A procedure to quantify the feed intake response of growing pigs to perturbations

Hieu Nguyen Ba, Jaap van Milgen, Masoomeh Taghipoor

August 05th 2019

Contents

Note:

- Authors of this documment are currently employed by French National Institute for Agricultural Research (INRA) at ***PEGASE*** and ***MoSAR*** units.
- The project receives fundings from European Union's H2020 project ***Feed-a-Gene*** under grant agreement no. 633531, INRA meta-programme Adaptation of Agriculture and Forests to Climate Change ***INRA-ACCAF*** and from the French Digital Agricultural Convergence Lab ***#DigitAg***.
- This R-script supports the article - the same title published in ***Animal*** - in describing the process of a data analysis procedure and modelling to detect and characterize the response of an individual pig to a perturbation.
- The process includes 1 step of missing data treatment and 3 steps of data analysis and modelling.

Change working directory and load data

```
rm(list=ls(all=TRUE)) #To remove the hisory
#Set working directory
setwd("C:/Users/hnguyenba/Desktop/Supplementary Materials")

#Packages
 library(ggplot2)
 library(dplyr)
 library(minpack.lm)
 library(nlstools)
 library(proto)
 library(nls2)
 library(fda)
 library(deSolve)

#Load dataset
 load("Data.RData")

# If the dataset includes more than one animal, choose animal's order number
 ID <- unique(JRP_NA$ANIMAL_ID)
 idc <- 1
 i <- ID[idc]
 Data <- JRP_NA[JRP_NA$ANIMAL_ID == i,]

#Vector contains Age of the chosen animal
 Age.plot <- Data$Age.plot

#Vector contains Daily Feed Intake (DFI) of the chosen animal
 DFI.plot <- Data$DFI.plot

#Vector contains Cumulative Feed Intake (CFI) of the chosen animal
 CFI.plot <- Data$CFI.plot

#Save data as a backup
JRP_new <- as.data.frame(cbind(Age.plot,DFI.plot, CFI.plot))
JRP_new_initial <-JRP_new
```

Step 1: Treatment of Missing Data

1.1. Special cases of missing data

Before detecting and estimating missing data we have to consider some special cases of missing data:

- Case 1: if there is missing data in the beginning of observations and number of days needed before missing data is not enough for estimation, the data from the beginning until missing data is removed from the dataset, e.g. if the dataset starts at day 80 and the first missing day is at day 82 so all these data are removed from the dataset (the dataset then starts at day 83)
- Case 2: if there are several series of missing data and the number of available data between two missing series is not enough for estimation (e.g. there are missing data at days 104, 105, 107 and 108) they are merged with each other and are considered as one missing series

- Case 3: if there is missing data in the end and number of days needed after missing data is not enough for estimation the data from the missing data onward is removed from the dataset, e.g. if the dataset ends at day 188 and the last missing days are at days 185 and 186, all these tree data are removed from the dataset (the dataset then ends at day 184)

a. Detect positions of missing data

```
#Expected number of rows
 Exp.row <- max(Age.plot) - min(Age.plot)+1

#observed number of rows
 obs.row <- as.numeric(length(Age.plot))

#number of missing days
 NumMissRow <-  Exp.row - obs.row

#To locate the missing rows
#if age of row_(n+1) - row_n > 1 then n is a missing row
 A1 <- Age.plot[1: length(Age.plot)-1] #
 A2 <- Age.plot[2:length(Age.plot)]
 A3 <- A2 - A1

#Days before missing series
 MissRow <- as.data.frame(JRP_new[A3!=1,])
 Missdays <- MissRow$Age.plot

#Extract A3 values different to 1
 A4 <- A3[A3!=1]
 if(length(A4) ==0){
     paste( "There are 0 missing data detected in the pig", i)
  } else {
     paste( "There are", length(A4), "missing position(s) detected from the pig", i)
 }
```

```
## [1] "There are 0 missing data detected in the pig 1"
```

b. List of ages associated with missing rows

```
#An empty list that will gather all missing rows for a given pig
 MissAgeT <- list()

#This dataframe will be used to compare with JRP_new_Initial if needed
 JRP_new_Final <-JRP_new

# Missing ages in the dataframe
 if (length(A4)>0){
     for (ii in 1:length(A4)){
          MissAgeT[[ii]] <- seq(Missdays[ii]+1,Missdays[ii]+A4[ii]-1,1)
     }
 }
```

```
if(length(A4) == 0){
  "No missing data"
  } else{
   paste( "Age (days) of missing data:", MissAgeT)
 }
```

```
## [1] "No missing data"
```

A **while loop** is created to automatically treat each special case of missing data

```
#Empty lists
b1l = list() #days before missing rows
b2l = list() #days after missing rows

Xl  = list()   #list of ages before and after missing rows
Yl  = list()   #list of CFI before and after missing rows
tal = c()      #the vector of first missing rows
dl  = list()   #list of coefficients for delta

#information of linear model
Par_initl = list() #Initial parameters for each set of missing rows
resl = list() #function optim to estimate parameters
pred_valuesl = list()
deltal = c()
P.dfil = list() #parameters of the linear function of DFI
FI.missl = list()
Corr.datal=list()
Res.RSS =list()

ki=1 #first value to start while loop

while(ki< length(MissAgeT)+1 ){

  k=ki
  MissAge = MissAgeT[[k]] #Missing rows at position k

#-----------------------------------------------------
# Extract age and CFI before and after missing data
# for the interpolation via linear regression
#-----------------------------------------------------

 L = as.numeric(length(MissAge))

#number of data before = length(missing rows) + 1
  b1l[[k]] <- seq(MissAge[1]-(length(MissAge) + 1), MissAge[1] - 1, 1)

#number of data after missing age
  b2l[[k]] <- seq( MissAge[length(MissAge)]+1,
                   MissAge[length(MissAge)]+
                   (length(MissAge)+1), 1)

#number of missing rows to be replaced at the end
```

```
  MissAge2 = MissAge

#-----------------------------------------------------
# Special cases
#-----------------------------------------------------

#1. When number of data before missing row is not enough for estimation
#E.g., 88, 89,..., 92, 93
#we remove all the rows until missing data (Age now starts at day 92)
 if(b1l[[k]][1] < Age.plot[1]){
    JRP_new <- JRP_new[!JRP_new$Age.plot %in% seq(Age.plot[1],
                                                  MissAge2[length(MissAge2)],
                                                  by=1),]
    MissAgeT[[1]] <- NULL
    MissAge = MissAgeT[[k]]
    MissAge2 = MissAge
    ki = 0
 }

 #2. when there are not enough given rows between two series of missing rows
 #E.g., 108, 109, 110, 112, 114, 115, 116
 #we merge these two missing rows with the rows between
 # when number of missing row in a serie > 1
 if(ki == 0){

   }else{
 if(length(MissAgeT)>1 && k < length(MissAgeT)){
   for(ii in 1:(length(MissAgeT)-1)){
    if(k + ii <= length(MissAgeT)){
    if(MissAgeT[[k+ii]][1] - MissAge[length(MissAge)] <= length(MissAge)+1){
      (MissAge = c(MissAge, seq(MissAge[length(MissAge)]+1,
                                MissAgeT[[k+ii]][1]-1,1),
                                MissAgeT[[k+ii]]))&&
                (b2l[[k]] <- seq( MissAge[length(MissAge)]+1,
                                  MissAge[length(MissAge)] + (length(MissAge)+1),
                                  1)) &&
                (b1l[[k]] <- seq(MissAge[1]-(length(MissAge)+1), MissAge[1]-1, 1))&&
                (ki=k+ii) &&
                (MissAge2 =  MissAge)
     }
    }
   }
  }
 }

  if(ki == 0){

   }else{
    if(b1l[[k]][1] < Age.plot[1]){
    JRP_new <- JRP_new[!JRP_new$Age.plot %in% seq(Age.plot[1],
                                                  MissAge2[length(MissAge2)],
                                                  by=1),]
 }
```

```
}

 #3. When number of data after missing row is not enough for estimation
 # E.g., 179, 180,..., 183, 184 we remove all the rows
 # from missing data (Age is now just until day 180)
 if(b2l[[k]][length(b2l[[k]])] > Age.plot[length(Age.plot)]){
   JRP_new <- JRP_new[!JRP_new$Age.plot %in%
                seq(MissAge2[1],
                    Age.plot[length(Age.plot)],
                    by=1),]
  MissAgeT[[length(MissAgeT)]] <- NULL
}

#-----------------------------------------------------
# Re-calculate CFI after filtrating process
#-----------------------------------------------------
#Age vector
 Age.plot = JRP_new$Age.plot[! JRP_new$Age.plot %in% MissAge2]

#DFI vector
 DFI.plot = JRP_new$DFI.plot[JRP_new$Age.plot %in% Age.plot]

#CFI
 CFI.plot = cumsum(DFI.plot)

#-----------------------------------------------------
# New dataset which contains Age, DFI and CFI
#-----------------------------------------------------
 JRP_new <- as.data.frame(cbind(Age.plot,DFI.plot, CFI.plot))
 ki = ki+1 #next series of missing rows

 } #end of WHILE loop
```

