Data Processing Code

Ryan Godin, Saman Ghazvini, William Amendola Bye, Jordyn Eovito, Vencke Gruening

2025-05-04

The following code prepares the raw data from Chen et al., 2023's *Deep Mutational Scanning of an Oxygen-Independent Fluorescent Protein CreiLOV for Comprehensive Profiling of Mutational and Epistatic Effects* for the analyses used to reproduce the figures. The raw single point mutation code is given in `data/raw/sb2c00662_si_001.xlsx` while the raw combinatorial mutation data is given in `data/raw/sb2c00662_si_002.xlsx`. The processed data is stored in the `data/processed` folder so that it can be loaded for analyses.

The following libraries are needed for the data processing.

```
rm(list = ls())
library(tidyr)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v purrr     1.0.4
## v forcats   1.0.0     v readr     2.1.5
## v ggplot2   3.5.1     v stringr   1.5.1
## v lubridate 1.9.4     v tibble    3.2.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(readxl)
library(pbapply)
library(openxlsx)
library(scales)
```

```
##
## Attaching package: 'scales'
##
## The following object is masked from 'package:purrr':
##
##     discard
##
## The following object is masked from 'package:readr':
##
##     col_factor
```

Single Mutation Data Processing

For the downstream analyses, the following data needs to be extracted from the entries and added as columns to the dataframe:

1. Mutation position

2. Wild-type amino acid
3. Mutant amino acid

Load the dataset and inspect the file.

```
single_mutant_data <- read_excel("data/raw/sb2c00662_si_001.xlsx")
```

```
## New names:
## * `` -> `...1`
```

```
colnames(single_mutant_data)[1] <- "mutants"
```

Checking the first few lines, last few lines, and dimensions of the file to see if it loaded correctly.

```
head(single_mutant_data)
```

```
## # A tibble: 6 x 9
##   mutants      rep1   rep2   rep3   mean log_rep1 log_rep2 log_rep3 log_mean
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 wt         13642. 14855. 15912. 14803.     4.13     4.17     4.20     4.17
## 2 p.Arg60Asp 25760. 26659. 26651. 26357.     4.41     4.43     4.43     4.42
## 3 p.Thr7Ser  25818. 26427. 25983. 26076.     4.41     4.42     4.41     4.42
## 4 p.Ala29His 26196. 26780. 24982. 25986.     4.42     4.43     4.40     4.41
## 5 p.Gly26Thr 25377. 26196. 24196. 25256.     4.40     4.42     4.38     4.40
## 6 p.Gln47Ile 24669. 26085. 23552. 24769.     4.39     4.42     4.37     4.39
```

```
tail(single_mutant_data)
```

```
## # A tibble: 6 x 9
##   mutants     rep1  rep2  rep3  mean log_rep1 log_rep2 log_rep3 log_mean
##   <chr>      <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 p.Ala11Arg  379.  469.  320.  389.     2.58     2.67     2.50     2.59
## 2 p.Ala11Tyr  457.  299.  392.  383.     2.66     2.48     2.59     2.58
## 3 p.Leu73Arg  396.  408.  339.  381.     2.60     2.61     2.53     2.58
## 4 p.Asn85Trp  302.  376.  459.  379      2.48     2.57     2.66     2.58
## 5 p.Gly32Pro  324.  404.  347.  358.     2.51     2.61     2.54     2.55
## 6 p.Leu20Trp  337.  259.  263.  287.     2.53     2.41     2.42     2.46
```

```
dim(single_mutant_data)
```

```
## [1] 2185    9
```

Extract the mutant amino acid position

```
single_mutant_data <-
  mutate(
    single_mutant_data,
    position = as.integer(str_extract(mutants, "\\d+")),
)
head(single_mutant_data[,c("mutants", "position")])
```

```
## # A tibble: 6 x 2
##   mutants    position
##   <chr>         <int>
## 1 wt               NA
## 2 p.Arg60Asp       60
## 3 p.Thr7Ser         7
## 4 p.Ala29His       29
## 5 p.Gly26Thr       26
## 6 p.Gln47Ile       47
```

Extract the wild-type amino acid

The amino acid will be extracted as a one letter code to make the replication of the figures easier. First, we define the table neccessary to convert the three letter to one letter amino acid abbreviation.

```
conversion_table <- c(
    Ala = "A", Arg = "R", Asn = "N", Asp = "D", Cys = "C",
    Gln = "Q", Glu = "E", Gly = "G", His = "H", Ile = "I",
    Leu = "L", Lys = "K", Met = "M", Phe = "F", Pro = "P",
    Ser = "S", Thr = "T", Trp = "W", Tyr = "Y", Val = "V"
)
```

