Water demand and network modelling with R

Bradley J. Eck

June 2018

Please cite this as: Eck, B. (2018) "Water demand and network modelling with R." *9th International Congress on Environmental Modelling and Software.* Fort Collins, Colorado, USA.

N.B. This version contains some slight updates compared to the conference proceedings.

Abstract

Internet of things sensors for water demand contribute to the big data water modelers can use to characterize and forecast demands at increasingly finer scales in space and time. Popular modeling techniques include artificial neural networks and autoregressive models but other methods such as deep learning and Gaussian mixture models are also in use. Leveraging these demand models to optimize planning and operational decisions often requires incorporating demands into simulations of system behavior. This paper describes solutions for water network simulations and demand modeling in the R environment. A key component is the recently released epanet2toolkit for water network simulations. Examples are provided for using the package for three tasks related to demand modeling. First, we update a network with forecasted demands. Next, we optimize the updated model to minimize energy use. Finally, we simulate system behavior driven by uncertainty in demand forecasts using Monte Carlo methods. The fact that these examples require only modest code writing should motivate researchers and practitioners to incorporate more demand data into operational and planning decisions.

1. Introduction

Modern software for data science crosses barriers between research disciplines by making techniques from many domains accessible on a common platform. In particular, the R environment for computing and graphics (R Core Team, 2013) is notable for the variety of add-on packages that provide specialized capability. After gaining basic familiarity with R, these packages enable workers to access sophisticated methods for analysis by invoking high-level functions from a familiar environment.

The community working in environmental software, especially software for urban water systems, has been adopting R packages as a format to build and share tools. The epanetReader package (Eck, 2016) for parsing EPANET files has been available since 2015. The swmmr package (Leutnant 2017) for using the storm water management model has been available since 2017. A new package, epanet2toolkit (Arandia and Eck, 2018), was recently added to this group. A more comprehensive survey of water related packages for R is provided by Arandia and Eck (2018).

The EPANET related packages epanetReader and epanet2toolkit provide distinct but complementary capability for analyzing water networks. epanetReader parses files in EPANET's 'inp' and 'rpt' formats into R. The 'inp' file specifies the structure of a network for simulation. The 'rpt' file records the results of a simulation. epanetReader enables analysis and visualization of water networks by importing data from these files into R. In contrast, epanet2toolkit provides the EPANET simulation engine and programmer's toolkit as R functions. These additional functions enable users to carry out standard or customized simulations from within R. epanet2toolkit makes it possible to connect the EPANET simulation engine with other functions in the R environment. This paper illustrates several such connections that are relevant for water demand modeling.

The remainder of the paper is organized as follows. First, the installation and basic usage of epanet2toolkit are described. Next, a sample time series of water demand is used to forecast demand one day ahead. The predicted demands are then pushed to the EPANET simulation engine for a sample network. The energy usage of the network is minimized by changing the pumping schedule using a genetic algorithm. Finally, the distribution of energy cost considering uncertainty in the demand forecast is estimated using Monte Carlo simulations. This series of small, linked, examples shows a few ways practitioners can incorporate information on water demand in system planning and operation.

2. Package Installation

Add-on packages for R can be installed over the internet from the Comprehensive R Archive Network (CRAN). Packages distributed on CRAN pass a series of tests to ensure that packages will install on different systems, that all user functions are documented, and that code examples in the documentation execute properly. This paper relies on add-on functionality provided by four packages: epanetReader, epanet2toolkit, forecast, and GA. epanetReader and epanet2toolkit are described in the introduction. The forecast package provides forecasting functions for time series and linear models (Hyndman and Khandakar, 2008). This paper uses the package to make water demand forecasts. The GA package provides genetic algorithms for optimization. This paper uses a genetic algorithm to minimize energy usage. The packages can be installed by issuing the following command within an R session:

```r
install.packages(c("epanetReader","epanet2toolkit","forecast","GA"))
```

