RB-TnSeq Analysis Pipeline

Read Me document

This pipeline has been designed to calculate and compare genome-wide gene fitness values across different growth conditions using RB-TnSeq (Wetmore et al., mBio May 2015, 6 (3) e00306-15). RB-TnSeq relies on the utilization of a pooled library of barcoded transposon mutants. Each mutant has integrated a single transposon that includes a unique 20-nucleotide barcode; this barcode allows tracking of individual mutants in the population. When integrated within the 10-90% of an ORF, the transposon is expected to lead to gene disruption. RB-TnSeq libraries with high genome coverage contain multiple mutants associated with the same gene. Mutant fitness is calculated by comparing the abundance of each barcoded mutant after growth to its abundance in the T0 sample (inoculation). The mutant fitness values for mutants within a single gene are then incorporated to calculate a gene fitness value. The gene fitness value indicates the importance of the gene for fitness in the studied condition.

While the current pipeline (https://bitbucket.org/berkeleylab/feba/src/master/) developed by Wetmore et al., is designed to calculate individual gene fitness values in different conditions, nothing is implemented to statistically compare gene fitness values across conditions. Therefore, starting from the allpoolcounts file generated by BarSeqTest.pl from Wetmore et al., 2015, we have developed our own pipeline to:

i) Generate normalized and comparable gene fitness values for each condition.
ii) Compare gene fitness values between two conditions.

The pipeline is designed to implement 3 biological replicates of each condition. To obtain the normalized gene fitness value, we first calculate the normalized gene fitness values for each replicate individually (Script I: Gene_Fitness_Replicate.R) and then average gene fitness across the three replicates (Script II: Averaging_Replicates.R). Then, a third script allows you to compare gene fitness values between two conditions and identify the significant differences according to a chosen threshold of confidence (Script III: 2conditions_FitnessComparison.R)

Part I: Calculation of gene fitness values for each replicate.

Associated script: Gene_Fitness_Replicate.R

For each replicate, a normalized gene fitness is obtained by

i) calculating fitness of each insertion mutant
ii) averaging the fitness of insertion mutants associated with the same gene
iii) normalizing gene fitness based on gene location on the chromosome
iv) normalizing by the mean (or the norm) of the gene fitness value distribution

Figure 1 describes the workflow of Script I as well as the associated functions.

