FEED-A-GENE

Adapting the feed, the animal and the feeding techniques to improve the efficiency and sustainability of monogastric livestock production systems

Deliverable D3.4

A 'Robustness' simulation model to predict short and long-term animal responses to fluctuations in nutrient supply and in environmental conditions

Due date of deliverable: M42

Actual submission date: M42

Start date of the project: March 1st, 2015 **Duration:** 60 months

Organisation name of lead contractor: INRA

Revision: V1

Dissemination level	
Public - PU	**X**
Confidential, only for members of the consortium (including Commission Services) - CO	
Classified, as referred to in Commission Decision 2001/844/EC - Cl	

Table of contents

1. Summary

Improving robustness for farm animals is seen as a new target of breeding strategies. However, robustness is a complex trait and not measurable directly. The objective of this deliverable is to quantify and characterize (elements of) robustness in growing pigs. Robustness can be characterized by examining the animal's response to environmental perturbations. Although the origin of environmental perturbations may not be known, its effect on the animal can be observed, for example through changes in voluntary feed intake. Perturbations such as heat stress and sanitary challenges typically have a negative and transitory effect on the voluntary feed intake. We developed a generic model and data analysis procedure to detect these perturbations, and subsequently characterize the feed intake response of growing pigs in terms of resistance and resilience as elements of robustness when faced with perturbations. We hypothesize that there is a target trajectory curve of cumulative feed intake, which is the amount of feed that a pig desires to eat when it is not facing any perturbations. Deviations from this target trajectory curve are considered as a perturbation, which can be characterized by its duration and magnitude. It is also hypothesized that, following a perturbation, animals strive to regain the target trajectory curve of cumulative feed intake. A model based on differential equations was developed to characterize the animal's response to perturbations. In the model, a single perturbation can be characterized by four parameters. The start and end times of the perturbation are "imposed" by the perturbing factor (e.g., a period of heat stress) while two other parameters describe the resistance and resilience potential of the animal to the perturbing factor. One parameter describes the immediate reduction in daily feed intake at the start of the perturbation (i.e., a "resistance" trait) while another parameter describes the capacity of the animal to adapt to the perturbation through compensatory feed intake to re-join the target trajectory of cumulative feed intake (i.e., a "resilience" trait). The model has been employed successfully to identify the target trajectory of cumulative feed intake in growing pigs and to quantify the animal's response to a perturbation by using feed intake as the response criterion.

The model has been developed at INRA and benefitted from feedback from frequent WP3 meetings and annual project meetings (especially from Kaposvár University, Newcastle University, Topigs Norsvin, and IRTA).

2. Results

2.1 General description of the model

Perturbations such as heat stress or sanitary challenges typically have transitory impacts on pigs during the growing period, resulting in reductions in feed intake and body weight gain. Although the cause of a perturbation is not always known, the consequences in terms of feed intake and body weight gain can be observed. Because of the rapid development in monitoring technologies, feed intake can be recorded in individual pigs with a very high frequency (i.e., up to the level of meal intake patterns). In this study, we focused on changes in voluntary feed intake patterns as indicators of perturbations. We considered perturbations that lasted at least for several days and that might have a relatively important effect on the performance of the pig. In addition, at this stage we chose not to (try to) identify the origin of the perturbing factor to "let the data speak for itself".

We hypothesized that the actual daily feed intake of a pig is the combination of a target trajectory curve (i.e., the amount of feed a pig desires to consume in a non-perturbed condition) and a change in feed intake due to a perturbation. During a perturbation, the feed intake of the pig will deviate from the target trajectory, but once the perturbation is over the pig will strive to increase its feed intake to re-join the target trajectory. We therefore assumed that compensatory feed intake might exist. We only targeted perturbations that have a negative impact on feed intake. Perturbations that result in an increase in feed intake (e.g., cold stress, immunocastration, or providing a diet with low energy content) are beyond the scope of this study.

Classical statistical techniques may be difficult to use in our approach because of the potentially large number of model parameters to be estimated. We therefore used a step-wise data analysis procedure in which first the target trajectory curve of feed intake is identified. Deviations of the actual feed intake from the target trajectory then represent the potential consequence of a perturbation. A classification process is then performed to identify the most important deviations. Lastly, a model based on differential equations is used to quantify the resistance and resilience of the pigs to the perturbation.

Identification of the target trajectory feed intake from actual feed intake data is a key element in the procedure. As indicated before, modern feeding stations can provide feed intake data very frequently. Although daily feed intake (DFI) is usually used as a production trait, we felt that DFI was not suitable to describe a target trajectory. Firstly, DFI for individual pigs is inherently variable from one day to another. Secondly, because of the existence of a target trajectory feed intake, the overall reduction in feed intake during the perturbed period should be compensated for by an equal increase in feed intake during the recovery period, due to compensatory feed intake. This would make the modelling of DFI very complicated. In contrast, cumulative feed intake (CFI) data do not suffer from these drawbacks because CFI data are much less variable and, more importantly, allow for an easier representation of a trajectory. In the absence of a perturbation, actual CFI would be similar to the target CFI. During a perturbation, the actual CFI would deviate from the target CFI (i.e., it would increase to a lesser extent) and, once the perturbing factor is over, the animal would seek to re-join the target CFI through compensatory feed intake, without surpassing it in a systematic way.

We acknowledge that many of the steps in the procedure that we propose are arbitrary and may be subjects to criticism. This concerns not only the choice of feed intake as the (only) response criterion, but also the model choices for the target trajectory feed intake and for the feed intake during and after the perturbed periods, and the step-wise method to quantify resistance and resilience traits. The objective of this deliverable is to propose the overall procedure, detailed elements of which can be adapted as judged necessary.

2.2 Data preparation

Data collection

A dataset of daily voluntary feed intake records of 116 growing-finishing pigs was used to calibrate the model. Feed and water were provided *ad libitum*. An automatic feeder recorded individual feed intake and data were averaged on a daily basis.

Missing data

Missing data is defined here as the period of one or more days for which there are no feed intake records for an animal. Missing data may be caused by a power outage, an

insensitive sensor, or loss of an identifying ear tag. Ignoring missing data would result in inappropriate CFI data, because the CFI curve would be shifted downward and the missing data could be identified as a perturbation. To correct for missing data we assumed that there is a continuous pattern of CFI data just before and after the period of missing data, characterized by a downward shift due to the missing data. When data were missing for n days, data for n+1 days before and after the period of missing data were used to perform a quadratic regression with a model that included a downward shift in CFI associated with the period of missing data. The estimated downward shift in CFI was used to calculate the values of missing feed intake.

2.3 The target trajectory CFI

The target trajectory CFI is the amount of feed a growing pig desires to eat when it is in a non-perturbed condition. In this study, we describe the target trajectory CFI by an empirical polynomial function of time, without pretending a mechanistic cause. The reason for this choice is that feed intake is recorded by time by the auto-feeders. Although feed intake has been often described as a function of body weight, it is more difficult to obtain the latter trait on a frequent basis. Moreover, describing feed intake as a function of body weight raised the issue of cause and effect (i.e., “Do animals eat because they want to grow?” or “Do they grow because they eat?”), whereas time is an independent variable. High-order polynomial functions are very flexible, but the flexibility may identify perturbations as being part of the target trajectory. Preliminary analysis using a large data set indicated that describing CFI data by a third-order polynomial in combination with a perturbation model could result biologically unrealistic predictions for DFI. Therefore, we chose a quadratic function as a basis to describe the target CFI, which implies that the target DFI (being the first-derivative of CFI) is described by a linear function of time. To improve and to control the flexibility of the function, we assumed that DFI should be either increasing or remain constant (i.e. a so-called linear-plateau model). The target CFI is therefore described by a quadratically or linearly increasing function of time:

- When $t < X_s$: $Target\_CFI(t) = a + b \cdot t + c \cdot t^2$ (E.1)
- When $t \geq X_s$: $Target\_CFI(t) = (a + b \cdot X_s + c \cdot X_s^2) + (b + 2 \cdot c \cdot X_s) * (t - X_s)$ (E.2)

where “t” is the age of the animal (days) and Target_CFI is the target CFI at day “t”. The parameters a, b, and c are the classical parameters of a polynomial function, and X_s is the day when the quadratic segment of the curve changes to the linear segment.

The parameters a, b, and c may be difficult to interpret biologically. Equation E.1 was therefore reparametrized to include the estimated age at which CFI=0 (i.e., X_0), the estimated CFI at the last observation (CFI_{last}), and the estimated CFI at the midpoint (i.e., $CFI_{mid\text{-}point}$, determined halfway between X_0 and the last observation). Equation E.2 includes one parameter more than equation E.1 (i.e., X_s), and was parameterized to include CFI_s, rather than $CFI_{mid\text{-}point}$. This parametrization facilitates the interpretation of model parameters. The model of the target CFI therefore includes three or four parameters to be estimated (i.e., X_0, CFI_{last}, and $CFI_{mid\text{-}point}$ for E.1 and X_0, CFI_{last}, X_S, and CFI_S for E.2).

When applying a linear-plateau function to estimate the target CFI, we encountered two possible problems for DFI. Firstly, the linear segment sometimes had a very modest negative slope, which would mean that the DFI decreased with increasing age. In that case, a constant value for DFI was assumed (rather than a linear-plateau model), resulting in a linearly increasing function for CFI. Secondly, X_s was sometimes estimated to be very close to either the first or the last observation and the linear-plateau curve would resemble a linear curve. A linear function was then used to describe DFI, resulting in a quadratic function for CFI.

2.4 Fitting the target trajectory CFI to the data

Fitting the target trajectory CFI to all data by the above-mentioned procedure results in an underestimation of the real target trajectory CFI, because it includes feed intake data during perturbation periods. We therefore used a fitting procedure to successively eliminate data that could result from perturbed periods. Data that were consistently below the fitted curve could be due to perturbations and an auto-correlation test was used as a selection criterion. If there was a significant autocorrelation for the residuals (i.e., the difference between the observed and predicted values), the data with negative residuals were (temporarily) removed from the dataset. The fitting procedure was then repeated on the resulting dataset until the autocorrelation of the residuals was no longer significant. Compared to fitting the curve to the original dataset of CFI, the

procedure results in moving the CFI curve upwards, while fitting to fewer data compared to the original dataset. Preliminary analysis indicated that a significant autocorrelation remained until few data remained. This is due to the existence of small oscillations in CFI, while these oscillations are not necessarily the result of perturbations. To ensure that the target CFI was estimated with a reasonable number of data, the fitting procedure was terminated when there was no autocorrelation among the residuals, or when at least 20 observations were remaining.

2.5 Identifying perturbations

Once the target trajectory CFI is identified, deviations between this curve and the actual CFI data can be used to quantify perturbations. As indicated above, small (natural) oscillations in feed intake patterns seem to exist and we only aim to quantify the most important deviations that are the result of perturbations. These perturbations can be characterized by the duration and magnitude of the deviation from the target CFI. The duration of each deviation is determined by the number of days in which the actual CFI data are below the target trajectory curve. We considered a deviation as a perturbation if it lasted at least 5 days, to ensure the correct identification of the parameters of the perturbation model. The magnitude is calculated by the maximum reduction of a deviation from the target trajectory. Because CFI is increasing continuously, deviations from the target CFI were expressed as a percentage and an arbitrary value of 5% was set as a threshold value to identify a perturbation. In summary, the actual CFI data was compared to the target CFI curve and any period in which the actual data deviated from the target CFI for more than 5 days and for more than 5% was considered the result of a perturbation. Other deviations were considered part of naturally occurring oscillations in feed intake patterns.

A B-spline function was fitted to all CFI data. This function was chosen because of its flexibility in capturing different shapes of deviations. The difference between the B-spline and target CFI was calculated to identify the number of perturbed periods, the approximate start and end, and the magnitude of each perturbation. The factors controlling smoothness of the B-spline were arbitrary chosen by visualization.

We decided not to consider deviations that occurred only during the first week because pigs may encounter many stressors then, and deviations from the target CFI

of more than 5% are quite common during the first week. 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 were considered the result of a perturbation.

2.6 Modelling the response to a perturbation

Once the perturbed period is identified, each single perturbation was characterized by a system of differential equations. The two main driving forces of the model are the impact of the perturbing factor and the adaptation capacity of the pig. We assumed that a perturbation would have a negative and constant impact on the DFI of the pig for the duration of the perturbing factor (Figure 1, red line). The negative impact of a perturbing factor is therefore characterized by three model parameters: the start and the end of the perturbing factor, and the instantaneous response of the pig to the perturbing factor. The reduction in DFI will result in that the actual CFI will start to deviate from the target CFI. The ratio between the actual CFI and the target CFI is used as a driving force to model the adaptation capacity of the pig (Figure 1, green line). The change DFI (compared with the target DFI) is the sum of the mechanisms depicted by the red line (resistance) and the green line (resilience and compensatory feed intake). During the perturbed period, the resilience mechanism limits the effect of the perturbing factor. As can been seen from Figure 1, at the end of the perturbed period, the negative effect of the perturbation (-80%) is partially compensated for by the resilience (+54%). Once the perturbing factor is over, the resilience mechanism induces compensatory feed intake so that the actual CFI can regain the target CFI. The area under the curve of the green line equals that of the red line, ensuring that the animal seeks complete recovery.

