Tutorial Code for estimating CovGE

Here we provide an example for the covariance analysis in which we apply the method described in Albecker et al. (In Review).

As always, the first steps are to load the necessary packages, data, and functions. These are available on the Github (https://github.com/RCN-ECS/CnGV/tree/master/src/)

```
# Packages
if (!require('lme4')) install.packages('lme4'); library('lme4')
```

```
## Loading required package: lme4
```

```
## Loading required package: Matrix
```

```
if (!require('emmeans')) install.packages('emmeans'); library('emmeans')
```

```
## Loading required package: emmeans
```

```
if (!require('progress')) install.packages('progress'); library('progress')
```

```
## Loading required package: progress
```

```
if (!require('viridis')) install.packages('viridis'); library('viridis')
```

```
## Loading required package: viridis
```

```
## Loading required package: viridisLite
```

```
if (!require('tidyverse')) install.packages('tidyverse'); library('tidyverse')
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
```

```
if (!require('devtools')) install.packages('devtools'); library('devtools')
```

```
## Loading required package: devtools
```

```
## Loading required package: usethis
```

```
##
## Attaching package: 'devtools'
```

```
## The following object is masked from 'package:emmeans':
##
##     test
```

```
if (!require('RCurl')) install.packages('RCurl'); library('RCurl')
```

```
## Loading required package: RCurl
```

```
##
## Attaching package: 'RCurl'
```

```
## The following object is masked from 'package:tidyr':
##
##     complete
```

```
emm_options(msg.interaction = FALSE)

# Functions
devtools::source_url("https://raw.githubusercontent.com/RCN-ECS/CnGV/master/src/EmpiricalDataFunctions.
```

```
## SHA-1 hash of file is 3430bde728c99a516257b8dc4cdcc81707455bc6
```

```
# Phenotype Datasets
x <- getURL("https://raw.githubusercontent.com/RCN-ECS/CnGV/master/data/Balanced_tutorial.csv")
balanced_df <- read.csv(text = x)
imbalanced_df <- filter(balanced_df, gen_factor != "G_3") # Omits one genotype from E_1

# Focal Dataset
data <- balanced_df # replace depending on which dataset you want to use

# Indicator data
x2 <- getURL("https://raw.githubusercontent.com/RCN-ECS/CnGV/master/data/Indicator_tutorial.csv")
indicator <-read.csv(text = x2)
colnames(indicator) <- c("gen_factor", "nat_env_factor", "indicator", "Notes")

# Format genotype and environments into factors
data$gen_factor = factor(data$gen_factor)                ### Correct call name for "genotype" or "popul
data$exp_env_factor = factor(data$exp_env_factor)        ### Correct call name for categorical "treatme
data$nat_env_factor = factor(data$nat_env_factor)        ### Correct call name for categorical "native

# Visualize data
head(data)
```

```
##   gen_factor exp_env_factor nat_env_factor      phen
## 1        G_1            E_1            E_1 0.2246446
## 2        G_1            E_1            E_1 0.5386430
## 3        G_1            E_1            E_1 0.1114093
## 4        G_1            E_1            E_1 1.4524258
## 5        G_1            E_1            E_1 1.3636567
## 6        G_1            E_2            E_1 0.9396904
```

Information on data

These data are simulated phenotypic data collected from an imagined common garden experiment. We created unitless phenotypic data for 8 genotypes across a 2 - environment gradient (four populations/genotypes per environment). Because we use categorical models for this analysis, `gen_factor` (genotype), `exp_env_factor` (experimental treatment), and `nat_env_factor` (information about which environment each genotype is

native to) should both be formatted as factors.

For example, this scenario resembles an experiment in which a researcher collects 10 individual from 4 northern, cold populations, and 10 individuals from 4 southern, warm populations and then in the laboratory, exposes 5 individuals from each population to warm conditions, and 5 individuals from each population to cold conditions. Phenotypic data are then collected.

We present phenotypic data that is unitless and has already been standardized. We standardized phenotypic data to a mean of zero by subtracting the overall group mean from each estimate and dividing by the standard deviation of group means. Groups are each genotype-experimental environmental group.

