A filtering algorithm for accurate determination of feed intake dynamics

Blavy, P.[1,2], Dhumez, O.[1], Giger-Reverdin, S.[1], Nielsen, B.L.[1], Muñoz-Tamayo, R.[1], and Friggens, N.C.[1]

[1]Université Paris-Saclay, INRAE, AgroParisTech, UMR Modélisation Systémique Appliquée aux Ruminants, 75005, Paris, France.
[2]Corresponding author pierre.blavy@agroparistech.fr

September 8, 2020

1 Abstract

Background : Feed intake is a useful measurement in many types of livestock studies. This variable can be obtained by measuring changes in feed weight (`weight_m`) over time using a weighing platform. The resulting data are a mixture of stable and variable (during animal or human interactions with the platform) feed weights.
Results : This article proposes an algorithm that extract the stable parts, allowing accurate computing of feed intake. A GPL-3.0+ implementation of this algorithm in R is provided. Application is illustrated using individual data from cows, goats and mice
Conclusions : Feed intake can be computed from weighing platform data in various species using our method.

Keywords: weighing platform, feed weight, algorithm, filtering, feed intake, feed distribution.

2 Introduction

Feed intake has a primary role in studies of animal physiology and behaviour. For research, feed intake is an important input especially for describing feeding

behaviour [6], or in models that predict methane emissions in ruminants [3, 11]. Within the context of precision livestock farming, there is a great interest in developing algorithms to monitor feed intake for individual animals [2] as total ingestion mainly influences milk and growth in ruminants[9, 8]. Changes in the time course of the feed intake can be seen as an indicator of miscellaneous troubles like acidosis [7] or sub-acidosis [5].

Feed bins on weighing platforms are classical devices to measure the feed consumed by animals. However, the data provided by this type of devices are often noisy and the subsequent data treatment is time consuming. Consequently, automatic filtering methods are needed to provide accurate information on the actual feed intake. Some manufacturers propose softwares that filter out the data, but they are closed source, making them less useful in an experimental context. In this article, we present an open-source (GPL-3.0+) algorithm that facilitates the accurate determination of feed intake from feed bins temporal data. The algorithm was applied to cows, goats, and mice data sets.

3 Material and methods

3.1 Goat data set and preprocessing

The goat data set was obtained from the experimental facilities of MoSAR, UMR 0791, INRAE AgroParisTech Université Paris Saclay. In this setup, weighing platforms were coupled to automatic gates (Gabard System, France) that allowed only a single designated animal to eat at a specific gate, and that prevented feeding during the twice a day feed distribution. The weight of the feed bins (`weight_m`) was measured with a precision of 5 grams, every two seconds with an automatic weighing platform (SWR3P-BMC 301x275; Balea, France), using the manufacturer's software. The data are available in `data/goat.csv`. The data used in this paper were extracted from the 1st to the 9th January 2018 from gate one.

The original data were preprocessed by removing all points below -300g, these points corresponds to periods when the feed box was removed from the weighing platform in order to refill it.

3.2 Cow data set and preprocessing

The cow data set was obtained from the experimental facilities of Agriculture Victoria, Australia. The data analyzed here were extracted fom the study of Moate et al [10]. Distributed lucerne hay weight was measured every second. Cows were fed two times 6 kg of lucerne twice a day on mornings and afternoons. Cows were also fed some concentrate that's not measured by the weighing platforms. Data is available in `data/cow.csv`.

The original data were preprocessed by removing all points below -100g, as such points corresponds to periods when the feed box was removed. All values above 6 kg were replaced by a value of 6 kg, as there were no locking

system to prevent cows from interacting with the weighing platform during feed distributions, creating erroneous high weights records at these times.

3.3 Mouse data set and preprocessing

The mouse set was obtained from the experimental facilities of PNCA, UMR-914, INRAE by Patrick Even [4]. Mice are housed in individual metabolic cages equipped with one or several food cups fixed on a weight gauge (ref AHA 500, Phimesure, scale 0-500g, sensitivity 0.01g). The output signal of the gauges are acquired at 100 Hz by a program of data acquisition designed in the laboratory (developed with Labview, National instrument). Data are converted into mg, averaged into 2 s time bins according to the desired time resolution before storage on hard disk. The original data were preprocessed by removing all points below 0g and above 100g, as they probably correspond to humans interracting with the weighing platform.

