Supplementary Method 1:

Illustration of the analytical pipeline of RB-TnSeq analysis and identification of interaction fitness

This document provides examples of the different steps of the computational analysis of RB-TnSeq data and identification of interaction fitness. They rely on 3 principal custom R scripts.

Script 1 (Script1_GeneFitness_Replicate.Rmd): Calculates gene fitness for of one replicate of a set of RB-TnSeq experiments of the same library and relying on the same T0 sample. Therefore, this script has to be run independently for each replicate of RB-TnSeq experiment using the same T0. In this exemple, we use 5 RB-TnSeq experiments (Condition 1 to 5) sharing the same T0 and performed in triplicate. While this script has to be run for each replicate independently, here, we only display the run for the first replicate to illustrate the script.

Script 2 (Script2_Averaging_Replicates.Rmd): Averages gene fitness values across replicates.

To run this script, you need the output of Script 1 for each replicate of the experiment. Then the script combines all the replicates of the 5 example conditions.

Note: Even if we only display the Script 1 run for the first replicate, we also run Script 1 for the second and third replicate of this example as required to run Script 2.

Script 3 (Script3_2conditions_Fitness_Comparison.Rmd): Performs the gene fitness comparisons between a given reference condition and the other conditions and identifies interaction fitness.

Here, we used the run that compared the final gene fitness of Conditions 2 to 5 to gene fitness of reference condition, Condition 1

STEP 1:Gene fitness values and associated variance for 1 replicate of a set of RB-TnSeq experiments (Same T0)

Script: Script1_GeneFitness_Replicate.Rmd

That script processes the raw counts data from the allpoolcounts.tab file generated by the perl script BarSeqTest.pl (Wetmore **et al.**, 2015). This script processes 1 biological replicate This script produces a final file containing normalized gene fitness and associated fitness variance. Plots are generated during data processing to visually follow data transformation.

Example description

This run illustrates the first step of RB-TnSeq analysis to identify interaction fitness. This is the run for the first replicate of an example comprised of 5 RB-TnSeq experiments using the *E. coli* RB-TnSeq library.

Run

Packages and functions

```
rm(list=ls())

# Step 1: Packages

library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
library(gridExtra)
```

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

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

```
# Step 2: Sourcing functions required in that script
source("Data_prep_viz10KB.R")
source("Ref_counts_CorrandNorm.R")
source("Gene_fitness.R")
source("Loc_smoothmed_norm.R")
```

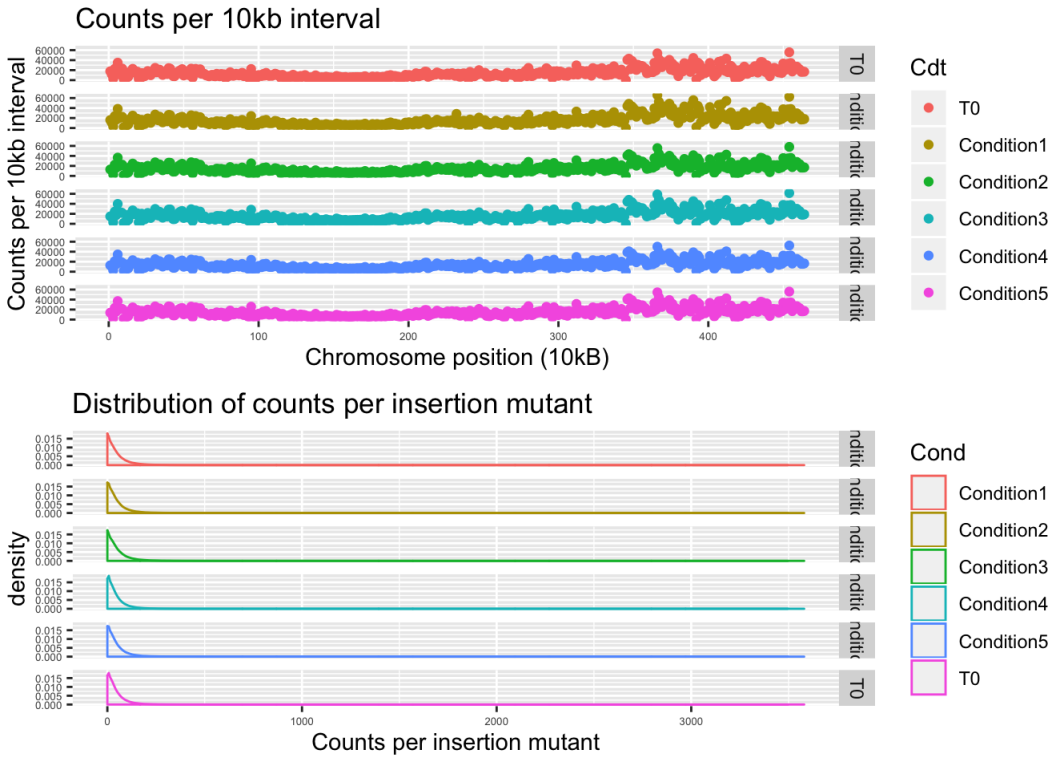
Input data and parameters set up

```
# Step 3: Parameters set up and data import

 #Different parameters have to be defined by the user:
  # org_locId: depending on your genome annotations, the locus_Id may be numerics or characters
  # cdnb: number of analyzed conditions (including T0)
  # scaffold: Set up the chromosome scaffold (for plot purposes only)
  # Indicate the locusId of the gene you will use as a reference to normalize counts

org_locId="Num"   # "Num" if numeric , "Char" if characters
cdnb=6
scaffoldX=7023
ref=c(17490,15396,14886,14220,18293) # here you write the locusId of your reference gene CAREFUL, depending on the locusId Type, it might be a numeric  or character

 # You need to import the table containing the number of counts per barcodes in each condition (allpoolcounts.tab transformed as a csv file - make sure to keep all columns)
 # The first 7 columns of your table should be: barcodes, rbarcode, scaffold, strand, pos, locusId and f, then each column should be a condition ==> THIS IS IMPORTANT THAT THE FIRST 7 COLUMNS ARE NOT COUNTS
 # ALSO COLUMN 8 MUST BE NAMED T0
 # THE CONDITIONS MUST HAVE THE SAME NAME IN EACH REPLICATE
 # You also need to import the genes.GC (as a .txt file) file used to run the perl script TestBarSeq.pl (Wetmore et al., 2015).

Data_original=read.csv("Ex_Run_R1.csv")  # import your allpoolcounts file

genes.tab <- readr::read_delim("genes.GC.txt",
                               "\t", escape_double = FALSE, trim_ws = TRUE)  # import your gene.GC file
```

```
## Parsed with column specification:
## cols(
##   locusId = col_double(),
##   sysName = col_character(),
##   type = col_double(),
##   scaffoldId = col_double(),
##   begin = col_double(),
##   end = col_double(),
##   strand = col_character(),
##   name = col_character(),
##   desc = col_character(),
##   GC = col_double(),
##   nTA = col_double()
## )
```

