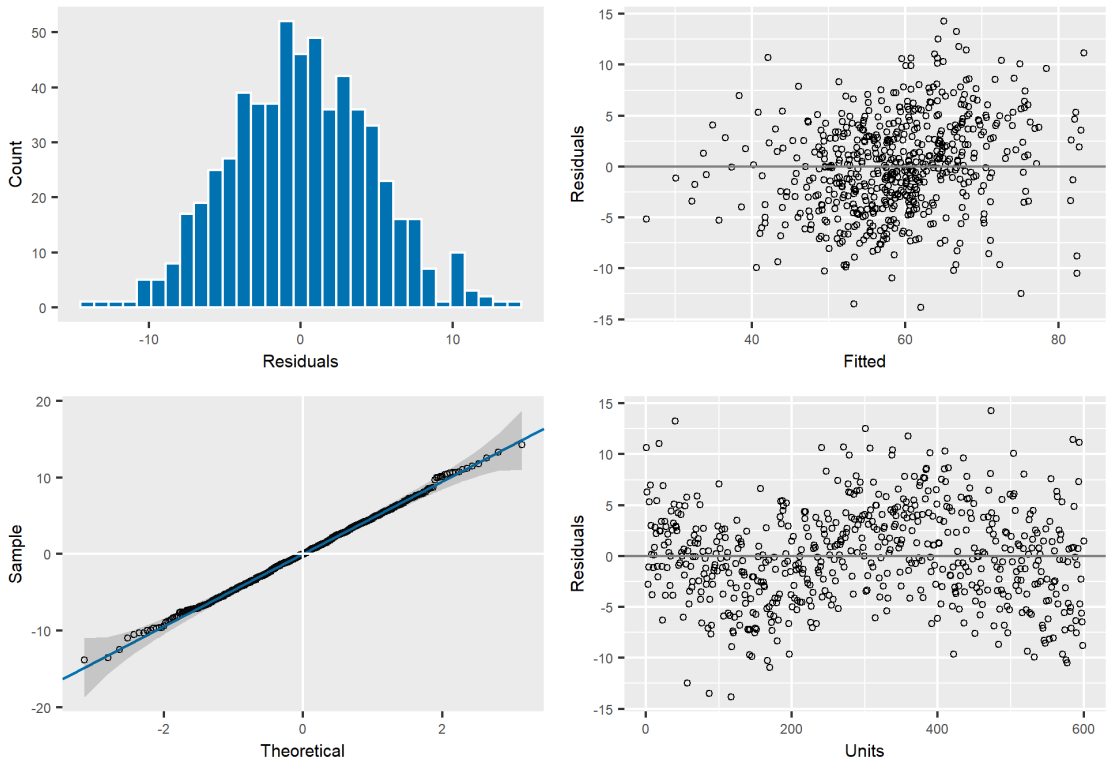
For users familiar with `ASReml-R` it is possible to identify each of the elements considered in the specification of the LMM. It is possible to use this `call` to fit the model directly with `ASReml-R` (and perform modifications if desired); however, note that the data frame referred here, `data`, needs to be replaced by your dataset.

The full `asreml` class object generated by fitting the single-trial model can be found under `$mod`. For example, we can request the variance component estimates and the residual plots using:

```
summary(mT1$mod)$varcomp
```

```
##                  component std.error  z.ratio bound %ch
## rep:ibk            0.96846  1.661305  0.58295     P 0.1
## at(check, 0):gen  68.51777  6.885433  9.95112     P 0.0
## row:col!R          1.00000        NA       NA     F 0.0
## row:col!row!cor    0.46699  0.057287  8.15167     U 0.0
## row:col!col!cor    0.26207  0.061640  4.25172     U 0.0
## row:col!col!var   30.52979  3.001035 10.17309     P 0.0
```

```
plot(mT1$mod)
```



The above output indicates that we have interesting, and possible significant, spatial correlations in both rows and columns.

Before exploring additional output, we should check what we have under `$outliers`. For the fitted model, there is nothing reported (indicated as `NA`, not shown), and this seems to agree with our residuals plot from above (which shows simple residuals not standardized residuals). If there were some suggested outliers, these will be identified with the number of their corresponding data row in the original data frame. Further evaluation of these outliers is required, and eventually these observations will need to be examined carefully directly from the data frame.

There are two other important elements from the output, these correspond to the goodness-of-fit statistics and the Wald statistics (or approximated ANOVA table in the context of LMM).

```
mT1$gof.stats
mT1$aov
```

```
##   n.vc    logL    AIC    BIC   Aopt logDopt   h2.vc h2.pev
## 1    5 -1511.6 3033.1 3055.1 9.4496  578.83 0.68507 0.8623
```

```
##                   Df denDF   F.inc         Pr
## (Intercept)        1  47.8 9949.00 1.0399e-52
## rep                1  16.7   50.38 2.4137e-06
## at(check, 1):gen   3 359.4  125.90 9.5818e-56
```

Several interesting goodness-of-fit statistics are presented. The most common ones are `logL`, `AIC` and `BIC`. These can be used only to compare models that have the same fixed effect model terms. The statistic `logL` can be used to perform some likelihood ratio tests (LRT) using the `ASReml-R` function `lrt.asreml()`. We have included the A and D optimality measures, corresponding to `Aopt` and `logDopt`, with the latter one expressed in a logarithm form. Both use the variance-covariance matrix of the prediction errors. The A-optimality value is defined as the average of the diagonal of this matrix, and therefore a lower value is associated with narrower confidence intervals of the genotype predictions. The D-optimality uses the determinant of the same matrix, and therefore a lower value represents a smaller volume of the hypersphere defined by the variance-covariance matrix of the prediction errors.

Finally, we have included some standard definitions of heritability that are only reported when genotype is assumed to be a random effect. Here, `h2.vc` is the heritability obtained as the ratio of the genetic variance component over the sum of all variance components (except correlations). A definition commonly used to compare spatial models is `h2.pev`, that uses the uses the predictor error variances (PEVs) from the BLUP values associated with each of the genotypes, its expression is: $h^2_{pev} = 1 - P\bar{E}V/{\sigma_g}^2$, where $P\bar{E}V$ is the PEVs mean and ${\sigma_g}^2$ is the genetic variance. For `ASRtriala` if checks are assumed fixed and test lines as random in the model, `Aopt`, `logDopt` and `h2.pev` only use the test genotypes for its calculations.

The Wald statistics reported were obtained by considering the calculation of denominator degrees of freedom using the algebraic calculations and by reporting the incremental (or sequential) sum of squares.

Finally, the additional output from this object is a data frame with the predictions of the genotypes for the fitted model with its corresponding weights. A few rows of this data frame are shown below:

```
head(as.data.frame(mT1$predictions), 10)
```

```
##    check        gen predicted.value std.error    status  weight
## 1      1    Camelot           64.082    1.1287 Estimable      NA
## 2      1    Freeman           79.075    1.1200 Estimable      NA
## 3      1 GOODSTREAK           54.790    1.1354 Estimable      NA
## 4      0    NE16401           61.801    3.0156 Estimable 0.13100
## 5      0    NE16402           71.016    3.0557 Estimable 0.13143
## 6      0    NE16403           56.209    3.0480 Estimable 0.13130
## 7      0    NE16404           53.541    3.0509 Estimable 0.13153
## 8      0    NE16405           57.505    3.0690 Estimable 0.12292
## 9      0    NE16406           66.729    3.1273 Estimable 0.12080
## 10     0    NE16407           45.167    3.0570 Estimable 0.13148
```

This data frame of predictions identifies in the column `check` the different types of genotypes (0 for check/controls and 1 for test lines). In addition, standard errors and weights are included. The latter are not available for the checks as these were treated as fixed effects. Further details of these weights will be presented later as part of the step 2 of the two-stage MET analyses.

The above example fitted a model that considered the trial's design structure together with an autorregressive error structure on both rows and columns. However, there are additional modelling options that can be used under the function `fit.single()`. For example, it is possible to include one or more covariates, if required, or add random effects of row and/or column within replicate by using the arguments `add.row` and `add.col`. Alternatively, spatial trends, in the form of polynomials of first and second order or cubic smoothing splines, can be included using the arguments `trend.row` and `trend.col`.

