GIS-FA: an approach to integrating thematic maps, factor-analytic, and envirotyping for cultivar targeting

Theor Appl Genet (2024). https://doi.org/10.1007/s00122-017-3033-y

Saulo F. S. Chaves, Mauricio S. Araujo, Matheus D. Krause, Kaio O. G. Dias

Contents

Introduction

This tutorial describes each step for performing the GIS-FA analysis. The pipeline was divided into four stages: 1- Phenotypic data; 2- Environmental data; 3- Prediction via PLS; and 4- Visualization using maps. The packages we will need are listed below.

```r
# For data management, and building and customizing plots:
library(tidyverse)
library(gghighlight)
library(ggrepel)
library(ggpubr)
library(ggspatial)

# For building Heatmaps:
## this is the only package that is not from CRAN,
## so a previous step may be necessary for installing it:
if(!require("BiocManager", quietly = TRUE)){
  install.packages("BiocManager")}
library(ComplexHeatmap)
library(circlize)

# For fitting the factor analytic mixed model and extracting its outputs straightforwadly
library(asreml)
source("https://raw.githubusercontent.com/saulo-chaves/May_b_useful/main/fa.outs.R")

# For using the partial least square method
require(pls)

# For managing rasters, shapefiles and other spatial data
library(raster)
library(sf)
library(spatstat)
library(geodata)
```

```r
# For parallelization
library(parallel)
library(doParallel)
library(foreach)
```

We will also use the `EnvRtype` package. However, there seems to be a conflict between `EnvRtype` and `asreml`, so that `asreml::asreml()` does not work when `EnvRtype` is loaded.

We express our gratitude to the developers of all packages used in this pipeline. The credits are properly given at the References section.

We recommend the creation of folders in the working directory to store the necessary data to perform the analyses. In this pipeline, there are four subfolders: `data` stores the phenotypic data, `rasters` stores some rasters that will be used to extract environmental data, `shapefiles` stores the shapefiles from where we will obtain coordinates and delimit the prediction area, and `saves` stores some `.RDA` and `.RDS` files that will be built throughout the document.

Section 1: Phenotypic data

Data description and visualization

It is important to know your data. The dataset used as example refers to soybean VCU (value of cultivation and use) trials. It has 182 genotypes and 49 environments laid out in randomized complete blocks with three blocks. Let's load the dataset into R memory and take a look in its structure:

```r
data = read.csv("data/data_soy.csv")
str(data)
```

```
## 'data.frame':    6456 obs. of  10 variables:
##  $ env  : chr  "E01" "E01" "E01" "E01" ...
##  $ plant: chr  "2019-11-21" "2019-11-21" "2019-11-21" "2019-11-21" ...
##  $ harv : chr  "2020-03-20" "2020-03-20" "2020-03-20" "2020-03-20" ...
##  $ lat  : num  -22.2 -22.2 -22.2 -22.2 -22.2 ...
##  $ lon  : num  -52.7 -52.7 -52.7 -52.7 -52.7 ...
##  $ block: int  1 2 3 1 2 3 1 2 3 1 ...
##  $ row  : int  32 13 13 6 18 18 27 8 8 36 ...
##  $ col  : int  1 2 3 1 2 3 1 2 3 1 ...
##  $ gen  : chr  "G001" "G001" "G001" "G002" ...
##  $ yield: num  5525 4380 4795 5860 5185 ...
```

The environments are distinguished in the column `env` (E01, E02, . . . , E49). `plant`, `harv`, `lat` and `lon` contain the planting date, harvesting date, latitude and longitude of each environment, respectively. `block`, `row` and `col` provide the information of the position (block, row and column, respectively) of each genotype (`gen`, G001, G002, . . . , G182). Finally, `yield` is the grain yield, the trait we are assessing. Note that `env` and `gen` are string vectors, and `block`, `row` and `col` are numeric vectors. To fit the model, we have to transform these columns into factors, so the model can treat them as such:

```r
data = transform(data,
                 gen = factor(gen),
                 block = factor(block),
                 env = factor(env),
                 row = factor(row),
                 col = factor(col))
```

After defining the factors, we store the number and name of genotypes and environments into objects. This will be useful latter:

```r
num.env = nlevels(data$env)
num.gen = nlevels(data$gen)
name.env = levels(data$env)
name.gen = levels(data$gen)
```

Factor analytic (FA) mixed models

This is the FA (Piepho, 1997; Smith et al., 2001) part of the GIS-FA method.

Selection of the best-fit FA model

The best-fit FA models are selected considering the proportion of genetic covariance explained by k factors. An ad-hoc threshold of 75% for the genetic covariance was adopted. We assessed the explicative power of the models by using the Average Semivariances Ratio (ASR) (Chaves, Alves, et al., 2023; Piepho, 2019):

$$ASR = \frac{\frac{2}{J(J-1)} \sum_{j=1}^{J-1} \sum_{j'=j+1}^{J} \frac{1}{2}(\sum_{k=1}^{K} \hat{\lambda}_{k_j}^{\star 2} d_k + \sum_{k=1}^{K} \hat{\lambda}_{k_{j'}}^{\star 2} d_k) - \sum_{k=1}^{K} \hat{\lambda}_{k_j}^{\star} \hat{\lambda}_{k_{j'}}^{\star} d_k}{\frac{2}{J(J-1)} \sum_{j=1}^{J-1} \sum_{j'=j+1}^{J} \frac{1}{2}[(\sum_{k=1}^{K} \lambda_{k_j}^{\star 2} d_k + \hat{\psi}_j) + (\sum_{k=1}^{K} \hat{\lambda}_{k_{j'}}^{\star 2} d_k + \hat{\psi}_{j'})] - \sum_{k=1}^{K} \hat{\lambda}_{k_j}^{\star} \hat{\lambda}_{k_{j'}}^{\star} d_k} \times 100$$

where d_k is the k^{th} element of the diagonal of $\mathbf{D}$, a $K \times K$ symmetric positive (semi)-definite factor score variance matrix; $\hat{\lambda}_{k_j}$ is the rotated loading of the $\mathtt{j}^{th}$ ($\mathrm{j} = 1, 2, \ldots, \mathrm{J}$) environment in the $\mathtt{k}^{th}$ factor ($\mathrm{k} = 1, 2, \ldots, \mathrm{K}$), and $\hat{\psi}_j$ is the specific variance of the $\mathtt{j}^{th}$ environment.

First, let's fit the models. You will notice that in the paper, we fitted a model adjusting spatial trends in multi-environment in a single step, and considered heterogeneous variances. Nevertheless, for the sake of simplicity, we opted to show a regular FA model, for the consideration of spatial trends in linear mixed models is not the aim of this tutorial. You can find the fitted model in this github page.

```r
asreml.options(maxit = 100, workspace = '1gb', pworkspace = '2gb')
# FA1
mod1 = asreml(fixed = yield ~ block:env + env,
              random = ~ gen:fa(env, 1),
              data = data,
              na.action = na.method(x="include", y = "include"), maxit = 50)

# FA2
mod2 = asreml(fixed = yield ~ block:env + env,
              random = ~ gen:fa(env, 2),
              data = data,
              na.action = na.method(x="include", y = "include"), maxit = 50)

# FA3
mod3 = asreml(fixed = yield ~ block:env + env,
              random = ~ gen:fa(env, 3),
              data = data,
              na.action = na.method(x="include", y = "include"), maxit = 100)


# FA4
mod4 = asreml(fixed = yield ~ block:env + env,
              random = ~ gen:fa(env, 4),
              data = data,
              na.action = na.method(x="include", y = "include"), maxit = 100)
```

Note that we changed some options before running the models. Usually, FA models need more iterations to converge. The memory usage will depend on the dataset.

The results showed below are from the model we fitted in the paper.

```r
load(file = 'saves/FAMM_soy.RData')
```

Now, we are going to compute the objective criteria to select the best-fit FA model:

```r
fa.models = list(mod1, mod2, mod3, mod4)
fa.res = lapply(fa.models, function(x) fa.outs(x, name.env = "env",
                                               name.gen = "gen"))
```

```r
diagnos = do.call(rbind,lapply(fa.res, function(x) x$diagnostics))
rownames(diagnos) = c("mod1", "mod2", "mod3", "mod4"); diagnos[,'ASVR']
```

```
##   mod1   mod2   mod3   mod4
## 22.578 54.083 69.084 76.627
```

`mod4` with $K = 4$ reach our criterion regarding explicative power, and therefore it is the best-fit model for further analyses. The remaining models can be removed from the workspace, so we can save some memory.

```r
fa4 = fa.res[[4]]
rm(mod1, mod2, mod3, fa.models, fa.res); gc()
```

FA(4)

The best-fit model `mod4` was previously saved in an object called `fa4`. Let's take a look at the needed parameters for downstream analyses. All parameters presented in this section were computed using the `fa.outs()` function, available here.

- Rotated factor loadings:

`asreml` does not rotate factors loading by default unless you set `rotate.fa = TRUE`. We personally like to rotate them manually using the singular value decomposition (SVD) method, as suggested by Smith et al. (2021). See the main manuscript for more details about the rotation process.

```r
head(fa4$rot.loads)
```

```
##            fa1          fa2         fa3          fa4
## E01 0.19332304  0.080907355 -0.20263846  0.071132347
## E02 0.06860044 -0.199689358 -0.20180917 -0.001911237
## E03 0.13790138  0.131566541 -0.09134606  0.023812729
## E04 0.02115538  0.041711945 -0.12819687  0.217929631
## E05 0.16426246  0.104474372 -0.02068189 -0.153304493
## E06 0.15198068 -0.003096557 -0.02703350 -0.160782802
```

- Matrix of genetic covariances between environments:

Given by:

$$\boldsymbol{\Sigma}_g = (\hat{\boldsymbol{\Lambda}}^{\star}\mathbf{D}\hat{\boldsymbol{\Lambda}}^{\star'} + \hat{\boldsymbol{\Psi}}) \otimes \mathbf{I}_V$$

where $\boldsymbol{\Sigma}_g$ is the matrix of genetic covariances (49×49 in our example), $\hat{\boldsymbol{\Lambda}}^{\star}$ is the matrix of rotated loadings (49×4, in our example), $\mathbf{D}$ was previously defined, and $\hat{\Psi}$ is a diagonal matrix (49×49) of specific variances.

```r
fa4$Gvcov[1:5, 1:5]
```

```
##           E01       E02       E03       E04        E05
## E01 198687.34  59471.13 123241.82 43676.588 123971.962
## E02  59471.13 151421.29  18238.38 14260.969  20868.986
## E03 123241.82  18238.38 114447.89 27316.533  98004.270
## E04  43676.59  14260.97  27316.53 57039.095   5674.497
## E05 123971.96  20868.99  98004.27  5674.497 128023.205
```

- Matrix of genetic correlations between environments:

Given by Cullis et al. (2010):

$$\boldsymbol{\Upsilon} = \boldsymbol{\Delta}\boldsymbol{\Sigma}_g\boldsymbol{\Delta}$$

where $\boldsymbol{\Upsilon}$ is the matrix of genetic correlations (49×49 in our example), and $\boldsymbol{\Delta}$ is a diagonal matrix whose elements are the inverse of the square roots of the diagonal elements of $\boldsymbol{\Sigma}_g$.

```
fa4$Gcor[1:5, 1:5]
```

```
##           E01       E02       E03        E04        E05
## E01 1.0000000 0.3428685 0.8172767 0.41027711 0.77730959
## E02 0.3428685 1.0000000 0.1385444 0.15345084 0.14988685
## E03 0.8172767 0.1385444 1.0000000 0.33809237 0.80964895
## E04 0.4102771 0.1534508 0.3380924 1.00000000 0.06640438
## E05 0.7773096 0.1498868 0.8096489 0.06640438 1.00000000
```

- Proportion of variance explained by each factor in each environment:

Given by Smith et al. (2015):

$$v_{k_j} = \frac{\hat{\lambda}^{\star 2}_{k_j} d_k}{\sum_{k=1}^{K} \hat{\lambda}^{\star 2}_{k_j} d_k + \hat{\psi}_j} \times 100$$

where v_{k_j} is the proportion of variance of the $\mathtt{j}^{th}$ environment explained by the $\mathtt{k}^{th}$ factor.

```
fa4$expvar_j[,1:5]
```

```
##           E01          E02        E03       E04        E05
## fa1 68.714769 18.893226505 73.0689444  4.679451 81.5810877
## fa2  3.694773 49.146333016 20.4181042  5.584742 10.1312094
## fa3 14.756477 31.958782266  6.2665899 33.586458  0.2527839
## fa4  1.051904  0.001658213  0.2463615 56.149350  8.0349190
```

- Rotated scores:

In the function, we opted to show the scores in a matrix form, rather than a vector form, for better visualization.

```
head(fa4$rot.scores)
```

```
##             fa1        fa2        fa3       fa4
## C006  -962.4416 1111.87158 -1038.4198  287.9607
## C049  1018.8383  390.53328  -884.6681 -212.1031
## C054  1894.1170 -444.45912  -867.8695  477.0996
## C084  2024.6305  -25.40594   631.5421  169.3687
## G001 -1239.7424 -823.04425  -226.0714  234.6343
## G002   741.7032 -395.93031  -372.9204  117.0934
```

- Conditional (accounting for the specific environmental effects, and summed to the intercept) and marginal empirical BLUPs:

The conditional EBLUPs estimated by the function is given by the following equation:

$$\mathbf{g} = (\mathbf{\Lambda}^{\star} \otimes \mathbf{I}_V)\tilde{\mathbf{f}}^{\star} + \tilde{\boldsymbol{\delta}} + \mu$$

where $\tilde{\mathbf{f}}^{\star}$ is the vector of rotated scores, $\mathbf{I}_g$ is an identity matrix whose dimension is the number of genotypes, $\boldsymbol{\delta}$ is the vector of specific effect (lack of fit of the multiple regression), and μ is the intercept of the model. The marginal EBLUPs are estimated using only the common effects, i.e., $\mathbf{g} = (\mathbf{\Lambda}^{\star} \otimes \mathbf{I}_V)\tilde{\mathbf{f}}^{\star}$.

```r
head(fa4$blups)
```

```
##   env  gen conditional std.error  marginal
## 1 E01 C006    5347.383  409.6906  134.8035
## 2 E01 C049    5566.562  110.7297  392.7423
## 3 E01 C054    5696.263  108.2500  540.0174
## 4 E01 C084    5494.399  174.7449  273.4251
## 5 E01 G001    4946.074  119.5839 -243.7602
## 6 E01 G002    5449.179  108.2943  195.2518
```

- Generalized heritabilities

Given by Cullis et al. (2006):

$$H_j^2 = \frac{\overline{V}(\Delta)}{2 \times \sigma^2_{g_j}}$$

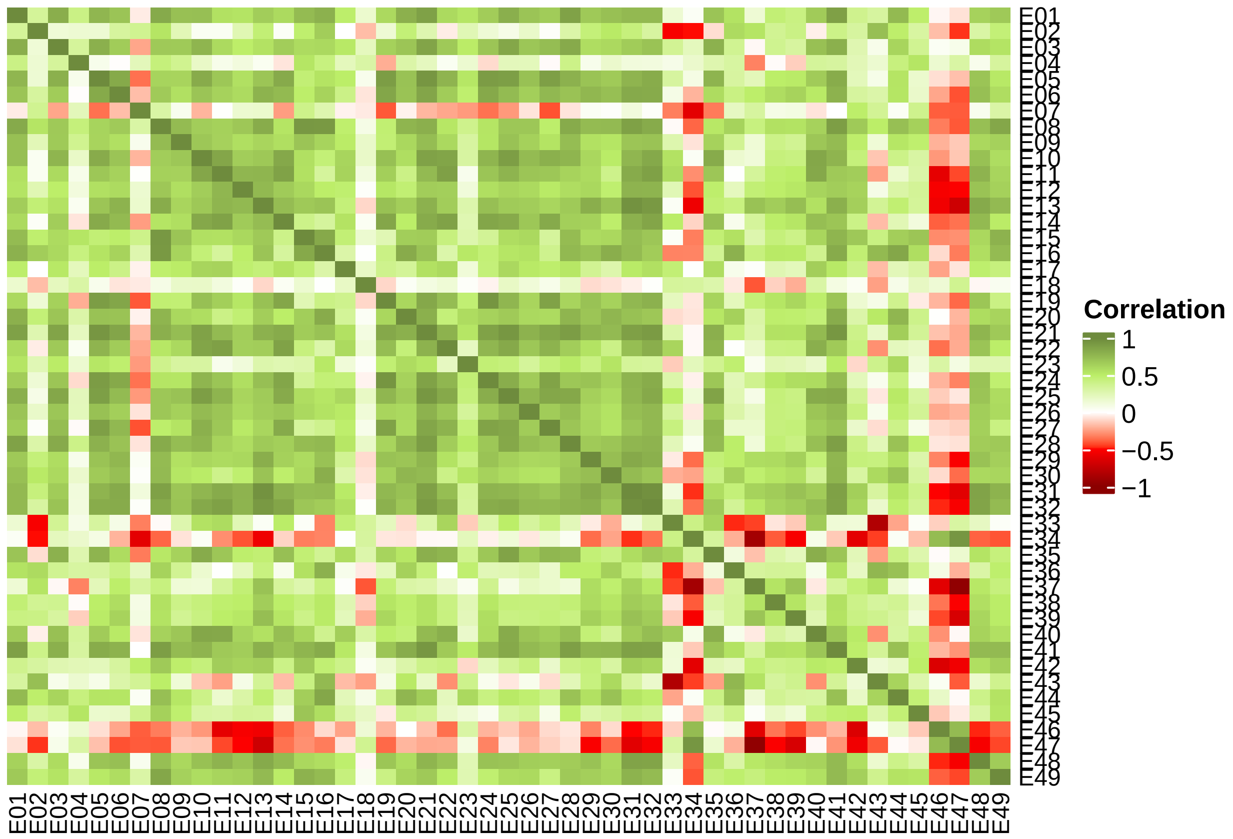
where H_j^2 is the generalized heritability of the jth environment, $\overline{V}(\Delta)$ is the mean pairwise prediction error variance, and $\sigma^2_{g_j}$ is the genetic variance of the jth environment.

```r
fa4$H2
```

```
##       E01       E02       E03       E04       E05       E06       E07       E08 
## 0.6584196 0.4414755 0.6183688 0.3895006 0.6818437 0.6410480 0.4178423 0.7144013 
##       E09       E10       E11       E12       E13       E14       E15       E16 
## 0.5887136 0.7072477 0.7497581 0.5907105 0.7423759 0.6994666 0.6241437 0.6780266 
##       E17       E18       E19       E20       E21       E22       E23       E24 
## 0.4124776 0.3154184 0.6548304 0.5778095 0.7506231 0.6829289 0.4064819 0.6510548 
##       E25       E26       E27       E28       E29       E30       E31       E32 
## 0.7164063 0.5470520 0.6212530 0.6672332 0.6212460 0.5927928 0.7752457 0.7648092 
##       E33       E34       E35       E36       E37       E38       E39       E40 
## 0.6239515 0.6480647 0.6474860 0.5036621 0.6730913 0.4259277 0.5470619 0.6701653 
##       E41       E42       E43       E44       E45       E46       E47       E48 
## 0.7218086 0.5268644 0.6278025 0.5878723 0.4188775 0.5678912 0.6744804 0.6809696 
##       E49 
## 0.6000386
```

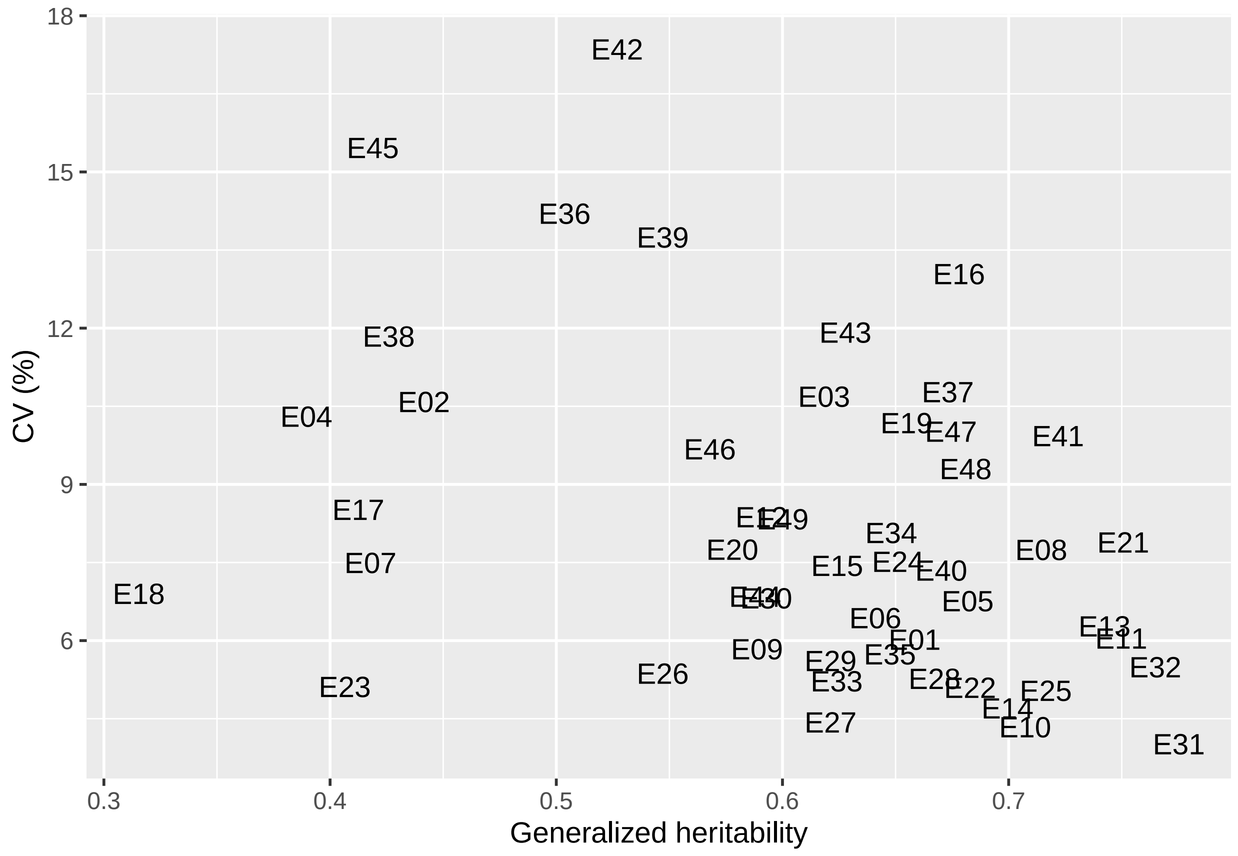
Plots can provide additional illustration of the results. For example, a heatmap of the genetic correlation between environment can be computed as follows:

```
col_fun = circlize::colorRamp2(c(-1,-.5,0,.5,1), c("red4","red1","white",
                                                  "darkolivegreen2","darkolivegreen4"))
ComplexHeatmap::Heatmap(
  fa4$Gcor,
  col=col_fun,
  cluster_rows = F,
  cluster_columns = F,
  show_heatmap_legend = T,
  show_row_names = T,
  show_column_names = T,
  heatmap_legend_param = list(title = "Correlation"),
  row_names_gp = gpar(fontsize = 9),
  column_names_gp = gpar(fontsize = 9)
  )
```



Other useful information can be extracted from the model, such as the generalized heritabilities and the experimental coefficients of variation:

```r
data.frame(
  H2 = fa4$H2,
  CV = (sqrt(summary(mod4)$varcomp[grep('!R', rownames(summary(mod4)$varcomp)),1])/
  tapply(data$yield, data$env, mean, na.rm = T)) * 100
) |>
  tibble::rownames_to_column("env") |>
  ggplot(aes(x = H2, y = CV)) +
  geom_text(aes(label = env)) +
  labs(x = "Generalized heritability", y = "CV (%)")
```



Factor Analytic Selection Tools

We can now employ the Factor Analytic Selection Tools (FAST) to estimate overall performance (OP) (Smith & Cullis, 2018; Stefanova & Buirchell, 2010) and root mean square deviation (RMSD) (Smith & Cullis, 2018), which are a measure of performance and stability, respectively. We can associate these metrics with the reliability (Mrode, 2014) of each selection candidate and build a selection index to facilitate the selection process.

