H2020-SFS-2018-2020

**DECIDE**

**Data-driven control and prioritisation of   
non-EU-regulated contagious animal diseases**

Deliverable D2.1

Open-source software tools to perform multivariate monitoring of time-series

WP 2 – Methods for data analysis and modelling

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| **Authors** |  | Leonardo de Knegt, Carolina Merca, Anders Kristensen |
| Lead participant |  | Anders Kristensen |
| Delivery date |  | 27 june 2023 |
| Dissemination level |  | Public / Confidential |
| Type |  | Report / Ethics / Other / DEC / ORDP |

Revision History

|  |  |  |
| --- | --- | --- |
| Author Name (Partner short name) | Description | Date |
|  | Draft deliverable | 26.06.2023 |
|  |  |  |
|  |  |  |
|  | Revision 1 | dd.mm.yyyy |
|  |  |  |
|  |  |  |
|  | Final version | dd.mm.yyyy |

**Content**

Executive Summary 5

1 Data preparation 7

2 Dynamic Linear Models 8

2.1 Univariate DLM on milk yield 8

2.1.1 Parametrization and training of the DLM using healhty cow parities 8

2.1.2 Applying the trained model to the sick data 12

2.1.3 Assessment of performance oft he DLM as part of an alarm system 12

2.2 Multivariate DLM 13

2.2.1 Parametrization and training of the multivariate DLM 13

2.2.2 Applying the trained multivariate model to the sick data 16

Abbreviations

|  |  |
| --- | --- |
| Abbreviation | Description |
| EU | European Union |
| H2020 | Horizon 2020 |
| WP | Work Package |
| DLM | Dynamic Linear Model |
| SCC | Somatic Cell Count |
|  |  |
|  |  |

Partner short names

|  |  |
| --- | --- |
| Short name | Organisation |
| UCPH | Københavns Universitet |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Executive Summary

Animal health surveillance requires a continual stream of data. It also needs a framework that allows knowledge from information previously acquired to accumulate, leading to an improved understanding of the present situation and the detection of apparent trends or unexpected changes. State-space models offer such a framework, in which relevant prior knowledge and current information are combined to detect changes in an observed process, thus allowing for a better understanding of the situation, and resulting in better-informed decisions.

A basic model to describe a time series comprises an expected underlying value (or “true mean”), a sample error and an observation error. The fundamental assumption behind this type of model is that the true underlying mean is constant over time, but this assumption is often dubious, since values can often suffer some random fluctuations, or vary over the day or according to seasons, or just systematically increase or decrease over time.

For that reason, a type of system is required, which allows the estimation of an observation based on the true underlying mean with its measurement or observation errors, but also of the dynamic aspects of those observations, meaning its systematic fluctuations and changes over time. Therefore, a state-space model is normally defined by two equations: an observation equation describing the data observed by a vector of parameters, and a system equation describing the dynamics of the parameter vector as a first order autocorrelation model. Components of the system equation may include, for example, fluctuations in water drinking during the day, lactation curves, disease seasonality or positive weight gain linear trends.

For the purposes of the present project, the most commonly used type of state space model is a Dynamic Linear Model (DLM). A DLM uses a Bayesian framework to estimate the underlying parameter vector from the observed data, while taking into account any prior information available before the observations are done. Values are forecast at each time-step, based on the theoretical true mean and prior knowledge on error and variance around the system and the data, and are then updated on the next time step, being “corrected” according to each new observation, as well as to the error and variance components already mentioned.

**Objectives of the Deliverable**

With the help of this deliverable, the reader will be walked through an approach to detect mastitis from standardized forecast errors yielding from DLMs, and will be given the pre-programmed functions needed to run such models. The present deliverable was based on R codes written by Dan Børge Jensen.

**Activities**

Two broad methods of DLM will be applied:

• The univariate DLM, where a DLM is defined and applied to each variable separately

• The multivariate DLM, where a single DLM is defined to describe all the relevant variables simultaneously

In order to perform those activities, three R code files accompany this deliverable:

**Outcome**

With the help of the R scripts containing the walkthrough and the functions used to define a DLM for mastitis detection, the reader should be able to develop their own DLMs for monitoring of their own variables of interest.

**Next steps**

The next step for WP2 is to prepare a similar document for multilevel monitoring of time-series data.

# Data preparation

The contents of the present report refer to the scripts named "DLM example, documentation and walkthrough – DECIDE.R",and "DLM functions\_DECIDE.R". The processes described here can be followed step by step by running or adapting those codes.

The type of data this script would be applied to is milk yield from dairy cows in a farm where mastitis is detected every now and then. Besides the variables containing yield per se, there are also variables describing milk composition, SCC and blood presence.