## 3.3 Selecting the *best* single-trial model

The above evaluation and fitting process can be repeated on an array of different model specifications where goodness-of-fit statistics can be extracted and used to select the *best* model for a single site. This process is facilitated in `ASRtriala` with the function `select.single()`. This function allows the evaluation of several spatial and non-spatial single-trial models using the same base input. All of the model specifications described above for the function `fit.single()` are available. The key of this procedure is the use of the data frame `model.setup`, which has a collection of 640 predefined spatial and non-spatial models with specifications (or configurations) to be used. The first and last rows of this data frame are shown below.

```
model.setup <- ASRtriala::model.setup
head(model.setup)
tail(model.setup)
```

```
##        model add.rep add.ibk add.row add.col
## 1 Model_0001   FALSE   FALSE   FALSE   FALSE
## 2 Model_0002   FALSE   FALSE   FALSE    TRUE
## 3 Model_0003   FALSE   FALSE    TRUE   FALSE
## 4 Model_0004   FALSE   FALSE    TRUE    TRUE
## 5 Model_0005   FALSE   FALSE    TRUE    TRUE
## 6 Model_0006   FALSE    TRUE   FALSE   FALSE
##   trend.row trend.col type.residual add.nugget
## 1      none      none          indep      FALSE
## 2      none      none          indep      FALSE
## 3      none      none          indep      FALSE
## 4      none      none    ar1.rowcol      FALSE
## 5      none      none          indep      FALSE
## 6      none      none          indep      FALSE
```

```
##          model add.rep add.ibk add.row add.col
## 635 Model_0635    TRUE    TRUE    TRUE    TRUE
## 636 Model_0636    TRUE    TRUE    TRUE    TRUE
## 637 Model_0637    TRUE    TRUE    TRUE    TRUE
## 638 Model_0638    TRUE    TRUE    TRUE    TRUE
## 639 Model_0639    TRUE    TRUE    TRUE    TRUE
## 640 Model_0640    TRUE    TRUE    TRUE    TRUE
##     trend.row trend.col type.residual add.nugget
## 635    spline    spline       ar1.col       TRUE
## 636    spline    spline       ar1.row      FALSE
## 637    spline    spline       ar1.row       TRUE
## 638    spline    spline    ar1.rowcol      FALSE
## 639    spline    spline    ar1.rowcol       TRUE
## 640    spline    spline         indep      FALSE
```

The above data frame contains several columns with the required specifications for each of the models. This is a large number of models but not all of the 640 models are always required. For example, if a trial does not contain incomplete blocks (*i.e.*, `add.ibk = FALSE`), then only a subset of these models needs to be considered. For this, the function `select.single()` will automatically select the corresponding subset according to which design factors are included. It is also important to respect the original design, for example, if a trial does include replicates, then these should be always present (*i.e.*, `add.rep = TRUE`).

Nevertheless, further reductions of this set might still be required. For example, in some experiments (*e.g.*, unreplicated trials) not all model terms are appropriate to be considered (such as row and/or columns within replicate). In order to do this, filtering or subsetting the data frame `model.setup` before being supplied to the function `select.single()` is necessary. In the following commands we will select a subset of models, specifically the ones that always include replicate effect and without any form of trend or nugget.

```
specs.single <- model.setup[model.setup$add.rep == TRUE &
    model.setup$trend.row == "none" & model.setup$trend.col ==
    "none" & model.setup$add.nugget == FALSE, ]
```

Now we can proceed to use the main function to fit these 14 models. Note that this subset of models are assigned using the argument `data.model = specs.single`.

```
models.stats <- select.single(data = st.data, gen = "gen",
    check = "check", rep = "rep", ibk = "ibk", row = "row",
    col = "col", resp = "yield", type.gen = "random",
    data.model = specs.single, threshold = 3.5, criteria = NULL)
```

Before we look at the goodness-of-fit statistics, it is important to check the output related to `$warnings` and `$outliers`. The warnings will report if there were some issues on fitting any of these 14 models, and the outliers will report a list with potential outliers per model fit, which is relevant to assess their quality. These tables are not shown here.

The most important part of the output from this function is the goodness-of-fit statistics shown below. If you note some rows filled with `NA` values, this will indicate that some models failed to converge. Often the reason is associated with an overparametrization and, in most cases this should not be of concern. All of these statistics were previously described, and they should help to rank and select the *best* model fit. Recall that models with different sets of fixed effects should not be compared using `logL`, `AIC` or `BIC`. In this particular case, using the criteria `Aopt` or `logDopt` the *best* model appears to be `Model_0014`, which does not include incomplete blocks but it does have the random effects of row and column and an autorregressive error structure on both rows and columns.

```
models.stats$gof.stats
```

```
##         model n.vc    logL    AIC    BIC    Aopt logDopt   h2.vc  h2.pev
## 1  Model_0011    2 -1545.2 3094.5 3103.2 12.5489  682.48 0.69144 0.81314
## 2  Model_0012    3 -1534.0 3074.0 3087.1 11.6662  659.01 0.69016 0.82669
## 3  Model_0013    3 -1533.9 3073.8 3087.0 11.8379  662.29 0.69039 0.82745
## 4  Model_0014    6 -1508.1 3028.3 3054.6  9.4286  578.66 0.68975 0.86402
## 5  Model_0015    4 -1515.9 3039.9 3057.4 10.4276  619.48 0.68897 0.85115
## 6  Model_0016    3 -1529.8 3065.5 3078.7 11.5598  654.96 0.69441 0.83366
## 7  Model_0017    4 -1529.1 3066.2 3083.7 11.5196  653.69 0.69275 0.83320
## 8  Model_0018    4 -1512.9 3033.9 3051.4 10.5074  620.37 0.69386 0.85482
## 9  Model_0019    7 -1505.7 3025.4 3056.1  9.7102  590.10 0.69094 0.86300
## 10 Model_0020    5 -1510.8 3031.6 3053.6 10.3430  614.53 0.69099 0.85625
## 11 Model_0411    5 -1515.2 3040.5 3062.4 10.3506  616.38 0.68938 0.85253
## 12 Model_0413    5 -1510.3 3030.6 3052.5  9.7270  592.08 0.68833 0.85958
## 13 Model_0566    6 -1510.6 3033.2 3059.5 10.3040  613.02 0.69029 0.85683
## 14 Model_0568    6 -1506.1 3024.1 3050.4  9.7847  593.08 0.69128 0.86223
```

```
##    add.rep add.ibk add.row add.col trend.row trend.col type.residual add.nugget
## 1     TRUE   FALSE   FALSE   FALSE      none      none          indep      FALSE
## 2     TRUE   FALSE   FALSE    TRUE      none      none          indep      FALSE
## 3     TRUE   FALSE    TRUE   FALSE      none      none          indep      FALSE
## 4     TRUE   FALSE    TRUE    TRUE      none      none     ar1.rowcol      FALSE
## 5     TRUE   FALSE    TRUE    TRUE      none      none          indep      FALSE
## 6     TRUE    TRUE   FALSE   FALSE      none      none          indep      FALSE
## 7     TRUE    TRUE   FALSE    TRUE      none      none          indep      FALSE
## 8     TRUE    TRUE    TRUE   FALSE      none      none          indep      FALSE
## 9     TRUE    TRUE    TRUE    TRUE      none      none     ar1.rowcol      FALSE
## 10    TRUE    TRUE    TRUE    TRUE      none      none          indep      FALSE
## 11    TRUE   FALSE    TRUE    TRUE      none      none        ar1.col      FALSE
## 12    TRUE   FALSE    TRUE    TRUE      none      none        ar1.row      FALSE
## 13    TRUE    TRUE    TRUE    TRUE      none      none        ar1.col      FALSE
## 14    TRUE    TRUE    TRUE    TRUE      none      none        ar1.row      FALSE
```

## 3.4 Fitting final selected *best* model.

Now we are ready to proceed to fit the selected model `Model_0014` using the following lines of code:

```
sel.m <- models.stats$gof.stats[4, ]
mT1.f <- fit.single(data = st.data, data.model = sel.m,
    threshold = 3.5)
```

The above way is more automatic. Alternatively we can fit the same model in a more detailed way using:

```
sel.m <- models.stats$gof.stats[4, ]
mT1.f <- fit.single(data = st.data, gen = "gen", check = "check",
    rep = "rep", ibk = "ibk", row = "row", col = "col",
    resp = "yield", type.gen = "random", type.rep = "fixed",
    add.rep = sel.m$add.rep, add.ibk = sel.m$add.ibk,
    add.row = sel.m$add.row, add.col = sel.m$add.col,
    trend.row = sel.m$trend.row, trend.col = sel.m$trend.col,
    type.residual = sel.m$type.residual, add.nugget = sel.m$add.nugget,
    threshold = 3.5)
```