The overall performance is given by:

$$OP_i = \frac{1}{J}\sum_{j=1}^{J} \lambda^{\star}_{1_j} f^{\star}_{1_j}$$

where J is the number of environment.

The root mean square deviation is given by:

$$RMSD_i = \sqrt{\frac{1}{J}\sum_{j=1}^{J}(\lambda^\star_{2_j} f^\star_{2_j} + \lambda^\star_{3_j} f^\star_{3_j} + \lambda^\star_{4_j} f^\star_{4_j})^2}$$

The reliability is given by:

$$r = 1 - \frac{PEV_i}{\overline{\sigma^2_g}}$$

where PEV_i is the prediction error variance of the ith genotype, and $\overline{\sigma^2_g}$ is the genetic variance across environments.

When building the index, you can prioritize what you think is more important for reaching your goals (performance over stability, or the other way around). For example, if we define performance is two times more important than stability, the index will have a weight of 2 for OP, and a weight of 1 for RMSD:

$$\texttt{Index}_i = \left[2 \times \frac{OP_i - \overline{OP}}{\sqrt{V(OP)}}\right] - \frac{RMSD_i - \overline{RMSD}}{\sqrt{V(RMSD)}}$$

The `R` code is presented below:

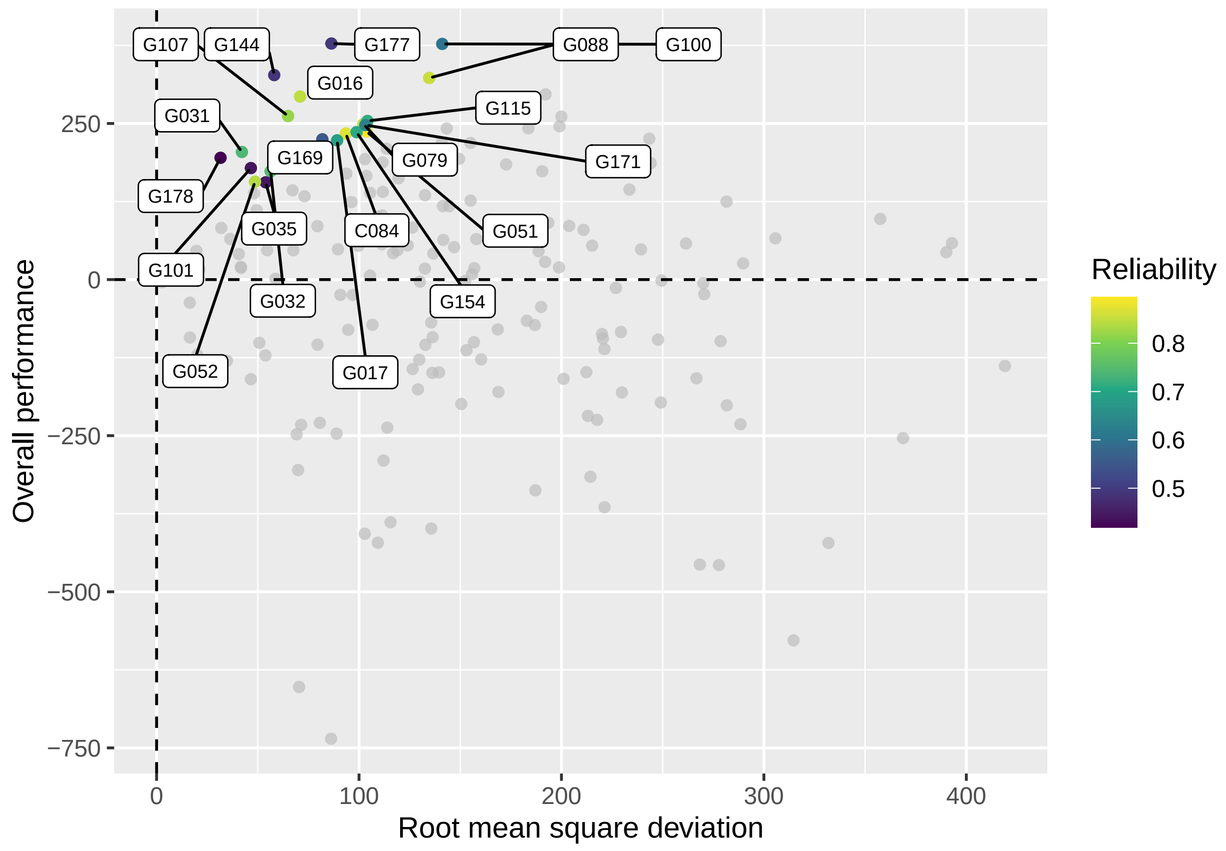
```r
# Reliability

rel = fa4$blups |>
  dplyr::reframe(rel = 1-(mean(std.error^2)/mean(diag(fa4$Gvcov))), .by = gen)

FAST = data.frame(
  env = fa4$blups$env,
  gen = fa4$blups$gen,
  RMSD = (fa4$blups$marginal - (kronecker(fa4$rot.loads[,1], diag(num.gen)) %*%
                     fa4$rot.scores[,1]))^2
) |> dplyr::reframe(RMSD = sqrt(mean(RMSD)), .by = gen) |>
  dplyr::mutate(OP = mean(fa4$rot.loads[,1]) * fa4$rot.scores[,1],
         index = 2*((OP-mean(OP))/sqrt(var(OP))) - ((RMSD-mean(RMSD))/sqrt(var(RMSD))),
         rel = rel$rel)
```

Now that we built our data frame, we can make the FAST plot. Here, we will highlight only the top 20 (11%) selection candidates:

```r
FAST |> ggplot(aes(x = RMSD, y = OP)) +
  geom_point(aes(color = rel)) +
  geom_vline(xintercept = 0, linetype = 'dashed') +
  geom_hline(yintercept = 0, linetype = "dashed") +
  gghighlight(gen %in% FAST[order(FAST$index, decreasing = T),'gen'][1:20]) +
  geom_label_repel(aes(label = gen), size = 2.5, box.padding = .5,
                   max.overlaps = 30) +
  labs(colour = "Reliability") +
  scale_colour_viridis_c() +
  labs(x = "Root mean square deviation", y = "Overall performance")
```



Do not forget to check if most of the loadings in the first factor are positive. If this is not the case, FAST is not applicable. Smith et al. (2021) and Chaves, Evangelista, et al. (2023) provided some options for when this happens.

We will save the `fa4` list for later. The other objects will be cleared from the memory:

```
save(fa4, file = "saves/FA4.RDA")
rm(list=ls())
```

Section 2: Environmental data

Download and compilation

There are several on-line platforms that have freely available environmental data such as soilgrids, nasapower, and GAEZ. We will only show two ways of downloading environmental data using R.

Also, keep in mind there are two separate procedures from now on: obtaining environmental data for the tested environments, and obtaining the environmental data for the geographical coordinates (i.e., points) at which you want to do the prediction (untested environments).

Obtaining environmental covariates for the tested environments

First, let's load again the phenotypic data:

```
data = read.csv("data/data_soy.csv")
```

The function `get_weather()` from the package `EnvRtype` downloads daily environmental data from the nasapower database according to the dates of planting and harvesting. After obtaining the weather data, we can use other functions such as `param_atmospheric()`, `param_temperature()`, and `param_radiation()` to process it into derived environmental covariates. The arguments `Tbase1` and `Tbase2` in `param_temperature()` below refers to the minimum and maximum cardinal temperature, respectively. The parameters `Topt1` and `Topt2` are the lower and upper bound temperature for phenological development, respectively. These values are species-dependent. See Costa-Neto et al. (2021) for more details.

```
# Summarizing trial information
coords = unique(data[,c('env', 'lat', 'lon', 'plant', 'harv')])

# Downloading weather data
covamb = EnvRtype::get_weather(env.id = coords$env,
                               lat = coords$lat,
                               lon = coords$lon,
                               start.day = coords$plant,
                               end.day = coords$harv)

# Processing the downloaded temperature environmental covariates
covamb = EnvRtype::param_temperature(env.data = covamb, Tbase1 = 8, Tbase2 = 45,
                                     Topt1 = 30, Topt2 = 35, merge = T)

# Processing the downloaded radiation environmental covariates
covamb = EnvRtype::param_radiation(env.data = covamb, merge = T)

# Processing the downloaded atmospheric environmental covariates
covamb = EnvRtype::param_atmospheric(env.data = covamb, merge = T)
```

Note we are overwriting the object `covamb` when using the processing functions. This will do no harm since these functions merge (`merge = T`) the columns with the new, derived environmental covariates into the original data.

We can now obtain soil-related environmental covariates from the soilgrids platform using the function soil_word() from `geodata`. The data will be downloaded as rasters (.tif) in a path that must be specified (in our case, 'Rasters').

```r
geodata::soil_world(var = 'bdod', depth = 15, stat = 'mean', path = 'Rasters')
geodata::soil_world(var = 'clay', depth = 15, stat = 'mean', path = 'Rasters')
geodata::soil_world(var = 'nitrogen', depth = 15, stat = 'mean', path = 'Rasters')
geodata::soil_world(var = 'phh2o', depth = 15, stat = 'mean', path = 'Rasters')
geodata::soil_world(var = 'sand', depth = 15, stat = 'mean', path = 'Rasters')
geodata::soil_world(var = 'silt', depth = 15, stat = 'mean', path = 'Rasters')
geodata::soil_world(var = 'soc', depth = 15, stat = 'mean', path = 'Rasters')
geodata::soil_world(var = 'cfvo', depth = 15, stat = 'mean', path = 'Rasters')
geodata::soil_world(var = 'ocd', depth = 15, stat = 'mean', path = 'Rasters')
```

Data from elevation (altitude) can be obtained from `geodata`:

```r
geodata::elevation_30s(country = 'BRA', path = 'Rasters')
```

Note that you just have to download the rasters once. We also manually downloaded rasters from the platform of Lembrechts et al. (2022) regarding the soil's mean temperature, isothermality, temperature seasonality and mean diurnal range. In the platform, there is a raster of soil's mean temperature for each month of the year. Nevertheless, since the trials were conducted between October and March (Brazil's soybean first cropping season), we only downloaded raster corresponding to months between this range.

We can now use the `extract_GIS()` from `EnvRtype` to extract the data for the corresponding coordinates:

```r
rastlist = list.files(path = 'Rasters/soil_world', pattern = '.tif', full.names = T)
rastlist = c(rastlist, list.files(path = 'Rasters', pattern = ".tif", full.names = T))
namerasts = c('bdod', 'cfvo', 'clay', 'nit', 'ocd', 'phh2o', 'sand', 'mdr', 'sts',
              'iso','silt', 'soc', 'st1', 'st10', 'st11', 'st12', 'st2', 'st3', 'alt')
soil = coords
for(i in 1:length(rastlist)){
  raster_file <- raster::raster(rastlist[i])
  soil <- dplyr::left_join(
    soil,
    EnvRtype::extract_GIS(covraster = raster_file, Latitude = 'lat',
                          Longitude = 'lon', name.out = namerasts[i],
                          env.data = coords, env.id = 'env')
    )
}
```

A last step is to take the mean of the soil temperature across the months in which the trial was conducted:

```r
soil$soiltemp = apply(as.matrix(soil[,c('st3', 'st2', 'st12', 'st11',
                                         'st10', 'st1')]), 1, mean)
soil = soil[,-which(colnames(soil) %in% c('st3', 'st2', 'st12', 'st11',
                                           'st10', 'st1'))]
```

Then, we just have to merge the data frame that contain climatic the environmental covariates with the one that has the soil environmental covariates:

```
covamb = left_join(covamb, soil[,-(2:5)], by = c('env'))
```

For saving memory, we save the `covamb` object as a `.RDS` data in our "saves" subdirectory and clear the internal memory:

```
saveRDS(covamb, file = "saves/covamb.RDS")
rm(list=ls())
```

Obtaining environmental covariates for the untested environments

Up to this point, we only obtained environmental data for the observed target population of environments (TPE). This section will obtain the environmental data needed to predict the performance of the selection candidates in untested environments from several geographic coordinates.

