Surface Water

The Prompt

Background: The attached Excel file (Dataset_C) contains analytical chemistry data from a variety of surface water stations.

Given the attached dataset:

- Develop a tool for plotting time series by station and analyte.
- Symbolize data by season.
- Add a horizontal line for the mean concentration to each plot.
- Which analytes have the most seasonal variability? What tools/approach did you use?
- Produce a pdf of your plots.

The Answer/Analysis

There seem to be two stages of tasks set forth here: the first, to do some data visualization of a multi-dimensional data set, and the second, to determine which of the analytes show strong seasonal variability. Before diving into the seasonality analysis, I first tackled the data visualization of the observations by analyte and station.

NB: This document is as a .pdf rendering of an R Markdown, which allows for the integration of documentation, code, and outputs. What follows is all of the code required to process the data, and generate the outputs for this exercise.

First Forays and Data Explorations

Reading in the data, loading relevant libraries:

```
# Surface Water Data Viz and Seasonality; RStubbs 02/2018
# Generates plots and calculates sesaonally-adjusted data
# to compare to raw data to determine magnitude of seasonality
# Input: Surface water observations by station-analyte

rm(list=ls()) # Clear working environment

library("MapSuite") #Self-written library, has many common libs as dependencies
library("hexbin") # For the hex-bin plots
library(lme4) # Mixed-effects modeling package
library(htmlTable)

# Read in surface water observations as data.table sw
#setwd("/Users/stubbsrw/Documents/git_code/stubbs_repo/fe_problems/code/")
sw<-fread("Dataset_C.csv")
```

Next, I calculate various columns describing the date of observations that may be useful later on.

```
# Parse out date information from character date column
  sw[,index:=seq(1:nrow(sw))]
  sw[,Month:=as.numeric(strsplit(SampleDate,"/")[[1]][1]),by=index]
  sw[,Day:=as.numeric(strsplit(SampleDate,"/")[[1]][2]),by=index]
  sw[,Year:=as.numeric(strsplit(SampleDate,"/")[[1]][3]),by=index]
# Create formal column of integer-date
  sw[,Date:=as.IDate(paste0((2000+Year),"-",Month,"-",Day))]
# For each analyte, discover the minimum date; generate a yr/month index from that date:
  sw[,year_index:=Year-min(Year,na.rm=T),by=StandardAnalyte]
# Number of months from the start of the samples
  sw[,month_index:=12*(Year-min(Year,na.rm=T)) + Month]
```

In order to symbolize by season, I am lumping observations into seasons by month. It's certainly possible to do this a more sophisticated way, based on cut points of the equinoxes; for now, however, month-by-month approximations seem adequate.

```
sw[Month %in% c(12,1,2), Season:='Winter']
sw[Month %in% c(3,4,5), Season:='Spring']
sw[Month %in% c(6,7,8), Season:='Summer']
sw[Month %in% c(9,10,11), Season:='Fall']
# Defining Season as a factor variable
sw[,Season:=factor(Season, levels = c("Winter","Spring","Summer","Fall"))]
```

Also, I calculate the mean concentration of each analyte, as well as a log-transformed version of the observations, for convenience later on.

```
# Add variables on mean concentration for each
#analyte by site and globally/for all samples
sw[,mean_conc:=mean(StandardResult,na.rm=T),
     by=list(StationName,StandardAnalyte)]

# Just in case a log-transform would be more informative,
# although NaNs will exist where the observation is negative
sw[,log_obs:=log(StandardResult)]
```

Plotting Each Analyte by Station, Over Time

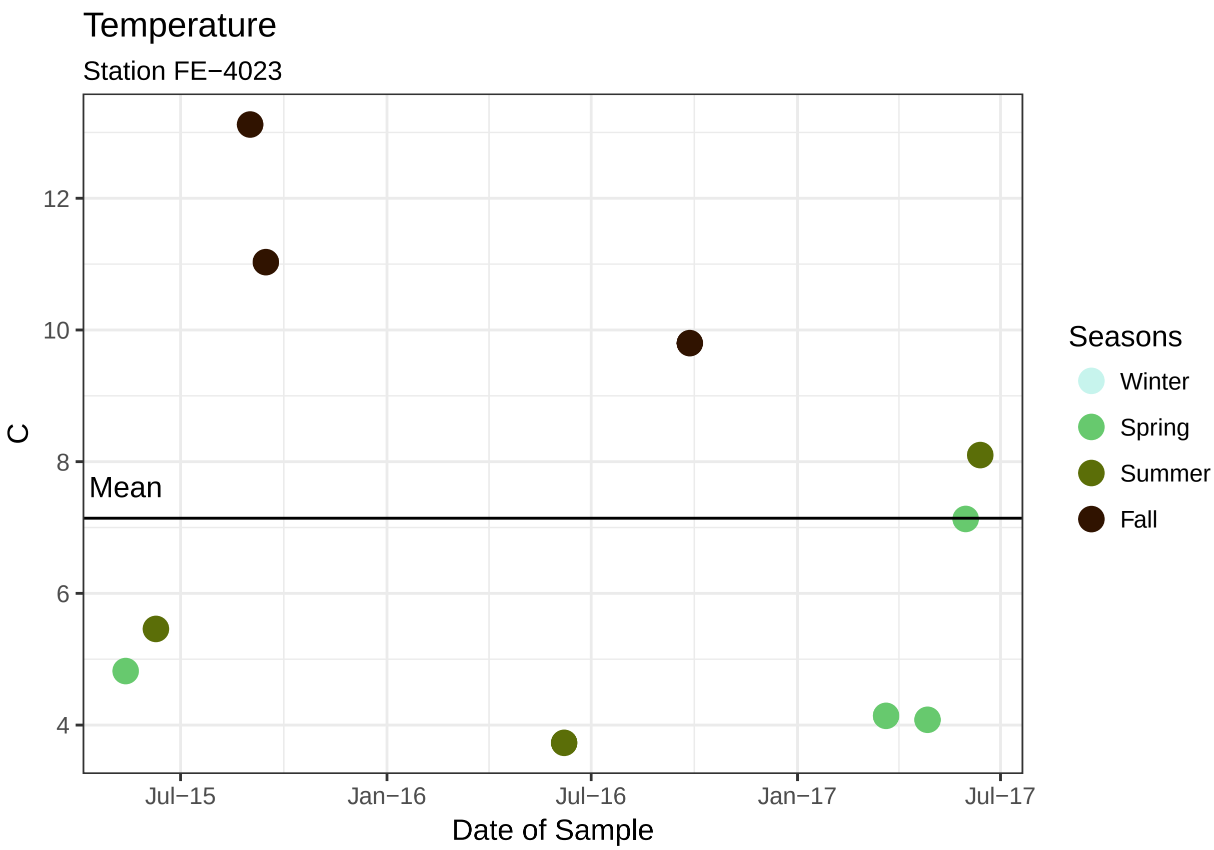
In order to make plotting by each station and analyte straightforward, I have written a function that takes a station name, and analyte name as input, and returns a plot of the points, color-coded by season, with a black line representing the mean of the analyte at that station across the full time period.

```
# Define a color pallette for the factor variable, Season
SeasonColors<-wpal("foliage",noblack=T,n=4) # grab colors from MapSuite's pallettes
names(SeasonColors) <- c("Winter","Spring","Summer","Fall")

# Define function to generate plot
  MakeAnalyteTSPlot<-function(a,s){

    p<-ggplot(sw[StandardAnalyte==a & StationName==s],
      aes(x= Date, y=StandardResult, color=Season)) + geom_point(size=4) +
      xlab("Date of Sample") +
      scale_x_date(labels = function(x) format(x, "%b-%y")) +
      ylab(sw[StandardAnalyte==a & StationName==s]$StandardUnit[1]) +
      ggtitle(paste0(a), subtitle=paste0("Station ",s)) + theme_bw()  +
```

```
    scale_colour_manual(name = "Seasons",values = SeasonColors, drop=F) +
    geom_hline(yintercept = sw[StandardAnalyte==a &
    StationName==s]$mean_conc[1]) +
    annotate("text", min(sw[StandardAnalyte==a & StationName==s]$Date),
             sw[StandardAnalyte==a & StationName==s]$mean_conc[1],
             vjust = -1, label = "Mean")

  return(p)
  }
```

Before iterating over all of the analytes and stations, and saving them to PDFs, I will test out the plotting function on one analyte, and one station:

```
# Make and print plot
  ts<-MakeAnalyteTSPlot(a="Temperature",s="FE-4023")
  print(ts)
```



Rather than simply having .PDFs, I sometimes find it useful to create a quick and dirty interactive visualization, with options to select and sub-set data. This won't work in a static file format like .PDF, but you can check out an interactive version of this plot, and some of the plots below,for each station and analyte, at this URL: **!!!!!!**

