Supplemental Material For

Multiple harvest crops: a differentiated approach for experiment evaluations

Diel, M. I; Lúcio, A.D.; Sari, B. G

This document contains a step-by-step analysis of data from an experiment with multiple harvest crops. Data from four treatments applied to strawberry cultivation were used, and the analysis was performed with the adjustment of non-linear models (logistic).

SOFTWARE USED FOR DATA ANALYSIS

Many computer software can be used for data analysis. In this approach, the software R (R Core Team, 2019), because it is free and open source. The R packages needed to reproduce the examples available in supplementary material must be previously installed. . Before analyzing the data, it is recommended that the researcher make descriptive analyzes of the data in order to verify that they are in agreement. In addition, the researcher must make a prior selection of models that he wants to adjust to his data.

The R codes are displayed, and all steps of the analysis are followed

ORGANIZE THE DATA TO MODEL PRODUCTION

The data used in the example are from an experiment conducted in a randomized block design, with a strawberry cultivar, and four fertilization treatments (Organomineral, chemical, mixed (organomineral + chemical) and peat substrate with mixed fertilization) T1, T2, T3 and T4 respectivey, in four repetitions and the experimental unit consisting of eight plants. After the fruits were red in color, they were harvested every seven days, totaling seven harvests. The variable evaluated in this example is the mass of fruits per plant (g plant-1).

The fruit mass must be accumulated in each harvest (from the first to the last) C1, C1+C2, C1+C2+C3, C1+. . . . . +C7, for each of the experimental treatments. The adjustment of the non-linear regression model can be performed by means of the repetitions of treatments or for each of the repetitions, in case of individual assessment of each plant that makes up the plot. The decision as to whether the adjustment will be made for the average of repetitions or each repetition rests with the researcher. It was decided, for example, to use the means of the repetitions:

```
Data=read.table("https://pastebin.com/raw/gU1RRVfX", header =T)
head(Data,7)
```

```
##   Treat Harvest DAT       Mass       Num
## 1    T1       1 175   3.729167  0.125000
## 2    T1       2 182  25.687500  1.020833
## 3    T1       3 189  57.875000  2.854167
## 4    T1       4 196 102.166667  6.145833
## 5    T1       5 203 141.250000  9.770833
## 6    T1       6 210 159.291667 11.520833
## 7    T1       7 217 161.437500 11.750000
```

SOFTWARE AND PACKAGES USED FOR DATA ANALYSIS

In this approach, the software R (R Core Team, 2019), because it is free and open source. The R packages needed to reproduce the examples in this document must be previously installed with the install.packages () function and loaded with the require () function.

```
require (metan)
require (nlme)
require (nlstools)
require (MASS)
require (lmtest)
require (car)
require (ggplot2)
require(cowplot)
```

```
desc_stat(Data,
          Mass,
          by=Treat)
```

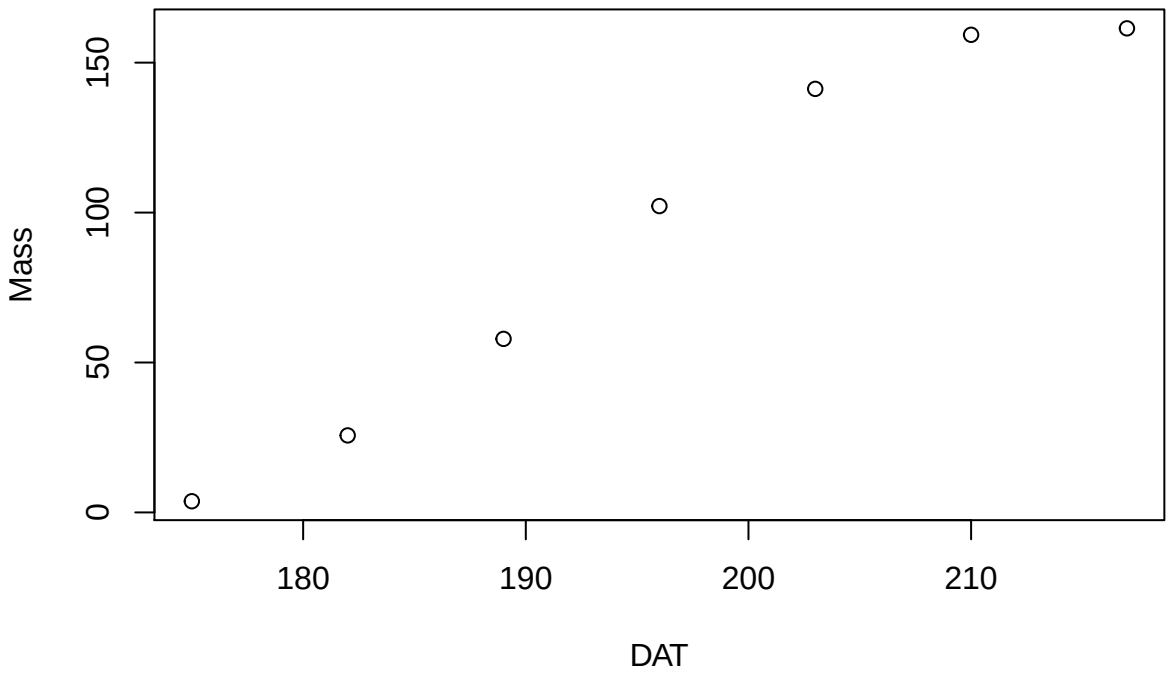
```
## # A tibble: 4 x 10
##   Treat variable    cv   max  mean median   min sd.amo    se    ci
##   <chr> <chr>    <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 T1    Mass      69.7  161.  93.1  102.   3.73   64.8  24.5  60.0
## 2 T2    Mass      82.3  277. 143.   147.   3.56  117.   44.3 108.
## 3 T3    Mass      72.9  160.  91.6   98.0  1.88   66.8  25.2  61.8
## 4 T4    Mass      63.9  172.  98.7  106.   5.29   63.1  23.8  58.3
```

