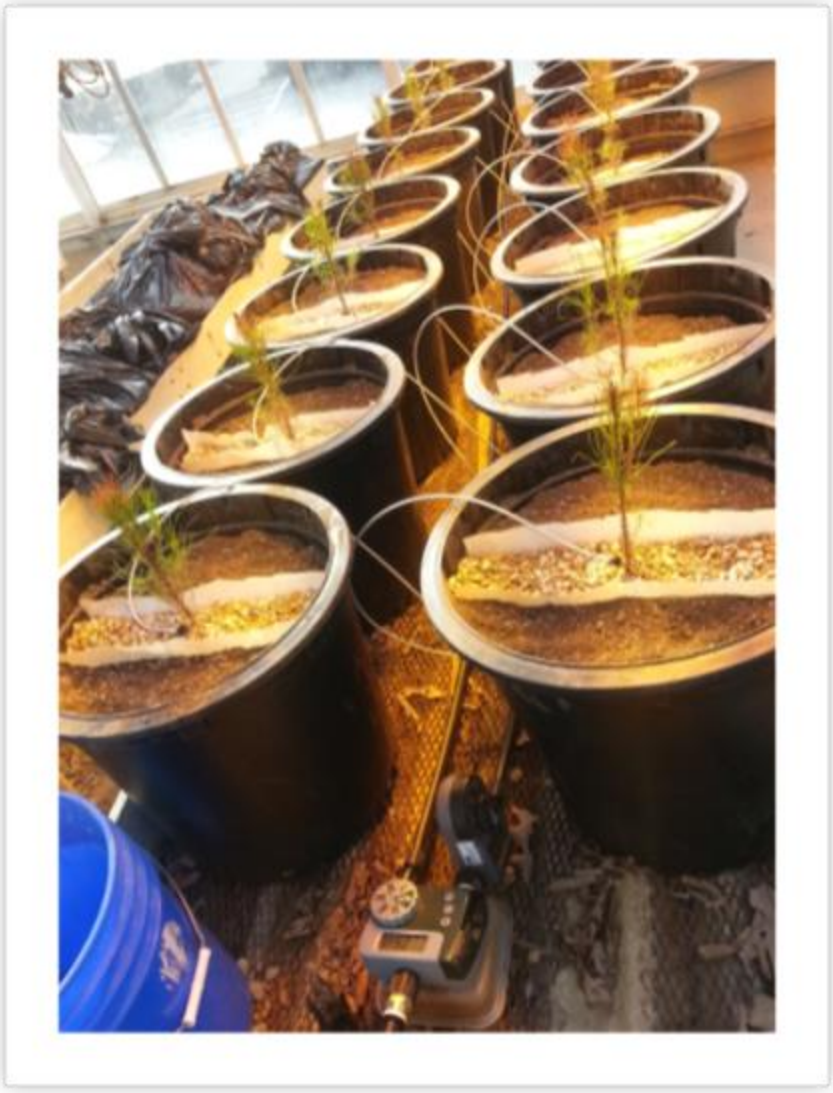


Sap Flux Measurement and Analysis

DATA IMPORT, MANIPULATION, GAP FILLING AND INTEGRATION

Ed Russell | Data Model Analytics | May 9, 2016

Data Import and Manipulation

First, we must get the data to be analyzed into R. The data represents sap flux probe measurements which are recorded every 15 minutes as microvolt readings which indirectly indicate the temperature difference between two thermocouples inserted radially into the trunk of a tree. One of the thermocouples is heated by applying a constant voltage, employing the Joule effect. When sap flows, the temperature difference decreases, as the heat is carried away from the upper (heated) thermocouple, bringing its temperature closer to that of the lower thermocouple.