We now extract the one letter amino acid abbreviation for the wild-type amino acid.

```
extract_wt_amino_acid <- function(mutant) {
    mutant <- as.character(mutant)
    if (mutant == "wt") {
        return(NA)
    } else {
        removed_prefix <- str_remove(mutant, "^p\\.")  # Remove "p." prefix
        wt_amino_acid_3_letter <- str_extract(removed_prefix, "^[A-Za-z]+")
        wt_amino_acid_1_letter <- conversion_table[wt_amino_acid_3_letter]
        return(wt_amino_acid_1_letter)
    }
}

single_mutant_data <- mutate(
    single_mutant_data,
    wt_amino_acid = sapply(mutants, extract_wt_amino_acid)
)

head(single_mutant_data[,c("mutants", "position", "wt_amino_acid")])
```

```
## # A tibble: 6 x 3
##   mutants    position wt_amino_acid
##   <chr>         <int> <chr>
## 1 wt               NA <NA>
## 2 p.Arg60Asp       60 R
## 3 p.Thr7Ser         7 T
## 4 p.Ala29His       29 A
## 5 p.Gly26Thr       26 G
## 6 p.Gln47Ile       47 Q
```

Extract the mutant amino acid

NOTE: You need to load the conversion table from extracting the wild-type amino acid by running the corresponding code block!

```
extract_mutant_amino_acid <- function(mutant) {
    mutant <- as.character(mutant)
    if (mutant == "wt") {
        return(NA)
    } else {
        removed_prefix <- str_remove(mutant, "^p\\.")  # Remove "p." prefix
        wt_amino_acid_3_letter <- str_extract(removed_prefix,  "[A-Za-z]+$")
        wt_amino_acid_1_letter <- conversion_table[wt_amino_acid_3_letter]
        return(wt_amino_acid_1_letter)
    }
}

single_mutant_data <- mutate(
    single_mutant_data,
    mutant_amino_acid = sapply(mutants, extract_mutant_amino_acid)
)

head(single_mutant_data[,c("mutants", "position", "wt_amino_acid", "mutant_amino_acid")])
```

```
## # A tibble: 6 x 4
##   mutants    position wt_amino_acid mutant_amino_acid
##   <chr>         <int> <chr>         <chr>
## 1 wt               NA <NA>          <NA>
## 2 p.Arg60Asp       60 R             D
## 3 p.Thr7Ser         7 T             S
## 4 p.Ala29His       29 A             H
## 5 p.Gly26Thr       26 G             T
## 6 p.Gln47Ile       47 Q             I
```

Save processed data

The processed single mutant data will be exported to `data/processed/single_mutant_data.csv` so that the single mutant data analysis can be run independently of the data processing.

```
write.csv(single_mutant_data, file = "data/processed/single_mutant_data.csv", row.names=FALSE)
```

Double Mutation Data Processing

To process the raw combinatorial mutation data, the following data needs to be extracted/calculated from the raw mutation data.

1. The position of the mutation.
2. Number of mutations.
3. Expected fluorescence based on summing the effects of the individual mutations.
4. Level of Epistasis.

We first load the raw combinatorial mutation datast.

```
combinatorial_mutation_data <- read_excel("data/raw/sb2c00662_si_002.xlsx")
```

```
## New names:
## * `` -> `...1`
```

```
colnames(combinatorial_mutation_data)[1] <- "mutants"
```