Quantifying Seasonality

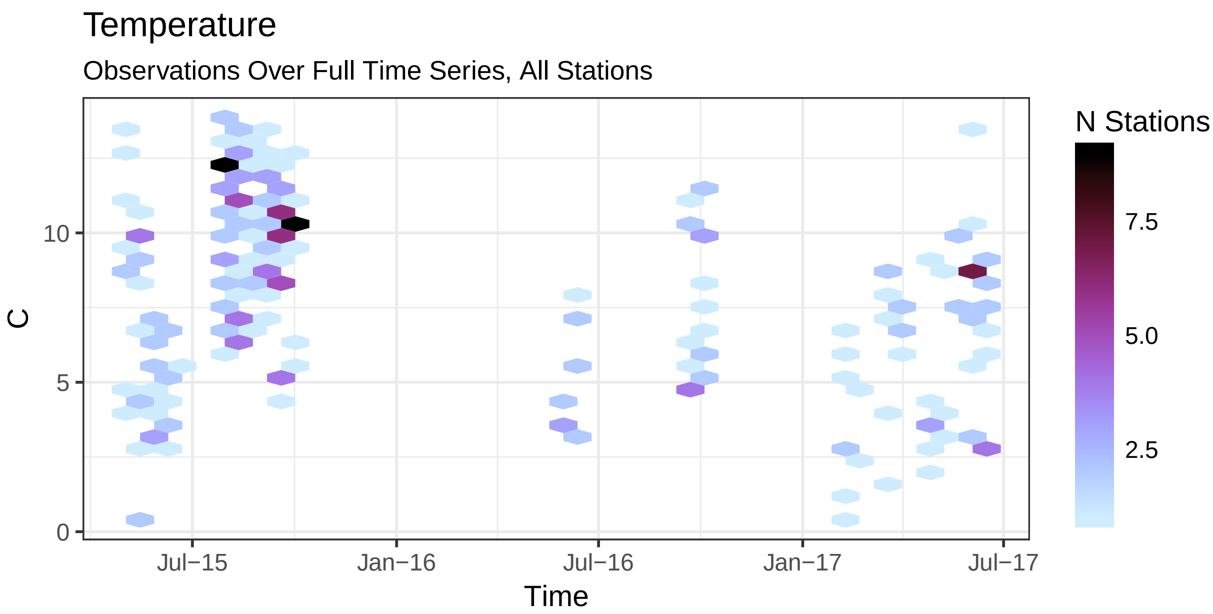
The fundamental question here seems to be, "to what extent is the variability in the data due to seasonal effects, rather than annual or other differences?" Unfortunately, there aren't very many data points per station within this data set, and the time period of observation only lasts a few years' time. This makes disentangling variation in observations for each site difficult–the change in observed data could be due to measurement error, a product of seasonal variation, inter-annual variation, random noise, or the effect of a recent event, or the influence of other unknown factors.

Without knowing the spatial location of any of the stations, I assumed that the stations would be nearby one another, and subject to (at least roughly) the same weather and insolation patterns— if these stations were far apart, it would be even more difficult to measure the "seasonal component" of each analyte, since the magnitude of seasonal changes could be subject to variables such as latitude and elevation.

Given the data constraints, a few different strategies came to mind. As a test case for each of them, I used temperature, since the right strategy would presumably show a seasonal effect for this analyte, and I had a sense of what probably "should" be happening (it presumably will get warmer in the summer months)!

As a first pass, I made a frequency plot that showed the observed values across the entire time period, for all stations.

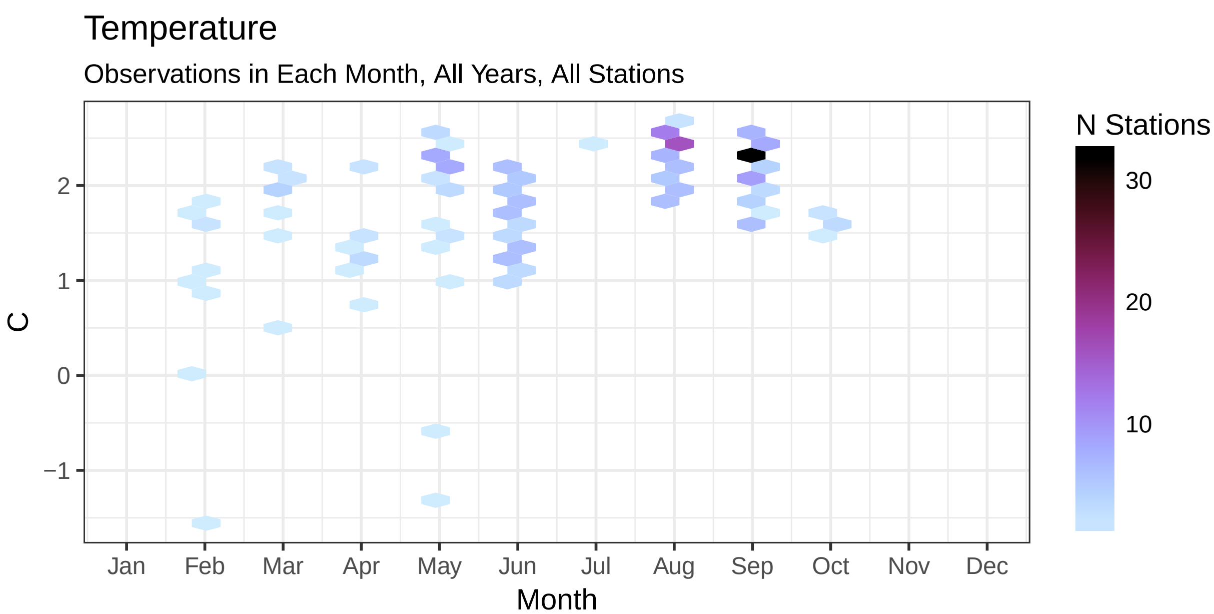
```
# Plotting using the HexBin Frequency graphics, where the number of
#stations with an observation in that category is essentially heat-mapped
plot_full_ts<-function(a){
p<-ggplot(sw[StandardAnalyte==a], aes(x=Date, y=StandardResult)) +
  geom_hex() + # honeycomb-plot geometry
  scale_x_date(labels = function(x) format(x, "%b-%y")) + xlab("Time") +
  ylab(sw[StandardAnalyte==a]$StandardUnit[1]) +
  ggtitle(paste0(a),
          (subtitle="Observations Over Full Time Series, All Stations")) +
  theme_bw() + scale_fill_gradientn(colors=wpal("berries")) +
  guides(fill=guide_colourbar(title="N Stations",
                              title.position="top", barheight=10, barwidth=1,
                              label=TRUE, ticks=FALSE, direction="vertical"))
return(p)
}
plot_full_ts("Temperature")
```



Temperature
Observations Over Full Time Series, All Stations
N Stations
7.5
5.0
2.5
C
10
5
0
Jul−15
Jan−16
Jul−16
Jan−17
Jul−17
Time

Let's try pooling observations across stations and also years, to get a rough sense of seasonality from a graphical persepective:

```
# Plotting by Month, for all years, all stations
plot_by_month<-function(a){
p<-ggplot(sw[StandardAnalyte==a], aes(x= Month, y=log_obs)) + geom_hex() +
  ggtitle(paste0(a),
          subtitle="Observations in Each Month, All Years, All Stations") +
  scale_x_continuous(limits=c(1,12),breaks=seq(1,12),
                   labels=c("Jan","Feb","Mar","Apr",
                   "May","Jun","Jul","Aug",
                   "Sep","Oct","Nov","Dec")) +
  ylab(sw[StandardAnalyte==a]$StandardUnit[1]) + theme_bw() +
    scale_fill_gradientn(colors=wpal("berries")) +
  guides(fill=guide_colourbar(title="N Stations", title.position="top",
                              barheight=10, barwidth=1, direction="vertical",
                              label=TRUE, ticks=FALSE))
return(p)
}
plot_by_month("Temperature")
```



It looks like there is a trend there to the casual eye, but it is interesting to note that there are very, very few observations during the winter months for this analyte. Furthemore, in the plot across all time, it appears that the summer months were colder in later years. Versions of these plots can also be found on the same interactive data visualization at the URL **!!!!!!URL!!!!!**

To determine the impact of seasonality on the measurements, it's necessary to disentangle how much of the apparent differences are caused by annual trends. To achieve this goal, I fit a model that included terms for both year, and season.

Using a mixed-effects* model to tease out seasonality from the data

I used a relatively bare-bones model, with parameters for an intercept with respect to all of the data, and deviations from that intercept (modeled as random effects) for each year and season. Including the season and year variables in the model this way, as sub-classifications of the data set, removes the idea of temporal "trajectory" or sequence, which may not exist, or which may reverse direction mid-way in the time period due to some sort of effect or intervention. Furthermore, from a theory standpoint, this model expresses that the deviations seen in the observations are the product of a stochastic process we have not observed, and that by default, these subgroups are likely to be deviations from an overall pattern (the 'global' intercept).

