Fitting Lotka-Volterra Equation to Data

Olusoji O. D.

First, let's get in some data and select the columns that is of interest. First things are demonstrated with a small part of the dataset, then later extended to the whole dataset.

```
library(minpack.lm)
library(deSolve)
library(tidyverse)
library(knitr)
library(gtable)
wouter_data <- read.csv("Data/dataWouter181017.csv", stringsAsFactors = F)
wouter_data <- wouter_data %>% dplyr::select(Strain, Treatment, Time = t, X)
# Strain = 3 & Treatment = 23A
expdata <- wouter_data %>% dplyr::filter(Strain == 3 & Treatment == "23A")
```

The codes below are to fit the **Lotka-Volterra (LVE)** equation for single specie to dataset using a combination of the tools in *tidyverse*, *desolve* and *minpack.lm* package in **R**.

The Lotka Volterra Equation

The LVE can be written as;

$$\frac{dx}{dt} = \mu x(1 - \frac{x}{K})$$

where μ is the inherent growth rate per capita and K is the carrying capacity. It can be rewritten as

$$\frac{dx}{dt} = x(\mu - Ax)$$

where A is then the ratio $\frac{\mu}{K}$.

The LVE in R

To fit this in **R**, let's start by writing the ODE above as a function to be solved by the *desolve* package.

```
LVE <- function(Time, X, parms, ...) {
    #mu =  inherent growth per-capita
    mu <- parms$mu
    #A = ratio of mu and carrying capacity
    A <- parms$A
    dX <- X * (mu - (A * X))
    list(dX)
}
```

- Time is the time point of meaurement
- X is the population at time t
- parms is a vector of starting values for μ and A.

Starting Values

Next is a function to obtain starting values for μ and A. Since μ is the inherent growth rate per capita, a good guess for this parameter will be any of the relative difference between X_t and X_{t+11} but let's go with the average of these relative difference. In the same vein, a good guess for A will be the guessed value for μ and the maximum value of X_t. Hence, to guess starting values for μ and A, compute;

$$\frac{X_{t+1} - X_t}{X_t} \times \frac{Time_t}{Time_{t+1}}$$

which is the inherent per capita growth rates relative to time. The average of these is guessed as the starting value for μ while the ratio of this maximum to that of X_t is guessed as starting value for A.

```
start_values <- sapply(1:nrow(expdata), function(i, x, y) {
    if(i == NROW(y) | x[i] == 0) return(NA)
      pcgr <- ((x[i + 1] - x[i]) / x[i]) / (y[i + 1] - y[i])
    return(pcgr)
  }, x = expdata$X, y = expdata$Time)
start_values
```

```
##  [1]  0.0958257160  0.0589566020  0.0166896077  0.0001706252  0.0004213862
##  [6] -0.0011564896  0.0003498461  0.0004709255 -0.0004538212  0.0002844685
## [11] -0.0006712517  0.0003561854            NA
```

```
parms_start <- c(mu = mean(start_values, na.rm = T),
                 A = mean(start_values, na.rm = T) / max(expdata$X))
parms_start
```

```
##           mu            A
## 0.0142703167 0.0002236376
```

Initial Fit

Next, fit the ODE with the starting values obtained above. To do this, use the **ode()** function in the *desolve* package. The function takes 4 important inputs namely;