We have two datasets available - one balanced and one imbalanced. There can be two sources of imbalance - in the number of samples and the number of genotypes. The estimated marginal means accounts for bias in sample size. However, imbalance in the number of genotypes from each native environment must be corrected because unbalanced designs can bias the overall mean ($\overline{y}$ in the CovGE equation) and environmental mean ($\overline{y_j}$ in the CovGE equation). True reciprocal transplant designs should always be balanced, but common garden may be more likely to be imbalanced. We present both datasets for purposes of instruction (the imbalanced data is the same as the balanced, but is missing 1 genotype from E_1 native environment.

Both can be run by changing `data <- imbalanced` or `data <- balanced` above.

Finally, because the analysis calculated covariance when genotype is correctly paired with its native environemt, we also include an indicator dataset which contains the indicator variable with the information about each genotype's native environment (see equation in main paper).

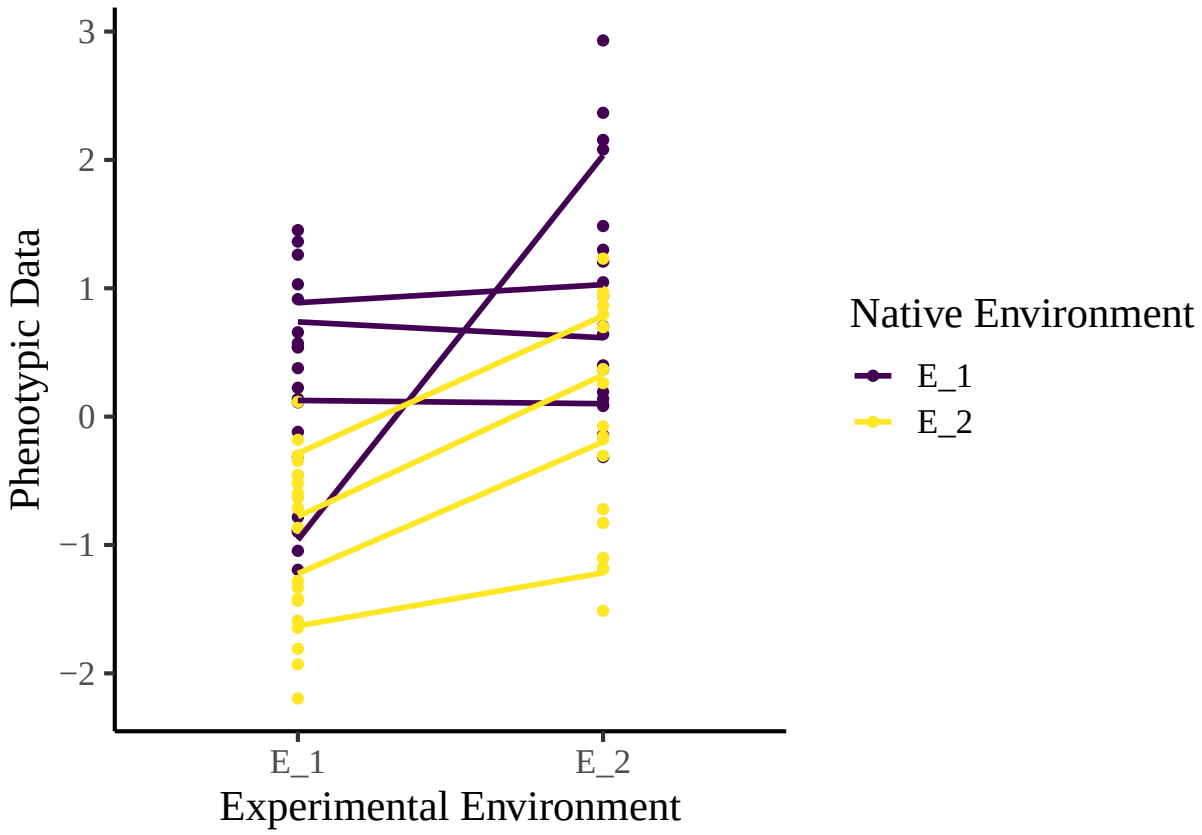
Below, we present 2 approaches: We begin by showing the i) step-by-step process to calculate covGE, but also present a single function at the end with all steps rolled into one.

```
# Example of standardization code
# Important: example data are ALREADY standardized- this is for heuristic purposes only

group = paste(data$gen_factor,data$exp_env_factor,sep = "-")
corrected_phenotype = (data$phen - mean(data$phen, na.rm = TRUE))/sd(tapply(data$phen, group, mean, na.
```

