By clustering of consumers of electricity load, we can extract typical load **profiles**, improve the accuracy of consequent electricity consumption forecasting, detect anomalies or monitor a whole smart grid (grid of consumers) (Laurinec et al. (2016), Laurinec and Lucká (2016)). I will show you the first use case, the extraction of typical electricity load profiles by [**K-medoids**](https://en.wikipedia.org/wiki/K-medoids) clustering method.

Firstly, let’s load the required packages, data (elec\_load) and return the dimensions of the dataset.

library(TSrepr)

library(ggplot2)

library(data.table)

library(cluster)

library(clusterCrit)

data("elec\_load")

dim(elec\_load)

## [1] 50 672

There are 50 time series (consumers) of length 672, specifically time series of length 2 weeks of electricity consumption.  
These measurements are from smart meters from an unknown country.

It is obvious that dimensionality is too high and the [**curse of dimensionality**](https://en.wikipedia.org/wiki/Curse_of_dimensionality) can happen.  
For this reason, we have to reduce dimensionality in some way. One of the best approaches is to use time series representations in order to reduce dimensionality, reduce noise and emphasize the main characteristics of time series.

For double seasonal time series of electricity consumption (daily and weekly seasonality), the model-based representation approaches seem to have best ability to extract typical profiles of consumption.  
However, I will use some other representation methods implemented in **TSrepr** too.

Let’s use one of the basic model-based representation methods – **mean seasonal profile**. It is implemented in the repr\_seas\_profile function and we will use it alongside repr\_matrix function that computes representations for every row of a matrix of time series. One more very important notice here, normalisation of time series is a necessary procedure before every clustering or classification of time series. It is due to a fact that we want to extract typical curves of consumption and don’t cluster based on an amount of consumption.  
By using the **TSrepr** package, we can do it all in one function – repr\_matrix. We will use z-score normalisation implemented in norm\_z function.  
We have 48 measurements during a day, so set freq = 48.

data\_seasprof <- repr\_matrix(elec\_load, func = repr\_seas\_profile,

args = list(freq = 48, func = mean),

normalise = TRUE, func\_norm = norm\_z)

dim(data\_seasprof)

## [1] 50 48

We can see that dimensionality was in fact reduced significantly. Now, let’s use the K-medoids (pam function from cluster package) clustering method to extract typical consumption profiles.  
Since we don’t know a proper number of clusters to create, i.e. a priori information about a number of clusters, an [internal validation index](https://en.wikipedia.org/wiki/Cluster_analysis#Internal_evaluation) is have to be used to determine an optimal number of clusters.  
I will use well known [Davies-Bouldin index](https://en.wikipedia.org/wiki/Davies%E2%80%93Bouldin_index) for this evaluation. By Davies-Bouldin index computation, we want to find a minimum of its values.

I will set the range of number of clusters to 2-7.

clusterings <- lapply(c(2:7), function(x)

pam(data\_seasprof, x))

DB\_values <- sapply(seq\_along(clusterings), function(x)

intCriteria(data\_seasprof, as.integer(clusterings[[x]]$clustering),

c("Davies\_Bouldin")))

Let’s plot results of internal evaluation.

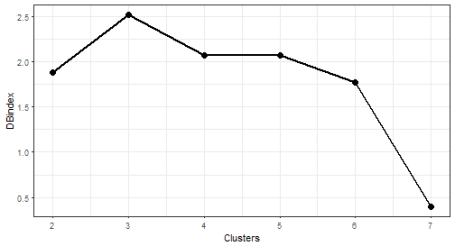
ggplot(data.table(Clusters = 2:7, DBindex = unlist(DB\_values)),

aes(Clusters, DBindex)) +

geom\_line(size = 1) +

geom\_point(size = 3) +

theme\_bw()



The “best” number of clusters is 7 (lowest value of the index).

Let’s plot the results of clustering with 7 number of clusters. I will use computed medoids stored in the object clustering. For a visualization, the facet\_wrap function is very handy here.

# prepare data for plotting

data\_plot <- data.table(melt(data.table(class = as.factor(clusterings[[6]]$clustering),

data\_seasprof)))

data\_plot[, Time := rep(1:ncol(data\_seasprof), each = nrow(data\_seasprof))]

data\_plot[, ID := rep(1:nrow(data\_seasprof), ncol(data\_seasprof))]

# prepare medoids

centers <- data.table(melt(clusterings[[6]]$medoids))

setnames(centers, c("Var1", "Var2"), c("class", "Time"))

centers[, ID := class]

# plot the results

ggplot(data\_plot, aes(Time, value, group = ID)) +

facet\_wrap(~class, ncol = 2, scales = "free\_y") +

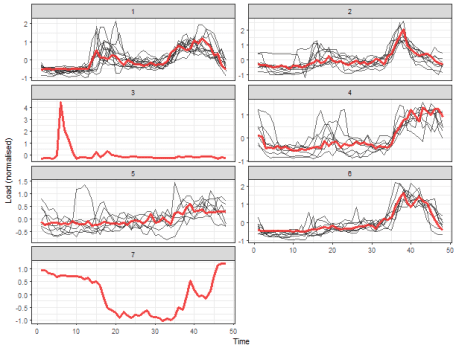
geom\_line(color = "grey10", alpha = 0.65) +

geom\_line(data = centers, aes(Time, value),

color = "firebrick1", alpha = 0.80, size = 1.2) +

labs(x = "Time", y = "Load (normalised)") +

theme\_bw()



We can see 5 typical extracted profiles – red lines (medoids of clusters). The next two clusters are one-elemental, so they can be referred as outliers.

Now, let’s try some more sophisticated method for the extraction of seasonal profiles – **GAM regression coefficients**. By repr\_gam function, we can extract double seasonal regression coefficients – daily and weekly (you can read more about the analysis of seasonal time series with **GAM** below).

Code Chunks – Analysis of seasonal time series with GAM

GAM methods are implemented in **R** in the awesome package mgcv

For visualizations packages ggplot2, grid and animation will be used. One useful function from package car will be used too.

Let’s scan all of the needed packages.

library(feather)

library(data.table)

library(mgcv)

library(car)

library(ggplot2)

library(grid)

library(animation)

Read the mentioned smart meter data by read\_feather to one data.table.

DT <- as.data.table(read\_feather("DT\_4\_ind"))

Prepare DT to work with a **GAM** regression model. Transform the characters of weekdays to integers and use function recode from package car to recode weekdays as there are coming in the week: 1. Monday, …, 7. Sunday.