1.2. Estimation of Missing Data

```
#Expected number of rows
 Exp.row <- max(JRP_new$Age.plot) - min(JRP_new$Age.plot)+1

#observed number of rows
 obs.row <- as.numeric(length(JRP_new$Age.plot))

#number of missing days
 NumMissRow <-  Exp.row - obs.row

#To locate the missing rows
#if age of row_(n+1) - row_n > 1 then miss row
 A1 <- Age.plot[1: length(JRP_new$Age.plot)-1] #
 A2 <- Age.plot[2:length(JRP_new$Age.plot)]
 A3 <- A2 - A1

#Day before missing series of data
```

```
 MissRow <- as.data.frame(JRP_new[A3!=1,])
 Missdays <- MissRow$Age.plot

# Extract A3 values different to 1
 A4 <- A3[A3!=1]

#-----------------------------------------------------
# List of ages associated with missing rows
#-----------------------------------------------------

#An emptylist that will gather all missing rows for a given pig
  MissAgeT <- list()

##This dataframe will be used to compare with JRP_new_Initial if needed
 JRP_new_Final <-JRP_new

# Missing ages in the dataframe
 if (length(A4)>0){
 for (ii in 1:length(A4)){
      MissAgeT[[ii]] <- seq(Missdays[ii]+1,Missdays[ii]+A4[ii]-1,1)
     }
 }

 if(length(A4) == 0){
     "No missing data"
    } else{
    paste( "Age (days) of missing data:", MissAgeT)
  }
```

```
## [1] "No missing data"
```

```
#=====================================================
#          ESTIMATING MISSING DATA
#=====================================================

#Empty lists
 b1l = list() #days before missing rows
 b2l = list() #days after missing rows

 Xl  = list()   #list of ages before and after missing rows
 Yl  = list()   #list of CFI before and after missing rows
 tal = c()      #the vector of first missing rows
 dl  = list()   #list of coefficients for delta

#information of linear model
 Par_initl = list() #Initial parameters for each set of missing rows
 resl = list() #function optim to estimate parameters
 pred_valuesl = list()
 deltal = c()
 P.dfil = list() #parameters of the linear function of DFI
 FI.missl = list()
 Corr.datal=list()
 Res.RSS =list()
```

```
ki=1 #first value to start while loop

    while(ki< length(MissAgeT)+1 ){

      k=ki
      MissAge = MissAgeT[[k]] #Missing rows at position k

    #-----------------------------------------------------
    # Extract age and CFI before and after missing data
    # for the interpolation of linear regression
    #-----------------------------------------------------

      L = as.numeric(length(MissAge))

      #number of data before = length(missing rows)+1
      b1l[[k]] <- seq(MissAge[1]-(length(MissAge)+1), MissAge[1]-1, 1)

      #number of data after missing age
      b2l[[k]] <- seq( MissAge[length(MissAge)]+1,
                       MissAge[length(MissAge)]+
                         (length(MissAge)+1), 1)

      #number of missing rows to be replaced at the end
      MissAge2 = MissAge

      b1 = b1l[[k]] # Ages before missing row(s)
      b2 = b2l[[k]] # Ages after missing row(s)

      # rows associated with these ages b1, b2
      B1 <- JRP_new[JRP_new$Age.plot %in% b1,]
      B2 <- JRP_new[JRP_new$Age.plot %in% b2,]

      #CFI associated with these ages b1, b2
      CFI_b <- B1$CFI.plot
      CFI_a <- B2$CFI.plot

      #DFI associated with these ages b1, b2
      DFI_b <-B1$DFI.plot
      DFI_a <-B2$DFI.plot

    #-----------------------------------------------------
    #          Regression
    #-----------------------------------------------------

      # Put all values of age together
      Xl[[k]] <-c(b1,b2)
      Yl[[k]] <-c(CFI_b,CFI_a)

      tal[k] <- MissAge[length(MissAge)] #from this point delta is valid

      X = Xl[[k]]
      Y = Yl[[k]]
      ta = tal[k]
```

```
 dl[[k]]  <- as.numeric(X>ta)
 d = dl[[k]] #on/off condition which activates function after missing rows

#Quadratic function of estimating missing CFI values
 pred_fn = function(P){
   a0=P[1];
   a1=P[2];
   a2=P[3];
   b0=P[4];
   return ( a2*X^2  +  a1*X + a0*(1-d)+b0*d);
 };

 # 2 days before and after for calculation of initial values

 X2 = b1[length(b1)-1] #X[2]
 X3 =  b1[length(b1)] #X[3]
 X4 = b2[1] #X[4]
 X5 = b2[2]#X[5]

 Y2 = CFI_b[length(CFI_b)-1]#Y[2]
 Y3 = CFI_b[length(CFI_b)]#Y[3]
 Y4 = CFI_a[1] #Y[4]
 Y5 = CFI_a[2] #Y[5]

 #Reparametrize parameters of quadratic function to
 # parameters from 2 days before and after missing rows

 a2_init = -(-Y3*X5+Y3*X4+Y2*X5-Y2*X4+X2*Y4-X2*Y5-X3*Y4+X3*Y5)/
   (X2*X5^2-X2*X4^2-X2^2*X5+X2^2*X4-X3*X5^2+X3*X4^2+X3^2*X5-X3^2*X4);


 a1_init = (-X5^2*Y3+X5^2*Y2-Y5*X2^2+Y5*X3^2+Y4*X2^2-X4^2*Y2-Y4*X3^2+X4^2*Y3)/
   ((X5-X4)*(X5*X2-X5*X3-X2^2+X3^2+X4*X2-X4*X3));

 a0_init = -(-X5^2*Y3*X2+X5^2*Y2*X3+X5*Y3*X2^2-
               X5*Y2*X3^2+Y4*X2^2*X3-Y3*X2^2*X4-
               X2^2*X3*Y5+X2*Y5*X3^2-Y4*X3^2*X2+Y3*X2*X4^2-Y2*X3*X4^2+Y2*X3^2*X4)/
   (X2*X5^2-X2*X4^2-X2^2*X5+X2^2*X4-X3*X5^2+X3*X4^2+X3^2*X5-X3^2*X4);


 b0_init = -(-Y4*X2*X5^2+Y4*X3*X5^2+X5^2*Y2*X4-
               X5^2*Y3*X4+Y4*X2^2*X5-Y4*X3^2*X5-
               X4^2*Y2*X5+X4^2*Y3*X5-Y5*X2^2*X4+Y5*X2*X4^2-Y5*X3*X4^2+Y5*X3^2*X4)/
   (X2*X5^2-X2*X4^2-X2^2*X5+X2^2*X4-X3*X5^2+X3*X4^2+X3^2*X5-X3^2*X4);

 # initial values
 Par_initl[[k]] <- c(a0_init, a1_init, a2_init, b0_init)
 Par_init = Par_initl[[k]]


 # Using optim function to minimize Residual Sum of Squares

 obj <- function(P){
```

```
    output   <- (Y -pred_fn(P));
    obj.func <- t(output)%*%output; # Residual sum of squares
    return(obj.func);
  }

  resl[[k]] <- optim (Par_init, obj); #function optim to estimate parameters
  res = resl[[k]]

  pred_valuesl[[k]] = pred_fn(res$par)
  pred_values= pred_valuesl[[k]]


  #parameters values
  a0=res$par[1];
  a1=res$par[2];
  a2=res$par[3];
  b0=res$par[4];
  deltal[k] = a0 - b0
  delta = deltal[k]

#-----------------------------------------------------
#         Attribution of data to missing age
#-----------------------------------------------------

  #Function of DFI
  DFI_pred <- function(age,P.dfi){
    return (2*P.dfi[2]*age  +  P.dfi[1]);
  };

  #parameters of the linear function of DFI
  P.dfil[[k]] <- c(a1,a2)
  P.dfi = P.dfil[[k]]

  #sum of estimated DFI for missing days
  CFI_miss <- sum(DFI_pred(MissAge2, P.dfi))

  FI.miss = c()
  for (kj in 1:length(MissAge2)){
    FI.miss[kj] <- DFI_pred(MissAge2[kj],P.dfi)*delta /CFI_miss}
  FI.missl[[k]] = FI.miss

  #Put estimation of DFI in right places (attributed rows)
  t1 <- as.numeric(which(JRP_new$Age.plot %in% c(MissAge2-1)))
  probs <- rep(TRUE, length(DFI.plot[!c(JRP_new$Age.plot %in% MissAge2)]))
  ind <- t1[1] + MissAge2/(sum(MissAge2)+.05)
  val.FI <- c( DFI.plot[!c(JRP_new$Age.plot %in% MissAge2)], FI.miss)
  val.age <- c( Age.plot[!c(JRP_new$Age.plot %in% MissAge2)],MissAge2)


  id  <- c( seq_along(probs), ind)

  Corr.DFI <- val.FI[order(id)]
  Corr.Age <- val.age[order(id)]
```

```
   #-----------------------------------------------------
   #     Recalculate CFI based on new DFI
   #-----------------------------------------------------

     Corr.CFI = c()
     Corr.CFI[1] = Corr.DFI[1]

     for(j in 2:length(Corr.Age)){
       Corr.CFI[j] <- Corr.CFI[j-1]+Corr.DFI[j]
     }

     #New dataframe containing estimated missing data
     Corr.datal[[k]] <- as.data.frame(cbind(Corr.Age, Corr.DFI, Corr.CFI))
     Corr.data = Corr.datal[[k]]
     colnames(Corr.data) = c("Age.plot", "DFI.plot", "CFI.plot")

     #RSS and delta
     Res.RSS[[k]] <- as.data.frame(c(res$value,delta) , row.names = c("RSS", "Delta"))
     #

 #-----------------------------------------------------
 #We replace new data set in the initial data set and we restart
 #the processus while the condition for ki is respected
 #-----------------------------------------------------

     JRP_new = Corr.data
     DFI.plot= JRP_new$DFI.plot
     Age.plot= JRP_new$Age.plot
     CFI.plot= JRP_new$CFI.plot

#-----------------------------------------------------
#   PLOT GRAPHS WITH TREATMENT OF MISSING DATA
#-----------------------------------------------------

  #Vector contains estimated missing data
   X1 <- c(b1, MissAge2, b2)
   Y1 <- JRP_new$CFI.plot[JRP_new$Age.plot %in% X1] #CFI data

  #Initial Age around missing row
   Age.ini <- JRP_new_initial$Age.plot[JRP_new_initial$Age.plot >= b1[1] &
                            JRP_new_initial$Age.plot <= b2[length(b2)]]
   #Initial DFI values
   Z <- JRP_new_initial$DFI.plot[JRP_new_initial$Age.plot %in% Age.ini]
   #Estimated DFI values
   Z1 <- JRP_new$DFI.plot[JRP_new$Age.plot %in% X1]

  #To indicate the location of missing rows in the title of graphs
     A <- c()
     for(i in 1:length(MissAge2)){
       A <- paste(A, MissAge2[i])
     }

   #PLOT
```

```
    par(mar=c(4,4.5,4,4.5))
    plot(X1, Y1,
         main = paste("Pig ID:", ID[idc], "\nMissing rows:", A, sep = ""),
         ylab = "Cummulative feed intake (kg)",
         xlab = "Age (days)",
         type = "o", lwd = 2, cex.main = 1.7,
         pch = 1, cex.lab = 1.3, cex.axis = 1.2 , cex = 2, col = 'blue')
    points(X, Y,
             lwd = 2, cex = 2,  type = "o")
    legend("topleft",c("real data", "estimated data"),
             pch = c(1,1), col = c("black", "blue"), bty = "n", cex = 1)

  ki = ki+1 #next series of missing rows

} #end of WHILE loop
```

1.2.2. Remove the last day from dataset because the pig may be fasted one day before slaughtering

```
JRP_new = JRP_new[c(1:length(JRP_new$Age.plot)-1),]
```

- New dataset after estimating missing data

```
ANIMAL_ID = rep( ID[i], dim(JRP_new)[1])
JRP_new_Final <- cbind (ANIMAL_ID,JRP_new)
No.NA.Data <- JRP_new_Final
rm(list=ls()[! ls() %in% c("No.NA.Data")])
```

Step 2: Target trajectory curve of CFI (target CFI)

```
#====================
# DATA PREPARATION
#====================
  #Load needed functions
  source("Functions.R")

  #Order number of Animal_ID
  ID = unique(as.factor(No.NA.Data$ANIMAL_ID))

  Data = No.NA.Data[No.NA.Data$ANIMAL_ID == ID,]

  x <- as.numeric(Data$Age.plot)
  Y <- as.numeric(Data$CFI.plot)
  Z <- as.numeric(Data$DFI.plot)
  Data.xy <- as.data.frame(cbind(x,Y))
```

2.1. Function of the target CFI

A quadratic-linear function (**QLM**) is chosen as the principal function to describe the target CFI:

When X $<$ Xs:

$$Y = a + bX + cX^2$$

When X $\geq$ Xs:

$$Y = (a + bXs + cXs^2) + (b + 2cXs) * (X - Xs)$$

Note: in the article, **X** is called **t** and **Y** is **Target_CFI(t)**

The model is reparametrized to:

- Age at which CFI is equal 0 (**X0**)
- The day where quadratic segment changes to linear segment (**Xs**)
- Estimated DFI and CFI at age Xs (**DFIs**)
- Estimated DFI age Xs (**CFIs**)

```
#=================================================================
# Process to choose suitable function for the target CFI
#=================================================================

 #Initial parameteres for the quadratic-linear function
    X0.0 = x[1]
    Xlast = x[length(x)]

#Provide set of initial parameters
#Provide a grid contains many initial values for Xs
# and choose the set of values with smallest Residual Sum of Square
    Xs.1 = round(seq(X0.0 + 1, Xlast - 1, len = 30), digits = 0)
    X0.1 = rep(X0.0, length(Xs.1))
    DFIs.1 <- c()
    CFIs.1 <- c()
    for(A in 1:length(Xs.1)){
      DFIs2 = Data[Data$Age.plot == Xs.1[A],]$DFI.plot
      CFIs2 = Data[Data$Age.plot == Xs.1[A],]$CFI.plot
      DFIs.1 <- c(DFIs.1, DFIs2)
      CFIs.1 <- c(CFIs.1, CFIs2)
    }

    st1 = data.frame(cbind(X0.1,
                           Xs.1,
                           DFIs.1,
                           CFIs.1))
    names(st1) <- c("X0","Xs", "DFIs","CFIs")

 #-----------------------------------------------------------------
 # Choose the best initial values by using optimization in NLS2
 #-----------------------------------------------------------------

    st2 = nls2(Y ~ nls.func.2(X0, Xs, DFIs, CFIs),
               Data.xy,
               start = st1,
               # weights = weight,
               # trace = T,
```

```
                algorithm = "brute-force")
    par_init <- coef(st2)
    par_init
```

```
##      X0      Xs    DFIs    CFIs
##  68.000 155.000   2.813 216.573
```

Based on the optimized **initial parameters** the function of target CFI can be chosen as:

- ***Quadratic function (QM)***: if the Xs (chosen by the grid) is too close to either the first or last observation
- ***Linear function (LM)***: if the slope of the function's derivative (slope of DFI curve) is negative
- ***Quandratic-linear function (QLM)***: none of these options above

```
#-----------------------------------------------------------------------
# Adapt the function for each animal
#-----------------------------------------------------------------------
 #Calculate Slope of DFI curve
    c <- abcd.2(par_init)[3]

 # Determine the value of Xs
    Xs.0 <- par_init[2]

    if(c < 0){
      FuncType <- "LM"
    } else if(c > 0 &&
              Xs.0 >= ((Xlast - X0.0)*0.2 + X0.0) #this condition was set because
              ){  #if initial value of Xs is too close to X0, Xs could not be estimated
      FuncType <- "QLM"
    } else {
      FuncType <- "QDR"
    }

  if(FuncType =="LM"){
      ABC <- "linear"
  } else if(FuncType =="QDR"){
      ABC <- "quadratic"
  } else{
     ABC <- "quadratic-linear"
  }

print(paste("A ", ABC,
  " function will be applied to estimate the target CFI", sep = ""))
```

```
## [1] "A quadratic-linear function will be applied to estimate the target CFI"
```

2.2. Estimate parameters of target CFI

a. If a linear function is chosen

Parameters a and b are reparametrized to:

- **X0**: the estimated age at which value of CFI is equal to 0 (or t0 in the article)
- **Ylast**: the estimated CFI value at the last observation (or CFI_last in the article)

```
if(FuncType == "LM"){
  #Remove unneccesary data
    rm(list=ls()[! ls() %in% c("ID","Data", "FuncType", "x", "Y", "Z", "Data.xy")])

    #load needed functions
    source("Functions.R")

    xlast <- x[length(x)] # last day of observation
    #Provide initial parameteres for the linear function
    x0.0 = x[1]
    ylast.0 = Y[length(x)]

    #Vector containing 2 initial parameters
    par_init <- c(x0.0, ylast.0)

    #-------------------------------------------
    # Create empty lists to store data after loop
    #-------------------------------------------

    par = list()
    AC.res = list()
    AC.pvalue = c()
    data2 = list()
    param <- data.frame(rbind(par_init))
    par.abcd = data.frame(rbind(abcd.0(as.vector(par_init))))
    param.2 <- data.frame(X0=double(),
                          Ylast=double(),
                          a=double(),
                          b=double(),
                          stringsAsFactors=FALSE)
    j = 2
    AC_pvalue = 0
    AC.pvalue[1] = AC_pvalue
    datapointsleft = as.numeric(dim(Data)[1])
    dpl = datapointsleft #vector of all dataponits left at each step

    #-------------------------------------------------------------------------------
    # Start the filtration procedure by using Non Linear Regression
    #-------------------------------------------------------------------------------
    #
    while (AC_pvalue<=0.05 && datapointsleft >= 20){
      weight = 1/Y^2 #weighing data
      nls.CFI <- nlsLM(Y ~ nls.func.0(X0, ylast),
                       Data.xy,
                       control = list(tol = 1e-2, printEval = TRUE, maxiter = 50),
                       start = list(X0 = par_init[1],
                                    ylast = par_init[2]),
                       trace = F,
                       weights = weight)
```

```
  #--------RESULTS analysis GOODNESS of fit
  #estimate params
  par[[j]] = coef(nls.CFI)
  par.abcd[j,] = abcd.0(as.vector(coef(nls.CFI) )) #calculation of a, and b
  param[j,] = par[[j]]
  param.2[j-1,] = cbind(param[j,],par.abcd[j,])

  #residuals
  res1 = nlsResiduals(nls.CFI)
  res2= nlsResiduals(nls.CFI)$resi1
  res = res2[,2]
  AC.res = test.nlsResiduals(res1)
  AC.pvalue[j] =  AC.res$p.value

  #---------Check for negative residuals----------

  #Add filtration step order to data
  Step <- rep(j - 1, length(x))
  #create a new dataset with predicted CFI included
  Data.new <- data.frame(cbind(x, Z, Y, pred.func.0(par[[j]],x)[[1]], res, Step))
  names(Data.new) <- c("Age",
                       "Observed_DFI",
                       "Observed_CFI",
                       "Predicted_CFI",
                       "Residual",
                       "Step")

  #remove negative residuals
  Data.pos <- Data.new[!Data.new$Residual<0,]

  #restart
  datapointsleft = as.numeric(dim(Data.pos)[1])
  par_init = par[[j]]
  AC_pvalue = AC.pvalue[j]
  j = j+1
  x <- Data.pos$Age
  Y <- Data.pos$Observed_CFI
  Z <- Data.pos$Observed_DFI
  Data.xy <- as.data.frame(cbind(x,Y))
  dpl = c(dpl, datapointsleft)
}

#Create a table to store estimated parameters
ANIMAL_ID = rep( ID, dim(param.2)[1])
XLAST =  rep( xlast, dim(param.2)[1])
param.2 = cbind(ANIMAL_ID,
                param.2,
                XLAST,
                AC.pvalue[2:length(AC.pvalue)],
                dpl[1:length(dpl)-1],
                rep(FuncType, dim(param.2)[1])
                )
colnames(param.2)= c("ANIMAL_ID",
```

```
                          "X0",
                          "Ylast",
                          "a",
                          "b",
                          "Xlast",
                          "P.runs.test",
                          "DPL",
                          "FuncType"
                          )

    #Add information of slope of DFI
    Slope <- rep(0, dim(param.2)[1])
    param.2$Slope <- Slope

    #-------------------------------------------------------------------------------
    # Give the ID back to the animal
    #-------------------------------------------------------------------------------
    ANIMAL_ID = rep( ID, dim(Data.new)[1])
    Data.new = cbind(ANIMAL_ID , Data.new)
    Data.remain = Data.new[,c(1:4)]

 } else{
  }
```

b. If a quadratic function is chosen

- **Xlast** is an observation indicates the estimated age at the end of the growing period

Parameters a and b are reparametrized to:

- **X0**: the estimated age at which value of CFI is equal to 0 (or t0 in the article)
- **y2**: the estimated CFI at the midpoint (or CFI_mid-point in the article)
- **ylast**: the estimated CFI value at the last observation (or CFI_last in the article)

```
if(FuncType == "QDR"){

    #Remove unneccesary data
    rm(list=ls()[! ls() %in% c("ID","Data", "FuncType", "x", "Y", "Z", "Data.xy")])

    # Load needed functions
    source("Functions.R")

    xlast <- x[length(x)] #last day of observation
    #Initial parameteres for the quadratic function
    x0.0 = x[1]
    y2.0 = Y[floor(length(x)/2)]
    ylast.0 = Y[length(x)]

    #Vector contains 3 initial parameters
    par_init <- c(x0.0, y2.0, ylast.0)

    x2 = (xlast - x[1])/2+x[1] # day at midpoint of the dataset
```

```
#-------------------------------------------
# Create empty lists to store data after loop
#-------------------------------------------

par = list()
AC.res = list()
AC.pvalue = c()
data2 = list()
data3 = list()
param <- data.frame(rbind(par_init))
par.abcd = data.frame(rbind(abcd.1(as.vector(par_init))))
param.2 <- data.frame(X0=double(),
                      Y2=double(),
                      Ylast=double(),
                      a=double(),
                      b=double(),
                      c=double(),
                      stringsAsFactors=FALSE)
j = 2
AC_pvalue = 0
AC.pvalue[1] = AC_pvalue
datapointsleft = as.numeric(dim(Data)[1])
dpl = datapointsleft #vector of all dataponitsleft at each step

#-------------------------------------------------------------------------------
# Start the filtration procedure by using Non Linear Regression
#-------------------------------------------------------------------------------
#
while (AC_pvalue<=0.05 && datapointsleft >= 20){
  weight = 1/Y^2 #weighing data
  nls.CFI <- nlsLM(Y ~ nls.func.1(X0, y2, ylast),
                   Data.xy,
                   control = list(tol = 1e-2, printEval = TRUE, maxiter = 50),
                   start = list(X0 = par_init[1], y2 = par_init[2],
                                ylast = par_init[3]),
                   trace = F,
                   weights = weight)


  #--------RESULTS analysis GOODNESS of fit
  #estimate params
  par[[j]] = coef(nls.CFI)
  par.abcd[j,] = abcd.1(as.vector(coef(nls.CFI) )) #calculation of a, b and c
  param[j,] = par[[j]]
  param.2[j-1,] = cbind(param[j,],par.abcd[j,])

  #Calculation of days associated with Y1 and Y2
  x2[j] = (xlast - coef(nls.CFI)[1])/2+coef(nls.CFI)[1]

  #residuals
  res1 = nlsResiduals(nls.CFI)
  res2= nlsResiduals(nls.CFI)$resi1
  res = res2[,2]
```

```
  AC.res = test.nlsResiduals(res1)
  AC.pvalue[j] =  AC.res$p.value

  #---------Check for negative residuals----------

  #Add filtration step order to data
  Step <- rep(j - 1, length(x))
  #create a new dataset with predicted CFI included
  Data.new <- data.frame(cbind(x, Z, Y, pred.func.1(par[[j]],x)[[1]], res, Step))
  names(Data.new) <- c("Age",
                       "Observed_DFI",
                       "Observed_CFI",
                       "Predicted_CFI",
                       "Residual",
                       "Step")

  #remove negative res
  Data.pos <- Data.new[!Data.new$Residual<0,]

  #restart
  datapointsleft = as.numeric(dim(Data.pos)[1])
  par_init = par[[j]]
  AC_pvalue = AC.pvalue[j]
  j = j+1
  x <- Data.pos$Age
  Y <- Data.pos$Observed_CFI
  Z <- Data.pos$Observed_DFI
  Data.xy <- as.data.frame(cbind(x,Y))
  dpl = c(dpl, datapointsleft)
}

#Create a table to store estimated parameters
ANIMAL_ID = rep( ID, dim(param.2)[1])
XLAST   =  rep( xlast, dim(param.2)[1])
param.2 = cbind(ANIMAL_ID,
                param.2,
                x2[2:length(x2)],
                XLAST,
                AC.pvalue[2:length(AC.pvalue)],
                dpl[1:length(dpl)-1],
                rep(FuncType, dim(param.2)[1])
                )
colnames(param.2) = c("ANIMAL_ID",
                      "X0",
                      "Y2",
                      "Ylast",
                      "a",
                      "b",
                      "c",
                      "X2",
                      "Xlast",
                      "P.runs.test",
                      "DPL",
```

```
                          "FuncType"
                          )

    #Add information of the slope of DFI
    Slope <- c()
    for(ii in 1:dim(param.2)[1]){

      if(param.2[ii,]$c < 0){
        Slope.1 <- -1
      } else {
        Slope.1 <- 1
      }

      Slope <- c(Slope, Slope.1)
    }

    param.2$Slope <- Slope

    #-------------------------------------------------------------------------------
    # Give the ID back to the animal
    #-------------------------------------------------------------------------------
    ANIMAL_ID = rep( ID, dim(Data.new)[1])
    Data.new = cbind(ANIMAL_ID , Data.new)

    Data.remain = Data.new[,c(1:4)]

} else{
 }
```

c. If a quadratic-linear function is chosen

```
if(FuncType == "QLM"){

 #Remove uneccesary data
  rm(list=ls()[! ls() %in%
                 c("ID", "Data", "FuncType", "x", "Y", "Z",
                   "Data.xy", "X0.0", "Xlast", "par_init", "st1", "TTC.param")])

  # Load needed functions
  source("Functions.R")

 #-------------------------------------------
 # Create empty lists to store data after loop
 #-------------------------------------------

 par = list()
 AC.res = list()
 AC.pvalue = c()
 data2 = list()
 data3 = list()
 param <- data.frame(rbind(par_init))
 par.abcd = data.frame(rbind(abcd.2(as.vector(par_init))))
```

```
param.2 <- data.frame(X0=double(),
                      Xs=double(),
                      DFIs=double(),
                      CFIs=double(),
                      a=double(),
                      b=double(),
                      c=double(),
                      stringsAsFactors=FALSE)
j = 2
AC_pvalue = 0
AC.pvalue[1] = AC_pvalue
datapointsleft = as.numeric(dim(Data)[1])
dpl = datapointsleft #vector of all dataponitsleft at each step

#-------------------------------------------------------------------------------
# Start the filtration procedure by using Non Linear Regression
#-------------------------------------------------------------------------------

while ((AC_pvalue<=0.05) && datapointsleft >= 20){
 weight = 1/Y^2 #weighing data
 st2 = nls2(Y ~ nls.func.2(X0, Xs, DFIs, CFIs),
            Data.xy,
            start = st1,
            weights = weight,
            trace = F,
            algorithm = "brute-force")
 par_init <- coef(st2)

 nls.CFI <- nlsLM(Y ~ nls.func.2(X0, Xs, DFIs, CFIs),
                  Data.xy,
                  control = list(tol = 1e-2, printEval = TRUE, maxiter = 1024),
                  start = list(X0 = par_init[1], Xs = par_init[2],
                               DFIs = par_init[3], CFIs = par_init[4]),
                  weights = weight,
                  algorithm = "port",
                  lower = c(-100000,X0.0+1, -100000, -100000),
                  upper = c(100000, Xlast-1, 100000, 100000),
                  trace = F)

 #--------RESULTS analysis GOODNESS of fit
 #estimate params
 par[[j]] = coef(nls.CFI)
 par.abcd[j,] = abcd.2(as.vector(coef(nls.CFI) )) #calculation of a, b and c
 param[j,] = par[[j]]
 param.2[j-1,] = cbind(param[j,],par.abcd[j,])

 #residuals
 res1 = nlsResiduals(nls.CFI)
 res2= nlsResiduals(nls.CFI)$resi1
 res = res2[,2]
 AC.res = test.nlsResiduals(res1)
 AC.pvalue[j] =  AC.res$p.value
```

```r
#---------Check for negative residuals----------

#Add filtration step order to data
Step <- rep(j - 1, length(x))
#create a new dataset with predicted CFI included
Data.new <- data.frame(cbind(x, Z, Y, pred.func.2(par[[j]],x)[[1]], res, Step))
names(Data.new) <- c("Age",
                     "Observed_DFI",
                     "Observed_CFI",
                     "Predicted_CFI",
                     "Residual",
                     "Step")

#remove negative res
Data.pos <- Data.new[!Data.new$Residual<0,]

#restart
datapointsleft = as.numeric(dim(Data.pos)[1])
par_init = par[[j]]
AC_pvalue = AC.pvalue[j]
j = j+1
x <- Data.pos$Age
Y <- Data.pos$Observed_CFI
Z <- Data.pos$Observed_DFI
Data.xy <- as.data.frame(cbind(x,Y))
dpl = c(dpl, datapointsleft)

#Create again the grid
X0.0 <- x[1]
Xlast <- x[length(x)]
#Xs
if(par_init[2] -15 <= X0.0){
   Xs.1 = round(seq(X0.0 + 5, Xlast - 5, len = 30), digits = 0)
} else if(par_init[2] + 5 >= Xlast){
  Xs.1 = round(seq(par_init[2]-10, par_init[2]-1, len = 6), digits = 0)
} else{
  Xs.1 = round(seq(par_init[2]-5, par_init[2] + 5, len = 6), digits = 0)
}
#
X0.1 = rep(X0.0, length(Xs.1))
DFIs.1 <- c()
CFIs.1 <- c()
for(A in 1:length(Xs.1)){
   DFIs2 = Data[Data$Age.plot == Xs.1[A],]$DFI.plot
   CFIs2 = Data[Data$Age.plot == Xs.1[A],]$CFI.plot
   DFIs.1 <- c(DFIs.1, DFIs2)
   CFIs.1 <- c(CFIs.1, CFIs2)
   }
st1 = data.frame(cbind(X0.1,Xs.1, DFIs.1, CFIs.1))

if(X0.0 <= par_init[2] && Xlast >=par_init[2]){
   st1 <- rbind(st1, par_init)
 }
```

```
  names(st1) <- c("XO","Xs", "DFIs","CFIs")
  }

  #Create a table to store estimated parameters
  ANIMAL_ID = rep( ID, dim(param.2)[1])
  param.2 = cbind(ANIMAL_ID,
                  param.2,
                  XLAST = rep( Data$Age.plot[length(Data$Age.plot)], dim(param.2)[1]),
                  AC.pvalue[2:length(AC.pvalue)],
                  dpl[1:length(dpl)-1],
                  rep(FuncType, dim(param.2)[1])
                  )
  colnames(param.2)= c("ANIMAL_ID",
                       "XO",
                       "Xs",
                       "DFIs",
                       "CFIs",
                       "a",
                       "b",
                       "c",
                       "Xlast",
                       "P.runs.test",
                       "DPL",
                       "FuncType"
                       )

  #Add information of the slope of DFI
  Slope <- c()
  for(ii in 1:dim(param.2)[1]){

  if(param.2[ii,]$c < 0){
      Slope.1 <- -1
  } else {
      Slope.1 <- 1
  }

  Slope <- c(Slope, Slope.1)
  }

  param.2$Slope <- Slope

  #-------------------------------------------------------------------------------
  # Give the ID back to the animal
  #-------------------------------------------------------------------------------
  ANIMAL_ID = rep( ID, dim(Data.new)[1])
  Data.new = cbind(ANIMAL_ID , Data.new)

  Data.remain = Data.new[,c(1:4)]
} else{

}
```