First, Let's visualize phenotypic data:

```
ggplot(data, aes(x = exp_env_factor, y = phen, group = gen_factor, colour = nat_env_factor)) +
  geom_point() + geom_smooth(method = "glm", se = FALSE)+
  labs(colour = "Native Environment")+
  xlab("Experimental Environment") + ylab("Phenotypic Data")+
  theme_classic(base_family = "Times", base_size = 16) + scale_colour_viridis(discrete = TRUE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Step By Step Calculation of CovGE:

To start out, we run basic categorical linear models to generate $\bar{y_i}$,$\bar{y_j}$, and $\bar{y}$ parameters via estimated marginal means. Estimated marginal means are more robust given unbalanced study designs.

If the design is unbalanced, we will run a second anova to generate $\bar{y_j}$ and $\bar{y}$ parameters that correct for the bias

After running the model, we extract estimated marginal means using function `emmeans()`.

```
# Anova for yi
aov.test <- lm(phen ~ exp_env_factor * gen_factor, data = data)

# Anova for yj and ybar
aov.test2 <- lm(phen ~ exp_env_factor * nat_env_factor, data = data) # native environment serves as the

# Estimated Marginal Means
emm_df1 = as.data.frame(emmeans(aov.test, ~ exp_env_factor*gen_factor))
emm_df2 = as.data.frame(emmeans(aov.test2, ~ exp_env_factor*nat_env_factor))
```

We next we need to calculate $\bar{y_i}$ (genotypic means) and $\bar{y_j}$ (experimental environment means) and $\bar{y}$ (overall means).

We do this using `tapply()` to calculate the mean phenotype for each genotype ACROSS environments (G_matrix), and then the mean phenotype for each environment ACROSS genotypes (E_matrix).

Because this is a common garden design, there should be more genotypic means than environmental means.

```
# Use emms from aov.test1
G_matrix <- data.frame("G_means" = tapply(emm_df1$emmean, emm_df1$gen_factor, mean, na.rm=TRUE),
```

```
                        "gen_factor" = unique(emm_df1$gen_factor))

# Use emms from aov.test2
E_matrix <- data.frame("E_means" = tapply(emm_df2$emmean, emm_df2$exp_env_factor, mean, na.rm=TRUE),
                       "exp_env_factor" = unique(emm_df2$exp_env_factor))
```

To match each genotypic mean with the correct environmental mean, we have to ensure genotypes are correctly matched to their native environment. This is what the "I" term in the CovGE equation refers to.

Because multiple genotypes are native to the same environment, environments will be used more than once. We'll use the indicator dataset above for this

```
# Next, we match up and reorder G-matrix and E-matrix to match the correct native environments
Cov_matrix = G_matrix
Cov_matrix$exp_env_factor = indicator$nat_env_factor[match(G_matrix$gen_factor,indicator$gen_factor)] #
Cov_matrix$E_means = E_matrix$E_means[match(Cov_matrix$exp_env_factor,E_matrix$exp_env_factor)]
Cov_matrix # y_i (G_means) are now correctly aligned with the y_j (E_means) of the correct native envir
```

```
##        G_means gen_factor exp_env_factor    E_means
## G_1  0.6763071        G_1            E_1 -0.3908772
## G_2  0.1132500        G_2            E_1 -0.3908772
## G_3  0.9576796        G_3            E_1 -0.3908772
## G_4  0.5372695        G_4            E_1 -0.3908772
## G_5 -0.2271096        G_5            E_2  0.4342528
## G_6 -1.4236659        G_6            E_2  0.4342528
## G_7  0.2498298        G_7            E_2  0.4342528
## G_8 -0.7100579        G_8            E_2  0.4342528
```

Because G_1, G_2, G_3, and G_4 are all native to the same environment, they share the same E_mean (ybar_j). Similarly, because G_5, G_6, G_7, and G_8 are all native to the same environment (freshwater environment), they likewise share the same E_mean (ybar_j).

Now we have the estimated marginal means and they are formatted properly, we can calculate CovGE.

