# Comparing the expression of J23106_with emrR, J23109_with emrR

*Joe Ho*

*August 04, 2015*

### Experiment description

Describe the experiment in detail here..

### Data upload and transformation

read.csv() uploads the data directly as a data frame. The raw data is taken directly from the TECAN, although the temperature values and well IDs were removed in excel. Also, the well IDs we're replaced with the sample names. I printed the top of the file using head to see the structure of the file.

```
GFP <- read.csv("2015_08_15_GFP.csv",
                header=T, check.names=F) # Uploads the data with header. Check names ensures that the h
OD <- read.csv("2015_08_15_OD.csv",
               header=T, check.names=F)
print(GFP[1:5,1:5]) # prints the top of the GFP file
```

```
##                             ID   0 900 1800 2700.1
## 1    J23106_emrR_0uM Vanillin 585 582  551    578
## 2    J23106_emrR_0uM Vanillin 596 596  572    592
## 3    J23106_emrR_0uM Vanillin 586 581  570    597
## 4 J23106_emrR_1280uM Vanillin 584 567  563    577
## 5 J23106_emrR_1280uM Vanillin 583 568  559    578
```

The reshape library is a set of tools for rearraging data frames. The melt() function puts all of the values in a single column by the ID. Column names then replaced the column names as they get messed up using the melt function. The indidual data frames are then combined with cbind() and the redundant columns are trimeed. Then, a Norm column is created by deviding the FU values by OD. A new data frame is created which only includes the normalized values (other columns are trimmed).

```
library(reshape) # loads the library
GFP <- melt(GFP, by = "ID") # Puts all data in single column using the individual IDs.
colnames(GFP)  <- c("ID", "Time", "FU") # Replaces column names
OD <- melt(OD, by = "ID")
colnames(OD)  <- c("ID", "Time", "OD")
data <- cbind(GFP, OD) # Combines the two data frames by columns.
data <- data[,-c(4,5)] # Trims columns 4 and 5
data$Norm <- data$FU / data$OD # Calculates the normalized values
data_norm <- data[,-c(3,4)] # Creates a new data frame by trimming columns 3 and 4
```

We then calculate the mean and standard deviation for the normalized data. The aggregate() function is used for this. You need to specify the data being used to calculate the mean, the lists needed to say whats a replicate, and the the function to be applied. The mean and standard deviation are calculated seperately and the combined into the data_transformed file. Again, the redundant columns are trimmed and the column names are re-added.

```
data_mean <- aggregate(data_norm$Norm, list(data_norm$ID, data_norm$Time) , FUN = mean) # Calculate mea
data_sd <- aggregate(data_norm$Norm, list(data_norm$ID, data_norm$Time) , FUN = sd) # Calculate standar
data_transformed <- cbind(data_mean, data_sd) # Bind the data together by columns
data_transformed <- data_transformed[,-c(4,5)] # Remove redundant columns
colnames(data_transformed)  <- c("ID", "Time", "Mean", "Sd") # Change column names
```

This just order the data by the time points. We need to turn factors into numbers first (has to go through characters).

```
data_transformed$Time <- as.numeric(as.character(data_transformed$Time)) # Converts factors to numbers
data_transformed <- data_transformed[order(data_transformed$Time),] #Orders the data by time
```

This converts seconds to minutes and hours, then rounds the numbers down.

```
data_transformed$Time_min <- data_transformed$Time / 60 # Devide time by 60
data_transformed$Time_hrs <- data_transformed$Time_min / 60 # Devide min by 60
data_transformed$Time_min <- floor(data_transformed$Time_min) # Round down to the nearest integer
data_transformed$Time_hrs <- round(data_transformed$Time_hrs, digits=2) # Round down to 2 digits
```

This just removes the first 7.5 hrs (first 870 rows in the data) as the data is chaotic.

```
data_transformed <- data_transformed[-c(1:400),] # Removes columns 1 through 870
```

This is for subseeting the data you want to look at. Data frames with specific IDs are assigned to variables and the combined into the subsetted data.