Results of the filtration process

Note:

- Each row corresponds to each filtration step (to eliminate data with negative residuals).
- **Xlast**: The last day of age.
- **P.runs.test**: The P-value of auto-correlation test (runs test).
- **DPL**: The data points left in each step.
- **Slope**: slope of DFI function (possitive when it = 1 and negative when = -1)

```
TTC.param <- param.2[dim(param.2)[1],]
TTC.param
```

```
##   ANIMAL_ID       X0       Xs    DFIs     CFIs         a        b
## 4         1 67.29436 161.7723 2.85089 239.1013 -133.4189 1.754597
##             c Xlast  P.runs.test DPL FuncType Slope
## 4 0.003388383   195 5.127604e-05  37      QLM     1
```

After the estimation, if the quadratic-linear function was chosen to estimate the target CFI, it may be replaced by:

1. **A quadratic function** if estimated Xs is still too close to either the first or the last observation
2. **A linear function** if the slope of DFI curve is still negative

A quadratic function is replaced the quadratic-linear function

A linear function is replaced the quadratic-linear function

Also, after the estimation, if a Quadratic function is chosen to represent the target CFI and the slope of DFI curve is negative, a Linear function is used to represent the target CFI

```
TTC.param <- param.2[dim(param.2)[1],]

if(TTC.param$FuncType == "QDR" && TTC.param$Slope == -1){
     #-------------------------------------------------------------------------------
     # A linear function is replaced the quadratic-linear function
     #-------------------------------------------------------------------------------
     # Remove uneccesary data
     rm(list=ls()[! ls() %in% c("DON_NA.0", "DON_NA", "No.NA.Data", "ID","i",
                                "idc","Data", "TTC.param", "ITC.param.pos2",
                                "ITC.param.pos1", "ITC.param.neg", "Data.remain")])

     # Load needed functions
     source("Functions.R")

     ####### Create data frame of x (Age) and y (CFI) ########
     x <- as.numeric(Data$Age.plot)
     Y <- as.numeric(Data$CFI.plot)
     Z <- as.numeric(Data$DFI.plot)
     Data.xy <- as.data.frame(cbind(x,Y))

     xlast <- x[length(x)] # last day of observation
     # Initial parameteres for parameter estimation
     x0.0 = x[1]
```

```
ylast.0 = Y[length(x)]

# Vector contains 2 initial parameters
par_init <- c(x0.0, ylast.0)

#-----------------------------------------
# Create empty lists to store data after loop
#-----------------------------------------

par = list()
AC.res = list()
AC.pvalue = c()
data2 = list()
param <- data.frame(rbind(par_init))
par.abcd = data.frame(rbind(abcd.0(as.vector(par_init))))
param.2 <- data.frame(X0=double(),
                      Ylast=double(),
                      a=double(),
                      b=double(),
                      stringsAsFactors=FALSE)
j = 2
AC_pvalue = 0
AC.pvalue[1] = AC_pvalue
datapointsleft = as.numeric(dim(Data)[1])
dpl = datapointsleft #vector of all dataponitsleft at each step

#-------------------------------------------------------------------------------
# Start the filtration procedure by using Non Linear Regression
#-------------------------------------------------------------------------------
#
while (AC_pvalue<=0.05 && datapointsleft >= 20){
  weight = 1/Y^2 #weighing data
  nls.CFI <- nlsLM(Y ~ nls.func.0(X0, ylast),
                   Data.xy,
                   control = list(tol = 1e-2, printEval = TRUE, maxiter = 50),
                   start = list(X0 = par_init[1],
                                ylast = par_init[2]),
                   trace = F,
                   weights = weight)


  #--------RESULTS analysis GOODNESS of fit
  # estimate params
  par[[j]] = coef(nls.CFI)
  par.abcd[j,] = abcd.0(as.vector(coef(nls.CFI) )) #calculation of a, b, c and d
  param[j,] = par[[j]]
  param.2[j-1,] = cbind(param[j,],par.abcd[j,])

  # residuals
  res1 = nlsResiduals(nls.CFI) #residuals
  res2= nlsResiduals(nls.CFI)$resi1
  res = res2[,2]
  AC.res = test.nlsResiduals(res1)
```

```
  AC.pvalue[j] =  AC.res$p.value

  #---------Check for negative residuals----------

  # Add filtration step order to data
  Step <- rep(j - 1, length(x))
  # Create a new dataset with predicted CFI included
  Data.new <- data.frame(cbind(x, Z, Y, pred.func.0(par[[j]],x)[[1]], res, Step))
  names(Data.new) <- c("Age", "Observed_DFI","Observed_CFI",
                       "Predicted_CFI", "Residual", "Step")

  # Remove negative res
  Data.pos <- Data.new[!Data.new$Residual<0,]

  # Restart
  datapointsleft = as.numeric(dim(Data.pos)[1])
  par_init = par[[j]]
  AC_pvalue = AC.pvalue[j]
  j = j+1
  x <- Data.pos$Age
  Y <- Data.pos$Observed_CFI
  Z <- Data.pos$Observed_DFI
  Data.xy <- as.data.frame(cbind(x,Y))
  dpl = c(dpl, datapointsleft); dpl
}

ANIMAL_ID = rep( i, dim(param.2)[1])
XLAST =  rep( xlast, dim(param.2)[1])
FuncType <- "LM"
param.2 = cbind(ANIMAL_ID,
                param.2,
                XLAST,
                AC.pvalue[2:length(AC.pvalue)],
                dpl[1:length(dpl)-1],
                rep(FuncType, dim(param.2)[1])
)
colnames(param.2)= c("ANIMAL_ID",
                     "X0",
                     "Ylast",
                     "a",
                     "b",
                     "Xlast",
                     "P.runs.test",
                     "DPL",
                     "FuncType"
)

# Add one column about the slope of DFI to the function
Slope <- rep(0, dim(param.2)[1])

param.2$Slope <- Slope

#-------------------------------------------------------------------------------
```

```
    # Give Animal ID for each animal and save them to dataframes in the loop
    #-------------------------------------------------------------------------------
    ANIMAL_ID = rep( i, dim(Data.new)[1])
    Data.new = cbind(ANIMAL_ID , Data.new)

    Data.remain1 = Data.new[,c(1:4)]
  } else{
  }
```

```
## NULL
```

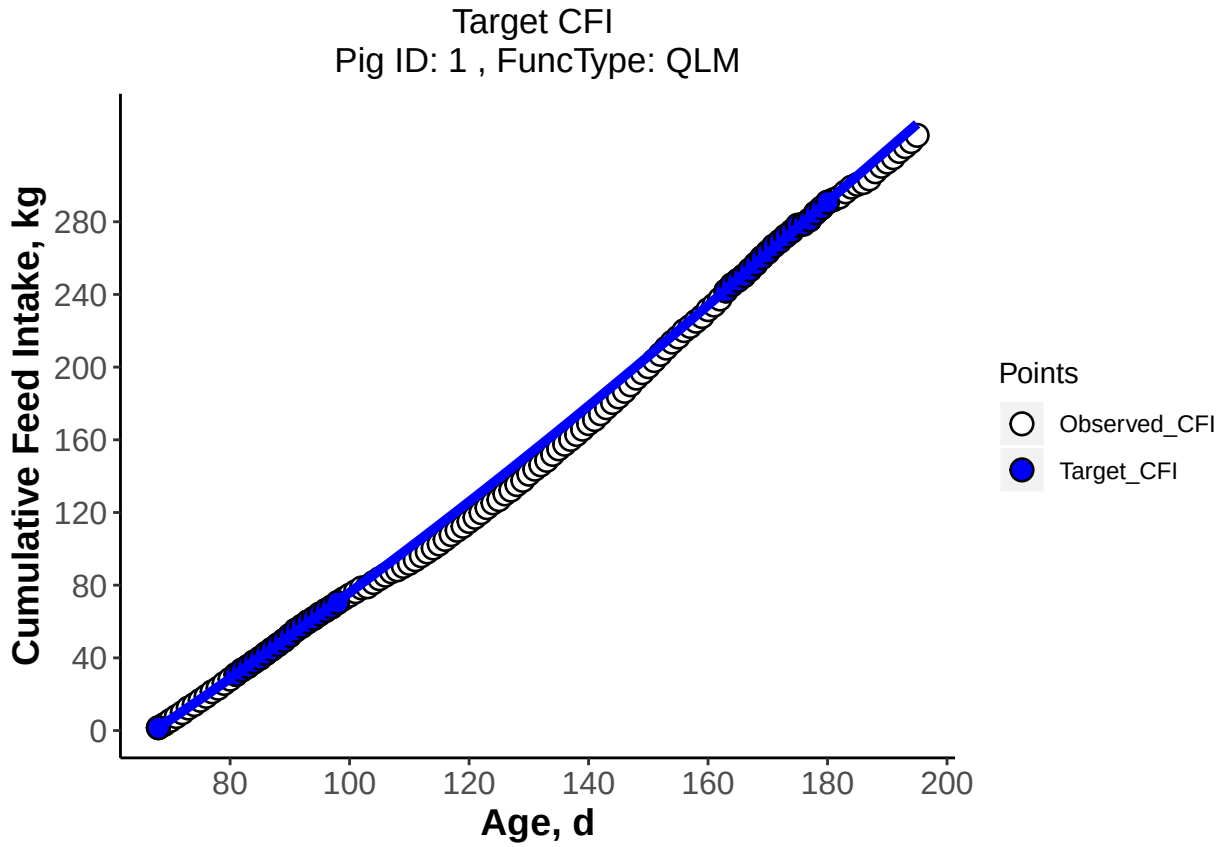
```
TTC.param <- param.2[dim(param.2)[1],]; TTC.param
```

```
##   ANIMAL_ID       X0       Xs    DFIs     CFIs         a        b
## 4         1 67.29436 161.7723 2.85089 239.1013 -133.4189 1.754597
##             c Xlast  P.runs.test DPL FuncType Slope
## 4 0.003388383   195 5.127604e-05  37      QLM     1
```

Plot the target CFI:

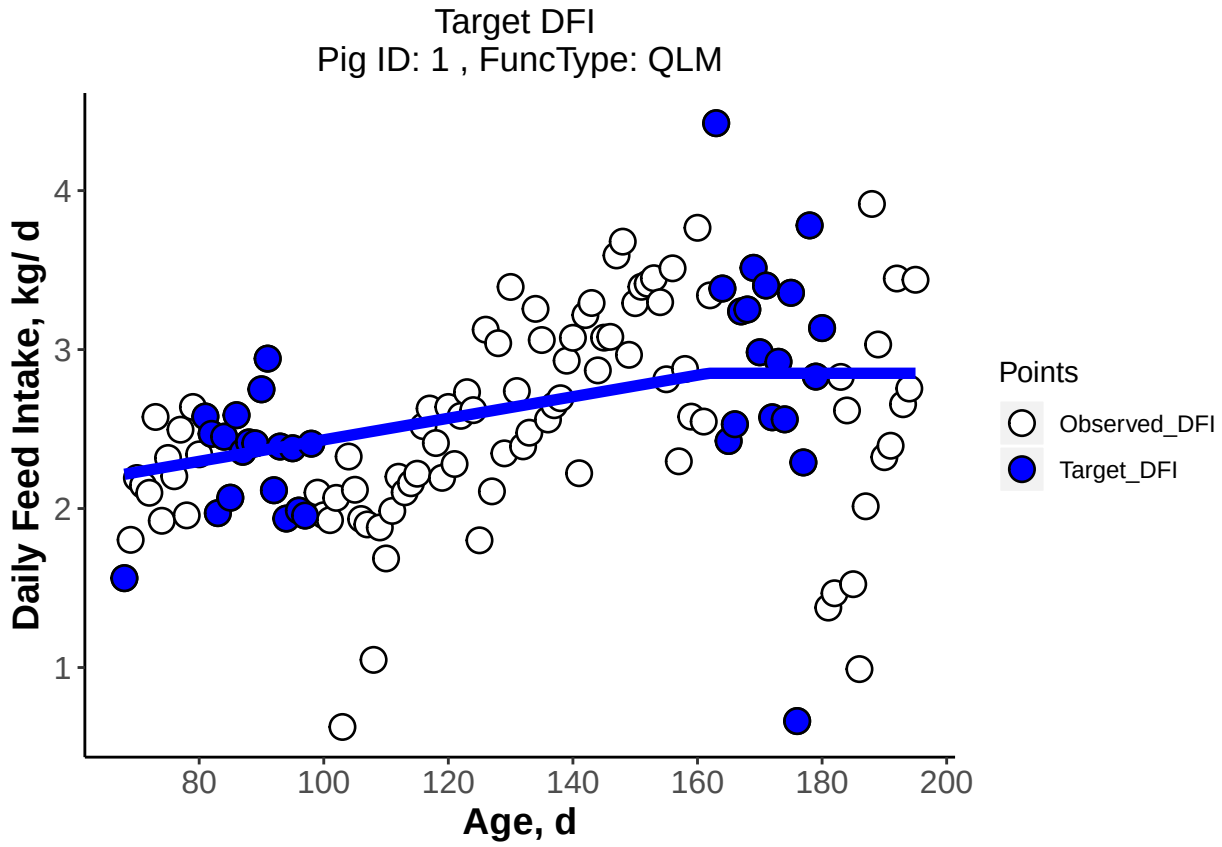
```
# Load needed functions
source("Functions.R")

#Extract Age, DFI and CFI
  Age = Data$Age.plot
  CFI  = Data$CFI.plot
  DFI = Data$DFI.plot

#Choose the right function for each animal

    FuncType <- TTC.param$FuncType
    Slope <- TTC.param$Slope

   if(FuncType == "LM"){
     param.i <- as.numeric(TTC.param[,c(4:5)])
     ITC <- pred.abcd.0(param.i, Age)[[1]]
     ITD <- rep(param.i[2], length(Age))

   } else if(FuncType == "QDR"){
     param.i <- as.numeric(TTC.param[,c(5:7)])
     ITC <- pred.abcd.1(param.i, Age)[[1]]
     ITD <- pred.abcd.1(param.i, Age)[[2]]

   } else{
     param.i <- as.numeric(TTC.param[,c(6:8)])
     Xs <- TTC.param$Xs
     ITC <- pred.abcd.2(param.i, Age)[[1]]
     ITD <- pred.abcd.2(param.i, Age)[[2]]

   }

#Plot
```

```r
#CFI data preparation
 cf1 <- Data %>% select(Age.plot, CFI.plot)
 cf1 <- cf1 %>% mutate(Points = rep("Observed_CFI", length(Age)))
 cf2 <- data.frame(cbind(Data$Age.plot, ITC)); names(cf2) <- c("Age.plot", "CFI.plot")
 cf2 <- cf2 %>% mutate(Points = rep("Target_CFI", length(Age)))
 cf <- rbind(cf1, cf2)

 AA1 <- Data.remain %>% select(Age, Observed_CFI) %>%
        mutate(Points = rep("Target_CFI", length(Age)))
 names(AA1) <- c("Age.plot", "CFI.plot", "Points")
 AA <- rbind(cf1, AA1)

 #To export figure in the same working directory as this R-script:
 # activate the row below and function dev.off() in the end
 # tiff(file = paste(Data$ANIMAL_ID,".","Target_CFI",".png",sep=""),
 # width = 5000, height = 3000, units = "px", res=600)
 cols <- c("Target_CFI" = "blue", "Observed_CFI" = "white")
 ggplot(data = AA, aes(x = Age.plot, y = CFI.plot)) +
 geom_point(aes(fill = Points), color = "black", shape = 21, size = 3.5, stroke = 0.7) +
 geom_line(data = cf2, aes(x = Age.plot, y = CFI.plot), color = "blue", size = 1.7) +
 scale_fill_manual(values = cols) +
 xlab("Age, d") +
 ylab("Cumulative Feed Intake, kg") +
 scale_y_continuous(breaks=seq(0, 300, 40)) +
 scale_x_continuous(breaks=seq(60, 240, 20)) +
 ggtitle(paste("Target CFI","\nPig ID:", ID, ", FuncType:", FuncType)) +
 theme(plot.title = element_text(hjust = .5)) +
 theme(panel.grid.major = element_blank(),
       panel.grid.minor = element_blank(),
       panel.background = element_blank(),
       axis.line = element_line(colour = "black")) +
theme(axis.text=element_text(size=12),
       axis.title=element_text(size=14,face="bold"))
```




```
# dev.off()

#DFI data preparation
 df1 <- Data %>% select(Age.plot, DFI.plot)
 df1 <- df1 %>% mutate(Points = rep("Observed_DFI", length(Age)))
 df2 <- data.frame(cbind(Data$Age.plot, ITD)); names(df2) <- c("Age.plot", "DFI.plot")
 df2 <- df2 %>% mutate(Points = rep("Target_DFI", length(Age)))
 df <- rbind(df1, df2)
 BB1 <- Data.remain %>% select(Age, Observed_DFI) %>%
        mutate(Points = rep("Target_DFI", length(Age)))
 names(BB1) <- c("Age.plot", "DFI.plot", "Points")
 BB <- rbind(df1, BB1)

 # To export figure in the same working directory as this R-script:
 # activate the row below and function dev.off() in the end
 # tiff(file = paste(Data$ANIMAL_ID,".","Target_DFI",".png",sep=""),
 # width = 5000, height = 3000, units = "px", res=600)
 cols <- c("Target_DFI" = "blue", "Observed_DFI" = "white")
 ggplot(data = BB, aes(x = Age.plot, y = DFI.plot)) +
 geom_point(aes(fill = Points), color = "black", shape = 21, size = 4, stroke = 0.7) +
 geom_line(data = df2, aes(x = Age.plot, y = DFI.plot), color = "blue", size = 2) +
 scale_fill_manual(values = cols) +
 xlab("Age, d") +
 ylab("Daily Feed Intake, kg/ d") +
 scale_y_continuous(breaks=seq(0, 8, 1)) +
 scale_x_continuous(breaks=seq(60, 240, 20)) +
```

```
ggtitle(paste("Target DFI","\nPig ID:", ID, ", FuncType:", FuncType)) +
theme(plot.title = element_text(hjust = .5)) +
theme(panel.grid.major = element_blank(),
panel.grid.minor = element_blank(),
panel.background = element_blank(),
axis.line = element_line(colour = "black")) +
theme(axis.text=element_text(size=12),
axis.title=element_text(size=14,face="bold"))
```




```
# dev.off()
```

Step 3: Detection of perturbations

Calculate the difference between observed CFI and target CFI The differences between CFI and Target CFI are presented as percentage (%)

```
# Calculate the differences between observed and target CFI
 res <- CFI - ITC
#Represent differences in percentage
 percent <- res/ITC*100
```

3.1. Set conditions for B-spline functions

Factors which determine the smoothness of B-slpine function are chosen by visualization:

- B-spline is a set of polynomial functions which joined end-to-end with each other at interval boundaries called **knots**.
- The **order of these polynomials** (one larger than the degree of polynomials) was given the value of 6 (therefore the degree of polynomial is 5).
- The **curvature of the function** (also is the second derivative of the function) which determines the shape of the curve was given the value of 4.
- **Lambda** - smoothing parameter (derivative penalty) which controls the smoothness of the curve by penalizing the curvature was given the value of 0.01.

```
#================================================================
# Fit a B-spline to the diferences between CFI and Target CFI (%)
#================================================================

 #number of basis functions = order + number of interior knots
 #Roughness penalty
   lambda = 1e-2
   pen_what = 4 # ensure the smoothness of 2nd derivative
   norderT = 6 # because we need the 4th derivative to be continue
   nby = 1 # which knots? = 1: all points, =2: every 2 points...
   rangeval=c(min(Age),max(Age))

 #start the process
 # create the time step based on nby
   idx = c(seq(1,length(Age), by=nby), length(Age))
   idx = unique(idx)  # in case the last value is taken twice

 # Create B-Spline basis of order order T with n knots
 #help(create.bspline.basis)
   basisobj <- create.bspline.basis(rangeval , norder=norderT , breaks=Age[idx])
   fdPar <- fdPar(basisobj, pen_what, lambda)
```

Choose a value of time interval and generate a series of values from the first to the last Age with the chosen time interval

E.g. if time interval is equal to 0.1, the series are 80.1, 80.2,...

```
 #evaluation days for CFI and spline functions :
 #time interval can be changed by = 0.1, 0.01,...
    dexima.i <- .1 # interval number
 # divide growing period by a smaller time step
    eval_day <- seq(min(Age), max(Age),by=dexima.i)
```

Fit B-spline function for differences between observed and target CFI

```
   difCFI.fd <- smooth.basis(Age, percent, fdPar)
   Smooth.difCFI  <- difCFI.fd$fd

#-------------------------------------------------------------------------
# Results of SPLINE
```

```
#-------------------------------------------------------------------------------

# Evaluation of B-spline for the difference between observed and target CFI
    val.CFI <- eval.fd(eval_day, Smooth.difCFI ) # Spline function
    vel.CFI <- eval.fd(eval_day, Smooth.difCFI, 1) # 1st derivative of Spline function
    acc.CFI <- eval.fd(eval_day, Smooth.difCFI, 2) # 2nd derivative of Spline function

# Call new names for val.CFI and vel.CFI
    dif.CFI <- val.CFI # Spline function of difference between observed and target CFI
    dif.DCFI <- vel.CFI # 1st derivative of Spline function (slope)

#-------------------------------------------------------------------------------
# Make data frame called "dif" only contains time, values of spline function,
# of its slope and difference between CFI and its target CFI
#-------------------------------------------------------------------------------
   dif = as.data.frame(cbind(eval_day, dif.CFI, dif.DCFI))
   names(dif) <- c("eval_day","dif.CFI", "dif.DCFI")
```

3.2. Detect deviations between observed CFI and target CFI

This step uses results obtained from B-spline functions to calculate:

- Start of each perturbation
- Time where CFI rejoins the target CFI
- Magnitude of each perturbation (the day and percentage where CFI deviates the most from target CFI)

```
#-------------------------------------------------------------------------------
# Detect changes in the sign of 1st derivative
#-------------------------------------------------------------------------------

#Criteria for a normal range from which everything below is considered as deviations
 crit1 = 0
#so whenever observed CFI decreases from target CFI we consider as deviations

#Days belong to normal range
 no.pert.age.val  = eval_day[which(dif.CFI >= crit1)]
#Determine when data is in normal range
 dif$CFIzero = rep(1,length(eval_day))
 dif$CFIzero[eval_day %in%no.pert.age.val] = 0

#-------------------------------------------------------------------------------
# Sign of 1st derivative of B-Spline function
#-------------------------------------------------------------------------------

#create a vector for sign of 1st derivative called "sign.vel"
 dif$sign.vel=rep(0,length(eval_day))

#When sign of 1st derivative is possitive (sign.vel = 1)
 dif$sign.vel[which(dif.DCFI >0)] = 1

#When sign of 1st derivative is negative (sign.vel = -1)
```

```
  dif$sign.vel[which(dif.DCFI <0)] = -1

#-------------------------------------------------------------------------------
#Calculate the variation in signs of the slope between the day before and the day after.
#EX: if the slope changes from negative to possitive from the day before to the day after.
#To do that: Create one data frame from "dif" in which the first row is removed
#And calculate the differences in slopes with "dif"
#-------------------------------------------------------------------------------

#Duplicate the last row so the new data frame will have equal length with "dif"
  lastrow <- cbind(Age[length(Age)] + dexima.i, dif[dim(dif)[1],][,-1])
  names(lastrow) <- c("eval_day","dif.CFI", "dif.DCFI","CFIzero", "sign.vel" )

#Create the new data frame, remove first row and duplicate the last row
  difp1 = rbind(dif[-1,], lastrow)

#-------------------------------------------------------------------------------
#Determine the days in which the deviation of CFI from target CFI are maximum and minimum
#-------------------------------------------------------------------------------

 difp1$maxmin = dif$sign.vel - difp1$sign.vel
#When the 1st derivative changes from negative to positive:
#sign = -1 - 1 = -2 -> maxmin = -2 is min
#When the 1st derivative changes from positive to negative:
#sign = 1 - (-1) = 2 -> therefore maxmin = +2 is max

#-------------------------------------------------------------------------------
# Determine the days in which CFI curves start
# to deviate from and come back to normal range
#-------------------------------------------------------------------------------

 difp1$end.p =  dif$CFIzero - difp1$CFIzero
#CFIzero = 0 when CFI is equal TTC, otherwise = 1
#Value of end.p is equal to 1: come back inside the normal range
#Value of end.p is equal to -1: go out of the normal range
```

3.3. Examining data from the first week

We do not take into consideration data in the first week of pig.