After installation, packages must be loaded into the current R session using the library function. After loading, the functions, data, and documentation contained in a package become available to the user.

```r
library(epanetReader)
library(epanet2toolkit)
library(forecast)
```

```
## This is forecast 8.3
##   Stackoverflow is a great place to get help on R issues:
##   http://stackoverflow.com/tags/forecasting+r.
```

```r
library(GA)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
##   ____    _
##  / ___|  / \      Genetic
## | |  _  / _ \     Algorithms
## | |_| |/ ___ \
##  \____/_/   \_\  version 3.1.1
## Type 'citation("GA")' for citing this R package in publications.
```

3. Basic Usage of epanet2toolkit

epanet2toolkit is an add-on package for R that provides EPANET's programmer's toolkit as R functions. The simulation engine version 2.1 from Open Water Analytics ships with the package, avoiding the need to compile it for your platform or put the dll/so in the search path for your operating system. Error checking is integrated with the R exception system and happens for every call to EPANET. Most, but not all of the EPANET functions from v2.1 are supported. Every function provided by the package has a documentation page accessed using the help or ? function.

```r
?epanet2toolkit
help(ENopen)
```

To illustrate the basic usage of epanet2toolkit, this section uses EPANET's example network 1, which is included with the package. Example network 1 contains one pump, one tank, one reservoir, 12 pipes and nine demand nodes. It is often used to demonstrate the essential capabilities of EPANET. The path to the network file is stored for use in subsequent commands.

```r
inp <- file.path( find.package("epanet2toolkit"), "extdata","Net1.inp")
```

3.1 Running a full simulation

A full simulation of the network is invoked with ENepanet. After the simulation completes, the results are stored in the file specified in the second argument, in this case "Net1.rpt", in the working directory.

```r
ENepanet(inp, "Net1.rpt")
```

The results can be viewed using a text editor and read into R using the package epanetReader.

3.2 Querying network properties

Network properties can be accessed using functions beginning with ENget. Note that the toolkit needs to be opened with ENopen before other functions are called and closed with ENclose upon completion.

```r
ENopen(inp, "Net1.rpt")
ENgetflowunits()
```

```
## EN_GPM
##      1
```

```r
ENgetqualtype()
```

```
## $qualcode
## [1] 1
##
## $tracenode
## [1] 0
```

```r
ENgetcount("EN_NODECOUNT")
```

```
## [1] 11
```

```r
ENgetcount("EN_LINKCOUNT")
```

```
## [1] 13
```

```r
ENgetnodeid(1)
```

```
## [1] "10"
```

```r
ENgetlinkid(1)
```

```
## [1] "10"
```

```r
ENclose()
```

3.3 Setting network properties

Setting network properties functions in a similar way using functions beginning with ENset.

```r
ENopen(inp, "Net1.rpt")
ENgetnodevalue(3, "EN_ELEVATION")
```

```
## [1] 700
```

```r
ENsetnodevalue(3, "EN_ELEVATION", 777)
ENgetnodevalue(3, "EN_ELEVATION")
```

```
## [1] 777
```

```r
ENclose()
```

4 Forecasting Water Demands

Forecasting water demands is an essential part of planning and operating water distribution systems. Forecasting can be done using Gaussian Mixture Models (McKenna et al. 2013) or auto-regressive models (Arandia et al. 2015) or other methods. In this paper we use an exponential smoothing state space model with Box-Cox transformation, Auto-regressive moving average errors, and components for Trend and Season (BATS). The BATS model of De Livera et al. (2011) is provided with the forecast package (Hyndman and Khandakar, 2008). Half-hourly demand data for an urban water user are read from a file. A BATS model is fit considering daily seasonality. A forecast of demands for the next day is created and visualized (Figure 1).

```r
hhd <- scan("half_hourly_demands.txt", numeric())
fit <- bats(hhd, seasonal.periods = 48)
day_ahead <- forecast(fit, h=48)
```

```r
plot(day_ahead, xlab = "day", ylab = "Water Demand")
```

5 Updating a Network with Forecasted Demands

Forecasted demands can be used within EPANET by changing the demand pattern for a junction. This example uses the Richmond test network (Van Zyl et al. 2004) as modified by Ghaddar et al. (2015), and further modified to use a new demand pattern called forecast for node 1302. This node has the largest demand in the system.