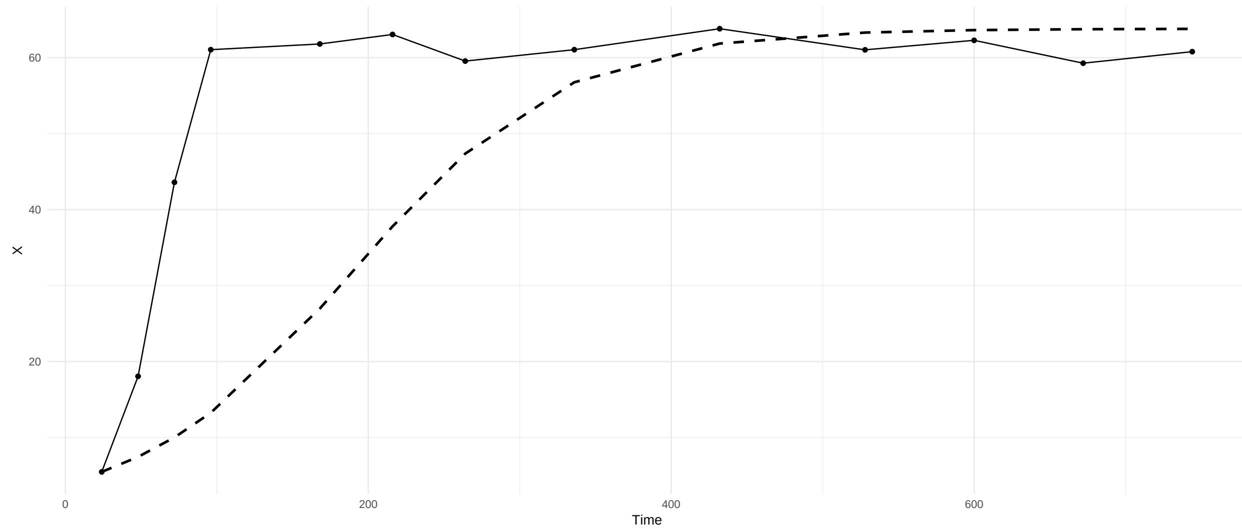
- y = initial state of the ODE (X_0)
- times = time sequnce for which we want a prediction
- func = function depicting the ODE (LVE)
- parms = values for the parameters in func

```
#value of X at the first time point
x0 <- expdata$X[1]
# a mix of a created time sequence with observed Time
times <- sort(unique(c(seq(min(expdata$Time), max(expdata$Time), length = 100),
                       expdata$Time)))
ini_fit <- ode(y = x0, times = times, func = LVE, parms =
                 list(A = parms_start[2], mu = parms_start[1]))
ini_fit_df <- as.data.frame(ini_fit)
names(ini_fit_df) <- c("Time", "X")

#to be used later for likelihood computation
ini_fit2 <- ode(y = x0, times = expdata$Time, func = LVE, parms =
                 list(A = parms_start[2], mu = parms_start[1]))
ini_fit2_df <- as.data.frame(ini_fit2)
names(ini_fit2_df) <- c("Time", "X")
```

Let's see where things are at the moment.

```
plot1 <- expdata %>% ggplot(aes(x = Time, y = X, group = 1)) + geom_point() +
  geom_line() + theme_minimal() +
```

```
  geom_line(data = ini_fit2_df, aes(x = Time, y = X),
            color = 1, linetype = "dashed", size = 1)
print(plot1)
```



The black points are observed data points, while the dashed line are predicted values from the **ode()** function using the guessed parameter values. Looks like this is a good start. Things can be improved by minimizing the squared error between our current prediction from the **LVE** model and the observed values of X, which will lead to optimal parameter estimates.

Estimating A and μ

To do this, let's write a function to get the residuals between the observed values and the predicted values. Then use the **nls.lm()** in the *minpack.lm* package. This function takes two important inputs namely;

- par = a named vector of starting values for the parameters to be estimated
- fn = a function that returns the residuals