The observational units are cow-parities, meaning that she same cow will appear as two different individuals at two different lactations. This allows for the separation of the data per lactation group, if necessary, and also isolates the developing production curve a cow has in an individual lactation. Milk yield measurements are presented on a daily basis, so the time unit is days, and the time-step unit is days in milking (dim). Cows on the first lactation are defined as "primiparous" cows, against "multiparous" cows from other lacatations. This information is contained in a variable named "ParityGroup".

The dataset is split between healthy cows (to train the model) and sick cows, to try and detect changes from the "normal" baseline obtained with the healthy cow data. In this code, those files are called "healthy.set" and "sick.set", respectively.

For the purposes of this report, we will only consider "Primiparous" cows. Use the \*subset\* function the extract the subset of \*healthy.set\* where \*ParityGroup\* == "Primiparous". Call the resulting data frame "healthy.set" (i.e. replace the existing data frame with the subset you extract). Do the same for the "sick.set".

healthy.set <- subset(healthy.set, healthy.set$ParityGroup == "Primiparous")

sick.set <- subset(sick.set, sick.set$ParityGroup == "Primiparous")

When variables differ too much in range and magnitude, larger numerical changes can drive the model, so it is customary to standardize them. It is not strictly necessary to standardize the data used in a univariate DLM.

names <- c("yield", "conductivity", "fat", "protein", "lactose", "scc", "blood")

for(name in names){

Mean <- mean(healthy.set[,name])

SD <- sd(healthy.set[,name])

healthy.set[,name] <- (healthy.set[,name] - Mean)/SD

sick.set[,name] <- (sick.set[,name] - Mean)/SD

}

Subset healthy.set to contain the earliest values in the series, from days in milking zero to five, and call the resulting data frame "healthy\_start.set".

healthy\_start.set <- subset(healthy.set, healthy.set$dim <= 5)

Obtain the unique identifiers for cow-parities in healthy.set and \*sick.set\*.

IDs.healthy <- unique(healthy.set$Cow.Parity)

IDs.sick <- unique(sick.set$Cow.Parity)

Create a new data frame called "meta.data.sick", consisting of the following columns from the "sick.set

"Cow.Parity", "cow", "lactno", "ParityGroup", "dim" and "mastitis".

meta.data.sick <- sick.set[, c("Cow.Parity", "cow", "lactno", "ParityGroup", "dim", "mastitis")])

# Dynamic Linear Models

## Univariate DLM on milk yield

Source the script containing the pre-programmed functions for the DLM.

source('DLM functions\_DECIDE.R')

### Parametrization and training of the DLM using healhty cow parities

Obtain the initial parameter vector (mu0) and the initial variance matrix (C0) for milk yield, by using functions get.mu0 and get.C0 from the sourced functions file. If you can’t understand what the input parameters for the functions are, find the description directly in the provided script file.

outmu <- get.mu0(Data = healthy\_start.set,

stratify.by=NA,

time.var = 'dim',

expected.start.time = 1,

relevant.names = c('yield'),

simple.linear = FALSE)

outc <- get.C0(Data = healthy\_start.set,

stratify.by=NA,

time.var = 'dim',

expected.start.time = 1,

relevant.names = c('yield'))

mu0 <- outmu$mu0

C0 <- outc$C0

Obtain the system matrix Gt and the design matrix Ft, by using functions get.Gt and get.Ft . When running the two functions, set relevant.names = c('yield'), because this is a univariate model. If your DLM contains a non-linear trend component, you need to first define a spline for the data series in order to build the Gt matrix, as explained in the code file. You can do that by using function get.spline.

Spline.list <- get.spline(Data = healthy.set,

stratify.by=NA,

time.var = 'dim',

relevant.names = c('yield'),

plot.it = FALSE)

Gt <- get.Gt(Data.A=healthy.set,

i.A=NA,

time.var.A='dim',

stratify.by.A=NA,

Spline.list.A=Spline.list,

relevant.names.A=c('yield'))

Ft <- get.Ft(relevant.names.A = c('yield'))

Obtain the observational variance (V) for yield using function get.V.