DT[, week\_num := as.integer(car::recode(week,

"'Monday'='1';'Tuesday'='2';'Wednesday'='3';'Thursday'='4';

'Friday'='5';'Saturday'='6';'Sunday'='7'"))]

Store informations in variables of the type of industry, date, weekday and period for simpler working.

n\_type <- unique(DT[, type])

n\_date <- unique(DT[, date])

n\_weekdays <- unique(DT[, week])

period <- 48

Let’s look at some data chunk of electricity consumption and do analysis on it. I have picked aggregate consumption of commercial properties for two weeks. Store it in variable data\_r and plot it.

data\_r <- DT[(type == n\_type[1] & date %in% n\_date[57:70])]

ggplot(data\_r, aes(date\_time, value)) +

geom\_line() +

theme(panel.border = element\_blank(),

panel.background = element\_blank(),

panel.grid.minor = element\_line(colour = "grey90"),

panel.grid.major = element\_line(colour = "grey90"),

panel.grid.major.x = element\_line(colour = "grey90"),

axis.text = element\_text(size = 10),

axis.title = element\_text(size = 12, face = "bold")) +

labs(x = "Date", y = "Load (kW)")



There is possible to see two main seasonalities in plotted time series: daily and weekly. We have 48 measurements during the day and 7 days during the week so that will be our independent variables to model response variable - electricity load. Let’s construct it:

N <- nrow(data\_r) # number of observations in the train set

window <- N / period # number of days in the train set

matrix\_gam <- data.table(Load = data\_r[, value],

Daily = rep(1:period, window),

Weekly = data\_r[, week\_num])

Here we are! Train our first **GAM** with function gam. Independent variables are modeled by smoothing function s, for daily seasonality cubic regression spline is used, for weekly seasonality, P-splines is used, a number of knots are logically set to the number of unique values. Let’s do it.

gam\_1 <- gam(Load ~ s(Daily, bs = "cr", k = period) +

s(Weekly, bs = "ps", k = 7),

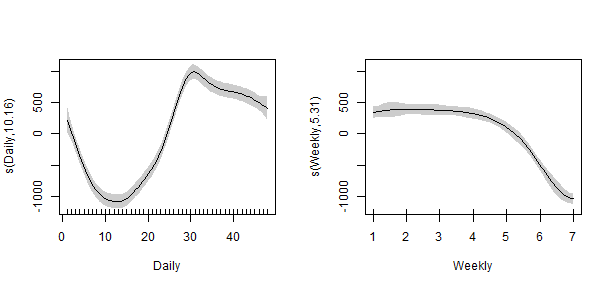
data = matrix\_gam,

family = gaussian)

Package mgcv have many advantages and nice features. First is its visualization capabilities. Let’s try it:

layout(matrix(1:2, nrow = 1))

plot(gam\_1, shade = TRUE)



That looks nice, right? We can see here the influence of variables to electricity load. In the left plot, the peak of the load is around 3 p.m. during the day. In the right plot, we can see that during weekends the consumption logically decreases.

Let’s use summary function to do the diagnostic of our first model.

summary(gam\_1)

##

## Family: gaussian

## Link function: identity

##

## Formula:

## Load ~ s(Daily, bs = "cr", k = period) + s(Weekly, bs = "ps",

## k = 7)

##

## Parametric coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 2731.67 18.88 144.7 <2e-16 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Approximate significance of smooth terms:

## edf Ref.df F p-value

## s(Daily) 10.159 12.688 119.8 <2e-16 \*\*\*

## s(Weekly) 5.311 5.758 130.3 <2e-16 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## R-sq.(adj) = 0.772 Deviance explained = 77.7%

## GCV = 2.4554e+05 Scale est. = 2.3953e+05 n = 672

What to look at here? EDF: estimated degrees of freedom - can be interpreted like how much given variable is smoothed (higher EDF value implies more complex splines). P-values: statistical significance of given variable to response variable, tested by F-test (lower is better). R2R2 - adjusted R-squared (higher is better). GCV: GCV score (mentioned above). In the summary, we can see that the R-sq. (adj) value is a little bit low…

Let’s plot fitted values:

datas <- rbindlist(list(data\_r[, .(value, date\_time)],

data.table(value = gam\_1$fitted.values,

data\_time = data\_r[, date\_time])))

datas[, type := c(rep("Real", nrow(data\_r)), rep("Fitted", nrow(data\_r)))]

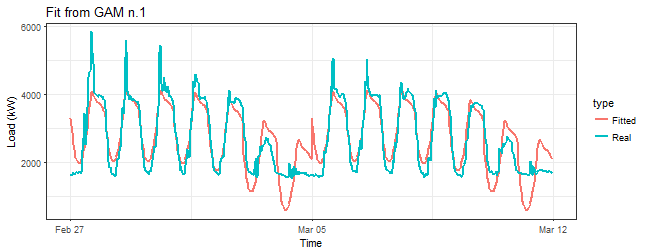
ggplot(data = datas, aes(date\_time, value, group = type, colour = type)) +

geom\_line(size = 0.8) +

theme\_bw() +

labs(x = "Time", y = "Load (kW)",

title = "Fit from GAM n.1")



That is, of course, a horrible result. We need to include interactions of two independent variables to the model.

One more thing before that… What is hiding behind the stored gam\_1 object and used optimizer?

gam\_1$optimizer

## [1] "magic"

Here is that “magic”! :smile:

The first type of interactions will be the third one mentioned in the section about interactions, so one smoothing function to both variables is used. In the term s(Daily, Weekly) simple smoothing terms s(Daily) and s(Weekly) are also automatically included. Let’s do it.

gam\_2 <- gam(Load ~ s(Daily, Weekly),

data = matrix\_gam,

family = gaussian)

summary(gam\_2)$r.sq

## [1] 0.9352108

R-squared value suggests that result is much better. Look at the smooth term:

summary(gam\_2)$s.table

## edf Ref.df F p-value

## s(Daily,Weekly) 28.7008 28.99423 334.4754 0

Seems good too, the p-value is 0, which means that independent variable is significant. Plot of fitted values:

datas <- rbindlist(list(data\_r[, .(value, date\_time)],

data.table(value = gam\_2$fitted.values,

data\_time = data\_r[, date\_time])))

datas[, type := c(rep("Real", nrow(data\_r)), rep("Fitted", nrow(data\_r)))]

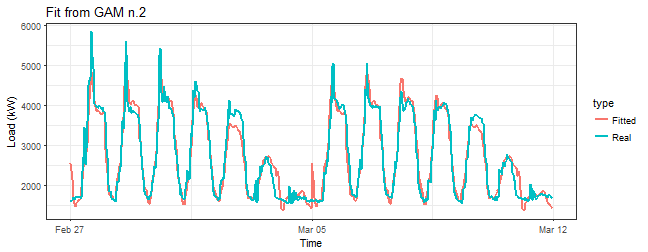
ggplot(data = datas, aes(date\_time, value, group = type, colour = type)) +

geom\_line(size = 0.8) +

theme\_bw() +

labs(x = "Time", y = "Load (kW)",

title = "Fit from GAM n.2")



It’s not bad, but it can be better. Now, let’s try the above mentioned tensor product interactions. That can be done by function te, basis functions can be defined as well.

gam\_3 <- gam(Load ~ te(Daily, Weekly,

bs = c("cr", "ps")),

data = matrix\_gam,

family = gaussian)

summary(gam\_3)$r.sq

## [1] 0.9268452

Similar to the previous model gam\_2. Look at the smooth term:

summary(gam\_3)$s.table

## edf Ref.df F p-value

## te(Daily,Weekly) 23.65709 23.98741 354.5856 0

Again very similar results. Let’s look at fitted values:

datas <- rbindlist(list(data\_r[, .(value, date\_time)],

data.table(value = gam\_3$fitted.values,

data\_time = data\_r[, date\_time])))

datas[, type := c(rep("Real", nrow(data\_r)), rep("Fitted", nrow(data\_r)))]