ADJUSTING THE LOGISTIC MODEL FOR FRUIT MASS

The model used will be the logistics $[y = \frac{\beta_1}{1+\exp(\beta_2-\beta_3 x)} + e]$

The variable that will be adjusted next will be the Mass variable and will be adjusted initially for the T1 treatment. Note that the variable has sigmoid growth, characteristic of a non-linear model.

```
plot (Mass ~ DAT, data = subset (Data, Treat == "T1"))
```




```
logi = function (x,b0,b1,b2){
  b0/(1+exp(b1-b2*x))
}
```

For each treatment the model will be adjusted:

```
log_T1=(nls(Mass~b0/(1+exp(b1-b2*DAT)),
                  data=subset(Data, Treat=="T1"),
                  start=list(b0 = 166, b1 = 30, b2 = 0.15)))
log_T1
```

```
## Nonlinear regression model
##   model: Mass ~ b0/(1 + exp(b1 - b2 * DAT))
##    data: subset(Data, Treat == "T1")
##       b0       b1       b2
## 166.1665  32.2891   0.1674
##  residual sum-of-squares: 37.94
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 6.756e-06
```

As an example, the result of the adjustment for the T1 treatment and then for the other treatments was represented:

```
log_T2=(nls(Mass~b0/(1+exp(b1-b2*DAT)),
                  data=subset(Data, Treat=="T2"),
                  start=list(b0 = 250, b1 = 38, b2 = 0.2)))
```

```
log_T3=(nls(Mass~b0/(1+exp(b1-b2*DAT)),
                   data=subset(Data, Treat=="T3"),
                   start=list(b0 = 165, b1 = 35, b2 = 0.20)))

log_T4=(nls(Mass~b0/(1+exp(b1-b2*DAT)),
                   data=subset(Data, Treat=="T4"),
                   start=list(b0 = 175, b1 =25, b2 = 0.13)))
```

TEST THE ASSUMPTIONS OF THE MATHEMATICAL MODEL

Shapiro-Wilk tests (Shapiro and Wilk, 1965)

```
SW_T1 = shapiro.test(residuals(log_T1))
SW_T1
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(log_T1)
## W = 0.91819, p-value = 0.4554
```

```
SW_T2 = shapiro.test(residuals(log_T2))
SW_T2
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(log_T2)
## W = 0.91794, p-value = 0.4536
```

```
SW_T3 = shapiro.test(residuals(log_T3))
SW_T3
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(log_T3)
## W = 0.89955, p-value = 0.3283
```

```
SW_T4 = shapiro.test(residuals(log_T4))
SW_T4
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(log_T4)
## W = 0.91219, p-value = 0.4112
```

Breusch-Pagan test (Breusch and Pagan, 1979)

```
gradiente=attr(log_T1$m$fitted(),"gradient")
m0_T1 <- lm(Mass~-1+gradiente, data=subset(Data, Treat=="T1"))
```

```
BP_T1 = bptest(m0_T1)
BP_T1
```

```
##
##  studentized Breusch-Pagan test
##
## data:  m0_T1
## BP = 1.807, df = 2, p-value = 0.4052
```

```
gradiente=attr(log_T2$m$fitted(),"gradient")
m0_T2 <- lm(Mass~-1+gradiente, data=subset(Data, Treat=="T2"))

BP_T2 = bptest(m0_T2)
BP_T2
```

```
##
##  studentized Breusch-Pagan test
##
## data:  m0_T2
## BP = 5.3011, df = 2, p-value = 0.07061
```

```
gradiente=attr(log_T3$m$fitted(),"gradient")
m0_T3 <- lm(Mass~-1+gradiente, data=subset(Data, Treat=="T3"))

BP_T3 = bptest(m0_T3)
BP_T3
```

```
##
##  studentized Breusch-Pagan test
##
## data:  m0_T3
## BP = 4.5046, df = 2, p-value = 0.1052
```

```
gradiente=attr(log_T1$m$fitted(),"gradient")
m0_T4 <- lm(Mass~-1+gradiente, data=subset(Data, Treat=="T4"))

BP_T4 = bptest(m0_T4)
BP_T4
```

```
##
##  studentized Breusch-Pagan test
##
## data:  m0_T4
## BP = 1.7689, df = 2, p-value = 0.4129
```

Durbin-Watson test(Durbin and Watson, 1950)

```
DW_T1 = durbinWatsonTest(m0_T1)
DW_T1
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1      -0.3750994      2.194551   0.466
##  Alternative hypothesis: rho != 0
```

```
DW_T2 = durbinWatsonTest(m0_T2)
DW_T2
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1      -0.8545776      3.583182    0.03
##  Alternative hypothesis: rho != 0
```

```
DW_T3 = durbinWatsonTest(m0_T3)
DW_T3
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1      -0.4592345      2.644361   0.968
##  Alternative hypothesis: rho != 0
```

```
DW_T4 = durbinWatsonTest(m0_T4)
DW_T4
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1      -0.3131853      1.813158    0.17
##  Alternative hypothesis: rho != 0
```