3.4 General principle of the algorithm

The principle of the algorithm is to first distinguish periods when the animal or someone is interacting with the weighing platform (i.e., erroneous data) from periods when the weighing platform is stable (i.e., plateaus). Then feed distributions are detected, by finding the highest increase in `weight_m` (i.e., food added to the bin) within a specific interval (i.e., feed distribution schedule). Finally, as we expect the food to be always decreasing except at feed distribution, the longest decreasing sub sequence of plateaus is computed between distribution.

3.5 Detection of plateaus

A zoom of preprocessed data (see Fig 1), shows periods of stable data (i.e., plateaus) when the weighing platform is left alone, and unstable periods that are caused by the animal interacting with the weighing platform. A plateau is defined by a `begin_time`, an `end_time` and a `weight_m`.

The complete algorithm for plateau detection is described in Algorithm 1 and implemented in `code/plateau.R`. Its parameters are in Table 1. Plateaus are computed by iterating through all data points. Points that are close together in time and in `weight_m` are grouped into the same plateau (see `max_gap` and `max_delta` in Algorithm 1).

As plateaus indicate periods when the weighting platform is stable, they need to have a minimal duration to enforce stability, therefore plateaus shorter than `min_duration` are discarded (see Algorithm 1).

3.6 Detection of feed distributions

As we expect `weight_m` to be decreasing only between feed distributions, we have to cut the data into pieces at distribution time. To detect feed distributions, we first define a schedule composed of arbitrary time intervals. They

are chosen to cover the periods when the feeding occurs. Then, as we expect the feed to increase at distribution, we search for the largest increase between two consecutive plateaus within each time interval. The difference of `weight_m` between these plateau corresponds to the amount of feed added to the bin. The code for schedule generation is in `code/schedule.R`, the code for distributions detection is in `code/distribution.R`.

3.7 Longest decreasing sub-sequence of plateau

Between two consecutive feed distributions, we expect feed to be decreasing as the animal eats. To enforce it, we generalize the already existing longest increasing subsequence algorithm in order to use an arbitrary function for comparing elements. By using `a`$\geq$`b` instead of `a<b` for comparing elements, we obtain the longest not-strictly decreasing sub-sequence.

The longest not-strictly decreasing sub-sequence algorithm expects a vector of points (i.e., one `weight_m` at one `time`), but plateaus are vectors of ranges (i.e., one `weight_m`, between `begin_time` and `end_time`). To make this algorithm work, we interpolate each plateau with a small constant step to obtain a vector of points. As the number of points in the interpolation is proportional to the plateau duration, and as the `a`$\geq$`b` function used in the algorithm keeps ex-aequo, this method has the beneficial effect of weighting the longest sub-sequence algorithm by plateau durations.

In the general case, the weighting is slightly wrong if the plateau boundaries are not synchronized with the interpolation times. When the period between two consecutive measurements is constant, choosing an interpolation step equal to this period guarantees synchronization and avoids this problem. When the period between two consecutive measurements changes, choosing an interpolation step smaller than the shortest plateau duration makes the ponderation error negligible at a price of higher CPU and memory consumption, which is in practice a usable workaround

The longest increasing sub-sequence code originates from the algorithm publicly available on Wikipedia[13] and is implemented in `code/liss.R`. The modified version to construct the plateaus and perform the interpolation is in `code/plateau_decreasing.R`.

3.8 Choosing `max_delta`

The `max_delta` parameter represents the acceptable difference in `weight_m` between points in a plateau. This parameter must always be higher than the precision of the weighing platform. A rule of thumb is to consider that `max_delta` is the acceptable plateau `weight_m` error, and to choose the highest value that corresponds to a reasonable error (guess: `3*platform_precision`). In order to ensure that an appropriate `max_delta` value has been chosen, it's necessary to look at the generated plateaus on a plot (code for plotting plateaus is in `code/plateau_plot.R`). If a plateau aggregates points that vary greatly, `max_delta` is too high. If there are two or more plateaus of slightly different

weights, in areas where data are stable, `max_delta` is too low. For an example, see Fig 2 A.