The first step is to obtain the coordinates, which can be done from shapefiles freely available from several platforms. For example, in Brazil, the Institute of Geography and Statistics (IBGE in the Portuguese acronym) provides shapefiles with several details regarding administrative boundaries (from municipalities to the whole country). These are the shapefiles we used for obtaining the coordinates. Another useful way of obtaining shapefiles is using the function `gadm()`, from the `geodata` package.

It is important to consider how you want to sample the breeding region. Here, we opted to sample coordinates within each municipality to represent the whole area well. Thus, we have to extract the coordinates of each municipality. So, first, the shapefile with the administrative boundaries of each municipality must be loaded as follows:

```
br_mun = raster::shapefile('shapefiles/municipality/BR_Municipios_2021.shp')
br_mun.sf = sf::st_as_sf(br_mun)
```

The object `br_mun.sf` will be used soon. Let's give an example first. Suppose you are interested in predicting the phenotypic performance in the city of Belém, Pará State. The steps would be the following:

1- Extract the coordinates of the whole country using `coordinates()` from the `raster` package. You can do this using the elevation raster, for example.

```
br_coord = as.data.frame(
  raster::coordinates(raster::raster('rasters/BRA_elv_msk.tif'))
  )
```

2- Filter the shapefiles to the appropriate State (since we obtained the shapefiles from a Brazilian source, the words are in Portuguese).

```
mun = br_mun[br_mun$SIGLA == 'PA', ]
mun.sf = br_mun.sf[br_mun.sf$SIGLA == "PA", ]
```

3- Filter to the desired municipality and determine a boundary box based on the municipality's boundaries. This boundary box is filled with coordinates, but not all of them correspond to the desired municipality.

```
coord = dplyr::filter(br_coord, x > mun[mun$NM_MUN == 'Belém',]@bbox['x','min'] &
                            x < mun[mun$NM_MUN == 'Belém',]@bbox['x','max'] &
                            y > mun[mun$NM_MUN == 'Belém',]@bbox['y','min'] &
                            y < mun[mun$NM_MUN == 'Belém',]@bbox['y','max'])
```

4- Extract the geometry of the desired municipality.

```
pts = unique(do.call(rbind, mun.sf[mun.sf$NM_MUN == "Belém",]$geometry[[1]][[1]]))
pts = as.data.frame(pts)
colnames(pts) = c('x','y')
```

5- Using the `point.in.polygon()` of the `sp` package, determine which points in the boundary box are within the municipality's geometry.

```
coord$filt = sp::point.in.polygon(coord$x, coord$y, pts$x, pts$y)
```

6- Exclude the points that are not within the municipality

```
coord = coord[which(coord$filt == 1),]
coord = subset(coord, select = -filt)
```

And voilà:

```
head(coord)
```

```
##            x         y
## 19 -48.47083 -1.020833
## 57 -48.47917 -1.029167
## 58 -48.47083 -1.029167
## 59 -48.46250 -1.029167
## 60 -48.45417 -1.029167
## 61 -48.44583 -1.029167
```

We have done the same procedure for all municipalities of Mato Grosso do Sul (`'MS'`) State - which is the State where the soybean trials were conducted - using a `for` loop. The resulting data frame (`mun.xy`) has all coordinates of these municipalities.

```
mun = br_mun[br_mun$SIGLA == 'MS',]
mun.sf = br_mun.sf[br_mun.sf$SIGLA == "MS", ]
mun_coords = list()
for (i in mun$NM_MUN) {
    coord = dplyr::filter(br_coord, x > mun[mun$NM_MUN == i,]@bbox['x','min'] &
                                x < mun[mun$NM_MUN == i,]@bbox['x','max'] &
                                y > mun[mun$NM_MUN == i,]@bbox['y','min'] &
                                y < mun[mun$NM_MUN == i,]@bbox['y','max'])

    pts = unique(do.call(rbind, mun.sf[mun.sf$NM_MUN == i,]$geometry[[1]][[1]]))
    pts = as.data.frame(pts)
    colnames(pts) = c('x','y')

    coord$filt = sp::point.in.polygon(coord$x, coord$y, pts$x, pts$y)
    if(all(coord$filt == 0)){
      cat('I was at', i, '-', 'MS', ', but found no points \n')
      next
    }
    coord = coord[which(coord$filt == 1),]
    coord = subset(coord, select = -filt)
    coord$mun = i
    coord$est = 'MS'

    mun_coords[[i]] = coord
    rm(coord, pts)
    cat('I am at', i, '-', 'MS', '\n')
  }
mun.xy = do.call(rbind, mun_coords)
colnames(mun.xy) = c('lon', 'lat', 'mun', 'sta')
rm(mun, mun_coords)
```

The object `mun.xy` contains the geographical coordinate of 79 cities in the Mato Grosso do Sul State.

After obtaining the geographical coordinates, we will randomly sample 50 points from each municipality to perform the predictions.

```
set.seed(100) #This seed guarantees the reproducibility of the results
samp = mun.xy %>% dplyr::group_by(mun) %>% dplyr::slice_sample(n = 50)
samp$env = paste0("P", 1:nrow(samp))
```

The next step is to download the environmental covariates. We had modified the original functions from the `EnvRtype` package to download monthly instead of daily averages. Thus, before extracting the climatic covariates, let's load the modified functions into `R`'s memory:

```
source("https://raw.githubusercontent.com/saulo-chaves/May_b_useful/main/get_weather2")
```

Now, we can extract and process the covariates:

```
covamb_pred = get_weather2(env.id = samp$env[1:50], lat = samp$lat[1:50],
                           lon = samp$lon[1:50],  start.year = '2000',
                           end.year = '2021', parallel = T)

covamb_pred = EnvRtype::extract_GIS(covraster = raster('rasters/BRA_elv_msk.tif'),
                                    Latitude = 'LAT', name.out = 'ALT',
                                    Longitude = 'LON', env.data = covamb_pred,
                                    env.id = 'env')

Tfun = c(8, 45, 30, 35)
names(Tfun) = c('Tbase1', 'Tbase2', 'Topt1', 'Topt2')

covamb_pred = covamb_pred |> dplyr::select(env, ALT, LON, LAT, PARAMETER, YEAR,
                                           OCT, NOV, DEC, JAN, FEB, MAR) |>
  tidyr::pivot_longer(cols = OCT:MAR, names_to = 'MONTH') |>
  tidyr::pivot_wider(names_from = PARAMETER, values_from = value) |>
  dplyr::mutate(
    T2RANGE = T2M_MAX - T2M_MIN
  ) |>
  dplyr::mutate(
    FRUE = F.RUE.Temperature(Tbase1 = Tfun['Tbase1'], Tmed = T2M,
                             Tbase2 = Tfun['Tbase2'], Topt1 = Tfun['Topt1'],
                             Topt2 = Tfun['Topt2']),
    GDD = ((adjust_for_Tbase2(adjust_for_Tbase(T2M_MAX, Tfun['Tbase1']),
                              Tfun['Tbase2']) +
              adjust_for_Tbase2(adjust_for_Tbase(T2M_MIN, Tfun['Tbase1']),
                                Tfun['Tbase2']))/2 -
             Tfun['Tbase1']),
    VPD = ((teten(T2M_MIN) + teten(T2M_MAX)/2)) - teten(T2MDEW),
    SPV = 4098 * (0.6108 * exp((17.27 * T2M)/(T2M + 237.3)))/(T2M + 237.2)^2
  ) |>
  dplyr::mutate(
    ETP = EToPT(alfa = 1.26, Srad = ALLSKY_SFC_SW_DWN, slope = SPV,
                psyc = psyco(AtmP(elevation = ALT)))
  ) |>
  dplyr::mutate(PETP = PRECTOTCORR - ETP)
```

To extract the soil covariates, we will use the same procedure previously described, as follows:

```
rastlist = list.files(path = 'Rasters/soil_world', pattern = '.tif', full.names = T)
namerasts = c('bdod', 'cfvo', 'clay', 'nit', 'ocd', 'phh2o', 'sand', 'mdr', 'sts',
              'iso','silt', 'soc', 'st1', 'st10', 'st11', 'st12', 'st2', 'st3')
soil = data.frame(samp)
for(i in 1:length(rastlist)){
  raster_file <- raster::raster(rastlist[i])
  soil <- dplyr::left_join(
    soil,
    EnvRtype::extract_GIS(covraster = raster_file, Latitude = 'lat',
                          Longitude = 'lon', name.out = namerasts[i],
                          env.data = data.frame(samp),
                          env.id = 'env'),
    by = c("env","lon", "lat", "mun", "sta")
    )
}
soil$soiltemp = apply(as.matrix(soil[,c('st3', 'st2', 'st12', 'st11',
                                       'st10', 'st1')]), 1, mean)
soil = soil[,-which(colnames(soil) %in% c('st3', 'st2', 'st12', 'st11',
                                           'st10', 'st1'))]
```

And finally, we merge the data frames with weather and soil environmental covariates:

```
covamb_pred = merge(covamb_pred, soil)

covamb_pred = covamb_pred |> dplyr::select(-lon, -lat) |>
  dplyr::relocate(sta, mun, .before = env) |>
  dplyr::relocate(YEAR, MONTH, ALT, LON, LAT, .before = T2M)
```

The previous step were computationally demanding and required several downloads. From time to time, we recommend saving the `R` environment [e.g., using `save.image("GIS-FA.RDA)`] so the pipeline can be continued at any point without the need to compute/download all the information again.

We will save the `covamb_pred` object as a `.RDS` file and clean the memory for the next step:

```
saveRDS(covamb_pred, file = "saves/covamb_pred.RDS")
rm(list=ls())
```

Data management

Now that we have all environmental covariates downloaded for the observed and unobserved environments, let's compute some important results. First, let's load the `.RDS` files we saved from the previous step:

```
covamb = readRDS("saves/covamb.RDS")
covamb_pred = readRDS("saves/covamb_pred.RDS")
```

Now, we will build matrices that contain the mean of each environmental covariate (columns) in each environment (rows):

```
# Observed environments
matcovamb = subset(covamb, select = c(-YEAR, -MM, -DD, -DOY, -YYYYMMDD,
                                      -daysFromStart, -n, -N, -RTA))
matcovamb = matcovamb |> dplyr::reframe(across(LON:soiltemp, mean), .by = 'env')
```

```r
matcovamb = tibble::column_to_rownames(matcovamb, var = 'env')

# Unobserved environments
covamb_pred = covamb_pred |>
  dplyr::filter(YEAR != 2000 | !MONTH %in% c('JAN','FEB','MAR')) |>
  dplyr::filter(YEAR != 2021 | !MONTH %in% c('OCT','NOV','DEC')) |>
  dplyr::select(-YEAR, -MONTH) |>
  dplyr::reframe(across(ALT:soiltemp, mean), .by = c(mun, env)) |>
  dplyr::rename(T2M_RANGE = T2RANGE, PRECTOT = PRECTOTCORR)
```

Environmental similarity

The prediction will surely be more accurate in the environments where trials were conducted since we are able to observe the relationship between selection candidates and environmental features. Nevertheless, we cannot be sure about how reliable the prediction will be for the untested environments. A metric that can give us a hint about this reliability is the environmental similarity. If a given unobserved environment has similar environmental conditions to any observed environment, it is likely that this environment's prediction will be more accurate.