Checking the first few lines, last few lines, and dimensions of the file to see if it loaded correctly.

```
head(combinatorial_mutation_data)
```

```
## # A tibble: 6 x 9
##   mutants      Rep1   Rep2   Rep3   mean Rep1_log Rep2_log Rep3_log mean_log
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 wt         11436. 11591.  8622. 10549.     4.06     4.06     3.94     4.02
## 2 p.Thr7Ser 15190. 15178. 10987. 13785.     4.18     4.18     4.04     4.14
## 3 p.Arg5Asp 11743. 14620. 12136. 12833.     4.07     4.16     4.08     4.11
## 4 p.Thr7His 12283. 13610. 12036. 12643.     4.09     4.13     4.08     4.10
## 5 p.Leu4Asn 12076. 10925.  8659. 10553.     4.08     4.04     3.94     4.02
## 6 p.Gly3Glu 11629.  9858.  9738. 10408.     4.07     3.99     3.99     4.02
```

```
tail(combinatorial_mutation_data)
```

```
## # A tibble: 6 x 9
##   mutants             Rep1  Rep2  Rep3  mean Rep1_log Rep2_log Rep3_log mean_log
##   <chr>              <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 p.Gly3Glu, p.Leu4~ 3431. 3833. 2841. 3369.     3.54     3.58     3.45     3.53
## 2 p.Gly3Glu, p.Leu4~ 2825. 2872. 1734. 2477.     3.45     3.46     3.24     3.39
## 3 p.Gly3Glu, p.Leu4~ 2210. 1530. 2580. 2107.     3.34     3.18     3.41     3.32
## 4 p.Gly3Glu, p.Leu4~ 1669. 2257. 2042. 1989.     3.22     3.35     3.31     3.30
## 5 p.Gly3Glu, p.Leu4~ 1933. 1633. 1767. 1778.     3.29     3.21     3.25     3.25
## 6 p.Gly3Glu, p.Leu4~ 1264. 1140. 1135. 1180.     3.10     3.06     3.06     3.07
```

```
dim(combinatorial_mutation_data)
```

```
## [1] 165428      9
```

Extract position

We now extract the position for the mutants by extracting them from the first column. Note that for mutants with more than one mutation, it extracts just the last mutation position.

```
combinatorial_mutation_data$position <- as.integer(gsub(".*[a-zA-Z](\\d+)[a-zA-Z]*", "\\1", combinatori
```

```
## Warning: NAs introduced by coercion
```

```
head(combinatorial_mutation_data[,c("mutants", "position")])
```

```
## # A tibble: 6 x 2
##   mutants   position
##   <chr>        <int>
## 1 wt              NA
## 2 p.Thr7Ser        7
## 3 p.Arg5Asp        5
## 4 p.Thr7His        7
## 5 p.Leu4Asn        4
## 6 p.Gly3Glu        3
```

```
tail(combinatorial_mutation_data[,c("mutants", "position")])
```

```
## # A tibble: 6 x 2
##   mutants                                                             position
##   <chr>                                                                  <int>
## 1 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29Lys, p.Gly34Thr, ~      113
## 2 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7Ser, p.Ala29Lys, p.Gly34Thr, ~      113
## 3 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29Lys, p.Gly34Thr, ~      113
## 4 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29Lys, p.Gly34Thr, ~      113
## 5 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29His, p.Gly34Thr, ~      113
## 6 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29His, p.Gly34Thr, ~      113
```

Extract the number of mutations

The number of mutations is extracted by calculating then number of entries in the mutants column.

```
count_mutations <- function(mutation_list) {
    mutation_list <- as.character(mutation_list)
    if (mutation_list == "wt") {
        return(0)
    } else {
        mutations_vector <- trimws(strsplit(mutation_list, ",")[[1]])
        return(length(mutations_vector))
    }
}

combinatorial_mutation_data <- mutate(
    combinatorial_mutation_data, mutation_count = sapply(mutants,count_mutations)
)

head(combinatorial_mutation_data[,c("mutants","mutation_count")])
```

```
## # A tibble: 6 x 2
##   mutants   mutation_count
##   <chr>              <dbl>
## 1 wt                     0
## 2 p.Thr7Ser              1
## 3 p.Arg5Asp              1
## 4 p.Thr7His              1
## 5 p.Leu4Asn              1
## 6 p.Gly3Glu              1
```

```
tail(combinatorial_mutation_data[,c("mutants","mutation_count")])
```

```
## # A tibble: 6 x 2
##   mutants                                                       mutation_count
##   <chr>                                                                  <dbl>
## 1 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29Lys, p.Gly3~             15
## 2 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7Ser, p.Ala29Lys, p.Gly3~             15
## 3 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29Lys, p.Gly3~             15
## 4 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29Lys, p.Gly3~             15
## 5 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29His, p.Gly3~             15
## 6 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.Ala29His, p.Gly3~             15
```

Calculate expected fluorescence

To calculate the expected fluorescence for the combinatorial mutants, we extract the mean log fluorescence for the single mutants and use the following formula provided in the paper where F_{com}, F_{sin}, and F_{wt} are the log-fluorescence values of combinatorial mutant, single mutant, and WT CreiLOV, respectively.

$$e = (F_{\text{com}} - F_{\text{wt}}) - \Sigma\,(F_{\text{sin}} - F_{\text{wt}})$$