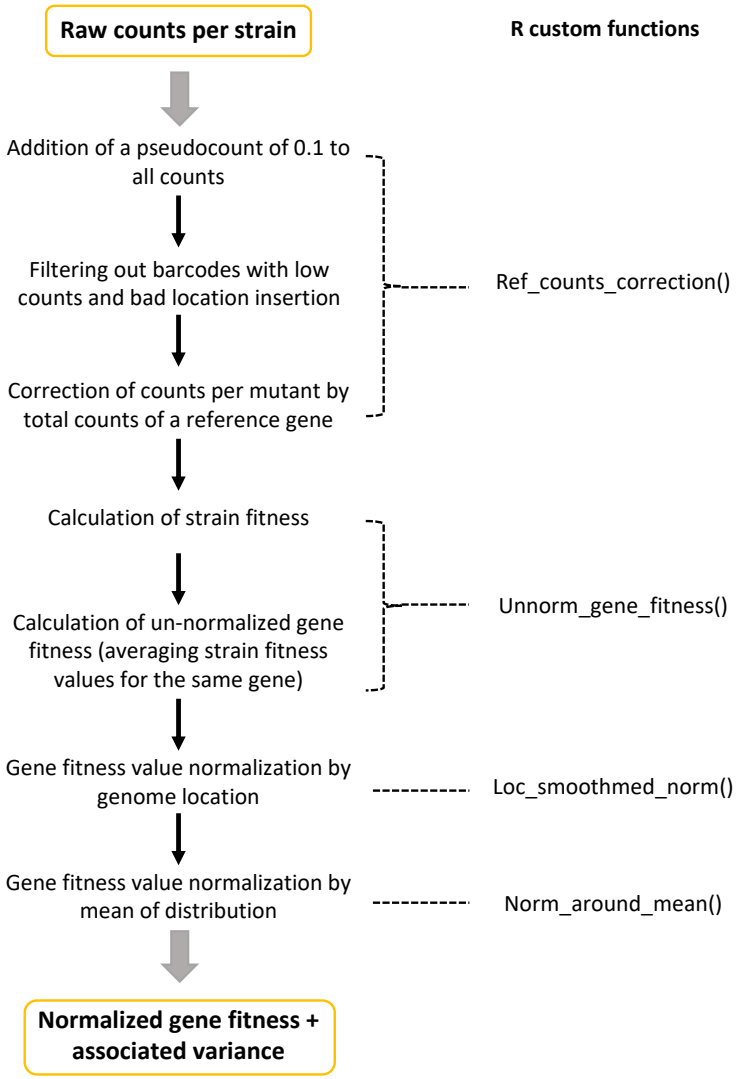

Figure 1: Workflow of gene fitness calculation per replicate

Briefly, to calculate gene fitness values, the script uses raw counts of number of reads per barcode per condition. First, as fitness calculation is based on a $\log_2$ transformation, a pseudocount of 0.1 is added to all counts to avoid any 0 values. Then, any barcode that is associated with a non-coding region, inserted within the first 10% or the last 10% of a gene or associated with 3 or less counts in the T0 sample are filtered out. To enable comparison across all conditions, raw counts are then corrected in each condition independently using the number of counts of a reference gene. These corrected counts are then used to calculate mutant fitness and then gene fitness. During data processing, different plots are generated to follow data modification and potentially spot some problems.

Inputs:

- **A .csv file adapted, as described below, from the allpoolcounts.tab file** generated by the Wetmore et al. perl script BarSeqTest.pl. This file contains the number of counts per barcode per condition.

IMPORTANT:
- the first 7 columns of this file are the first 7 columns of the allpoolcounts.tab file
are the raw counts for each condition. They will be named by the condition.
- the 8th column is the T0 column named "T0".
- you will need to generate a .csv file for each replicate and maintain the same name for each condition across replicates.
See example: "Raw_Data_Example.csv"

- The **gene.GC** file used to run the perl script TestBarSeq.pl

See example: "genesExample.GC.txt"

Parameters set up by user:

In the script you have to specify a couple of parameters:

- **org_locId**: "Num" if the locusId of the genes in your gene.GC file are numerics, "Char" if they are character. Note that when locusId's are characters the script may take longer to run
- **cdnb**: the number of conditions (including T0)
- **scaffoldX**: the scaffold ID of the chromosome (plots are coded only for data on the chromosome even if the fitness values are calculated for insertions on chromosome and plasmids)
- **ref**: the locusId of the reference gene chosen for raw counts correction

Outputs:

An RData object containing:

- All_data_Replicate: table containing the final normalized gene fitness values for all the genes in each condition
- Data_norm_mean_OP: similar to All_data_Replicate but formatted differently
- Data_norm_mode: table containing gene fitness values normalized around the mode and not the mean
- Data_norm_mean: vector containing the mean value of fitness distribution of each condition
- Unnorm_values: table containing the fitness values before normalization by insertion location and correction by the mean (or mode)
- Data_ref_corrected: Table containing the number of reads per barcode after correction by the reference gene
- Data_original: input data
- genes.tab: gene.GC file

Functions:

- **Data_prep_viz10KB()**:

This function processes the number of reads per barcode so that we can visualize the number of reads per 10kb along the chromosome for each condition.

- **Ref_counts_correction()**:

For each condition, this function corrects the number of reads per barcode according to the number of reads associated with a reference gene. This is performed after adding the pseudocount and filtering out the barcodes with low counts or inappropriate insertion location.

- **Unnorm_gene_fitness()**:

This function produces the unnormalized fitness value for each gene in each condition. It returns a list containing one table with insertion mutant fitness values, one table with unnormalized gene fitness values, and one table with gene fitness variance.

First, we calculate for each insertion mutant the strain fitness as:

$$f_s = log_2(\frac{Corrected\ counts\ in\ Condition}{Corrected\ counts\ in\ T0})$$

To account for low counts (Corrected counts in Condition=0), we identify Cmin: smallest corrected count different from 0. Then all 0s are corrected counts are replaced by Cmin/10.
Then, gene fitness values are calculated as the average of the insertion mutant fitness values associated with that gene. Similarly, gene fitness variance is calculated as the variance of the insertion mutant fitness associated with that gene.

- **Loc_smoothmed_norm():**

This function performs the first normalization step of the gene fitness values: normalization based on the insertion location (as genes close to the replication fork may have higher count if cells are dividing, which would bias fitness calculation). This normalization is performed as described in Wetmore et al., 2015. It is performed for each scaffold independently, and the transformation used depends on the length of the scaffold. For scaffolds with more than 251 genes: genes are normalized using the smoothed median in a window of 251 genes. For scaffolds with between 10 and 251 genes, we correct fitness values by the median fitness of the scaffold. If the scaffold has less than 10 genes, no normalization is performed.