Original data visualization

```
# Step 4: Data pre-visualization

 # Before processing the data we represent, for each condition (T0 included) the number of counts per 10kB a
nd the distribution of number of counts per insertion mutant

Data=Data_original

    # Number of counts per 10kB

Data_prep=Data_prep_viz10KB(Data,cdnb,scaffoldX)    # format the data for vizualisation as counts per 10kB

dat_format <- data.frame(Data_prep[ncol(Data_prep)], stack(Data_prep[1:cdnb]))
colnames(dat_format)=c("Rank","Counts","Cdt")

plot1=ggplot(dat_format, aes(x=Rank,y=Counts,col=Cdt)) + geom_point(shape=19) +
  theme(axis.text.x = element_text(size=5),axis.text.y = element_text(size=5)) +
  ggtitle("Counts per 10kb interval") +
  facet_grid(Cdt ~ .) + labs(x = "Chromosome position (10kB)", y="Counts per 10kb interval")

#Note:if a couple of outliers points prevent from accuretely observing the number of counts per 10kB, you ca
n change y=Counts to y=log10(Counts)

# Distribution of number of counts per insertion mutant

Data_distr=gather(Data,"Cond","Counts",T0:ncol(Data))

plot1a=ggplot(Data_distr, aes(x=Counts,col=Cond)) + geom_density() +
  theme(axis.text.x = element_text(size=5),axis.text.y = element_text(size=5)) +
  ggtitle("Distribution of counts per insertion mutant") +
  facet_grid(Cond ~ .) + labs(x = "Counts per insertion mutant")



grid.arrange(plot1, plot1a, nrow=2)
```



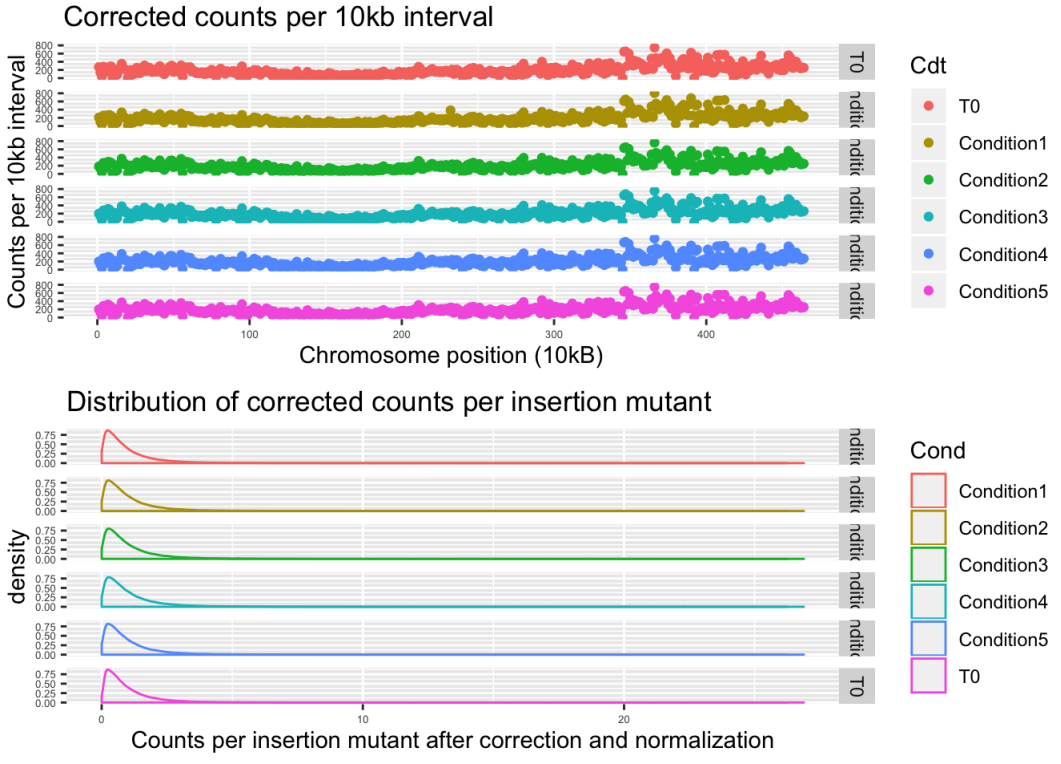
Selection of insertion mutants, followed by counts correction and normalization

Selection: raw counts have to be processed prior to fitness calculation. Insertion mutants in intergenic regions and that are located outside of the ORF (f<0.1 and f>0.9) are filtered out. Also, any mutants with a low abundance in the T0 condition are filtered out.

Count correction: a pseudocount of .1 is added to all counts to avoid counts of 0

Normalization: corrected counts are normalized by the average number of reads per insertion mutant calculated using a set of reference genes associated with neutral fitness in all tested conditions

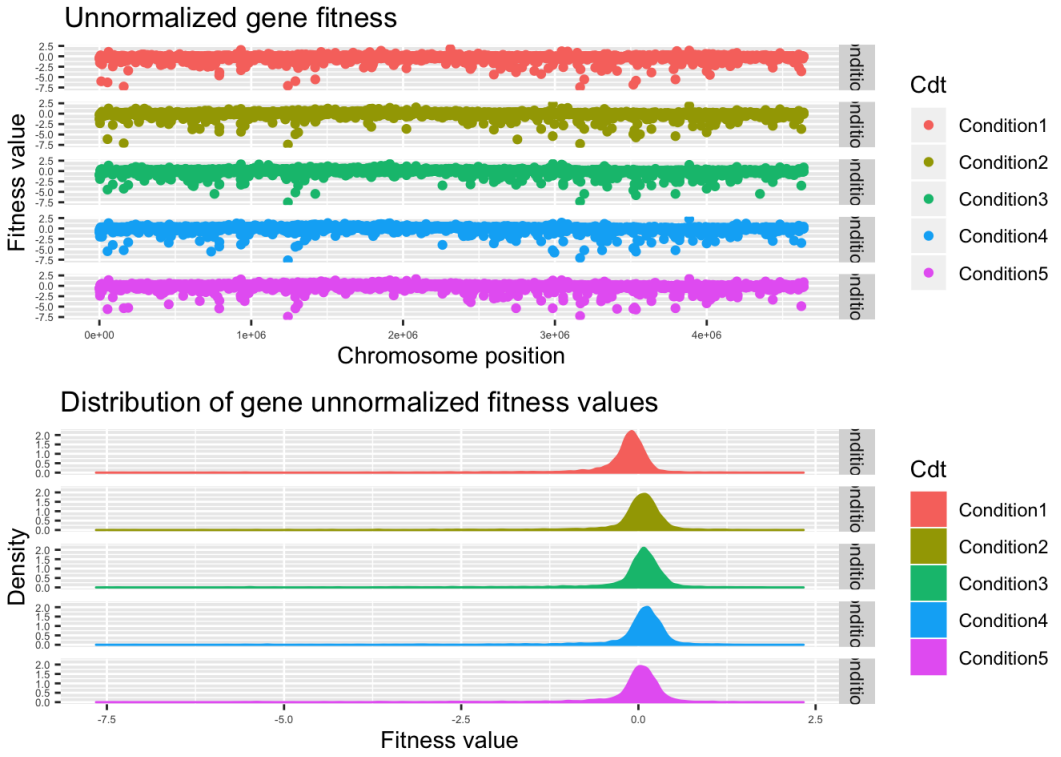
```
# Step 5: Count correction before fitness calculation

 # A pseudocount of .1 is added to each count to avoid counts of 0
 # Insertions mutants that are outside of the ORF (f<0.1 and f>0.9) are filtered out
 # Insertion mutants that do not pass the T0 count threshold (before correction relative to reference ; Coun
ts<3.1) are filtered out
 # Counts are normalized using at least one reference gene (no fitness effect in all tested conditions) - Th
e reference are used to calculate the average number of read per insertion mutant which is in turn used for
normalization
 # Data are visualized again after correction (Number of corrected counts per 10kb and Distribution of corre
cted counts)

  # The function Data_counts_CorrandNorm perform all the aformentioned modifications
Data_ref_corrected=Ref_counts_CorrandNorm(Data, ref, cdnb)

  # Number of corrected counts per 10kB
Dat_viz=Data_prep_viz10KB(Data_ref_corrected,cdnb,scaffoldX)
dat_format <- data.frame(Dat_viz[ncol(Dat_viz)], stack(Dat_viz[1:cdnb]))
colnames(dat_format)=c("Rank","Counts","Cdt")

plot2=ggplot(dat_format, aes(x=Rank,y=Counts,col=Cdt)) + geom_point(shape=19) +
  theme(axis.text.x = element_text(size=5),axis.text.y = element_text(size=5)) + ggtitle("Corrected counts p
er 10kb interval") +
  facet_grid(Cdt ~ .) + labs(x = "Chromosome position (10kB)", y="Counts per 10kb interval")

#Note:if a couple of outliers points prevent from accuretely observing the number of counts per 10kB, you ca
n change y=Counts to y=log10(Counts)

  # Distribution of number of counts per insertion mutant

Data_distr=gather(Data_ref_corrected,"Cond","Counts",T0:ncol(Data_ref_corrected))

plot2a=ggplot(Data_distr, aes(x=Counts,col=Cond)) + geom_density() +
  theme(axis.text.x = element_text(size=5),axis.text.y = element_text(size=5)) +
  ggtitle("Distribution of corrected counts per insertion mutant") +
  facet_grid(Cond ~ .) + labs(x = "Counts per insertion mutant after correction and normalization")



grid.arrange(plot2, plot2a, nrow=2)
```