The test data contains of 32 such time series, each representing readings taken from a single tree. I arbitrarily selected the last 16 time series. In this case, the data is formatted as a .dat file. This file type is importable by calling the *read.csv()* function, as it is a comma delimited file. The file has four lines of headers, which means that we must use the optional arguments, *skip=3,* and *header=TRUE* for the data to be imported correctly. Next, we remove unnecessary columns and then replace extreme values with NAs. Since the original data is in microvolts (uV), we need to convert it into Δtemperature in degrees C. This is accomplished by defining a parameter, the Seebeck coefficient, for type T thermocouples which gives the slope of the linear relationship between voltage and temperature. This step is accomplished by simply dividing the data frame by the Seebeck coefficient. Next, the data, which are now temperature differences, must be converted into sap flux density, a.k.a. one dimensional sap flux divergence. Then, the sap flux density values must be scaled by cross sectional area to arrive at a volumetric flow rate for each time point. Those steps were accomplished with the following R code:

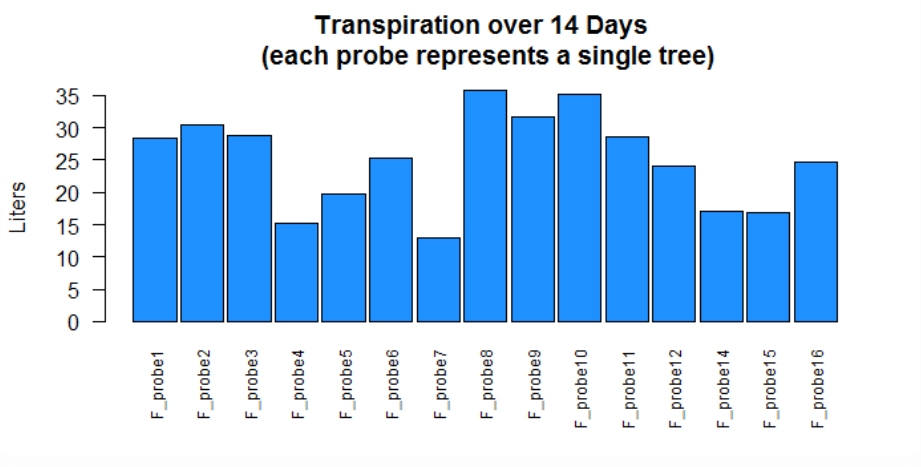
```
sap<-read.csv("VTSap1_110315SapFlow.dat",header=TRUE,skip=3)
sap<-sap[,-3] #remove the battery voltage column
#  head(sap) # there are some strange values..
#  tail(sap) # there are some strange values..
  sap.mat2<-sap[19:ncol(sap)] # select data for just the last 16 probes
    #replace strange values with NAs
    for(i in 1:nrow(sap.mat2)){
      for(j in 1:ncol(sap.mat2)){
        if(sap.mat2[i,j] > 10 | sap.mat2[i,j] < 1) {
          sap.mat2[i,j] <- NA
          }
      }
    }
SBCoef<- 40.5 # Parameter for type T thermocouples, in units of uV/deg C
SWa<-rep(1,times=16) # sapwood area of each tree

sap.mat2<-sap.mat2/SBCoef ##### this makes the dataset in deg C
# now we need to calculate the flux density using the temp diff values, then flow rates
# using the product of flux density and cross sectional area
## see 'Granier 1987 - Evaluation of transpiration in a Douglas-fir stand by means of
## sap flow measurements' for details
sap.mat3<-data.frame()
for(i in 1:nrow(sap.mat2)){
  for(j in 1:(ncol(sap.mat2))){
    sap.mat3[i,j]<-(max(sap.mat2[,j],na.rm=TRUE) - sap.mat2[i,j])/sap.mat2[i,j] # = K
    sap.mat3[i,j]<- (sap.mat3[i,j]^1.231)*.000119 # convert values to u, flux density
    sap.mat3[i,j]<- sap.mat3[i,j]*SWa[j] # u*sapwood area = Flow in m^3/s
  }
}

sap.mat2<-sap.mat3 # change name back to sap.mat2
rm(sap.mat3) # housekeeping
for(z in 1:ncol(sap.mat2)){names(sap.mat2)[z]<-paste0("F_probe",z)} # add names for the columns
sap.mat2<-cbind(sap.mat2,t=seq(1,nrow(sap.mat2),by=1)) # add a named time step column
```