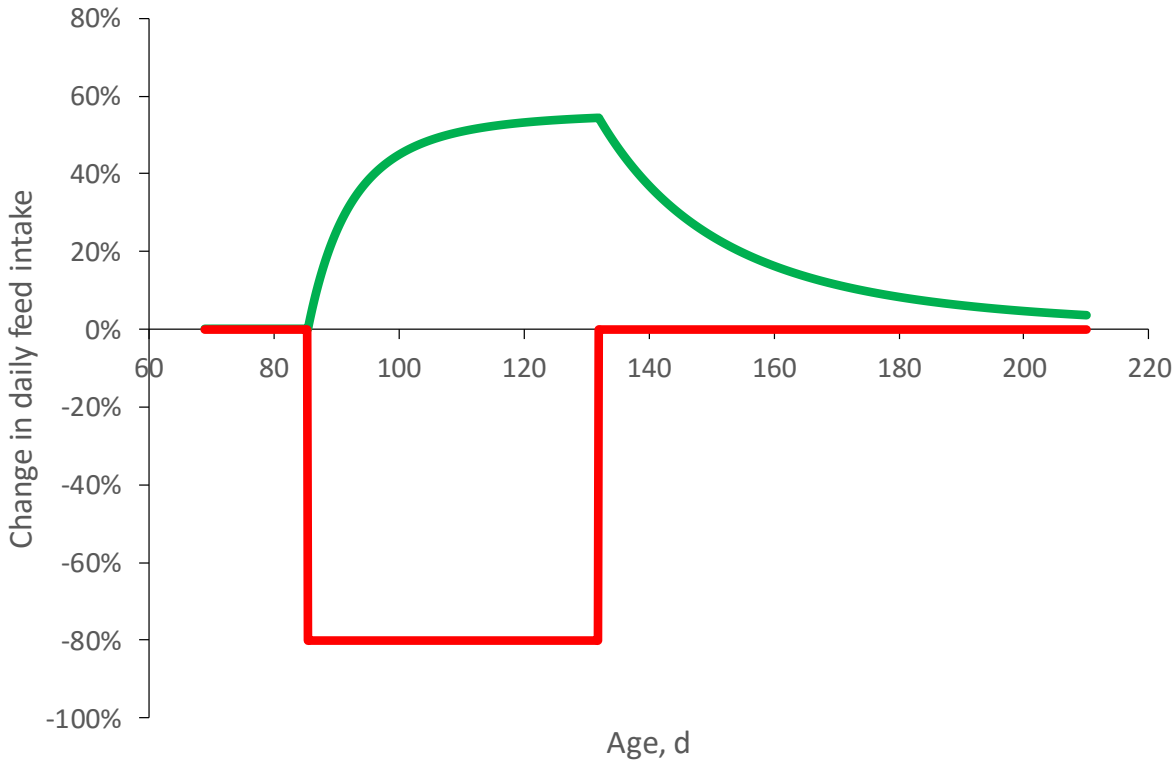


Figure 1. Mechanisms that determine the response of a pig to a single perturbation. The perturbation is assumed to occur between days 85 and 132 of age. The red line indicates the constant and negative impact of the perturbation on the pig, resulting in a 20% reduction in DFI. The green line panel represents the resilience capacity (during the perturbation) and compensatory feeding of the pig (after day 132).

As indicated in Figure 1, resistance and resilience operate at the level of DFI, but the control of resilience and compensatory feed intake is regulated at the level of CFI (Figure 2). The DFI encompasses three elements: the target DFI, the impact of perturbation on DFI, and the response of the pig to the perturbation. In the absence of a perturbation, the actual DFI is equal to the target DFI (① in Figure 2). The initiation of a perturbation (② in Figure 2) has a negative and constant effect on DFI and, because of the reduction in DFI, the actual CFI curve starts to deviate from the target CFI. The ratio between the actual CFI and the target CFI triggers the pig's resilience mechanism in a proportional way (③ in Figure 2). The smaller the ratio between the actual and the target CFI, the greater will be the resilience mechanism for DFI. Once the perturbing factor is over (④ in Figure 2), its effect on DFI disappears, but the CFI ratio will still be smaller than one. This results in compensatory DFI where the actual DFI will be greater than the target DFI feed intake that, in turn, results in that the actual CFI will approach the target CFI.

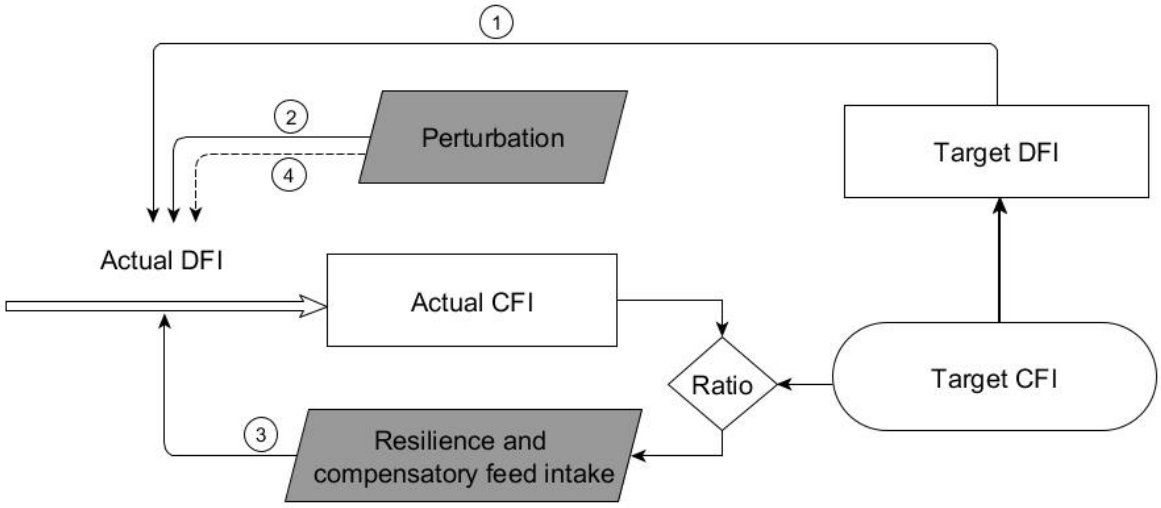


Figure 2. General mechanism of a model that quantifies the pig's response to a perturbation**.** Solid arrows indicate causal relationships in the model, the double arrow indicates the flux, and the dashed arrow indicates the disappearance of perturbing factor. Numbers indicate the response's steps of a pig when a perturbation occurs.

The perturbation model is conceptualized in a way that the impact of a single perturbation on the feed intake of the pig and its response can be characterized by four parameters. Two parameters indicate the start and end times of the perturbing factor, while the third parameter describes the constant negative impact of the perturbation on DFI (red line in Figure 1). The fourth parameter (k) is the marginal response in DFI due to a change in the ratio between the actual and the target CFI, and describes the capacity of the animal to adapt to the perturbation through resilience and compensatory feed intake (green line in Figure 1). The DFI is affected in a proportional way by the ratio between the actual and target CFI and the parameter "k".

The model equations are written as follows:

- $Actual\_DFI(t) = Target\_DFI(t) \times (1 - perturbation(t) + resilience(t))$ (E.3)

The "perturbation(t)" takes a constant value (e.g., an 80% reduction in DFI) between the start and end time of the perturbing factor, and is zero otherwise. The "resilience(t)" is described by:

- $Resilience(t) = (1 - Ratio) \times k = (1 - Actual\_CFI(t) / Target\_CFI(t)) \times k$ (E.4)

The target DFI and target CFI are given by equations E.1 and E.2, while the actual CFI is the result of integrating the actual DFI between the time when the estimated CFI equalled zero (X_0) and the last observation (i.e., X_{last}), i.e.,:

$$\text{Actual\_CFI}(t) = \int_{X_0}^{X_{last}} \text{Actual\_DFI}(t) \quad \text{(E.5)}$$

2.7 Statistical analysis

All statistical and optimization procedures of the model have been performed in R software version 3.5.0 (http://cran.r-project.org/). To account for scale differences in CFI, a weighted regression procedure was used (using $(1/CFI)^2$ as statistical weight) to estimate the parameters of the target CFI (equations E.1 and E.2). The optimization was performed using non-linear function *nlsLM* of the package "minpack.lm". The test for auto-correlation was performed by Wald-Wolfowitz runs test. To identify perturbations from the CFI curve, the package "fda" was utilized. To characterize the pig's response to a perturbation, the differential equations E.4, E.5, and E.6 were solved using the *ode* function of the "desolve" package with an integration step size (dt) of one day. The optimization was conducted by the *optim* function.

2.8 An example of "the model at work"

Figure 3 is an example of the change in DFI for an individual animal. It illustrates the large day-to-day variation in DFI, but it also shows a strong reduction in DFI somewhere between 70 and 130 days of age.

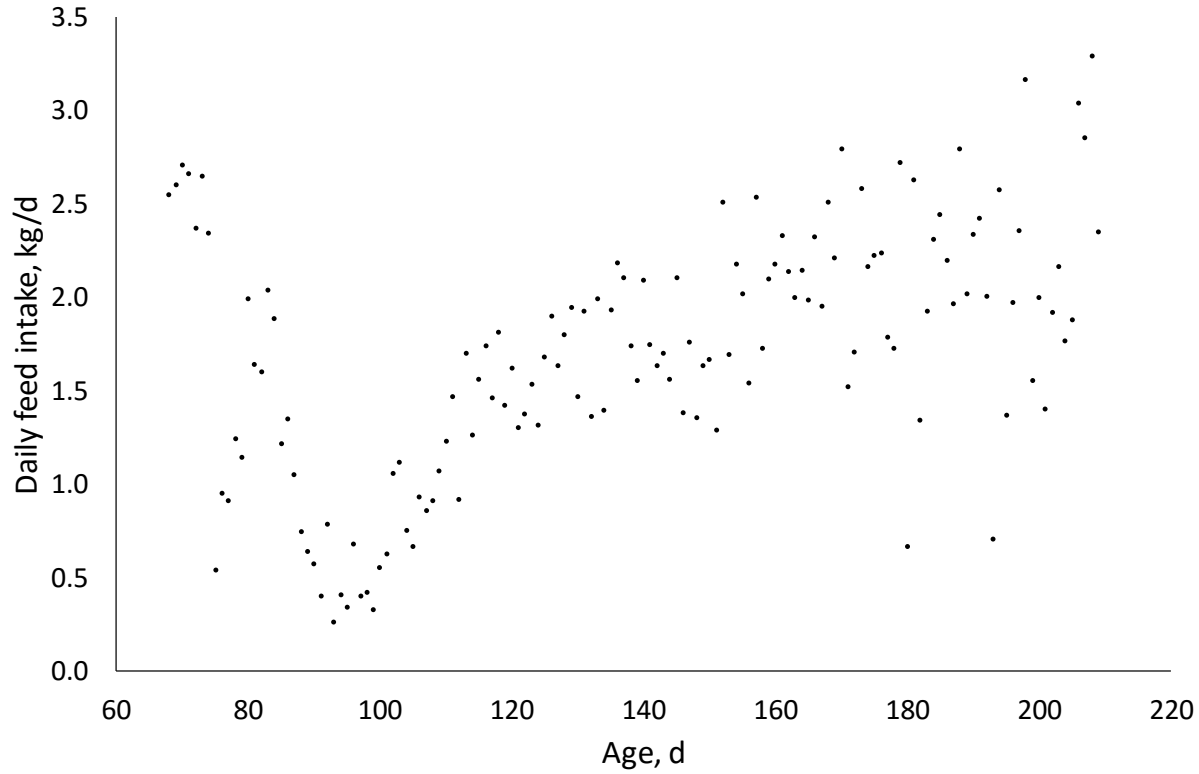


Figure 3. Daily feed intake of an individual pig.

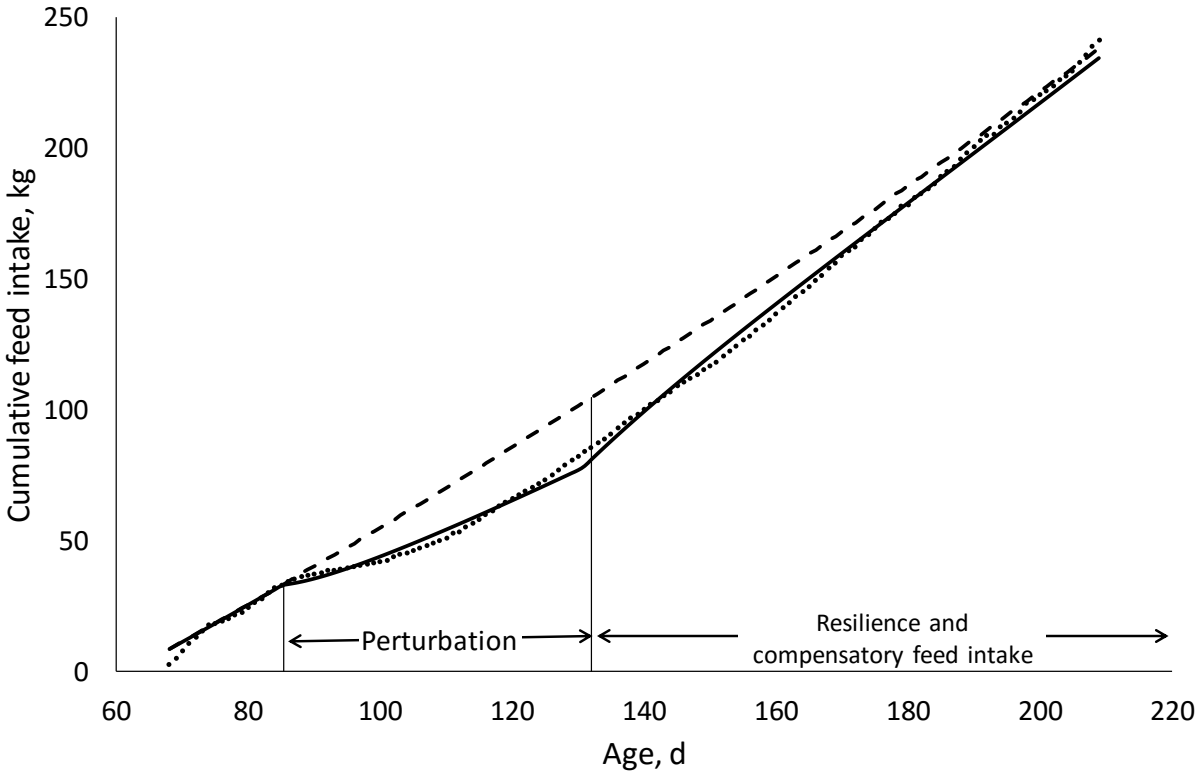


Figure 4. Cumulative feed intake of an individual pig and modelling results. Observations are given as individual points, the target trajectory of CFI by the dashed line, and the fit of the model including the target trajectory and the perturbation module by the solid line.