However, deviations that started during the first week and for which the selection criteria of duration and magnitude continue to hold during the second week are considered to be the result of a perturbation.

```
#Criteria to remove the first period of mixing group effect on pigs
 crit2 = 7 #we remove 1st week

#Find the end of the first perturbation
  normal.day <- difp1 %>% filter(end.p ==1); normal.day <- normal.day$eval_day
#Find the begining of the first perturbation
  pertub.day <- difp1 %>% filter(end.p == -1); pertub.day <- pertub.day$eval_day

#After remove the first growing period (7 days)
#if the 1st deviation has its magnitude larger than 5%
```

```
#We consider it as a perturbation, thus, we
#include all data from first period to the data again

 if(normal.day[1] - difp1$eval_day[1] < crit2 &
    normal.day[1] < pertub.day[1] &
    pertub.day[1] < difp1$eval_day[1] + crit2){

    crit2 = 0
 } else if(normal.day[1] - difp1$eval_day[1] < crit2 &
           normal.day[1] < pertub.day[1] &
           pertub.day[1] > difp1$eval_day[1] + crit2 ){

    crit2 = 7
 } else if(normal.day[1] - difp1$eval_day[1] < crit2 &
           normal.day[1] > pertub.day[1] &
           pertub.day[2] > difp1$eval_day[1] + crit2){

   crit2 = 7
 } else if(normal.day[1] - difp1$eval_day[1] < crit2 &
           normal.day[1] > pertub.day[1] &
           pertub.day[2] < difp1$eval_day[1] + crit2){
   crit2 = 0
 } else{
   normal.day1 <- ifelse(normal.day[1] > difp1$eval_day[1] + crit2,
                         normal.day[1],
                         normal.day[2])

#Test if after removing the first week,
#magnitude of deviation is still larger than 5%;
#we keep 1st week, otherwise we remove 1st week
 test.data <- filter(difp1,
                     eval_day %in% seq(difp1$eval_day[1] + crit2,
                                       normal.day1,
                                       by = 0.1))
 dif.CFI1 <- test.data %>%
               group_by(dif.CFI) %>%
                summarise(count = n()) %>%
                 arrange(desc(count)) %>%
                  top_n(2)

 as.numeric(dif.CFI1$dif.CFI)
 crit2 = ifelse(dif.CFI1$dif.CFI[1] <= -5 &&
                  (normal.day1 - (difp1$eval_day[1] + crit2)) >= 5,
                0, 7)
}
```

```
## Selecting by count
```

```
#Define the days in which observed CFI is deviated from the target CFI
#deviated ages (which have CFI values < normal range)
 dev.age = difp1[difp1$dif.CFI < crit1 & difp1$eval_day > eval_day[1] + crit2,]$eval_day
#data frame contains all information of the perturbed days
 dev.df <- difp1[difp1$eval_day %in% dev.age,]
```

```
#Check if crit2 = 0, we have to include the first week to perturbation
 crit2
```

```
## [1] 7
```

3.3. Determine the duration of each deviation

```
#-------------------------------------------------------------------------------
# Extract min , max and start and end points of the deviations
#-------------------------------------------------------------------------------

  #days when deviations have maximum values
    max = dev.df$eval_day[which(dev.df$maxmin == -2)]

  # values of max
    dif.max = dif$dif.CFI[dif$eval_day%in%max]

  #days when deviations have minimum values
    min = dev.df$eval_day[which(dev.df$maxmin == 2)]

  # values of min
    dif.min = dif$dif.CFI[dif$eval_day%in%min]

  # start days of deviations
    start.raw <- dev.df$eval_day[which(dev.df$end.p == -1)]
    if(sum(start.raw) == 0){
      start = dev.df$eval_day[1]
    } else{
      if(start.raw[1] == dev.age[1]){
      start = start.raw
      } else{
      start = c(dev.age[1], start.raw)
      }
    }
  # values of start
    dif.start = dif$dif.CFI[dif$eval_day%in%start]

  # end days of deviations
    end = difp1[difp1$end.p == 1 & difp1$eval_day > eval_day[1] + crit2,]$eval_day
  # values of end
    dif.end = dif$dif.CFI[dif$eval_day%in%end]

  #Create an empty table.i which has the length equal to the number of "start" and "end"
  #per = number of deviation
    table.i <- data.frame(per=1:max(length(end),
                                    length(dif.end),
                                    length(start),
                                    length(dif.start)))

  #Fill in the vectors of days and values of start and end points to table
    table.i$start <- c(start, rep(NA, nrow(table.i)-length(start)))
```

```
    table.i$dif.start <- c(dif.start, rep(NA, nrow(table.i)-length(dif.start)))
    table.i$end <- c(end, rep(NA, nrow(table.i)-length(end)))
    table.i$dif.end <- c(dif.end, rep(NA, nrow(table.i)-length(dif.end)))

#-------------------------------------------------------------------------------
# Modify the table for statrt and the end days if there is NA in table
#-------------------------------------------------------------------------------

#Create a table.dev only contains start and end points (without values)
 dev.table <- cbind()

#If CFI value is lower than target CFI in the first day of dataset, we
#involve the first day of age in the data as the start day of the deviation

 for( ii in 1: length(table.i$per)){
#in case pig starts perturbed before scale of dataset
#and does not recover to normal range; i.e. i = 1
#
 if(length(table.i$per) == 1 & sum(is.na(table.i$start)) == 0 &
    sum(is.na(table.i$end)) > 0){
    dev.table <- cbind(table.i$start,Age[length(Age)])
    } else if(start[1] > end[1] & sum(is.na(table.i$start)) == 0) {
#
#set 1st day of age as start day of 1st deviation
#and last day of age as end day of last deviation
  dev.table <- cbind(c(Age[1],
                       table.i$start),
                     c(table.i$end, Age[length(Age)]))
#in case number of end days is larger than no of start days
#eg. i = 8, lambda = 0.01
  } else if(start[1] > end[1] & sum(is.na(table.i$start)) > 0){
#Remove last day of start
#and add 1st day of age as start day of 1st deviation
  dev.table <- cbind(c(Age[1],
                       table.i$start[-length(table.i$start)]),
                     c(table.i$end))

#in case last end day is NA because pig does not recover completely
 } else if (start[1] < end[1] & sum(is.na(table.i$end)) > 0){ # i = 45
 #
#change last end day by last day of age
 dev.table <- cbind(table.i$start,
                    c(table.i$end[-length(table.i$end)],
                      Age[length(Age)]))

#Normal case
#
 } else{
 dev.table <- cbind(table.i$start, table.i$end) #
 }
}

 dev.table <- as.data.frame(dev.table)
```

```
 colnames(dev.table) <- c("Start", "End")

# Add values of start and end days to table
 value.start <- dif$dif.CFI[dif$eval_day %in% dev.table$Start]
 value.end <- dif$dif.CFI[dif$eval_day %in% dev.table$End]
 dev.table <- cbind(dev.table[1], value.start, dev.table[2], value.end)

#-------------------------------------------------------------------------------
# Calculate the duration of each deviation
#-------------------------------------------------------------------------------
 dev.table$Devia.days <- dev.table$End - dev.table$Start
 dev.table
```

```
##   Start value.start   End    value.end Devia.days
## 1  75.1 -3.75646023  89.0  0.041011907       13.9
## 2  95.5 -0.01346570 168.1  0.003384033       72.6
## 3 175.6 -0.01912935 195.0 -1.887161916       19.4
```

3.4. Consider the most important deviations as perturbations

```
#min number of days for a pert
    crit3 = 5
#create a table which includes all information of perturbations
    pertub.table <- data.frame()
    for(ii in 1:dim(dev.table)[1]){
      if(dev.table$Devia.days[ii] < crit3){
        next
      }
      pertub.table <- rbind(pertub.table, dev.table[ii,])
    }

#-------------------------------------------------------------------------------
# Give a name to each perturbation
#-------------------------------------------------------------------------------

    Per <- c()
    Per1<- factor()
    for(ii in 1:dim(pertub.table)[1]){
      Per1 <- paste("P", ii, sep = "")
      Per <- c(Per, Per1)
    }

    pertub.table <- cbind(as.factor(Per), pertub.table)
    names(pertub.table) <- c("Per", "Start", "value.start",
                             "End", "value.end", "Devia.days")

#Check the magnitude of each deviation
#if it decreases less than 5% from the TTC, we do not consider it as a perturbation
  A <- data.frame()
  for(ii in 1:dim(pertub.table)[1]){
        A1 <- dif %>%
```

```
            filter(eval_day %in% as.character(seq(pertub.table$Start[ii],
                                                  pertub.table$End[ii],
                                                  by = dexima.i))) %>%
            select(eval_day, dif.CFI) %>%
            arrange(dif.CFI)
        A1 <- A1[1,]
        A <- rbind(A, A1)
    }
  names(A) <- c("Min.Day", "Min.perc")
  pertub.table <- cbind(pertub.table, A)

#Remove the deviations which have their duration less than 5 days from the table
  pertub.table <- pertub.table %>% filter(Min.perc <= -5)

#-------------------------------------------------------------------------------
# Provide each perturbation with a lable
#-------------------------------------------------------------------------------

 if(dim(pertub.table)[1] == 0){
    difp1$ppert <- NA
    dev.df$ppert <- NA
 } else{
   for(ii in 1:dim(pertub.table)[1]){
    pert.int = as.character(seq(pertub.table$Start[ii],
                                pertub.table$End[ii],
                                by=dexima.i))
difp1$ppert[as.character(difp1$eval_day) %in% pert.int] <- paste("P", ii, sep = "")
 }


# Include label for dataset only contains deviations "dev.df"
for(ii in 1:dim(pertub.table)[1]){
 pert.int = as.character(seq(pertub.table$Start[ii],
                             pertub.table$End[ii],
                             by=dexima.i))

 dev.df$ppert[as.character(dev.df$eval_day) %in% pert.int] <- paste("P", ii, sep = "")
 }
}
```

Detected perturbation:

```
 pertub.table
```

```
##   Per Start value.start   End   value.end Devia.days Min.Day  Min.perc
## 1  P2  95.5  -0.0134657 168.1 0.003384033       72.6   114.7 -9.274392
```

Note:

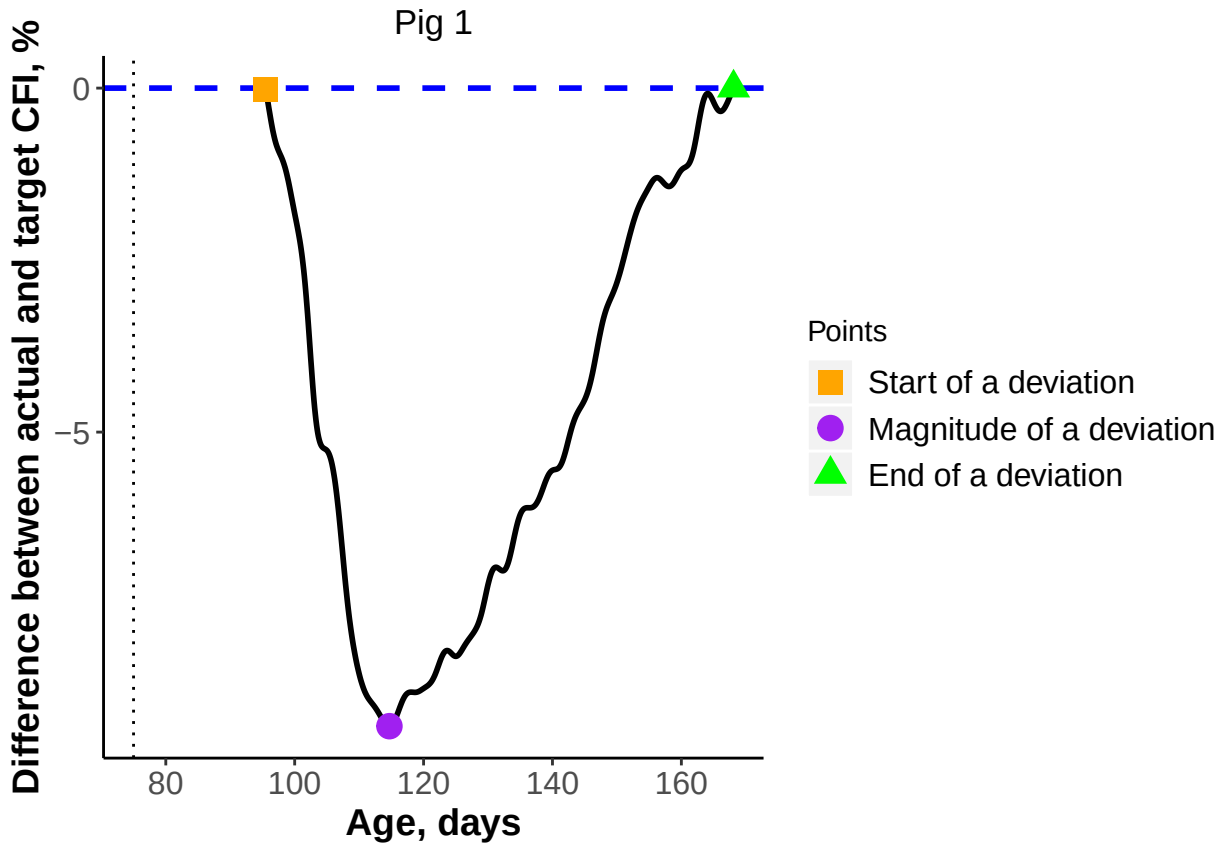
- **Start**: Age the perturbation started (days)
- **End**: Age the observed CFI rejoined the target CFI (days)
- **Devia.days**: Duration of perturbation (days)

- **Min.Day**: Age when perturbation ended (days)
- **Min.perc**: Magnitude of the perturbation (in %) at day `Min.Day`