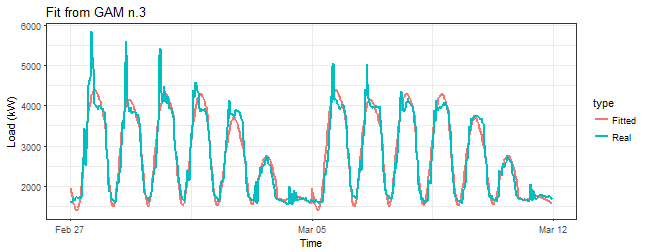
ggplot(data = datas, aes(date\_time, value, group = type, colour = type)) +

geom\_line(size = 0.8) +

theme\_bw() +

labs(x = "Time", y = "Load (kW)",

title = "Fit from GAM n.3")



Just a little differences in comparison to the gam\_2 model, looks like with te fit is more smoothed. Are we missing something? Of course! Function te has a default for a number of knots 5d5d, where d is a number of dimensions (variables), which is in our case too small. Now, I will set a number of knots to the maximal possible value k = c(period, 7), what means that upper boundary for EDF (Estimated Degrees of Freedom) will be 48\*7 - 1 = 335.

gam\_4 <- gam(Load ~ te(Daily, Weekly,

k = c(period, 7),

bs = c("cr", "ps")),

data = matrix\_gam,

family = gaussian)

summary(gam\_4)$r.sq

## [1] 0.9727604

summary(gam\_4)$sp.criterion

## GCV.Cp

## 34839.46

summary(gam\_4)$s.table

## edf Ref.df F p-value

## te(Daily,Weekly) 119.4117 149.6528 160.2065 0

We can see here that R-squared jumped little bit up and edf value has increased five times (!) in comparison to the model gam\_3. Let’s plot fitted values:

datas <- rbindlist(list(data\_r[, .(value, date\_time)],

data.table(value = gam\_4$fitted.values,

data\_time = data\_r[, date\_time])))

datas[, type := c(rep("Real", nrow(data\_r)), rep("Fitted", nrow(data\_r)))]

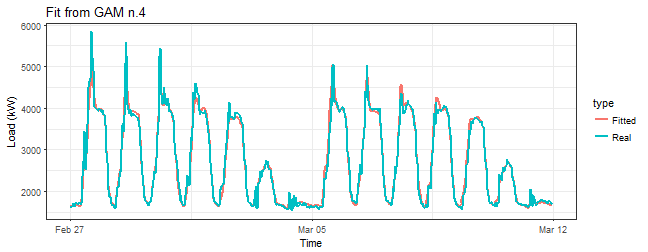
ggplot(data = datas, aes(date\_time, value, group = type, colour = type)) +

geom\_line(size = 0.8) +

theme\_bw() +

labs(x = "Time", y = "Load (kW)",

title = "Fit from GAM n.4")



This seems much better than it was with model gam\_3.

Now I will prove my statement about the upper boundary for EDF. There is a possibility to fix number of smooth basis (i.e. EDF) in smooth terms. Just use an argument fx = TRUE.

gam\_4\_fx <- gam(Load ~ te(Daily, Weekly,

k = c(period, 7),

bs = c("cr", "ps"),

fx = TRUE),

data = matrix\_gam,

family = gaussian)

summary(gam\_4\_fx)$r.sq

## [1] 0.965618

summary(gam\_4\_fx)$s.table

## edf Ref.df F p-value

## te(Daily,Weekly) 335 335 57.25389 5.289648e-199

EDF = 335, here we are. We can see that R-squared is lower than with the model gam\_4, it is due to that 335 is too high and we [overfitted](https://en.wikipedia.org/wiki/Overfitting) the model. It is prove to that GCV procedure (estimation of lambda and EDF) is working properly. You can read more about the optimal setting (choosing) of k argument at [?choose.k](https://stat.ethz.ch/R-manual/R-devel/library/mgcv/html/choose.k.html) and of course in [the book from Simon Wood](https://www.crcpress.com/Generalized-Additive-Models-An-Introduction-with-R/Wood/p/book/9781584884743).

With the mgcv package we have two more opportunities, methods, how to include tensor product interactions term - with functions ti and t2. ti produces a tensor product interaction, appropriate when the main effects (and any lower interactions) are also present, while te produces a full tensor product smooth. t2 is an alternative function to te and uses different penalization method. You can read more about it in a documentation of the package mgcv: use ?ti and ?t2 to see more.

So, let’s try both methods in the our case (model). First, let’s use ti:

gam\_5 <- gam(Load ~ s(Daily, bs = "cr", k = period) +

s(Weekly, bs = "ps", k = 7) +

ti(Daily, Weekly,

k = c(period, 7),

bs = c("cr", "ps")),

data = matrix\_gam,

family = gaussian)

summary(gam\_5)$r.sq

## [1] 0.9717469

summary(gam\_5)$sp.criterion

## GCV.Cp

## 35772.35

summary(gam\_5)$s.table

## edf Ref.df F p-value

## s(Daily) 22.583649 27.964970 444.19962 0

## s(Weekly) 5.914531 5.995934 1014.72482 0

## ti(Daily,Weekly) 85.310314 110.828814 41.22288 0

Then let’s use t2. I set argument full = TRUE because it gives strict invariance to penalties.

gam\_6 <- gam(Load ~ t2(Daily, Weekly,

k = c(period, 7),

bs = c("cr", "ps"),

full = TRUE),

data = matrix\_gam,

family = gaussian)

summary(gam\_6)$r.sq

## [1] 0.9738273

summary(gam\_6)$sp.criterion

## GCV.Cp

## 32230.68

summary(gam\_6)$s.table

## edf Ref.df F p-value

## t2(Daily,Weekly) 98.12005 120.2345 86.70754 0

I have also printed the GCV score value for the last three models, which is also a good criterion to choose optimal model among a set of fitted models. We can see that with t2 term and corresponding model gam\_6 the GCV value is the lowest. It may be indicating that gam\_6 is our best model so far.