The data in Figure 4 is the same as that of Figure 3, but feed intake is now expressed on a cumulative basis. The dashed line indicates the estimated target trajectory of CFI

using a quadratic function of age. This target trajectory of CFI resulted applying the model first to all the data, followed a test for autocorrelation of the residuals and, if significant, removal of the data points for which the residuals were negative. The procedure was repeated until the autocorrelation was no longer significant or until 20 data points remained. As can be seen in Figure 4, the estimated target trajectory of CFI is determined to a great extent by the highest data points. The functional data analysis procedure was then used to estimate the approximate duration and magnitude of the perturbation. The perturbation model was then fitted to the data to estimate the parameters of the perturbation model (i.e., the start and end of the perturbation period, the instantaneous reduction in DFI during the perturbation period, and the resilience parameter "k". The parameters of the model are given below:

Target trajectory of CFI		Perturbation model	
X_0, days	61.7	$X_{start\ of\ perturbation}$, days	85.4
CFI at the last observation, kg	240.1	$X_{end\ of\ perturbation}$, days	132.0
CFI at the midpoint, kg	111.0	Instantaneous reduction in DFI	80%
		Resilience parameter k	2.3

Although the quadratic model can be described as $CFI = a + b \cdot age + c \cdot age^2$, the parameters a, b, and c were re-parameterized to include the age at which the estimated CFI equals zero (X_0), the estimated CFI at the last observation (i.e., at 210 days of age here), and the estimated CFI at the midpoint (i.e., at (210+61.7)/2= 135.85 days of age here). The start of the perturbation was estimate at 85.4 days of age and lasting until 132 days of age. At the start of the perturbation, DFI was reduced by 80%, and this negative effect on the DFI continued to be exerted during the perturbed period. Although the instantaneous reduction in DFI will be 80%, the reduction will be less with progressing time because of the resilience mechanism (driven by the ratio between the actual CFI and the target CFI). The resilience parameter (k = 2.3) indicates that if the actual CFI is 1% below the target CFI, the pig would strive to eat 2.3% more compared to the target DFI (i.e., 0.01 * 2.3 = 2.3%). At 132 days of age, the negative effect of the perturbing factor will cease, but the resilience mechanism remains active because the actual CFI is still well below the target CFI. With time progressing, the actual CFI will gradually approach the target CFI through compensatory feed intake.

3. Conclusions

A mathematical and statistical data analysis procedure was developed to characterize the response of an animal to a perturbation. The response on feed intake was described here, but the method can be extended to quantify other response traits. The different steps in the analysis procedure can easily be adapted by end-users. For example, in the procedure given here, the response to a perturbation of unknown origin was described, but the origin of some perturbations (e.g., heat stress) can be easily identified. Moreover, high temperatures affect all animals in the building, but the way they respond to it can vary from one animal to another. Differences in the response of animals to a common stressor can be quantified by adapting the procedure. The quantification of resistance and resilience traits has a great potential for use in animal breeding programs to improve efficiency and robustness.

The analysis procedure will be submitted for publication in a peer-reviewed journal and the accompanying R-script will be deposited in a public repository.

4. Annexes

The R-script of the deliverable is given in the annex, including a description of the different steps.

Detection and characterization of the response of a growing pig to a single perturbation

Hieu Nguyen Ba, Masoomeh Taghipoor, Jaap van Milgen

August 10th 2018

Note:

- Authors of this documment are currently employed by French National Institute for Agricultural Research (INRA) at ***PEGASE*** and ***MoSAR*** units.
- The project receives funding from ***Feed-a-Gene*** and ***INRA-ACCAF*** programs.
- This R-script describes the process of a data analysis procedure and modelling to detect and characterize the response of an individual pig to perturbation.
- The process includes 1 step of data preparation and 4 steps of data analysis.

Step 0: Data preparation

The purpose of this step is to import data into R and for the treatment of missing data

1. Set a working directory and load required packages

Note: Make sure to install all the packages below if you have not had them yet.

```
#To remove the hisory and reset the whole project
rm(list=ls(all=TRUE))

#Set working directory
setwd("C:/Users/hnguyenba/Dropbox/INRA/Modelling_Perturbation/Feed-a-gene/Deliverable3.4")

#Packages
 ##Data manipulation and graph
 library(ggplot2)
 library(dplyr)
 ##Estimate parameters by NLS
 library(minpack.lm)
 library(nlstools)
 library(proto)
 library(nls2)
 ##Functional Data Analysis
 library(fda)
 ##Functions for solving differential equations
 library(deSolve)
```

2. Import data intoto R and attribute a name

```
JRP_NA.0<- read.csv("RFI_JRP.csv", header =TRUE, sep=";", dec=",")
```

3. Homogenous missing data types

There are two types of missing data:

- One or several rows are missing
- One or several days recorded as (.)

We homogenize two types of missing data into one (missing row) by omitting the rows containing missing feed intake data. Therefore, one homogeneous treatment can be applied later on for both types of missing data.

```
# Replace missing values by NA (NA means not available)
  JRP_NA.0[JRP_NA.0 == '.'] <- NA
#Obmit all places containing missing data (for further treatment)
  JRP_NA<-JRP_NA.0[!is.na(JRP_NA.0$FEED_INTAKE),]
```

4. Correct and clean data

Correct data:

- `ID` of pig has to be in *factor*
- `Age` and `Feed intake` of the pig have to be in *numeric*

```
  JRP_NA$ANIMAL_ID <- as.factor(JRP_NA$ANIMAL_ID)
  JRP_NA$AGE <- as.numeric(JRP_NA$AGE)
  JRP_NA$FEED_INTAKE <- as.numeric(as.character(JRP_NA$FEED_INTAKE))
```

Clean data:

- Only information of animal **ID**, **Age** (days) and daily feed intake (**DFI**) are needed

```
JRP_NA <- JRP_NA[, c(-4,-5)]
```

- Choose one animal to analyse by its order number

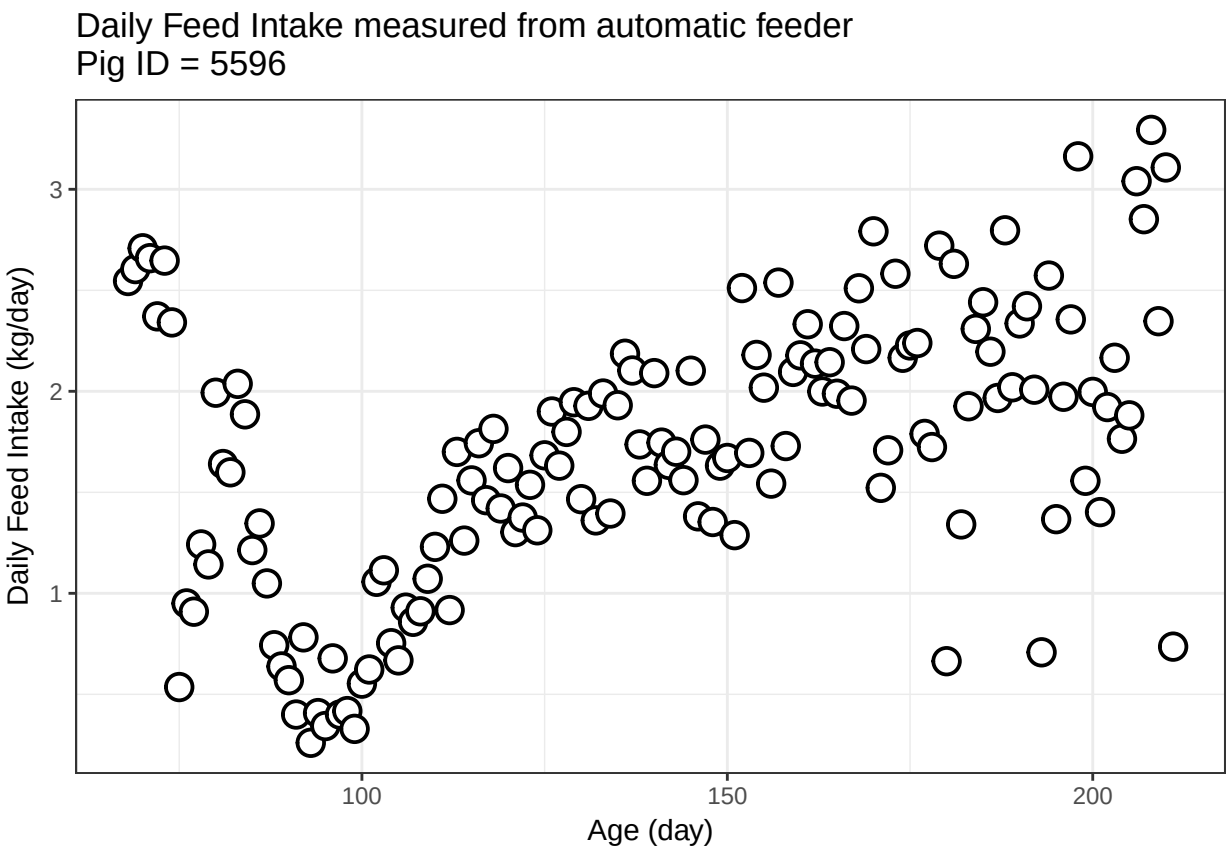
```
#Order number of the animal
 idc = 66 #Example: of LM: 41, 42; QLM: 33; QDR: 63
 i = ID[idc]

#extract dataset for the chosen animal
 Data <- JRP_NA[JRP_NA$ANIMAL_ID == i,]

#Age vector associated with the chosen animal
 Age.plot = Age[Pig_ID == i]

#DFI vector associated with the chosen animal
 DFI.plot = DFI[Pig_ID == i]
```

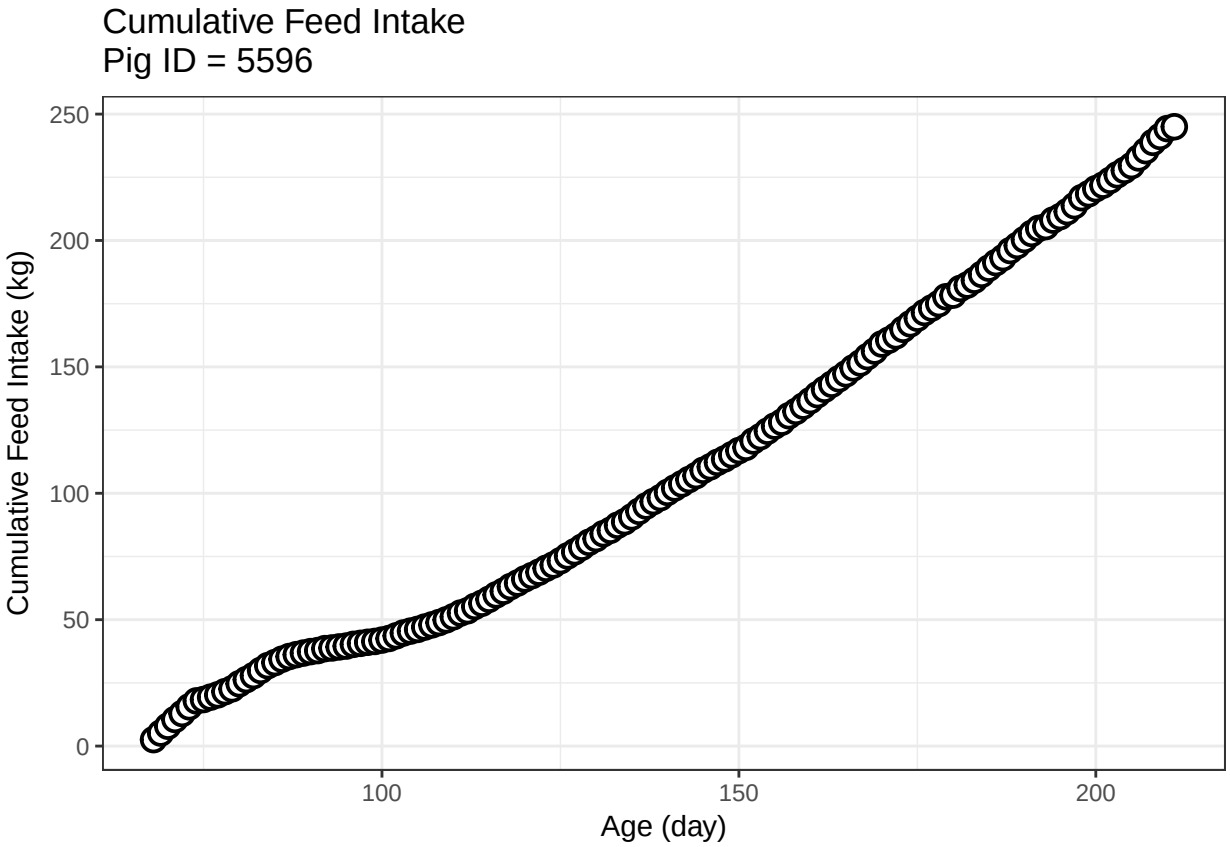
Calculate cumulative feed intake (**CFI**) from **DFI** data

```
  CFI.plot = c()
  CFI.plot[1] = DFI.plot[1]

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

Data overview:

- Plot


Daily Feed Intake measured from automatic feeder
Pig ID = 5596
Daily Feed Intake (kg/day)
3
2
1
100
150
200
Age (day)



- Dataset (six first rows)

```
##   Age.plot DFI.plot CFI.plot
## 1       68    2.546    2.546
## 2       69    2.605    5.151
## 3       70    2.708    7.859
## 4       71    2.659   10.518
## 5       72    2.371   12.889
## 6       73    2.647   15.536
```

Step 1: Missing Data Treatment

Missing data is defined here as the period of one or more days for which there are no feed intake records for an animal. Missing data may be caused by a power outage insensitive sensor or loss of an identifying ear tag. Ignoring missing data would result in inappropriate CFI data, because the CFI curve would be shifted downward and the missing data could be identified as a perturbation.

Therefore, missing data needs to be estimated before doing further analysis.

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 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 day is at day 185 and 186, all these tree data removed from the dataset (the dataset then ends at day 184)

1.1.1. Detect missing data

a. Detection of missing lines

```
#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

#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]
if(length(A4) ==0){
     paste( "There are 0 missing row from data of the pig", i)
  } else {
     paste( "There are", length(A4), "missing row(s) from data of the pig", i)
 }
```

```
## [1] "There are 0 missing row from data of the pig 5596"
```

If there is no missing Data, we can skip steps below; otherwise, we continue

```
## [1] "skip"
```

b. List of ages associated with missing rows

```
#If there is no missing Data, we can skip steps below
if(length(A4) == 0){
        "skip"
} else{
```

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

#JRP_new with no missing rows
#This dataframe will be used to compare with JRP_new_Final 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)
     }
 }
 paste( "Age (days) of missing data:", MissAgeT)
}
```

```
## [1] "skip"
```

1.1.2. Classifying missing data

A **while loop** is created to automatically classify all missing data in dataset.

```
## [1] "skip"
```

While loop:

```
#If there is no missing Data, we can skip steps below
if(length(A4) == 0){
        "skip"
      } else{

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
#Ex. 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
 }


#2. when there are not enough given rows between two series of missing rows
#Ex. 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(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)
   }
  }
 }
}

#3. When number of data after missing row is not enough for estimation
#Ex. 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
 break
}
```

```
#-----------------------------------------------------
# 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 = c()
 CFI.plot[1] = DFI.plot[1]
 for(j in 2:length(Age.plot)){
    CFI.plot[j] <- CFI.plot[j-1]+DFI.plot[j]
    }

#-----------------------------------------------------
# New dataset with removed NA only containing 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] "skip"
```

1.2. Missing data estimation

To correct for missing data we assumed that there is a pattern for CFI data before and after the period of missing data, characterized by a downward shift due to the missing data. When data were missing for n days, data for n+1 days before and after the period of missing data were used to perform a quadratic regression with a model that included a downward shift in CFI after the period of missing data. The estimated downward shift in CFI was used to replace missing values.