In both of the above cases, we have used the setting/configuration for `Model_0014` by filtering row four on the table `$gof.stats`, and using that information directly into the `fit.single()` function, but the former is using the convenient argument `data.model` referring to the meta-data of the selected model. As done before, we can take a look at all output, but in this case we will present only the variance component estimates and a few of the lines of predictions.

```
summary(mT1.f$mod)$varcomp
head(as.data.frame(mT1.f$predictions))
```

```
##                   component std.error z.ratio bound %ch
## rep:row             3.30774  1.726214 1.91618     P 0.1
## rep:col             1.47740  1.895125 0.77958     P 0.3
## at(check, 0):gen   68.72016  6.890146 9.97369     P 0.0
## row:col!R           1.00000        NA      NA     F 0.0
## row:col!row!cor     0.42743  0.068949 6.19920     U 0.1
## row:col!col!cor     0.18089  0.070847 2.55327     U 0.1
## row:col!col!var    26.12490  2.991930 8.73179     P 0.0
```

```
##   check        gen predicted.value std.error    status  weight
## 1     1    Camelot           63.929    1.1539 Estimable      NA
## 2     1    Freeman           79.334    1.1520 Estimable      NA
## 3     1 GOODSTREAK           54.933    1.1552 Estimable      NA
## 4     0    NE16401           61.644    3.0229 Estimable 0.12969
## 5     0    NE16402           71.450    3.0446 Estimable 0.13010
## 6     0    NE16403           56.159    3.0437 Estimable 0.13003
```

A few differences can be noted from our previous model fit for the same data, but this is expected as we are using what we consider the *best* model. Now it is possible to use the relevant output to draw our conclusions and perform breeding decisions as required.

If several trials need to be analyzed, a similar procedure as the one executed above can be repeated. If these single-site analyses need to be combined into a MET analyses, then there are some aspects to take into consideration, which will be the focus in the next section.

# 4 Multi-Environmental Trial Analyses

One of the main objectives of `ASRtriala` is to perform MET analyses in an easy and efficient manner. In this package, we focus on the implementation of two-stage analyses as proposed by Smith et al. (2001a). Here, the first step is to collect the information from each of the single-trial analyses with their corresponding weights, and the second step focuses on fitting complex variance-covariance structures to model GxE as nested effects.

The workflow for MET analyses based on this two-stage procedure involves the following tasks:

- Collecting pre-processed data (predictions) from single-trial analyses.
- Preparing and auditing MET data.
- Selecting the *best* MET model.
- Fitting final selected *best* MET model.
- Enhancing output (*e.g.*, biplots, stability indexes).
- Extracting and interpreting relevant output.

## 4.1 Collecting predictions from single-trial analyses

All procedures described in the previous sections about fitting and selecting the *best* model for spatial or non-spatial analysis on each of the trials applies to this part. In terms of workflow, the process involving single-trial analyses is assumed to have occurred earlier, and possibly a database system that manages and stores these analyses is available. Hence, in this step, we collect the relevant trials (for example, if the interest is on a range of years for a specific breeding zone), and relevant genotypes tested on these trials (often all genotypes but it could be a subset according to specific objectives or good representation).

In any case, for each of the trials it is required to have the mean genotype predictions with their corresponding weights. `ASRtriala` follows the procedure proposed by Smith et al. (2001a) and with additional details presented by Gogel et al. (2018). The logic is based on mimicking a single-stage MET analysis as much as possible, where first the predictions from the single-trial analyses are based on variance components estimated with genotypes considered as random effects (as would be done in a single-stage MET analysis) but avoiding the shrinkage that occurs on BLUP values. This is achieved by fitting the single-site model first with genotype as a random effect, and then fixing all variance components into a re-fitting of the same model, but this time with the genotypes considered as fixed effects. This will be illustrated later in a detailed example.

The second aspect is to obtain weights for each of the predictions to be included into a **weighted linear mixed model** analysis for the MET model. Here, the weights correspond to the diagonal of the inverse of the variance-covariance matrix of the prediction errors. This concept approximates the inverse of the mixed model equations that will be obtained under a single-stage MET analysis.

Earlier, we fitted our *best* model for location `Alliance` and this was stored in the object `mT1.f`. We will use the same arguments but this time the additional argument `fix.vc = TRUE` and store the output in the object `mT1.r`:

```
mT1.r <- fit.single(data = st.data, data.model = sel.m,
    fix.vc = TRUE, threshold = 3.5)
```

The argument `fix.vc = TRUE` will first fit genotype as random (as specified under `type.gen = "random"`), and then as fixed to obtain un-shrunken predictions. An important warning about the output from the the latter object, `mT1.r`, is that its goodness-of-fit statistics are based on the model with both checks and test lines are treated as a single fixed effect model term.

Some of the relevant output from this model is:

```
ls(mT1.r)
summary(mT1.r$mod)$varcomp
head(as.data.frame(mT1.r$predictions))
```

```
## [1] "aov"         "call"        "gof.stats"   "mod"         "outliers"
## [6] "predictions" "vc.table"    "warnings"
```

```
##                  component std.error z.ratio bound %ch
## rep:row            3.30774        NA      NA     F   0
## rep:col            1.47740        NA      NA     F   0
## row:col!R          1.00000        NA      NA     F   0
## row:col!row!cor    0.42743        NA      NA     F   0
## row:col!col!cor    0.18089        NA      NA     F   0
## row:col!col!var   26.12490        NA      NA     F   0
```

```
##   check        gen predicted.value std.error    status  weight
## 1     1    Camelot          63.887    1.1659 Estimable 1.10413
## 2     1    Freeman          79.619    1.1653 Estimable 1.12409
## 3     1 GOODSTREAK          54.567    1.1680 Estimable 1.10690
## 4     0    NE16401          62.667    3.2765 Estimable 0.11559
## 5     0    NE16402          73.800    3.3087 Estimable 0.11561
## 6     0    NE16403          55.822    3.3054 Estimable 0.11561
```

First, we have the same list of objects as before, and again it is important to verify the model fit with `$warnings` and `$outliers`. Then, as expected, the variance components are the same as in the previous model fit `mT1.f`, but you will notice that the predictions are slightly different and also with slightly different weights. An interesting difference here is that both checks and test genotypes all have weights.

## 4.2 Preparing and auditing MET data.

To illustrate the implementation of the two-stage analysis within `ASRtriala` we are going to continue using the study from wheat (Belamkar et al. 2018) presented before. However, in this case we will use the predictions of those eight locations based on fitting and selecting, individually in each trial the *best* model as shown earlier. This has already been done, and the model results collected in a data frame, for which we show only the first six rows of data together with a simple enumeration of all levels for the factor location.

```
data.wheat.preds <- ASRtriala::pheno.wheat.preds
head(data.wheat.preds)
```

```
##   location check        gen predicted.value std.error    status  weight
## 1 Alliance     1    Camelot          66.718    1.4601 Estimable 1.06040
## 2 Alliance     1    Freeman          82.237    1.4400 Estimable 1.06634
## 3 Alliance     1 GOODSTREAK          57.315    1.4924 Estimable 1.05411
## 4 Alliance     0    NE16401          65.637    3.3369 Estimable 0.10842
## 5 Alliance     0    NE16402          75.964    3.3544 Estimable 0.10919
## 6 Alliance     0    NE16403          58.878    3.3147 Estimable 0.10907
```

```
levels(data.wheat.preds$location)
```

```
## [1] "Alliance"     "Clay_Center"  "Grant_D"      "Lincoln"      "Lincoln_IM"
## [6] "McCook"       "North_Platte" "Sidney"
```

Now, we can proceed with some auditing of this predictions dataset. For this we use the function `audit.met()` which provides a couple of important elements to assess the quality of the data and if it will be adequate to use on downstream MET analyses. This is done using the code below that requires the specification of the columns in the data frame where trial, genotype and the response variable are found.

```
audit.all <- audit.met(data = data.wheat.preds, trial = "location",
    gen = "gen", resp = "predicted.value")
```