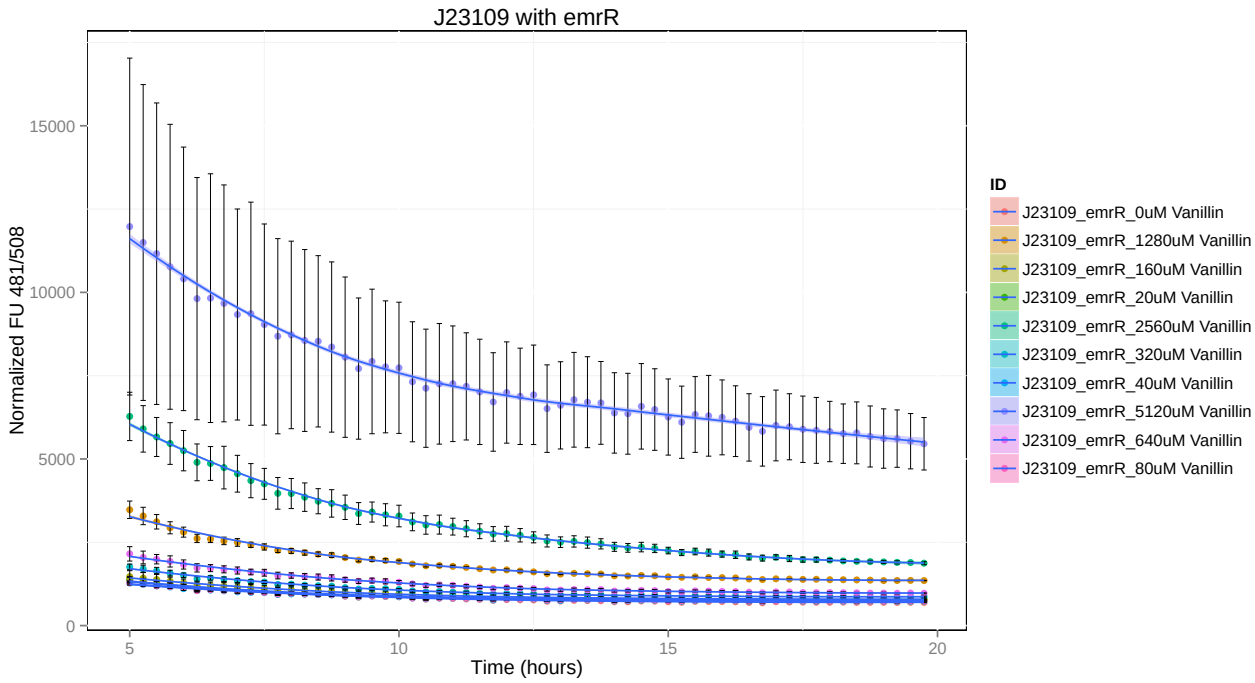
```
a <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_0uM Vanillin")) # Assigns ID to a
b <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_20uM Vanillin"))
c <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_40uM Vanillin"))
d <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_80uM Vanillin"))
e <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_160uM Vanillin"))
f <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_320uM Vanillin"))
g <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_640uM Vanillin"))
# Add high concentrations
h <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_1280uM Vanillin"))
i <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_2560uM Vanillin"))
j <- subset(data_transformed, data_transformed$ID == c("J23106_emrR_5120uM Vanillin"))

J23106_emrR <- rbind(a, b, c, d, e, f, g, h, i, j)

a <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_0uM Vanillin"))
b <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_20uM Vanillin"))
c <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_40uM Vanillin"))
d <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_80uM Vanillin"))
e <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_160uM Vanillin"))
f <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_320uM Vanillin"))
g <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_640uM Vanillin"))
# Add high concentrations
h <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_1280uM Vanillin"))
i <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_2560uM Vanillin"))
j <- subset(data_transformed, data_transformed$ID == c("J23109_emrR_5120uM Vanillin"))
```

```
J23109_emrR <- rbind(a, b, c, d, e, f, g, h, i, j)


J23106_emrR <- J23106_emrR[order(J23106_emrR$Time),]
J23109_emrR <- J23109_emrR[order(J23109_emrR$Time),]
```