Other model selection criterion, which is widely used in statistics, is AIC ([Akaike Information Criterion](https://en.wikipedia.org/wiki/Akaike_information_criterion)). Criterion has equation AIC=2k−2ln(^L)AIC=2k−2ln⁡(L^), where k is the number of parameters to be estimated and ^LL^ is the maximized value of the likelihood function of the model. So lower values are better for AIC. Let’s look at them for our three models:

AIC(gam\_4, gam\_5, gam\_6)

## df AIC

## gam\_4 121.4117 8912.611

## gam\_5 115.8085 8932.746

## gam\_6 100.1200 8868.628

The lowest value is in the gam\_6 model, so again it’s the winning model. Let’s again look at fitted values to be sure that everything is OK.

datas <- rbindlist(list(data\_r[, .(value, date\_time)],

data.table(value = gam\_6$fitted.values,

data\_time = data\_r[, date\_time])))

datas[, type := c(rep("Real", nrow(data\_r)), rep("Fitted", nrow(data\_r)))]

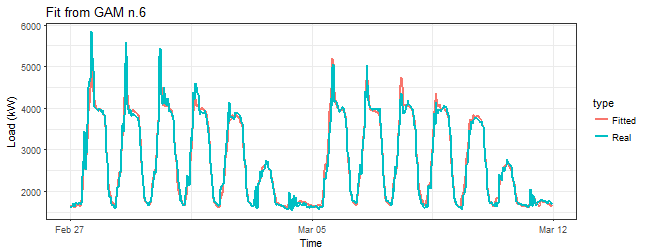
ggplot(data = datas, aes(date\_time, value, group = type, colour = type)) +

geom\_line(size = 0.8) +

theme\_bw() +

labs(x = "Time", y = "Load (kW)",

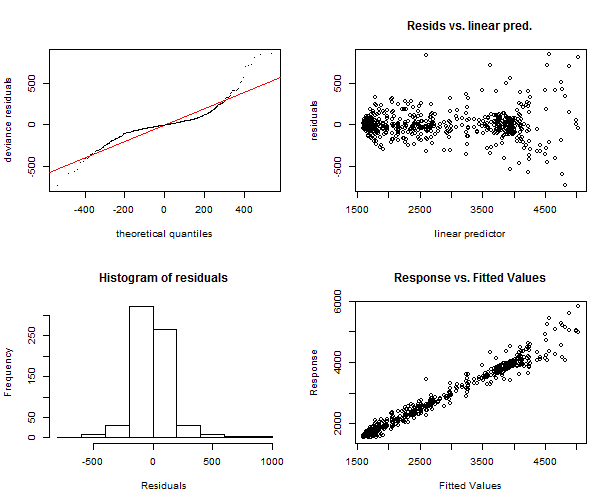
title = "Fit from GAM n.6")



Seems OK. We can see that fitted values for models gam\_4 and gam\_6 are very similar. It’s very difficult to choose which model is better. It’s possible to use more visualization and model diagnostic capabilities of package mgcv to compare these two models.

The first one is function gam.check, which makes four plots: QQ-plot of residuals, linear predictor vs. residuals, the histogram of residuals and the plot of fitted values vs. response. Let’s make them for models gam\_4 and gam\_6.

gam.check(gam\_4)



##

## Method: GCV Optimizer: magic

## Smoothing parameter selection converged after 7 iterations.

## The RMS GCV score gradiant at convergence was 0.2833304 .

## The Hessian was positive definite.

## The estimated model rank was 336 (maximum possible: 336)

## Model rank = 336 / 336

##

## Basis dimension (k) checking results. Low p-value (k-index<1) may

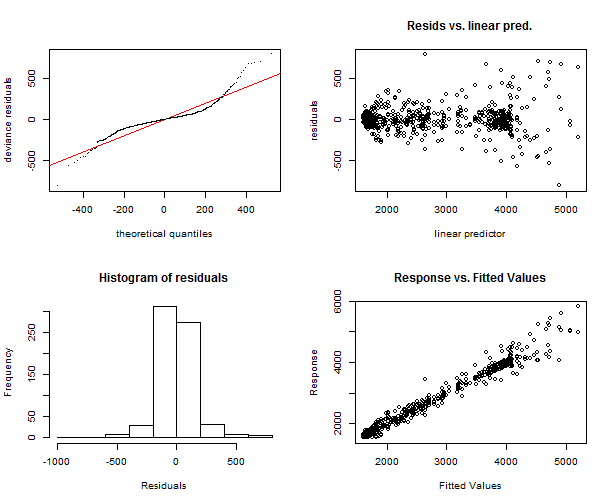
## indicate that k is too low, especially if edf is close to k'.

##

## k' edf k-index p-value

## te(Daily,Weekly) 335.00 119.41 1.22 1

gam.check(gam\_6)



##

## Method: GCV Optimizer: magic

## Smoothing parameter selection converged after 9 iterations.

## The RMS GCV score gradiant at convergence was 0.05208856 .

## The Hessian was positive definite.

## The estimated model rank was 336 (maximum possible: 336)

## Model rank = 336 / 336

##

## Basis dimension (k) checking results. Low p-value (k-index<1) may

## indicate that k is too low, especially if edf is close to k'.

##

## k' edf k-index p-value

## t2(Daily,Weekly) 335.00 98.12 1.18 1

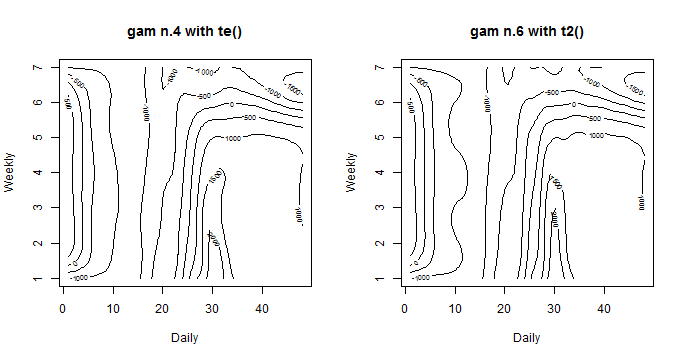
The function gam.check makes also output to the console of more useful information. We can see again that models are very similar, just in histograms some differences can be seen, but it’s also insignificant.

We didn’t use so far the default plot function to gam object for models with tensor product interactions. Let’s use it and explore what it brings us.

layout(matrix(1:2, nrow = 1))

plot(gam\_4, rug = FALSE, se = FALSE, n2 = 80, main = "gam n.4 with te()")

plot(gam\_6, rug = FALSE, se = FALSE, n2 = 80, main = "gam n.6 with t2()")

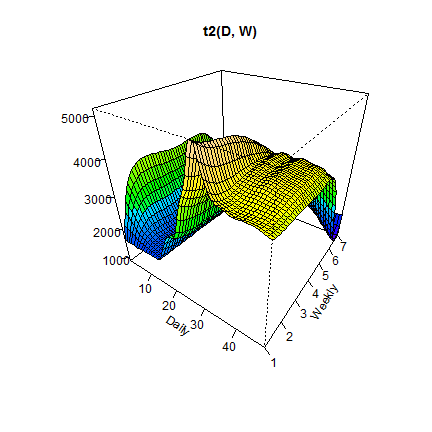


It’s a nice contour line plot. On axes are our independent variables, contour lines and corresponding numbers on them represents effects of ind. variables to the response variable. Now it is possible to see some little differences. The model gam\_6 with t2 have more “wavy” contours. So it implies that it adapts more to response variable and smoothing factor is lower. In the next parts of the post, the model gam\_6 will be used for the analysis.