The output from the above function is:

```
audit.all$met.stats
audit.all$gen.inc
```

```
##       location   n    min   mean     max      sd missing    CVp
## 1     Alliance 273 28.647 61.133  82.237  9.0423       0 14.791
## 2  Clay_Center 273 10.904 37.243  72.208 10.1882       0 27.356
## 3      Grant_D 269 40.198 71.938 105.185  9.9343       4 13.810
## 4      Lincoln 273 20.400 54.700  80.209 11.0673       0 20.233
## 5   Lincoln_IM 272 31.700 79.757 121.779 13.0327       1 16.340
## 6       McCook 272  4.836 68.458 109.703 15.0114       1 21.928
## 7 North_Platte 273 33.646 65.816  92.421 11.2352       0 17.070
## 8       Sidney 273 22.758 56.806  84.375 11.0618       0 19.473
```

```
##
##                Alliance Clay_Center Grant_D Lincoln Lincoln_IM McCook North_Platte Sidney
##   Alliance          273         273     269     273        272     272          273    273
##   Clay_Center       273         273     269     273        272     272          273    273
##   Grant_D           269         269     269     269        268     268          269    269
##   Lincoln           273         273     269     273        272     272          273    273
##   Lincoln_IM        272         272     268     272        272     271          272    272
##   McCook            272         272     268     272        271     272          272    272
##   North_Platte      273         273     269     273        272     272          273    273
##   Sidney            273         273     269     273        272     272          273    273
```

The first element is a data frame with the summary statistics by trial or location. This is relevant to assess the quality of the phenotypic response. However, note that these statistics are based on model predictions (or adjusted means) and therefore they present less variability than the raw observations. Here, we can for example check for unusual overall means on some locations, or large numbers of missing values. In this example, the column `CVp`, that corresponds to the coefficient of variation, shows for location `Clay_Center` a value of 27.4% which is an indication of large background noise. Also this location has the lowest mean value.

The other element reported is a matrix of incidence of genotypes by trials. This is critical to assess the level of **connectivity** between trials. In this particular case, values in the off-diagonal represent the common genotypes between locations, all of these numbers are 268 or larger (note that this function does not count those observations on the response variable that are `NA`). The values on the diagonal are the count of non-`NA` observations by location. This matrix indicates a very good connectivity between locations, enabling us to estimate genetic correlations between trials with sufficient information. In general, we recommend that a minimum of 5 genotypes should be common between any pair of trials.

## 4.3 Selecting the *best* MET model.

In a similar way as done with the single-trial analysis flow, we are in a position to fit different MET structures to select the *best* model. For this task we will use the function `select.met()`. In contrast with the function `select.single()`, this function is simpler with a limited set of options, as most of the design terms have been considered in the first step.

For our MET dataset the code to use is:

```
met.stats <- select.met(data = data.wheat.preds, trial = "location",
    gen = "gen", resp = "predicted.value", weight = "weight",
    type.trial = "fixed", vc.models = c("corv", "corh",
        "fa1", "fa2", "fa3", "fa4"), criteria = "AIC")
```

In the above code, besides the required specification of the columns from our data frame (including weights), we have indicated that we want the trial effect to be `fixed`. Also there

is a total of six complex GxE nested models that we want to evaluate, these are: a common correlation and variance (`corv`), a common correlation but with heterogeneous variances (`corh`), and four factor analytic models (`fa1` to `fa4`). Descriptions of these models can be found in the `ASReml-R` manual, and statistical details on the factor analytic (FA) model can be found in Smith et al. (2001b). Finally, we have specified that the criteria to identify the *best* model is based on `AIC`.

It is important to note that both `select.met()` and `fit.met()` functions will eliminate from the data frame supplied any observation (prediction) that has no weight (*i.e.*, `NA`). If an analysis is of interest for data with no available weights, it is recommended that a vector of ones is used as weights. For the `fa4` model there is a message associated with being over-parametrized (not shown). This occurs when the FA structure has more variance components than are required under an unstructured (or corgh) model structure. However, `ASReml-R` will deal with this issue by fixing some components to zero.

The main output from the above function is the table of goodness-of-fit statistics, which is shown below.

```
met.stats$gof.stats
```

```
##   vc.models n.vc    logL   AIC   BIC
## 6       fa4   33 -5981.9 12030 12217
## 5       fa3   29 -5987.2 12032 12197
## 4       fa2   22 -5997.8 12040 12165
## 1      corh   29 -5998.4 12055 12220
## 3       fa1   16 -6028.4 12089 12180
## 2      corv    2 -6098.2 12200 12212
```

The above table is sorted according to AIC, and indicates that model `fa4` is the *best* model. This result does not agree with the use of the more conservative BIC. Also, the difference in log-likelihood values (`logL` between `fa3` and `fa4` is relatively small given the additional four parameters. These two models can be compared formally with a likelihood ratio test (LRT) using the `ASReml-R` function `lrt.asreml()`. In the above table there is no heritability reported as there is no clear definition of this statistic when weights are incorporated into the model fit.

There is additional output associated with the *best* model. It is important to check the `$best.warning` in case the selected model has not fully converged or has any issues. For this example, there are no issues (not shown). Another important output is the model call, as shown below, that provides the confirmation of the definition of the MET model:

```
met.stats$best.call
```

```
## asreml::asreml(fixed = predicted.value ~ 1 + location, random = ~fa(location,
##     4):id(gen), data = data, na.action = list(x = "include",
##     y = "include"), weights = weight, family = asreml::asr_gaussian(dispersion = 1),
##     workspace = 1.28e+08)
```

From the above ouput it is easy to identify which elements were considered in the selected model. As indicated before, this `call` can be used to fit the model directly with `ASReml-R`.

## 4.4 Fitting final selected *best* MET model.

Now we are in a position to fit the final selected model `fa4`. This is shown in the following code:

```
met.model.fa4 <- fit.met(data = data.wheat.preds, trial = "location",
    gen = "gen", resp = "predicted.value", weight = "weight",
    type.gen = "random", type.trial = "fixed", vc.model = "fa4")
```

The input for this function is almost identical to `select.met()`. The output from fitting this model contains several elements, these are:

```
## [1] "call"        "corr.g"      "fa.loadings" "gof.stats"   "mod"
## [6] "predictions" "vcov.g"      "vcov.pred"   "warnings"
```

We will present and describe in detail a few of these elements in the following sections. As before, one of the most important reports to look at is the messages under `$warnings`, which is `NA` in this case. As with `fit.single()` we can access the full `asreml` class object or parts of it under `$mod`, for example with `summary(met.model.fa4$mod)$varcomp`. The goodness-of-fit statistics, `$gof.stats`, provides the same results as the table from `select.met()` but only for the fitted model.

Probably the most relevant output is the estimation of the **type B** genetic correlation, corresponding to a matrix with the genetic correlation between locations, which is critical to assess and study the patterns of genotype-by-environment interactions. This can be obtained with:

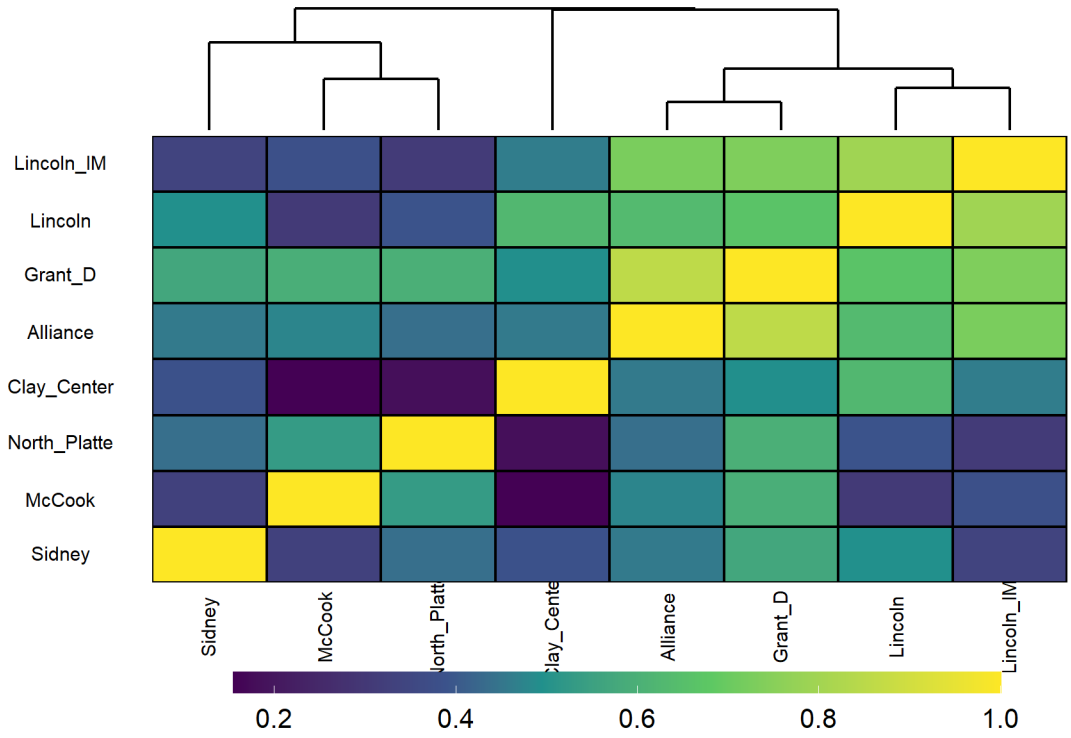
```
met.model.fa4$corr.g
```

```
##              Alliance Clay_Center Grant_D Lincoln
## Alliance        1.000       0.457   0.856   0.635
## Clay_Center     0.457       1.000   0.493   0.627
## Grant_D         0.856       0.493   1.000   0.666
## Lincoln         0.635       0.627   0.666   1.000
## Lincoln_IM      0.724       0.462   0.734   0.795
## McCook          0.475       0.156   0.595   0.306
## North_Platte    0.438       0.180   0.599   0.389
## Sidney          0.457       0.385   0.570   0.496
```

```
##              Lincoln_IM McCook North_Platte Sidney
## Alliance          0.724  0.475        0.438  0.457
## Clay_Center       0.462  0.156        0.180  0.385
## Grant_D           0.734  0.595        0.599  0.570
## Lincoln           0.795  0.306        0.389  0.496
## Lincoln_IM        1.000  0.381        0.310  0.339
## McCook            0.381  1.000        0.530  0.330
## North_Platte      0.310  0.530        1.000  0.438
## Sidney            0.339  0.330        0.438  1.000
```

Interestingly, for this example, all type B correlations are positive ranging from 0.156 to 0.856. In general, it is easier to visualize these correlations using a heatmap. We will use the functions from library **ASRgenomics** (available at https://vsni.co.uk/free-software/asrgenomics) as shown in the code below:

```
library(ASRgenomics)
ASRgenomics::kinship.heatmap(K = met.model.fa4$corr.g,
    dendrogram = TRUE, dist.method = "canberra", row.label = TRUE,
    col.label = TRUE)
```



The above plot shows clustering of some locations that can be useful to verify or define new breeding zones, or to identify some sites with problems. In this example the location `Clay_Center` appears somehow separated from the other sites.

It is also possible to obtain the genetic variance-covariance matrix between locations, which is included within the list `$vcov.g`. This matrix will be important for obtaining biplots, as shown later.

We can proceed to see the loadings for each of the factors, which correspond to variance components estimated by `ASReml-R` and they can be found under `$fa.loadings`.

```
met.model.fa4$fa.loadings
```

```
##                 FA1      FA2     FA3     FA4
## Alliance     7.4591  0.00000  0.0000 0.00000
## Clay_Center  5.0937 -4.68862  0.0000 0.00000
## Grant_D      7.8899  0.33312  1.5713 0.00000
## Lincoln      7.3556 -5.18506 -1.3120 4.43939
## Lincoln_IM   7.8676 -0.55089 -3.6830 2.44714
## McCook       6.7223  3.30239  3.2618 2.32268
## North_Platte 5.0262  1.71091  5.1911 4.10106
## Sidney       5.5359 -2.44554  3.6174 0.99827
```

However, to facilitate interpretation it is recommended to use some form of rotation, this will be shown in the next section.

And finally, the most important result is the predictions based on the MET analysis using the `fa4` structure. This is obtained from `$predictions`. In the code below we only present the results for the variety `Freeman`.

```
met.model.fa4$predictions[met.model.fa4$predictions$gen ==
    "Freeman", ]
```

```
##          location     gen predicted.value std.error    status extrap
## 2        Alliance Freeman          81.922   0.94647 Estimable  FALSE
## 275   Clay_Center Freeman          45.342   0.88876 Estimable  FALSE
## 548       Grant_D Freeman          87.910   1.59695 Estimable  FALSE
## 821       Lincoln Freeman          68.734   1.42379 Estimable  FALSE
## 1094   Lincoln_IM Freeman          98.328   2.44535 Estimable  FALSE
## 1367       McCook Freeman          93.007   2.59150 Estimable  FALSE
## 1640 North_Platte Freeman          81.416   1.55919 Estimable  FALSE
## 1913       Sidney Freeman          74.637   0.91991 Estimable  FALSE
```

This is very important output as it provides the expected performance of the genotype of interest in each of the locations. For most GxE structures (except for `diag`), `ASReml-R`

will calculate predictions for each genotype in each location, even on those locations where they are not present. This is facilitated by having a matrix of type B genetic correlation estimates; however, this means that some genotype estimates are **extrapolations** based on the other sites. For this reason, `ASRtriala` includes in the prediction table, as shown above, an additional column called `extrap` that identifies those predictions on which a genotype was not present on a given site.

# 5 Enhancing output from MET models.

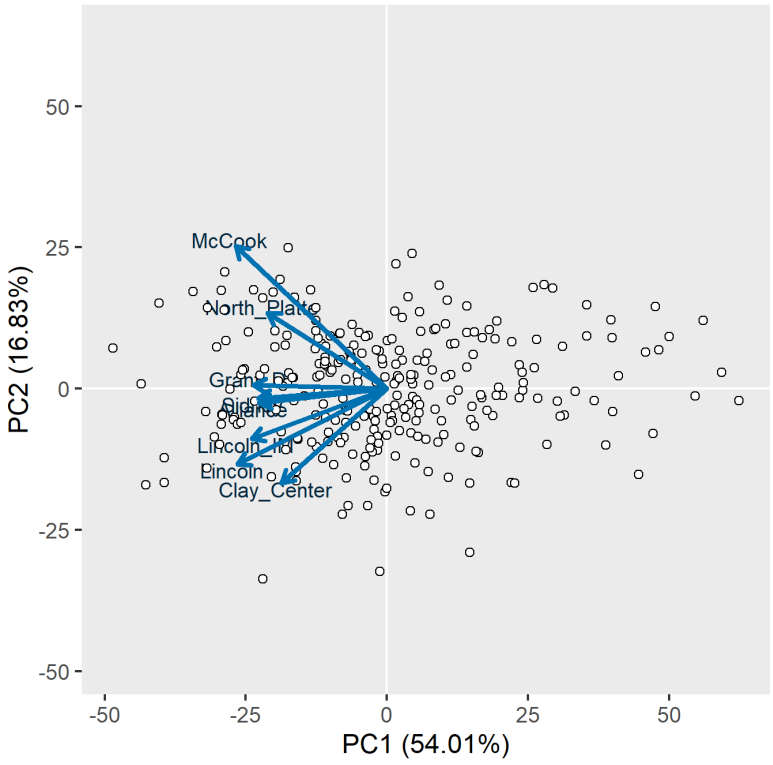
In this section we will present a few additional functions from `ASRtriala` that can be useful to make the most of your results from the MET analyses.

## 5.1 Obtaining a Biplot

First we will start by generating a **biplot**, which is a type of scatterplot that is useful to display multivariate data. For example, in the context of predictions from a MET analysis, where the trials or locations correspond to the vectors and the genotypes to the units. The function `gbiplot()` can be used within `ASRtriala` for this purpose. The input corresponds to a data frame with the predictions by genotype and location (as with `met.model.fa4$predictions`), and from here a variance-covariance (or correlation) matrix between locations is calculated. However, it is recommended to use the actual variance-covariance matrix estimated from the MET analysis (*i.e.*, `$vcov.g`).

In the example below we use the predictions from the model fit object `met.model.fa4`, and we provide the `$vcov.g` directly from there. Also, other arguments have been used to help display the data.

```
gbiplot(data = met.model.fa4$predictions, pcs = c(1,
    2), vector = "location", unit = "gen", resp = "predicted.value",
    vcov.g = met.model.fa4$vcov.g, scale = FALSE, vector.label = TRUE,
    unit.label = FALSE)
```



In the above graphical display it is clear that all vectors (in this case locations) are pointing more or less to the same direction (center left), and their angles are relatively small indicating high levels of correlation between the vectors (*i.e.*, locations).