Looking at the histogram of absolute differences between `weight_m` and local median of `weight_m` may also give an idea of `max_delta` value. We expect to see a large proportion of values in range $[0; platform\_precision]$ that correspond to plateaus, and higher values that correspond to parts when the animal is interacting with the scale or distributions. Code for computing the local function is in `code/local_fn.R`, an example of histograms can be seen in `code/ex_goat.R` and `code/ex_cow.R`.

3.9 Choosing `max_gap`

The `max_gap` parameter is the maximum duration between two consecutive points within a plateau. A rule of thumb is to consider that `max_gap` is the acceptable plateau duration error (guess:`3*sampling_period`). Then, always look at the generated plateau on a plot: if there are too many breaks within regions of data with the same `weight_m`, `max_gap` is too small. If the same plateau aggregate data across too large gaps, `max_gap` is too high. This latter case may happen when data is missing or removed by preprocessing. For an example, see Fig 2 B.

3.10 Choosing `min_duration`

The `min_duration` parameter is the minimal duration of a plateau. We want the smallest value that allows proper `weight_m` measurement, but that does not create fake plateaus. A reasonable place to start is to choose it to have at least 5 data points in a plateau (guess: `5*sampling_period`). Then, always look at the generated plateau on a plot. If there are too many small plateaus (typically in noisy areas), `min_duration` is too small. If stable series of short duration are missed, `min_duration` is too large. For an example, see Fig 2 C.

3.11 The median for aggregation

The current plateau detection algorithm compares each point to the median of the buffer (i.e., the current potential plateau) in order to decide whether to add it to the buffer or not. Comparison to the median was chosen for its robustness but if the `weight_m` is slowly changing (for example due to weighing platform drift), a plateau may aggregate points with very different values, which is not wanted. A less robust but drift-proof solution is to replace distance to median by the maximum distance between current point and all points in the buffer (see (2.1) in Algorithm 1).

4 Results and discussion

4.1 Reasonable computation times

The procedure (plateau detection, feed distribution detection and longest weighted not-strictly decreasing sub-sequence of plateau) was run on a single 2.10 GHz CPU core.

On goat data (326 385 points) the code (see `code/ex_goat_bench.R`) took about 3 min to run, with 80% of the time spend on computing the plateau. On cow data (80 001 points) the code (see `code/ex_cow_bench.R`) ran in 31 seconds, with 83% of the time spend on computing the plateaus. So even if the current implementation is not optimized for performance, it is fast enough to be usable in practice.

4.2 Practical use cases of the algorithm

The method used to compute feed distribution is acurate as long as there is one plateau just before the distribution, one just after it and no plateau in between. In practice, it means that all the food of a distribution must be added in a single move, and that nothing must interact with the scale before and after the distribution. This typically require a kind of locking system that prevents the animal from eating in the seconds after the food was distributed.

The algorithm that detect plateau is acurate, as long as sampling is fast enough to have many points per plateau (in practice at least a measurment every 2 seconds), and as long as the measurment errors are smaller than the minimum amount an animal eats in a single visit.

4.3 A generic algorithm

The different pieces of this algorithm were built to be as independent as possible, so that they can be recycled to handle different kinds of data. For example, the current plateau algorithm can easily be used in any system that produces time series composed of a mixture of stable relevant and unstable irrelevant data in order to filter out the unstable part, or as a destructive compression algorithm.

In practice, the full pipeline (plateau, detection of distribution, longest ponderated not-strictly decreasing sub-sequence) is an easy to use tool for preprocessing raw weighing platform data in order to compute feed consumption from weighting platform data. The examples shown in the present paper illustrate this for data from different species and different weighing platforms.

The code for analysing cow, goat and mice data is available in `code/ex_cow.R`, `code/ex_goat.R` and `code/ex_mouse.R`.

5 Conclusion

The proposed algorithm allows to compute feed intake from weighing platform data with reasonable computation times. Morever, the algorithm is generic

enough to handle data from different brand of scales and from different species.