To calculate the environmental similarity, we must first sort the columns of the matrices that contain the environmental information for the observed and unobserved environments, in a way that they follow the same order:

```r
# Observed environments
matcovamb = matcovamb[, sort(colnames(matcovamb))]

# Yet-to-be-seen environments (sampled points)
matcovamb_pred = covamb_pred |>
  tibble::column_to_rownames('env') |>
  dplyr::select(-mun)
matcovamb_pred = matcovamb_pred[, sort(colnames(matcovamb_pred))]
matcovamb_pred = as.matrix(matcovamb_pred)

# Let's confirm both matrices have the same order of columns:
all(tolower(colnames(matcovamb_pred)) == tolower(colnames(matcovamb)))
```

```
## [1] TRUE
```

Then, the next step is calculating the Euclidean distances between the observed and unobserved environments to quantify the environmental similarity. Let $\mathbf{W}$ be a $J \times P$ matrix that contains the scaled values of P environmental covariates with the J observed environments, and let $\mathbf{\Omega}$ be the matrix with the same information but for the U unobserved environments. The Euclidean distance between an observed environment j and an unobserved environment u is given by the distances between the rows of $\mathbf{W}$ and $\mathbf{\Omega}$ that correspond to j and u, respectively:

$$D_{ju} = \sqrt{\sum_{p=1}^{P} (w_{jp} - \omega_{up})^2}$$

where w_{jp} is an entry of $\mathbf{W}$ that contains the value of the p^{th} environmental covariate for the j^{th} environment, and ω_{up} is the same for the unobserved environments.

```r
eucl = as.matrix(pdist::pdist(scale(matcovamb), scale(matcovamb_pred)))
rownames(eucl) = rownames(matcovamb)
colnames(eucl) = rownames(matcovamb_pred)
```

In the next step, we transform the distance matrix to a long format and divide it into data frames stored in a list. In total, there will be 49 data frames, each one representing an observed environment from the target population of environments (TPE).

```r
dist.env = lapply(apply(eucl, 1, function(x) reshape2::melt(x)),
       function(x) x |>
         dplyr::rename(dist = value) |>
         tibble::rownames_to_column('point') |>
         dplyr::left_join(covamb_pred[,c('env', 'LON', 'LAT')],
                   by = c('point' = 'env')))
rm(eucl)
```

Each data frame has 3,891 sampled prediction points. But recall that we want to predict the genotypes' performance to the whole state. In other words, we have to expand the results of each sampled point by doing an interpolation process. Here, we used the inverse distance weighted (IDW) interpolation method. IDW interpolation operates on the principle that objects nearby exhibit greater similarity than those farther apart. When estimating the value for an unobserved environment, IDW considers the measured values surrounding that environment (i.e., observation window). The measured values closest to the prediction environment carry more weight in determining the predicted value than those farther away. The weights are defined using the Power function (see below). For example, the Euclidean distance between the trial network and a given point (say, u^*) is given by:

$$D_{u^*j} = \frac{\sum_{u=1}^{U} \frac{1}{||u^*-x_u||^\tau} D_{uj}}{\sum_{u=1}^{U} \frac{1}{||u^*-x_u||^\tau}}$$

where x_u refers to a given sampled point u in the observation window, and τ is a power defined by cross-validation (see below).

For interpolating, we must first define an observation window. This window is defined by a bounding box, that corresponds to the administrative limits of the state. We will extract these limits from a shapefile downloaded at IBGE that has the administrative limits for all Brazilian states, as follows:

```r
br_est = raster::shapefile("shapefiles/states/BR_UF_2021.shp")
est_sf = sf::st_as_sf(br_est)

# Selecting only the state of Mato Grosso do Sul
MS = est_sf[est_sf$SIGLA == "MS",]
MS_pol = br_est[br_est$SIGLA == 'MS',]
```

```r
# Building the observation window
obs_window = spatstat.geom::owin(xrange = MS_pol@bbox[1,], yrange = MS_pol@bbox[2,])
```

After defining the observation window, we have to perform an iterative process to define an optimum power to define the weights for the IDW interpolation. We define the optimum power as the one that provides the lower mean square prediction error (MSPE). After defining the optimum power, we perform the interpolation using this value and create a mask to overlap the area of the state. This process is shown below using parallel processing to speed up computational time.

```r
cl = parallel::makeCluster(parallel::detectCores(logical = F)-1)
parallel::clusterEvalQ(cl, {library(spatstat)
                            library(raster)
                            library(sf)})
parallel::clusterExport(cl, varlist = c('obs_window', 'MS'))

dist.list = parallel::parLapply(cl = cl, dist.env, function(x){

  ppp_test = spatstat.geom::ppp(x = x$LON, y = x$LAT,
                 marks = x$dist, window = obs_window)

  powers <- seq(0.1, 5, 0.1)
  mse_result <- NULL
  for(power in powers){
    CV_idw <- spatstat.explore::idw(ppp_test, power=power, at="points")
    mse_result <- c(mse_result, mean((ppp_test$marks-CV_idw)^2, na.rm=T))
  }
  optimal_power <- powers[which.min(mse_result)]

  tr_idw = spatstat.explore::idw(ppp_test, power = optimal_power, at = 'pixels')

  idw_raster = as(raster::mask(raster::raster(tr_idw, crs = crs(br_est)), MS),
                  "SpatialPixelsDataFrame")

  idw_raster.df = as.data.frame(idw_raster)
})
parallel::stopCluster(cl)
rm(cl)
```

This is a computationally intensive process, so we will save the outcome for avoiding doing it again:

```r
save(dist.list, file = 'saves/dist_list.RData')
```

By doing that, we can always reach `dist.list` using the following command:

```r
load(file = 'saves/dist_list.RData')
```

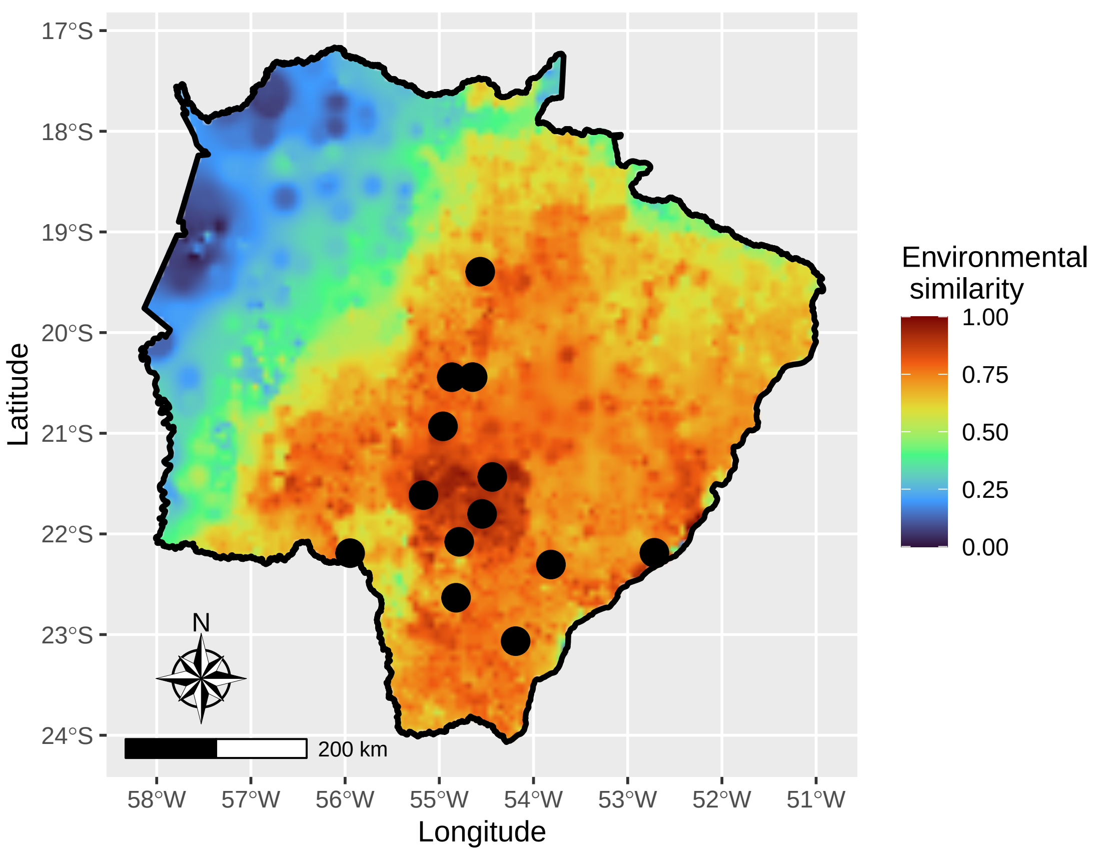
Once we performed the interpolation for all list elements (i.e., for the observed 49 environments), the next step is binding the lists by columns and selecting the minimum value of each row. In other words, the environmental similarity between the observed and unobserved environments (S_u and $S_u^\star$) is given by the minimum distance between the points to be predicted and the TPE:

$$S_u = min(D_{uj}) \quad S_{u^\star} = min(D_{u^\star j})$$

```r
mindist = apply(do.call(cbind, lapply(dist.list, function(x) x[,-c(2,3)])),
                1, function(x) x[which.min(x)])
```

Finally, we can build a map for illustrating how reliable the prediction will be across the whole state:

```r
cbind(dist.list$E01[,-1],
      layer = abs((mindist - min(mindist))/(max(mindist) - min(mindist))-1)) |>
  ggplot() +
  ggspatial::annotation_scale(location = "bl", width_hint = 0.3) +
  ggspatial::annotation_north_arrow(location = "bl", which_north = "true",
                         pad_x = unit(0.5,"cm"), pad_y = unit(0.7,"cm"),
                         style = ggspatial::north_arrow_nautical())+
  geom_raster(aes(x = x, y = y , fill = layer), interpolate = T) +
  geom_sf(data = MS, fill = NA, linewidth = 1, color = "black")  +
  labs(x = 'Longitude', y = 'Latitude', fill = 'Environmental \n similarity') +
  scale_fill_viridis_c(option = 'turbo', direction = 1,
                       limits = c(0, 1)) +
  theme(legend.position = 'right') +
  geom_point(
    data = data.frame("lat" = c(-22.187, -22.191,-21.434,-22.634,
                                -20.442,-22.078,-22.304,-21.614,
                                -23.065,-21.801,-19.395,-20.931,-20.443),
                      "lon" = c(-52.717, -55.947,-54.438,-54.822,-54.646,
                                -54.789,-53.815,-55.168,-54.190,-54.546,
                                -54.566,-54.961,-54.868)),
    aes(x = lon, y = lat, size = 1.2), show.legend = F, color = 'black',
  )
```



```r
rm(dist.list, dist.env)
```

Section 3: Prediction via PLS

After obtaining the environmental data and the factor loadings, it is time to predict the expected performance of selection candidates in untested environments using partial least squares (PLS). Initially, we will train a model with environmental features to predict the loadings for all environments (i.e., observed and unobserved). With the predicted loadings and the previously estimated scores for each genotype from the FA model, we can predict the empirical BLUPs of the genotypes in untested environments.

GIS-FA

The first step is to fit the PLS using the function `plsr()`, of the `pls` package. Initially, we must define the optimum number of components for prediction using leave-one-out cross-validation. In this scheme, we delete the data of one environment and predict it using the PLS model. Then, we compute the correlation between the predicted and observed values, which indicates the model's accuracy. Naturally, we will choose the model with a number of components that provides the highest prediction accuracy.