PARAMETRIC NONLINEARITY MEASURES

```
log_der=deriv3(~b0/(1+exp(b1-b2*DAT)),
               c("b0","b1","b2"),
               function(DAT,b0,b1,b2)NULL)

log_T1_der=nls(Mass~log_der(DAT,b0,b1,b2),
                data=subset(Data, Treat=="T1"),
                   start=list(b0 = 166, b1 = 30, b2 = 0.15))



rms.curv(log_T1_der)
```

```
## Parameter effects: c^theta x sqrt(F) = 0.4294
##         Intrinsic: c^iota  x sqrt(F) = 0.1188
```

```
log_T2_der=nls(Mass~log_der(DAT,b0,b1,b2),
               data=subset(Data, Treat=="T2"),
                   start=list(b0 = 250, b1 = 38, b2 = 0.2))

rms.curv(log_T2_der)
```

```
## Parameter effects: c^theta x sqrt(F) = 0.1752
##         Intrinsic: c^iota  x sqrt(F) = 0.0548
```

```
log_T3_der=nls(Mass~log_der(DAT,b0,b1,b2),
               data=subset(Data, Treat=="T3"),
                   start=list(b0 = 165, b1 = 35, b2 = 0.20))
rms.curv(log_T3_der)
```

```
## Parameter effects: c^theta x sqrt(F) = 0.6228
##         Intrinsic: c^iota  x sqrt(F) = 0.1972
```

```
log_T4_der=nls(Mass~log_der(DAT,b0,b1,b2),
               data=subset(Data, Treat=="T4"),
                   start=list(b0 = 175, b1 =25, b2 = 0.13))

rms.curv(log_T4_der)
```

```
## Parameter effects: c^theta x sqrt(F) = 1.1306
##         Intrinsic: c^iota  x sqrt(F) = 0.1938
```

SAVING THE ADJUSTMENT RESULTS

```
sink("Results_adjust.txt", append = FALSE, split = FALSE)
options(max.print = 100000)
cat("-------------------------------Treat 1-------------------------------------\n")
print(log_T1)
rms.curv(log_T1_der)
cat("\n")
cat("-------------------------------Treat 2-------------------------------------\n")
print(log_T2)
rms.curv(log_T2_der)
cat("\n")
cat("-------------------------------Treat 3-------------------------------------\n")
print(log_T3)
rms.curv(log_T3_der)
cat("\n")
cat("-------------------------------Treat 4-------------------------------------\n")
print(log_T4)
rms.curv(log_T4_der)
cat("\n")
sink()
```

ESTIMATION OF CONFIDENCE INTERVALS BY RESAMPLING

The confidence intervals (CI) of the parameters are estimated using the function nlsBoot (). The CI is obtained by the difference between the quantiles 95^{th} and 2.5^{th}, using the values of the previously adjusted model:

```
log_T1_boot = nlsBoot(log_T1, niter = 10000)

log_T2_boot = nlsBoot(log_T2, niter = 10000)

log_T3_boot = nlsBoot(log_T3, niter = 10000)

log_T4_boot = nlsBoot(log_T4, niter = 10000)
```

Save the bootstrap results

```
results_boot = data.frame(
  Treatment = c(rep("1",nrow(log_T1_boot$coefboot)),
                rep("2",nrow(log_T2_boot$coefboot)),
                rep("3",nrow(log_T3_boot$coefboot)),
                rep("4",nrow(log_T4_boot$coefboot))),
  rbind (log_T1_boot$coefboot,
         log_T2_boot$coefboot,
```

```
        log_T3_boot$coefboot,
        log_T4_boot$coefboot))
```

CRITICAL POINTS OF THE MODEL

These are estimated by equalizing the partial derivatives of the function to zero, according to the methodology described in Mischan et al. (2011) and Mischan and Pinho (2014): maximum acceleration point (MAP), inflection point (IP), maximum deceleration point (MDP) and asymptotic deceleration point (ADP). The concentration of production is defined by the difference between MDP and MAP. The CI of each critical point and the concentration of production are obtained by the difference between the quantiles 95^{th} and $2,5^{th}$.

```
XMAP = (results_boot$b1 - 1.3170)/results_boot$b2
XIP = results_boot$b1/results_boot$b2
XMDP = (results_boot$b1 + 1.3170)/results_boot$b2
XADP = (results_boot$b1 + 2.2924)/results_boot$b2
ConcenTreation = XMDP - XMAP
```

To save the results in the same table created earlier, use the results_boot function

```
results_boot$XMAP = XMAP
results_boot$XIP = XIP
results_boot$XMDP =XMDP
results_boot$XADP = XADP
results_boot$Concentration = ConcenTreation

write.table(results_boot, "results_boot .csv", sep=";", dec=".", row.names=FALSE)
```

Based on these results, it is possible to create confidence interval graphs to interpret the results and compare estimates between treatments. First, a database must be organized based on the results already obtained. To plot the results on the graph, use the function ggplot().