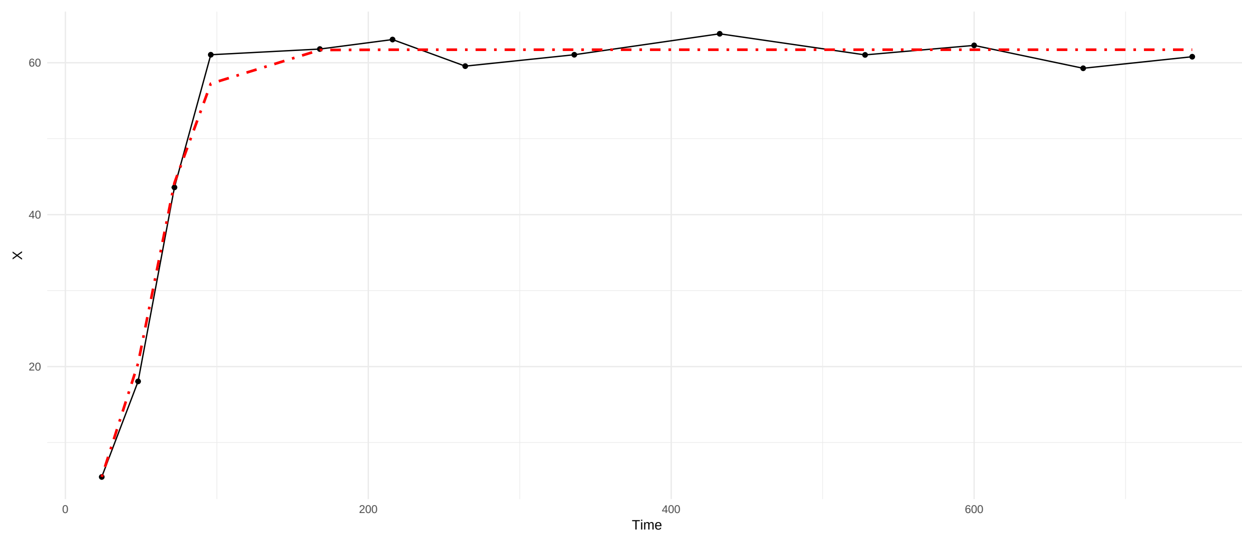
```
#function to return the residuals
SSR <- function(parms) {
   #parameters
    A <- parms[2]
    mu <- parms[1]
    #
    ini_fit <- deSolve::ode(y = x0, times = times, func = LVE,
                            parms = list(A = A, mu = mu))
    ini_fit_df <- as.data.frame(ini_fit)
    names(ini_fit_df) <- c("Time", "X")
    #filtering out data for observed time points
    ini_fit_df2 <- ini_fit_df[ ini_fit_df$Time %in% expdata$Time, ]
    #residual
    ini_fit_df2$X - expdata$X
}

final_fit <- minpack.lm::nls.lm(par = parms_start, fn = SSR)
#extracting the final estimates
final_estimates <- coef(final_fit)
#summary
```

```
final_fit_summary <- summary(final_fit)
final_fit_summary
```

```
##
## Parameters:
##     Estimate Std. Error t value Pr(>|t|)
## mu 0.0677982  0.0026117   25.96 3.21e-11 ***
## A  0.0010986  0.0000463   23.73 8.47e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.893 on 11 degrees of freedom
## Number of iterations to termination: 8
## Reason for termination: Relative error in the sum of squares is at most `ftol'.
```

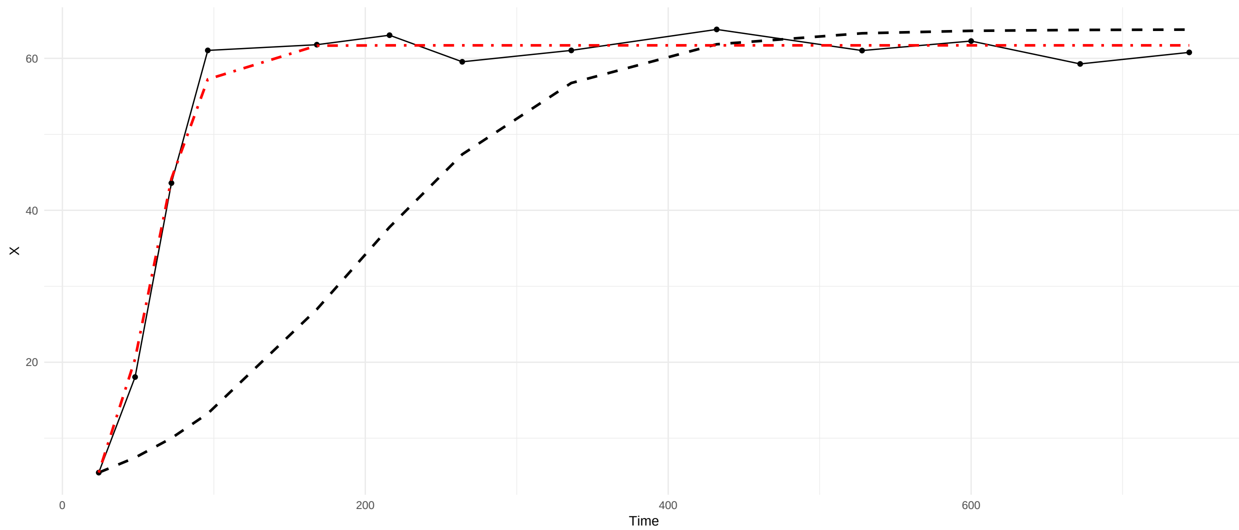
Now let's see what the final fit looks like. To do this, refit the ODE with the parameter estimates obtained from the **nls.lm()** function and plot against the observed dataset.

```
#predicted values from the final fit
final_fit2 <- ode(y = x0, times = expdata$Time, func = LVE, parms =
                  list(A = final_estimates[2], mu = final_estimates[1]))
final_fit_df <- as.data.frame(final_fit2)
names(final_fit_df) <- c("Time", "X")

expdata %>% ggplot(aes(x = Time, y = X, group = 1)) + geom_point() +
  geom_line() + theme_minimal() +
  geom_line(data = final_fit_df, aes(x = Time, y = X),
            color = "red", linetype = 4, size = 1)
```