Calculation of gene fitness values

A gene fitness value is calculated as the average of the fitness values of associated insertion mutants. The fitness of insertion mutants is the log2 of the ratio of the insertion mutant's normalized counts in a given condition and at T0.

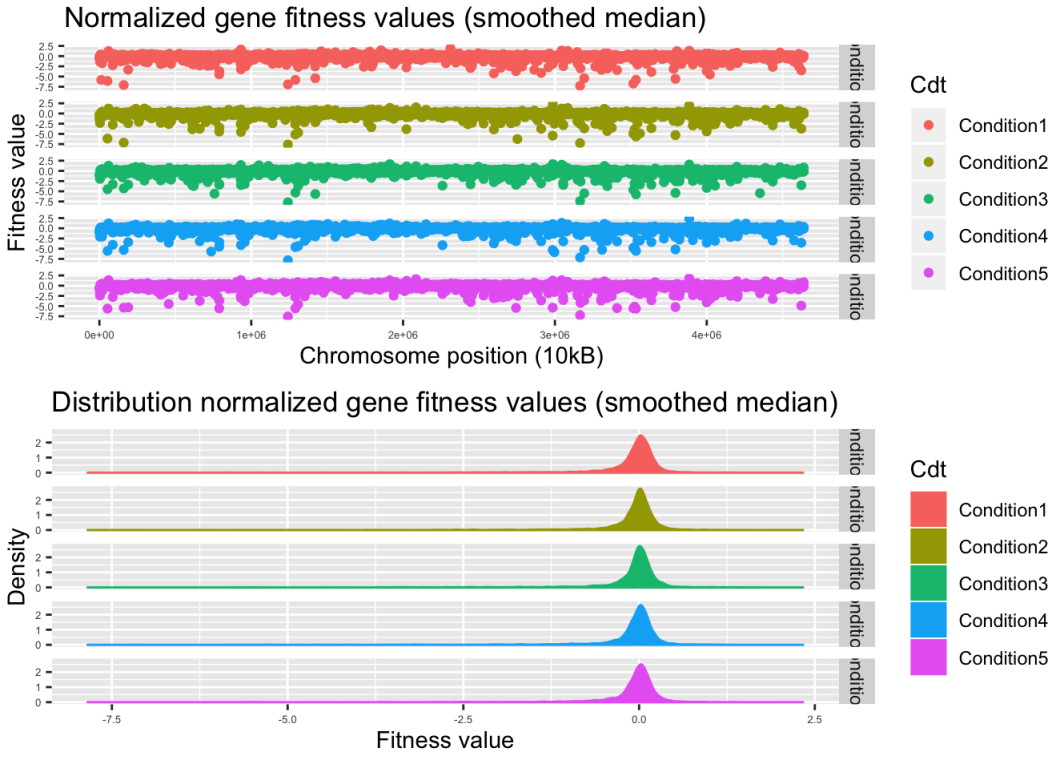
```
# Step 6: Calculation of gene fitness
 # Calculation of the strain fitness (fitness for each insertion with central insertion as the log2 of the r
atio of the corrected counts in the condition and the corrected counts in the T0)
 # Calculation of gene fitness values along with associated variance (average of all the insertion mutants f
itness in that gene)
 # Vizualization of the gene fitness values (along the chromose (plot3) or as a distribution (plot4))

Data_for_fit=Data_ref_corrected
cdnbF=(cdnb-1)    # number of conditions for which we will get a fitness values (=everything but T0)

# The following function returns a list containing 3 tables: "Strain fitness": strain fitness values, "Gene_
fitness": unnormalized gene fitness values and "Gene_fitness_variance": variance assocaited with gene fitnes
s values

Raw_values=Gene_fitness(Data_for_fit,cdnbF,genes.tab,org_locId)

Table_GeneFitness_Raw=Raw_values[[2]]  # We extract the table with the unnormalized fitness values for data
vizualisation

 #We format the data to vizualise them
dat_format <- data.frame(Table_GeneFitness_Raw[1], Table_GeneFitness_Raw[(ncol(Table_GeneFitness_Raw)-1):nco
l(Table_GeneFitness_Raw)], stack(Table_GeneFitness_Raw[2:(2+cdnbF-1)]))
colnames(dat_format)=c("locusId","sysName","begin","RawFitness","Cdt")

plot3=ggplot(dat_format, aes(x=begin,y=RawFitness,col=Cdt)) + geom_point(shape=19) +
  theme(axis.text.x = element_text(size=5),axis.text.y = element_text(size=5)) + ggtitle("Unnormalized gene
fitness") +
  facet_grid(Cdt ~ .) + labs(x = "Chromosome position", y="Fitness value")

plot4=ggplot(dat_format, aes(x=RawFitness,col=Cdt)) + geom_density(aes(fill=Cdt)) +
  theme(axis.text.x = element_text(size=5),axis.text.y = element_text(size=5))+ ggtitle("Distribution of gen
e unnormalized fitness values") +
  facet_grid(Cdt ~ .) + labs(x = "Fitness value", y="Density")


grid.arrange(plot3, plot4, ncol=1)
```



Chromosome position normalization of gene fitness values

As described in Wetmore *et al.*, 2015, gene fitness values are normalized based on the gene location on the chromosome using the smoothed median.

```
# Step 7: Gene fitness normalization
 # Normalize gene fitness based on the position of the gene on the chromosome (smooth median normalization ; Wetmore et al., 2015)
 # Data visualization like Step 6

Data_to_norm=Table_GeneFitness_Raw
Data_norm_loc=Loc_smoothmed_norm(Data_to_norm,genes.tab,cdnbF,org_locId)  # we obtain the gene fitness values normalized by the smoothed median ==> normalization for gene location


 # Data visualization
dat_format <- data.frame(Data_norm_loc[1:4], stack(Data_norm_loc[5:ncol(Data_norm_loc)]))
colnames(dat_format)=c("locusId","sysName","begin","scaffold","NormFitness","Cdt")
dat_format$begin=as.numeric(dat_format$begin)
dat_format$NormFitness=as.numeric(dat_format$NormFitness)

plot5=ggplot(dat_format, aes(x=begin, y=NormFitness, col=Cdt)) + geom_point(shape=19)+
  theme(axis.text.x = element_text(size=5),axis.text.y = element_text(size=5)) + ggtitle("Normalized gene fitness values (smoothed median)") +
  facet_grid(Cdt ~ .) + labs(x = "Chromosome position (10kB)", y="Fitness value")

plot6=ggplot(dat_format, aes(x=NormFitness, col=Cdt)) + geom_density(aes(fill=Cdt)) +
  theme(axis.text.x = element_text(size=5),axis.text.y = element_text(size=5)) + ggtitle("Distribution normalized gene fitness values (smoothed median)") + facet_grid(Cdt ~ .) + labs(y = "Density", x="Fitness value")

grid.arrange(plot5, plot6, ncol=1)
```