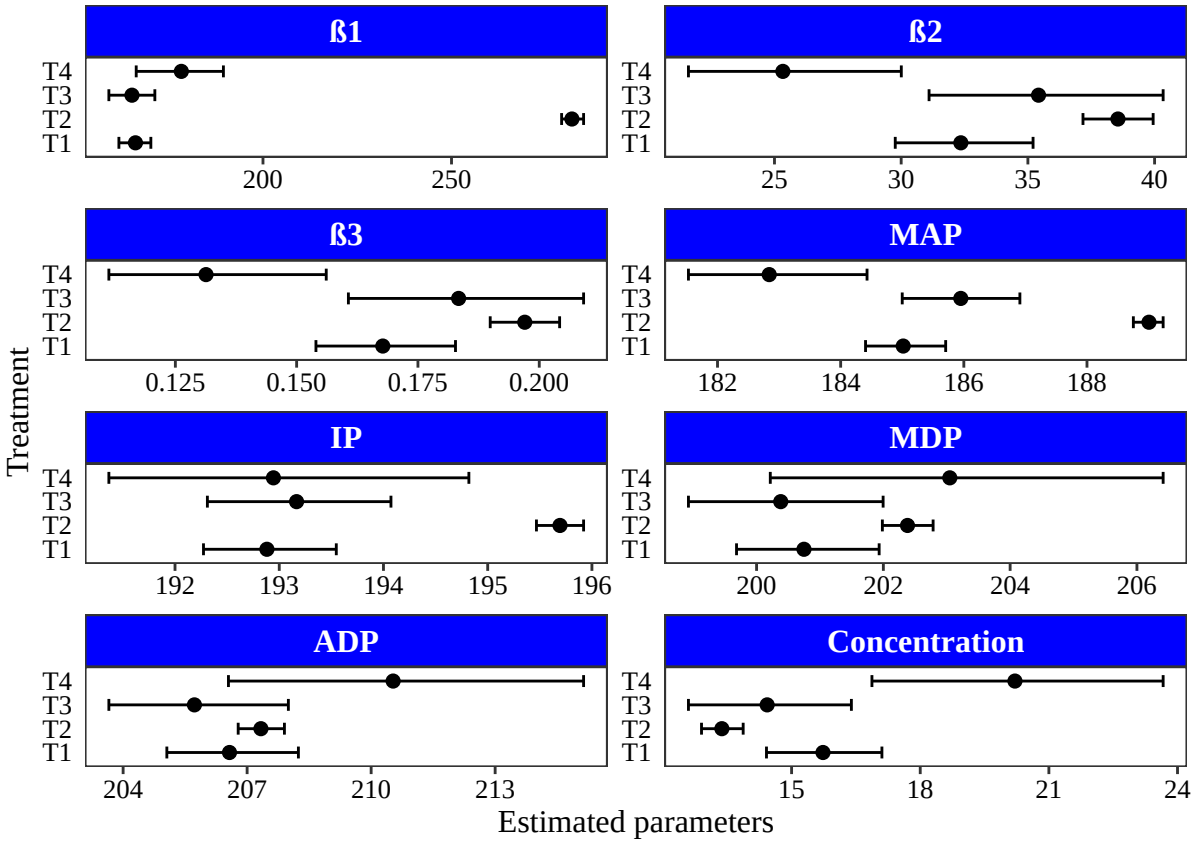
```
results_interval_LL = data.frame(matrix(nrow = 8, ncol = 4))
results_interval_Mean = data.frame(matrix(nrow = 8, ncol = 4))
results_interval_UL = data.frame(matrix(nrow = 8, ncol =4))
results_interval = data.frame(matrix(nrow = 32, ncol = 4))
names(results_interval) = c("LL","Mean","UL")

for(i in 1:4){
  Treat = subset(results_boot, Treatment==i)
  LL = c(quantile(Treat$b0, 0.025),
         quantile(Treat$b1, 0.025),
         quantile(Treat$b2, 0.025),
         quantile(Treat$XMAP, 0.025),
         quantile(Treat$XIP, 0.025),
         quantile(Treat$XMDP, 0.025),
         quantile(Treat$XADP, 0.025),
         quantile(Treat$Concentration, 0.025))
  Mean = c(mean(Treat$b0),
           mean(Treat$b1),
           mean(Treat$b2),
           mean(Treat$XMAP),
           mean(Treat$XIP),
```

```
           mean(Treat$XMDP),
           mean(Treat$XADP),
           mean(Treat$Concentration))
  UL = c(quantile(Treat$b0, 0.975),
         quantile(Treat$b1, 0.975),
         quantile(Treat$b2, 0.975),
         quantile(Treat$XMAP, 0.975),
         quantile(Treat$XIP, 0.975),
         quantile(Treat$XMDP, 0.975),
         quantile(Treat$XADP, 0.975),
         quantile(Treat$Concentration, 0.975))
  results_interval_LL[i] = LL
  results_interval_Mean[i] = Mean
  results_interval_UL[i] = UL
}

results_interval$LL = c(
  results_interval_LL$X1,
  results_interval_LL$X2,
  results_interval_LL$X3,
  results_interval_LL$X4)

results_interval$Mean = c(
  results_interval_Mean$X1,
  results_interval_Mean$X2,
  results_interval_Mean$X3,
  results_interval_Mean$X4)

results_interval$UL = c(
  results_interval_UL$X1,
  results_interval_UL$X2,
  results_interval_UL$X3,
  results_interval_UL$X4)

results_interval$Treat = c(rep("T1",8),rep("T2",8),rep("T3",8), rep ("T4",8))
results_interval$Var = c("b1","b2",
                         "b3","MAP","IP","MDP","ADP","Concentration")

beta1 <- paste0('\u03b2', '\u2081')
beta2 <- paste0('\u03b2', '\u2082')
beta3<- paste0('\u03b2', '\u2083')

results_interval$Var = factor(results_interval$Var,
                              levels=c("b1",
                                       "b2",
                                       "b3",
                                       "MAP",
                                       "IP",
                                       "MDP",
                                       "ADP",
                                       "Concentration"),
                              labels = c("b1"= beta1,
                                         "b2"= beta2,
```

```
                                        "b3"=beta3,
                                        "MAP",
                                        "IP",
                                        "MDP",
                                        "ADP",
                                        "Concentration"))
```