1.2.1. Estimating missing data

We need to empty the lists before we run the new estimation (similar to previous step).

```
## [1] "skip"
```

Therefore, the while loop is run again (similar to previous step).

```
## [1] "skip"
```

Missing age and its feed intake value

```
## [1] "skip"
```

1.2.2. Remove the last day from dataset

Given that animals are normally fasted one day before sending to slaughter, the feed intake information of all pig in the last day was eliminated from the dataset.

- Before last row is removed

```
##     Age.plot DFI.plot CFI.plot
## 139      206    3.040  232.661
## 140      207    2.852  235.513
## 141      208    3.294  238.807
## 142      209    2.347  241.154
## 143      210    3.108  244.262
## 144      211    0.736  244.998
```

- Remove information of the last day:

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

- After last row is removed:

```
##     Age.plot DFI.plot CFI.plot
## 138      205    1.881  229.621
## 139      206    3.040  232.661
## 140      207    2.852  235.513
## 141      208    3.294  238.807
## 142      209    2.347  241.154
## 143      210    3.108  244.262
```

Step 2: Identify the target trajectory curve of CFI

- We hypothesize that there is a *target trajectory curve* (**TTC**) of CFI, which is the amount of feed that a pig desires to eat when it is not facing any perturbations. This TTC is then used as a benchmark for detecting and perturbation.
- The actual CFI curve is assumed to be a combination of the target trajectory curve and a change in feed intake due to a perturbation. During a perturbation, the feed intake of the pig will deviate from the target trajectory, but once the perturbation is over the pig will strive to increase its feed intake to re-join the target trajectory.
- Therefore, this target trajectory can be identified by excluding from the actual CFI curve the feed intake in periods during which the pig could have been perturbed.

```
#=============================================================
# DATA PREPARATION
#=============================================================
  #Functions
  source("abcd.R")

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

  #Load data
  Data = No.NA.Data[No.NA.Data$ANIMAL_ID == ID,]

  ####### 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))
```

2.1. Finding the suitable function for TTC curve:

A quadratic-linear function (**QLM**) is chosen as the function to describe TTC curve of CFI:

When X $<$ Xs:

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

When X $\geq$ Xs:

$$Y = (a + bXs + cXs^2) + (b + 2cXs) * (X \check{} Xs)$$

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

- Where X is the age of the animal (day)
- Y is the CFI at day X
- The parameters a, b and c are the classical parameters of a polynomial function
- Xs the day when the quadratic segment of the curve changes to the linear segment

The parameters a, b, and c were reparametrized to:

- Age at which CFI is equal 0 (**X0**)
- **Xs**
- Estimated DFI and CFI at age Xs (**DFIs**)
- Estimated DFI age Xs (**CFIs**)

Note: This is a little different from what is described in the deliverable. In the deliverable, we explained that the parameters **a**, **b** and **c** were reparametrized by **X0**, **Xs**, **CFIs** and **CFI_last** (the estimated CFI at the last observation). The two ways of reparametrization are both possible and provide the same results in parameters **a**, **b** and **c** for the target trajectory curve. The reason we changed this reparametrization in the deliverable is to make it consistant with the reparametrization in other functions (quadratic and linear, see coming sections for more detail).

The function with initial parameters are fitted to CFI data:

```
source("abcd.R")
#=============================================================
# Filtration process to choose suitable function for TTC of CFI
# We fit a quadratic-linear (Q-LM) function for CFI data
# Thus a linear-plateau function is fitted to DFI data
#=============================================================

#Initial parameteres for parameter estimation
   X0.0 = x[1]
   Xlast = x[length(x)]

#Provide set of initial parameters
   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")

 #---------------------------------------------------------------------
 # Estimate parameters by NLS2 with upper and lower bound
 #---------------------------------------------------------------------

    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
##  68.000 209.000   2.347 241.154
```

Based on the estimated parameters obtained from `par_init` the function of TTC can be chosen:

- If the estimated Xs is too closed to either the begining or the end of growing period (10% each side) -> ***Quadratic function***
- If the slope of the function's derivative (slope of DFI curve) is negative -> ***Linear function***
- None of these option above -> ***Quandratic-linear function***

Note: FuncType means function type

- **LM** means linear function
- **QDR** means quadratic function
- **QLM** means quadratic-linear function

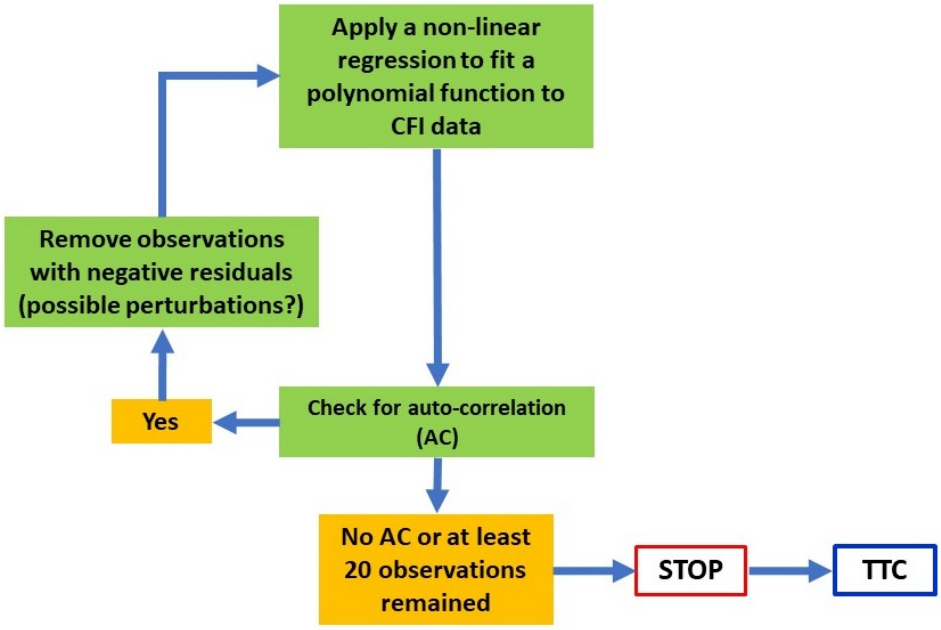
```
#---------------------------------------------------------------------
# Classify CFI data to 3 groups with different TTC function
#---------------------------------------------------------------------
 #Calculate Slope of DFI curve
    c <- abcd.2(par_init)[3]
 #Determine the end of growing period (then compared to Xs)
    Xs.0 <- par_init[2]

    if(c < 0){
      FuncType <- "LM"
    } else if(c > 0 &&
              Xs.0 >= ((Xlast - X0.0)*0.2 + X0.0)
              ){
      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 for CFI data to estimate the TTC", sep = ""))
```

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

2.2. Identify the TTC by filtration process:

After the correct function is chosen, the TTC of CFI can be identified by this process:



2.2.1. A linear function is applied for CFI

The equation of the linear function:

$$Y = a + bX$$

Parameters a and b are reparametrized to:

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

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

    #Initial parameteres for parameter estimation
    x0.0 = x[1]
    xlast <- x[length(x)]
    ylast.0 = Y[length(x)]
```

```
#Vector contains 4 initial parameters
par_init <- c(x0.0, ylast.0)


#########################################################################
# 1. reparametrization CFI at X0 = 0
#function used for reparametrization in MAPLE
# solve({0 = X0*b+a, Ylast = Xlast*b+a}, {a, b});
# a = (Ylast*X0)/(X0 - Xlast), b = -(Ylast)/(X0-Xlast)
#  2. with the source of the function abcd and pred
#########################################################################


#-------------------------------------------
# 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 procedure of Non Linear Regression
#-----------------------------------------------------------------------------------
#
while (AC_pvalue<=0.05 && datapointsleft >= 20){
  weight = 1/Y^2
#---------------- NON linear reg applied to log(Y) ---------------

  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,])

  # #summary
  # summ = overview((nls.CFI))  #summary
  #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")
  # plot(Data.new$Age, Data.new$Predicted_CFI, type = "l", col = "black",lwd = 2,
  #      ylim = c(0, max(Data.new$Predicted_CFI, Data.new$Observed_CFI)))
  # lines(Data.new$Age, Data.new$Observed_CFI, type = "p", cex = 1.5)
  #
  #remove negative res
  Data.pos <- Data.new[!Data.new$Residual<0,]
  # lines(Data.pos$Age, Data.pos$Predicted_CFI, type = "l", col = j-1, lwd = 2)
  # lines(Data.pos$Age, Data.pos$Observed_CFI, type = "p", col = j, cex = 1.5)

  #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( 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.trst",
                         "DPL",
                         "FuncType"
    )

    param.2

    #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( ID, dim(Data.new)[1])
    Data.new = cbind(ANIMAL_ID , Data.new)

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

2.2.2. A quadratic function is applied for CFI

The equation of the linear function:

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

Parameters a and b are reparametrized to:

- **X0**: the estimated age at which value of CFI is equal to 0
- **Xlast**: the estimated age at the end of the growing period
- **y2**: the estimated CFI at the midpoint (or CFImid-point in the deliverable)
- **ylast**: the estimated CFI value at the last observation (or CFIlast in the deliverable)

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

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

    #Initial parameteres for parameter estimation
    xlast <- x[length(x)]
    x0.0 = x[1]
    y2.0 = Y[floor(length(x)*2/3)]
    ylast.0 = Y[length(x)]
```

```
#Vector contains 4 initial parameters
par_init <- c(x0.0, y2.0, ylast.0)

x2 = 2*(xlast - x[1])/3+x[1]

######################################################################
# 1. reparametrization CFI at X0 = 0
#function used for reparametrization in MAPLE
# solve({0=a+b*X0+c*X0**2+d*X0**3,
# y1 = a + b*(X0+1/3*(xlast-X0)) +
#   c*(X0+1/3*(xlast-X0))**2 + d*(X0+1/3*(xlast-X0))**3,
# y2 = a + b*(X0+2/3*(xlast-X0)) + c*(X0+2/3*(xlast-X0))**2 +
#   d*(X0+2/3*(xlast-X0))**3,
# ylast = a+b*xlast+c*xlast**2+d*xlast**3},{a,b,c,d});
#  2. with the source of the function abcd and pred
######################################################################

#-------------------------------------------
# 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 procedure of Non Linear Regression
#-------------------------------------------------------------------------------
#
while (AC_pvalue<=0.05 && datapointsleft >= 20){
  weight = 1/Y^2
#---------------- NON linear reg applied to log(Y) -----------------

  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, c and d
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])*2/3+coef(nls.CFI)[1]
# #summary
# summ = overview((nls.CFI))  #summary
#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.1(par[[j]],x)[[1]], res, Step))
names(Data.new) <- c("Age",
                     "Observed_DFI",
                     "Observed_CFI",
                     "Predicted_CFI",
                     "Residual",
                     "Step")
# plot(Data.new$Age, Data.new$Predicted_CFI, type = "l", col = "black",lwd = 2,
#      ylim = c(0, max(Data.new$Predicted_CFI, Data.new$Observed_CFI)))
# lines(Data.new$Age, Data.new$Observed_CFI, type = "p", cex = 1.5)
#
#remove negative res
Data.pos <- Data.new[!Data.new$Residual<0,]
# lines(Data.pos$Age, Data.pos$Predicted_CFI, type = "l", col = j-1, lwd = 2)
# lines(Data.pos$Age, Data.pos$Observed_CFI, type = "p", col = j, cex = 1.5)

#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( 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.trst",
                          "DPL",
                          "FuncType"
    )
    param.2
    #Add one column about the slope of DFI to the function
    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; param.2

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

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

} else{
 }
```

2.2.3. A quadratic-linear function is applied for CFI

The reparametrization was described above (see **2.2** for detail).