Saving data

```
# Step 8: Data formating and save

  # Data formating

Data_Fitness_Replicate=dat_format

Table_GeneFitness_VAR=Raw_values[[3]]
Data_Fitness_VAR=data.frame(Table_GeneFitness_VAR[1],
                            stack(Table_GeneFitness_VAR[2:ncol(Table_GeneFitness_VAR)]))
colnames(Data_Fitness_VAR)=c("locusId","Fitness_Variance","Cdt")


if (org_locId=="Num"){
  Data_Fitness_Replicate$locusId=as.numeric(Data_Fitness_Replicate$locusId)
}

All_data_Replicate=left_join(Data_Fitness_Replicate,Data_Fitness_VAR,by=c("locusId","Cdt"))
All_data_Replicate=All_data_Replicate[c(1,2,3,4,6,5,7)]


save(All_data_Replicate, Data_norm_loc,
     Raw_values, Data_ref_corrected, Data_original, genes.tab,
     file="Ex_Run_R1.RData")
```

STEP 2 :Gene fitness calculation: averaging across replicates

Script: Script2_Averaging_Replicates.Rmd

That second part of the analysis averages gene fitness across replicates. If all studied conditions have the same T0 sample, it requires for each replicate the .RData files generated in the first part of the analysis “Script1_GeneFitness_Replicate.Rmd”. If studied conditions have a different T0 sample, it requires the output of the script Multiple_T0s.R Before averaging replicates, we perform a quick analysis of correlation between replicates of the same condition. It produces a final file containing final normalized gene fitness and associated fitness variance across replicates. Plots are generated during data processing to visually follow data transformation.

Example description

This run illustrates the second step of RB-TnSeq analysis to identify interaction fitness. This is the run for averaging fitness values for all the replicates of the 5 conditions RB-TnSeq example.

Run

Packages and functions

```
rm(list=ls())

library(dplyr)
library(ggplot2)
library(ggrepel)
library(DescTools)
library(gridExtra)

source("Correlation_Rep.R")
source("Weighted_average.R")
```

Data upload and parameters set up

```
# Step1: Parameters set up and Data import
 # Import .Rdata files generated for each replicate in Script1_GeneFitness_Replicate.Rmd
 # Isolate and rename All_data_Replicate just after loading to avoid overwritting.
 # Note: if conditions have different T0s and have been processed independently in "Gene_Fitness_Replicate.R
", Step 1 is replaced by running the script: "Multiple_T0s.R"


  # Parameter set up
org_locId="Num"   # "Num" if numeric , "Char" if characters
multiT=FALSE  # Switch to TRUE if you used different T0s for a set of conditions and have to run Multiple_T0
to generate the table containing all replicates.

if (multiT==FALSE){
  load("Ex_Run_R1.Rdata") #load the .Rdata file from Gene_Fitness_Replicate.R for replicate 1
  Replicate1=All_data_Replicate

  load("Ex_Run_R2.Rdata") #load the .Rdata file from Gene_Fitness_Replicate.R for replicate 2
  Replicate2=All_data_Replicate

  load("Ex_Run_R3.Rdata") #load the .Rdata file from Gene_Fitness_Replicate.R for replicate 3
  Replicate3=All_data_Replicate

  # We bind all replicates together and add a column "Rep" to identify were it is coming from

  Replicate1$Rep="R1"
  Replicate2$Rep="R2"
  Replicate3$Rep="R3"

  AllReplicate=rbind(Replicate1,Replicate2,Replicate3)
  head(AllReplicate, n=5)
}
```

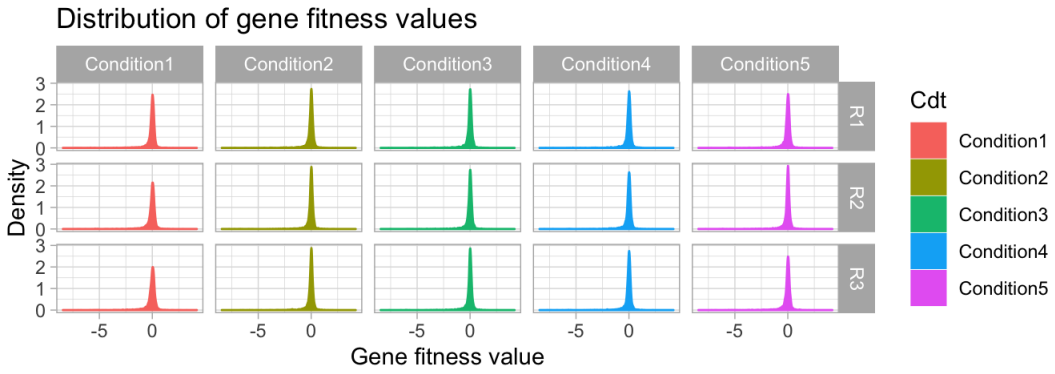
```
##   locusId sysName begin scaffold        Cdt NormFitness Fitness_Variance Rep
## 1   14146   b0001   190     7023 Condition1 -0.10763776      0.004234506  R1
## 2   14147   b0002   337     7023 Condition1 -0.39564492      0.158219137  R1
## 3   14148   b0003  2801     7023 Condition1 -0.60484100      0.198116180  R1
## 4   14149   b0004  3734     7023 Condition1 -1.37486386      0.416450536  R1
## 5   14150   b0005  5234     7023 Condition1  0.06967739      0.054488610  R1
```

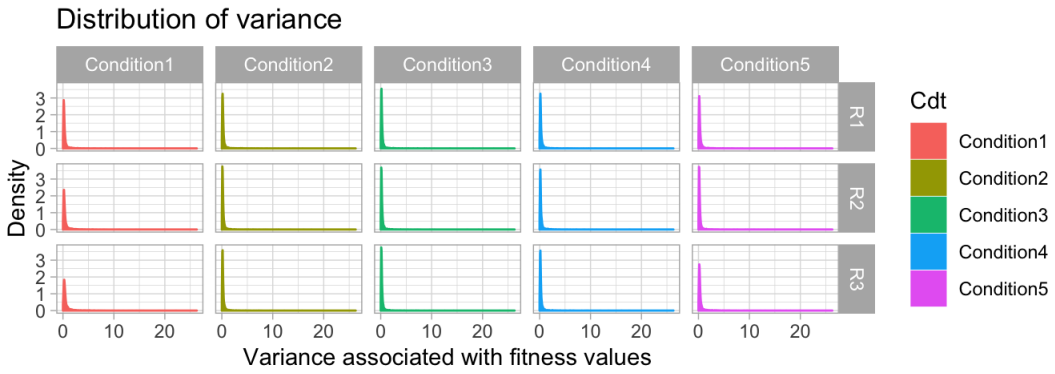
```
if (multiT==TRUE){
  load(".Rdata")    #Generated in Multiple_T0s.R
  genes.tab <- readr::read_delim("genes.GC.txt",
                                 "\t", escape_double = FALSE, trim_ws = TRUE)  # import your gene.GC file
}
```