Let's load into R memory the object we used to save the outputs of the best-fit FA model we previously selected:

```r
load('saves/FA4.RDA')
```

The computations in `R` work as follows. The object `data.pls.lamb` below is a data frame with the left-hand side composed of dependent variables, i.e., the matrix of rotated loadings, and the right-hand side composed of predictors, i.e. ,the matrix of environmental covariates in the tested environments of the PLS model. Note that we are scaling the matrix of predictors.

```r
data.pls.lamb = data.frame(lambda = I(fa4$rot.loads),
                           CovAmb = I(scale(as.matrix(matcovamb))))
```

Then, an initial PLS model is fitted to find out the maximum number of components that can be used for prediction:

```r
pls1.lamb = pls::plsr(lambda ~ CovAmb, validation = 'CV', data = data.pls.lamb)
```

This information will be used in the cross-validation process, where we will defined the best PLS model based on the prediction accuracy:

```r
ncomp = list()
for (j in 1:pls1.lamb$ncomp) {
  acc_eblup = vector()
  acc_lamb = vector()
  for (i in rownames(matcovamb)) {
    train = data.pls.lamb[rownames(data.pls.lamb) != i,]
    test = data.pls.lamb[rownames(data.pls.lamb) == i,]

    cvloo = pls::plsr(lambda ~ CovAmb, validation = 'CV', data = train,
                 ncomp = j)

    acc_lamb = c(
      acc_lamb,
      cor(
        t(test$CovAmb %*% coef(cvloo, intercept = T)[-1,,1] +
            coef(cvloo, intercept = T)[1,,1]),
        t(test$lambda)
      )
    )

    acc_eblup = c(
      acc_eblup,
      cor(
        t(tcrossprod(
          test$CovAmb %*% coef(cvloo, intercept = T)[-1,,1] +
            coef(cvloo, intercept = T)[1,,1],
          fa4$rot.scores
        )),
        fa4$blups[fa4$blups$env == i, 'marginal'], method = 'spearman'
      )
    )
    rm(cvloo, train, test)
  }
  ncomp[[j]] = data.frame(ncomp = j, lambda = mean(acc_lamb),
                          eblup = mean(acc_eblup))
}
cv.results = do.call(rbind, ncomp)

cv.results[which.max(cv.results$lambda),] # Best results for the prediction of loadings
```

```
##   ncomp    lambda     eblup
## 8     8 0.5722463 0.7399924
```

```
cv.results[which.max(cv.results$eblup),] # Best results for the prediction of eBLUPs
```

```
##   ncomp    lambda     eblup
## 3     3 0.5603842 0.7436089
```

Thus, the PLS model with 3 components had the best outcomes. We will now fit the proper prediction model, as follows:

```
pls2.lamb = pls::plsr(lambda ~ CovAmb, validation = 'CV', data = data.pls.lamb,
                 ncomp = cv.results[which.max(cv.results$eblup),'ncomp'])
pls::explvar(pls2.lamb)
```

```
##   Comp 1   Comp 2   Comp 3
## 33.20491 18.36629 10.00163
```

This model explains 61.57% of the total variance.

Using the same model, we can investigate its prediction ability within environments using eBLUEs. Let's first fit a model per environment considering fixed genetic and block main effects:

```
data = read.csv("data/data_soy.csv")
data = transform(data,
                 gen = factor(gen),
                 block = factor(block),
                 env = factor(env),
                 row = factor(row),
                 col = factor(col))
eblues = lapply(split(data, f = data$env), function(x){
    mod = asreml(fixed = yield ~ gen + block,
                 na.action = na.method(x="include", y = "include"),
                 data = x)
    predict(mod, classify = 'gen')$pvals[,1:2]
  })
```

Now, we can perform the cross-validation:

```
acc_eblue = vector()
for (i in rownames(matcovamb)) {
  train = data.pls.lamb[rownames(data.pls.lamb) != i,]
  test = data.pls.lamb[rownames(data.pls.lamb) == i,]

  cvloo = pls::plsr(lambda ~ CovAmb, validation = 'CV', data = train,
               ncomp = cv.results[which.max(cv.results$eblup),'ncomp'])

  pred_blup = tcrossprod(
    test$CovAmb %*% coef(cvloo, intercept = T)[-1,,1] +
      coef(cvloo, intercept = T)[1,,1],
    fa4$rot.scores
  )

  blue = eblues[[i]]|>
    dplyr::left_join(data.frame(t(pred_blup)) |>
                       tibble::rownames_to_column('gen'), by = 'gen')

  acc_eblue = c(acc_eblue, cor(blue[,2], blue[,3], use = "na.or.complete",
                method = "spearman"))
  rm(cvloo, train, test, blue, pred_blup)
}

mean(acc_eblue)
```

```
## [1] 0.5490957
```

GIS-GGE

The GIS-GGE is a PLS model whose training dataset uses the estimated eBLUP values of genotypic means within observed environments (Costa-Neto et al., 2022). We hypothesize the GIS-FA model has a higher prediction accuracy than the GIS-GGE model.

In the dataset used for this tutorial, there are 182 genotypes and 49 observed environments. Let's now apply the GIS-GGE model and compare its results with the GIS-model, as follows:

```
# Building the data frame
GGE = fa4$blups
GGE = reshape(GGE[,1:3], idvar = 'gen', timevar = 'env', direction = 'wide')
colnames(GGE)[-1] = sub("conditional.","",colnames(GGE)[-1])
```

```r
rownames(GGE) = levels(fa4$blups$gen)
GGE = t(GGE[,-1])
data.pls.gge = data.frame(GGE = I(GGE), CovAmb = I(scale(matcovamb)))

# Fitting the first PLS model
pls1.gge = pls::plsr(GGE ~ CovAmb, validation = 'CV', data = data.pls.gge)

# Performing the cross-validation
ncomp = list()
for (j in 1:pls1.gge$ncomp) {
  acc_eblue.gge = vector()
  acc_eblup.gge = vector()
  for (i in rownames(matcovamb)) {
    train = data.pls.gge[rownames(data.pls.gge) != i,]
    test = data.pls.gge[rownames(data.pls.gge) == i,]

    cvloo = pls::plsr(GGE ~ CovAmb, validation = 'CV', data = train,
                      ncomp = j)

    pred_blup = test$CovAmb %*% coef(cvloo, intercept = T)[-1,,1] +
      coef(cvloo, intercept = T)[1,,1]

    blue = eblues[[i]] |>
      dplyr::left_join(data.frame(t(pred_blup)) |>
                         tibble::rownames_to_column('gen'), by = 'gen')
    blup = fa4$blups[fa4$blups$env == i, c('gen', 'conditional')] |>
       dplyr::left_join(data.frame(t(pred_blup)) |>
                          tibble::rownames_to_column('gen'), by = 'gen')

    acc_eblue.gge = c(acc_eblue.gge, cor(blue[,2], blue[,3],
                                         use = "na.or.complete", method = 'spearman'))
    acc_eblup.gge = c(acc_eblup.gge, cor(blup[,2], blup[,3],
                                         use = "na.or.complete", method = 'spearman'))
    rm(cvloo, train, test, blue, pred_blup, blup)
  }
  ncomp[[j]] = data.frame(ncomp = j, eblue = mean(acc_eblue.gge),
                          eblup = mean(acc_eblup.gge))
}
cv.results.gge = do.call(rbind, ncomp)

# Choosing the number of components
cv.results.gge[which.max(cv.results.gge$eblup),] #Highest accuracy for predicting eBLUPs
```

```
##   ncomp     eblue     eblup
## 8     8 0.5137178 0.7068775
```

```
cv.results.gge[which.max(cv.results.gge$eblue),] #Highest accuracy for predicting eBLUEs
```

```
##   ncomp     eblue     eblup
## 1     1 0.5305078 0.6896101
```

GIS-FA vs GIS-GGE

The results are presented in the table below. The predictive ability of the GIS-FA model was 5.2% and 3.5% superior than the GIS-GGE model when predicting eBLUP and eBLUE values, respectively.

Models	eBLUPs	eBLUEs
GIS-FA	0.7384869	0.5481807
GIS-GGE	0.7071189	0.5306063

Let's clean R memory before jumping into the next section:

```
rm(cv.results.gge, cv.results, data.pls.gge, data.pls.lamb, eblues, GGE, ncomp,
   pls1.gge, pls1.lamb, acc_eblue, acc_eblue.gge, acc_eblup, acc_eblup.gge,
   acc_lamb, i, j)
```

Prediction in untested environments

Given the GIS-FA model was superior to the GIS-GGE model, let's predict the phenotypic performance of the 182 genotypes in untested environments using the GIS-FA model. The first step is extracting the regression coefficients from the fitted model:

```
coef_pls = coef(pls2.lamb, intercept = T)[,,1]
```

Then, we will use the environmental data from the unobserved environments to estimate their factor loadings:

```
pred_lambda = scale(matcovamb_pred) %*% coef_pls[-1, ]

pred_lambda = cbind(
  fa1 = pred_lambda[,'fa1'] + coef_pls["(Intercept)", "fa1"],
  fa2 = pred_lambda[,'fa2'] + coef_pls["(Intercept)", "fa2"],
  fa3 = pred_lambda[,'fa3'] + coef_pls["(Intercept)", "fa3"],
  fa4 = pred_lambda[,'fa4'] + coef_pls["(Intercept)", "fa4"]
)
```

And finally, let's combine multiple regression with the FA model (i.e., the GIS-FA) to predict the eBLUP values in unobserved environments:

```r
EBLUPs_pred = as.matrix(pred_lambda[,'fa1']) %*% fa4$rot.scores[,'fa1'] +
              as.matrix(pred_lambda[,'fa2']) %*% fa4$rot.scores[,'fa2'] +
              as.matrix(pred_lambda[,'fa3']) %*% fa4$rot.scores[,'fa3'] +
              as.matrix(pred_lambda[,'fa4']) %*% fa4$rot.scores[,'fa4'] +
              mean(data$yield, na.rm = T)

colnames(EBLUPs_pred) = levels(data$gen)
```

Section 4: Visualization using maps

After performing the predictions, we can build thematic maps to aid the visualization. In addition to the shapefile of the Brazilian state's administrative boundaries already loaded into `R`, we will also load a shapefile with the boundaries of Brazilian biomes. We intend to highlight in the map the portion corresponding to Pantanal, a protected area. Pantanal is represented by a contour with the same colour as the background in maps.

```r
br_bio = raster::shapefile("shapefiles/biomes/lm_bioma_250.shp")
bio_sf = sf::st_as_sf(br_bio)
Pant = bio_sf[bio_sf$Bioma == 'Pantanal',]
Pant_crop = sf::st_intersection(Pant, MS)
```

Interpolation

Recall that we made the prediction only for sampled points. To expand the prediction to the whole state, we have to perform an interpolation process. The procedure is similar to what was previously described.

```r
# Building a list with data frames containing the eBLUPs of each sampled point
pred.list = lapply(apply(EBLUPs_pred, 2, function(x) reshape2::melt(x)),
                          function(x) x |>
                     dplyr::rename(blup = value) |>
                     tibble::rownames_to_column('point') |>
                     dplyr::left_join(
                       unique(covamb_pred[,c('env', 'LON', 'LAT')]),
                                    by = c('point' = 'env'))
                   )

# Initializing the parallelization
cl = parallel::makeCluster(parallel::detectCores(logical = F)-1)
parallel::clusterEvalQ(cl, {library(spatstat)
                            library(raster)
                            library(sf)})
parallel::clusterExport(cl, varlist = c('obs_window', 'MS'))

pred.list = parallel::parLapply(cl, pred.list, function(x){
  ppp_test = spatstat.geom::ppp(x = x[which(!is.na(x$blup)),'LON'],
                  y = x[which(!is.na(x$blup)),'LAT'],
                  marks = na.exclude(x$blup), window = obs_window)

  powers <- seq(0.1, 5, 0.05)
  mse_result <- NULL
  for(power in powers){
    CV_idw <- spatstat.explore::idw(ppp_test, power=power, at="points")
    mse_result <- c(mse_result, mean((ppp_test$marks-CV_idw)^2, na.rm=T))
  }
  optimal_power <- powers[which.min(mse_result)]

  tr_idw = spatstat.explore::idw(ppp_test, power = optimal_power, at = 'pixels')

  idw_raster = as(raster::mask(raster::raster(tr_idw, crs = crs(br_est)), MS),
                   "SpatialPixelsDataFrame")

  idw_raster.df = as.data.frame(idw_raster)
})
parallel::stopCluster(cl)
```