Note: This calculation takes some time for the 160,000 mutants. To speed up this calculation, we extract the single mutant data first and store them so that the program doesn't have to search through 160,000 entries when extracting the single mutants. This sped up runtime from 20 min to 1 min. However, we still include a progress bar for the calculation using `papply` so that progress can be tracked.

```
extracted_single_mutants <- filter(combinatorial_mutation_data, mutation_count < 2)
dim(extracted_single_mutants)
```

```
## [1] 21 11
```

```r
get_single_fluorescence <- function(mutation, mutation_data){
    single_mutation_row <- which(mutation_data[[1]] == mutation)
    return(mutation_data[single_mutation_row,"mean_log"])
}

get_expected_fluorescence <- function(mutation_list, mutation_data){
    mutation_list <- as.character(mutation_list)
    mutations_vector <- trimws(strsplit(mutation_list, ",")[[1]])
    single_mutant_fluorescence <- sapply(mutations_vector, get_single_fluorescence, mutation_data = muta
    wild_type_fluorescence <- mutation_data[1,"mean_log"]
    expected_fluorescence <- sum(single_mutant_fluorescence[[1]] - wild_type_fluorescence) + wild_type_f
    return(expected_fluorescence[[1]])
}

combinatorial_mutation_data <- mutate(
    combinatorial_mutation_data, expected_fluorescence = pbapply::pbsapply(mutants,get_expected_fluoresc
)

head(combinatorial_mutation_data[,c("mutants","mean_log", "expected_fluorescence")])
```

```
## # A tibble: 6 x 3
##   mutants   mean_log expected_fluorescence
##   <chr>        <dbl>                 <dbl>
## 1 wt            4.02                  4.02
## 2 p.Thr7Ser     4.14                  4.14
## 3 p.Arg5Asp     4.11                  4.11
## 4 p.Thr7His     4.10                  4.10
## 5 p.Leu4Asn     4.02                  4.02
## 6 p.Gly3Glu     4.02                  4.02
```

```r
tail(combinatorial_mutation_data[,c("mutants","mean_log", "expected_fluorescence")])
```

```
## # A tibble: 6 x 3
##   mutants                                    mean_log expected_fluorescence
##   <chr>                                         <dbl>                 <dbl>
## 1 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.~     3.53                  4.02
## 2 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7Ser, p.~     3.39                  4.02
## 3 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.~     3.32                  4.02
## 4 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.~     3.30                  4.02
## 5 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.~     3.25                  4.02
## 6 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.Thr7His, p.~     3.07                  4.02
```

Calculate epistasis

We now calculate the level of epistasis (difference betweene expected and observed fluorescence).

```r
combinatorial_mutation_data <- mutate(
    combinatorial_mutation_data, epistasis = mean_log - expected_fluorescence
)

head(combinatorial_mutation_data[,c("mutants","mean_log", "expected_fluorescence", "epistasis")])
```

```
## # A tibble: 6 x 4
##   mutants   mean_log expected_fluorescence epistasis
##   <chr>        <dbl>                 <dbl>     <dbl>
## 1 wt             4.02                    4.02         0
## 2 p.Thr7Ser      4.14                    4.14         0
## 3 p.Arg5Asp      4.11                    4.11         0
## 4 p.Thr7His      4.10                    4.10         0
## 5 p.Leu4Asn      4.02                    4.02         0
## 6 p.Gly3Glu      4.02                    4.02         0
```

```
tail(combinatorial_mutation_data[,c("mutants","mean_log", "expected_fluorescence", "epistasis")])
```

```
## # A tibble: 6 x 4
##   mutants                               mean_log expected_fluorescence epistasis
##   <chr>                                    <dbl>                 <dbl>     <dbl>
## 1 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.T~     3.53                  4.02    -0.490
## 2 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.T~     3.39                  4.02    -0.623
## 3 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.T~     3.32                  4.02    -0.694
## 4 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.T~     3.30                  4.02    -0.719
## 5 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.T~     3.25                  4.02    -0.768
## 6 p.Gly3Glu, p.Leu4Asn, p.Arg5Asp, p.T~     3.07                  4.02    -0.946
```

Identify mutants with strong epistasis

```
combinatorial_mutation_data <- mutate(
    combinatorial_mutation_data, strong_epistasis = abs(epistasis) > 0.6
)
```

Save processed data

Like the processed single mutant data, the processed combinatorial mutation data will be exported to `data/processed/combinatorial_mutant_data.csv` so that the data analysis can be run independently of the data processing.

```
write.csv(combinatorial_mutation_data, file = "data/processed/combinatorial_mutant_data.csv", row.names
```

Analyze Data

Now that the raw data has been processed, the data analysis (figure generation) can now be performed using the designated RMD file: `data_analysis.RMD`.