To put it all into perspective;

```
plot1 + geom_line(data = final_fit_df, aes(x = Time, y = X),
            color = "red", linetype = 4, size = 1)
```



The black dashed line was what we started with from the initial parameter guess, while the dashed red line was the fit obtained from minimizing the sum of squared error between the black line and the observed data points.

Inference and Fit Assessment

It is normally of interest to compare models and since this approach doesn't involve building a likelihood, it becomes a little tricky to compare fit of models obtained via this approach. However there is a way out. In this part, we limit the discussion to model comparison involving loglikelihoods (likelihood ratio test and AIC).

To obtain the loglikelihood of the models in question, we assume the observed data follows a normal distribution. Then the problem reduces to obtaining the loglikelihood at the estimated parameter values. To do this, we simply employ the **dnorm()** function. We plug in the observed data as the data points, the fitted values (these are typically mean values) from the final fit as the mean and then the estimated sigma (standard deviation of the residuals) from **minpack.lm()** as sigma.

Note that we returned log of the densities and summed it up, this is from $log(\prod_{i=1}^{n} x_i = \sum_{i=1}^{n} log(x_i)$. Let's do this for both the initial fit using our guessed values for the parameters and the final fit. It is important to stress that, for this to be valid, the dataset should remain the same, hence we used fitted values obtained from the observed time in the dataset and not the time sequence created for the initial fit (see the initial fit section).

```
#likelihood of the initial fit
loglike_initial <- sum(dnorm(x = expdata$X, mean = ini_fit2_df$X,
                             sd = final_fit_summary$sigma, log = T))
loglike_initial
```

```
## [1] -799.5507
```

```
#likelihood of the final fit
loglike_final <- sum(dnorm(x = expdata$X, mean = final_fit_df$X,
                           sd = final_fit_summary$sigma, log = T))
loglike_final
```

```
## [1] -25.74077
```

```
### AIC = -2*loglikelihood + 2*npar.
#we treated sigma as a parameter to be estimated as well,
#hence the need for + 1
```

```r
#initial fit
AIC_initial <- (-2*loglike_initial) + (length(final_estimates) + 1)*2
AIC_initial
```

```
## [1] 1605.101
```

```r
#final fit
AIC_final <- (-2*loglike_final) + (length(final_estimates) + 1)*2
AIC_final
```

```
## [1] 57.48155
```

So models can be compared if need be, it just requires the extra steps above. To summarise;

1. write up the Lotka-Volterra equation as a function **(LVE)**
2. guess starting vlaues for A and μ, we presented an approach to obtain good starting values
3. perform an initial fit using **ode()** function from the *desolve* package
4. write a function to obtain residuals using the observed data and fitted values from 3
5. use **nls.lm()** function to estimate the optimum values for A and μ