Plot the difference between observed and target CFI

```
#Arrange Dataset
B <- dev.df[!is.na(dev.df$ppert),]
C <- difp1 %>% filter(eval_day %in% pertub.table$End)
BC <- rbind(B, C)
BC <- arrange(BC, eval_day)

#To plot discreted perturbations:
BC1 <- BC %>% filter(ppert == "P1")
BC2 <- BC %>% filter(ppert == "P2")
BC3 <- BC %>% filter(ppert == "P3")
BC4 <- BC %>% filter(ppert == "P4")

#Prepare data for points
#Start of each deviation
BC.start <- pertub.table[,c(2,3)]
names(BC.start) <- c("eval_day","dif.CFI")
BC.start <- BC.start %>% mutate(Points = rep("1Dev_Start", length(eval_day)))
#End of each deviation
BC.end <- pertub.table[,c(4,5)]
names(BC.end) <- c("eval_day","dif.CFI")
BC.end <- BC.end %>% mutate(Points = rep("3Dev_End", length(eval_day)))

#Magnitude of each perturbation
BC.mag <- pertub.table[,c(7,8)]
names(BC.mag) <- c("eval_day","dif.CFI")
BC.mag <- BC.mag %>% mutate(Points = rep("2Dev_Mag", length(eval_day)))

#All points together
BC.points <- rbind(BC.start, BC.end, BC.mag)

AA <- 1

# tiff(file = paste(Data$ANIMAL_ID,".","Detect_Per_col",".png",sep=""),
# width = 6000, height = 3500, units = "px", res=700)
cols <- c("1Dev_Start" = "orange", "2Dev_Mag" = "purple", "3Dev_End" = "green")
shapes <- c("1Dev_Start" = 15, "2Dev_Mag" = 19, "3Dev_End" = 17)
ggplot(data = BC, aes(x = eval_day, y = dif.CFI)) +
geom_vline(xintercept=eval_day[1] + 7, linetype="dotted", color = "black", size = 0.5)+
geom_hline(yintercept=0, linetype="dashed", color = "blue", size = AA)+
geom_line(data = BC1, aes(x = eval_day, y = dif.CFI), col = "black", size = AA) +
geom_line(data = BC2, aes(x = eval_day, y = dif.CFI), col = "black", size = AA) +
geom_line(data = BC3, aes(x = eval_day, y = dif.CFI), col = "black", size = AA) +
geom_line(data = BC4, aes(x = eval_day, y = dif.CFI), col = "black", size = AA) +
geom_point(data = BC.points, aes(color = Points, shape = Points), size = 4) +
scale_color_manual(values = cols, labels = c("Start of a deviation",
                                             "Magnitude of a deviation",
                                             "End of a deviation"))+
scale_shape_manual(values = shapes, labels = c("Start of a deviation",
                                               "Magnitude of a deviation",
```

```
                                                "End of a deviation")) +
xlab("Age, days") +
ylab("Difference between actual and target CFI, %") +
scale_y_continuous(breaks=seq(-20, 0, 5)) +
scale_x_continuous(breaks=seq(60, 240, 20)) +
ggtitle(paste("Pig", ID)) +
theme(plot.title = element_text(hjust = .5)) +
theme(panel.grid.major = element_blank(),
panel.grid.minor = element_blank(),
panel.background = element_blank(),
axis.line = element_line(colour = "black")) +
theme(axis.text=element_text(size=12),
axis.title=element_text(size=14,face="bold"))+
theme(legend.text=element_text(size=12))
```




```
# dev.off()
```

```
#=============================================================
#Save results for next steps
#=============================================================
 res.data <- as.data.frame(cbind(Age, res, percent))
 res.data <- cbind(rep(ID, length(res)), res.data)
 names(res.data) <- c("ANIMAL_ID", "Age", "res", "percent")
```

Step 4: Modelling the response of the pig to a perturbation

Our model contains 4 parameters to quantify the impact of a perturbation on the pig

- **tbeg1t**: the start age of perturbing factor (**t_start** in the article)
- **tstop1**: the end age of perturbing factor (**t_stop** in the article)
- **p1**: Percentage of DFI left under the period of perturbation (%). It is opposite to **k1** in the article: ** k1 = 1 - p1 **
- **max.compFI1**: the capacity of the animal to adapt to the perturbation through resistant and compensatory feed intake (**k2** in the article)

```
#Remove unneccesary data
 rm(list=ls()[! ls() %in% c("Data",
                            "dev.df",
                            "difp1",
                            "param.2",
                            "pertub.table",
                            "res.data")])

#Load needed functions
source("Functions.R")
options(digits=3)

#================================================================
# DATA PREPARATION
#================================================================

#Order number of Animal_ID
ID = unique(as.factor(Data$ANIMAL_ID))

#----------------------------
# Extract Age, DFI and CFI
#----------------------------
Age = Data$Age.plot
DFI.obs = Data$DFI.plot
CFI.obs  = Data$CFI.plot

#Difference between observed and target CFI (kg)
res <- res.data$res

#Information of TTC function
 TTC.param <- param.2[dim(param.2)[1],]
 FuncType <- TTC.param$FuncType
 Slope <- TTC.param$Slope

#Magnitude of the perturbation
 magnitude = res.data %>% filter(Age >= pertub.table$Start & Age <= pertub.table$End)
 magnitude = magnitude %>% filter(res == min(magnitude$res))
```

Extract suitable function for target CFI

```r
#-------------------------------------------------------------------------------
# Calulate target CFI using suitable function
#-------------------------------------------------------------------------------

  if(FuncType == "LM"){
    param.i <- as.numeric(TTC.param[,c(4:5)])
    ITC <- pred.abcd.0(param.i, Age)[[1]]
    ITD <- rep(param.i[2], length(Age))

  } else if(FuncType == "QDR"){
    param.i <- as.numeric(TTC.param[,c(5:7)])
    ITC <- pred.abcd.1(param.i, Age)[[1]]
    ITD <- pred.abcd.1(param.i, Age)[[2]]

  } else{
    param.i <- as.numeric(TTC.param[,c(6:8)])
    Xs <- TTC.param$Xs
    ITC <- pred.abcd.2(param.i, Age)[[1]]
    ITD <- pred.abcd.2(param.i, Age)[[2]]
  }
```

4.1. If the target CFI is defined by a linear function

```r
if(FuncType == "LM"){

##-----------------------------
## initial values of parameters
##-----------------------------
tbeg1       = pertub.table$Start
tstop1      = magnitude$Age
p1          = 0.3
#because we modified negative impact of perturbation as (1-p1)
#the smaller p1 is the more severe impact is
max.compFI1 = 2
a           = TTC.param$a
b           = TTC.param$b

  yinit <- c(CumFI = ITC[1]) #state
  times.ode <- seq(from = Age[1], to = Age[length(Age)], by = .1)

  # ==============  Estimation of parameters ================== ==================

  ##-----------------------------
  ## parameters
  ##-----------------------------

  par.init = c( p1, max.compFI1, tbeg1, tstop1)

  ##-----------------------------
  #  run the optimization
  ##-----------------------------
  Data.xy = Data
```

```
  times = Data.xy$Age.plot
  ODE.CFI.obj.0(par.init, Data.xy)

  #Estimate parameters by Optim and the best initial parameters
  optim.res = optim(par.init, ODE.CFI.obj.0, hessian = TRUE)

  P.optim = c(optim.res$par[1], optim.res$par[2], optim.res$par[3], optim.res$par[4])

  ODE.CFI.obj.0(P.optim, Data.xy)

  # Simulate the ode function
  yout   <- ode(yinit, times, ODE.CFI.optim.0, P.optim)

} else{}
```

4.2. If the target CFI is defined by a quadratic function

```
if(FuncType == "QDR"){

 ##-----------------------------
 ## initial values and times
 ##-----------------------------
  tbeg1       = pertub.table$Start
  tstop1      = magnitude$Age
  p1          = 0.3
  #because we modified negative impact of perturbation as (1-p1)
  #the smaller p1 is the more severe impact is
  max.compFI1 = 6
  a           = TTC.param$a
  b           = TTC.param$b
  c           = TTC.param$c

  yinit <- c(CumFI = ITC[1]) #state
  times.ode <- seq(from = Age[1], to = Age[length(Age)], by = .1)

 # ==============  Estimation of parameters ================== ==================

 ##-----------------------------
 ## parameters
 ##-----------------------------

  par.init = c( p1, max.compFI1, tbeg1, tstop1)

 ##-----------------------------
 #  run the optimization with NLS2
 ##-----------------------------
 Data.xy = Data
 times = Data.xy$Age.plot
 ODE.CFI.obj.1(par.init, Data.xy)

 #Estimate parameters by Optim and the best initial parameters
```

```
 optim.res = optim(par.init, ODE.CFI.obj.1, hessian = TRUE)

 P.optim = c(optim.res$par[1], optim.res$par[2], optim.res$par[3], optim.res$par[4])
 ODE.CFI.obj.1(P.optim, Data.xy)

 # Simulate the ode function
 yout   <- ode(yinit, times.ode, ODE.CFI.optim.1, P.optim)

} else{}
```

4.3. If the target CFI is defined by a quadratic-linear function

```
if(FuncType == "QLM"){

  ##-----------------------------
  ## initial values and times
  ##-----------------------------

  tbeg1       = pertub.table$Start
  tstop1      = magnitude$Age
  p1          = 0.3
  #because we modified negative impact of perturbation as (1-p1)
  # the smaller p1 is the more severe impact is
  max.compFI1 = 7
  a           = TTC.param$a
  b           = TTC.param$b
  c           = TTC.param$c
  Xs          = TTC.param$Xs

  yinit <- c(CumFI = ITC[1]) #state
  times.ode <- seq(from = Age[1], to = Age[length(Age)], by = .1)

# ==============  Estimation of parameters ================== ==================

  ##-----------------------------
  ## parameters
  ##-----------------------------
    par.init = c( p1, max.compFI1, tbeg1, tstop1)

  ##-----------------------------
  #  run the optimization with NLS2
  ##-----------------------------
  Data.xy = Data
  times = Data.xy$Age.plot
  ODE.CFI.obj.2(par.init, Data.xy)

  #Estimate parameters by Optim and the best initial parameters
  optim.res = optim(par.init, ODE.CFI.obj.2, hessian = TRUE)

  P.optim = c(optim.res$par[1], optim.res$par[2], optim.res$par[3], optim.res$par[4])
```

```
  ODE.CFI.obj.2(P.optim, Data.xy)

  # Simulate the ode function
  yout   <- ode(yinit, times.ode, ODE.CFI.optim.2, P.optim)

} else{}
```

Estimated paramters of the model

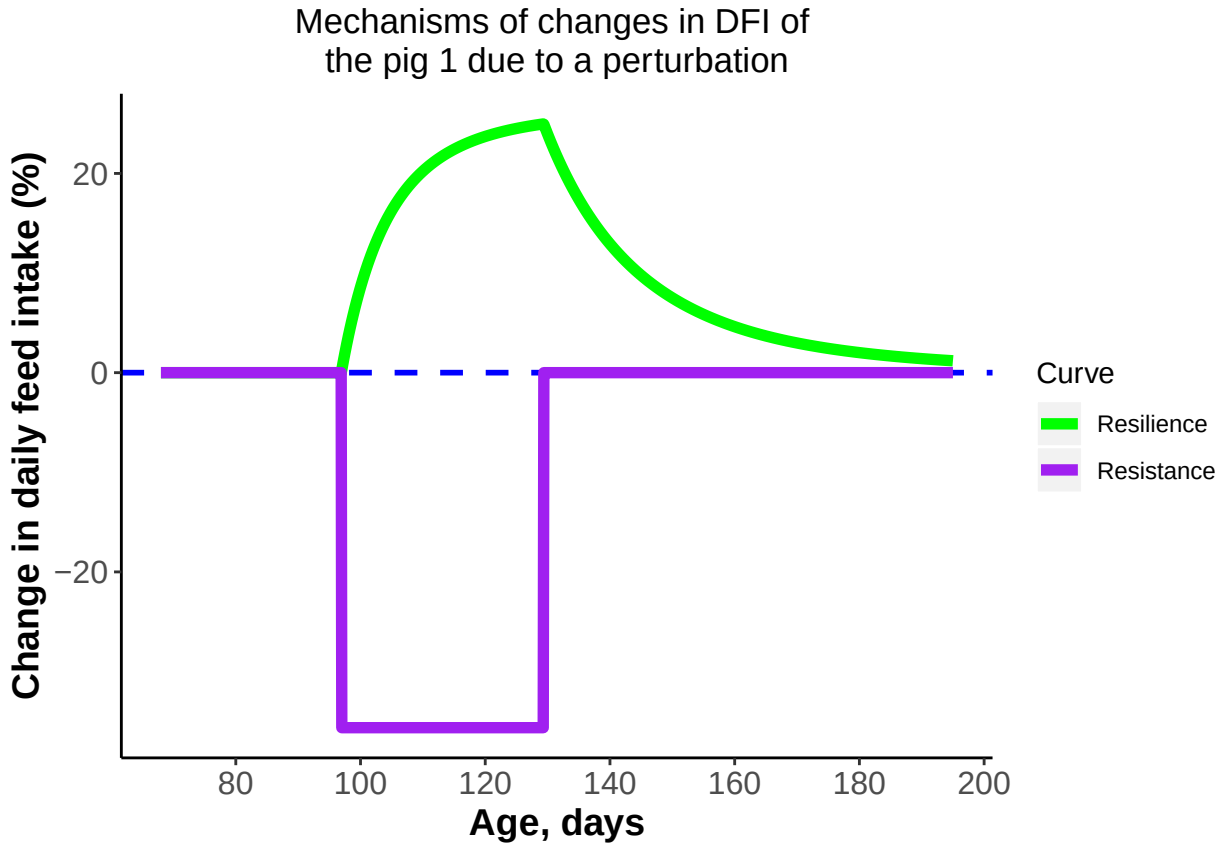
```
k1 <- -(1- P.optim[1])*100
k2 <- P.optim[2]
t_start <- P.optim[3]
t_stop <- P.optim[4]

Per.para <- c(round(t_start, digits = 0),
              round(t_stop, digits = 0),
              round(k1, digits = 1),
              round(k2, digits = 2))
names(Per.para) <- c("t_start", "t_stop", "k1", "k2")
Per.para
```

```
## t_start  t_stop      k1      k2
##   97.00  129.00  -35.60    2.81
```