Biplots can be used to facilitate the identification of the *best* genotypes. For example, in the above plot, the four genotypes located on the extreme left are all outstanding genotypes performing consistently, across all locations, some of the highest predicted values, with the variety `Freeman` corresponding to the one farthest to the left. However, note that the two dimensions presented in the above biplot correspond to ~71% of the total variability; hence, some information is lost in this representation.

This function also allows for adding the labels to the units (*i.e.*, genotypes) or to show different pairs of principal components (*e.g.*, PC1 vs. PC3). In addition, the function `gbiplot()` as implemented in `ASRtriala` can read prediction tables where some of the genotypes have missing records on some locations. In this case, the function will proceed to perform **imputation** using the library `mice`. This should be done with care, and only for small proportions of missing data. Further details can be found in the help associated with this function.

## 5.2 Calculating Stability Indexes

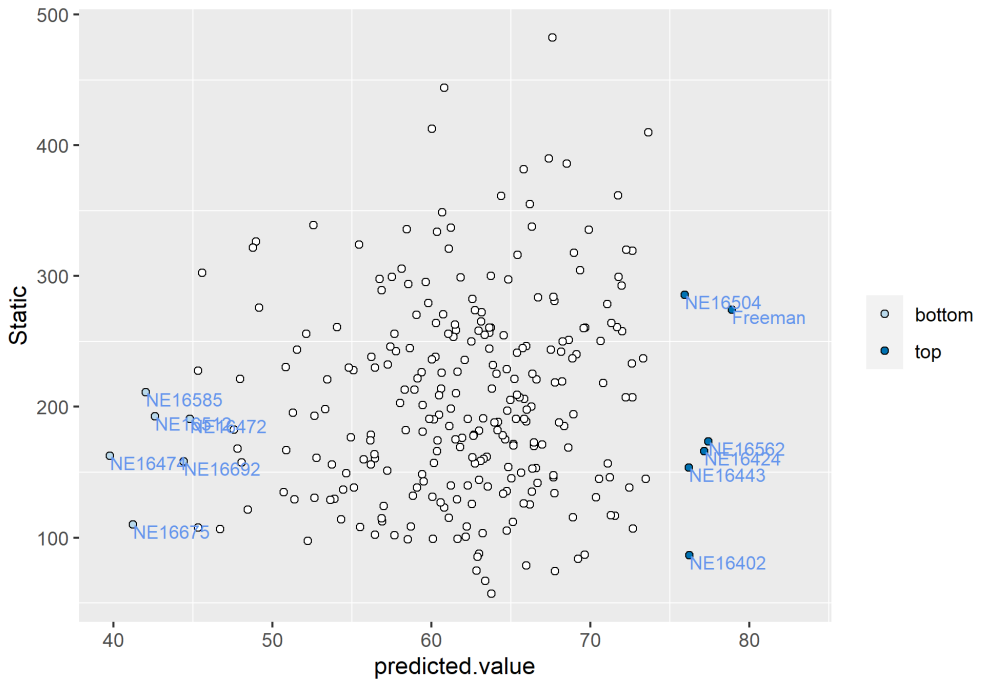
The selection of the best genotypes by observing their performance across several sites or environments is often challenging. This task is facilitated by the use of biplots (as indicated above) but also with the calculation of stability measures or coefficients. `ASRtriala` includes the function `stability()` that calculates four indexes: superiority, static, Wricke and rank. More details can be found in the documentation for this function. However, these are only a few of the measures available in the literature.

We will illustrate this function by using the same predictions table used in `gbiplot()`. The code below calculates the `static` stability coefficient, which is defined, for each genotype, as the variance between its predictions between locations. Hence, the smaller the value the more stable a genotype is. This index only provides a measure of the consistency of the genotype and not its performance, but this is helped by some enhanced graphical output.

```
stab.index <- stability(data = met.model.fa4$predictions,
    trial = "location", gen = "gen", resp = "predicted.value",
    method = "static", best = "min", plot = TRUE, top = TRUE,
    bottom = TRUE, percentage = 2)
```

The function above, besides requiring the specification of the data frame and its columns, allows for additional options of plots and levels of genotypes to assist with interpretation. The plot shown below is found under `$stability.plot` and you can find on the top right some of the most unstable genotypes. In the bottom right we have genotype `NE16402`, which is one of the most stable ones, and also presents a very good phenotypic mean (~76). There are other genotypes with better yield, for example `Freeman` is the one farthest to the right

but it is quite unstable. Their predicted values are presented in the table below, where it is easy to see that NE16402 is more stable.




```
subset.preds <- met.model.fa4$predictions[met.model.fa4$predictions$gen ==
    "NE16402" | met.model.fa4$predictions$gen == "Freeman",
    ]
subset.preds[order(subset.preds$gen), ]
```

```
##          location     gen predicted.value std.error    status extrap
## 2        Alliance Freeman          81.922   0.94647 Estimable  FALSE
## 275   Clay_Center Freeman          45.342   0.88876 Estimable  FALSE
## 548       Grant_D Freeman          87.910   1.59695 Estimable  FALSE
## 821       Lincoln Freeman          68.734   1.42379 Estimable  FALSE
## 1094   Lincoln_IM Freeman          98.328   2.44535 Estimable  FALSE
## 1367       McCook Freeman          93.007   2.59150 Estimable  FALSE
## 1640 North_Platte Freeman          81.416   1.55919 Estimable  FALSE
## 1913       Sidney Freeman          74.637   0.91991 Estimable  FALSE
## 5        Alliance NE16402          74.848   2.63940 Estimable  FALSE
## 278   Clay_Center NE16402          62.311   2.58402 Estimable  FALSE
## 551       Grant_D NE16402          83.796   3.10765 Estimable  FALSE
## 824       Lincoln NE16402          72.317   3.72169 Estimable  FALSE
## 1097   Lincoln_IM NE16402          93.219   4.71437 Estimable  FALSE
## 1370       McCook NE16402          79.373   6.49888 Estimable  FALSE
## 1643 North_Platte NE16402          73.101   4.25059 Estimable  FALSE
## 1916       Sidney NE16402          70.793   2.69808 Estimable  FALSE
```

## 5.3 Enhancing factor analytic output

One advantage of fitting a factor analytic model for multi-environment data is that both the variance components and the random effects associated with the GxE nested structure provide relevant information to understand the dynamics of this GxE. They also help with selecting the *best* genotypes across environments in terms of identifying those adapted to some specific environments (specificity) or those that are stable across most environments (broad adaptability). A more detailed case illustrating an application of this under an FA structure can be found at Oliveira et al. (2020).

We will be using the function `fa.summary()` from `ASRtriala` to obtain additional output and to generate informative plots. In the following code we are requesting several elements:

```
fa.extras <- fa.summary(mod.met = met.model.fa4, gen.id = "Freeman",
    factor.loading = 1, type.resp = "blup", type.plot = "regression.plot",
    rotation = "svd", trend = "coefficient")
```

Let's first start by looking at some of the elements that are part of the FA model. Recall that earlier we observed the loadings for each of the factors, but their interpretation is facilitated with the use of a rotation; in this case we have requested the Singular Value Decomposition (`svd`) rotation which is shown below:

```
fa.extras$fa.loadings.rot
```

```
##                 FA1       FA2     FA3      FA4
## Alliance     7.0615 -0.099745  1.7615 -1.63152
## Clay_Center  5.4564 -3.351806 -2.0365 -1.66581
## Grant_D      7.6214  1.243889  1.0574 -2.01984
## Lincoln      8.6726 -4.354766 -1.7765  2.25524
## Lincoln_IM   7.7067 -2.937918  3.4668  1.30847
## McCook       6.9399  4.700439  1.3020  0.42163
## North_Platte 6.2613  5.101829 -1.7890  1.87697
## Sidney       6.2892  0.851746 -2.9464 -1.32241
```

This rotation affects not only the loadings (variance components) but also some of the BLUP values, particularly the **coefficients**, which are associated with the slope of each of the factor for a given genotype. These are shown below for `Freeman`, and a large positive (or negative) value indicates a strong sensitivity to the underlying (unknown) factor.

```
fa.extras$comp.gen.rot$solution
```

```
## [1] 1.96595 0.43299 0.77080 0.13012
```

Another very important output that is part of the complete model is the proportion of the cumulative variance explained by each of the factors (in this case four). This can be obtained for all locations together or by trial, as shown below.

```
fa.extras$cum.var.rot
```

```
##     var.exp% cum.var.exp%
## FA1  50.4873       50.487
## FA2  11.2843       61.772
## FA3   4.6940       66.466
## FA4   2.7522       69.218
```