Another great feature of the package mgcv is the plotting function vis.gam. It makes 3D view (or 2D) of surface of fitted values according to independent variables. Let’s look at it.

vis.gam(gam\_6, n.grid = 50, theta = 35, phi = 32, zlab = "",

ticktype = "detailed", color = "topo", main = "t2(D, W)")

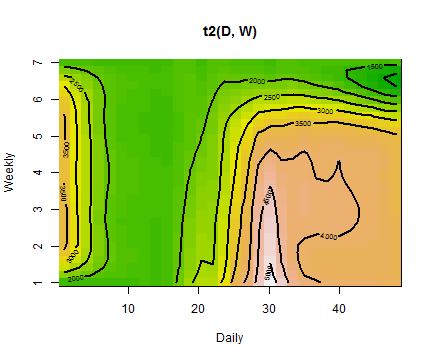


We can see that the highest peak is when the Daily variable has values near 30 (3 p.m.) and the Weekly variable has value 1 (it is a Monday).

2D view, contour lines type of plot, can be visualized too. Just set argument plot.type = "contour".

vis.gam(gam\_6, main = "t2(D, W)", plot.type = "contour",

color = "terrain", contour.col = "black", lwd = 2)



Again we can see that the highest value of electricity load is on Monday at 3:00 pm, it is very similar till Thursday, then the load is decreasing (weekends).

Another interesting contribution to the **time series regression model** can be an inclusion of [**autoregressive model**](https://en.wikipedia.org/wiki/Autoregressive_model) for serially correlated errors. Let’s look at it.

**Autoregressive models and GAMS**

**GAM** or **MLR** have assumptions in a model that errors (residuals) are identically and independently distributed (i.i.d.). In the case of the time series regression, it is very strong assumption, which is here, logically, not fulfilled. Present time series values are highly correlated with past values, so errors of the model will be correlated too. This phenomenon is called an [autocorrelation](https://en.wikipedia.org/wiki/Autocorrelation). This implies that estimated regression coefficients and residuals of a model might be negatively biased, which also implies that previously computed p-values of statistical tests or confidence intervals are wrong.

How can we handle this situation? By inclusion of autoregressive model (AR) for errors in our model. So we have the model with the term for errors like this:

yi=βX+εi,εi=ϕεi−1+vi,yi=βX+εi,εi=ϕεi−1+vi,

where the second equation is a classical AR(1) process and ϕϕ is an unknown autoregressive coefficient to be estimated. Errors can be also nested within a week, which is in our case more appropriate, because of the double seasonal character of our time series. You can add also higher orders of AR process and also MA ([moving average](https://en.wikipedia.org/wiki/Moving-average_model)) model.

It’s possible to add correlation term for errors with function gamm, which stands for **GAM** mixture models. It calls the lme function from package nlme. Now, train basic model with function gamm and an another model with AR(1) process nested within a week - just add this argument to gamm function: correlation = corARMA(form = ~ 1|Weekly, p = 1).

gam\_6\_ar0 <- gamm(Load ~ t2(Daily, Weekly,

k = c(period, 7),

bs = c("cr", "ps"),

full = TRUE),

data = matrix\_gam,

family = gaussian,

method = "REML")

gam\_6\_ar1 <- gamm(Load ~ t2(Daily, Weekly,

k = c(period, 7),

bs = c("cr", "ps"),

full = TRUE),

data = matrix\_gam,

family = gaussian,

correlation = corARMA(form = ~ 1|Weekly, p = 1),

method = "REML")

We can use an [ANOVA](https://en.wikipedia.org/wiki/Analysis_of_variance) to compare these two models and pick the right one. It does generalized likelihood ratio test to our nested models.

anova(gam\_6\_ar0$lme, gam\_6\_ar1$lme)

## Model df AIC BIC logLik Test L.Ratio p-value

## gam\_6\_ar0$lme 1 10 9070.525 9115.568 -4525.263

## gam\_6\_ar1$lme 2 11 8751.361 8800.908 -4364.680 1 vs 2 321.1644 <.0001

AIC value of the model with AR(1) term is lower, so it seems better, also p-value is very low which indicates that second model is better to choose.

Estimated ϕϕ coefficient of AR(1) process can be seen here:

intervals(gam\_6\_ar1$lme, which = "var-cov")$corStruct

## lower est. upper

## Phi 0.6363101 0.7107914 0.7721434

## attr(,"label")

## [1] "Correlation structure:"

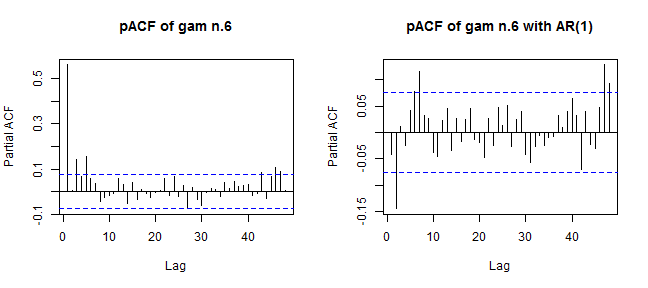
Value 0.71 is pretty high, which indicates a strong dependency on previous values of errors (lags).

Now let’s look another diagnostic for these two models. Plot of values of a [partial autocorrelation function](https://en.wikipedia.org/wiki/Partial_autocorrelation_function) applied to normalized residuals. It gives the partial correlation of residuals with its own lagged values, controlling for the values of the residuals at all shorter lags.

layout(matrix(1:2, ncol = 2))

pacf(resid(gam\_6\_ar0$lme, type = "normalized"), lag.max = 48, main = "pACF of gam n.6")

pacf(resid(gam\_6\_ar1$lme, type = "normalized"), lag.max = 48, main = "pACF of gam n.6 with AR(1)")



We can see a significant difference between these two plots and corresponding values of pACF. Optimal values of pACF should be under dashed blue lines, which isn’t completely this scenario on the right plot (model with the AR(1)). It can be better…

I will do little hack now. For optimal choose of AR(p) and MA(q) orders auto.arima function, from the library forecast, will be used on residuals from the model gam\_6\_ar0. It automatically chooses optimal orders of ARMA (in our case) based on AIC criterion. As we can use just ARMA models in gamm, so nonstationarity isn’t allowed, set an argument stationary = TRUE.

library(forecast)

arma\_res <- auto.arima(resid(gam\_6\_ar0$lme, type = "normalized"),

stationary = TRUE, seasonal = FALSE)

arma\_res$coef

## ar1 ma1 ma2

## 0.9102517 -0.3763428 -0.2862549

It’s interesting. auto.arima chooses also two orders of MA model. Let’s fit a new model and again do ANOVA.

gam\_6\_arma<- gamm(Load ~ t2(Daily, Weekly,

k = c(period, 7),

bs = c("cr", "ps"),

full = TRUE),

data = matrix\_gam,

family = gaussian,

correlation = corARMA(form = ~ 1|Weekly, p = 1, q = 2),

method = "REML")

anova(gam\_6\_ar0$lme, gam\_6\_ar1$lme, gam\_6\_arma$lme)

## Model df AIC BIC logLik Test L.Ratio

## gam\_6\_ar0$lme 1 10 9070.525 9115.568 -4525.263

## gam\_6\_ar1$lme 2 11 8751.361 8800.908 -4364.680 1 vs 2 321.1644

## gam\_6\_arma$lme 3 13 8731.750 8790.306 -4352.875 2 vs 3 23.6107

## p-value

## gam\_6\_ar0$lme

## gam\_6\_ar1$lme <.0001

## gam\_6\_arma$lme <.0001

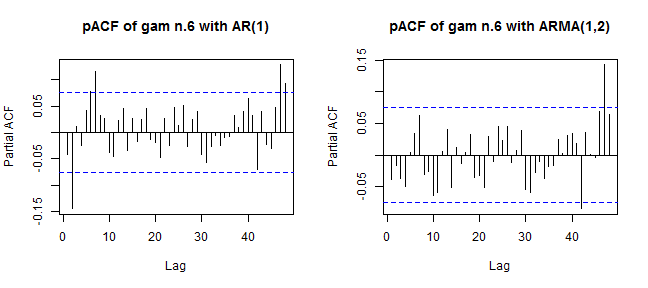
The new model gam\_6\_arma is slightly better in the meaning of AIC. P-value is again very small, which means that gam\_6\_arma is significantly better than gam\_6\_ar1.