EPANET's API operates on element indices so the first step after opening EPANET is to extract the index of the node in question. Node 1302 is seen to have a base demand of 1.

```r
ENopen("richmond-modified.inp", "rich.rpt")
myNodeIndex <- ENgetnodeindex("1302")
ENgetnodevalue(myNodeIndex, "EN_BASEDEMAND")
```

```
## [1] 1
```

```r
ENclose()
```

To change the demand at the node, we store the forecast demand in the pattern. Updating the pattern is encapsulated in a function to facilitate repeated usage.

```r
update_pattern <- function( inp_file, pattern_id, new_pattern){
  ENopen( inp_file, "temp.rpt")
  pattern_index <- ENgetpatternindex(pattern_id)
  ENsetpattern( pattern_index, new_pattern)
```

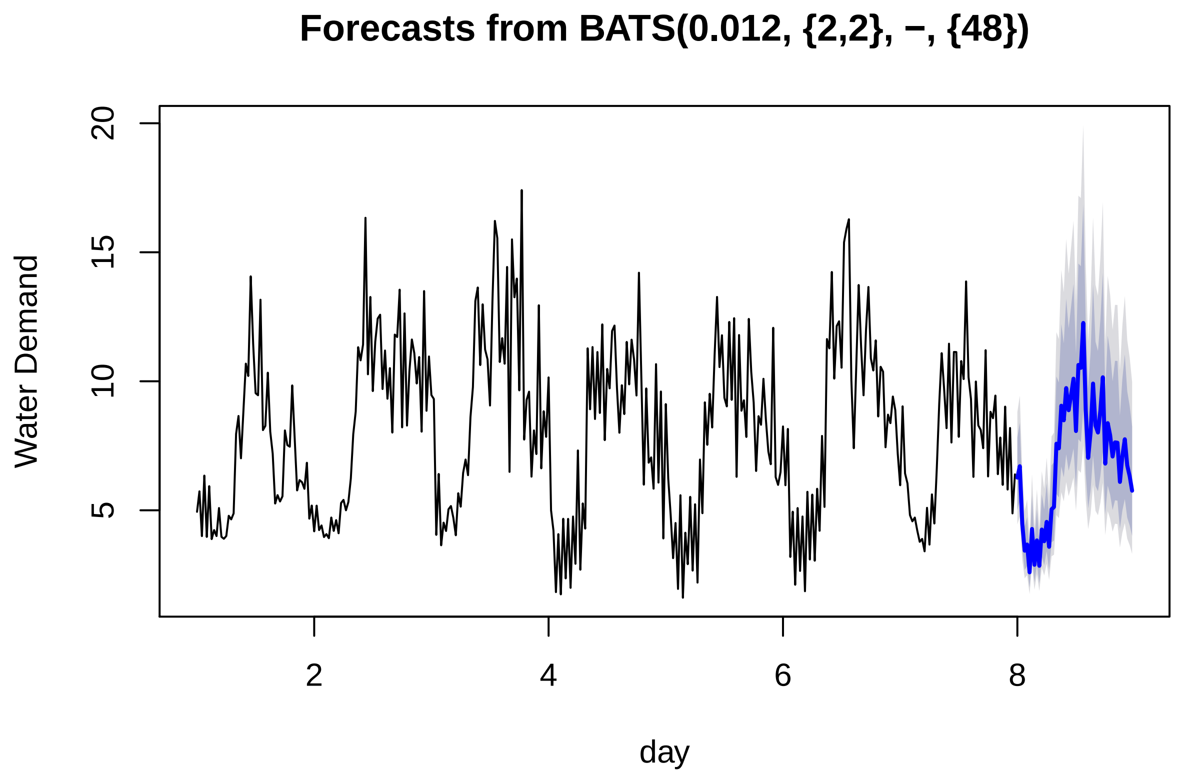

Figure 1: Forecasted demands using BATS algorithm

```
  ENsaveinpfile(inp_file)
  ENclose()
}
```