- **Norm_around_mean() and Norm_around_mode():**

These functions are the second normalization step. For one condition, it centers the gene fitness value distribution either around the mean (Norm_around_mean()) or the mode (Norm_around_mode()) of the distribution. This is according to the assumption that most of the genes are expected to have no fitness effect.

Part II: Averaging gene fitness values across replicates

Associated script: Averaging_Replicates.R

This script allows you to investigate correlation between your biological replicates and to average the normalized genes fitness values across the replicates.
Figure 2 describes the workflow of Script II as well as the associated functions.

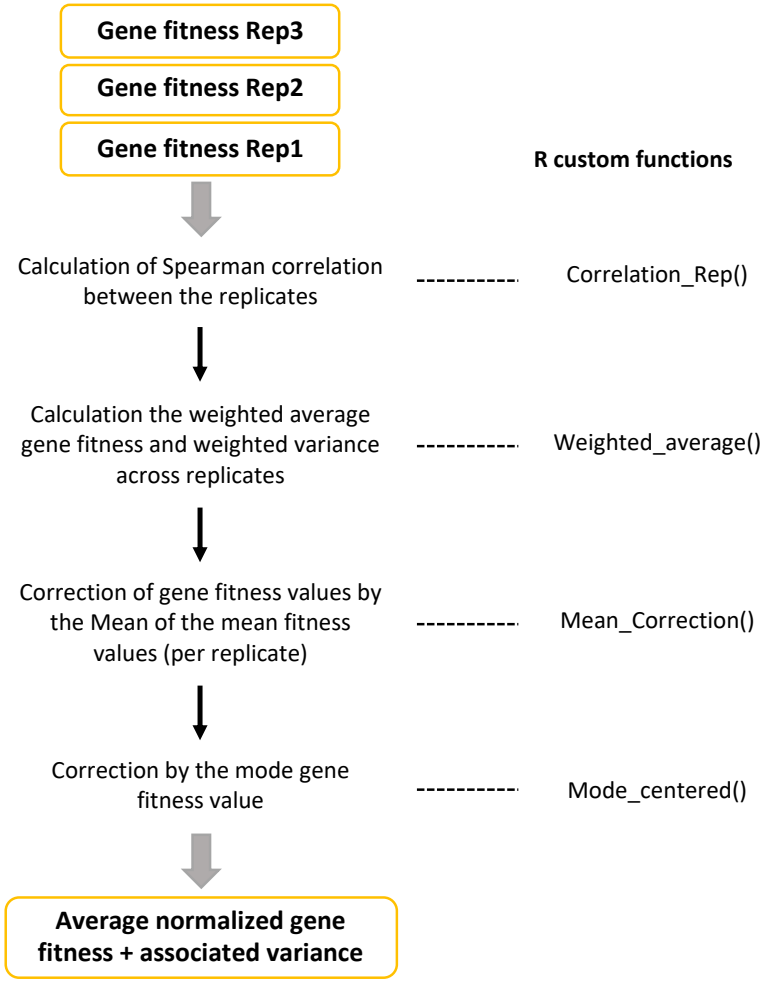

Figure 2: Workflow of averaging normalized fitness values across replicates

Averaged genes fitness values are calculated as the weighted average of replicate gene fitness. Then, for each condition, average gene fitness values are corrected by the mean of the replicate mean. Finally, each gene fitness distribution is centered around the mode.

Inputs:

- **the .RData generated for each replicate in Script I ➔** it should be 3 different .RData files

Parameters:

In this script you have to specify 1 parameter:

- **Org_locId**: “Num” if the locusId of the genes in your gene.GC file are numerics, “Char” if they are character.

Outputs:

An RData object containing (along with all the plots):

- Final_gene_Fitness: table containing the final averaged and normalized gene fitness values for all the genes in each condition
- Mean_corrected_averageF: table containing the average gene fitness corrected by the mean of the replicate mean (not centered around the mode)
- Average_fitness: table containing the weighted average gene fitness across replicates (not corrected by mean nor centered around mode)
- Correlation_table_fit: table containing for each condition the Pearson correlation coefficient for all pairwise comparisons of replicates
- AllReplicate: single table containing the normalized gene fitness values for each replicate
- mean1, mean2 and mean3: the means of gene fitness in Replicates 1,2 and 3
- genes.tab: gene.GC file

Functions:

- **Correlation_Rep():**

This function calculates the Pearson, Spearman and Lin's correlation coefficients for all the replicate combinations for each condition (Replicate 1 versus Replicate 2, Replicate 1 versus Replicate 3 and Replicate 2 versus Replicate 3). This function produces a correlation table and automatically saves correlation graphs with Pearson R squared as a pdf document called "Correlation_plots_fit.pdf"

- **Weighted_average():**

This function averages each gene fitness value across replicates. More specifically, the gene fitness value is the weighted average of gene fitness across replicates. Similarly, for each gene, this function calculates weighted variance for each gene fitness value.