```
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")])
  source("abcd.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 procedure of Non Linear Regression
 #-------------------------------------------------------------------------------

 while ((AC_pvalue<=0.05) && datapointsleft >= 20){
  weight = 1/Y^2
  # ---------------- NON linear reg applied to log(Y) ----------------
  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)
  par_init
  # st1 <- st1[!(st1$Xs == par_init[2]),]
  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, c and d
param[j,] = par[[j]]
param.2[j-1,] = cbind(param[j,],par.abcd[j,])

#summary
# summ = overview((nls.CFI))  #summary
#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.2(par[[j]],x)[[1]], res, Step))
names(Data.new) <- c("Age",
                     "Observed_DFI",
                     "Observed_CFI",
                     "Predicted_CFI",
                     "Residual",
                     "Step")
# plot(Data.new$Age, Data.new$Predicted_CFI, type = "l", col = "black",lwd = 2,
#      ylim = c(0, max(Data.new$Predicted_CFI, Data.new$Observed_CFI)))
# lines(Data.new$Age, Data.new$Observed_CFI, type = "p", cex = 1.5)

#
#remove negative res
Data.pos <- Data.new[!Data.new$Residual<0,]
# lines(Data.pos$Age, Data.pos$Predicted_CFI, type = "l", col = j-1, lwd = 2)
# lines(Data.pos$Age, Data.pos$Observed_CFI, type = "p", col = j, cex = 1.5)

#restart
#Criteria to stop the loop when the estimated parameters are equal to initial parameters
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$Observed_DFI
Data.xy <- as.data.frame(cbind(x,Y))
dpl = c(dpl, datapointsleft)
dpl
#Create again the grid
X0.0 <- x[1]
Xlast <- x[length(x)]
#Xs
if(par_init[2] -15 <= X0.0){
   Xs.1 = round(seq(X0.0 + 5, Xlast - 5, len = 30), digits = 0)
} else if(par_init[2] + 5 >= Xlast){
  Xs.1 = round(seq(par_init[2]-10, par_init[2]-1, len = 6), digits = 0)
} else{
  Xs.1 = round(seq(par_init[2]-5, par_init[2] + 5, len = 6), digits = 0)
}
#
X0.1 = rep(X0.0, length(Xs.1))
DFIs.1 <- c()
CFIs.1 <- c()
for(A in 1:length(Xs.1)){
   DFIs2 = Data[Data$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("X0","Xs", "DFIs","CFIs")
}

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",
                     "X0",
                     "Xs",
                     "DFIs",
                     "CFIs",
                     "a",
                     "b",
                     "c",
                     "Xlast",
```

```
                      "P.runs.trst",
                      "DPL",
                      "FuncType"
  )

  #Add one column about the slope of DFI to the function
  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 Animal ID for each animal and save them to dataframes in the loop
  #-------------------------------------------------------------------------------
  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).
- **X0**, **Xs**, **DFIs**, **CFIs**, **a**, **b**, **c** are the origin and reparametrized parameters estimated.
- **Xlast**: The last day of age.
- **P.runs.trst**: The P-value of auto-correlation test (runs test).
- **DPL**: The data points left in each step.
- **Slope**: slope of the function (possitive when it = 1 and negative when = -1)

```
##   ANIMAL_ID       X0  Xs     DFIs     CFIs         a         b
## 2      5596 64.78902 209 1.917971 224.7833 -67.25417 0.8766461
## 3      5596 64.57491 209 1.664576 228.2906 -94.23315 1.4217750
## 4      5596 63.25414 209 1.748924 235.2599 -89.88121 1.3624740
## 5      5596 61.71493 209 1.861073 238.2424 -78.50205 1.1699743
##              c Xlast  P.runs.trst DPL FuncType Slope
## 2 0.0024912069   210 1.365028e-30 143      QLM     1
## 3 0.0005808645   210 2.409260e-16  78      QLM     1
## 4 0.0009245224   210 2.178170e-07  44      QLM     1
## 5 0.0016533473   210 2.799182e-02  21      QLM     1
```

Estimated parameters are obtained:

```
##   ANIMAL_ID       X0  Xs     DFIs     CFIs         a        b           c
## 5      5596 61.71493 209 1.861073 238.2424 -78.50205 1.169974 0.001653347
##   Xlast P.runs.trst DPL FuncType Slope
```

```
## 5   210  0.02799182  21      QLM     1
```

When applying a linear-plateau function for DFI, we encountered two possible problems:

- Firstly, the linear segment sometimes had a very modest negative slope, which would mean that the DFI decreased with increasing age. In that case, a constant value for DFI was assumed, resulting in a linearly increasing function for CFI.
- Secondly, Xs was sometimes estimated to be very close to either the first or the last observation and the linear-plateau curve would resemble a linear curve. A linear function was then used to describe DFI, resulting in a quadratic function for CFI.

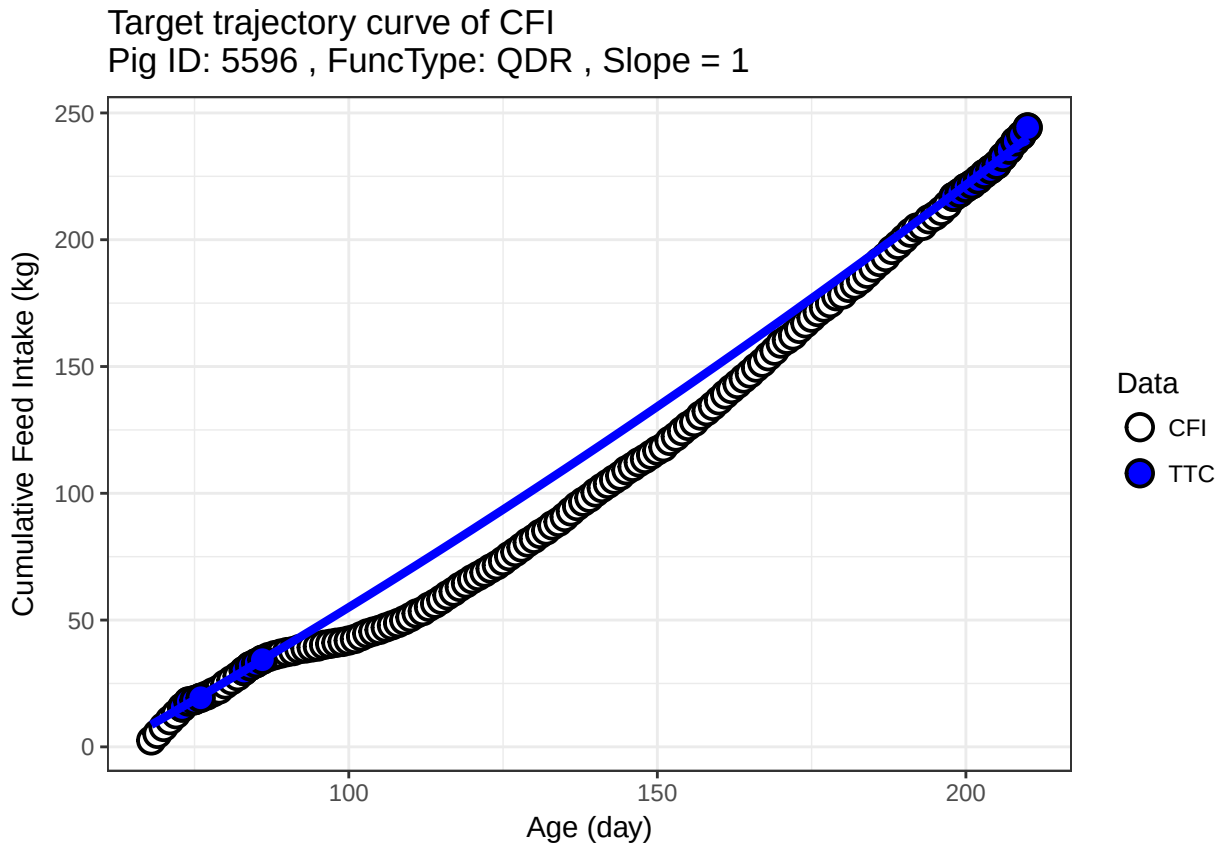
a. A quadratic function is replaced the quadratic-linear function

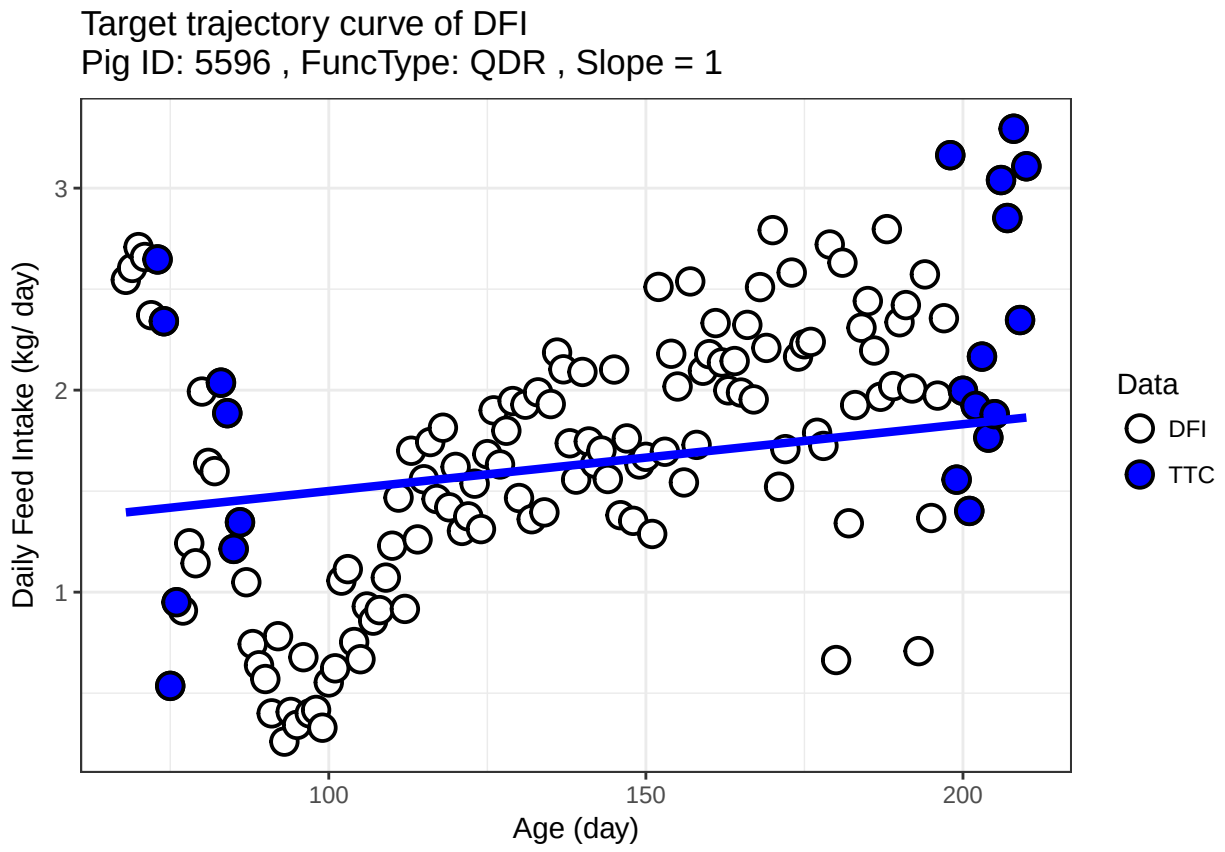
The R-code is similar to 2.1; thus, we will not display here.

b. A linear function is replaced the quadratic-linear function

The R-code is similar to 2.1; thus, we will not display here.

Plot the target trajectory curve:





Step 3: Identify perturbations

Once the target trajectory curve of CFI is identified, deviations between this curve and the actual CFI data can be used to quantify perturbations. However, not all deviations are indicative of perturbations. In fact, small (natural) oscillations in the feed intake patterns seem to exist and we only aim to quantify the most important deviations that are the result of perturbations. The purposes of this step are to define how many perturbations are in the growing period of the pig, approximate start and end days and magnitude of each perturbation. B-spline functions in Functional Data Analysis are chosen to identify the perturbation periods.

3.1. Calculate the difference between actual CFI target CFI

The differences between CFI and TTC are presented as percentage (%)

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

3.2. Set conditions for B-spline functions

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

- B-spline is a set of polynomial functions which joined end-to-end with each other at interval boundaries called knots. **Number of knots** was chosen as the length of growing period
- The **order of these polynomials** (one larger than the degree of polynomials) was given the value of 6 (therefore the degree of polynomial was 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 which controls the smoothness of the curve by penalizing the curvature was given the value of 0.01

```
#================================================================
# Fit a curve to the diferences between CFI and TTC (percent) - use of B spline function
#================================================================

 #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 #round(length(Age)/3)    # which knots? =1 all points, =2, every 2...
   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)
 #nbasis = norder + (number of interior knots) - 2  nbasis = 62,
   basisobj <- create.bspline.basis(rangeval , norder=norderT , breaks=Age[idx])
   fdPar <- fdPar(basisobj, pen_what, lambda)
```

3.3. Choose a value of time interval

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 TTC and spline functions :
 #time interval can be changed by = 0.1, 0.01 , ...
    dexima.i <- .1 # interval number
    eval_day <- seq(min(Age), max(Age),by=dexima.i) #days for FI estimation
```

3.4. Fit B-spline function for percentage of differences between CFI and TTC

```
# Fit B Spline to amount of differences between TTC and CFI (percentage)
   difCFI.fd <- smooth.basis(Age, percent, fdPar)
   Smooth.difCFI  <- difCFI.fd$fd
```

3.5. Determine the values of that smoothing function and its first derivative

Determine the values of that smoothing function and its first derivative. Save the series of days with chosen time interval, smoothing values and their first derivative into one dataframe. This dataframe is used to detect perturbations.

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

#Evaluation of B-spline for the difference between TTC and CFI (percentage)
    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 CFI and TTC
    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 TTC
#-------------------------------------------------------------------------------

 #For the difference
   dif = as.data.frame(cbind(eval_day, dif.CFI, dif.DCFI))
   names(dif) <- c("eval_day","dif.CFI", "dif.DCFI")
     head(dif)
```

```
##   eval_day   dif.CFI dif.DCFI
## 1     68.0 -70.77459 23.86018
## 2     68.1 -68.40658 23.49817
## 3     68.2 -66.07533 23.12490
## 4     68.3 -63.78198 22.74035
## 5     68.4 -61.52764 22.34453
## 6     68.5 -59.31344 21.93752
```

3.6. Detect deviations of CFI from TTC

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

- Start time of each perturbation
- End time of each perturbation
- Its magnitude

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

#Criteria for a normal range from which everything below is considered as deviations
 crit1 = 0
#we set this criteria is 0,
#so whenever actual CFI decreases from TTC we consider as deviations

#Days belong to normal range
```

```
 no.pert.age.val  = eval_day[which(dif.CFI >= crit1)]
#Determine when res 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 "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 TTC 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
#therefore maxmin = -2 is min,
#
#When the 1st derivative changes from pos to neg, 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
```

3.7. Examining data from the first week

Because the begining time is mixing period, pigs encounter many stressors which decrease feed intake. In the begining only a small reduction in FI can lead to huge difference in %, so we removed the first period.

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 were 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)
}

#Define the days in which actual 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
```

3.8. Determine the duration of each perturbation

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

  # perturbed days when res 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]

  # perturbed days when res have maximum 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 TTC  in the first day of dataset,
#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  75.2 -0.321914197  83.2 0.01312561        8.0
## 2  86.0 -0.009909733 206.2 0.03443202      120.2
```

3.9. Identify perturbations from deviations

Duration