An initial stab at this model also included the sampling station such that the model would have a random intercept for each station as well– however, insufficient data for many of the observation sites (on the order of 2-3 observations for the full time period for Temperature, for instance) makes inference about the individual station's expected deviation from the mean dubious, and including them seemed like an excessive number of parameters to fit for this exercise. Knowing that certain sites group together might be one way to improve this– for instance, if all of the "FE-ET" stations could be considered together, it would likely improve the model's estimate, and would better disentangle variation due to site-specific conditions rather than seasonality or annual differences.

- Note that this modeling technique is called something different by almost every discipline… Wikipedia informs me that these models are also called multilevel models, hierarchical linear models, nested data models, random coefficient, random-effects models, random parameter models, or split-plot designs.

The model:

```
analyte<-sw[StandardAnalyte=="Temperature"]

# Testing out a linear model with temperature
mod <- lmer(StandardResult  ~ 1+ (1|Year)+(1|Season),data=analyte)
summary(mod)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: StandardResult ~ 1 + (1 | Year) + (1 | Season)
##    Data: analyte
##
## REML criterion at convergence: 1255.4
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.8791 -0.7695  0.1593  0.7642  2.5436
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Season   (Intercept) 2.605    1.614
##  Year     (Intercept) 1.849    1.360
##  Residual             7.616    2.760
## Number of obs: 255, groups:  Season, 4; Year, 3
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)    6.691      1.160   5.766
```

The model has fit intercept shifts for each season category based on the idea that each season's deviation from the global mean is pulled from a mean 0, normal distribution, with the standard deviation of this distribution as a modeled parameter– the fact that the distribution for 'year' has a lower standard deviation than the distribution for 'season' is interesting here, but doesn't directly speak to the impact of seasonality on the data. Taking a look at the coefficients, we can see the estimated values of the random intercepts based on each year and season category:

```
htmlTable(coef(mod)$Season) # See Season modeled random effects
```

(Intercept)

Winter

4.59581173002103

Spring

6.75683194126523

Summer

7.18713938186203

Fall

8.22351409544767

To determine how much seasons "matter" for each analyte, we can compare the observed data with and "without" seasonality. I will subtract the model estimated intercept shifts from the observed data points, based on which season the data were taken– this is functionally the same as having fed the model new data to 'predict' values, in which the new, season-free data belongs to none of the seasonal sub-groups that the model was fit on. Then, I calculate how different, on average, the seasonally adjusted data points are from the raw data points.

```
# Merge on information from model to analyte data.table
analyte<-merge(analyte,data.table(Season=levels(analyte$Season),ranef(mod)$Season),
               by="Season")
setnames(analyte,"(Intercept)","Seasonal_Adjustment")
# Create new varaible of seasonally-adjusted data
analyte[,seasonally_adjusted:=StandardResult-Seasonal_Adjustment]
# Calculate absolute % difference for each observation
# % diff equation--> 100 * |a - b| / ((a + b) * 2)
analyte[,pct_diff:=abs(100*(seasonally_adjusted-StandardResult)/((seasonally_adjusted+StandardResult)*2)
```

```
mean(analyte$pct_diff)
```

```
## [1] 3.139293
```

```
median(analyte$pct_diff)
```

```
## [1] 2.000676
```

Now, we take this process to scale, running the same analysis for each of the analytes. I am excluding the analytes from this analysis for which fewer than 4 seasons were observed.

```
seasonal_summary<-list() # Create empty list for summary results to go into
seasonally_adjusted_data<-list() # Create emtpy list for data, adjusted and non, to go to
mods<-list() # A list to explore the model attributes for each analyte

# Determine list of analytes
  analytes<-unique(sw$StandardAnalyte)
# Exclude dissolved mercury; that model is unidentifable apparently
  analytes<-analytes[!analytes %in% c("Mercury, dissolved")]

  skipped<-c()
for (a in analytes){ # For each analyte

  # Check to see if all seasons are observed, and proceed, else skip
  if(length(unique(sw[StandardAnalyte==a]$Season))==4){
    analyte<-sw[StandardAnalyte==a] # Subset data
    mod <- lmer(StandardResult  ~ 1+ (1|Year)+(1|Season),data=analyte) # Fit model
    mods[[a]]<-mod
```

```r
    analyte<-merge(analyte,data.table(Season=levels(analyte$Season),ranef(mod)$Season),
                 by="Season") # Merge intecept shifts onto raw data
    setnames(analyte,"(Intercept)","Seasonal_Adjustment") # Clarify name
    # Create new var seasonally-adj data
    analyte[,seasonally_adjusted:=StandardResult-Seasonal_Adjustment]
    # Create summary measures; pct_diff
    analyte[,pct_diff:=abs(100*(seasonally_adjusted-StandardResult)/
                             ((seasonally_adjusted+StandardResult)*2))]

    # Add results to lists for easy access later
    seasonal_summary[[a]]<-data.table(Ananlyte=a,
                                      median_diff=quantile(analyte$pct_diff,.5),
                                      mean_diff=mean(analyte$pct_diff),
                                      # Add columns on n observations
                                      N=nrow(analyte),
                                      Winter=nrow(analyte[Season=="Winter"]),
                                      Spring=nrow(analyte[Season=="Spring"]),
                                      Summer=nrow(analyte[Season=="Summer"]),
                                      Fall=nrow(analyte[Season=="Fall"]))
    seasonally_adjusted_data[[a]]<-analyte
  }else{
    skipped<-c(skipped,a)
  }
}
  print("Skipped analytes:")
```

```
## [1] "Skipped analytes:"
```

```r
  print(skipped)
```

```
##  [1] "Cation-Anion Balance" "Hardness as CaCO3, T" "Lithium, total"
##  [4] "TSS"                  "Silica, total"        "Strontium, total"
##  [7] "Sum of Anions"        "Sum of Cations"       "Alkalinity, Total"
## [10] "DOC"                  "Mercury, total"       "TDS, calculated"
## [13] "TDS, ratio"           "Hardness as CaCO3"    "TOC"
```

Now that we have the summary information from each of the analytes, we can see what analytes had the highest degrees of seasonality:

```r
seasonal_summary<-rbindlist(seasonal_summary)
seasonal_summary<-seasonal_summary[order(-median_diff)] # Rank-order
setnames(seasonal_summary,"mean_diff","% Diff (Mean)")
setnames(seasonal_summary,"median_diff","% Diff (Median)")
htmlTable(seasonal_summary, align="l")
```

Ananlyte

% Diff (Median)

% Diff (Mean)