6 Declarations

6.1 Ethics approval and consent to participate

The data used in this paper were historical data from pre-existing studies. For the goat and mouse datasets, all procedures were conducted in accordance with the French legislation on controlling experiments/procedures of live animals and the European Convention for the protection of vertebrates used for experimental purposes or for other scientific purposes (European Directive 86/609). For the cow dataset, all procedures were conducted in accordance with the Australian Code of Practice for the Care and Use of Animals for Scientific Purposes (NHMRC 2013 [1]). Animal use was approved by the Agricultural Research & Extension Animal Ethics Committee of the Department of Jobs, Precincts and Regions–Victoria.

6.2 Consent for publication

Not applicable

6.3 Availability of data and materials

The data sets, the code, and the source file for making this article are included within the article in the additional file : `supplementary.zip`.

The feed intake algorithm code can be found in the `code/` folder. It is published under the GPL-3.0+ license and runs with the R software version 3.6.3 [12].

The data sets remains the properties of their authors, they can be found in the `data/` folder. The goat data set was obtained from the experimental facilities of MoSAR, UMR 0791, INRAE AgroParisTech Université Paris Saclay. The cow data set was obtained from the experimental facilities of Agriculture Victoria, Australia by PJ. Moate et al [10]. The mouse data set was obtained from the experimental facilities of PNCA, UMR-914, INRAE by Patrick Even [4].

The main latex file used to generate this article is `pblavy.tex` and detailed instruction for compiling it are in `make-pdf.sh`. This article is published under the Creative Commons Attribution License 4.0 (CC BY 4.0).

6.4 Competing interests

The authors declare that they have no competing interests

6.5 Funding

This work was funded by INRAE, the French National Research Agency for Agriculture, Food and the Environment. INRAE had no role in the design of

the study nor in the collection, analysis, and interpretation of data and writing of the manuscript.

6.6 Authors' contributions

OD, SG and BN gave feedback on animal behaviour interpretation. OD, NF, BN and RMT contributed to data interpretation and evaluation of algorithms against raw data. PB wrote the algorithm and the code. All authors read and approved the final manuscript.

7 Acknowledgements

We thank the technical staff of the experimental farm of MoSAR, INRAE at Thiverval-Grignon for their assistance in the daily monitoring of goats.

We thank Peter J. Moate and Richard Williams (Agriculture Victoria, Ellinbank, Australia) for providing access to their data on cows ; Patrick Even (Physiologie de la Nutrition et du Comportement Alimentaire (PNCA, INRAE, AgroParisTech) for data on mice ; Pierre-Emmanuel Robert (MoSAR) for the SYRINX database containg the goat data ; and Baptiste Ruiz (AgroParisTech) for his input on data analysis.

References

[1] *Australian Code for the Care and Use of Animals for Scientific Purposes, 8th ed.* National Health and Medical Research Council: Canberra, Australia, 2013. ISBN:1864965975, Reference number:EA28.

[2] Andriamasinoro Lalaina Herinaina Andriamandroso, Frédéric Lebeau, Yves Beckers, Eric Froidmont, Isabelle Dufrasne, Bernard Heinesch, Pierre Dumortier, Guillaume Blanchy, Yannick Blaise, and Jérôme Bindelle. Development of an open-source algorithm based on inertial measurement units (imu) of a smartphone to detect cattle grass intake and ruminating behaviors. *Computers and electronics in agriculture*, 139:126–137, 2017. doi:10.1016/j.compag.2017.05.020.

[3] Jayasooriya ADRN Appuhamy, James France, and Ermias Kebreab. Models for predicting enteric methane emissions from dairy cows in North America, Europe, and Australia and New Zealand. *Global Change Biology*, 22(9):3039–3056, 2016. doi:10.1111/gcb.13339.

[4] Catherine Chaumontet, Dalila Azzout-Marniche, Anne Blais, Julien Piedcoq, Daniel Tomé, Claire Gaudichon, and Patrick C Even. Low-protein and methionine, high-starch diets increase energy intake and expenditure, increase FGF21, decrease IGF-1, and have little effect on adiposity in mice. *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, 316(5):R486–R501, 2019. doi:10.1152/ajpregu.00316.2018.

[5] Jörg Matthias Dehn Enemark, RJ Jorgensen, and Peter St Enemark. Rumen acidosis with special emphasis on diagnostic aspects of subclinical rumen acidosis: a review. *Veterinarija ir zootechnika*, 20(42):16–29, 2002.