```
#min number of days for a pert
    crit3 = 1
#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)
```

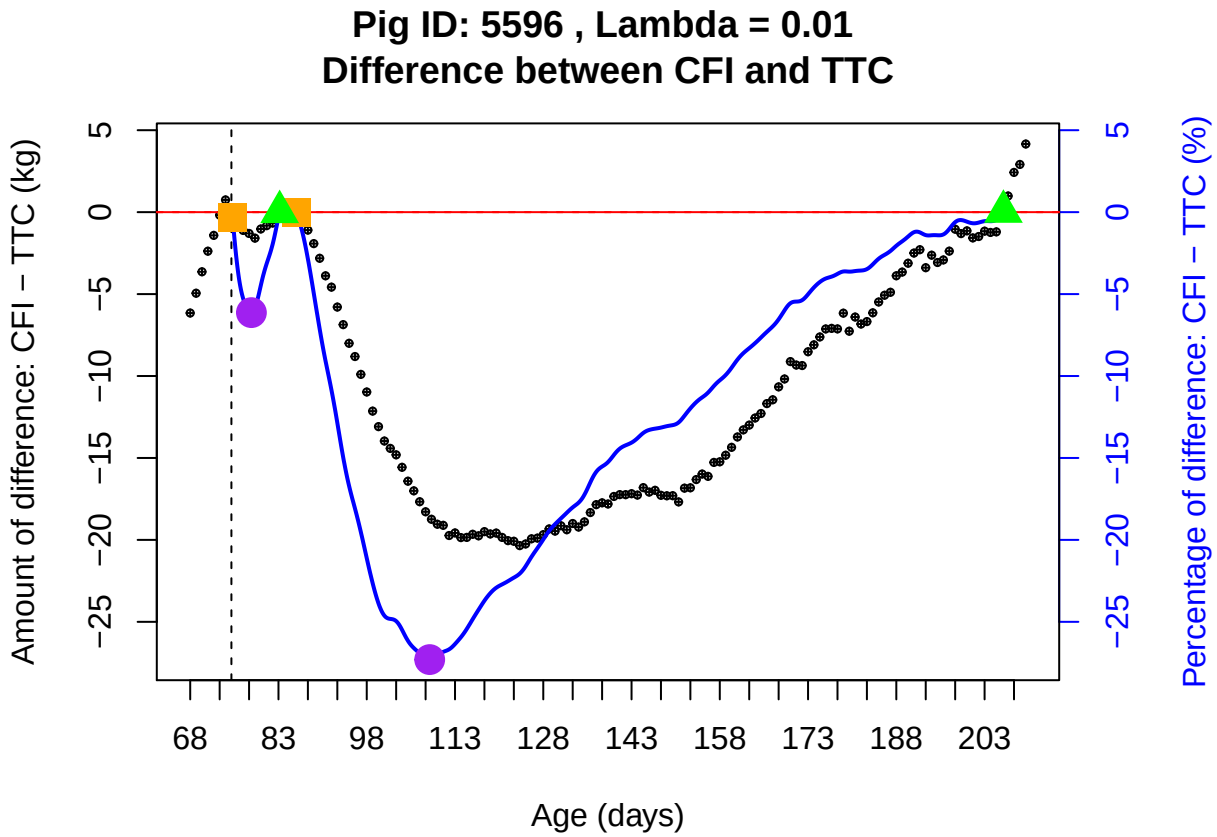
3.10. Label each perturbation

```
#-------------------------------------------------------------------------------
# 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 = "")
 }
}

#Final check the table
 pertub.table
```

```
##   Per Start  value.start   End  value.end Devia.days Min.Day   Min.perc
## 1  P1  75.2 -0.321914197  83.2 0.01312561        8.0    78.4  -6.132095
## 2  P2  86.0 -0.009909733 206.2 0.03443202      120.2   108.7 -27.303450
```

3.11. Plot the perturbation



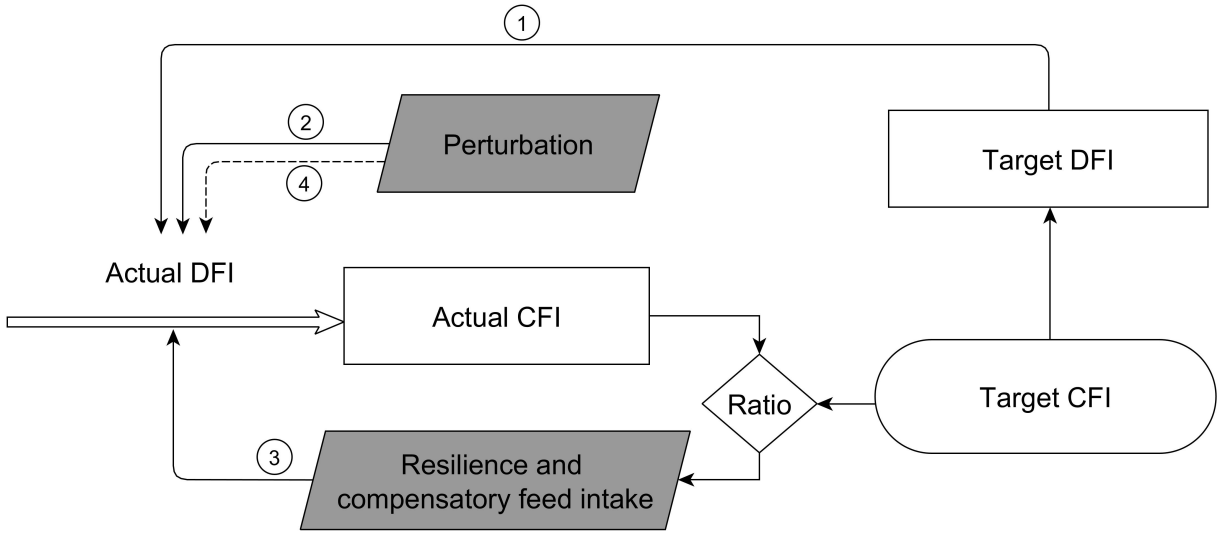
Note: We detected two perturbations from the feed intake data of this animal; nevertheless, for the purpose of illustrating the model's concept, we only describe here the response of the animal against the second perturbation.

Step 4: Modelling the response to a perturbation

Once the perturbed period is identified, a single perturbation was characterized by a system of differential equations.

Model's mechanism is described very simply in the figure below. Daily feed intake of the pig is the factor to be modelled. It encompasses three elements: **desired feed intake**, **impact of perturbation** and **its own adaptation capacity**:

- In the absence of a perturbation, the actual DFI is equal to the target DFI (1).
- The initiation of a perturbation has a negative and constant effect on DFI and because of the reduction in DFI, the actual CFI curve starts to deviate from the target CFI (2).
- The ratio between the target CFI and the actual CFI triggers the pig's resilience mechanism in a proportional way, the greater the ratio between the target and actual CFI, the greater will be the resilience mechanism for DFI (3).
- Once the perturbing factor is over, its effect on DFI disappears but the ratio will still be greater than one. This results in compensatory DFI where the actual DFI will be greater than the target DFI feed intake which, in turn, results in that the actual CFI will approach the target CFI (4).



Our model is conceptualized in a way that the impact of a single perturbation to the feed intake of the pig and its response can be characterized by four parameters:

- **tbeg1t**: the start time of perturbing factor (day of age). It is called **Start** in the deliverable.
- **tstop1**: the end time of perturbing factor (*day of age*). It is called **End** in the deliverable.
- **p1**: Percentage of DFI left under the period of perturbation (*%*). It is opposite to the value **Instantaneous reduction in DFI** in the deliverable. The **Instantaneous reduction in DFI** can then be calculated as **1 - p1**
- **max.compFI1**: the capacity of the animal to adapt to the perturbation through resistant and compensatory feed intake. It is called **Resilience parameter k** in the deliverable.

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

```r
#Load needed functions
source("Step4_functions.R")
source("abcd.R") #functions
options(digits=3)

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

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

#----------------------------
# Extract data for animal i
#----------------------------

CFI.obs  = Data$CFI.plot
DFI.obs = Data$DFI.plot
Age = Data$Age.plot

#Difference between actual 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

 if(FuncType =="LM"){
   print("A Linear function was used to identify the TTC of CFI")
 } else if(FuncType =="QDR"){
   print("A Quadratic function was used to identify the TTC of CFI")
 } else{
   print("A Quadratic-linear function was used to identify the TTC of CFI")
 }
```

```
## [1] "A Quadratic function was used to identify the TTC of CFI"
```

Calculate magnitude (the position when the actual CFI reduced maximum from target CFI) of the perturbation:

```r
#Magnitude of the perturbation
 # magnitude = res.data %>% filter(Age >= pertub.table$Start & Age <= pertub.table$End)
 magnitude   = res.data %>% filter(Age >= pertub.table$Start[2] &
                                     Age <= pertub.table$End[2])
 magnitude = magnitude %>% filter(res == min(magnitude$res))
```

Extract suitable function for TTC (depends on the type of function we fit to CFI):

```r
#-------------------------------------------------------------------------------
# Calulate TTC using abcd 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]]
  }
```

Modelling the response of animal to a single perturbation

4.1. TTC is defined by a linear function

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

  ##------------------------------
  ## initial values of parameters
  ##------------------------------
  tbeg1       = pertub.table$Start # tbeg1 corresponds to the start point
  tstop1      = magnitude$Age
  p1          = 0.18
  #because we modified negative impact of perturbation as (p1 -1)
  #the smaller p1 is the more severe impact is
  max.compFI1 = 4
  a           = TTC.param$a
  b           = TTC.param$b

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

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

  param = c( p1, max.compFI1, tbeg1, tstop1, a, b)

  ##------------------------------
  ## solve the model
  ##------------------------------

  yout <- ode(y= yinit,
              times = times.ode,
              func = ODE.CFI.0,
              parms = param,
              atol = 1e-10,
              rtol = 1e-10)
  summary(yout)
```

```
# ##------------------------------------------------
# ## plot when fitting initial parameters to the model
# ##------------------------------------------------
#
# Time = times.ode
# onoff = ifelse(Time>tbeg1 & Time<tstop1,1,0)
# ITC.sim <- pred.abcd.0(param.i, Time)[[1]]
#
# #Difference between actual and target CFI (simulation)
# plot(yout[ , 1], yout[ ,2] -(ITC.sim),
#      main = "When initial parameters were fitted to data",
#      xlab = "Age (days)", type = "l", ylab = "CFI - TTC(kg)",
#      ylim = c(min(CFI.obs - ITC, yout[ ,2] -(ITC.sim)),
#               max(CFI.obs - ITC, yout[ ,2] -(ITC.sim))),
#      lwd = 2,
#      cex.main = 1.5, cex.axis = 1.5, cex.lab = 1.5)
# points ( Age,  CFI.obs - ITC, type = "p", col = "red")
# abline(v=tbeg1, lty=2)
# abline(v=tstop1, lty = 2)
# abline(0,0, col = "red")
# par(mfrow = c(1,1))
# # dev.off()

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

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

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

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

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

##-----------------------------
## solve the model
##-----------------------------

yout <- ode(y= yinit,
            times = times.ode,
            func = ODE.CFI.optim.0,
            parms = par.init,
            atol = 1e-10,
            rtol = 1e-10)

summary(yout)


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

```
  #  run the optimization with NLS2
  ##------------------------------
  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{}
```

```
## NULL
```

4.2. TTC is defined by a quadratic function

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

 ##------------------------------
 ## initial values and times
 ##------------------------------
  tbeg1       = pertub.table$Start # tbeg1 corresponds to the start point
  tstop1      = magnitude$Age
  p1          = 0.2
  #because we modified negative impact of perturbation as (p1 -1)
  #the smaller p1 is the more severe impact is
  max.compFI1 = 2
  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)

 ##------------------------------
 ## parameters
 ##------------------------------
 param = c( p1, max.compFI1, tbeg1, tstop1, a, b, c)

 ##------------------------------
 ## solve the model
 ##------------------------------

 yout <- ode(y= yinit,
             times = times.ode,
             func = ODE.CFI.1,
```

```
            parms = param,
            atol = 1e-10,
            rtol = 1e-10)
summary(yout)

# ##------------------------------------------------
# ## plot when fitting initial parameters to the model
# ##------------------------------------------------
#
# Time = times.ode
# onoff = ifelse(Time>tbeg1 & Time<tstop1,1,0)
# ITC.sim <- pred.abcd.1(param.i, Time)[[1]]
#
# #Difference between actual and target CFI (simulation)
# plot(yout[ , 1], yout[ ,2] -(ITC.sim),
#      xlab = "Age (days)", type = "l", ylab = "CFI - TTC(kg)",
#      ylim = c(min(CFI.obs - ITC, yout[ ,2] -(ITC.sim)),
#               max(CFI.obs - ITC, yout[ ,2] -(ITC.sim))),
#      lwd = 2,
#      cex.main = 1.5, cex.axis = 1.5, cex.lab = 1.5)
# points ( Age,  CFI.obs - ITC, type = "p", col = "red")
# abline(v=tbeg1, lty=2)
# abline(v=tstop1, lty = 2)
# abline(0,0, col = "red")
# par(mfrow = c(1,1))
# # dev.off()

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

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

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

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

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

##-----------------------------
## solve the model
##-----------------------------

 yout <- ode(y= yinit,
            times = times.ode,
            func = ODE.CFI.optim.1,
            parms = par.init,
            atol = 1e-10,
            rtol = 1e-10)
summary(yout)
```

```
 ##-----------------------------
 #  run the optimization with NLS2
 ##-----------------------------
 Data.xy = Data
 times = Data.xy$Age.plot
 RSQ1 <- 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])

 RSQ2 <- ODE.CFI.obj.1(P.optim, Data.xy)

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

 print(paste("Residual sum of square before optimization:", round(RSQ1, digits = 2)))
 print(paste("Residual sum of square after optimization:", round(RSQ2, digits = 2)))

} else{}
```

```
## [1] "Residual sum of square before optimization: 142.42"
## [1] "Residual sum of square after optimization: 32.12"
```

4.3. TTC is defined by a quadratic-linear function

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

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

  tbeg1       = pertub.table$Start # tbeg1 corresponds to the start point
  tstop1      = magnitude$Age
  p1          = 0.08
  #because we modified negative impact of perturbation as (p1 -1)
  #the smaller p1 is the more severe impact is
  max.compFI1 = 9
  a           = TTC.param$a
  b           = TTC.param$b
  c           = TTC.param$c
  Xs          = TTC.param$Xs

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

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

 param = c( p1, max.compFI1, tbeg1, tstop1, a, b, c, Xs)
```

```
##-----------------------------
## solve the model
##-----------------------------

 yout <- ode(y= yinit,
             times = times.ode,
             func = ODE.CFI.2,
             parms = param,
             atol = 1e-10,
             rtol = 1e-10)
 summary(yout)