Fit to Whole Dataset

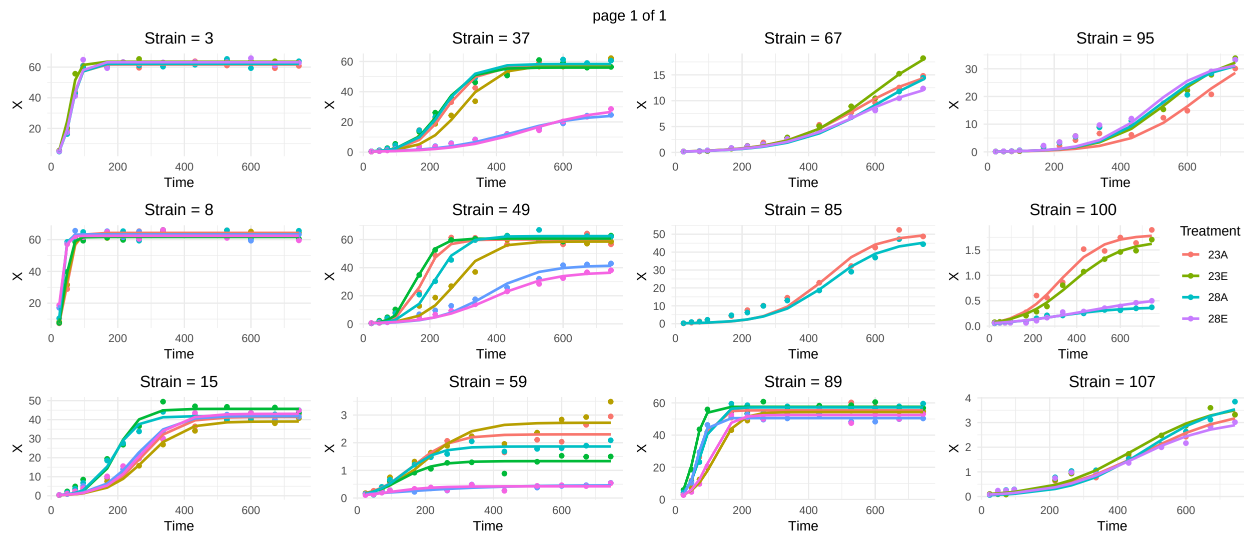
```r
model_data <- wouter_data %>% group_by(Strain, Treatment) %>% nest()

model_data$LVE_Fit <- map(model_data$data, function(.x) {
  ddat <- as.data.frame(.x)
  #starting values for the parameters
  start_values <- sapply(1:nrow(.x), function(i, x, y) {
    if(i == NROW(y) | x[i] == 0) return(NA)
      pcgr <- ((x[i + 1] - x[i]) / x[i]) / (y[i + 1] - y[i])
    return(pcgr)
  }, x = ddat$X, y = ddat$Time)
  parms_start <- c(mu = mean(start_values, na.rm = T),
                  A = mean(start_values, na.rm = T) / max(ddat$X))

  # initial fit with the starting values above
  x0 <- ddat$X[1]
  ini_fit <- ode(y = x0, times = ddat$Time, func = LVE, parms =
                  list(A = parms_start[2], mu = parms_start[1]))
  ini_fit_df <- as.data.frame(ini_fit)
  names(ini_fit_df) <- c("Time", "X")
  #ddat %>% ggplot(aes(x = Time, y = X), group = 1) + geom_point() +
  #  geom_line(data = ini_fit_df, aes(x = Time, y = X))
  #Final Fit
    ##function to return the residuals
    SSR <- function(parms) {
      #parameters
      A <- parms[2]
      mu <- parms[1]
      #
      ini_fit <- deSolve::ode(y = x0, times = ddat$Time, func = LVE,
                              parms = list(A = A, mu = mu))
      ini_fit_df <- as.data.frame(ini_fit)
      names(ini_fit_df) <- c("Time", "X")
```

```r
      #residual
      ini_fit_df$X - ddat$X
    }
    final_fit <- minpack.lm::nls.lm(par = parms_start, fn = SSR,
                                    lower = c(-Inf, -1.0E-9))
    #extracting the final estimates
    final_estimates <- coef(final_fit)
    #summary
    final_fit_summary <- summary(final_fit)
    #predicted values for the final fits
    final_fit2 <- ode(y = x0, times = ddat$Time, func = LVE, parms =
                 list(A = final_estimates[2], mu = final_estimates[1]))
    final_fit_df <- as.data.frame(final_fit2)
    names(final_fit_df) <- c("Time", "X")
    return(list(PredictedValues = final_fit_df,
           Estimates = final_fit_summary$coefficients[, 1:2])
    )
})
#separating things
#predicted values in a column
model_data$PredictedValues <- map(model_data$LVE_Fit,
                                  function(.x) return(.x$PredictedValues))
#estimates in another column
model_data$Estimates <- map(model_data$LVE_Fit, function(.x) {
  return(cbind(Parameter = c("mu", "A"),
               as.data.frame(.x$Estimates)) )
  })
```