As before, this was computationally intensive process, so we will save the outcome as follows:

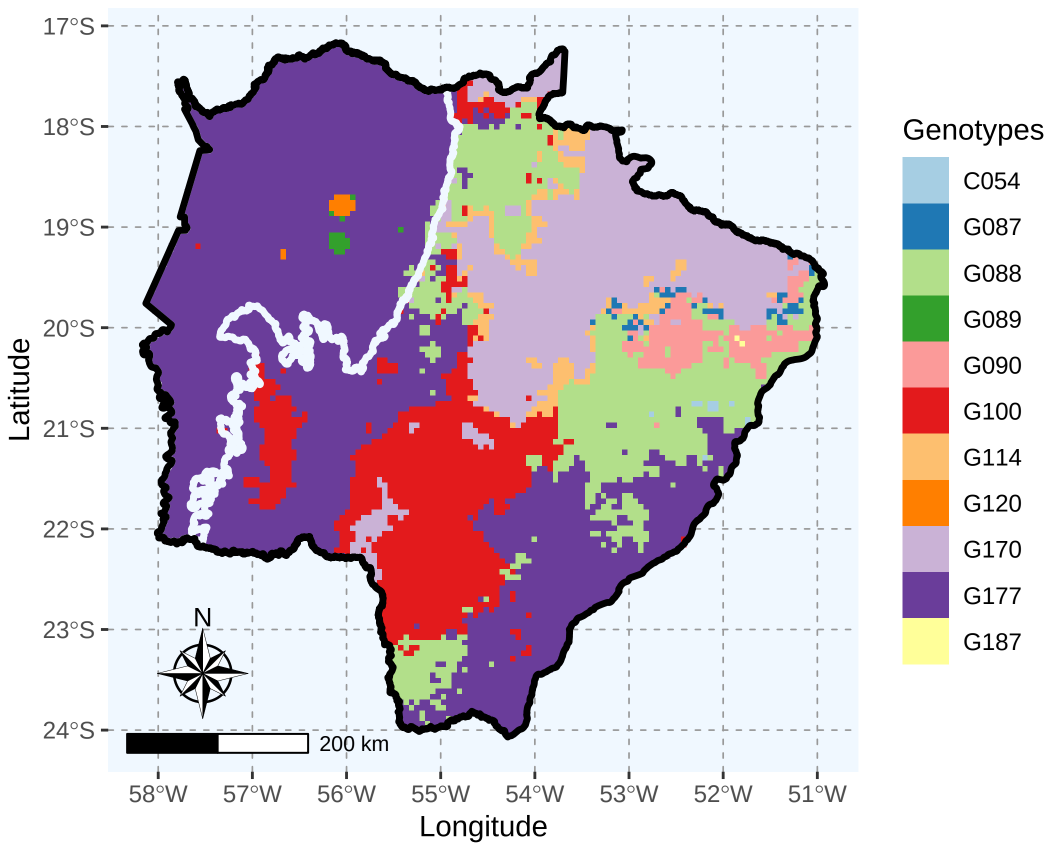
```r
save(pred.list, file = "saves/pred_blups.RData")
```

Now everything is set for building the maps.

Which-won-where

The "which-won-where" map shows which genotype performed best on each unobserved environment:

```r
cbind(pred.list[[1]][,c('x','y')],
            winner = apply(cbind(pred.list[[1]][,c('x','y')],
                do.call(cbind, lapply(pred.list, function(x) x[,'layer']))),
          1, function(x) names(which(x == max(x, na.rm=T))), simplify = T)) |>
  ggplot() +
  ggspatial::annotation_scale(location = "bl", width_hint = 0.3) +
  ggspatial::annotation_north_arrow(location = "bl", which_north = "true",
                         pad_x = unit(0.5,"cm"), pad_y = unit(0.7,"cm"),
                         style = ggspatial::north_arrow_nautical())+
  geom_raster(aes(x = x, y = y , fill = winner)) +
  geom_sf(data = Pant_crop, fill = NA, lwd=1.2, linetype = 'solid',
          color = 'aliceblue', show.legend = T, na.rm = T)+
  geom_sf(data= MS, fill = NA, linewidth = 1.2, color = 'black')+
  labs(x = 'Longitude', y = 'Latitude',
       fill = "Genotypes",color = "")+
  theme(panel.grid.major = element_line(color = gray(.6), linetype = "dashed",
                                        linewidth = 0.3),
        panel.background = element_rect(fill = "aliceblue"),
        legend.position = 'right',
        legend.spacing = unit(.02,'cm')) +
    scale_fill_brewer(type = 'qual', palette = 'Paired') +
  guides(fill = guide_legend(order = 1, override.aes = list(colour = 0)))
```

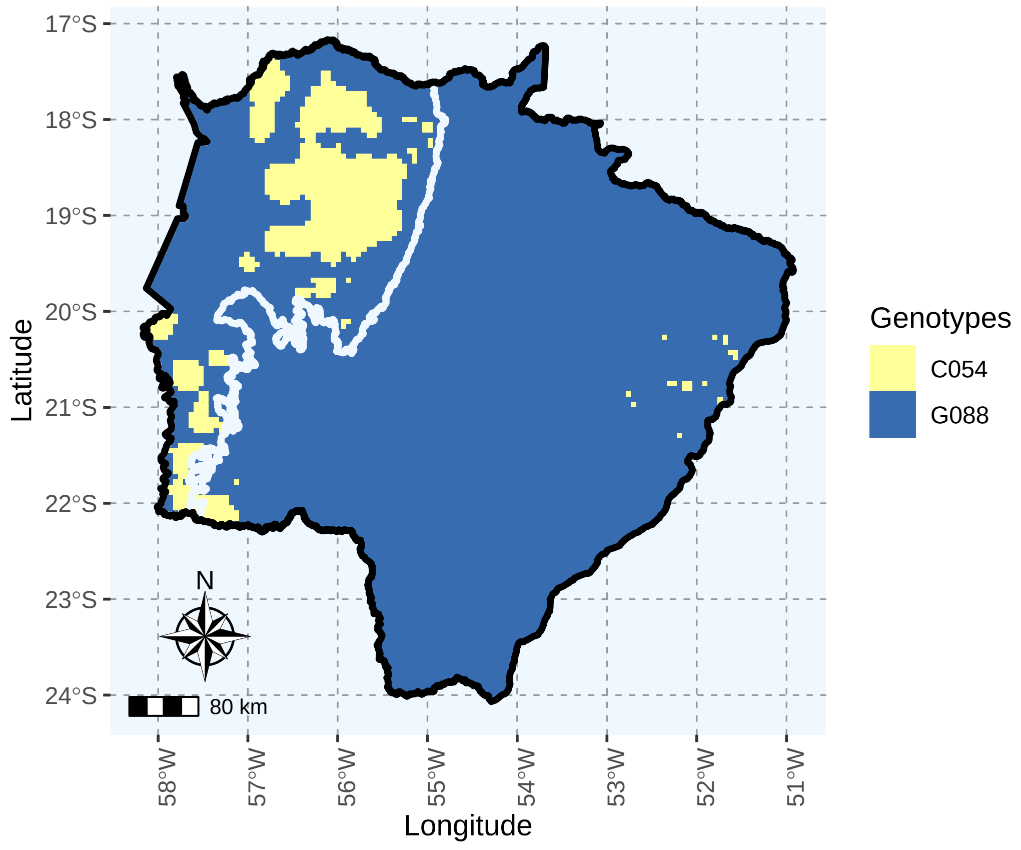


Pairwise comparisons

The following map presents the pairwise comparison between an experimental genotype (G088) and a check cultivar (C054). The winner is represented by its colour, as in the "which-won-where" plot.

```
cbind(pred.list[[1]][,c('x','y')],
      winner = apply(
        do.call(cbind, lapply(pred.list, function(x) x[,"layer"]))[, c("C054",
                                                                      "G088")],
        1, function(x) names(which.max(x))
```

```
    )) |>
ggplot() +
ggspatial::annotation_scale(location = "bl", width_hint = .1) +
ggspatial::annotation_north_arrow(location = "bl", which_north = "true",
                       pad_x = unit(0.5,"cm"), pad_y = unit(0.7,"cm"),
                       style = ggspatial::north_arrow_nautical())+
geom_raster(aes(x = x, y = y , fill = winner)) +
geom_sf(data = Pant_crop, fill = NA, lwd=1.2, linetype = 'solid',
        color = 'aliceblue', show.legend = T, na.rm = T)+
geom_sf(data= MS, fill = NA, linewidth = 1.2, color = 'black')+
labs(x = 'Longitude', y = 'Latitude',
     fill = "Genotypes",color = "")+
theme(panel.grid.major = element_line(color = gray(.6), linetype = "dashed",
                                      linewidth = 0.3),
      panel.background = element_rect(fill = "aliceblue"),
      legend.position = 'right',
      #legend.box.margin = margin(.05,.05,.05,.05,'cm'),
      legend.spacing = unit(.02,'cm'),
      axis.text.x = element_text(angle = 90)) +
scale_fill_manual(values = c("C054" = "#ffff99", "G088" = "#386cb0")) +
guides(fill = guide_legend(order = 1, override.aes = list(colour = 0)))
```



Genotypes' performance map

The following map depict the expected performance of genotypes in untested environments, which might be useful for recommendation purposes.

Let's first compile the `pred.list` object into a single data frame:

```
EBLUPs_pred = do.call(rbind, pred.list)  |>
  rownames_to_column('gen') |>
  separate(gen, into = c('gen', 'trash')) |>
  dplyr::select(-trash) |>
  rename(blup = layer)

str(EBLUPs_pred)
```

```
## 'data.frame':    1842750 obs. of  4 variables:
##  $ gen : chr  "C006" "C006" "C006" "C006" ...
##  $ blup: num  3368 3386 3401 3342 3362 ...
##  $ x   : num  -56.2 -56.1 -56 -56.2 -56.2 ...
##  $ y   : num  -17.2 -17.2 -17.2 -17.2 -17.2 ...
```

Then, we will divide the genotypes' predicted performance into groups to facilitate interpretation. For example, performances superior to 4000 kg ha^{-1} are classified as group 8, between 3750 and 4000 kg ha^{-1} as group 7, and so on. This classification helps to determine if a given genotype will have a minimum predicted performance in a specific environment or region.

```
EBLUPs_pred = EBLUPs_pred %>% mutate(
  class = case_when(
    blup > min(blup, na.rm = T) & blup <= 2500 ~ 1,
    blup > 2500 & blup <= 2750 ~ 2,
    blup > 2750 & blup <= 3000 ~ 3,
    blup > 3000 & blup <= 3250 ~ 4,
    blup > 3250 & blup <= 3500 ~ 5,
    blup > 3500 & blup <= 3750 ~ 6,
    blup > 3750 & blup <= 4000 ~ 7,
    blup > 4000 & blup <= max(blup, na.rm = T) ~ 8
  )
)
```

Lastly, we will calculate the Overall Performance (OP) and the root mean square deviation ($RMSD$) using the factor analytic selection tools. These metrics can be employed as a filter for choosing which genotypes the breeder should focus on.