Side de by side Replicate visualization

```
# Step 2: Replicates visualization

  # Plots for fitness values or variance distribution in each condition and replicate
plot1_fit=ggplot(AllReplicate, aes(x=NormFitness,col=Cdt))+geom_density(aes(fill=Cdt)) + theme_light()+
  ggtitle("Distribution of gene fitness values") + facet_grid(Rep ~ Cdt) + labs(x = "Gene fitness value", y=
"Density")

plot1_var=ggplot(AllReplicate, aes(x=Fitness_Variance,col=Cdt))+geom_density(aes(fill=Cdt)) + theme_light()+
  ggtitle("Distribution of variance") + facet_grid(Rep ~ Cdt) + labs(x = "Variance associated with fitness v
alues", y="Density")

grid.arrange(plot1_fit, plot1_var, ncol=1)
```

```
## Warning: Removed 2145 rows containing non-finite values (stat_density).
```





Replicates correlation analysis

Note for this example: while the script generates plots for each correlation analysis (Pearson, Spearman and Lin), here we only display the plot for the Pearson correlation

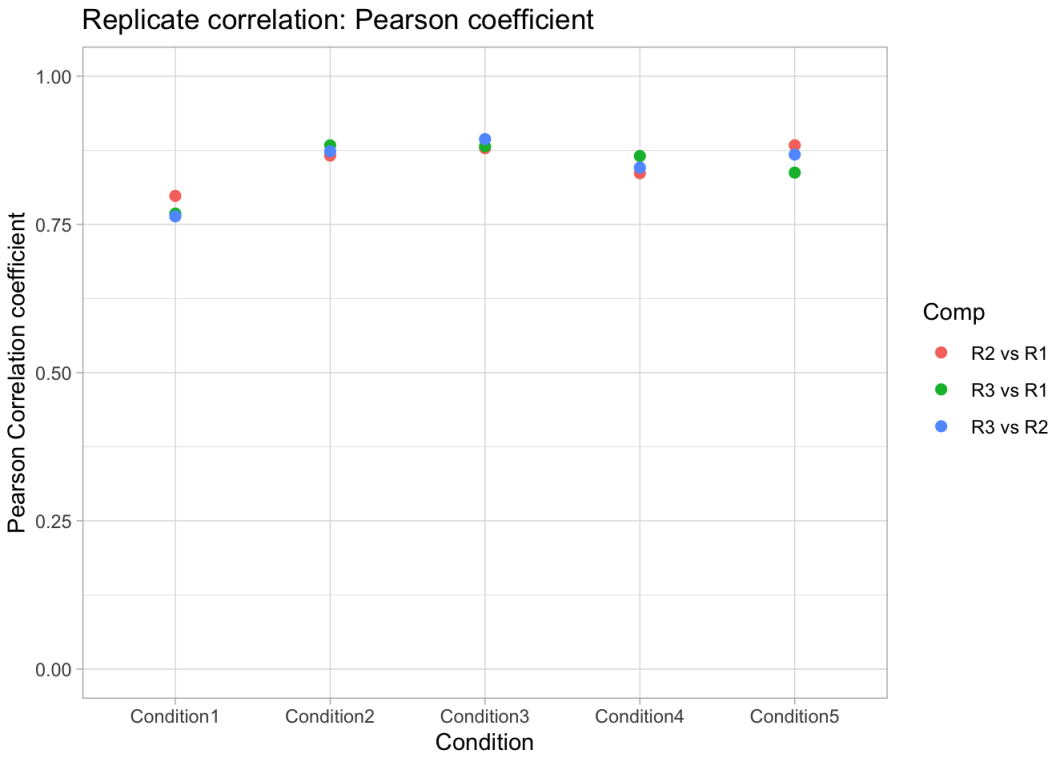
```
# Step 3: Replicate correlation calculation and visualization (for gene fitness values)
  # Determines the correlation between each replicate, stores them in a matrix
  # Visualizes correlation in two different ways: (i)  usual correlation plots and (ii) distribution of corr
elation across all conditions and replicates

  # Calculation of correlation for all pairs of replicate (and each condition) + plots
Correlation_table_fit=Correlation_Rep(AllReplicate) # Calculates different correlation coefficient + save pl
ots
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
Correlation_table_fit
```

```
## # A tibble: 15 x 5
##    Cond       Comp     P_Rsquared S_Rsquared L_Rsquared
##    <chr>      <chr>         <dbl>      <dbl>      <dbl>
##  1 Condition1 R2 vs R1      0.798      0.591      0.797
##  2 Condition1 R3 vs R1      0.768      0.572      0.763
##  3 Condition1 R3 vs R2      0.764      0.553      0.762
##  4 Condition2 R2 vs R1      0.866      0.696      0.865
##  5 Condition2 R3 vs R1      0.883      0.689      0.881
##  6 Condition2 R3 vs R2      0.873      0.711      0.873
##  7 Condition3 R2 vs R1      0.879      0.704      0.878
##  8 Condition3 R3 vs R1      0.882      0.703      0.881
##  9 Condition3 R3 vs R2      0.894      0.725      0.894
## 10 Condition4 R2 vs R1      0.836      0.669      0.835
## 11 Condition4 R3 vs R1      0.865      0.667      0.865
## 12 Condition4 R3 vs R2      0.846      0.687      0.845
## 13 Condition5 R2 vs R1      0.883      0.704      0.883
## 14 Condition5 R3 vs R1      0.837      0.652      0.836
## 15 Condition5 R3 vs R2      0.868      0.676      0.864
```

```
plot_allPcor_fit=ggplot(Correlation_table_fit, aes(x=Cond,y=P_Rsquared)) + geom_point(size=2, aes(col=Comp))
+
  theme_light() + labs(y = "Pearson Correlation coefficient", x="Condition") + ylim(0,1) +  ggtitle("Replica
te correlation: Pearson coefficient")

plot_allScor_fit=ggplot(Correlation_table_fit, aes(x=Cond,y=P_Rsquared)) + geom_point(size=2, aes(col=Comp))
+
  theme_light() + labs(y = "Spearman Correlation coefficient", x="Condition") + ylim(0,1) + ggtitle("Replica
te correlation: Spearman coefficient")

plot_allLcor_fit=ggplot(Correlation_table_fit, aes(x=Cond,y=P_Rsquared)) + geom_point(size=2, aes(col=Comp))
+
  theme_light() + labs(y = "Lin's Correlation coefficient", x="Condition") + ylim(0,1) + ggtitle("Replicate
correlation: Lin coefficient")

grid.arrange(plot_allPcor_fit, nrow=1)
```