```
N = length(Cov_matrix$gen_factor) # Length of number of independent genotypes/populations
overallmean = mean(c(Cov_matrix$G_means,Cov_matrix$E_means),na.rm=TRUE) # ybar (overall estimated margi
numerator = sum((Cov_matrix$G_means - overallmean)*(Cov_matrix$E_means - overallmean)) # Follows Numera

standardize_max = max(var(Cov_matrix$E_means),var(Cov_matrix$G_means)) # standardize CovGE by max varia
CovGE = (1/(N-1))*(numerator/standardize_max)
CovGE
```

```
## [1] -0.4200424
```

Bootstrapped Confidence Intervals

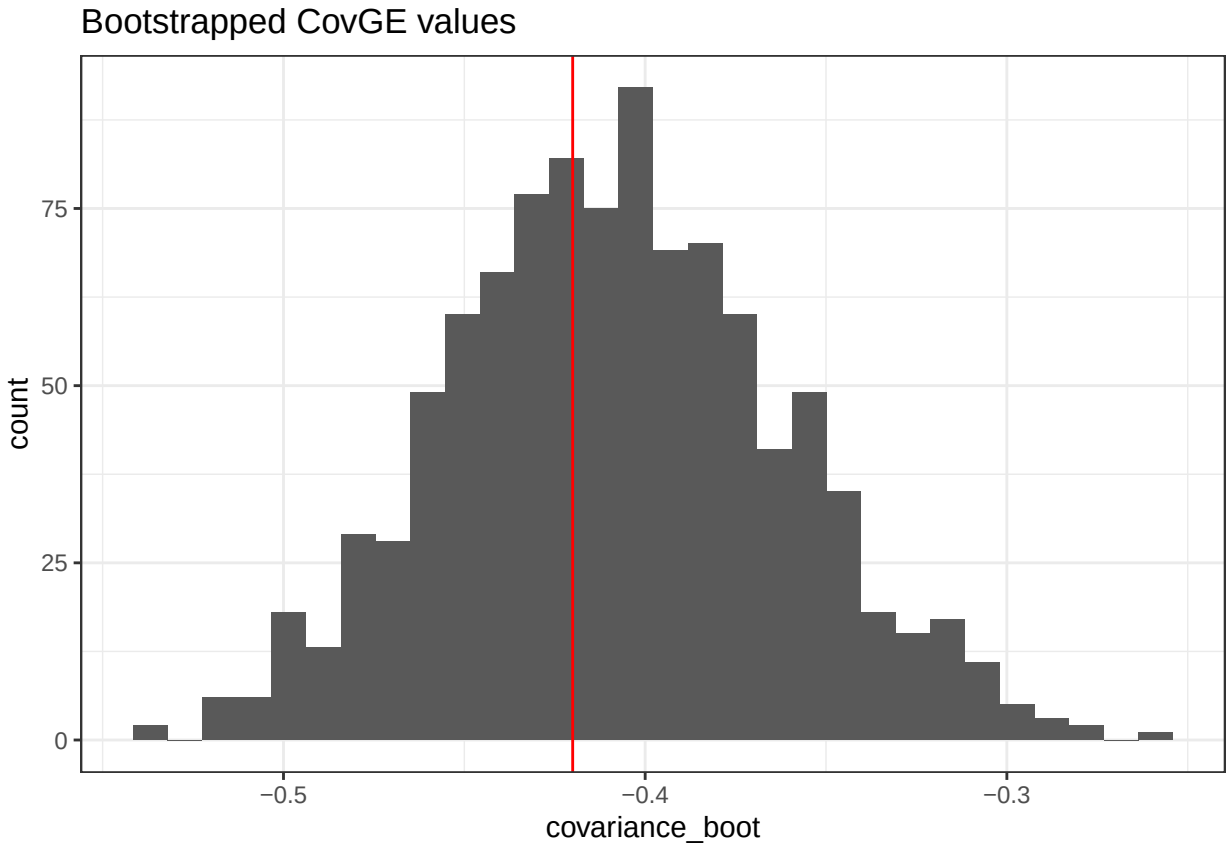
To estimate confidence intervals, we use bootstrapping, in which we shuffle phenotype within each genotype/environment and recalculate covGE after each reshuffle. This generates a distribution of CovGE estimates that form 95% confidence intervals.

For this step, I am going to use imported functions that do the same thing as shown above. "bootstrap_raw" function shuffles the raw datadata, "mod.GxE" function compiles the cov_matrix dataframe "cov.function" function calculates CovGE

** Important **: `cov.function()` and `mod.Cov()` requires input on whether the design is balanced or imbalanced. If using the balanced data, use `balanced = TRUE`; if using imbalanced data, use `balanced = FALSE`