N

Winter

Spring

Summer

Fall

1

Vanadium, total

45.1490649178612

39.0953059776149

265

4

100

101

60

2

Thallium, total

44.1369306609008

31.1012737391413

285

4

100

111

70

3

Selenium, total

40.1402886832743

32.1725936582006

285

4

100

111

70

4

Arsenic, total

28.9728294385027

50.4112808770284

305

4

120

111

70

5

Antimony, total

27.172567053508

22.6469301672633

268

4

83

111

70

6

Iron, dissolved

21.7698670463248

23.9359829103358

314

13

120

111

70

7

Vanadium, dissolved

21.5812735573917

109.4070874163

274

13

100

101

60

8

Silver, dissolved

20.9656274921314

454.728717763721

314

13

120

111

70

9

Silver, total

17.8660740998672

59.3509783288579

305

4

120

111

70

10

Nickel, total

17.3692519927757

46.9937051656558

305

4

120

111

70

11

Lead, total

15.1383886030274

386.087666001871

302

4

117

111

70

12

Thallium, dissolved

14.5799962258943

30.0527731097526

294

13

100

111

70

13

Arsenic, dissolved

12.7583877043768

20.9007058087891

314

13

120

111

70

14

Turbidity

12.4632343301045

23.6110193587465

172

9

52

52

59

15

Selenium, dissolved

12.2935371528071

35.4848242344336

294

13

100

111

70

16

Molybdenum, total

12.1484482055452

70.2325642857507

249

4

103

82

60

17

Lead, dissolved

11.4852255043853

19.6051267175783

430

104

145

111

70

18

Chromium, dissolved

9.95618167139467

26.1070475385914

430

104

145

111

70

19

Copper, dissolved

7.3148203725558

22.0681213952788

430

104

145

111

70

20

Copper, total

7.10528419827551

71.9429743170597

305

4

120

111

70

21

Antimony, dissolved

7.04040483598041

8.46917007410665

277

13

83

111

70

22

Beryllium, total

6.79912386405165

25.9158971796574

285

4

100

111

70

23

Strontium, dissolved

6.41714245214573

9.7552858297489

222

9

71

82

60

24

Beryllium, dissolved

5.40930592608698

18.8707550650934

294

13

100

111

70

25

Iron, total

5.33744436081287

17.53605223021

305

4

120

111

70

26

Barium, total

4.91357076214369

5.86871214156178

265

4

100

101

60

27

Cobalt, dissolved

4.48054982455248

5.99683289705831

274

13

100

101

60

28

TDS, measured

4.46906932647731

7.39860848810517

182

9

51

72

50

29

Total Alkalinity

4.16441174018284

5.20853060482079

209

9

48

82

70

30

Calcium, dissolved

3.90693008234381

5.47939722628943

449

104

145

130

70

31

Magnesium, total

3.88655832308238

3.93017910973376

305

4

120

111

70

32

Cobalt, total

3.7874135654136

4.55436205172346

265

4

100

101

60

33

Barium, dissolved

3.47717114679597

3.82663649407993

274

13

100

101

60

34

Molybdenum, dissolved

3.46042645528221

4.84655909116532

255

13

100

82

60

35

Potassium, dissolved

3.42482547805873

3.76509406230971

274

13

100

101

60

36