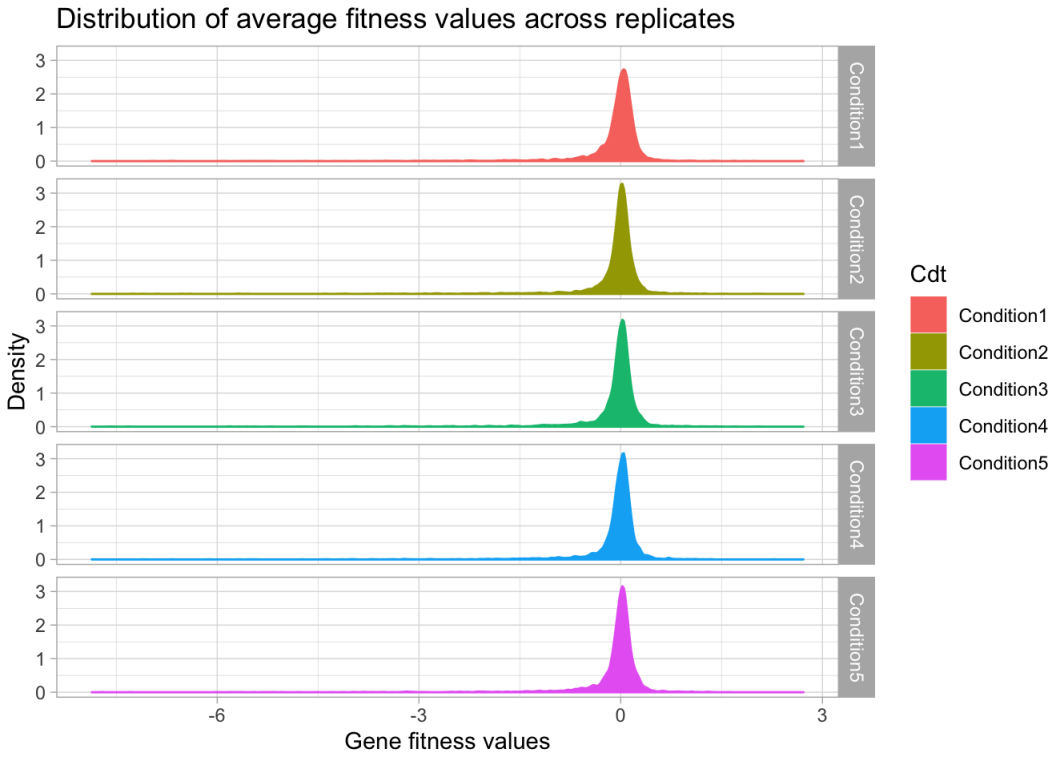


Averaging replicates

Gene fitness values are averaged across replicates using the inverse-variance weighted average. Associated squared standard error (var) and associated standard deviations are also calculated.

Note for this example: while the script generates plots for the average gene fitness, a plot for gene fitness variance and a plot for gene fitness standard deviation, here we only display the plot for the average fitness

```
# Step 4: Weighted average of gene fitness across replicate
 # Averages gene fitness values across replicates for each condition
 # Visualizes fitness values distrbutions, squared standard error (var) ditributions and standard deviation
distributions

Average_fitness=Weighted_average(AllReplicate, org_locId)

plot2_fit=ggplot(Average_fitness, aes(x=WeightedFit,col=Cdt))+geom_density(aes(fill=Cdt)) + theme_light()+
  ggtitle("Distribution of average fitness values across replicates") + facet_grid(Cdt ~ .) + labs(x = "Gene
fitness values", y="Density")

plot2_var=ggplot(Average_fitness, aes(x=WeightedVar,col=Cdt))+geom_density(aes(fill=Cdt)) + theme_light()+
  ggtitle("Distribution of squared standard error") + facet_grid(Cdt ~ .) + labs(x = "Squared standard error
associated with fitness values", y="Density")

plot2_sd=ggplot(Average_fitness, aes(x=Weightedsdev,col=Cdt))+geom_density(aes(fill=Cdt)) + theme_light()+
  ggtitle("Distribution of standard deviation") + facet_grid(Cdt ~ .) + labs(x = "Standard deviation associa
ted with fitness values", y="Density")

grid.arrange(plot2_fit, ncol=1)
```



Saving data

```
# Step5: Saving data
if(org_locId=="Num"){
  Final_gene_Fitness=left_join(genes.tab,Average_fitness, by=c("locusId")) %>% select(-c(type,strand,GC,nTA)
)
  Final_gene_Fitness=na.omit(Final_gene_Fitness)
}

if(org_locId=="Char"){
  Average_fitness$locusId=as.character(Average_fitness$locusId)  # the locusId in the Mean table are factors
, we need to turn then into charcater for the left_join
  Final_gene_Fitness=left_join(genes.tab,Average_fitness, by=c("locusId")) %>% select(-c(type,strand,GC,nTA)
)
  Final_gene_Fitness$name="No_name"   #replace the NA by something else, otherwise the next NAomit removes e
verything
  Final_gene_Fitness=na.omit(Final_gene_Fitness)
}

write.csv(Final_gene_Fitness,"All_Fitness_Values_Exemple.csv")

save(Final_gene_Fitness, Average_fitness, genes.tab,
     Correlation_table_fit,AllReplicate, file="All_Fitness_Values_Exemple.RData")
```

STEP 3: Comparison of fitness values and identification of interaction fitness

Script: Script3_2condidtions_FitnessComparison.Rmd

Compares gene fitness values of all conditions against a chosen reference condition. Appends a “Category” to each compared gene to indicate if fitness values are significantly different or not based on chosen statistical criteria.

Example description

This run illustrates the third step of RB-TnSeq analysis to identify interaction fitness. This is the run that compares gene fitness values for E. coli growth in Conditions 2 to 5 versus Condition 1.

Run

Packages and functions

```
rm(list=ls())

# Package and functions upload

library(ggplot2)
library(dplyr)
library(tidyr)
library(gridExtra)

source("Category_definition.R")
source("Comparison_test.R")
```

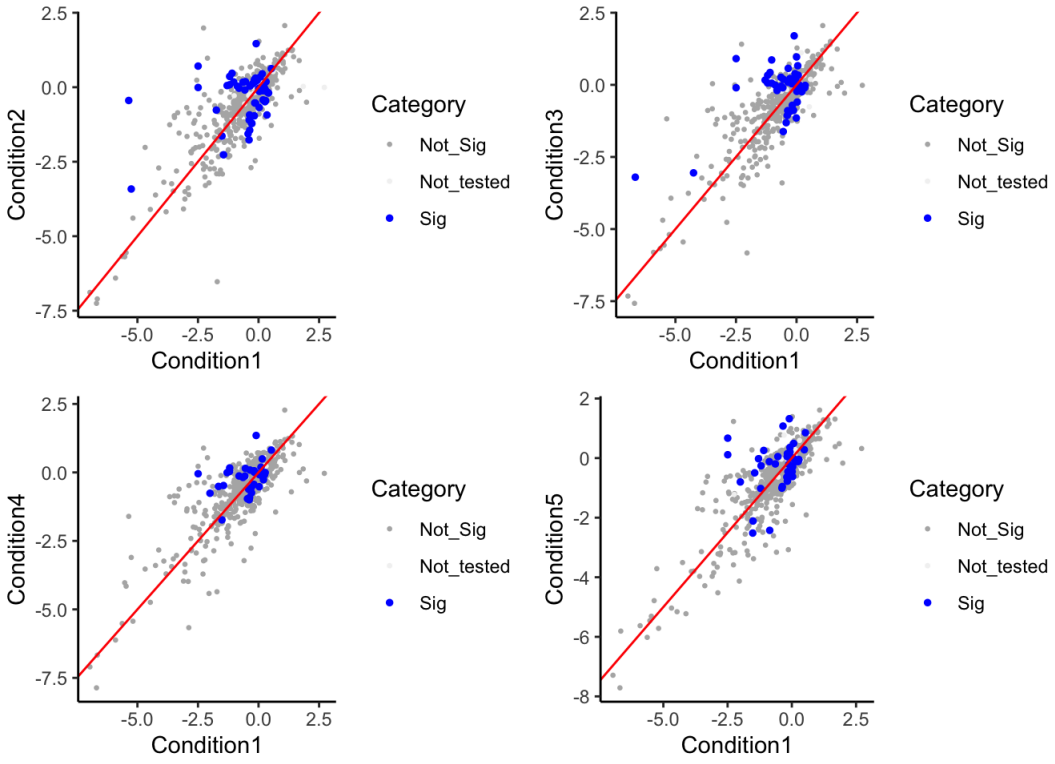
Data upload and parameters set up

```
# Step 1: Data import and parameters settings
  # Import the .Rdata file generated in Averaging_replicates.R
  # Set up org_locId
  # Set up your condition of reference (has to be one of your conditions)
  # Set up your alpha value for the T-test
  # Set up whether you want to performed correction for multiple comparison testing and screen on adjusted-p
value

load("All_Fitness_Values_Exemple.RData")   # here you upload the .RData ouput of Averaging_replicates.R
Data_Fitness=Final_gene_Fitness

org_locId="Num"  # Again, you set up your organisms whether "Ecoli" if the locusId are in numeric form or "P
seudo" if the locusId are in character form
Condition1="Condition1"  #here you write the name of one of the 2 conditions you want to compare. Make sure
it is the same name than previously used
alphaF=0.002    #here you choose any alpha you want for the Fisher test (Test for equal variance) You can ch
oose 0.05 or 0.002.
alphaT=0.05   #here you choose any alpha you want. Just be aware that it is where you can control the amount
of false discovery you allow
multi=1    # here you decide if you want to correct for multiple comparison (method=fdr) multi = 0 ==> no co
rrection; multi=1 correction
```