Invoking the function with the forecast generated above updates the network file.

```
update_pattern( "richmond-modified.inp",
                "forecast", as.numeric(day_ahead$mean))
```

6 Minimizing Energy Cost

To minimize energy cost considering the forecasted demands we search for a low-cost pumping schedule using a genetic algorithm. The GA package by Scrucca (2013) accepts a fitness function of that takes string input and returns a number for output. Here, an objective function is implemented to compute the pumping energy cost for a given schedule.

The function implemented below as objFun is specific to the modified Richmond network. It is assumed that the argument x contains a binary string corresponding to the on-off status of each pump at each 30-minute interval of a 24-hour simulation period. The function divides the input vector into seven vectors, one for each pump, and pushes this vector as the pattern for the respective pump. Hydraulic simulation of the full 24-hour period is invoked with ENsolveH. The solve function is called within a tryCatch function to allow the calling code to handle exceptional conditions. In this case, exceptional conditions of errors or warnings returned by the simulator are used to penalize the objective function. The penalty steers the optimizer away from hydraulically infeasible schedules. Calling ENsolveQ makes any water quality simulations. ENreport writes the results to a file. Finally, the energy cost for each pump is retrieved from the result file, and summed to get the objective value. Any penalty value arising from a hydraulic error or warning is added to the energy cost.

```r
updatePumpSchedule <- function(x){

  pump_patterns <- c( "pmp1Apat", "pmp2Apat", "pmp3Apat", "pmp4Bpat",
                      "pmp5Cpat", "pmp6Dpat", "pmp7Fpat")

  labeled_inputs <- split( x, rep(pump_patterns, each = 48 ) )

  # Update the pattern/schedule for each pump
  for( i in 1:length(pump_patterns)){
    pattern_index <- ENgetpatternindex( pump_patterns[i] )
    new_schedule <- labeled_inputs[[i]]
    ENsetpattern(pattern_index, new_schedule)
  }
}
```

```r
objFun <- function(x){

  ENopen("richmond-modified.inp","temp.rpt","temp.bin")

  updatePumpSchedule(x)

  # penalty for a hydraulic warning or error
  penalty <- tryCatch({
                        ENsolveH()
                        0 # no penalty if all is well
                      }, warning = function(w){
                        1000 # penalty for warning
                      }, error   = function(e){
                        5000 # penalty for error
                      })
  # solve & save & close
  ENsolveQ(); ENreport(); ENclose()

  # read the result out of the rpt file
  energyUsage <-suppressWarnings( read.rpt("temp.rpt")$energyUsage )
  cost <- sum(energyUsage$dailyCost) + penalty
}
```

Using the objective function, a genetic algorithm can be used to find a pumping schedule with lower energy cost. By default the ga function maximizes the fitness function. To minimize the fitness instead we apply a sign change. The number of bits must be supplied for binary encoded optimizations. The network used here includes seven pumps, each operating for one day on a pattern with a 30-minute time step. Thus 7*48 bits are required to define the pumping schedule.

```r
# use existing schedule as starting point
rmod <- read.inp("richmond-modified.inp")
sugg <-  c(rmod$Patterns$pmp1Apat, rmod$Patterns$pmp2Apat, rmod$Patterns$pmp3Apat,
           rmod$Patterns$pmp4Bpat, rmod$Patterns$pmp5Cpat,  rmod$Patterns$pmp6Dpat,
           rmod$Patterns$pmp7Fpat)
```

```r
# apply genetic algorithm
myga <- ga( type = "binary", fitness = function(x) -objFun(x),
            nBits = 7*48, maxiter=10, monitor = FALSE, suggestions = sugg)
summary(myga)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  50
## Number of generations =  10
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
## Suggestions =
##   x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x335 x336
## 1  1  1  1  1  1  1  1  1  1   1          0    0
##
## GA results:
## Iterations             = 10
## Fitness function value = -138.35
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x335 x336
## [1,]  1  1  1  1  1  1  1  1  1   1          0    0
```

```r
# save optimal schedule in new file
ENopen("richmond-modified.inp","temp.rpt","temp.bin")
updatePumpSchedule( myga@solution[1,] )
ENsaveinpfile("richmond-modified-solved.inp")
```

```
## NULL
```

```r
ENclose()
```