```r
n_boot <- 999
balanced = TRUE # Denotes imbalance in number of genotypes needed for functions

boot_dat_raw = boot_df_raw = data.frame()

for(i in 1:n_boot){

  # Shuffle Data
  data$phen_corrected = data$phen # These functions use phenotypic data as phen_corrected...
  shuffle_dat <- bootstrap_raw(data)

  # Anova model fit & GxE estimates
  m2 <- mod.Cov(shuffle_dat, balanced) # Insert shuffled raw phenotype dataframe

  # Pull info from mod.GxE output
  cov_matrix_boot <- m2[[1]]

  # Covariance Estimates
  cov_corrected_boot = round(cov.function(cov_matrix_boot,balanced),3)

  # Bootstrap dataframe
  boot_dat_raw <- data.frame("covariance_boot" = cov_corrected_boot)
  boot_df_raw <- rbind(boot_df_raw,boot_dat_raw)
}

# Check: Histograms of distribution around CovGE - Should be normally distributed around red line (CovG
ggplot(boot_df_raw, aes(x = covariance_boot), alpha = 0.5) + geom_histogram() + geom_vline(aes(xinterce
  ggtitle("Bootstrapped CovGE values") + theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Covariance Confidence Intervals
cov_CI = quantile(boot_df_raw$covariance_boot, probs=c(0.025, 0.975), type=1)
cov_CI
```

```
##   2.5%  97.5%
## -0.498 -0.315
```

Hypothesis Testing using Permutation

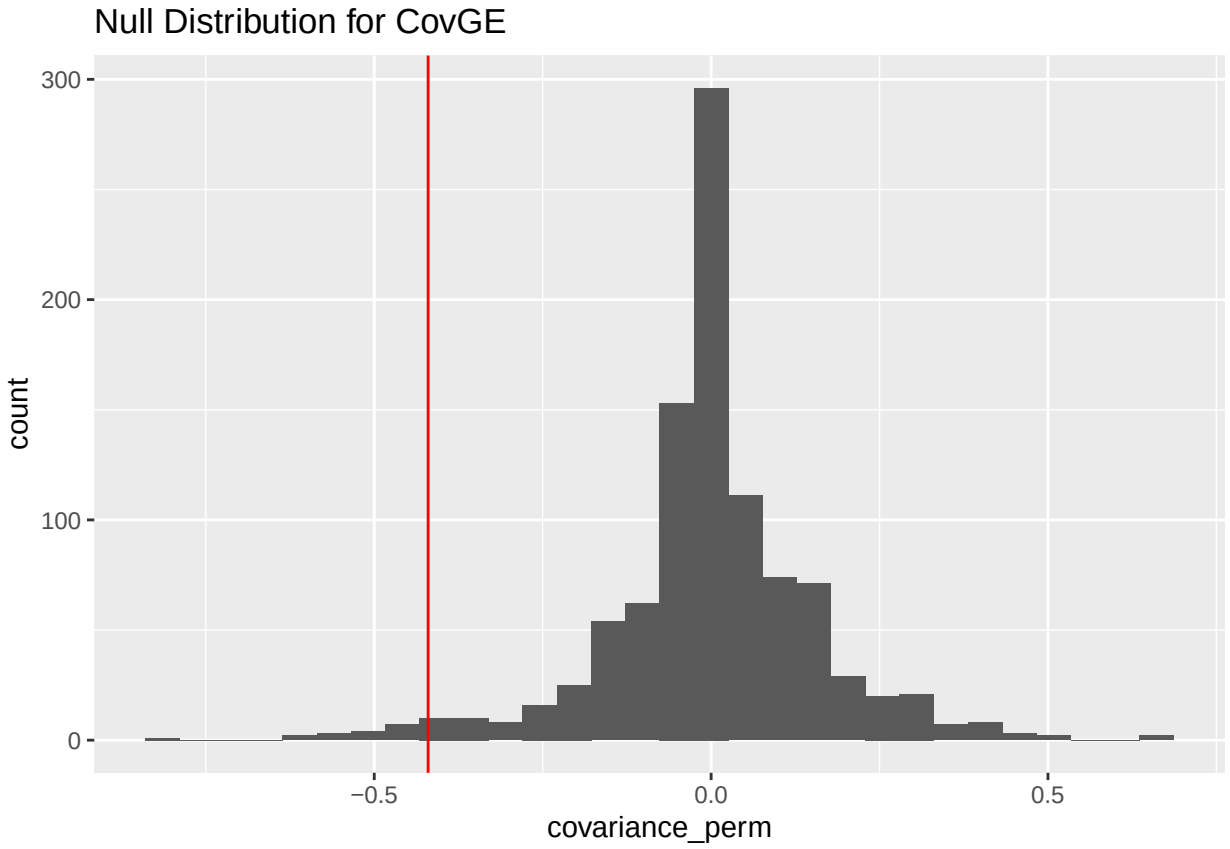
For hypothesis testing, we used permutation. Permutation shuffles phenotypic data but does not resample or maintain the genotypic/environmental levels. As a result, it creates a distribution around the null expectation that CovGE = 0. If the CovGE estimate is outside of the tails of this null distribution, it is considered statistically significant.

Again, I will used some canned functions that simply speed up the above CovGE calculation. `permutation_raw()` permutes data. `mod.Cov()` again takes those permuted data and generates Cov_matrix dataframes `cov.function()` calculates CovGE of permuted data