Now do a comparison of these two models on theirs pACF values.

layout(matrix(1:2, ncol = 2))

pacf(resid(gam\_6\_ar1$lme, type = "normalized"), lag.max = 48, main = "pACF of gam n.6 with AR(1)")

pacf(resid(gam\_6\_arma$lme, type = "normalized"), lag.max = 48, main = "pACF of gam n.6 with ARMA(1,2)")



It’s better now. Can we be happy? Up to now, everything seems fine.

Let’s make one more visualization of residuals of two models to confirm our little uncertainty about the correctness of the model with ARMA(1,2). Compare it with baseline model gam\_6\_ar0.

datas <- data.table(Fitted\_values = c(gam\_6\_ar0$gam$fitted.values,

gam\_6\_arma$gam$fitted.values),

Residuals = c(gam\_6\_ar0$gam$residuals,

gam\_6\_arma$gam$residuals),

Model = rep(c("Gam n.6", "Gam n.6 with ARMA(1,2)"), each = nrow(data\_r)))

ggplot(data = datas,

aes(Fitted\_values, Residuals)) +

facet\_grid(Model~., switch = "y") +

geom\_point(size = 1.7) +

geom\_smooth(method = "loess") +

geom\_hline(yintercept = 0, color = "red", size = 1) +

theme(panel.border = element\_rect(fill = NA, color = "black"),

plot.title = element\_text(size = 14, hjust = 0.5, face = "bold"),

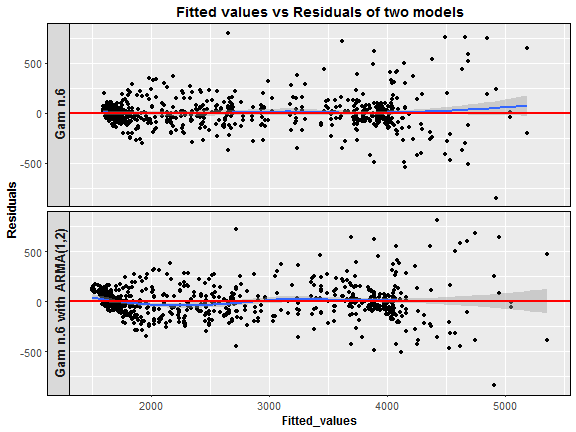
axis.text = element\_text(size = 11),

axis.title = element\_text(size = 12, face = "bold"),

strip.text = element\_text(size = 13, face = "bold"),

strip.background = element\_rect(color = "black")) +

labs(title = "Fitted values vs Residuals of two models")



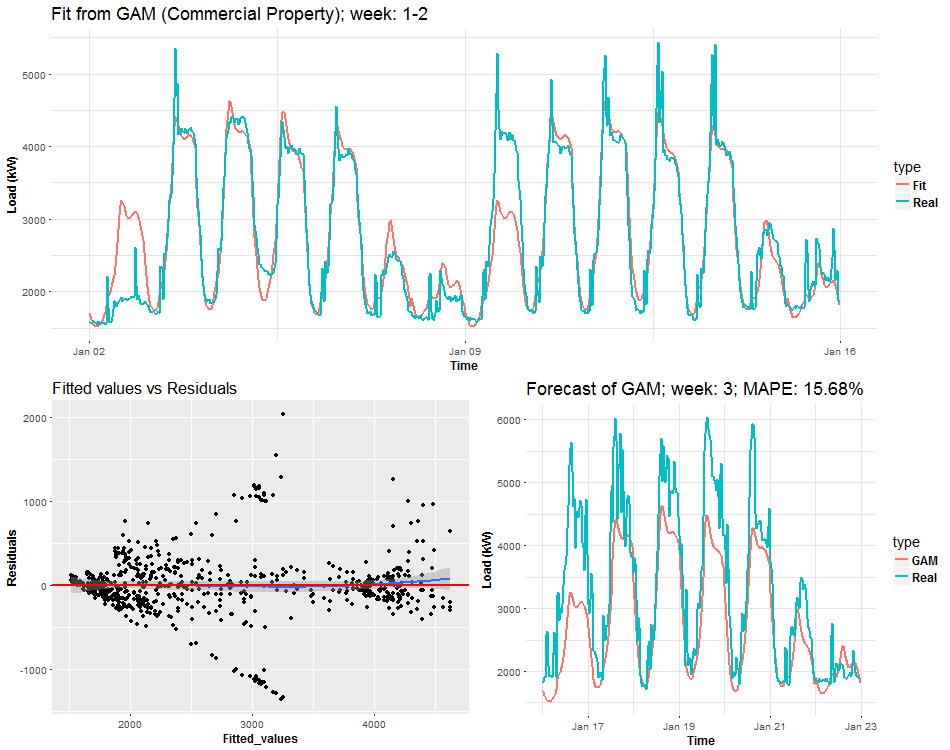
Here we are. The model with an ARMA(1,2) process has residuals at low fitted values somehow correlated, which is not good (heteroscedasticity). What now? I have to be honest, I can’t explain why this interesting thing happened. So, I would be very happy when someone guides me to the right direction and writes some notes to the discussion, it would really help. One more note about the inclusion of ARMA models to **GAM** or **MLR**. It didn’t help in the meaning of forecast accuracy, I made a lot of experiments, but MAPE was higher than with models without ARMAs. The big disadvantage is also higher time complexity than with the simple model without ARMA.