[6] Sylvie Giger-Reverdin, Émilie Lebarbier, Christine Duvaux-Ponter, and Marion Desnoyers. A new segmentation-clustering method to analyse feeding behaviour of ruminants from withinday cumulative intake patterns. *Computers and Electronics in Agriculture*, 83:109–116, 2012. doi:10.1016/j.compag.2012.02.007.

[7] LA González, X Manteca, S Calsamiglia, KS Schwartzkopf-Genswein, and A Ferret. Ruminal acidosis in feedlot cattle: Interplay between feed ingredients, rumen function and feeding behavior (a review). *Animal Feed Science and Technology*, 172(1-2):66–79, 2012. doi:10.1016/j.anifeedsci.2011.12.009.

[8] R Jarrige et al. *Alimentation des bovins ovins et caprins*. Inra-Quae, 1989. ISBN:2-7380-0021-5, EAN:9782738000217.

[9] Siem Korver. *Feed intake and production in dairy breeds dependent on the ration*. PhD thesis, Korver, 1982.

[10] PJ Moate, SRO Williams, MH Deighton, MC Hannah, BE Ribaux, GL Morris, JL Jacobs, J Hill, and WJ Wales. Effects of feeding wheat or corn and of rumen fistulation on milk production and methane emissions of dairy cows. *Animal Production Science*, 59(5):891–905, 2019. doi:10.1071/AN17433.

[11] Rafael Munoz-Tamayo, JF Ramírez Agudelo, Richard J Dewhurst, Gemma Miller, Tess Vernon, and Helen Kettle. A parsimonious software sensor for estimating the individual dynamic pattern of methane emissions from cattle. *animal*, 13(6):1180–1187, 2019. doi:10.1017/S1751731118002550.

[12] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. `http://www.R-project.org"`.

[13] Wikipedia contributors. Longest increasing subsequence — Wikipedia, the free encyclopedia, 2019. [Online; accessed 21-August-2019].

Algorithm 1 Detection of plateaus (part 1/2)

Input	*data*	an array of *Point*s, ordered by time
	max_delta	maximum between current weight and plateau median weight
	max_gap	max duration between two consecutive points in a plateau
	min_duration	minimum plateau duration
Output	*result*	an array of *Plateau*s, ordered by time
Point	*.t*	time
	.w	`weight_m` measured by the weighing platform
Plateau	*.begin*	begin time
	.end	end time
	.wm	`weight_m` median weight during the plateau

```
1:  procedure PLATEAU(data, max_delta, max_gap, min_duration)
2:      buffer ← an empty array of Points
3:      for i = 0; i < length(data); ++i do
4:          if length(buffer) == 0 then                              ▷ (1)
5:              append(buffer, data[i])
6:          else
7:              delta ← data[i].w − median(buffer.w)                 ▷ (2.1)
8:              delta_ok = abs(delta) ≤ max_delta
9:              gap ← data[i].t − buffer[length(buffer) − 1].t       ▷ (2.2)
10:             gap_ok ← abs(gap) ≤ max_gap
11:             if not(gap_ok) or not(delta_ok) then
12:                 plateau_eval(&buffer, &result, min_duration)     ▷ (2.3)
13:             end if
14:             append(buffer, data[i])                              ▷ (3)
15:         end if
16:     end for
17:     plateau_eval(buffer, result, min_duration)                   ▷ (4)
18:     return result
19: end procedure
```

The PLATEAU algorithm uses a buffer to store points that potentially belong to a plateau. Points are read one by one, ordered by time. For each point, if the buffer is empty, we start a plateau by adding the current point to it (1) and we go to the next point. If there is at least one point in the buffer, we compute the boolean *delta_ok* as true if the current point `weight_m` is close enough (distance $\leq$ *max_delta*) to the median weight buffer (2.1) ; and the boolean *gap_ok* as true if the duration betwen the current point and the last point of the plateau is $\leq$ *max_gap* (2.2). If *delta_ok* or *gap_ok* is false, we consider that there is a plateau end, and we call the procedure PLATEAU_EVAL (2.3). This procedure checks if a plateau is kept, and clears the buffer. Regardless of PLATEAU_EVAL result, the current point is added the buffer. After processing all points, the buffer may not be empty, we call the procedure PLATEAU_EVAL a last time in order not to lose the last plateau (4).