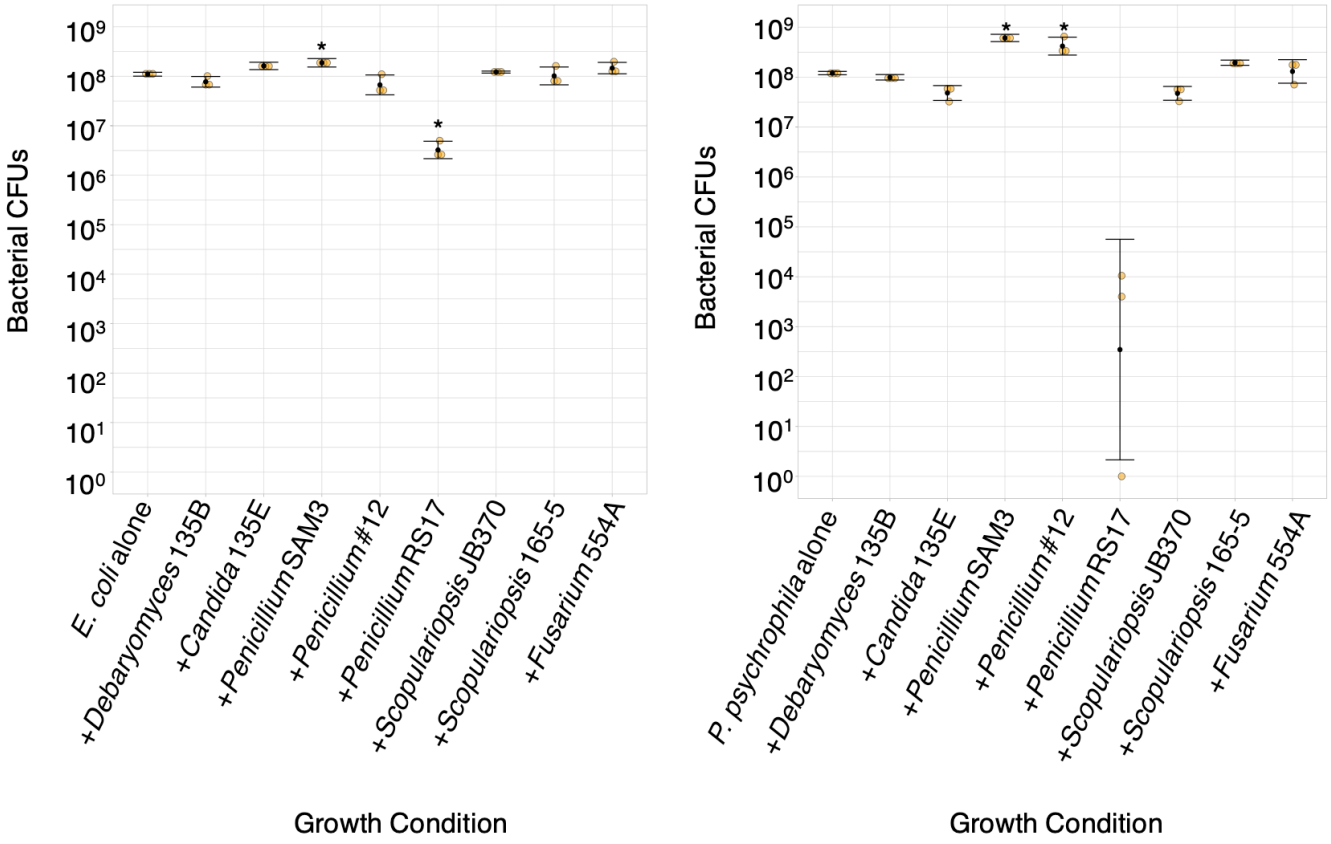
2 conditions comparison against a chosen reference condition (Condition 1)

In the following plots: “Sig” means that gene fitness values are significantly different for that gene between the compared conditions, “Not_Sig” means that fitness values are not significantly different, and “Not_tested” means that the comparison has not been performed for that gene due to unequal variances