Integration

Now that the data columns represent time series of flow rates in m^3*s^{-1}, the next step for complete data sets is to perform one dimensional numerical integration of the flow rates in order to produce a single value of volume transpired by each tree over the domain of the time variable. Of course, the data is not always complete, and I addressed that situation, but will detail that later in this project summary. In order to implement most methods of numerical integration it is necessary to have a function which gives a value of the dependent variable. There are a number of techniques which can be used to approximate a definite integral without a defining function. The most simple is the implementation of Reimann sums. Implementing the trapezoid rule is more accurate, as it represents the average of the left and right handed Reimann sums, and implementing Simpson's Quadrature, which allows for quadratic curvature is more accurate yet. I implemented both right-handed Reimann sums, and Simpson's Quadrature to accomplish my integration goal, and compared the results. I used a cumulative sum of values of the dependent variable weighted by the difference between time points to approximate the definite integral with a Reimann sums approach. I also wrote a function to perform Simpson's Quadrature. The Reimann sum consistently overestimated the value of the integrals by approximately 1-2‰. The code for these operations is below:

```
# This Portion of the script can be run without doing any of the modeling/forecasting if
# the datasets are all complete, or incomplete datasets are to be discarded

hf= 900 # is number of seconds per time step (seconds because the rates are m^3/s)
testCS<-cumsum(sap.mat2[-nrow(sap.mat2),13]*(hf*diff(sap.mat2[,ncol(sap.mat2)])))
  testCS[length(testCS)] # check one instance (one probe)

csTRANS<-list() # create list and then fill it using cumsum()
  csTRANS<-lapply(1:(ncol(sap.mat2)-1),
                  function (z) cumsum(sap.mat2[-nrow(sap.mat2),z]*
                                        (hf*diff(sap.mat2[,ncol(sap.mat2)]))))
  csTotTRANS<-vector() # create a vector of total transpiration and then name entries
    csTotTRANS<-sapply(1:(length(csTRANS)), function(z) csTRANS[[z]][length(csTRANS[[z]])])
      names(csTotTRANS)<-colnames(sap.mat2)[-(ncol(sap.mat2))]
print(csTotTRANS) # view the total transpiration values

simpson <- function(y, a=0, b=1350, n=1350, f=900) {
  # numerical integral using Simpson's rule
  # assume a < b and n is an even positive integer
  h <- (b-a)*f/n
  x <- seq(a, b, by=h)
  if (n == 2) {
    s <- y[1] + 4*y[2] +y[3]
  } else {
    s <- y[1] + y[n+1] + 2*sum(y[seq(2,n,by=2)]) + 4 *sum(y[seq(3,n-1, by=2)])
  }
  s <- s*h/3
  return(s)
}

SimpTRANS<-vector()
  SimpTRANS<-sapply(1:(ncol(sap.mat2)-1), function(z) simpson(y=sap.mat2[,z],
                                                              a=0,b=1350,n=1350,f=900))
    names(SimpTRANS)<-colnames(sap.mat2)[-(ncol(sap.mat2))]

# create a simple bar plot to visualize the relative transpiration levels
barplot(SimpTRANS[-13], space=.1,cex.names = .75,horiz = FALSE, col="dodgerblue",
        las=2,main="Transpiration over 14 Days \n (each probe represents a single tree)",
        ylab="Liters")

csTotTRANS<-as.data.frame(csTotTRANS)
SimpTRANS<-as.data.frame(SimpTRANS)
# here we directly observe differences between approximations
print(SimpTRANS)
print(csTotTRANS)
# explicitly, the differences as a proportion of the Simpson's Approximations
(SimpTRANS-csTotTRANS)/SimpTRANS  # we see the largest difference is ~ 0.23%, most are ~0.1%
# and that the cumsum() estimates are consistently larger than the Simpson's estimates
```

Here is the bar plot produced in the previous code snippet:



When comparing the two methods of approximating a definite one-dimensional integral, it becomes clear that for data with such high resolution, i.e. small time intervals between sampling points, there is a very small difference between methods. That being said, it's know that Simpson's approach is more accurate. I also compared the function I wrote in R to a Simpson's integrator I previously produced in MS Excel.