Graph of the confidence intervals of the parameters and critical points of the function

```
ggplot(results_interval, aes(y = Mean, x = factor(Treat), fill = Var)) +
  geom_point(size = 2) +
  geom_errorbar(aes(ymax = UL,
                    ymin = LL),
                width = 0.5) +
  facet_wrap( ~ Var, scales = "free", ncol=2) +
  coord_flip() +
  theme_bw() +
  theme(axis.text.y = element_text(family = "serif", size = 10, colour = "black"),
        axis.text.x = element_text(family = "serif",size = 10, colour = "black"),
        axis.title = element_text(family = "serif",size = 12, colour = "black"),
        axis.ticks.y = element_blank(),
        strip.background = element_rect(fill="blue"),
        strip.text.x = element_text(family = "serif",size=12,face="bold",color="white"),
        legend.position = "none",
        legend.text = element_blank(),
        legend.title = element_blank(),
        plot.margin = margin(0.2, 0.2, 0.2, 0.7, "cm"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())+
  labs(x = "Treatment", y = "Estimated parameters")
```



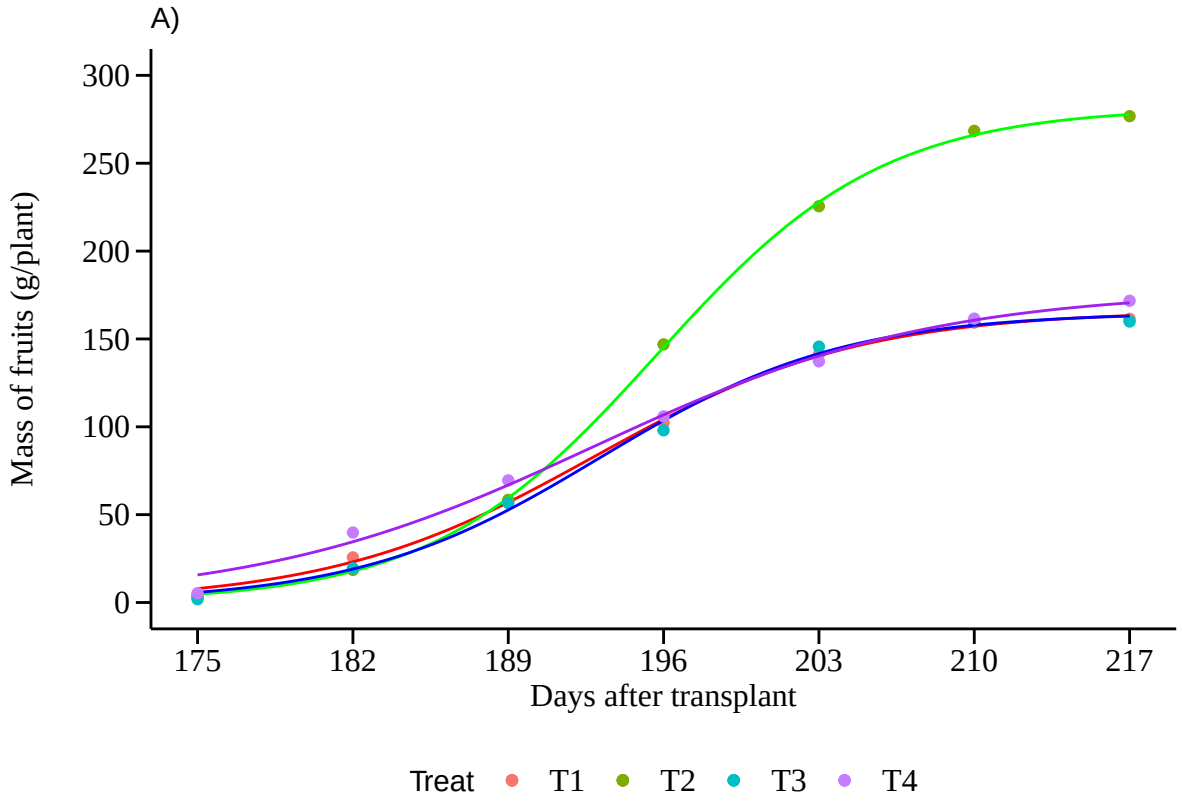
Model Critical Points Chart

A graph plotting the adjusted model (A), the inflection point (IP) (B), and all critical points, IP, MAP, MDP, and ADP (C) can be performed using the function ggplot (). First, the logistic function is plotted for the four experimental treatments evaluated:

```
formula = as.formula("y ~ b0/(1 + exp(b1-b2*x))")

A=ggplot(Data,aes(x = DAT, y = Mass, colour= Treat)) +
  geom_point(data=subset(Data, Treat=="T1"))+
  geom_smooth(method = "nls",
              method.args = list(formula = formula,
                                 start = list(b0 = 166, b1 = 30, b2 = 0.15)), se = F,
              data=subset(Data, Treat=="T1"),lwd=0.5, colour="red") +
  geom_point(data=subset(Data, Treat=="T2"))+
  geom_smooth(method = "nls",
              method.args = list(formula = formula,
                                 start = list(b0 = 250, b1 = 38, b2 = 0.2)), se = F,
              data=subset(Data, Treat == "T2"),lwd=0.5, colour="green") +
  geom_point(data=subset(Data, Treat == "T3"))+
  geom_smooth(method = "nls",
              method.args = list(formula = formula,
                                 start = list(b0 = 165, b1 = 35, b2 = 0.20)), se = F,
              data=subset(Data, Treat == "T3"),lwd=0.5, colour="blue")+
  geom_point(data=subset(Data, Treat == "T4"))+
  geom_smooth(method = "nls",
```

```
            method.args = list(formula = formula,
                               start = list(b0 = 175, b1 =25, b2 = 0.13)), se = F,
            data=subset(Data, Treat == "T4"),lwd=0.5, colour="purple")+