 # ##-----------------------------------------------------
 # ## plot when fitting initial parameters to the model
 # ##-----------------------------------------------------
 #
 # Time = times.ode
 # onoff = ifelse(Time>tbeg1 & Time<tstop1,1,0)
 # ITC.sim <- pred.abcd.2(param.i, Time)[[1]]
 #
 # par(mar=c(4.5,4.5,4.5,1.5))
 #
 # #Difference between actual and target CFI (simulation)
 #   plot(yout[ , 1], yout[ ,2] -(ITC.sim),
 #        xlab = "Age (days)", type = "l", ylab = "CFI - TTC(kg)",
 #        ylim = c(min(CFI.obs - ITC, yout[ ,2] -(ITC.sim)),
 #                 max(CFI.obs - ITC, yout[ ,2] -(ITC.sim))),
 #        lwd = 2,
 #        cex.main = 1.5, cex.axis = 1.5, cex.lab = 1.5)
 #   points ( Age,  CFI.obs -( ITC), type = "p", col = "red")
 #   abline(v=tbeg1, lty=2)
 #   abline(v=tstop1, lty = 2)
 #   abline(0,0, col = "red")
 #   par(mfrow = c(1,1))
 #   # dev.off()

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

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

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

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

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

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

```
 ## solve the model
 ##-----------------------------

 yout <- ode(y= yinit,
             times = times.ode,
             func = ODE.CFI.optim.2,
             parms = par.init,
             atol = 1e-10,
             rtol = 1e-10)

 summary(yout)


  ##-----------------------------
  #  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.CFI.optim.2, P.optim)

} else{}
```

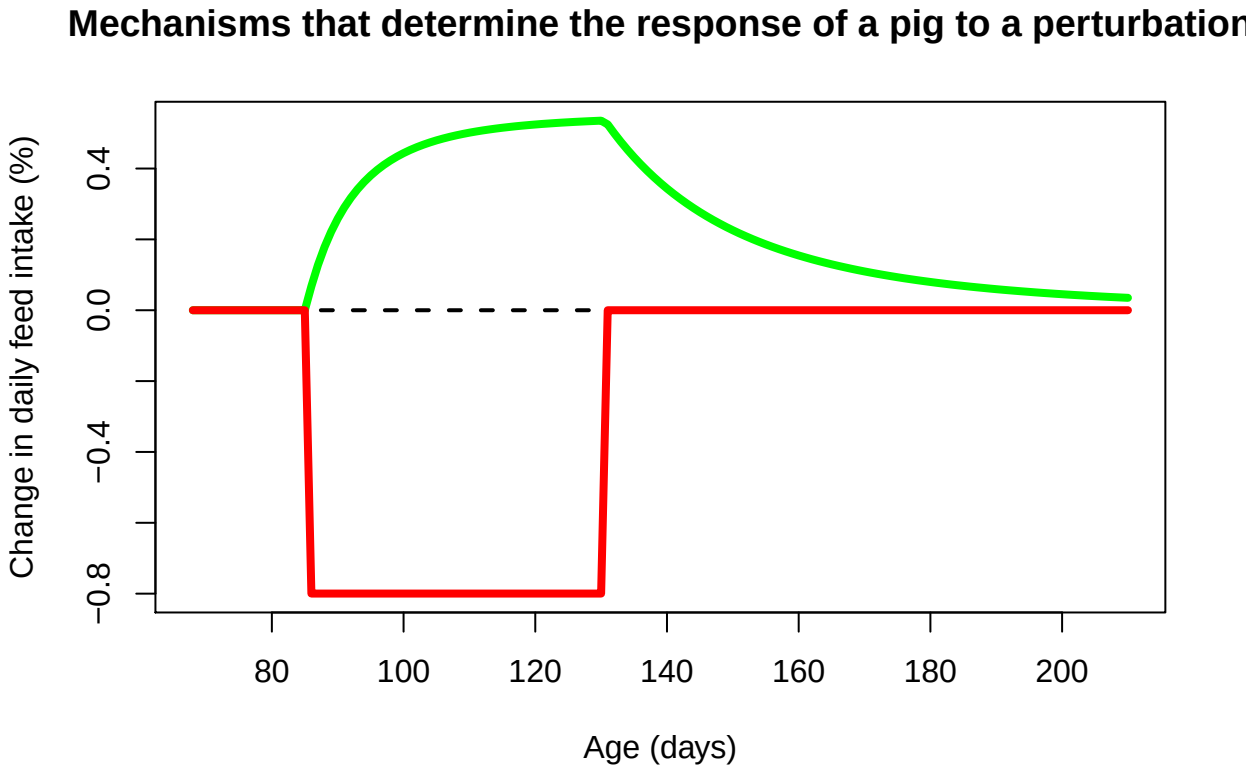
```
## NULL
```

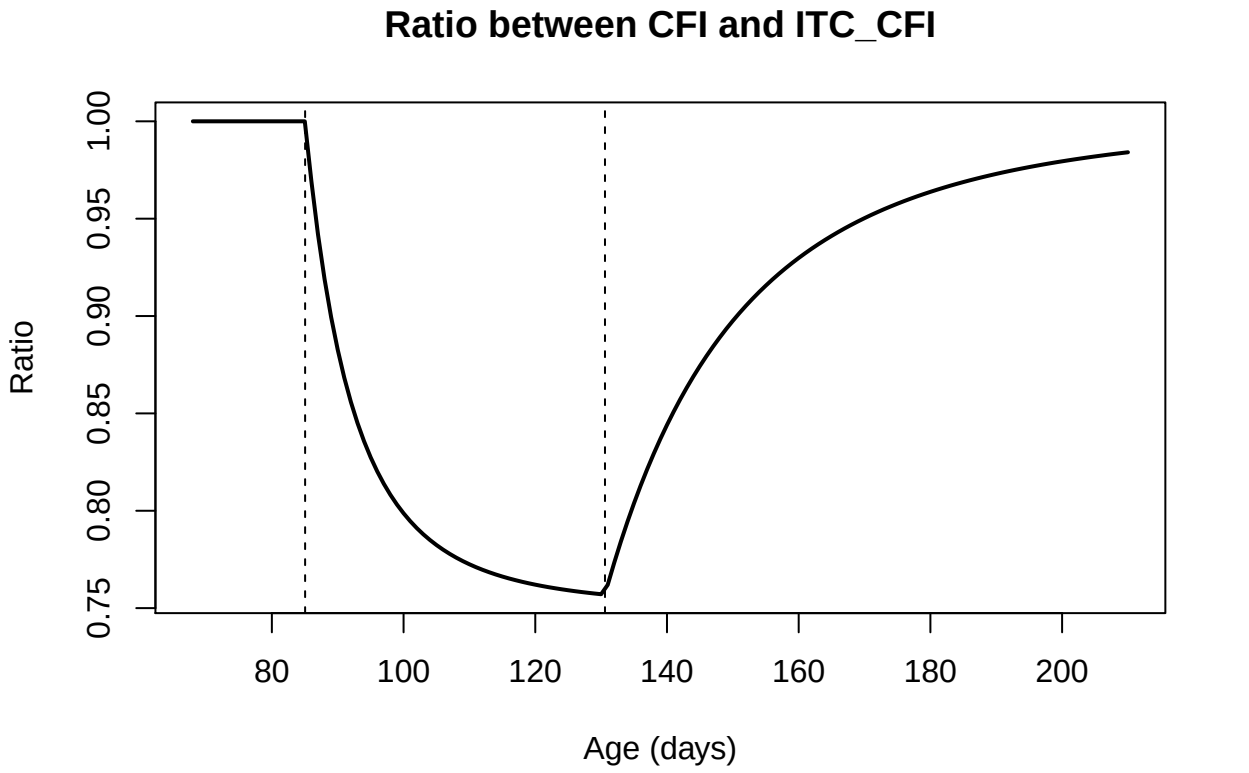
Estimated paramters of the model

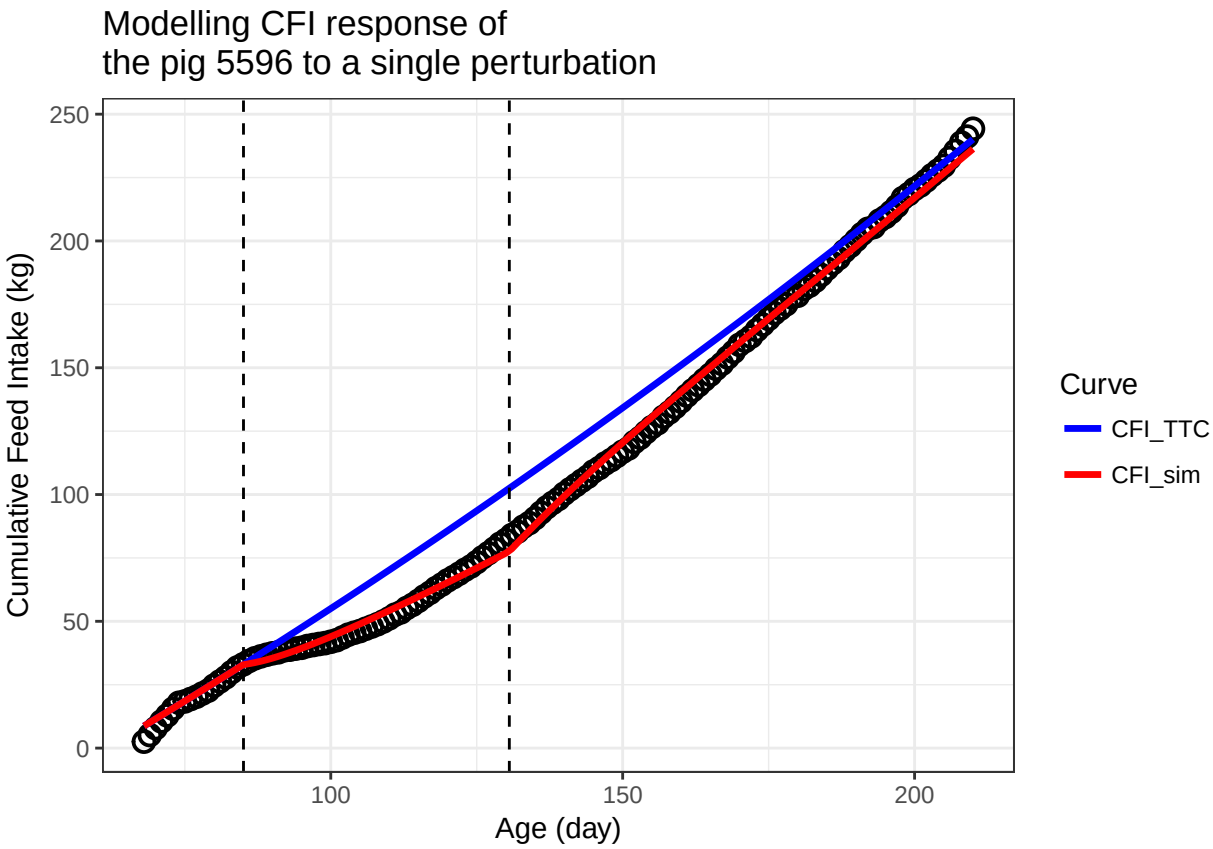
```
name.param <- c("p1", "max.compFI1", "tbeg1", "tstop1")
Explanation <- c("% DFI left under impact of perturbation",
             "Adaptation capacity",
             "Start of perturbation",
             "End of perturbation")
Table.per <- data.frame(cbind(name.param, round(P.optim, digits = 3), Explanation))
names(Table.per) <- c("Parameters", "Values", "Explanation"); Table.per
```

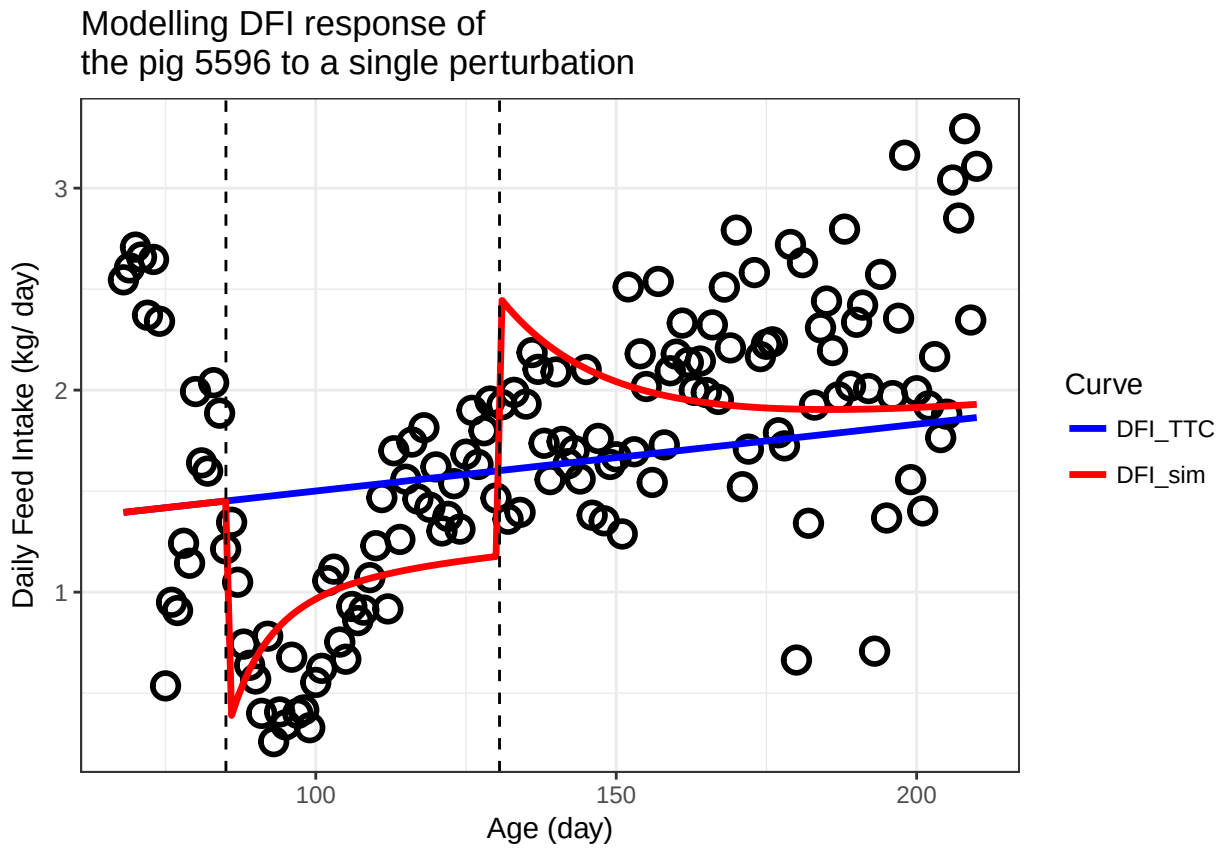
```
##    Parameters  Values                             Explanation
## 1          p1     0.2 % DFI left under impact of perturbation
## 2 max.compFI1   2.203                       Adaptation capacity
## 3       tbeg1   85.05                    Start of perturbation
## 4      tstop1 130.584                      End of perturbation
```