Prediction and Gap Filling

Perhaps one of the most daunting tasks associated with dealing with time series data is that of making predictions or gap filling missing values. These activities are challenging because time series data does not adhere to the usual assumptions necessary for implementing common approaches to interpolation such as linear regressions which assume independence of errors. This is true because the time points are almost always correlated with one another. This can be addressed using autocorrelation models where values depend on previous ones. There may also be 'random walks' in the mean value which are addressed with moving average models. Before those issues can be addressed, there are frequently linear or polynomial trends associated with the data which much be effectively modeled before the standard approaches to modeling time series data can be employed. Additionally, any seasonality which can be modeled with a harmonic regression or Fourier approach must also be accounted for. Specifically, employing auto regressive and moving average models require that the series be stationary. Stationarity is colloquially defined as a stochastic process whose statistical properties such as central tendency and variance, i.e. probability density, do not vary over time.

The first part of the larger process of prediction and gap filling starts with utilizing harmonic regression. This is a regression technique whereby periodic changes which can be visualized as sinusoid functions can be addressed by using a seasonal model. The mathematical background for implementing a regression model at the Fourier frequencies follows. Details can be found at: http://www-stat.wharton.upenn.edu/~stine/stat910/lectures/06_harmonic_regr.pdf , or here: http://stat.epfl.ch/files/content/sites/stat/files/shared/Applied%20Statistics/notes9.pdf or here: http://stat.epfl.ch/files/content/sites/stat/files/shared/Applied%20Statistics/notes10.pdf

A sine wave can be expressed as:

$$A\sin(2\pi \cdot f \cdot t + \phi) = \alpha_s \sin(2\pi \cdot f \cdot t) + \alpha_c \cos(2\pi \cdot f \cdot t)$$

with f, A, and ϕ being frequency (number of cycles per unit time), amplitude, and phase shift, respectively.

The harmonic seasonal model can be defined by:

$$x_t = m_t + \sum_{i=1}^{[s/2]} \{S_i \sin(2\pi \cdot f_i \cdot t) + C_i \cos(2\pi \cdot f_i \cdot t)\} + z_t$$

Where:

$$f_i = i \cdot f, \qquad i = 1, 2, \cdots, [s/2]$$

f is the basic frequency, s is the number of seasons

For the case that the seasons are the *12* months in a year:

- $f = 1$ if the time unit is year
- $f = 1/12$ if the time unit is one month

It follows that if there is 1 cycle per year, then 1/12 cycle per month

For the data I'm working with, I defined a season as the 15-minute period between measurements, making s/2 = 48, as there are 96, 15-minute periods in a day. The basic frequency is 1/96, which follows from the assumption that there is a periodic component with cycle length of one day. Now, to determine what frequencies, phases, and magnitudes contribute significantly to the individual time series we fit linear regression models to the data with regressors which are Fourier frequencies. These Fourier frequencies are multiples of the basic frequency up to ½. This is done using the following code:

```
cycl<-96 # this is the number of timepoints in a cycle, here 96 1/4hrs in a day
Time<-1:nrow(sap.mat2) #create vector of time points
# center and normalize time so O.rdinary L.east S.quares calcs are on smaller numbers
TIME<-scale(Time)
SIN<-COS<-matrix(nr=length(Time),nc=(cycl/2))# matrices for sin and cos, each of full potential
# number of harmonic terms, columns are for all different scaled frequencies

# populate the COS and SIN matrices with values with raw time as input
for (i in 1:(cycl/2)) {
 COS[,i]<-cos(2*pi*i*Time/cycl)
 SIN[,i]<-sin(2*pi*i*Time/cycl)
}
```