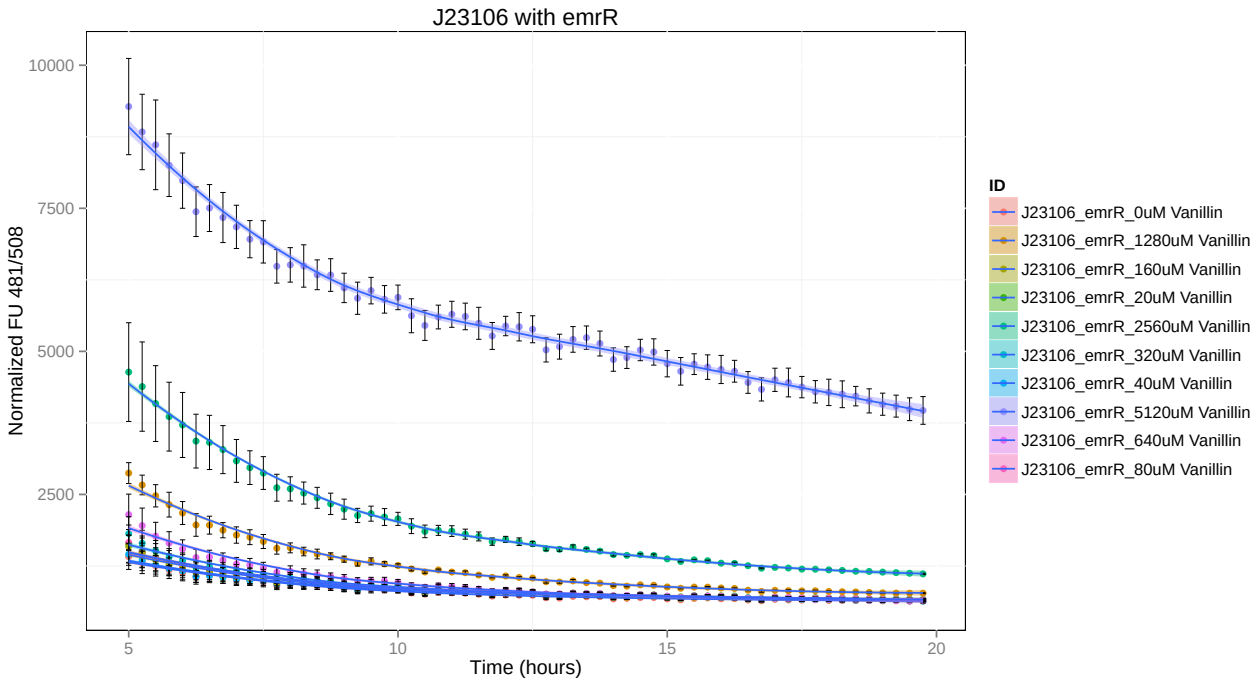
### Plotting

This uses ggplot2 to make scatter plots of the data. The x and y need to be specified, along with what is going to be shown for the error bars (in this case standard deviation). The plots are made seperately for the constructs with and without emrR.

```
library(ggplot2) # loads the ggplot library
library(gridExtra) # library used to arrange plots together

ggplot(J23109_emrR, aes( x=Time_hrs, y=Mean))+ geom_point(aes(group=ID, colour=ID)) + geom_errorbar(aes
```




```
ggplot(J23106_emrR, aes( x=Time_hrs, y=Mean))+ geom_point(aes(group=ID, colour=ID)) + geom_errorbar(aes
```


J23106 with emrR
Normalized FU 481/508
10000
7500
5000
2500
5
10
15
20
Time (hours)
ID
J23106_emrR_0uM Vanillin
J23106_emrR_1280uM Vanillin
J23106_emrR_160uM Vanillin
J23106_emrR_20uM Vanillin
J23106_emrR_2560uM Vanillin
J23106_emrR_320uM Vanillin
J23106_emrR_40uM Vanillin
J23106_emrR_5120uM Vanillin
J23106_emrR_640uM Vanillin
J23106_emrR_80uM Vanillin