```
# Output dataframe
perm_df_raw = perm_dat_raw = data.frame()

for(i in 1:n_boot){

  # Resample Data
  perm_dat <- permutation_raw(data)

  # Anova model fit & GxE estimates
  m3 <- mod.Cov(perm_dat, balanced) # Insert permuted data
```

```r
  # GxE Estimates
  cov_matrix_perm <- m3[[1]]

  # Covariance Estimates
  cov_corrected_perm = round(cov.function(cov_matrix_perm,balanced),3)

  # Permutation dataframe
  perm_dat_raw <- data.frame("covariance_perm" = cov_corrected_perm)
  perm_df_raw <- rbind(perm_df_raw,perm_dat_raw)
}

# Check: Permutation histogram - should be normally distributed around zero - if red line is outside di
ggplot(perm_df_raw, aes(x = covariance_perm), alpha = 0.5)+ geom_histogram() +
  geom_vline(aes(xintercept = CovGE),color = "red") + ggtitle("Null Distribution for CovGE")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Now to calculate the p-value from this null distribution:

```r
# Covariance P-values (proportion of values that fall outside the estimated covGE value)
cov_pvalue <- sum(abs(perm_df_raw$covariance_perm) >= abs(CovGE))/(n_boot+1) # Two-tailed
cov_pvalue
```

```
## [1] 0.026
```

Thus completes the step-by-step tutorial on how CovGE, 95% confidence intervals, and the P-value are estimated for phenotypic data. From these results, we can infer this simulated dataset exhibits significant counter gradient variation.

Single Function

For faster implementation, we have a single function that performs all these individual pieces as follows:

Note the data formatting requirements listed below.

```
balanced = TRUE # Change to FALSE if using unbalanced data
n_boot = 999
data_type = "raw" # Can estimate covariance on "means" data too, but that is not applicable to this tut
data$phen_corrected = data$phen

# Important: for this to work, columns on dataset must be renamed to the following and formatted into f
#  Genotype/Population = "gen_factor"
#  Experimental Environment = "exp_env_factor"
#  Native Environment = "nat_env_factor"
#  Phenotypic data = "phen_corrected"

singularity = covariance.test(data, n_boot, data_type, balanced) # Data, Number of bootstraps, data_typ
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
singularity # Note that this function can take upwards of 3-5 minutes if running all 999 bootstraps - w
```

```
##   Covariance.Estimate Covariance.Lower.CI Covariance.Upper.CI
## 1               -0.42              -0.494              -0.305
##   Covariance.p.value GxE.Estimate GxE.p.value
## 1              0.034    0.3619601       0.999
```