Calcium, total

3.31705284270725

6.54363696298637

305

4

120

111

70

37

Zinc, dissolved

3.05468700990113

4.02481697325262

430

104

145

111

70

38

Chromium, total

3.02830205254307

12.3990694883638

305

4

120

111

70

39

Specific Conductance

2.70704666338294

3.36110937118146

257

9

53

114

81

40

Zinc, total

2.34147780835187

2.68312870514147

305

4

120

111

70

41

Cadmium, dissolved

2.01581114922725

3.26778688212127

430

104

145

111

70

42

Temperature

2.00067600274688

3.13929339215375

255

9

53

113

80

43

Manganese, dissolved

1.94277069925394

3.44155489230038

430

104

145

111

70

44

Hardness as CaCO3, D

1.78364979492109

4.12015081723512

146

9

51

38

48

45

Potassium, total

1.59584450125479

1.58318866334795

265

4

100

101

60

46

ORP

1.42774944848922

1.76757575612124

201

9

53

74

65

47

Sodium, total

1.40716635538389

6.26358178653235

265

4

100

101

60

48

Sodium, dissolved

1.33404129892563

6.88907523156946

274

13

100

101

60

49

Silica, dissolved

0.901715164926332

3.69920197522055

222

9

71

82

60

50

Magnesium, dissolved

0.736498526149315

3.29856885208357

449

104

145

130

70

51

pH

0.495808324510282

0.509799200203557

244

9

53

112

70

52

Lithium, dissolved

0.308610925554087

0.225177154261881

182

9

51

72

50

53

Nickel, dissolved

7.21644966006341e-13

7.61773219341738e-13

314

13

120

111

70

54

Cadmium, total

1.90323947078598e-13

2.40163023655784e-13

305

4

120

111

70

55

Oxygen, Dissolved

1.62049862351247e-13

1.57688444503231e-13

254

9

53

111

81

56

Aluminum, dissolved

0

0

430

104

145

111

70

57

Aluminum, total

0

0

305

4

120

111

70

58

Manganese, total

0

0

305

4

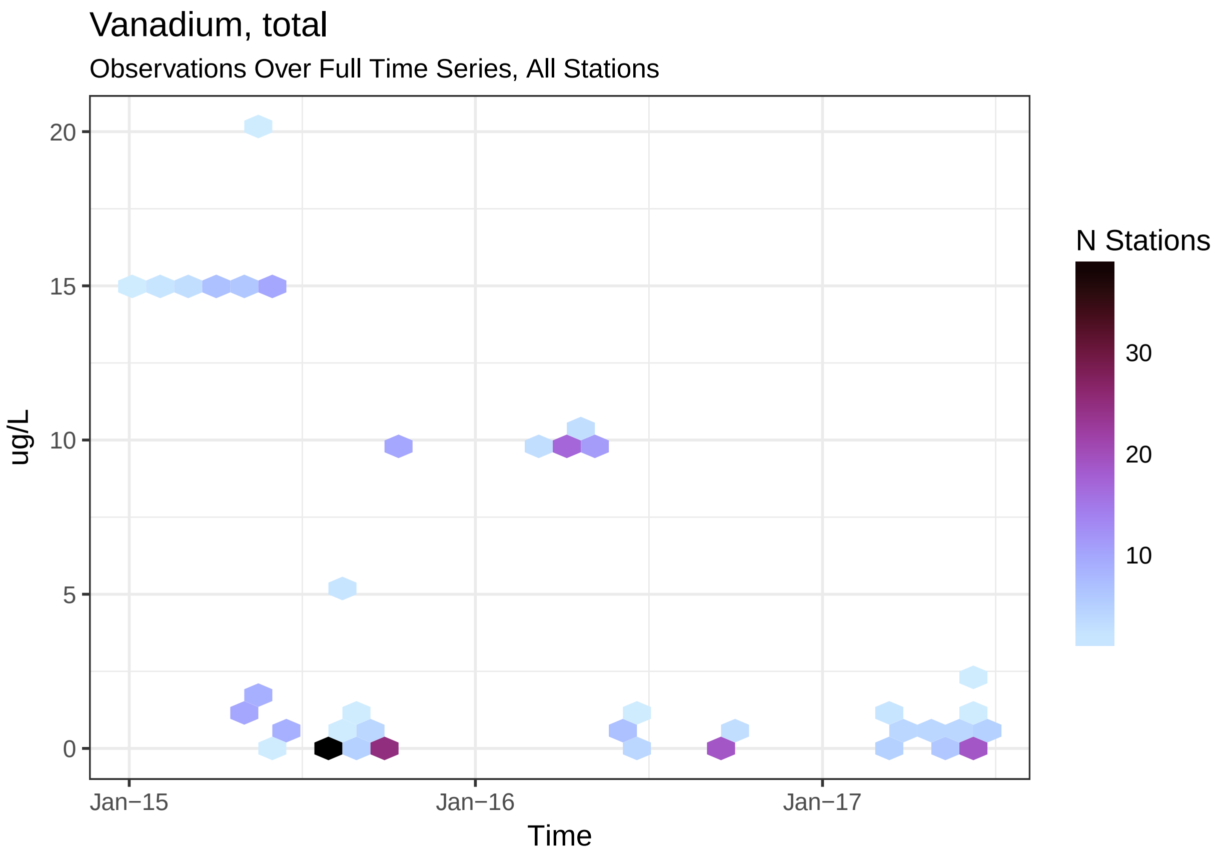
120

111

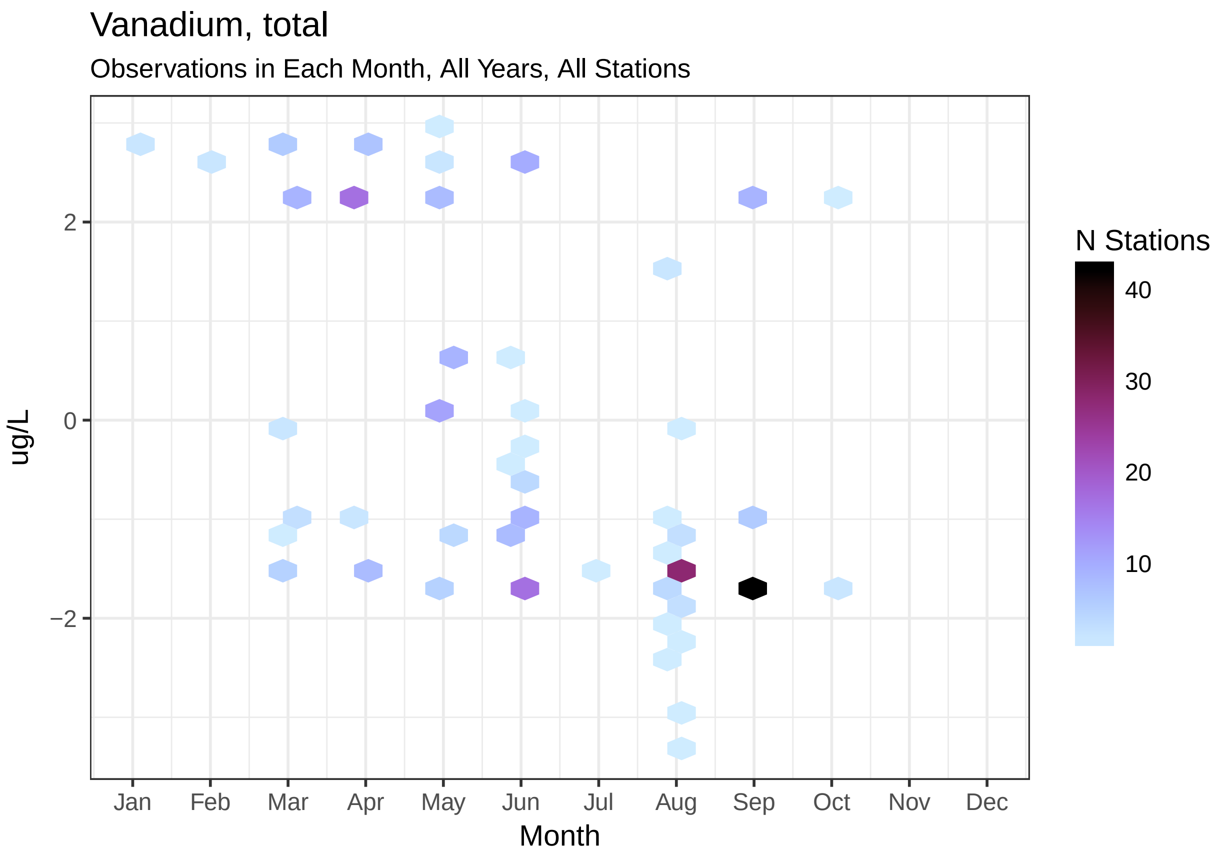
70

Some really interesting results here– it seems like Temperature, This stats.stacksexchange post goes into this in detail–

```
plot_full_ts("Vanadium, total")
```



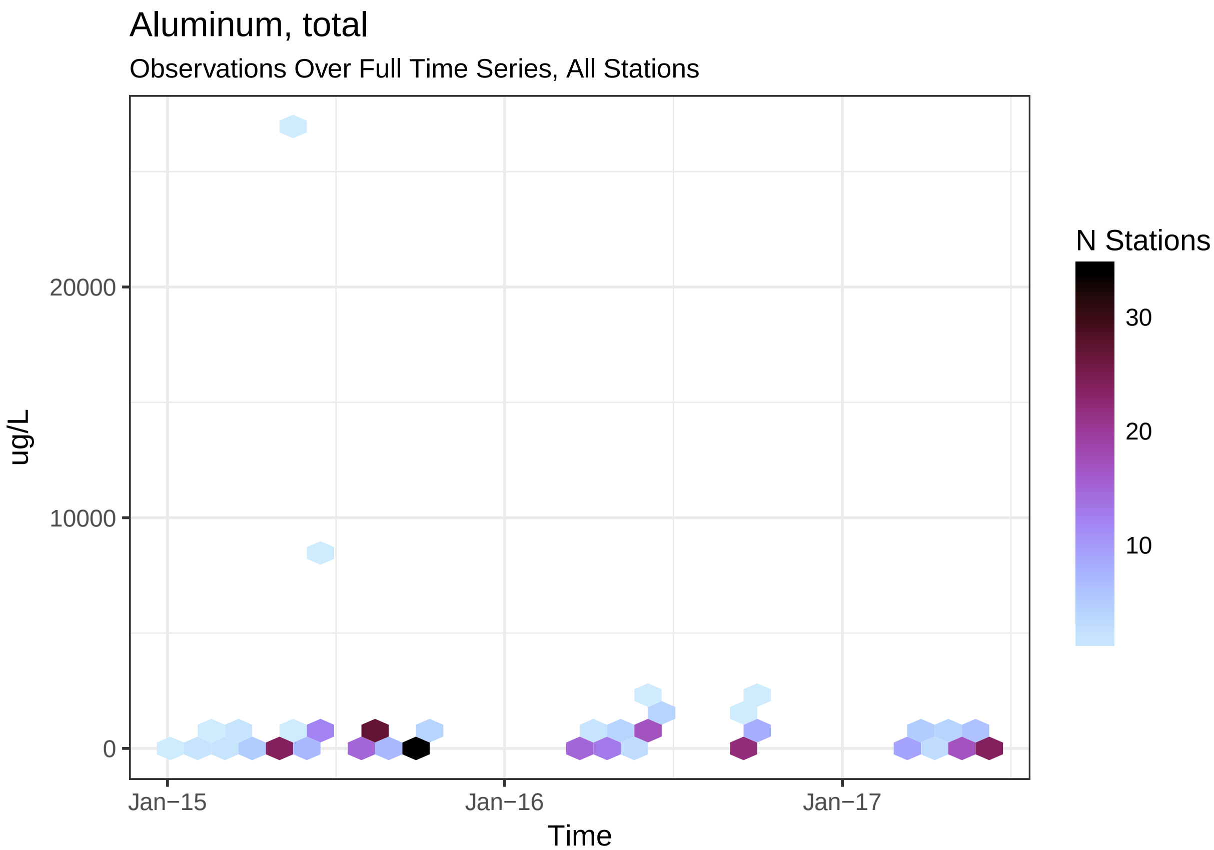

```r
plot_by_month("Vanadium, total")
```



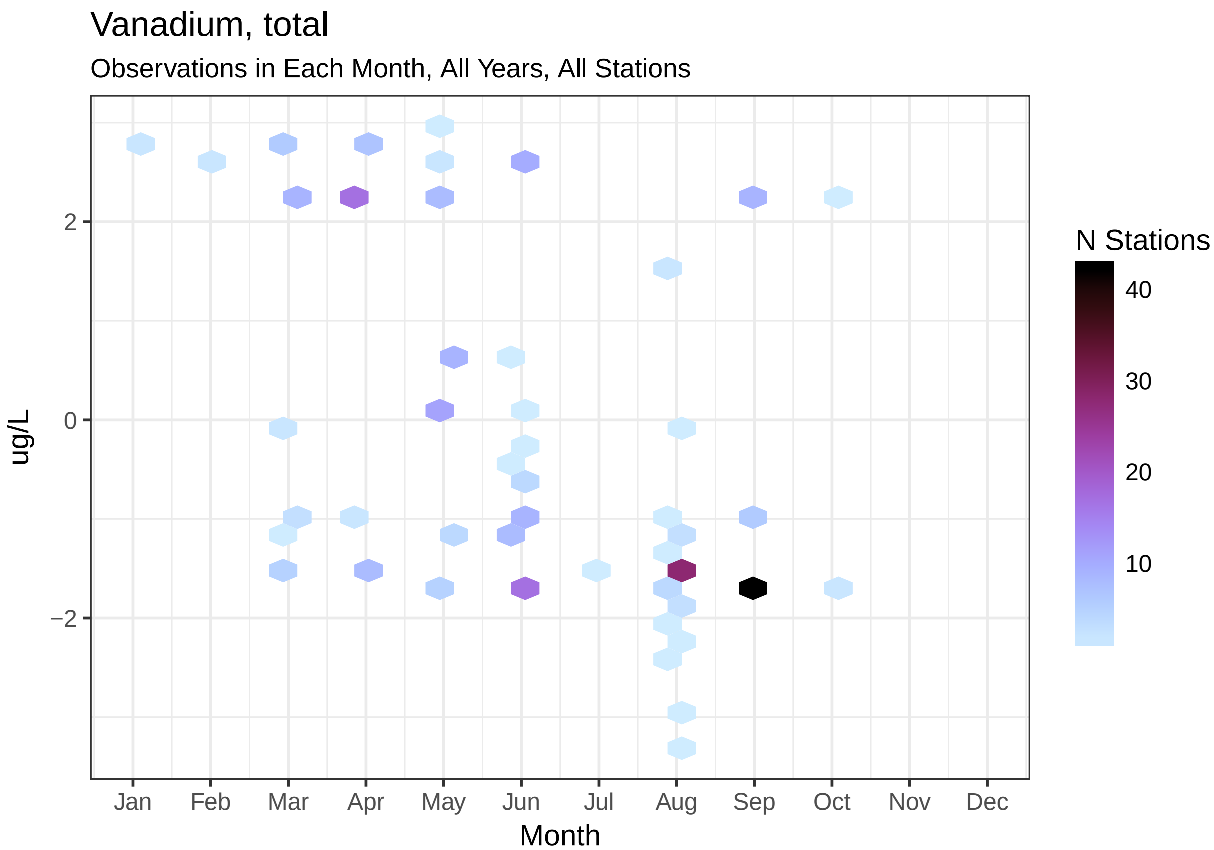

```r
alm<-mods[["Aluminum, total"]]
summary(alm)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: StandardResult ~ 1 + (1 | Year) + (1 | Season)
##    Data: analyte
##
## REML criterion at convergence: 5356.8
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -0.2591 -0.1963 -0.1285  0.0022 16.3414
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Season   (Intercept)       0     0
##  Year     (Intercept)       0     0
##  Residual             2583017  1607
## Number of obs: 305, groups:  Season, 4; Year, 3
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   436.45      92.03   4.743
```

```r
plot_full_ts("Aluminum, total")
```




```
plot_by_month("Vanadium, total")
```