4.4. Plot the results

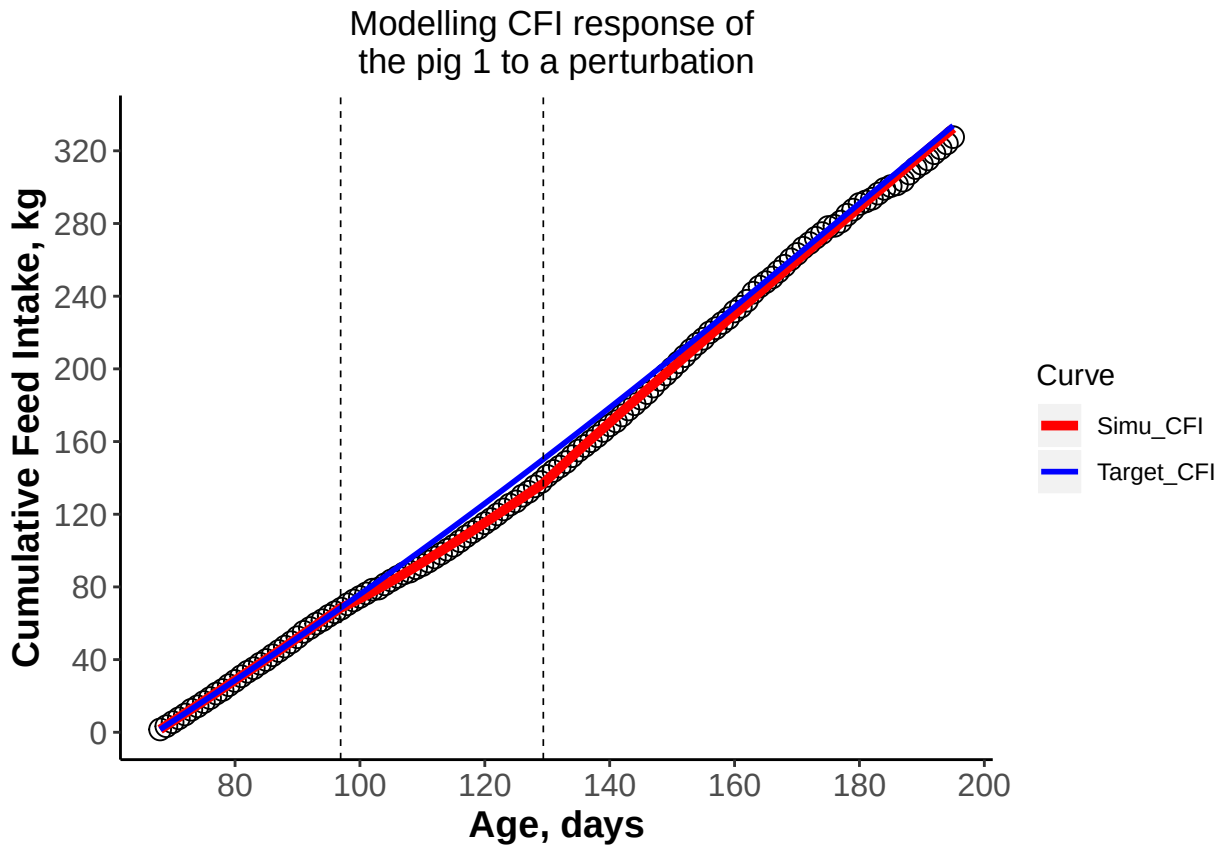
```
##-----------------------------
#  Prepare for plotting
##-----------------------------

  # Re-calculate the target CFI and DFI according to the time step
   if(FuncType == "LM"){
      ITC <- pred.abcd.0(param.i, times.ode)[[1]]
      ITD <- rep(pred.abcd.0(param.i, times.ode)[[2]], length(times.ode))
   } else if(FuncType == "QDR"){
      ITC <- pred.abcd.1(param.i, times.ode)[[1]]
      ITD <- pred.abcd.1(param.i, times.ode)[[2]]
   } else{
      ITC <- pred.abcd.2(param.i, times.ode)[[1]]
      ITD <- pred.abcd.2(param.i, times.ode)[[2]]
   }

  p1          = P.optim[1]
  max.compFI1 = P.optim[2]
  tbeg1       = P.optim[3]
  tstop1      = P.optim[4]

  Time = times.ode
  onoff = ifelse(Time>P.optim[3] & Time<P.optim[4],1,0)

  #Compensatory feed intake
```

```
  CompFI  = (1-yout[,2]/ITC)*max.compFI1
  #Simulation of DFI
  DFI.sim = (onoff*(p1-1) + CompFI + 1)*ITD
  #Ratio of DFI
  Ratio.DFI = DFI.sim/ITD

  ##-----------------------------
  #  plot the results
  ##-----------------------------
  #Compensatory feed intake and perturbation effect
  AF1 <- data.frame(cbind(Time, CompFI*100))
  names(AF1) <- c("Age.plot", "Percent")
  AF1 <- AF1 %>% mutate(Curve = rep("Resilience", length(Time)))

  AF2 <- data.frame(cbind(Time, (0- onoff*(1-p1))*100))
  names(AF2) <- c("Age.plot", "Percent")
  AF2 <- AF2 %>% mutate(Curve = rep("Resistance", length(Time)))
  AF <- rbind(AF1, AF2)

#tiff(file = paste(Data$ANIMAL_ID,".","Ratio",".png",sep=""),
#width = 5500, height = 3000, units = "px", res=600)
cols.CFI <- c("Resilience" = "green", "Resistance" = "purple")
ggplot(data = AF, aes(x = Age.plot, y = Percent)) +
geom_hline(yintercept=0, linetype="dashed", color = "blue", size = 1)+
geom_line(aes(color = Curve), size = 2) +
scale_color_manual(values = cols.CFI) +
xlab("Age, days") +
ylab("Change in daily feed intake (%)") +
# expand_limits(y=-40,100)+
scale_y_continuous(breaks=seq(-100, 100, 20)) +
scale_x_continuous(breaks=seq(60, 240, 20)) +
ggtitle(paste("Mechanisms of changes in DFI of",
              "\nthe pig",
              ID,
              "due to a perturbation")) +
theme(plot.title = element_text(hjust = .5)) +
theme(panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      panel.background = element_blank(),
      axis.line = element_line(colour = "black")) +
theme(axis.text=element_text(size=12),
axis.title=element_text(size=14,face="bold"))
```



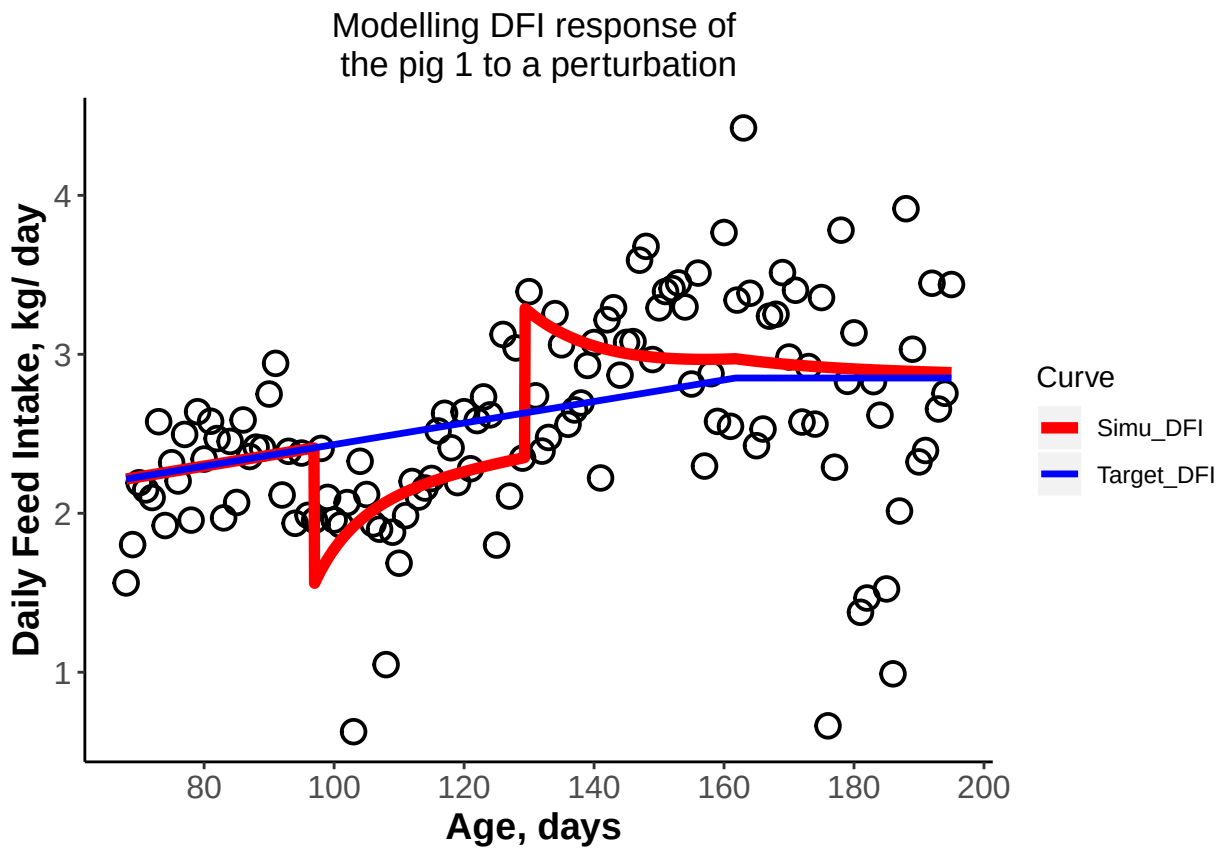

```
# dev.off()

#CFI data preparation
cf1 <- data.frame(cbind(Time, ITC))
names(cf1) <- c("Age.plot", "CFI.plot")
cf1 <- cf1 %>% mutate(Curve = rep("Target_CFI", length(Time)))

cf2 <- data.frame(cbind(yout[ , 1], yout[ ,2]))
names(cf2) <- c("Age.plot", "CFI.plot")
cf2 <- cf2 %>% mutate(Curve = rep("Simu_CFI", length(Time)))

cf <- rbind(cf1, cf2)
cf3 <- Data %>% select(Age.plot, CFI.plot)

#tiff(file = paste(Data$ANIMAL_ID,".","Simu_CFI",".png",sep=""),
# width = 5500, height = 3000, units = "px", res=600)
cols.CFI <- c("Target_CFI" = "blue", "Simu_CFI" = "red")
typs.CFI <- c("Target_CFI" = "solid", "Simu_CFI" = "solid")
size.CFI <- c("Target_CFI" = 1, "Simu_CFI" = 1.7)
ggplot(data = cf, aes(x = Age.plot, y = CFI.plot)) +
geom_point(data = cf3, aes(x = Age, y = CFI.obs),
           color = "black",
           shape = 21,
           size = 3.5,
           stroke = 0.5) +
geom_line(aes(color = Curve, linetype = Curve, size = Curve)) +
```

```
scale_color_manual(values = cols.CFI) +
scale_linetype_manual(values = typs.CFI) +
scale_size_manual(values = size.CFI) +
geom_vline(xintercept=tbeg1, linetype="dashed",
           color = "black", size = 0.3)+
geom_vline(xintercept=tstop1, linetype="dashed",
           color = "black", size = 0.3)+
xlab("Age, days") +
ylab("Cumulative Feed Intake, kg") +
scale_y_continuous(breaks=seq(0, 360, 40)) +
scale_x_continuous(breaks=seq(60, 240, 20)) +
ggtitle(paste("Modelling CFI response of", "\nthe pig", ID, "to a perturbation")) +
theme(plot.title = element_text(hjust = .5)) +
theme(panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      panel.background = element_blank(),
      axis.line = element_line(colour = "black")) +
theme(axis.text=element_text(size=12),
      axis.title=element_text(size=14,face="bold"))
```




```
# dev.off()

#DFI data preparation
df1 <- data.frame(cbind(Time, ITD))
names(df1) <- c("Age.plot", "DFI.plot")
```

```r
df1 <- df1 %>% mutate(Curve = rep("Target_DFI", length(Time)))

df2 <- data.frame(cbind(Time, DFI.sim))
names(df2) <- c("Age.plot", "DFI.plot")
df2 <- df2 %>% mutate(Curve = rep("Simu_DFI", length(Time)))

df <- rbind(df1, df2)
df3 <- Data %>% select(Age.plot, DFI.plot)

#tiff(file = paste(Data$ANIMAL_ID,".","Simu_DFI",".png",sep=""),
# width = 5000, height = 3000, units = "px", res=600)
cols.DFI <- c("Target_DFI" = "blue", "Simu_DFI" = "red")
typs.DFI <- c("Target_DFI" = "solid", "Simu_DFI" = "solid")
size.DFI <- c("Target_DFI" = 1.2, "Simu_DFI" = 2)
ggplot(data = df, aes(x = Age.plot, y = DFI.plot)) +
geom_point(data = df3, aes(x = Age, y = DFI.obs),
           color = "black",
           shape = 21,
           size = 3.5,
           stroke = 1) +
geom_line(aes(color = Curve, linetype = Curve, size = Curve)) +
scale_color_manual(values = cols.DFI) +
scale_linetype_manual(values = typs.DFI) +
scale_size_manual(values = size.DFI) +
# geom_vline(xintercept=tbeg1, linetype="dashed",
#  color = "black")+
# geom_vline(xintercept=tstop1, linetype="dashed",
#  color = "black")+
xlab("Age, days ") +
ylab("Daily Feed Intake, kg/ day") +
scale_y_continuous(breaks=seq(0, 8, 1)) +
scale_x_continuous(breaks=seq(60, 240, 20)) +
ggtitle(paste("Modelling DFI response of", "\nthe pig", ID, "to a perturbation")) +
theme(plot.title = element_text(hjust = .5)) +
theme(panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      panel.background = element_blank(),
      axis.line = element_line(colour = "black")) +
theme(axis.text=element_text(size=12),
      axis.title=element_text(size=14,face="bold"))
```


Modelling DFI response of
the pig 1 to a perturbation
Daily Feed Intake, kg/ day
Age, days
Curve
Simu_DFI
Target_DFI
1
2
3
4
80
100
120
140
160
180
200

```
# dev.off()
```