The following creates matrices of the regressors which are required for a downstream step where we fit ARIMA (auto regressive integrated moving average) models to the data.

```
#create a matrix of all of the possible regressors (W/O TIME) (for later w/ auto.arima)
xreg.a<-matrix(0,nr=nrow(sap.mat2),nc=cycl)
  for(i in 1:(cycl/2)) {
    xreg.a[,(2*i-1)]<-COS[,i]
    xreg.a[,(2*i)]<-SIN[,i]
  } # here, we left out the time and time^2 columns as predictors

### create a matrix of all of the possible regressors (WTH TIME) (for later w/ auto.arima)
xreg<-matrix(0,nr=nrow(sap.mat2),nc=cycl+2)
  xreg[,1:2]<-cbind(TIME=TIME,timeSQ=((TIME)^2))
    xregnames<-c("TIME","timeSQ")
      # create a vector of column names
      for( i in 1:(cycl/2)) {
        xregnames[(2*i)+1]<- paste0("COS",i)
        xregnames[(2*i)+2]<-paste0("SIN",i)
      }
        # fill the columns of xreg from SIN and COS matrices
        for(i in 1:(cycl/2)) {
          xreg[,(2*i+1)]<-COS[,i]
          xreg[,(2*i+2)]<-SIN[,i]
        }
    colnames(xreg)<-xregnames
    rm(xregnames) # housekeeping
```

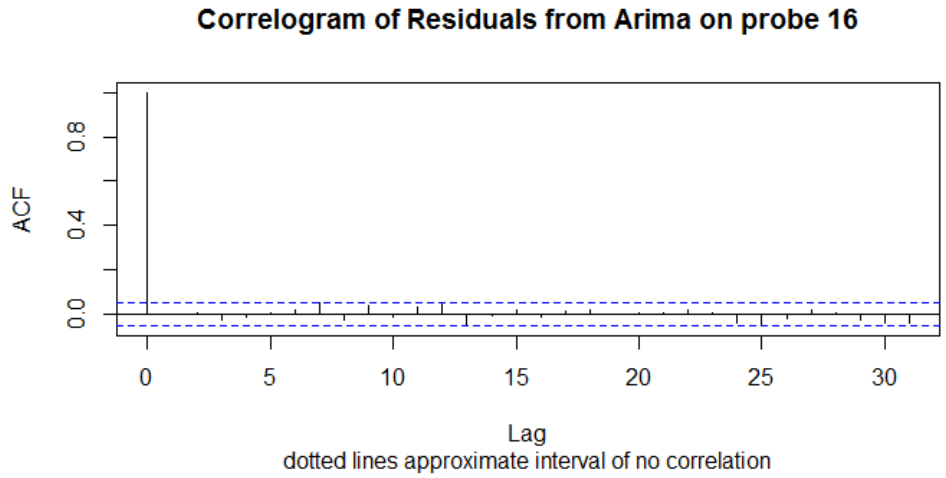
Next, we must fit linear regression models to the individual time series using the SIN and COS matrices created above. Once we have fitted the models, we must select the most parsimonious models. I used AIC (Akaike Information Criteria) for this purpose by implementing a stepwise model selection. Then I print the selected models by probe.

```
LMs<-list()
LMs<-lapply(1:length(outlist),
       function(x) lm(outlist[[x]][,1]~TIME + I(TIME^2) + COS[,1] + SIN[,1] + COS[,2] + SIN[,2] +
                                COS[,3] + SIN[,3] + COS[,4] + SIN[,4] + COS[,5] + SIN[,5] +
                                COS[,6] + SIN[,6] + COS[,7] + SIN[,7] + COS[,8] + SIN[,8] +
                                COS[,9] + SIN[,9] + COS[,10] + SIN[,10] + COS[,11] + SIN[,11] +
                                COS[,12] + SIN[,12] + COS[,13] + SIN[,13] + COS[,14] + SIN[,14] +
                                COS[,15] + SIN[,15] + COS[,16] + SIN[,16] + COS[,17] + SIN[,17] +
                                COS[,18] + SIN[,18] + COS[,19] + SIN[,19] + COS[,20] + SIN[,20] +
                                COS[,21] + SIN[,21] + COS[,22] + SIN[,22] + COS[,23] + SIN[,23] +
                                COS[,24] + SIN[,24] + COS[,25] + SIN[,25] + COS[,26] + SIN[,26] +
                                COS[,27] + SIN[,27] + COS[,28] + SIN[,28] + COS[,29] + SIN[,29] +
                                COS[,30] + SIN[,30] + COS[,31] + SIN[,31] + COS[,32] + SIN[,32] +
                                COS[,33] + SIN[,33] + COS[,34] + SIN[,34] + COS[,35] + SIN[,35] +
                                COS[,36] + SIN[,36] + COS[,37] + SIN[,37] + COS[,38] + SIN[,38] +
                                COS[,39] + SIN[,39] + COS[,40] + SIN[,40] + COS[,41] + SIN[,41] +
                                COS[,42] + SIN[,42] + COS[,43] + SIN[,43] + COS[,44] + SIN[,44] +
                                COS[,45] + SIN[,45] + COS[,46] + SIN[,46] + COS[,47] + SIN[,47] +
                                COS[,48] + SIN[,48]))


STEPPED<-list() # create blank list
  system.time(STEPPED<- lapply(1:length(LMs), function(x) step(LMs[[x]]) )) # fill list
    lapply(1:length(STEPPED), function(z) STEPPED[[z]]$call) # print selected harmonic models
```