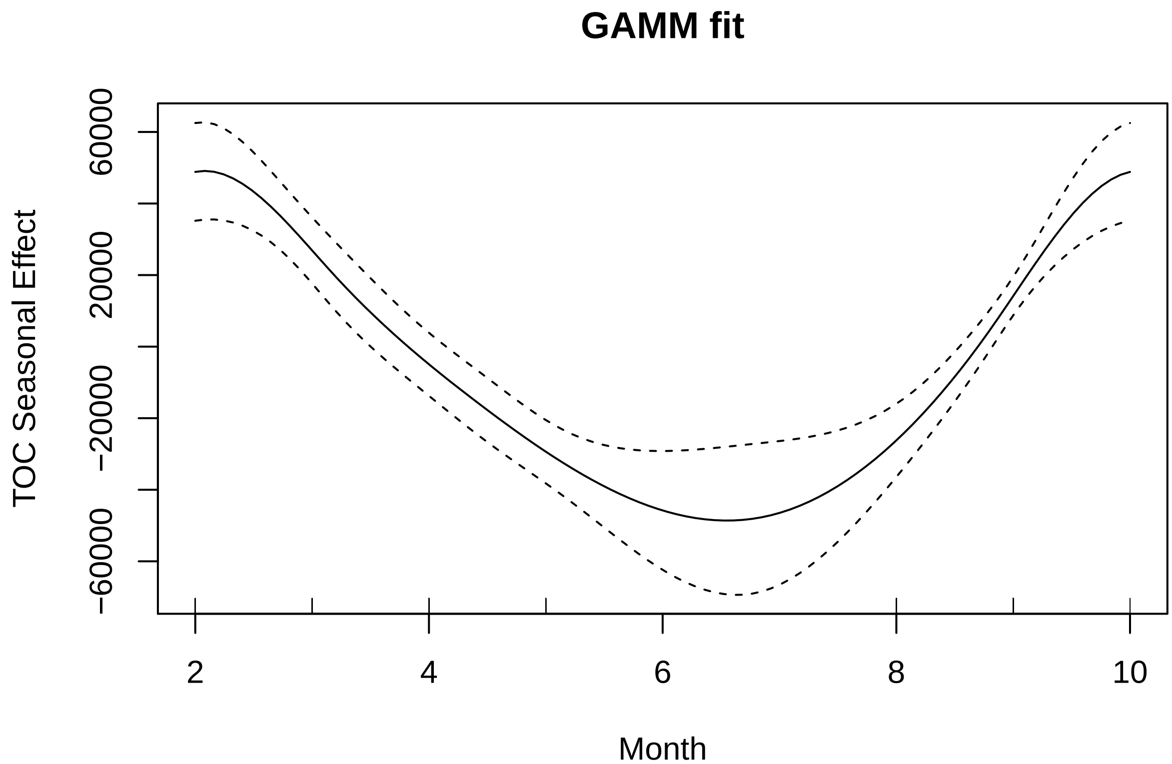
To get a sense of how substantial each season's effect is, we can compare the magnitude of the seasonal effect to the intercept (the otherwise expected value, once year is controlled for):

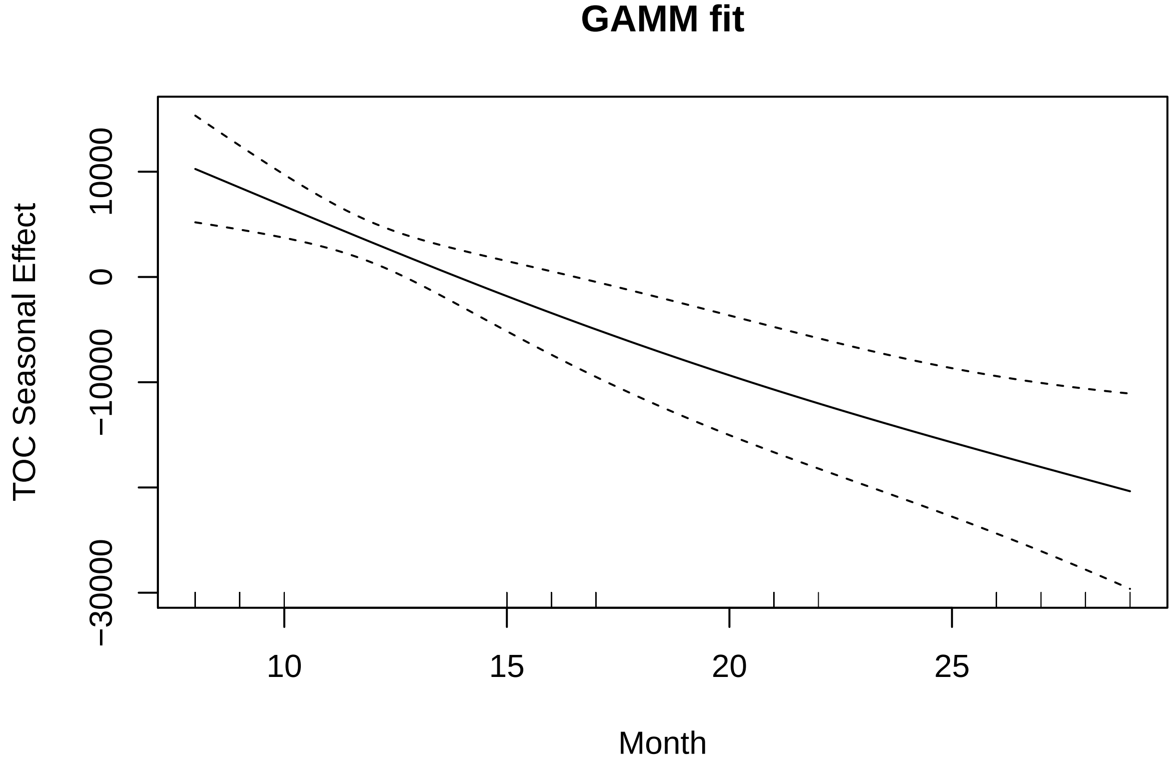
Fit GAMM with knots for each season and year

Another possible approach is to fit a general additive mixed model, where the long-term time trend and the short-term, periodic seasonal trends are conceptualized by cubic splines.

There is some discussion of the number of knots that are appropriate for seasonal land (not water) surface temperature changes in this paper here: http://www.mdpi.com/2072-4292/9/12/1254/pdf; however, it is discussed that the ideal number of knots for their model to represent seasonality (8) may not be appropriate for other applications.

```
# Pulling from https://www.fromthebottomoftheheap.net/2014/05/09/modelling-seasonal-data-with-gam/
# 1 knot every 6 months, 1 knot for each year of data?
m0 <- mgcv::gamm(StandardResult ~ s(Month, bs = "cc", k = 6) + s(month_index, k = 3) + as.factor(Statio

plot(m0$gam, scale = 0, main = "GAMM fit", xlab = "Month", ylab = paste0(a," Seasonal Effect"))
```





Let's try a linear model with a sin and cosine function

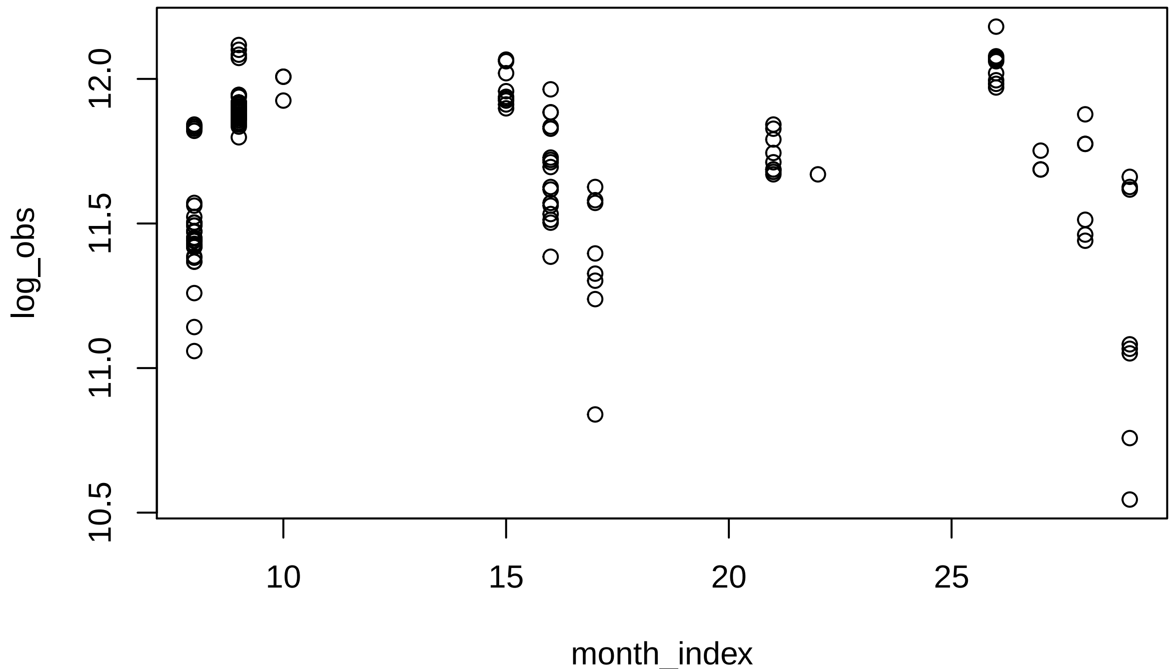
```
mod <- lm(log_obs  ~ sin(2*pi*(month_index/12))+cos(2*pi*(month_index/12))+StationName,data=analyte)
summary(mod)
```

```
##
## Call:
```