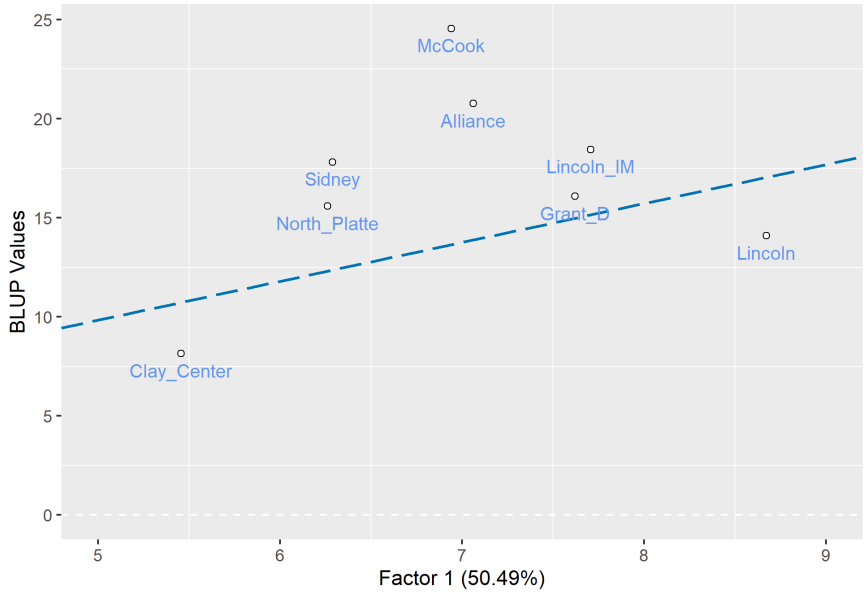
```
fa.extras$cum.var.trial.rot
```

```
##                  FA1       FA2     FA3     FA4 cum.var.exp%
## Alliance      68.469  0.013661  4.2606 3.65501       76.398
## Clay_Center   31.408 11.851971  4.3754 2.92742       50.563
## Grant_D       89.596  2.386613  1.7246 6.29291      100.000
## Lincoln       73.437 18.516016  3.0813 4.96597      100.000
## Lincoln_IM    65.907  9.578025 13.3366 1.89987       90.722
## McCook        31.451 14.428110  1.1070 0.11609       47.103
## North_Platte  38.900 25.826722  3.1757 3.49567       71.398
## Sidney        35.351  0.648374  7.7588 1.56292       45.321
```

The above output indicates that approximately 69.2% of the total genetic variability (genotype + genotype-by-environment) is explained by these four rotated factors, where the first rotated factor explains a large portion (50.5%). Also we observe that for some for locations the genetic component is completely described by these four factors (*e.g.*, `$Lincoln`), whilst others have a large portion of unexplained genetic variation (*e.g.*, `$Sidney`). All of these are good result to help make further inferences on this analysis.

In the above code we requested some specific output for the genotype `Freeman` but this can be requested for any genotype. Let's now obtain a plot for this genotype using its BLUP values (`type.resp = "blup"`) with a regression (`type.plot = "regression.plot"`) where the slope of the regression line is described by its BLUP component coefficient (`trend = "coefficient"`). We will also request the associated data frame (`$fa.gen`) that presents the relevant information for this genotype.

```
fa.extras$fa.plot
fa.extras$fa.gen
```



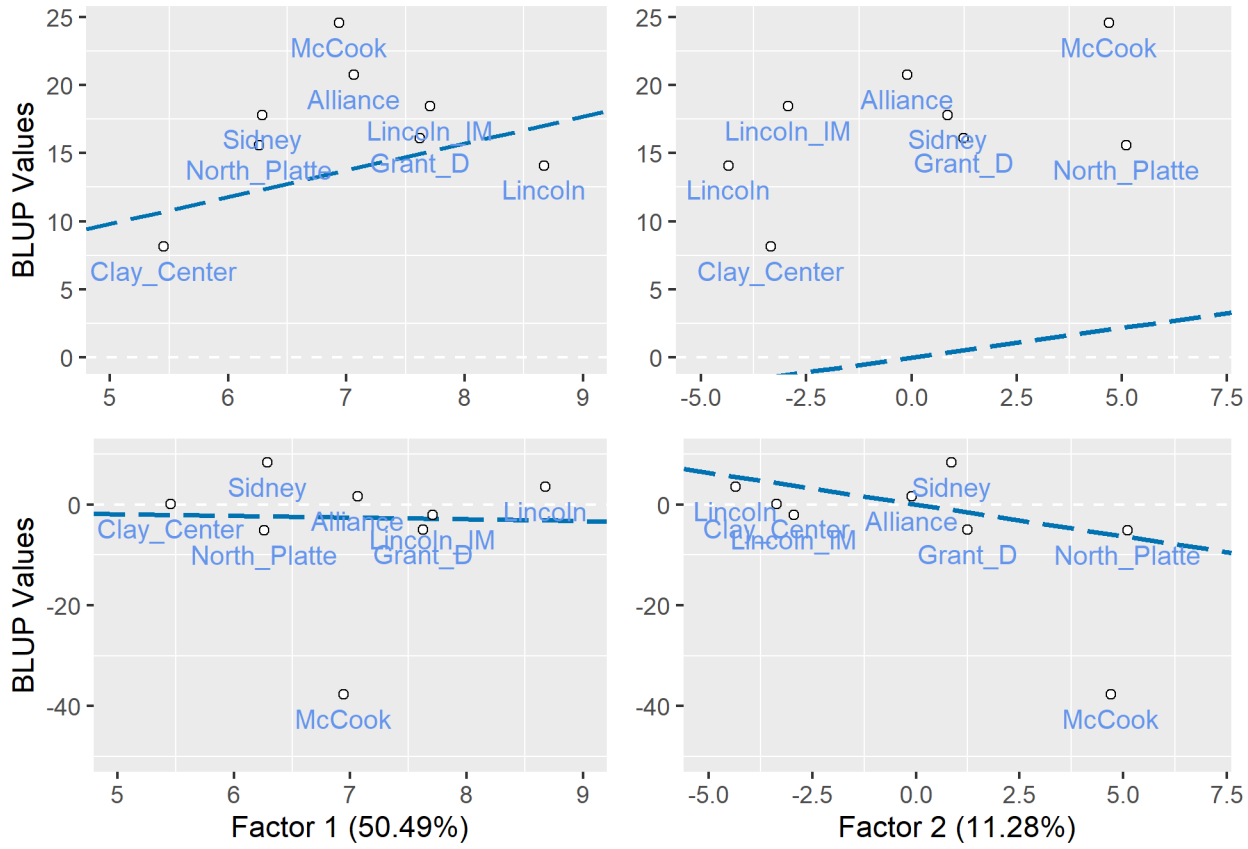

The above plot indicates a strong association of the genotype, with the first factor loading showing a positive trend (in this case with a slope of 1.97). However, the relationship is not very strong with some large deviations of the BLUP values with respect to the regression line, reflecting unexplained GxE by this factor (although some is explained by the other factors). This indicates an important sensitivity of this genotype to the first underlying factor. Similar plots can be obtained for the other factors, but also it is possible to request the added variable plot (`type.plot = "added.plot"`) that will show only the contribution of the respective factor to the BLUP (or predicted) value.

```
##          location     gen predicted.value blup.value
## 2        Alliance Freeman          81.922    20.7622
## 275   Clay_Center Freeman          45.342     8.1452
## 548       Grant_D Freeman          87.910    16.0782
## 821       Lincoln Freeman          68.734    14.0865
## 1094   Lincoln_IM Freeman          98.328    18.4494
## 1367       McCook Freeman          93.007    24.5526
## 1640 North_Platte Freeman          81.416    15.5879
## 1913       Sidney Freeman          74.637    17.8141

##      x.value y.value   dfa1      dfa2     dfa3      dfa4
## 2     7.0615 20.7622 13.882 -0.043189  1.35776 -0.212301
## 275   5.4564  8.1452 10.727 -1.451306 -1.56976 -0.216763
## 548   7.6214 16.0782 14.983  0.538594  0.81504 -0.262831
## 821   8.6726 14.0865 17.050 -1.885581 -1.36930  0.293463
## 1094  7.7067 18.4494 15.151 -1.272096  2.67218  0.170264
## 1367  6.9399 24.5526 13.643  2.035254  1.00359  0.054865
## 1640  6.2613 15.5879 12.309  2.209053 -1.37896  0.244240
## 1913  6.2892 17.8141 12.364  0.368800 -2.27109 -0.172078
```

The output data frame `$fa.gen` shown above presents several elements of information about this genotype and is the one used to generate the previous plot (see columns `x.value` and `y.value`). Besides the predicted and BLUP values for this genotype we have the differentials (added values) that a given loading explains on the response variable selected for the genotype of interest (these are `dfa1`, `dfa2`, `dfa3` and `dfa4`).