  theme_bw()+
  theme_classic()+
  theme(axis.ticks.length = unit(.2, "cm"),
        axis.text = element_text(family = "serif",size = 12, colour = "black"),
        axis.title = element_text(family = "serif",size = 12, colour = "black"),
        axis.ticks = element_line(colour = "black"),
        plot.margin = margin(0.5, 0.5, 0.2, 0.6, "cm"),
        axis.title.y = element_text(margin = margin(r=16)),
        legend.position = "bottom",
        legend.text = element_text(family = "serif",size=12),
        panel.grid.major.x = element_blank(), panel.grid.major.y = element_blank(),
        panel.grid.minor.x = element_blank(), panel.grid.minor.y = element_blank())+
  labs(subtitle = "A)", x = "Days after transplant", y = "Mass of fruits (g/plant)")+
  scale_x_continuous(limits = c(175, 217),
                     breaks = seq(175, 217,7))+
  scale_y_continuous(limits = c(0,300),
                     breaks = seq(0,300, 50))
```

```
A
```



From this moment on, it is possible to form a new database from the results of the estimates of the model parameters and their critical points, to plot them in new graphs. For this, the following script is used:

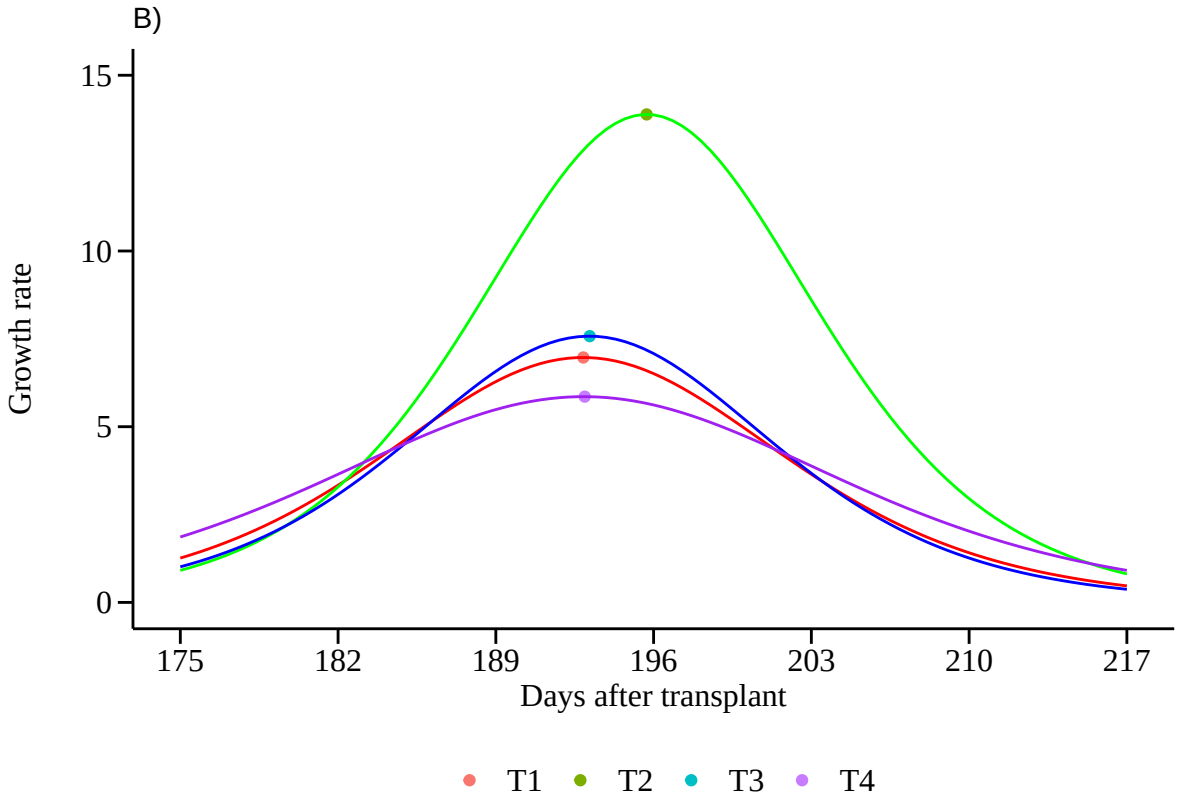
```
CRITP = c(rep("IP",4),rep("MAP",4),rep("MDP",4), rep ("ADP",4), rep("IP",4))
Treatment = Treat = rep (c("T1", "T2", "T3", "T4"),5)
PC= c (results_interval_Mean$X1[5],
       results_interval_Mean$X2[5],
       results_interval_Mean$X3[5],
       results_interval_Mean$X4[5],
       results_interval_Mean$X1[4],
       results_interval_Mean$X2[4],
       results_interval_Mean$X3[4],
       results_interval_Mean$X4[4],
       results_interval_Mean$X1[6],
       results_interval_Mean$X2[6],
       results_interval_Mean$X3[6],
       results_interval_Mean$X4[6],
       results_interval_Mean$X1[7],
       results_interval_Mean$X2[7],
       results_interval_Mean$X3[7],
       results_interval_Mean$X4[7],
       results_interval_Mean$X1[5],
       results_interval_Mean$X2[5],
       results_interval_Mean$X3[5],
       results_interval_Mean$X4[5])

b0= rep(c(results_interval_Mean$X1[1],
      results_interval_Mean$X2[1],
      results_interval_Mean$X3[1],
      results_interval_Mean$X4[1]),5)

b1= rep(c(results_interval_Mean$X1[2],
      results_interval_Mean$X2[2],
      results_interval_Mean$X3[2],
      results_interval_Mean$X4[2]),5)

b2= rep(c(results_interval_Mean$X1[3],
      results_interval_Mean$X2[3],
      results_interval_Mean$X3[3],
      results_interval_Mean$X4[3]),5)



IP1 <- data.frame(CRITP, Treat, PC, b0, b1, b2)
IP1$Treat = as.character(IP1$Treat)
IP1$CRITP = as.character(IP1$CRITP)
```