The selected linear models fit the data well, but we're still working on the false assumption that the residuals are independent and identically distributed, which we know to be false. Our next step is to fit ARIMA models which address that problem and account for the linear trends fit by the models we just generated. To do this we need to load the *forecast* package into R. Then we need to run an automated, stepwise selection function to determine the best ARIMA model for each time series given our selection of regressors. That is accomplished using the code on the following page.

```
library(forecast) # load 'forecast' package to use the auto.arima() and forecast() functions
ARIMAs<-list()
  ARIMAs<-lapply(1:length(STEPPED),
                 function(z) auto.arima(x=STEPPED[[z]]$model[1], xreg=STEPPED[[z]]$model[-1],
                                        max.p = 5, max.q = 5, stationary = TRUE))
    attributes(ARIMAs[[1]])#look at element names
      for(z in 1:length(ARIMAs)) {
        cat('\n','\n','ARIMA MODEL FOR PROBE #',z,'\n', sep = "");
          print(summary(ARIMAs[[z]]))
      }
 for(z in 1:length(ARIMAs)) {
   temp.acf<-acf(ARIMAs[[z]]$residuals[-(1:6)],plot = FALSE)
   plot(temp.acf, main=paste("Correlogram of Residuals from Arima on probe",z),
        sub= "dotted lines approximate interval of no correlation")
   tsdiag(ARIMAs[[z]]) # look at time series diagnostics
}
```