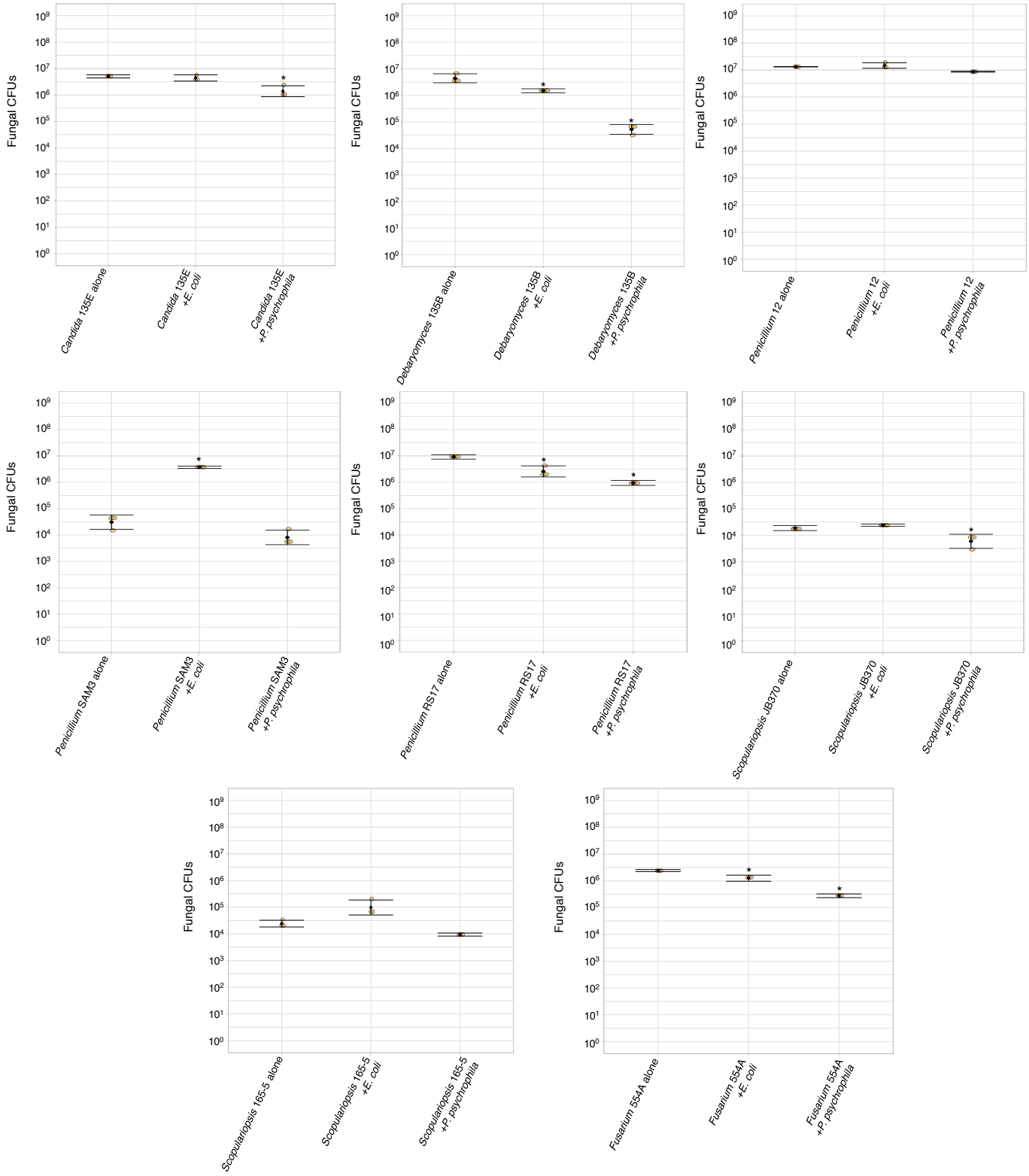
```
# Step 2: Run the comparison for all conditions against the reference one
  # Each conditions that is not the reference is going to be tested against the refence one after the other
using a for loop embedding the comparison function


Table_all=data.frame() # creates a table to eventually store the data
List_plots=list(0,0,0,0,0,0,0) # creates a list to store comparison of plots for each comparison in the "for
loop" .
                               # needs to have as many spots in the list than you have comparison

Conditions=as.vector(unique(Data_Fitness$Cdt))
Conditions=Conditions[Conditions!=Condition1]# generates a vector with all the conditions to compare against
the reference one

for (i in 1:length(Conditions)){
  Cdt=Conditions[i]
  Condition2=paste(Cdt)
  Test1=Comparison_test(Data_Fitness,Condition1, Condition2, alphaT, alphaF, org_locId, multi)
  #You obtain a list
  #list[[1]]= the table containing all the genes tested + fitness + variance values in both conditions + Fca
lc (Fisher value calculates for the test of variance + Tcalc (Tscore value for the student test))
  # and the category column indicates if the difference of fitness is significant (Sig), if it is not (Not_S
ig) or if the student was not performed because of unequal variances (Not_tested)
  #list[[2]]=the scatter plot representing the data

  Table_all=rbind(Table_all, Test1[[1]])
  List_plots[[i]]= Test1[[2]]
}

#Visualize all the scatter plots
grid.arrange(List_plots[[1]],List_plots[[2]],List_plots[[3]],List_plots[[4]], nrow=2,ncol=2)
```



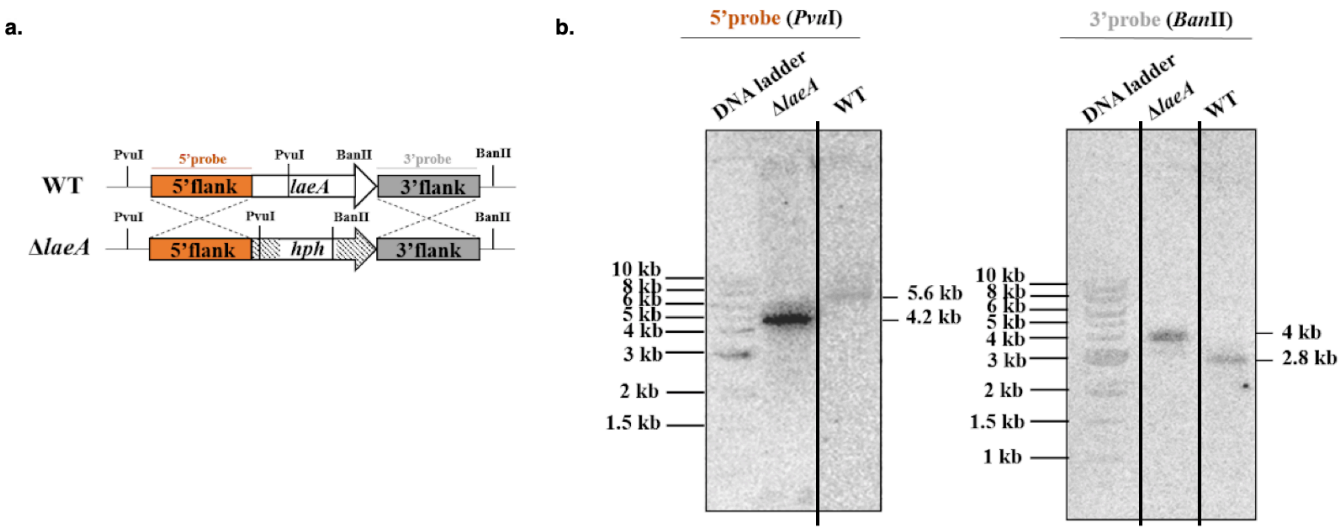

```
write.csv(Table_all, "Comparison_Ecoli_versusAlone.csv")
save(Table_all, List_plots, file="Comparison_Ecoli_versusAlone.RData")
```



Supplementary Figure 1: **Impacts of fungal species on bacterial growth after 7 days of co-culture on cheese curd agar, pH 7**. CFU: colony forming units. N=3 biologically independent samples, error bars show standard deviation and black point is the mean. Asterisks represent a significant difference in bacterial growth in the presence of the fungal partner relative to alone (two-sided Dunnett's test, p-value <0.05). Exact p-values associated with asterisks (from left to right): 0.008, 0.002, 0.002, 0.02.



Supplementary Figure 2: **Impacts of bacterial species on fungal growth after 7 days of co-culture on cheese curd agar, pH 7**. For filamentous fungi, spore counts were used as a proxy for fungal CFUs. N=3 biologically independent samples, error bars show standard deviation and black point is the mean. Asterisks represent a significant difference in fungal growth in the presence of the bacterial partner relative to alone (two-sided Dunnett's test, p-value <0.05). Exact p-values associated with asterisks (from left to right, top to bottom): 0.004, 0.022, 0.004, 7.6e-05, 0.002, 0.00043, 0.013, 0.002, 6.2e-05.



Supplementary Figure 3. Deletion of *laeA* gene in *Penicillium* sp. str. #12. **a,** Schematic representation of the genetic construct for *laeA* deletion in *Penicillium* sp. str. #12. The *hph* gene confers resistance to hygromycin. The positions of the restriction enzyme cutting sites are shown on the map. **b,** Southern blot analyses of genomic DNA from the WT and the Δ*laeA* strains. Ten micrograms of total DNA from each strain was digested with the appropriate enzymes and subjected to Southern blot analysis using respectively the 5' flank fragment (orange) and the 3'fragment (grey) as probes. The 1 kilobase DNA ladder from New England Biolabs was used to determine the size of the expected bands. The blot images were cropped to place the confirmed mutant adjacent to the WT strain. Black lines were added to the blot images to indicate where the cropping occurred. The blot images were also cropped on the top (around the wells) and bottom without interfering with the DNA ladder bands. The transformants that were confirmed to not have the correct insertion were not included in the figure. For the 3' blot image, an aligned overlay of the gel image and the blot was made allowing a clear visualization of the DNA ladder. Southern blots to confirm the Δ*laeA* strain were only performed once.