**Data Source**

The simulated data consists  of three fields:

* *DevEUI*: Unique identifier of the digital water meter.
* *Timestamp* in [Coordinated Universal Time](https://en.wikipedia.org/wiki/Coordinated_Universal_Time)
* Cumulative *count* of pulses.

All analysis is undertaken in the local Australian Eastern Standard Time (AEST). The input to all functions is thus in AEST. The digital water meters send an hourly pulse at a random time within the hour. Each transmitter (RTU) uses a random offset to avoid network congestion. The digital meter counts each time the impeller makes a full turn, and for this analysis, we assume that this equates to a five-litre volume. The ratio between volume and count depends on the meter brand and type. The image below shows a typical data set for an RTU, including some missing data points.

Simulated water consumption (red: measured points, blue: interpolated points.

To analyse the data we need two auxiliary functions: one to slice the data we need and one to interpolate data for the times we need it. The Tidyverse heavily influences the code in this article. I like the Tidyverse way of doing things because it leads to elegant code that is easy to understand.

library(tidyverse)

library(lubridate)

library(magrittr)

meter\_reads <- read.csv("Hydroinformatics/DigitalMetering/meter\_reads.csv")

rtu <- unique(meter\_reads$DevEUI)

meter\_reads$TimeStampUTC <- as.POSIXct(meter\_reads$TimeStampUTC, tz = "UTC")

**Slicing Digital Water Metering Data**

Data analysis is undertaken on slices of the complete data set. This function slices the available data by a vector of RTU ids and a timestamp range in AEST. This function adds a new timestamp variable in AEST. If no date range is provided, all available data for the selected RTUs is provided. The output of this function is a data frame (a Tibble in Tydiverse language).

slice\_reads <- function(rtus, dates = range(meter\_reads$TimeStampUTC)) { filter(meter\_reads, DevEUI %in% rtus) %>%

mutate(TimeStampAEST = as.POSIXct(format(TimeStampUTC, tz = "Australia/Melbourne"))) %>%

filter(TimeStampAEST >= as.POSIXct(dates[1]) &

TimeStampAEST <= as.POSIXct(dates[2])) %>%

arrange(DevEUI, TimeStampAEST)

}

**Interpolation of Meter Reads**

This function interpolates the cumulative counts for a series of RTUs over a vector of timestamps in AEST. The function creates a list to store the results for each RTU, interpolates the data using the approx function and then flattens the list back to a data frame. The interpolation function contains a different type of pipe because of the approx for interpolation function does not take a data argument. The %$% pipe from the Magrittr package solves that problem.

Approx Function Example

require(graphics)

x <- 1:10

y <- rnorm(10)

par(mfrow = c(2,1))

plot(x, y, main = "approx(.) and approxfun(.)")

points(approx(x, y), col = 2, pch = "\*")

points(approx(x, y, method = "constant"), col = 4, pch = "\*")

f <- approxfun(x, y)

curve(f(x), 0, 11, col = "green2")

points(x, y)

is.function(fc <- approxfun(x, y, method = "const")) # TRUE

curve(fc(x), 0, 10, col = "darkblue", add = TRUE)

## different extrapolation on left and right side :

plot(approxfun(x, y, rule = 2:1), 0, 11,

col = "tomato", add = TRUE, lty = 3, lwd = 2)

### Treatment of 'NA's -- are kept if na.rm=FALSE :

xn <- 1:4

yn <- c(1,NA,3:4)

xout <- (1:9)/2

## Default behavior (na.rm = TRUE): NA's omitted; extrapolation gives NA

data.frame(approx(xn,yn, xout))

data.frame(approx(xn,yn, xout, rule = 2))# -> \*constant\* extrapolation

## New (2019-2020) na.rm = FALSE: NA's are "kept"

data.frame(approx(xn,yn, xout, na.rm=FALSE, rule = 2))

data.frame(approx(xn,yn, xout, na.rm=FALSE, rule = 2, method="constant"))

## NA's in x[] are not allowed:

stopifnot(inherits( try( approx(yn,yn, na.rm=FALSE) ), "try-error"))

## Give a nice overview of all possibilities rule \* method \* na.rm :

## ----------------------------- ==== ====== =====

## extrapolations "N":= NA; "C":= Constant :

rules <- list(N=1, C=2, NC=1:2, CN=2:1)

methods <- c("constant","linear")

ry <- sapply(rules, function(R) {

sapply(methods, function(M)

sapply(setNames(,c(TRUE,FALSE)), function(na.)

approx(xn, yn, xout=xout, method=M, rule=R, na.rm=na.)$y),

simplify="array")

}, simplify="array")

names(dimnames(ry)) <- c("x = ", "na.rm", "method", "rule")

dimnames(ry)[[1]] <- format(xout)

ftable(aperm(ry, 4:1)) # --> (4 \* 2 \* 2) x length(xout) = 16 x 9 matrix

## Show treatment of 'ties' :

x <- c(2,2:4,4,4,5,5,7,7,7)

y <- c(1:6, 5:4, 3:1)

(amy <- approx(x, y, xout = x)$y) # warning, can be avoided by specifying 'ties=':

op <- options(warn=2) # warnings would be error

stopifnot(identical(amy, approx(x, y, xout = x, ties=mean)$y))

(ay <- approx(x, y, xout = x, ties = "ordered")$y)

stopifnot(amy == c(1.5,1.5, 3, 5,5,5, 4.5,4.5, 2,2,2),

ay == c(2, 2, 3, 6,6,6, 4, 4, 1,1,1))

approx(x, y, xout = x, ties = min)$y

approx(x, y, xout = x, ties = max)$y

options(op) # revert 'warn'ing level

The output is a data frame with DevEUI, the timestamp in AEST and the interpolated cumulative count. The image above shows the counts for two meters over two days an the graph superimposes an interpolated point over the raw data. Although the actual data consists of integer counts, interpolated values are numeric values. The decimals are retained to distinguish them from real reads.