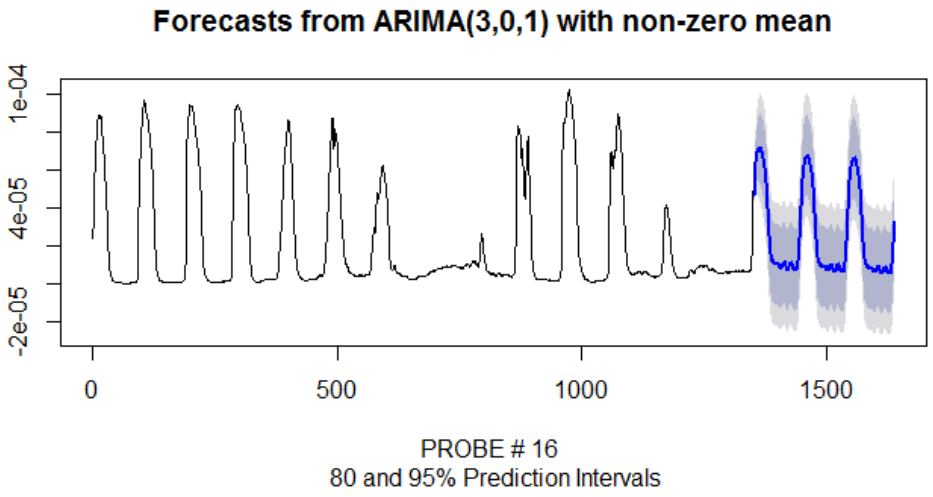
Below is an example of one of the diagnostic plots generated from the code block above. It is a correlogram of the residuals from the ARIMA on probe 16. We can see that the model has adequately removed autocorrelation from the residuals.



Now that we have ARIMA models that we find acceptable, we can use the *forecast* function from the *forecast* package to predict the value of the time series into the future, or over gaps in the data set, as the case may be. We pass the *forecast* function our ARIMA model, along with a matrix of future regressor values for the missing time period, or for the future. It is possible to generate these matrices since the frequency scaled SIN and COS regressor values are functions of time, so all we really need is future time values. We accomplish this with the following code:

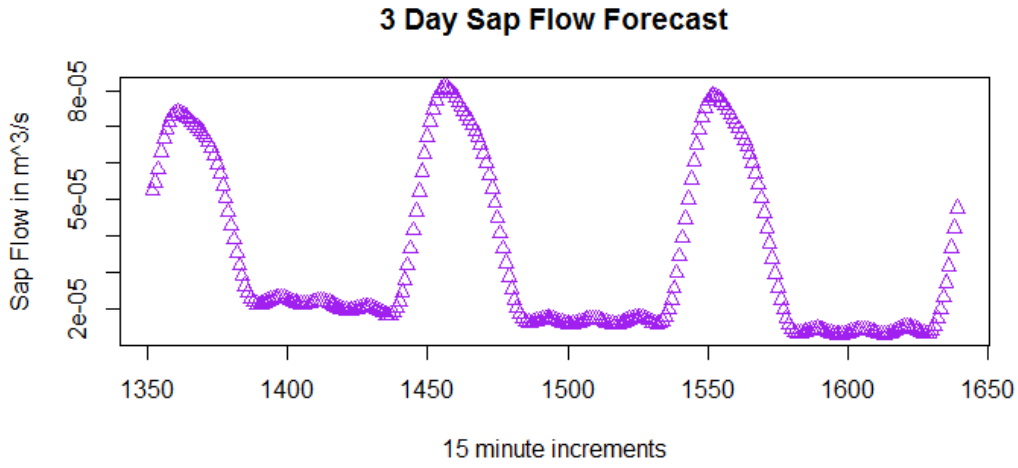
```
FORECASTs<-list()
  FORECASTs<-lapply(1:length(ARIMAs),
                    function(z) forecast(ARIMAs[[z]],xreg=STEPPED[[z]]$model[1:288,-1],
                                         level=c(80,95)))
    for(i in 1:length(FORECASTs)) {
      plot(FORECASTs[[i]],sub=paste("PROBE #",i,'\n',
                                    "80 and 95% Prediction Intervals",sep = " "))
      } # clearly the data from probe 13 is garbage
```

Plots of the predictions are produced for each time series. One of these plots is presented on the next page.