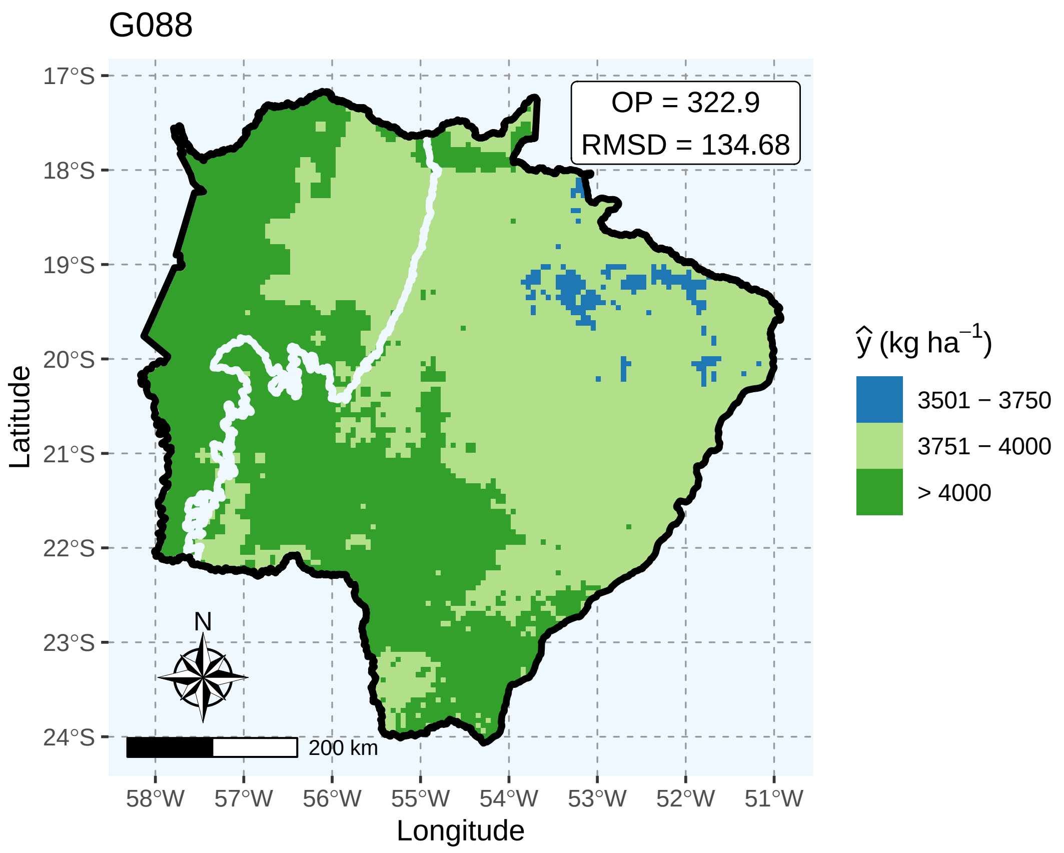
```
rel = fa4$blups |> reframe(rel = 1-(mean(std.error^2)/mean(diag(fa4$Gvcov))),
                           .by = gen)

FAST = data.frame(
  env = fa4$blups$env,
  gen = fa4$blups$gen,
  RMSD = (fa4$blups$marginal - kronecker(fa4$rot.loads[,1], diag(nlevels(data$gen))) %*%
                        fa4$rot.scores[,1])^2
) |> reframe(RMSD = sqrt(mean(RMSD)), .by = gen) |>
  mutate(OP = mean(fa4$rot.loads[,1]) * fa4$rot.scores[,1],
         index = 2*((OP-mean(OP))/sqrt(var(OP))) - ((RMSD-mean(RMSD))/sqrt(var(RMSD))),
         rel = rel$rel)
```

Finally, we can build the adaptation maps. For example, for G088:

```
g = "G088"

EBLUPs_pred |> filter(gen == g) |>
  ggplot() +
  ggspatial::annotation_scale(location = "bl", width_hint = 0.3) +
  ggspatial::annotation_north_arrow(location = "bl", which_north = "true",
                          pad_x = unit(0.5,"cm"), pad_y = unit(0.7,"cm"),
                          style = ggspatial::north_arrow_nautical())+
  geom_raster(aes(x = x, y = y , fill = factor(class))) +
  geom_sf(data = Pant_crop, fill = NA, lwd=1.2, linetype = 'solid',
          color = 'aliceblue', show.legend = T, na.rm = T)+
  geom_sf(data= MS, fill = NA, linewidth = 1.2, color = 'black')+
  labs(x = 'Longitude', y = 'Latitude',
       fill = expression(widehat(y)~"("*kg~ha^{-1}*")"), color = "",
       title = g)+
  theme(panel.grid.major = element_line(color = gray(.6), linetype = "dashed",
                                        linewidth  = 0.3),
        panel.background = element_rect(fill = "aliceblue"),
        legend.position = 'right',
        legend.spacing = unit(.02,'cm')) +
  scale_fill_manual(values = c("1" = '#e31a1c',
                               '2' = '#fb9a99',
                               '3' = '#fdbf6f',
                               '4' = '#ff7f00',
                               '5' = '#a6cee3',
                               '6' = '#1f78b4',
                               '7' = '#b2df8a',
                               '8' = '#33a02c'),
                     labels = c("1" = 'Non-adapted',
                               '2' = '2500 - 2750',
                               '3' = '2751 - 3000',
                               '4' = '3001 - 3250',
                               '5' = '3251 - 3500',
                               '6' = '3501 - 3750',
                               '7' = '3751 - 4000',
                               '8' = '> 4000')) +
guides(fill = guide_legend(order = 1, override.aes = list(colour = 0))) +
  annotate(
    'label', x = -52, y = -17.5,
    label = paste(
      paste0(
        "OP = ", round(FAST[FAST$gen == g, "OP"],2)
        ),
      paste0(
        "RMSD = ", round(FAST[FAST$gen == g, "RMSD"],2)
        ),
      sep = '\n'
      )
    )
```



Closing remarks

In this tutorial, we presented the main ideas from our paper. We hope the GIS-FA model to be routinely incorporated in breeding programs as an important tool for cultivar recommendation in MET.

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*(6), 716–723. https://doi.org/10.1109/TAC.1974.1100705

Butler, D. (2022). *Asreml: Fits the linear mixed model*. www.vsni.co.uk

Chaves, S. F. S., Alves, R. S., Dias, L. A. S., Alves, R. M., Dias, K. O. G., & Evangelista, J. S. P. C. (2023). Analysis of repeated measures data through mixed models: An application in *Theobroma grandiflorum* breeding. *Crop Science*, *63*(4), 2131–2144. https://doi.org/10.1002/csc2.20995

Chaves, S. F. S., Evangelista, J. S. P. C., Trindade, R. S., Dias, L. A. S., Guimarães, P. E., Guimarães, L. J. M., Alves, R. S., Bhering, L. L., & Dias, K. O. G. (2023). Employing factor analytic tools for selecting high-performance and stable tropical maize hybrids. *Crop Science*, *63*(3), 1114–1125. https://doi.org/10.1002/csc2.20911

Corporation, M., & Weston, S. (2022). *doParallel: Foreach parallel adaptor for the 'parallel' package*. https://CRAN.R-project.org/package=doParallel

Costa-Neto, G., Crespo-Herrera, L., Fradgley, N., Gardner, K., Bentley, A. R., Dreisigacker, S., Fritsche-Neto, R., Montesinos-López, O. A., & Crossa, J. (2022). Envirome-wide associations enhance multi-year genome-based prediction of historical wheat breeding data. *G3: Genes|Genomes|Genetics*, *13*(2), jkac313. https://doi.org/10.1093/g3journal/jkac313

Costa-Neto, G., Gilli, G., Carvalho, H. F., Crossa, J., & Fritsche-Neto, R. (2021). EnvRtype: A software to interplay enviromics and quantitative genomics in agriculture. *G3 Genes|Genomes|Genetics*, *11*, jkab040. https://doi.org/10.1093/g3journal/jkab040

Cullis, B. R., Smith, A. B., Beeck, C. P., & Cowling, W. A. (2010). Analysis of yield and oil from a series of canola breeding trials. Part II. Exploring variety by environment interaction using factor analysis. *Genome*, *53*(11), 1002–1016. https://doi.org/10.1139/G10-080

Cullis, B. R., Smith, A. B., & Coombes, N. E. (2006). On the design of early generation variety trials with correlated data. *Journal of Agricultural, Biological, and Environmental Statistics*, *11*(4), 381–393. https://doi.org/10.1198/108571106X154443

Dunnington, D. (2023). *Ggspatial: Spatial data framework for ggplot2*. https://CRAN.R-project.org/package=ggspatial

Gu, Z. (2022). Complex heatmap visualization. *iMeta*.

Gu, Z., Gu, L., Eils, R., Schlesner, M., & Brors, B. (2014). Circlize implements and enhances circular visualization in r. *Bioinformatics*, *30*, 2811–2812.

Hijmans, R. J. (2023). *Raster: Geographic data analysis and modeling*. https://CRAN.R-project.org/package=raster

Hijmans, R. J., Barbosa, M., Ghosh, A., & Mandel, A. (2023). *Geodata: Download geographic data*. https://CRAN.R-project.org/package=geodata

Kassambara, A. (2023). *Ggpubr: 'ggplot2' based publication ready plots*. https://CRAN.R-project.org/package=ggpubr

Lembrechts, J. J., Hoogen, J. van den, Aalto, J., Ashcroft, M. B., De Frenne, P., Kemppinen, J., Kopecký, M., Luoto, M., Maclean, I. M. D., Crowther, T. W., Bailey, J. J., Haesen, S., Klinges, D. H., Niittynen, P., Scheffers, B. R., Van Meerbeek, K., Aartsma, P., Abdalaze, O., Abedi, M., . . . Lenoir, J. (2022). Global maps of soil temperature. *Glob Chang. Biol.*, *28*(9), 3110–3144. https://doi.org/10.1111/gcb.16060

Liland, K. H., Mevik, B.-H., & Wehrens, R. (2022). *Pls: Partial least squares and principal component regression*. https://CRAN.R-project.org/package=pls

Microsoft, & Weston, S. (2022). *Foreach: Provides foreach looping construct*. https://CRAN.R-project.org/package=foreach

Mrode, R. A. (2014). *Linear models for the prediction of animal breeding values* (3rd ed). CABI.

Pebesma, E., & Bivand, R. (2023). *Spatial Data Science: With applications in R* (p. 352). Chapman and Hall/CRC. https://r-spatial.org/book/

Piepho, H.-P. (1997). Analyzing genotype-environment data by mixed models with multiplicative terms. *Biometrics*, *53*(2), 761–766. https://doi.org/10.2307/2533976

Piepho, H.-P. (2019). A coefficient of determination (r^2) for generalized linear mixed models. *Biometrical Journal*, *61*(4), 860–872. https://doi.org/10.1002/bimj.201800270

R Core Team. (2023). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Slowikowski, K. (2023). *Ggrepel: Automatically position non-overlapping text labels with 'ggplot2'*. https://CRAN.R-project.org/package=ggrepel

Smith, A. B., & Cullis, B. R. (2018). Plant breeding selection tools built on factor analytic mixed models for multi-environment trial data. *Euphytica*, *214*(8), 143. https://doi.org/10.1007/s10681-018-2220-5

Smith, A. B., Cullis, B. R., & Thompson, R. (2001). Analyzing variety by environment data using multiplicative mixed models and adjustments for spatial field trend. *Biometrics*, *57*(4), 1138–1147. https://doi.org/10.1111/j.0006-341X.2001.01138.x

Smith, A. B., Ganesalingam, A., Kuchel, H., & Cullis, B. R. (2015). Factor analytic mixed models for the provision of grower information from national crop variety testing programs. *Theoretical and Applied Genetics*, *128*(1), 55–72. https://doi.org/10.1007/s00122-014-2412-x

Smith, A. B., Norman, A., Kuchel, H., & Cullis, B. R. (2021). Plant variety selection using interaction classes derived from factor analytic linear mixed models: Models with independent variety effects. *Frontiers in Plant Science*, *12*, 1857. https://doi.org/10.3389/fpls.2021.737462

Stefanova, K. T., & Buirchell, B. (2010). Multiplicative mixed models for genetic gain assessment in lupin breeding. *Crop Science*, *50*(3), 880–891. https://doi.org/10.2135/cropsci2009.07.0402

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Wong, J. (2022). *Pdist: Partitioned distance function*. https://CRAN.R-project.org/package=pdist

Yutani, H. (2022). *Gghighlight: Highlight lines and points in 'ggplot2'*. https://CRAN.R-project.org/package=gghighlight