- **Mean_Correction():**

Once average gene fitness has been calculated, each fitness value is corrected by the Mean of the replicates' means of distribution.

- **Mode_Centered():**

This function concludes the calculation of gene fitness values. It centers the fitness distribution around the mode of the distribution, based on the assumption that most of the genes are associated with a neutral fitness.

Part III: Comparing gene fitness values between two conditions

Associated script: 2conditions_FitnessComparison.R

The focus of part III is the comparison of gene fitness values between two conditions. In the script, you determine a reference condition against which all the other conditions are going to be tested.

The comparison of gene fitness values between two conditions relies on an unpaired two-sample Student test for samples with equal variance. Before running the Student test, we test for equal variance using a Fisher test. If variances are unequal between samples for a given gene, we do not perform the t-test.

The Fisher test calculates the ratio of the variance of the gene fitness in both conditions. The greatest variance has to be the numerator. If the ratio is EQUAL or GREATER than the F statistic,

then variances are not equal. In the script, the default threshold is set up for alpha=0.025, F(2,2)=39 (degree of freedom is n-1).

In the case of equal variance, a t-statistic is calculated. In the script, you can choose the value of alpha, to screen for significant comparisons. This alpha value is mostly used in the second function of the script (Category_Definition.R) which assigns labels to each gene comparison, whether it is significant ("Sig"), not significant ("Not_Sig") or not tested ("Not_tested").

The workflow is described in Figure 3:

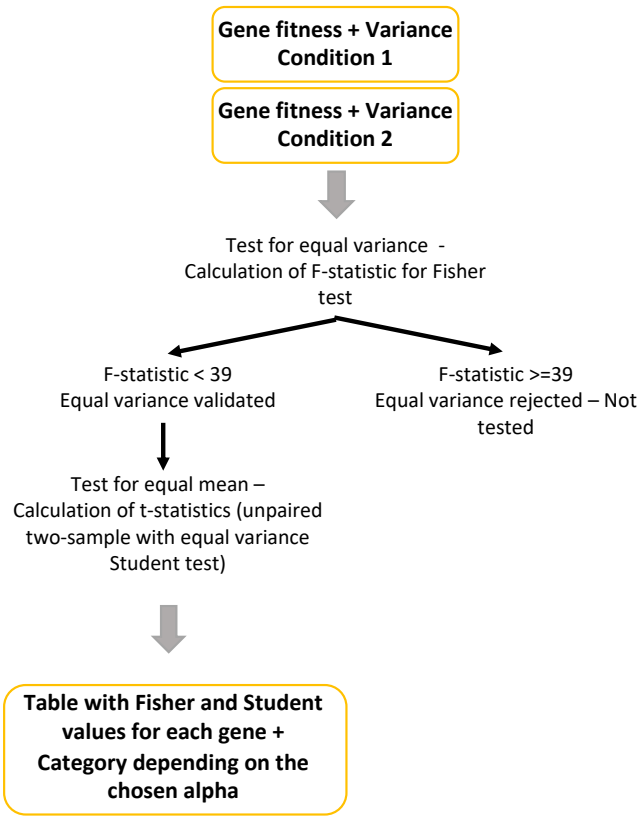

Figure 3: Workflow of gene fitness value comparison

Inputs:

- **the final .RData** generated from Script II

Parameters:

In the script you have to specify these parameters:

- **Org_locId**: "Num" if the locusId of the genes in your gene.GC file are numerics, "Char" if they are character.
- **Condition1**: the condition against which you want to test all the other conditions
- **Alpha**: alpha you want to use for your student test to reject H0.

Outputs:

A list containing two elements:

- Table_all: the table containing, for each comparison, the gene fitness values, compared conditions, F-statistics, T-statistics and label for comparison status ("Sig", "Not_sig", "Not_tested")
- List_plots: list containing all the comparison plots

Functions:

- **Comparison_test():**

Performs for each comparison the Fisher test and the Student test. While producing the final table, it also produces a scatter plot for each comparison highlighting the genes with significantly different fitness values.

- **Category_definition():**

This function is called within the Comparison_test() function. Based on the chosen alpha and the t-statistic calculated by the student test, this function assigns a category label to each test. "Sig": the gene fitness values between the two conditions are significantly different based on your alpha choice; "Not_Sig": the gene fitness values between the two conditions are not significantly different based on your alpha choice; "Not_tested": the student test was not performed for that gene because the variances were unequal according to the Fisher test.