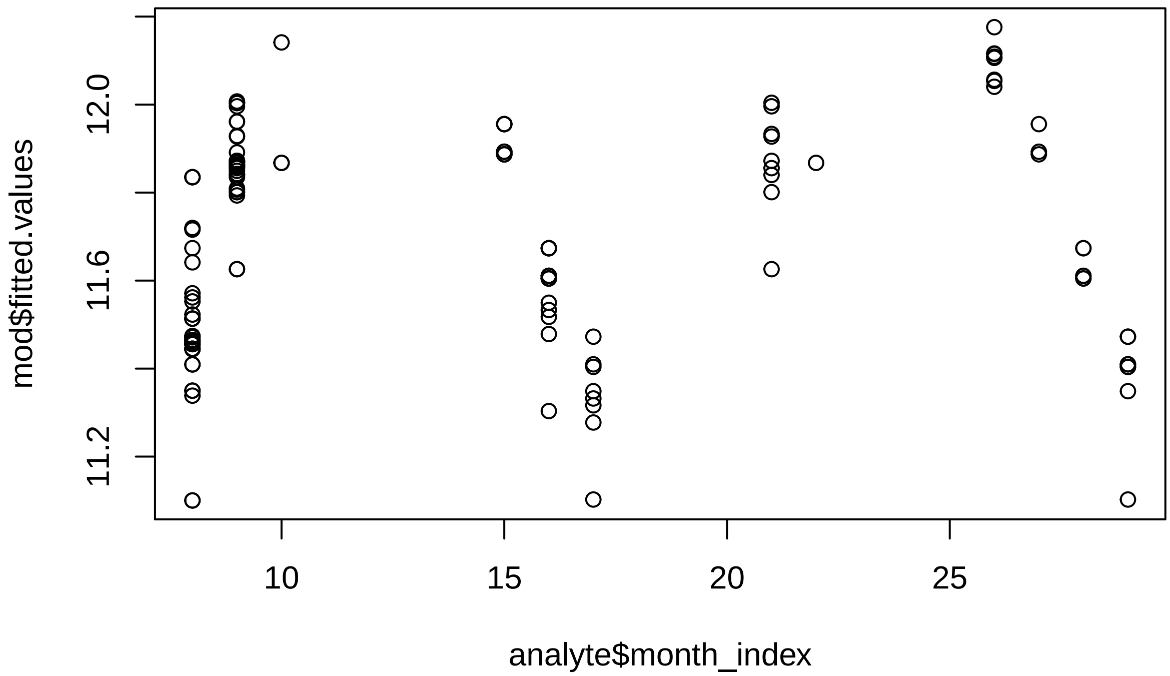
```
## lm(formula = log_obs ~ sin(2 * pi * (month_index/12)) + cos(2 *
##     pi * (month_index/12)) + StationName, data = analyte)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.59074 -0.06254  0.00000  0.07943  0.31226
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     1.185e+01  1.858e-01  63.764  < 2e-16 ***
## sin(2 * pi * (month_index/12)) -2.023e-02  2.595e-02  -0.779   0.4376
## cos(2 * pi * (month_index/12))  5.695e-01  5.733e-02   9.933 2.55e-16 ***
## StationNameFE-4166             -6.384e-02  2.274e-01  -0.281   0.7795
## StationNameFE-4220              5.432e-14  2.603e-01   0.000   1.0000
## StationNameFE-4353B            -2.837e-02  2.603e-01  -0.109   0.9134
## StationNameFE-4520B            -1.408e-02  2.603e-01  -0.054   0.9570
## StationNameFE-4581             -7.712e-02  2.274e-01  -0.339   0.7352
## StationNameFE-4656             -7.018e-03  2.603e-01  -0.027   0.9785
## StationNameFE-4749             -3.559e-02  2.603e-01  -0.137   0.8915
## StationNameFE-4916             -6.111e-02  2.274e-01  -0.269   0.7887
## StationNameFE-5038B            -2.120e-02  2.603e-01  -0.081   0.9352
## StationNameFE-5356B             1.327e-01  2.258e-01   0.587   0.5583
## StationNameFE-5448              2.518e-01  2.618e-01   0.962   0.3385
## StationNameFE-5756              2.518e-01  2.618e-01   0.962   0.3385
## StationNameFE-5858B             1.365e-01  2.258e-01   0.604   0.5471
## StationNameFE-5938              5.826e-02  2.132e-01   0.273   0.7853
## StationNameFE-6274             -1.108e-02  2.274e-01  -0.049   0.9612
## StationNameFE-6528             -1.487e-03  2.274e-01  -0.007   0.9948
## StationNameFE-6768             -8.723e-03  2.274e-01  -0.038   0.9695
## StationNameFE-7688             -2.484e-03  2.274e-01  -0.011   0.9913
## StationNameFE-7858             -1.134e-02  2.143e-01  -0.053   0.9579
## StationNameFE-A31              -2.851e-01  2.145e-01  -1.329   0.1871
## StationNameFE-A33              -2.447e-01  1.985e-01  -1.232   0.2209
## StationNameFE-A35               1.549e-03  2.007e-01   0.008   0.9939
## StationNameFE-A40              -6.963e-02  1.964e-01  -0.354   0.7238
## StationNameFE-A40A              1.335e-01  2.125e-01   0.628   0.5313
## StationNameFE-A41A             -3.056e-02  1.999e-01  -0.153   0.8788
## StationNameFE-A55               6.266e-02  1.960e-01   0.320   0.7499
## StationNameFE-A56               5.674e-02  1.949e-01   0.291   0.7716
## StationNameFE-A64               9.060e-02  2.127e-01   0.426   0.6711
## StationNameFE-A68               1.254e-01  1.949e-01   0.643   0.5215
## StationNameFE-ET-FP01          -1.737e-01  2.272e-01  -0.765   0.4464
## StationNameFE-ET-FP02          -4.828e-01  2.272e-01  -2.126   0.0362 *
## StationNameFE-ET-FP03          -1.224e-01  2.272e-01  -0.539   0.5912
## StationNameFE-ET-FP04          -1.180e-01  2.272e-01  -0.520   0.6045
## StationNameFE-ET-FP05          -1.275e-01  2.272e-01  -0.561   0.5760
## StationNameFE-ET-FP06          -1.202e-01  2.272e-01  -0.529   0.5979
## StationNameFE-ET-FP07          -1.275e-01  2.272e-01  -0.561   0.5760
## StationNameFE-ET-FP08          -1.111e-01  2.618e-01  -0.424   0.6724
## StationNameFE-ET-FP09          -1.090e-01  2.618e-01  -0.416   0.6781
## StationNameFE-ET-FP11          -1.197e-02  2.618e-01  -0.046   0.9636
## StationNameFE-ET-FP13          -2.145e-02  2.618e-01  -0.082   0.9349
## StationNameFE-ET-FP14          -6.029e-02  2.618e-01  -0.230   0.8184
## StationNameFE-ET-FP16          -1.121e-01  2.618e-01  -0.428   0.6695
```

```
## StationNameFE-KT-FP02          -1.380e-01  2.272e-01  -0.607   0.5450
## StationNameFE-KT-FP03          -1.391e-01  2.272e-01  -0.612   0.5419
## StationNameFE-LA3              -1.505e-02  2.075e-01  -0.073   0.9423
## StationNameFE-MT-FP01          -2.333e-01  2.272e-01  -1.027   0.3070
## StationNameFE-OP-67             2.076e-02  2.603e-01   0.080   0.9366
## StationNameFE-OP-72             2.518e-01  2.618e-01   0.962   0.3385
## StationNameFE-OP-73            -7.018e-03  2.603e-01  -0.027   0.9785
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.184 on 94 degrees of freedom
## Multiple R-squared:  0.7362, Adjusted R-squared:  0.5931
## F-statistic: 5.144 on 51 and 94 DF,  p-value: 3.832e-12
```

```r
plot(log_obs~month_index,data=analyte)
```




```r
plot(analyte$month_index,mod$fitted.values)
```