Algorithm 1 Detection of plateaus (part 2/2)

```
1: procedure PLATEAU_EVAL(&buffer, &result, min_duration)
2:     if length(buffer) ≥ min_duration then                    ▷ (5)
3:         p ← a Plateau
4:         p.begin ← buffer[0].t
5:         p.end ← buffer[length(buffer)].t
6:         p.wm ← median(buffer.w)
7:         append(result, p)
8:     end if
9:     buffer ← an empty array of Points                        ▷ (6)
10: end procedure
```

The procedure PLATEAU_EVAL examines the buffer and decides if it's a valid plateau or not. To do so, if the buffer contains a plateau longer than min_length, we consider it as valid and we add it to $result$ (5). A valid plateau starts at the first point in the buffer, finishes at the last one, and has a weight equal to the median of all weights in the buffer. Regardless of plateau validity, we clean the buffer (6).
In procedure definition, the & symbol indicates that the following argument is passed by reference and can therefore be modified by the procedure.

Table 1: Parameters of Algo 1

dataset	parameter	value
cow	`max_delta`	30 g
	`max_gap`	2 s
	`min_duration`	7 s
goat	`max_delta`	15 g
	`max_gap`	6 s
	`min_duration`	10 s
mice	`max_delta`	0.04 g
	`max_gap`	5 s
	`min_duration`	10 s