Graph B

```
critpoints = function(data){
  D1 = data.frame(matrix(nrow = 4, ncol = 4))
  D2 = data.frame(matrix(nrow = 16, ncol = 4))
  names(D1) = c("point", "Treat", "x", "y")
  names(D2) = c("point", "Treat", "x", "y")
  for (i in 1:nrow(data)){
    if(i <= 4){
      x = data[i, 3]
```

```
      b0 = data[i, 4]
      b1 = data[i, 5]
      b2 = data[i, 6]

      D1[i, 1] = data[i, 1]
      D1[i, 2] = data[i, 2]
      D1[i, 3] = data[i, 3]
      D1[i, 4] =  b0*b2*exp(b1-b2*x)/(1+exp(b1-b2*x))^2

    }

    if(i > 0){
      x = data[i,3]
      b0 = data[i, 4]
      b1 = data[i, 5]
      b2 = data[i, 6]

      D2[i, 1] = data[i, 1]
      D2[i, 2] = data[i, 2]
      D2[i, 3] = data[i, 3]
      D2[i, 4] = (b0*(b2^2)*(exp(b1-b2*x))*(exp(b1-b2*x)-1))/((1 + exp(b1-b2*x))^3)




    }

  }
  return(list(D1 = D1,
              D2 = D2))

}

crit = critpoints(IP1)

# PONTO DE INFLEXãO IP
sf1 = as.function(alist(x =, b0 = , b1 = , b2 = ,  b0*b2*exp(b1-b2*x)/(1+exp(b1-b2*x))^2))
IP1 = as.data.frame(IP1)


IP2 = subset(IP1,  CRITP == "IP")
b01 = IP2[1,4]
b02 = IP2[2,4]
b03 = IP2[3,4]
b04 = IP2[4,4]
b11 = IP2[1,5]
b12 = IP2[2,5]
b13 = IP2[3,5]
b14 = IP2[4,5]
b21 = IP2[1,6]
b22 = IP2[2,6]
b23 = IP2[3,6]
b24 = IP2[4,6]
```

```
B=ggplot(data.frame(x=seq(175,210)), aes(x, colour= Treat))+
  geom_point(aes(x, y), data = subset(crit$D1))+
  stat_function(fun = sf1, args = list(b0 = b01, b1 = b11, b2 = b21), colour="red")+
  stat_function(fun = sf1, args = list(b0 = b02, b1 = b12, b2 = b22), colour="green")+
  stat_function(fun = sf1, args = list(b0 = b03, b1 = b13, b2 = b23), colour="blue")+
  stat_function(fun = sf1, args = list(b0 = b04, b1 = b14, b2 = b24), colour="purple")+
  theme_bw()+
  theme_classic()+
  theme(axis.ticks.length = unit(.2, "cm"),
        axis.text = element_text(family = "serif", size = 12, colour = "black"),
        axis.title = element_text(family = "serif",size = 12, colour = "black"),
        axis.ticks = element_line(colour = "black"),
        plot.margin = margin(0.5, 0.5, 0.2, 0.6, "cm"),
        axis.title.y = element_text(margin = margin(r=16)),
        legend.title = element_blank(),
        legend.position = "bottom",
        legend.text = element_text(family = "serif",size=12),
        panel.grid.major.x = element_blank(), panel.grid.major.y = element_blank(),
        panel.grid.minor.x = element_blank(), panel.grid.minor.y = element_blank())+
  labs(subtitle = "B)", x = "Days after transplant", y = "Growth rate")+
  scale_x_continuous(limits = c(175, 217),
                     breaks = seq(175, 217,7))+
  scale_y_continuous(limits = c(0,15),
                     breaks = seq(0,15, 5))

B
```