Plotting Fitted Values

To plot the predicted values against the observed data for the whole dataset

```r
plotdata <- model_data %>% unnest(data, PredictedValues)
allplots <- map(unique(plotdata$Strain), function(.x) {
    ggplotGrob(plotdata %>% dplyr::filter(Strain == .x) %>%
      ggplot(aes(x = Time, group = Treatment, color = Treatment)) +
      geom_point(aes(y = X), shape = 19) +
      geom_line(aes(y = X1), size = 1) +
      theme_minimal() + ggtitle(paste0("Strain = ", .x)) + guides(colour = "none") +
        theme(plot.title = element_text(hjust = 0.5))
  )
})
allplots[[11]] <- ggplotGrob(plotdata %>%
                               dplyr::filter(Strain == 100) %>%
      ggplot(aes(x = Time, group = Treatment, color = Treatment)) +
      geom_point(aes(y = X), shape = 19) +
      geom_line(aes(y = X1), size = 1) +
      theme_minimal() + ggtitle(paste0("Strain = ", 100))  +
        theme(plot.title = element_text(hjust = 0.5))
)
gridExtra::marrangeGrob(allplots, nrow = 3, ncol = 4)
```



Plotting Estimated Parameter Values

To extract the parameter estimates with trheir standard errors.

```
Estimates_data <- model_data %>% unnest(Estimates)
write_csv(Estimates_data, "Data/LotkaVolterraEstimates.csv")
knitr::kable(Estimates_data[1:38, ], format = "markdown")
```

Strain	Treatment	Parameter	Estimate	Std. Error
3	23A	mu	0.0677981	0.0026117
3	23A	A	0.0010986	0.0000463
3	23E	mu	0.0795186	0.0036597
3	23E	A	0.0012517	0.0000624
3	28A	mu	0.0683192	0.0028716
3	28A	A	0.0010998	0.0000508
3	28E	mu	0.0674344	0.0035651
3	28E	A	0.0010673	0.0000620
8	23A	mu	0.0847320	0.0043075
8	23A	A	0.0013193	0.0000727
8	23E	mu	0.0864701	0.0053489
8	23E	A	0.0013638	0.0000909
8	28A	mu	0.1057983	0.0055422
8	28A	A	0.0017151	0.0000959
8	28E	mu	0.1731001	0.0247697
8	28E	A	0.0027442	0.0004042
8	35A	mu	0.1395318	0.0174563
8	35A	A	0.0022042	0.0002834
8	35E	mu	0.1346099	0.0177761
8	35E	A	0.0021581	0.0002925
15	23A	mu	0.0195198	0.0006440
15	23A	A	0.0004713	0.0000226
15	23E	mu	0.0182351	0.0005421
15	23E	A	0.0004671	0.0000203
15	28A	mu	0.0288481	0.0013214
15	28A	A	0.0006316	0.0000393

Strain	Treatment	Parameter	Estimate	Std. Error
15	28E	mu	0.0285528	0.0011080
15	28E	A	0.0006821	0.0000353
15	35A	mu	0.0197228	0.0005347
15	35A	A	0.0004705	0.0000182
15	35E	mu	0.0195192	0.0007102
15	35E	A	0.0004531	0.0000238
37	23A	mu	0.0222107	0.0008457
37	23A	A	0.0003906	0.0000213
37	23E	mu	0.0184864	0.0007087
37	23E	A	0.0003214	0.0000183
37	28A	mu	0.0226213	0.0007797
37	28A	A	0.0004042	0.0000196

```r
knitr::kable(Estimates_data[39:77, ], format = "markdown")
```