V <- get.V(Data = healthy.set, identifyer='Cow.Parity', stratify.by=NA, time.var='dim', relevant.names=c('yield'))

After obtaining a data-based V from the function above, it is possible to directly estimate W and update/optimize the V initially obtained with get.V, by using an Estimation Maximization (EM) algorithm. The EM function uses several functions that have been sourced from the functions file, such as getVSumElement, runDLM (we will come back to this one later), runSmoother, and the EM algorythm for a specific number of steps (runEM), since we are going to run a version of the same algorythm with an early-stopping feature that stops running when the values of V and W yielding the best model performance are found.

varcom <- runEM\_earlyStopping(Data=healthy.set, stratify.by=NA, Spline.list=Spline.list, identifyer='Cow.Parity', V0=V$V\_1, W0=NA, C0.list=outc, mu0.list=outmu, no.better.limit=1, time.var='dim', relevant.names=c('yield'), round.by=4)

The output "varcom" should be a list containing objects V.list, W.list, muo.list and C0.list. Those, in turn, contain V\_1 (which is the EM-optimized version of the V\_1 initially obtained with get.V), W\_1 (the value of W estimated by the EM algorithm), mu0\_1 (the same mu0 obtained earlier by get.mu0 and stored in mu0) and C0\_1 (the same C0 obtained earlier using get.C0 and stored in outc). Those should be the variance components used as input for your DLM on the next step. In situations for which there is not enough data, not enough time or not enough computing power to use the EM algorithm to optimize V and obtain W, it is possible to use a discount factor (delta) as the percentage of the total variance C that corresponds to W. As in the EM algorithm, several values of delta can be tested and optimized.

If you need to find the optimal delta value to model yield with the DLM, you can use the function optimize.delta :

delta <- optimize.delta(deltas=seq(from=0.8, to=1, by=0.01),

healthy.set,

identifyer='Cow.Parity',

mu0.list=outmu,

C0.list=outc,

V.list=V$V\_1,

relevant.names=c('yield'),

Spline.list=Spline.list,

time.var='dim',

stratify.by=NA)

Apply the optimized DLM to the yield of each cow-parity in the healthy set, in order to estimate the parameters we need for the control chart. Use the extract.res function to extract the output of the DLM (which is returned as lists) as a data frame, and call the resulting data frame "extracted". From this data frame, get the standardized forecast errors from the column called "ut\_yield" and call it "ut". Add the standardized forecast errors to the ut.all vector.

ut.all <- c()

extracted.all <- data.frame()

for(ID in IDs.healthy){

#Extract the subset for the current ID

ID.set <- subset(healthy.set, healthy.set$Cow.Parity == ID)

# Apply the DLM and call the result "res".

res <- runDLM(ID.set,

mu0=varcom$mu0.list$mu0\_1,

C0=varcom$C0.list$C0\_1,

V=varcom$V.list$V\_1,

W=varcom$W.list$W\_1,

adjust.W=FALSE,

delta=NA,

relevant.names=c('yield'),

Spline.list=Spline.list,

time.var='dim',

stratify.by=NA)

# Use the \*extract.res\* function to extract the output of the DLM (which is returned as lists) as a data frame, and call the resulting data frame "extracted".

extracted <- extract.res(res = res, relevant.names=c('yield'))

extracted.all <- rbind(extracted.all, extracted)

# From \*extracted\*, get the standardized forecast errors from the column called "ut\_yield" and call it "ut". Then add the standardized forecast errors to the ut.all vector.

ut <- extracted$ut\_yield

ut.all <- c(ut.all, ut)

}

Now validate the DLM by assessing whether the standardized forecast errors follow a standard normal distribution, plotting it and getting the percentage outside of the 95 % CI. Use function \*assess.ut\* for that :

assess.ut(extracted.all)

### Applying the trained model to the sick data

ut.all.sick <- c()

extracted.all.sick <- data.frame()

for(ID in IDs.sick){

ID.set <- subset(sick.set, sick.set$Cow.Parity == ID)

res <- runDLM(ID.set,

mu0=varcom$mu0.list$mu0\_1,

C0=varcom$C0.list$C0\_1,

V=varcom$V.list$V\_1,

W=varcom$W.list$W\_1,

adjust.W=FALSE,

delta=NA,

relevant.names=c('yield'),

Spline.list=Spline.list,

time.var='dim',

stratify.by=NA)

extracted <- extract.res(res = res, relevant.names=c('yield'))

extracted.all.sick <- rbind(extracted.all.sick, extracted)

ut <- extracted$ut\_yield

ut.all.sick <- c(ut.all.sick, ut)

}

### Assessment of performance oft he DLM as part of an alarm system

After applying the optimized DLM to the data from cows with mastitis and extracted the standardized forecast errors, it is now possible to assess the performance achieved when using the standardized forecast errors of the DLM as part of a mastitis alarm system. Apply the alarmsMontgomery function from the sourced script, and call the result "alarms", and create a vector called "Rules", containing the column names of the "alarms" data set. .

alarms <- alarmsMontgomery(k = ut.all.sick,

cl = mean(ut.all.sick),

SD = sd(ut.all.sick),

Ylim = NA,

Main = 'yield',

plot.it = FALSE)

Rules <- colnames(alarms)

Apply the getPerformanc\* function from the sourced script, and call the result "performance".