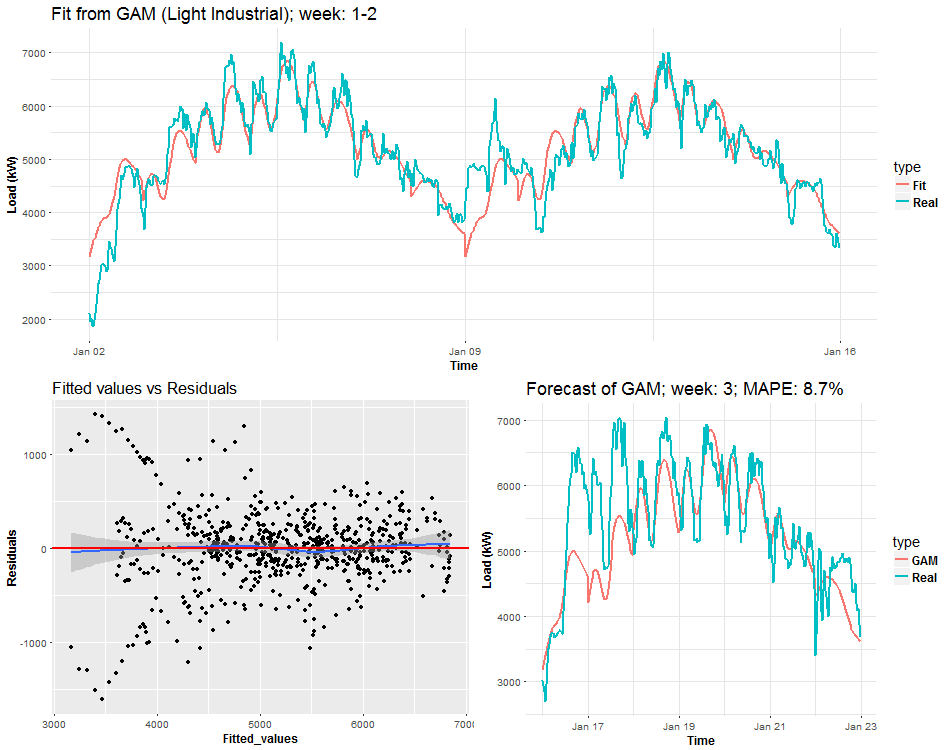
**Animations and conclusions**

I will end this post with a visualization analysis of the model gam\_6. I will try to do some animated dashboards of main characteristics of the model thru time. It should be very interesting how electricity load changes and how the model can adapt to these changes. In the first two animations you can see three plots on the one figure:

1. Fitted values by GAM and real response values of electricity consumption.
2. Fitted values vs. residuals of GAM.
3. Forecast for one week ahead vs. real consumption plus forecast accuracy in MAPE.

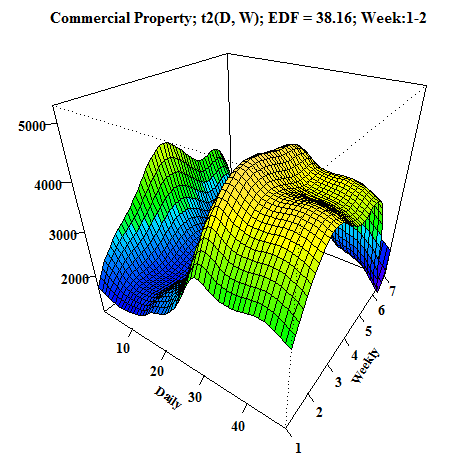
Animations were created by functions from package grid to layout ggplots to one figure, and function saveGIF from package animation to create the final GIF. Here there are:





We can see the behavior of **GAMs** on two times series from two different types of industries: commercial property and light industrial. Notice (look), that fitted values and forecasts are smoother than where were with Multiple linear regression from the previous post. It’s good information to know, it implies that our model isn’t overfitted. On the other hand, I also compared forecast performance of **GAM** and **MLR**, but GAM was slightly worse than MLR, which can be a little bit surprising for somebody. Seems that usage of penalized least squares (GAM) for forecasting time series doesn’t bring improvement against classical OLS (MLR). But there are other advantages, which I will point out in the end.

Another animation, which I created, is a 3D visualization of fitted values like a surface. As I showed you before, it can be done very easily by function vis.gam.



We can compare it with the first dashboard because again it’s time series from commercial properties. EDF is also printed to the title of GIF for reasons to see how it changes depending on the behavior of electricity consumption.

With this, I would like to end the main part of this tutorial and conclude it with some remarks.

In the beginning of the post, motivation and the theory behind GAM method was introduced. Next, the analytical (“magical”) part continued with explaining various features of the package mgcv. Different types of **interactions** were showed and analyzed - best chose was tensor product interactions (t2). Next, consideration about the correctness of an autoregressive model (or ARMA) for errors was discussed - ended with an open question. In the end, I tried to make a big view of a behavior of **GAMs** by animations of its performance on **double seasonal time series**.

Everything that was said implies these advantages and disadvantages of using **GAM** for your problem:

Advantages:

* Many possibilities how to model your independent variables (many types of smooth functions).
* Model response variable by all known distribution families (Normal, Gamma, Poisson etc.).
* Define interactions by tensor product, which means usage of different basis functions for interacted variables.
* Test statistical significance of a non-linear relationship of the independent variable to dependent.

Disadvantages:

* Complex method, difficult to learn.
* It’s sometimes difficult to interpret results.
* Several parameters to tune, which have to be set carefully.

However, the number of weekly seasonal regression coefficients will be only 6, because the number of the second seasonality coefficients is set to freq\_2 / freq\_1, so 48\*7 / 48 = 7, and 7 – 1 = 6 because of splines computation (differentiation). I will again use repr\_matrix function.

data\_gam <- repr\_matrix(elec\_load, func = repr\_gam, args = list(freq = c(48, 48\*7)),

normalise = TRUE, func\_norm = norm\_z)

dim(data\_gam)

## [1] 50 53

So the dimension is 47 + 6 = 53 because of usage of splines in GAM method. Let’s cluster the data and visualize results of it.

clusterings <- lapply(c(2:7), function(x)

pam(data\_gam, x))

DB\_values <- sapply(seq\_along(clusterings), function(x)

intCriteria(data\_gam, as.integer(clusterings[[x]]$clustering),

c("Davies\_Bouldin")))

Let’s plot results of internal evaluation.

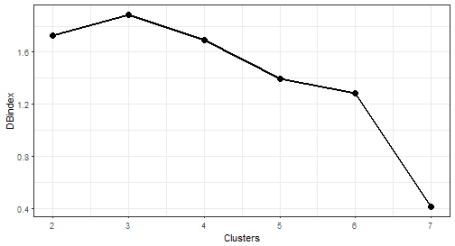
ggplot(data.table(Clusters = 2:7, DBindex = unlist(DB\_values)),

aes(Clusters, DBindex)) +

geom\_line(size = 1) +

geom\_point(size = 3) +

theme\_bw()



The optimal number of clusters is again 7. Let’s plot results.

# prepare data for plotting

data\_plot <- data.table(melt(data.table(class = as.factor(clusterings[[6]]$clustering),

data\_gam)))

data\_plot[, Time := rep(1:ncol(data\_gam), each = nrow(data\_gam))]

data\_plot[, ID := rep(1:nrow(data\_gam), ncol(data\_gam))]

# prepare medoids

centers <- data.table(melt(clusterings[[6]]$medoids))

setnames(centers, c("Var1", "Var2"), c("class", "Time"))

centers[, ID := class]

# plot the results

ggplot(data\_plot, aes(Time, value, group = ID)) +

facet\_wrap(~class, ncol = 2, scales = "free\_y") +

geom\_line(color = "grey10", alpha = 0.65) +

geom\_line(data = centers, aes(Time, value),

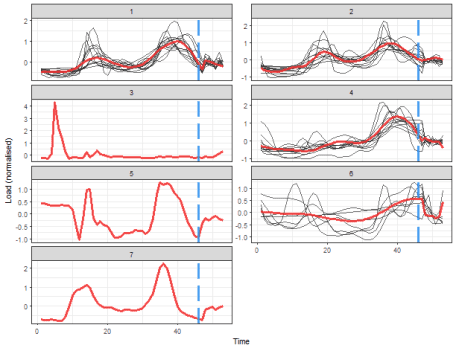
color = "firebrick1", alpha = 0.80, size = 1.2) +

geom\_vline(xintercept = 46, color = "dodgerblue2",

size = 1.4, linetype = 5, alpha = 0.8) +

labs(x = "Time", y = "Load (normalised)") +

theme\_bw()



That is a more delightful result, isn’t it? Extracted consumption profiles are smoother than in the case of average seasonal profiles. The blue dashed line borders daily and weekly seasonal coefficients.  
The K-medoids now extracted 4 typical profiles and determined 3 one-element clusters.