7 MONTE CARLO SIMULATION

Monte Carlo simulation can be used to see how this schedule performs under demand uncertainty. Realizations of demand for the simulation are created by applying random deviates of the standard normal distribution to the prediction equation.

$$\hat{y}(h) = \bar{y}(h) \pm z_{\alpha/2}\sqrt{Var[error(h)]} \tag{1}$$

In 1, $\hat{y}$ is the prediction interval of simulated demand at time horizon h. The mean prediction at time h is $\bar{y}$. The standard normal quantile for level of significance $\alpha/2$ is z. The term $\sqrt{Var[error(h)]}$ is the standard error of the prediction. The prediction intervals shown in the forecast figure are for a known confidence level and so can be used to extract the standard error via Eq. 1.

```r
pred_mean <- day_ahead$mean
# standard normal quantile for upper 80% interval
Z_10 <- qnorm( 0.90, mean=0, sd=1)
# calc the standardized error of the prediction
pred_std_err <- (day_ahead$upper[,"80%"] - pred_mean ) / Z_10
# now we can use this to sample
sampled_demands <- function( pred, se ){
   N <- length(pred)
   Z <- rnorm( N, mean=0, sd = 1)
   x <- pred + Z * se
}
```

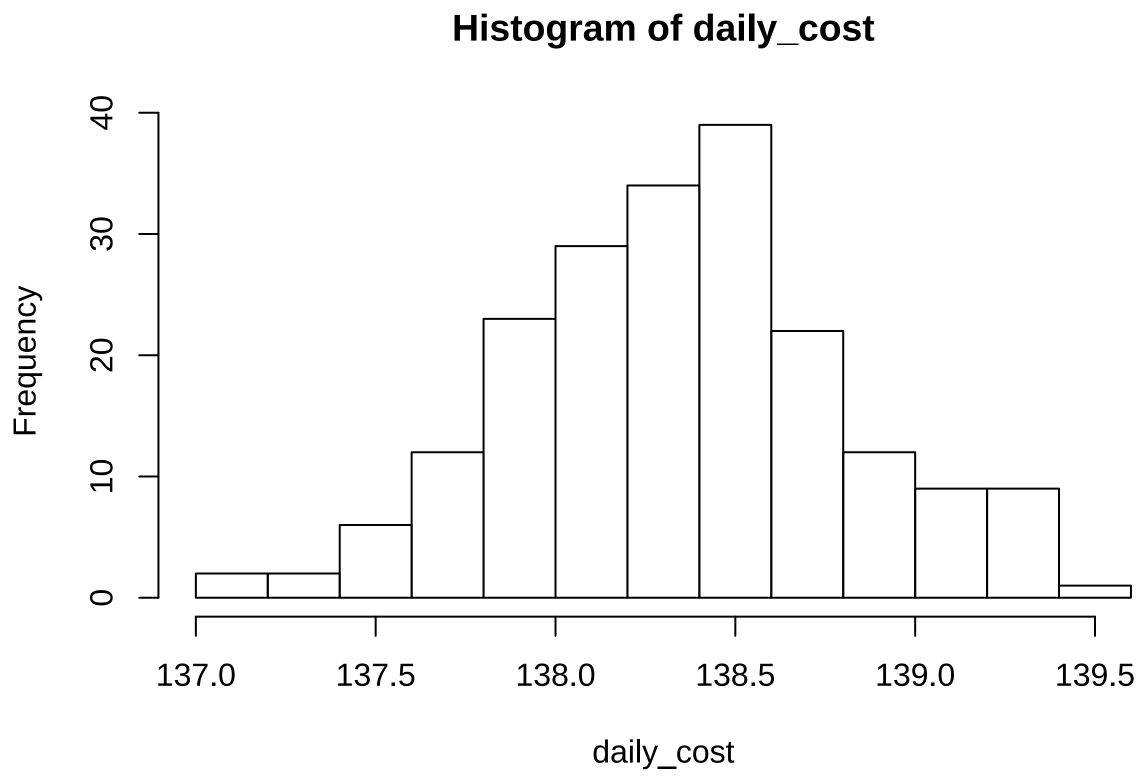

Figure 2: Histogram of daily pumping cost derived from Monte Carlo simulation.

Monte Carlo simulation can be carried out using a simple for loop. Each iteration samples a new demand pattern based on the forecast. This pattern is pushed to EPANET. A simulation under the sampled demand allows computation of energy cost. The cost distribution can be visualized using a histogram (Figure 2).

```
numRealizations <- 200
daily_cost <- rep(NA, numRealizations)

for(i in 1:numRealizations){
   dmd <- sampled_demands( pred_mean, pred_std_err)
   update_pattern("richmond-modified-solved.inp", "forecast", dmd)
   ENepanet("richmond-modified-solved.inp", "mc.rpt")
   rm_rpt <- read.rpt("mc.rpt")
   daily_cost[i] <- sum(rm_rpt$energyUsage$dailyCost)
}
```

```
hist(daily_cost)
```

8 Conclusions