4.4. Plot the results






Modelling CFI response of
the pig 5596 to a single perturbation
Cumulative Feed Intake (kg)
250
200
150
100
50
0
100
150
200
Age (day)
Curve
CFI_TTC
CFI_sim



Appendix 1: Reparametrization for the TTC function in step 2

The set of these functions is saved in file ***abcd.R***

1. Linear function

```
#-------------------------------------------------------------
# Reparametrize a, b by X0 and ylast
#-------------------------------------------------------------

abcd.0 <- function(P){
  X0 = P[1]
  ylast = P[2]
  a = (ylast*X0)/(X0 - ylast)
  b = -(ylast)/(X0-xlast)


  pp = as.vector(c(a,b))
  return(pp)
}

#-------------------------------------------------------------
# NLS function
```

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

nls.func.0 <- function(X0, ylast){
  pp = c(X0, ylast)
  #calculation of a and b using these new parameters

  b = abcd.0(pp)[2]
  a =abcd.0(pp)[1]

  return (a+b*x)
}

#-----------------------------------------------------------------
# Fit new parameters to a linear function of CFI
#-----------------------------------------------------------------

pred.func.0 <- function(pr,age){
  #
  X0 = pr[1]
  ylast = P[2]

  #
  x = age
  #calculation of a and b using these new parameters
  b = abcd.0(pr)[2]
  a =abcd.0(pr)[1]

  #
  results = list()
  cfi = a+b*x
  dfi = b
  results[[1]]=cfi
  results[[2]]=dfi
  return (results)
}

#-------------------------------------------------------------------
# Linear function of CFI curve and its 1st derivative (DFI)
# with original parameters (only a and b)
#-------------------------------------------------------------------

pred.abcd.0 <- function(pr,age){
  #
  a = pr[1]
  b = pr[2]

  x = age

  results = list()
  cfi = a+b*x
  dfi = b
  results[[1]]=cfi
  results[[2]]=dfi
```

```
  return (results)
}
```

2. Quadratic function

```
#-----------------------------------------------------------------
# Reparametrize a, b and c by X0, y2 and ylast
#-----------------------------------------------------------------

abcd.1 <- function(P){
  X0 = P[1]
  y2 = P[2]
  ylast = P[3]

  a = (xlast*ylast-4*y2*xlast+X0*ylast)*X0/(xlast^2-2*X0*xlast+X0^2)
  b = -(xlast*ylast-4*y2*xlast+3*X0*ylast-4*y2*X0)/((xlast-X0)^2)
  c = 2*(-2*y2+ylast)/(xlast^2-2*X0*xlast+X0^2)

  pp = as.vector(c(a,b,c))
  return(pp)
}

#-----------------------------------------------------------------
# NLS function
#-----------------------------------------------------------------

nls.func.1 <- function(X0, y2, ylast){
  pp = c(X0, y2, ylast)
  #calculation of a,b and c using these new parameters

  c = abcd.1(pp)[3]
  b = abcd.1(pp)[2]
  a =abcd.1(pp)[1]

  return (a+b*x+c*x^2)
}

#-----------------------------------------------------------------
# Fit new parameters to a quadratic function of CFI
#-----------------------------------------------------------------

pred.func.1 <- function(pr,age){
  #
  X0 = pr[1]
  y2 = P[2]
  ylast = P[3]
  #
  x = age

  c = abcd.1(pr)[3]
  b = abcd.1(pr)[2]
  a =abcd.1(pr)[1]
```

```
  #
  results = list()
  cfi = a+b*x+c*x^2
  dfi = b+2*c*x
  results[[1]]=cfi
  results[[2]]=dfi
  return (results)
}

#-----------------------------------------------------------------
# Quadratic function of CFI curve and its 1st derivative (DFI)
# with original parameters (only a, b and c)
#-----------------------------------------------------------------

pred.abcd.1 <- function(pr,age){
  #
  a = pr[1]
  b = pr[2]
  c = pr[3]

  x = age

  results = list()
  cfi = a+b*x+c*x^2
  dfi = b+2*c*x
  results[[1]]=cfi
  results[[2]]=dfi
  return (results)
}
```

3. Quadratic-linear function

```
#-----------------------------------------------------------------
# Reparametrize a, b and c by X0, Xs (breaking point),
# DFIs (DFI at Xs) and CFIs (CFI at Xs)
#-----------------------------------------------------------------

abcd.2 <- function(P){
  X0 = P[1]
  Xs = P[2]
  DFIs = P[3]
  CFIs = P[4]

  a = -X0*(2*CFIs*Xs-CFIs*X0-Xs^2*DFIs+Xs*DFIs*X0)/(Xs^2-2*X0*Xs+X0^2)
  b = (-Xs^2*DFIs+DFIs*X0^2+2*CFIs*Xs)/(Xs^2-2*X0*Xs+X0^2)
  c = -(CFIs-Xs*DFIs+X0*DFIs)/(Xs^2-2*X0*Xs+X0^2)

  pp = as.vector(c(a,b,c))
  return(pp)
}
```

```r
#-------------------------------------------------------------
# NLS function
#-------------------------------------------------------------

nls.func.2 <- function(X0, Xs, DFIs, CFIs){
  pp = c(X0, Xs, DFIs, CFIs)

  #-------------------------------------------------------
  # Function to reparameter a, b and c  of quadratic-linear function:
  #-------------------------------------------------------
  c = abcd.2(pp)[3]
  b = abcd.2(pp)[2]
  a =abcd.2(pp)[1]

  ind1 <- as.numeric(x < Xs)
  return (ind1*(a+b*x+c*x^2)+(1-ind1)*((a+b*(Xs)+c*(Xs)^2)+(b+2*c*(Xs))*(x-(Xs))))
}

#-------------------------------------------------------------
# Fit new parameters to a quadratic-linear function of CFI
#-------------------------------------------------------------

pred.func.2 <- function(pr,age){
  #
  X0 = pr[1]
  Xs = pr[2]
  DFIs = pr[3]
  CFIs = pr[4]
  #
  x = age
  #calculation of a,b and c using these new parameters
  c = abcd.2(pr)[3]
  b = abcd.2(pr)[2]
  a = abcd.2(pr)[1]

  #
  ind1 <- as.numeric(x < Xs)
  #
  results = list()
  cfi = ind1*(a+b*x+c*x^2)+(1-ind1)*((a+b*(Xs)+c*(Xs)^2)+(b+2*c*(Xs))*(x-(Xs)))
  dfi = ind1*(b+2*c*x) + (1 - ind1)*(b+2*c*(Xs))
  results[[1]]=cfi
  results[[2]]=dfi
  return (results)
}

#-------------------------------------------------------------------
# Quadratic-linear function of CFI curve and its 1st
# derivative (DFI) with original parameters (only a, b and c)
#-------------------------------------------------------------------

pred.abcd.2 <- function(pr,age){
  #
```

```
  a = pr[1]
  b = pr[2]
  c = pr[3]

  x = age
  #
  ind1 <- as.numeric(x < Xs)
  #
  results = list()
  cfi = ind1*(a+b*x+c*x^2)+(1-ind1)*((a+b*(Xs)+c*(Xs)^2)+(b+2*c*(Xs))*(x-(Xs)))
  dfi = ind1*(b+2*c*x) + (1 - ind1)*(b+2*c*(Xs))
  results[[1]]=cfi
  results[[2]]=dfi
  return (results)
}
```

Appendix 2: Functions of differential equations in step 4

The set of these functions is saved in file ***Step4__functions.R***

1. Linear function

```
#------------------------------
# Linear function for TTC
#------------------------------

#Fitting initial parameters to the model
  ODE.CFI.0 <- function(Time, State, Pars) {
  #
  p1          = Pars[1]
  max.compFI1 = Pars[2]
  tbeg1       = Pars[3]
  tstop1      = Pars[4]
  a           = Pars[5]
  b           = Pars[6]


  #
  with(as.list(c(State, Pars)), {
    #
    x = Time
    #
    ITC.CFI = a +b*x
    ITC.DFI = b
    #
    onoff   = ifelse(Time>tbeg1 & Time<tstop1,1,0)
    CompFI  = (1-CumFI/ITC.CFI)*max.compFI1
    #
    dCumFI  =   (onoff*(p1-1) + CompFI + 1)*ITC.DFI
    #
```

```
    return(list(c(dCumFI)))
  })
}

#Function to estimate parameters
ODE.CFI.optim.0 <- function(Time, State, Pars) {
  #
  p1          = Pars[1]
  max.compFI1 = Pars[2]
  tbeg1       = Pars[3]
  tstop1      = Pars[4]
  # a           = Pars[5]
  # b           = Pars[6]
  # c           = Pars[7]
  # d           = Pars[8]

  #
  with(as.list(c(State, Pars)), {
    #
    x = Time
    #
    ITC.CFI = a +b*x
    ITC.DFI = b
    #
    onoff = ifelse(Time>tbeg1 & Time<tstop1,1,0)
    CompFI = (1-CumFI/ITC.CFI)*max.compFI1
    #
    dCumFI  =   (onoff*(p1-1) + CompFI + 1)*ITC.DFI
    #
    return(list(c(dCumFI)))
  })
}

#Objective function
ODE.CFI.obj.0 <- function(P, data){
  data = Data.xy
  f <- ode(yinit, times, ODE.CFI.optim.0, P)
  optim.f = sqrt(sum((data$CFI.plot- f[,2])^2))
  return (optim.f)
}
```

2. Quadratic function

```
#------------------------------
# Quadratic function for TTC
#------------------------------

#Fitting initial parameters to the model
ODE.CFI.1 <- function(Time, State, Pars) {
  #
  p1          = Pars[1]
  max.compFI1 = Pars[2]
```

```
  tbeg1       = Pars[3]
  tstop1      = Pars[4]
  a           = Pars[5]
  b           = Pars[6]
  c           = Pars[7]

  #
  with(as.list(c(State, Pars)), {
    #
    x = Time
    #
    ITC.CFI = a + b*x + c*x^2
    ITC.DFI = b+2*c*x
    #
    onoff   = ifelse(Time>tbeg1 & Time<tstop1,1,0)
    CompFI  = (1-CumFI/ITC.CFI)*max.compFI1
    #
    dCumFI  =   (onoff*(p1-1) + CompFI + 1)*ITC.DFI
    #
    return(list(c(dCumFI)))
  })
}

#Function to estimate parameters
ODE.CFI.optim.1 <- function(Time, State, Pars) {
  #
  p1          = Pars[1]
  max.compFI1 = Pars[2]
  tbeg1       = Pars[3]
  tstop1      = Pars[4]
  # a           = Pars[5]
  # b           = Pars[6]
  # c           = Pars[7]
  # d           = Pars[8]

  #
  with(as.list(c(State, Pars)), {
    #
    x = Time
    #
    ITC.CFI = a + b*x + c*x^2
    ITC.DFI = b+2*c*x
    #
    onoff = ifelse(Time>tbeg1 & Time<tstop1,1,0)
    CompFI = (1-CumFI/ITC.CFI)*max.compFI1
    #
    dCumFI  =   (onoff*(p1-1) + CompFI + 1)*ITC.DFI
    #
    return(list(c(dCumFI)))
  })
}

#Objective function
```

```
ODE.CFI.obj.1 <- function(P, data){
  data = Data.xy
  f <- ode(yinit, times, ODE.CFI.optim.1, P)
  optim.f = sqrt(sum((data$CFI.plot- f[,2])^2))
  return (optim.f)
}
```

3. Quadratic-linear function

```
#-----------------------------------
# Quadratic-linear function for TTC
#-----------------------------------

#Fitting initial parameters to the model
 ODE.CFI.2 <- function(Time, State, Pars) {
  #
  p1          = Pars[1]
  max.compFI1 = Pars[2]
  tbeg1       = Pars[3]
  tstop1      = Pars[4]
  a           = Pars[5]
  b           = Pars[6]
  c           = Pars[7]

  #
  with(as.list(c(State, Pars)), {
  #
  x = Time
  ind1 <- as.numeric(x < Xs)
  #
  ITC.CFI = ind1*(a+b*x+c*x^2)+(1-ind1)*((a+b*(Xs)+c*(Xs)^2)+(b+2*c*(Xs))*(x-(Xs)))
  ITC.DFI = ind1*(b+2*c*x) + (1 - ind1)*(b+2*c*(Xs))
  #
  onoff   = ifelse(Time>tbeg1 & Time<tstop1,1,0)
  CompFI  = (1-CumFI/ITC.CFI)*max.compFI1
  #
  dCumFI  =   (onoff*(p1-1) + CompFI + 1)*ITC.DFI
  #
  return(list(c(dCumFI)))
  })
}

#Function to estimate parameters
ODE.CFI.optim.2 <- function(Time, State, Pars) {
   #
   p1          = Pars[1]
   max.compFI1 = Pars[2]
   tbeg1       = Pars[3]
   tstop1      = Pars[4]
   # a           = Pars[5]
   # b           = Pars[6]
   # c           = Pars[7]
```

```
   # d            = Pars[8]

   #
   with(as.list(c(State, Pars)), {
   #
   x = Time
   ind1 <- as.numeric(x < Xs)
   #
   ITC.CFI = ind1*(a+b*x+c*x^2)+(1-ind1)*((a+b*(Xs)+c*(Xs)^2)+(b+2*c*(Xs))*(x-(Xs)))
   ITC.DFI = ind1*(b+2*c*x) + (1 - ind1)*(b+2*c*(Xs))
   #
   onoff = ifelse(Time>tbeg1 & Time<tstop1,1,0)
   CompFI = (1-CumFI/ITC.CFI)*max.compFI1
   #
   dCumFI  =   (onoff*(p1-1) + CompFI + 1)*ITC.DFI
   #
   return(list(c(dCumFI)))
   })
 }

#  objective function
ODE.CFI.obj.2 <- function(P, data){
   data = Data.xy
   f <- ode(yinit, times, ODE.CFI.optim.2, P)
   optim.f = sqrt(sum((data$CFI.plot- f[,2])^2))
   return (optim.f)
 }
```