Library(Magrittr)

interpolate\_count <- function(rtus, timestamps) {

timestamps <- as.POSIXct(timestamps, tz = "Australia/Melbourne")

results <- vector("list", length(rtus))

for (r in seq\_along(rtus)) {

interp <- slice\_reads(rtus[r]) %$%

approx(TimeStampAEST, Count, timestamps)

results[[r]] <- data\_frame(DevEUI = rep(rtus[r], length(timestamps)), TimeStampAEST = timestamps, Count = interp$y) } return(do.call(rbind, results)) } interpolate\_count(rtu[2:3], seq.POSIXt(as.POSIXct("2020-02-01"), as.POSIXct("2020-02-2"), by = "day")) slice\_reads(rtu[2], c("2020-02-06", "2020-02-08")) %>%

ggplot(aes(x = TimeStampAEST, y = Count)) +

geom\_line(col = "grey", size = 1) +

geom\_point(col = "red") +

geom\_point(data = interpolate\_count(rtu[2], as.POSIXct("2020-02-06") + (0:2)\*24\*3600), colour = "blue") +

ggtitle(paste("DevEUI", rtu[2]))

With these two auxiliary functions, we can start analysing the data.

**Daily Consumption**

Daily consumption for each connection is a critical metric in managing water resources and billing customers. The daily consumption of any water connection is defined by the difference between the cumulative counts at midnight. The interpolation function makes it easy to determine daily consumption. This function interpolates the midnight reads for each of the RTUs over the period, starting the previous day. The output of the function is a data frame that can be piped into the plotting function to visualise the data. When you group the data by date, you can also determine the total consumption over a group of services.

daily\_consumption <- function(rtus, dates) {

timestamps <- seq.POSIXt(as.POSIXct(min(dates)) - 24 \* 3600, as.POSIXct(max(dates)), by = "day") interpolate\_count(rtus, timestamps) %>%

group\_by(DevEUI) %>%

mutate(Consumption = c(0, diff(Count)) \* 5,

Date = format(TimeStampAEST, "%F")) %>%

filter(TimeStampAEST != timestamps[1]) %>%

select(DevEUI, Date, Consumption)

}

daily\_consumption(rtu[32:33], c("2020-02-01", "2020-02-7")) %>%

ggplot(aes(x = Date, y = Consumption)) + geom\_col() +

facet\_wrap(~DevEUI) +

theme(axis.text.x = element\_text(angle = 90, hjust = 1))

Analysing digital water meter data: Daily consumption.

**Diurnal Curves**

The diurnal curve is one of the most important pieces of information used in the design of water supply systems. This curve shows the usage of one or more services for each hour in the day. This curve is a reflection of human behaviour, as we use most water in the morning and the evenings.

This function slices data for a vector of RTUs over a period and then plots the average diurnal curve. The data is obtained by interpolating the cumulative counts for each whole hour in the period. The function then calculates the flow in litres per hour and visualises the minimum, mean and maximum value.

plot\_diurnal\_connections <- function(rtus, dates) {

timestamps <- seq.POSIXt(as.POSIXct(dates[1]), as.POSIXct(dates[2]), by = "hour") interpolate\_count(rtus, timestamps) %>%

mutate(Rate = c(0, diff(Count \* 5)),

Hour = as.integer(format(TimeStampAEST, "%H"))) %>%

filter(Rate >= 0) %>%

group\_by(Hour) %>%

summarise(min = min(Rate), mean = mean(Rate), max = max(Rate)) %>%

ggplot(aes(x = Hour, ymin = min, ymax = max)) +

geom\_ribbon(fill = "lightblue", alpha = 0.5) +

geom\_line(aes(x = Hour, y = mean), col = "orange", size = 1) +

ggtitle("Connections Diurnal flow") + ylab("Flow rate [L/h]")

}

plot\_diurnal\_connections(rtu[12:20], c("2020-02-01", "2020-03-01"))

Analysing digital water meter data: Diurnal curve.

Boxplots are also an informative way to visualise this curve. This method provides more statistical information on one page, and the ggplot function does all the statistical analysis.

plot\_diurnal\_box <- function(rtus, dates) {

timestamps <- seq.POSIXt(as.POSIXct(dates[1]), as.POSIXct(dates[2]), by = "hour") interpolate\_count(rtus, timestamps) %>%

mutate(Rate = c(0, diff(Count \* 5)),

Hour = as.integer(format(TimeStampAEST, "%H"))) %>%

filter(Rate >= 0) %>%

group\_by(Hour) %>%

ggplot(aes(x = factor(Hour), y = Rate)) +

geom\_boxplot() +

ggtitle("Diurnal flow") + ylab("Flow rate [L/h]") + xlab("Time")

}

plot\_diurnal\_box(rtu[12:20], c("2020-02-01", "2020-03-01"))