This paper has provided several short examples related to water demand modeling and water network simulation using the R environment. One notable aspect is the opportunity to connect techniques developed by researchers around the globe. Indeed, the packages used here reflect years of work on the topics of water network analysis, time series forecasting, and stochastic optimization. The availability of these tools on a common platform reduces the effort required to implement existing analysis pipelines and creates opportunities to test and develop new methods.

REFERENCES

Arandia, E., Ba, A., Eck, B., and McKenna, S. (2015). “Tailoring Seasonal Time Series Models to Forecast Short-Term Water Demand.” J. Water Resour. Plann. Manage., 10.1061/(ASCE)WR.1943-5452.0000591, 04015067.

Arandia, E. and Eck, B.J. (2018) “An R package for EPANET simulations.” Environmental Modelling & Software, Volume 107, September 2018, Pages 59-63.

De Livera, A.M., Hyndman, R.J., & Snyder, R. D. (2011), Forecasting time series with complex seasonal patterns using exponential smoothing, Journal of the American Statistical Association, 106(496), 1513-1527.

Eck, B. (2016). An R package for reading EPANET files. Environmental Modelling & Software, Volume 84, October 2016, Pages 149-154.

Ghaddar, B, J. Naoum-Sawaya, A. Kishimoto, N. Taheri, B. Eck. (2015). “A Lagrangian decomposition approach for the pump scheduling problem in water networks.” European Journal of Operational Research 241 (2), 490-50.

Hyndman RJ and Khandakar Y (2008). “Automatic time series forecasting: the forecast package for R.” Journal of Statistical Software, 26(3), pp. 1-22.

Leutnant, D. (2017). swmmr: R Interface for US EPA’s SWMM. R package version 0.7.0.

McKenna, S.A., F Fusco, BJ Eck (2013) “Water demand pattern classification from smart meter data” Procedia Engineering 70, 1121-1130.

R Core Team, 2013. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

Scrucca, L. (2013) GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, 53(4), 1-37, http://www.jstatsoft.org/v53/i04/.

Van Zyl, J.E., Savic, D.A., Walters, G.A. (2004). Operational Optimization of Water Distribution Systems Using a Hybrid Genetic Algorithm, Journal of Water Resources Planning and Management, ASCE, 130 (3), 160-170.