Strain	Treatment	Parameter	Estimate	Std. Error
37	28E	mu	0.0219200	0.0005359
37	28E	A	0.0003760	0.0000130
37	35A	mu	0.0095119	0.0004010
37	35A	A	0.0003748	0.0000339
37	35E	mu	0.0091147	0.0004429
37	35E	A	0.0003098	0.0000386
49	23A	mu	0.0326032	0.0011226
49	23A	A	0.0005429	0.0000245
49	23E	mu	0.0200581	0.0007851
49	23E	A	0.0003425	0.0000196
49	28A	mu	0.0351582	0.0007855
49	28A	A	0.0005788	0.0000163
49	28E	mu	0.0256725	0.0007571
49	28E	A	0.0004111	0.0000169
49	35A	mu	0.0132912	0.0006636
49	35A	A	0.0003191	0.0000270
49	35E	mu	0.0112898	0.0003200
49	35E	A	0.0003046	0.0000154
59	23A	mu	0.0177706	0.0023154
59	23A	A	0.0077211	0.0012554
59	23E	mu	0.0152266	0.0017644
59	23E	A	0.0055924	0.0008487
59	28A	mu	0.0198814	0.0031057
59	28A	A	0.0149019	0.0027391
59	28E	mu	0.0208252	0.0015907
59	28E	A	0.0111749	0.0010270
59	35A	mu	0.0069462	0.0022015
59	35A	A	0.0149808	0.0060859
59	35E	mu	0.0161922	0.0048081
59	35E	A	0.0382400	0.0128680
67	23A	mu	0.0087409	0.0002062
67	23A	A	0.0005180	0.0000378
67	23E	mu	0.0084975	0.0001752
67	23E	A	0.0003685	0.0000293
67	28A	mu	0.0081664	0.0002004

Strain	Treatment	Parameter	Estimate	Std. Error
67	28A	A	0.0004396	0.0000427
67	28E	mu	0.0085054	0.0001841
67	28E	A	0.0005974	0.0000401
85	23A	mu	0.0130186	0.0005594

```r
knitr::kable(Estimates_data[77:116, ], format = "markdown")
```

Strain	Treatment	Parameter	Estimate	Std. Error
85	23A	mu	0.0130186	0.0005594
85	23A	A	0.0002585	0.0000232
85	23E	mu	0.0118387	0.0005078
85	23E	A	0.0002527	0.0000241
89	23A	mu	0.0516656	0.0023527
89	23A	A	0.0009331	0.0000469
89	23E	mu	0.0300639	0.0008527
89	23E	A	0.0005532	0.0000179
89	28A	mu	0.0674732	0.0036137
89	28A	A	0.0011721	0.0000683
89	28E	mu	0.0467298	0.0019511
89	28E	A	0.0008169	0.0000375
89	35A	mu	0.0652511	0.0022302
89	35A	A	0.0012906	0.0000490
89	35E	mu	0.0360469	0.0024634
89	35E	A	0.0006867	0.0000520
95	23A	mu	0.0096343	0.0004042
95	23A	A	0.0002550	0.0000494
95	23E	mu	0.0108826	0.0004588
95	23E	A	0.0003041	0.0000390
95	28A	mu	0.0115022	0.0005206
95	28A	A	0.0003465	0.0000422
95	28E	mu	0.0120725	0.0005380
95	28E	A	0.0003693	0.0000407
100	23A	mu	0.0104411	0.0005100
100	23A	A	0.0057974	0.0004474
100	23E	mu	0.0086572	0.0002864
100	23E	A	0.0051278	0.0003058
100	28A	mu	0.0065017	0.0005235
100	28A	A	0.0171981	0.0022441
100	28E	mu	0.0051298	0.0004205
100	28E	A	0.0085506	0.0016106
107	23A	mu	0.0077931	0.0003583
107	23A	A	0.0021860	0.0002428
107	23E	mu	0.0083795	0.0003459
107	23E	A	0.0022097	0.0002006
107	28A	mu	0.0090329	0.0005652
107	28A	A	0.0023242	0.0003607
107	28E	mu	0.0084761	0.0005447
107	28E	A	0.0027034	0.0003856