*performance <- getPerformance(*

*observations = sick.set[,'mastitis'],*

*alarms = alarms[,'AnyRule'])*

*print(performance)*

## Multivariate DLM

Now we wish to make a single multivariate DLM, which simultaneously describes all the numerical variables, while taking their mutual co-variances into account.

### Parametrization and training of the multivariate DLM

First, we create the initial parameters, mu0 and C0, for the multivariate DLM.

names <- c("yield", "conductivity", "fat", "protein", "lactose", "SCS", "blood")

outmu <- get.mu0(Data = healthy\_start.set,

stratify.by=NA,

time.var = 'dim',

expected.start.time = 1,

relevant.names = names,

simple.linear = FALSE)

outc <- get.C0(Data = healthy\_start.set,

stratify.by=NA,

time.var = 'dim',

expected.start.time = 1,

relevant.names = names)

mu0 <- outmu$mu0

C0 <- outc$C0

Create Gt and Ft matrices for the multivariate model.

Spline.list <- get.spline(Data = healthy.set,

stratify.by=NA,

time.var = 'dim',

relevant.names = names,

plot.it = FALSE)

Gt <- get.Gt(Data.A=healthy.set,

i.A=NA,

time.var.A='dim',

stratify.by.A=NA,

Spline.list.A=Spline.list,

relevant.names.A=names)

Ft <- get.Ft(relevant.names.A = names)

Calculate a data-based initial value for V.

V <- get.V(Data = healthy.set,

identifyer='Cow.Parity',

stratify.by=NA,

time.var='dim',

relevant.names=names)

Apply the EM algorithm or optimize delta.

varcom <- runEM\_earlyStopping(Data=healthy.set,

stratify.by=NA,

Spline.list=Spline.list,

identifyer='Cow.Parity',

V0=V$V\_1,

W0=NA,

C0.list=outc,

mu0.list=outmu,

no.better.limit=1,

time.var='dim',

relevant.names=names,

round.by=4)

delta <- optimize.delta(deltas=seq(from=0.8, to=1, by=0.01),

healthy.set,

identifyer= 'Cow.Parity',

mu0.list=outmu,

C0.list=outc,

V.list=V$V\_1,

relevant.names= c('yield'),

Spline.list=Spline.list,

time.var='dim',

stratify.by=NA)

Apply the optimized multivariate DLM to the yield of each cow-parity in the healthy set.

ut.all <- c()

extracted.all <- data.frame()

for(ID in IDs.healthy[c(1:3)]){

ID.set <- subset(healthy.set, healthy.set$Cow.Parity == ID)

res <- runDLM(ID.set,

mu0=varcom$mu0.list$mu0\_1,

C0=varcom$C0.list$C0\_1,

V=varcom$V.list$V\_1,

W=varcom$W.list$W\_1,

adjust.W=FALSE,

delta=NA,

relevant.names=names,

Spline.list=Spline.list,

time.var='dim', stratify.by=NA)

extracted <- extract.res(res = res, relevant.names=names)

extracted.all <- rbind(extracted.all, extracted)

ut <- data.frame(extracted[,grep(pattern = 'ut\_', x = colnames(extracted))])

colnames(ut) <- colnames(extracted)[grep(pattern = 'ut\_', x = colnames(extracted))]

ut.all <- rbind(ut.all, ut)

}

### Applying the trained multivariate model to the sick data

ut.all.sick <- c()

extracted.all.sick <- data.frame()

for(ID in IDs.sick[c(1:4)]){

ID.set <- subset(sick.set, sick.set$Cow.Parity == ID)

res <- runDLM(ID.set,

mu0=varcom$mu0.list$mu0\_1,

C0=varcom$C0.list$C0\_1,

V=varcom$V.list$V\_1,

W=varcom$W.list$W\_1,

adjust.W=FALSE,

delta=NA,

relevant.names=names,

Spline.list=Spline.list,

time.var='dim',

stratify.by=NA)

extracted <- extract.res(res = res, relevant.names=names)

extracted.all.sick <- rbind(extracted.all.sick, extracted)

ut <- data.frame(extracted[,grep(pattern = 'ut\_', x = colnames(extracted))])

colnames(ut) <- colnames(extracted)[grep(pattern = 'ut\_', x = colnames(extracted))]

ut.all <- rbind(ut.all, ut)

}

We will not test the performances of each variable on their own again, because in this specific case, the performances would be comparable to what we saw before in the univariate example. Instead, we will save the data that we can extract from the output of the DLM :

sick.set\_after.DLM <- cbind(meta.data.sick, extracted.all.sick)

saveRDS(object = sick.set\_after.DLM, file = "sick.set\_after.DLM.RDS")