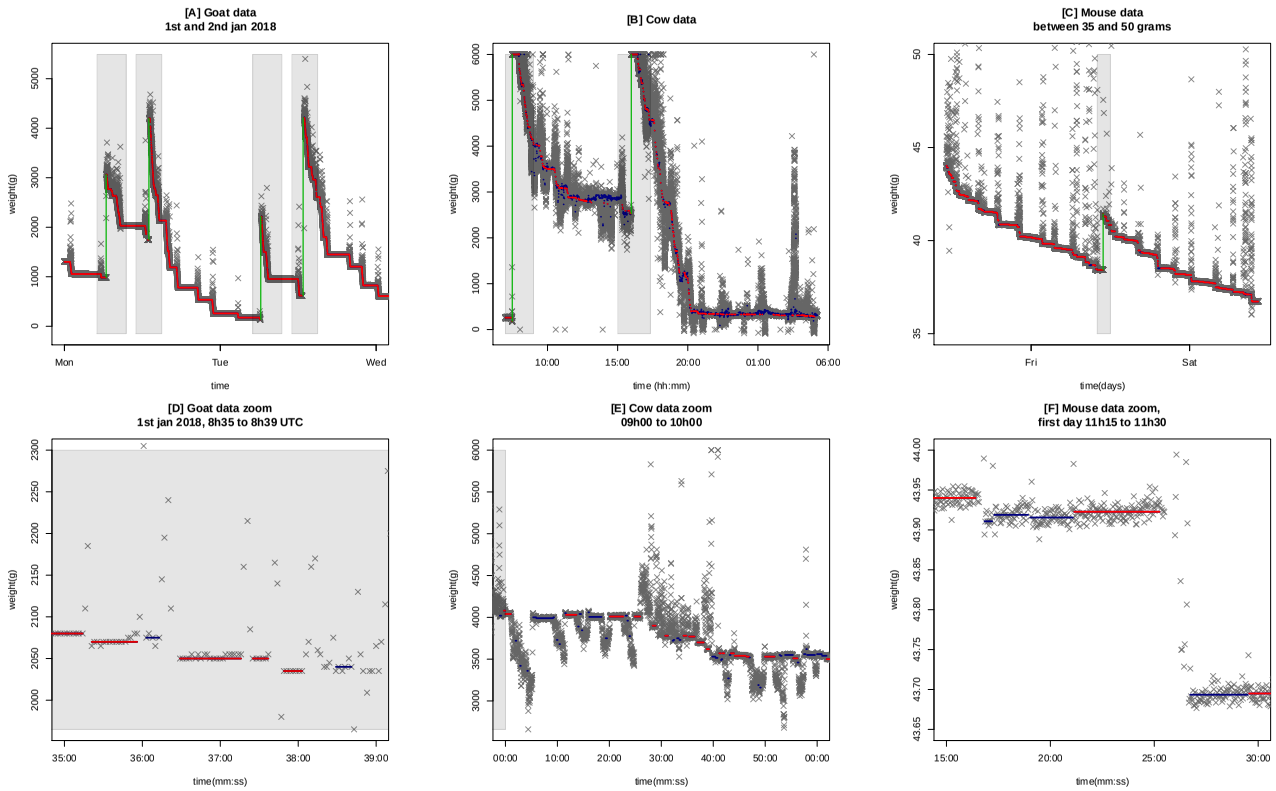

Figure 1: **Algorithm results**

× Measured data. Food distribution search range (see section 3.6). Food distribution (see section 3.6). Plateau **not in** longest weighted decreasing sub sequence (see section 3.5). Plateaus **in** longest weighted decreasing sub sequence (see section 3.7).

[**A**] Goat data (see section 3.1). Raw data were filtered to remove points below -300 grams that correspond to feed bin removal during distribution. [**B**] Cow data (see section 3.2). Raw data were filtered by removing all points below -0.1 kg, and by replacing all points above 6kg by 6kg. [**C**] Mouse data (see section 3.3). Raw data were filtered by removing all points below 0g and above 100g. [**D,E,F**] Zoom of goat, cow and mice data.

As seen on the zoom plots, the plateaus correctly match the parts of the curve where the data are stable, and discard the rest of the data. The plots [**A,B,C**] show that distributions are correctly detected, and that the longest weighted decreasing sub-sequence of plateaus correctly matches the general trend of the curve.

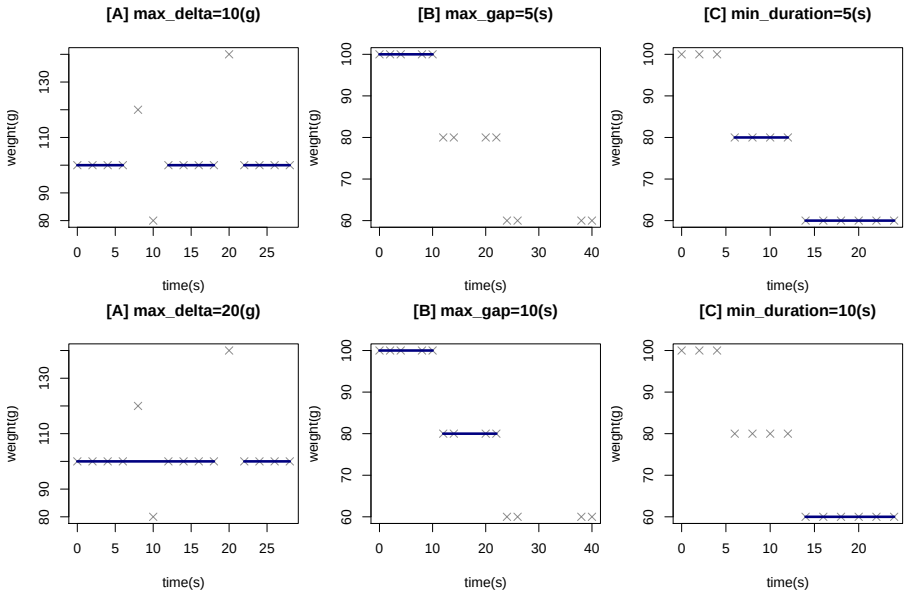

Figure 2: **Effects of parameter values on plateau detection** × Raw data (see `code/parameters.R`). — Plateau (see section 3.5). In this example plateaus are computed on raw data. When not specified otherwise in the plot title, parameters are : `max_delta`=10g, `max_gap`=5s, `min_duration`=5s. [**A**] When the distance between the current plateau median weight and the current weight is higher than `max_delta`, the plateau is broken. Therefore increasing `max_delta` will aggregate in the same plateau points of different weight. [**B**] When there is no point for a duration longer than `max_gap`, the plateau is broken. [**C**] Plateaus shorter than `min_duration` are discarded.