In the composite figure shown below, we have selected two genotypes `Freeman` and `NE16401`. Here, we can observe the different response patterns from each of the genotypes. For `Freeman` (top row) clearly larger BLUP values are observed as the loadings of each of the factors increases. In contrast, for `NE16401` it can be noticed an almost complete insensitivity to the loadings of the first factor and some mild negative sensitivity to the second factor. However, note that the ranges of BLUP values between these two genotypes differs considerably.

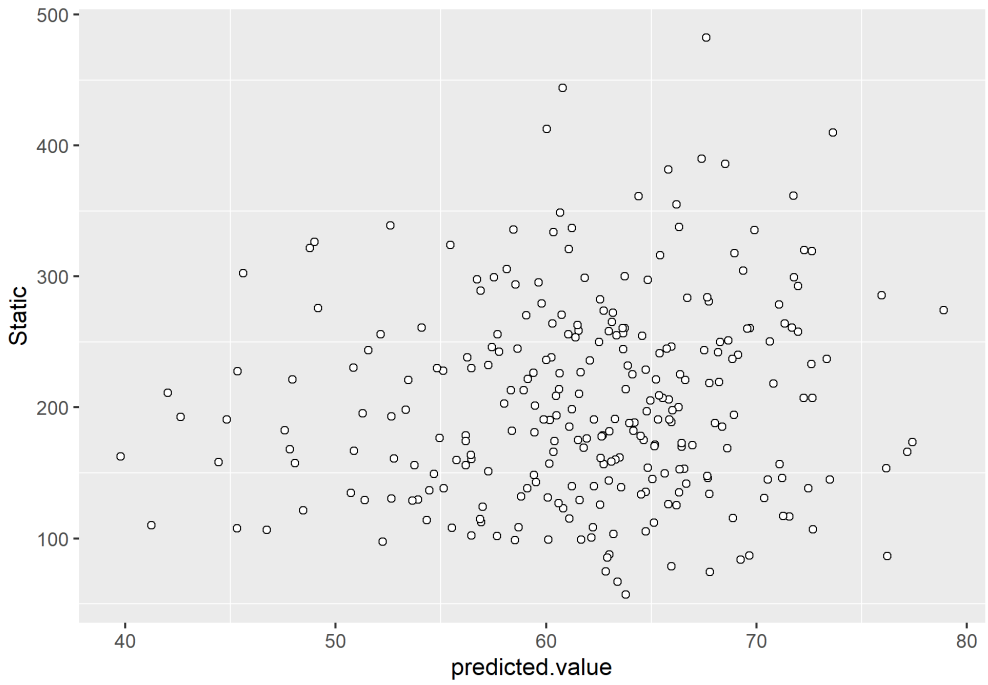



## 5.4 Enhancing graphical output from `ASRtriala`

Most plots generated by `ASRtriala` have a restricted set of options and in some cases we might like to modify them or add additional attributes. In `ASRtriala` all plots were produced using the package `ggplot2`, which is a very flexible library and, as shown below, allows for further modifications outside of their function. We illustrate this in the following example.

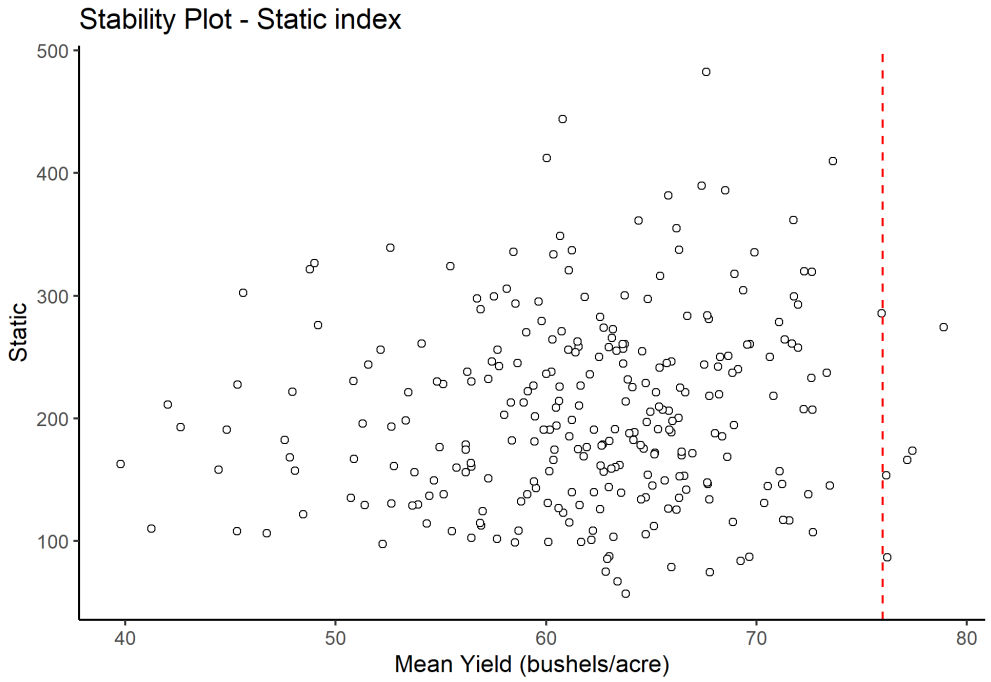
We will be using the stability plot generated earlier by the function `stability()`. The code below presents the original figure with some small modifications.

```
stab.index <- stability(data = met.model.fa4$predictions,
    trial = "location", gen = "gen", resp = "predicted.value",
    method = "static", best = "min", plot = TRUE, top = FALSE,
    bottom = FALSE, percentage = 2)
stab.index$stability.plot
```




Now we will like to alter this plot. For example, we want to add a vertical line cutting the axis on a prediction value of 76 and a title. This is all shown below.

```
stab.index$stability.plot + ggplot2::theme_classic() +
    ggplot2::geom_vline(xintercept = 76, linetype = "dashed",
        color = "red") + ggplot2::ggtitle("Stability Plot - Static index") +
    ggplot2::xlab("Mean Yield (bushels/acre)")
```




To make further enhancement on any of these plots we recommend reading the help associated with the package `ggplot2`.

# Bibliography

Belamkar, V., M. J. Guttieri, W. Hussain, D. Jarquin, I. El-Basyoni, J. Poland, A. J. Lorenz, and P. S. Baenziger. 2018. Genomic selection in preliminary yield trials in a winter wheat breeding program. *G3 Genes, Genomes, Genetics*. 8(8):2735–2747.

Butler, D. G., B. R. Cullis, A. R. Gilmour, B. J. Gogel, and R. Thompson. 2017. *ASReml-r reference manual version 4*. VSN International Ltd, Hemel Hempstead, HP1 1ES, UK.

Gezan, S. A., T. L. White, and D. A. Huber. 2010. Accounting for spatial variability in breeding trials: A simulation study. *Agronomy Journal*. 102(6):1562–1571.

Gogel, B. J., A. Smith, and B. R. Cullis. 2018. Comparison of a one-and two-stage mixed model analysis of australia's national variety trial southern region wheat data. *Euphytica*. 214(2):1–21.

Oliveira, I. C. M., J. H. S. Guilhen, P. C. O. Ribeiro, S. A. Gezan, R. E. Schaffert, M. L. F. Simeone, C. M. B. Damasceno, et al. 2020. Genotype-by-environment interaction and yield stability analysis of biomass sorghum hybrids using factor analytic models and environmental covariates. *Field Crops Research*. 257.

Smith, A., B. R. Cullis, and A. R. Gilmour. 2001a. The analysis of crop variety evaluation data in Australia. *Australian & New Zealand Journal of Statistics*. 43(2):129–145.

Smith, A., B. R. Cullis, and R. Thompson. 2001b. Analysing variety by environment data using multiplicative mixed models and adjustments for spatial field trend. *Biometrics*. 57(4):1138–1147.