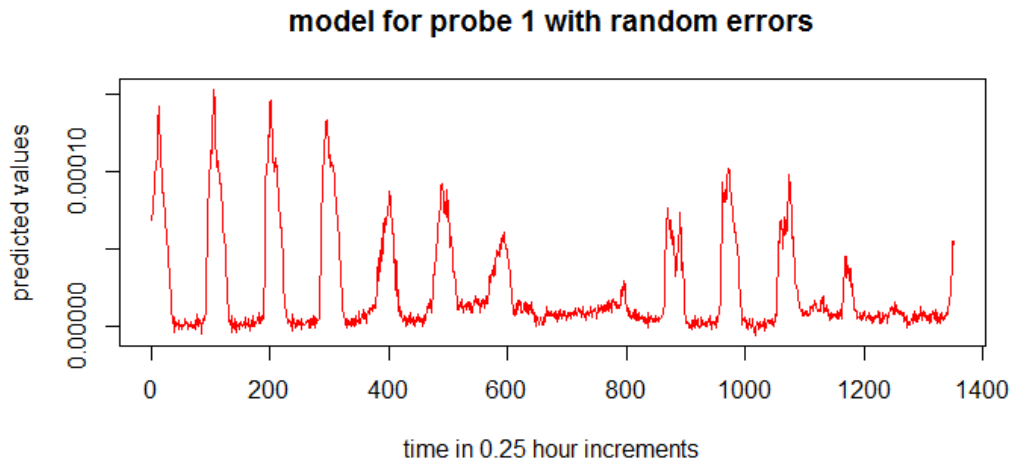
Here is example code that produces a plot of just the predictions alone for probe 1:

```
plot((length(Time)+1):(length(Time)+288),FORECASTs[[1]]$mean,
       ylab="Sap Flow in m^3/s", xlab="15 minute increments", col= "purple",
       main="3 Day Sap Flow Forecast", type="b",pch=2) #plot probe 1 forecast alone as one examp
        ##### if we were going to use these predicts we'd add a random error term to them
        ##### based on the evidence below that residuals are ~random
```



The residuals of the forecast models were examined to see if there was any remaining autocorrelation. There was virtually none. Predicted time series values are stored in the forecast objects, and can easily be specified, as in the above call to *plot()* for gap filling purposes. I also included code to generate simulations of the different time series based on the fitted values from the ARIMA models. The residuals were roughly normally distributed, so I simply added a normal variate to each model where the standard deviation specified in the call to *rnorm()* was the corresponding standard deviation of the residuals from the respective ARIMA model. That code is shown on the next page, along with a plot of one of the simulated data sets (probe 1).

```
ResidSD<-list()
  ResidSD<-lapply(1:length(FORECASTs), function(z) sd(FORECASTs[[z]]$residuals[-(1:6)]))

  for(z in 1:length(FORECASTs)) {
    temp.acf<-acf(FORECASTs[[z]]$residuals[-(1:6)],plot = FALSE)
    plot(temp.acf, main=paste("Correlogram of Residuals from Forecast on probe",z),
         sub= "dotted lines approximate interval of no correlation")
  } # checked that there is virtually no autocorr in residuals, i.e. only white noise remains

# we add a white noise component to the fitted models (for simulating data sets):
Yhat<-list()
  Yhat<-lapply(1:length(FORECASTs),
               function(z) FORECASTs[[z]]$fitted + rnorm(length(FORECASTs[[z]]$fitted),
                                                         0,ResidSD[[z]]))

  for(i in 1:length(FORECASTs)) {
  plot((STEPPED[[i]]$model[,2]*sd(Time)+mean(Time)),Yhat[[i]],type="l",ylab="predicted values",
       xlab="time in 0.25 hour increments", main=paste("model for probe",i,
                                                       "with random errors"),col="red")
  }
```



So, in summary, I succeeded in accomplishing my project goals of being able to inport, manipulate, gap-fill/predict sap flux time series values, plot various models and predictions, and integrate those values using a numerical technique. I am satisfied with the outcome, and it has motivated me to continue to learn more about modeling time series data.