I will show you also result of clustering of some nondata adaptive representation, let’s pick for example **DFT** (Discrete Fourier Transform) method and extract first 48 DFT coefficients.

data\_dft <- repr\_matrix(elec\_load, func = repr\_dft, args = list(coef = 48),

normalise = TRUE, func\_norm = norm\_z)

dim(data\_dft)

## [1] 50 48

clusterings <- lapply(c(2:7), function(x)

pam(data\_dft, x))

DB\_values <- sapply(seq\_along(clusterings), function(x)

intCriteria(data\_dft, as.integer(clusterings[[x]]$clustering),

c("Davies\_Bouldin")))

Let’s plot results of internal evaluation.

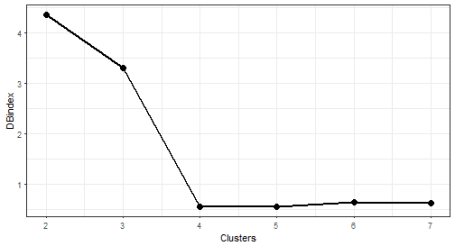
ggplot(data.table(Clusters = 2:7, DBindex = unlist(DB\_values)),

aes(Clusters, DBindex)) +

geom\_line(size = 1) +

geom\_point(size = 3) +

theme\_bw()



We can see nice [“elbow”](https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set#The_elbow_method) here at 4 number of clusters.

# prepare data for plotting

data\_plot <- data.table(melt(data.table(class = as.factor(clusterings[[3]]$clustering),

data\_dft)))

data\_plot[, Time := rep(1:ncol(data\_dft), each = nrow(data\_dft))]

data\_plot[, ID := rep(1:nrow(data\_dft), ncol(data\_dft))]

# prepare medoids

centers <- data.table(melt(clusterings[[3]]$medoids))

setnames(centers, c("Var1", "Var2"), c("class", "Time"))

centers[, ID := class]

# plot the results

ggplot(data\_plot, aes(Time, value, group = ID)) +

facet\_wrap(~class, ncol = 2, scales = "free\_y") +

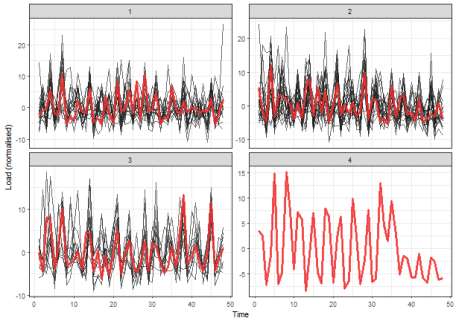
geom\_line(color = "grey10", alpha = 0.65) +

geom\_line(data = centers, aes(Time, value),

color = "firebrick1", alpha = 0.80, size = 1.2) +

labs(x = "Time", y = "Load (normalised)") +

theme\_bw()



The interpretability of these results would be difficult. Therefore, model-based time series representations are very effective in this use case (so the typical profiles extraction).

I will show you the usage of one more representation method – **FeaClip**. The **FeaClip** is feature extraction method from a clipping representation. The windowing approach alongside FeaClip is recommended to use for every day of time series. The big advantage is that normalisation is not needed alongside FeaClip method. Let’s use it in our case. The windowing method is also directly implemented in repr\_matrix function.

data\_feaclip <- repr\_matrix(elec\_load, func = repr\_feaclip,

windowing = TRUE, win\_size = 48)

dim(data\_feaclip)

## [1] 50 112

clusterings <- lapply(c(2:7), function(x)

pam(data\_feaclip, x))

DB\_values <- sapply(seq\_along(clusterings), function(x)

intCriteria(data\_feaclip, as.integer(clusterings[[x]]$clustering),

c("Davies\_Bouldin")))

Let’s plot results of internal evaluation.

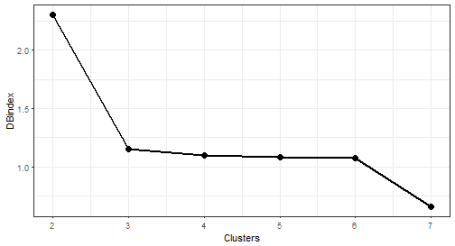
ggplot(data.table(Clusters = 2:7, DBindex = unlist(DB\_values)),

aes(Clusters, DBindex)) +

geom\_line(size = 1) +

geom\_point(size = 3) +

theme\_bw()



We can see that now 2 “elbows” appeared. The biggest change is between 2 and 3 number of clusters, so I will choose number 3.

# prepare data for plotting

data\_plot <- data.table(melt(data.table(class = as.factor(clusterings[[2]]$clustering),

data\_feaclip)))

data\_plot[, Time := rep(1:ncol(data\_feaclip), each = nrow(data\_feaclip))]

data\_plot[, ID := rep(1:nrow(data\_feaclip), ncol(data\_feaclip))]

# prepare medoids

centers <- data.table(melt(clusterings[[2]]$medoids))

setnames(centers, c("Var1", "Var2"), c("class", "Time"))

centers[, ID := class]

# plot the results

ggplot(data\_plot, aes(Time, value, group = ID)) +

facet\_wrap(~class, ncol = 2, scales = "free\_y") +

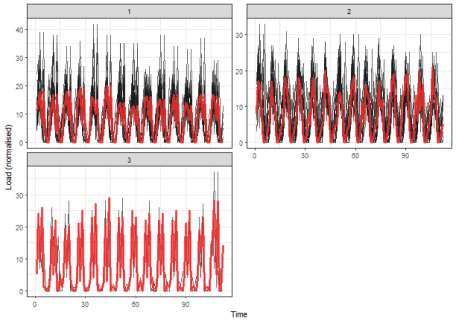
geom\_line(color = "grey10", alpha = 0.65) +

geom\_line(data = centers, aes(Time, value),

color = "firebrick1", alpha = 0.80, size = 1.2) +

labs(x = "Time", y = "Load (normalised)") +

theme\_bw()



It seems like good separability of clusters, better than in the case of DFT.  
But you can check also other results of clusterings with the different number of clusters.

**Conclusion**

In this tutorial, I showed you the usage of time series representation methods to create more characteristic profiles of consumers.Then, time series representations, calculated by **TSrepr** package, were clustered by K-medoids and typical consumption profiles were extracted from created clusters.

Time series representations can be helpful also in other use cases as classification or time series indexing.