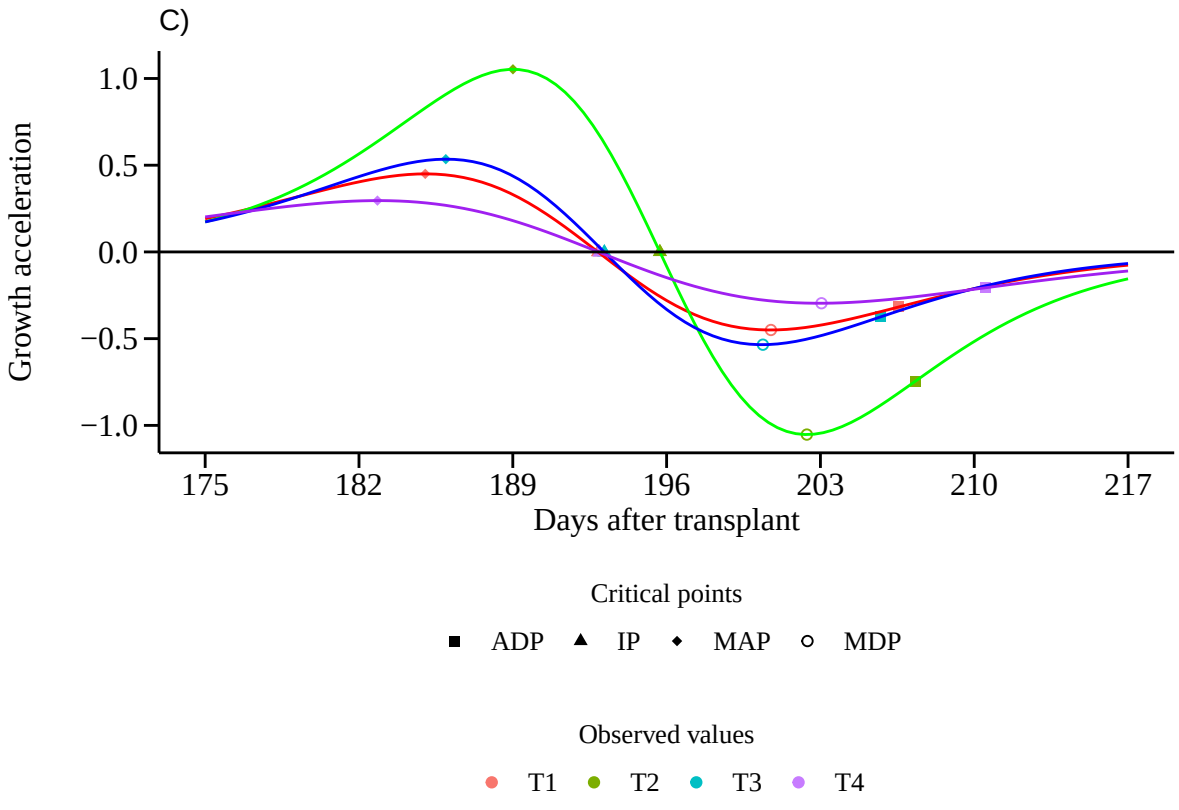


For graph C, the following code is executed:

```
sf2 = as.function(alist(x =, b0 = , b1 = , b2 = ,  (b0*(b2^2)*(exp(b1-b2*x))*(exp(b1-b2*x)-1))/((1 + ex

IP2 = subset(IP1, CRITP == "MAP")
b01 = IP2[1,4]
b02 = IP2[2,4]
b03 = IP2[3,4]
b04 = IP2[4,4]
b11 = IP2[1,5]
b12 = IP2[2,5]
b13 = IP2[3,5]
b14 = IP2[4,5]
b21 = IP2[1,6]
b22 = IP2[2,6]
b23 = IP2[3,6]
b24 = IP2[4,6]

C=ggplot(data.frame(x=seq(175,210)), aes(x, colour = Treat))+
  geom_point(aes(x, y, colour = Treat, shape = point), data = crit$D2)+
  stat_function(fun = sf2, args = list(b0 = b01, b1 = b11, b2 = b21), colour="red")+
  stat_function(fun = sf2, args = list(b0 = b02, b1 = b12, b2 = b22), colour="green")+
  stat_function(fun = sf2, args = list(b0 = b03, b1 = b13, b2 = b23), colour="blue")+
  stat_function(fun = sf2, args = list(b0 = b04, b1 = b14, b2 = b24), colour="purple")+
  theme_bw()+
  theme_classic()+
  theme(axis.ticks.length = unit(.2, "cm"),
```

```
        axis.text = element_text(family = "serif",size = 12, colour = "black"),
        axis.title = element_text(family = "serif",size = 12, colour = "black"),
        axis.ticks = element_line(colour = "black"),
        plot.margin = margin(0.5, 0.5, 0.2, 0.6, "cm"),
        axis.title.y = element_text(margin = margin(r=16)),
        legend.position = "bottom",
        legend.box = "vertical",
        legend.text = element_text(family = "serif",size=10),
        legend.title = element_text(family = "serif",size=10),
        legend.title.align = 0.5,
        panel.grid.major.x = element_blank(), panel.grid.major.y = element_blank(),
        panel.grid.minor.x = element_blank(), panel.grid.minor.y = element_blank())+
  scale_shape_manual(values=c(15, 17, 18, 1))+
  labs(color="Observed values", shape="Critical points")+
  guides(colour = guide_legend(title.position = "top", family = "serif"),
         shape=guide_legend(title.position = "top"))+
  labs(subtitle = "C)", x = "Days after transplant", y = "Growth acceleration")+
  geom_hline(yintercept = 0) +
  scale_x_continuous(limits = c(175, 217),
                     breaks = seq(175, 217, 7))
```

```
c
```



Finally, it is possible to group the three figures created into a single one, using the plot_grid () function of the cowplot package. Then just export the single figure created, in the desired size, to your directory where it will be archived for later use in the manuscripts.

```
plot_grid(A, B, C, ncol=1,
                   rel_heights = c(1,1,1.5))
```