A note on the analytes for which https://stats.stackexchange.com/questions/115090/why-do-i-get-zero-variance-of-a-random-eff

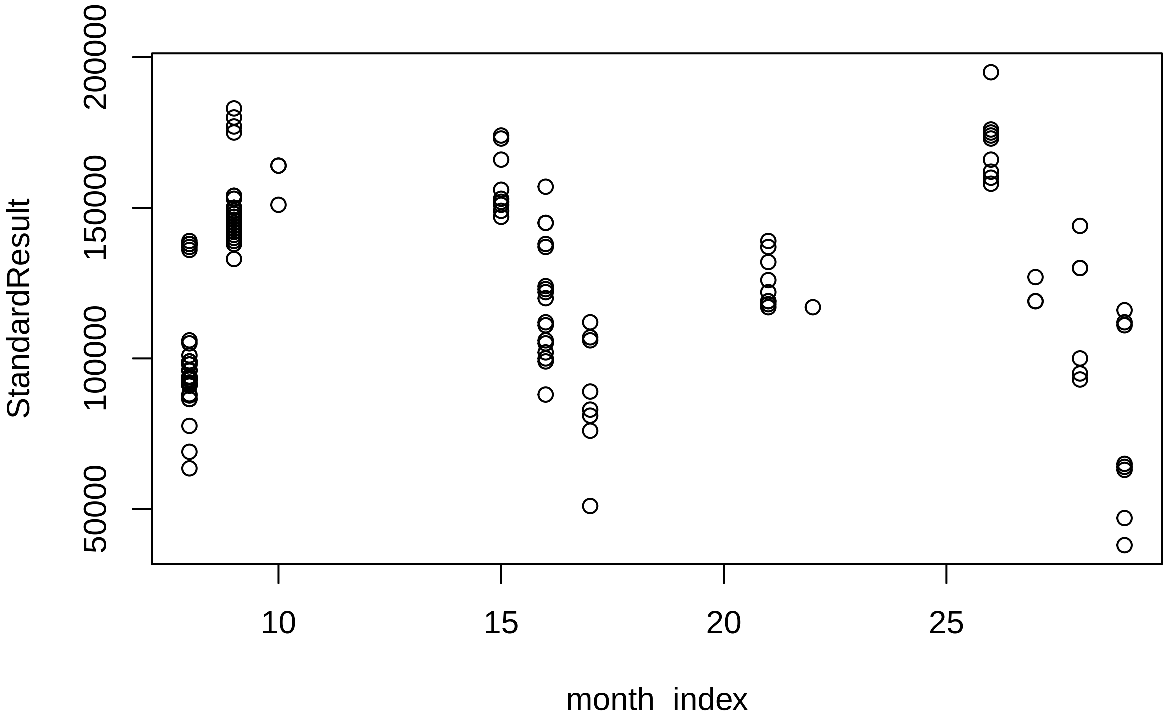
Linear Model with

```
mod <- lm(StandardResult  ~ sin(2*pi*(month_index/12))+cos(2*pi*(month_index/12))+StationName+Year,data=
summary(mod)
```

```
##
## Call:
## lm(formula = StandardResult ~ sin(2 * pi * (month_index/12)) +
##     cos(2 * pi * (month_index/12)) + StationName + Year, data = analyte)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -34365  -8290      0   5222  36039
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     4.305e+05  5.481e+04   7.854 6.83e-12 ***
## sin(2 * pi * (month_index/12))  1.189e+04  3.122e+03   3.809 0.000251 ***
## cos(2 * pi * (month_index/12))  6.422e+04  5.072e+03  12.662  < 2e-16 ***
## StationNameFE-4166              2.219e+02  2.015e+04   0.011 0.991238
## StationNameFE-4220              6.421e-09  2.301e+04   0.000 1.000000
## StationNameFE-4353B            -4.000e+03  2.301e+04  -0.174 0.862387
## StationNameFE-4520B            -2.000e+03  2.301e+04  -0.087 0.930931
## StationNameFE-4581             -1.778e+03  2.015e+04  -0.088 0.929871
## StationNameFE-4656             -1.000e+03  2.301e+04  -0.043 0.965433
## StationNameFE-4749             -5.000e+03  2.301e+04  -0.217 0.828473
## StationNameFE-4916              7.219e+02  2.015e+04   0.036 0.971498
## StationNameFE-5038B            -3.000e+03  2.301e+04  -0.130 0.896561
## StationNameFE-5356B             1.376e+04  1.997e+04   0.689 0.492524
```

```
## StationNameFE-5448            2.552e+04  2.315e+04   1.102 0.273161
## StationNameFE-5756            2.552e+04  2.315e+04   1.102 0.273161
## StationNameFE-5858B           1.426e+04  1.997e+04   0.714 0.476977
## StationNameFE-5938            1.132e+04  1.887e+04   0.600 0.550005
## StationNameFE-6274            8.722e+03  2.015e+04   0.433 0.666121
## StationNameFE-6528            1.022e+04  2.015e+04   0.507 0.613146
## StationNameFE-6768            9.222e+03  2.015e+04   0.458 0.648257
## StationNameFE-7688            1.022e+04  2.015e+04   0.507 0.613146
## StationNameFE-7858            9.580e+03  1.896e+04   0.505 0.614660
## StationNameFE-A31            -2.658e+04  1.898e+04  -1.400 0.164790
## StationNameFE-A33            -1.472e+04  1.758e+04  -0.837 0.404605
## StationNameFE-A35             1.289e+04  1.778e+04   0.725 0.470232
## StationNameFE-A40            -7.678e+03  1.737e+04  -0.442 0.659559
## StationNameFE-A40A            2.779e+04  1.882e+04   1.476 0.143203
## StationNameFE-A41A           -2.777e+03  1.769e+04  -0.157 0.875577
## StationNameFE-A55             7.487e+03  1.736e+04   0.431 0.667290
## StationNameFE-A56             6.788e+03  1.726e+04   0.393 0.695002
## StationNameFE-A64             9.506e+03  1.881e+04   0.505 0.614461
## StationNameFE-A68             1.605e+04  1.726e+04   0.930 0.354680
## StationNameFE-ET-FP01        -2.223e+04  2.009e+04  -1.107 0.271257
## StationNameFE-ET-FP02        -4.623e+04  2.009e+04  -2.302 0.023595 *
## StationNameFE-ET-FP03        -1.748e+04  2.009e+04  -0.870 0.386388
## StationNameFE-ET-FP04        -1.708e+04  2.009e+04  -0.850 0.397305
## StationNameFE-ET-FP05        -1.798e+04  2.009e+04  -0.895 0.373006
## StationNameFE-ET-FP06        -1.728e+04  2.009e+04  -0.860 0.391823
## StationNameFE-ET-FP07        -1.798e+04  2.009e+04  -0.895 0.373006
## StationNameFE-ET-FP08        -1.648e+04  2.315e+04  -0.712 0.478268
## StationNameFE-ET-FP09        -1.628e+04  2.315e+04  -0.703 0.483610
## StationNameFE-ET-FP11        -6.482e+03  2.315e+04  -0.280 0.780107
## StationNameFE-ET-FP13        -7.482e+03  2.315e+04  -0.323 0.747279
## StationNameFE-ET-FP14        -1.148e+04  2.315e+04  -0.496 0.621080
## StationNameFE-ET-FP16        -1.658e+04  2.315e+04  -0.716 0.475610
## StationNameFE-KT-FP02        -1.903e+04  2.009e+04  -0.947 0.345868
## StationNameFE-KT-FP03        -1.913e+04  2.009e+04  -0.952 0.343352
## StationNameFE-LA3            -2.045e+03  1.836e+04  -0.111 0.911586
## StationNameFE-MT-FP01        -2.718e+04  2.009e+04  -1.353 0.179281
## StationNameFE-OP-67           3.000e+03  2.301e+04   0.130 0.896561
## StationNameFE-OP-72           2.552e+04  2.315e+04   1.102 0.273161
## StationNameFE-OP-73          -1.000e+03  2.301e+04  -0.043 0.965433
## Year                         -1.837e+04  3.345e+03  -5.493 3.42e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16270 on 93 degrees of freedom
## Multiple R-squared:  0.8311, Adjusted R-squared:  0.7367
## F-statistic: 8.802 on 52 and 93 DF,  p-value: < 2.2e-16
```

```r
plot(StandardResult~month_index,data=analyte)
lines(mod$fitted.values,analyte$month_index,col=2)
```




```
analyte[,fitted:=mod$fitted.values]
```

Generate PDF of results for each Analyte

```
for ( a in unique(sw$StandardAnalyte)){
  analyte<-sw[StandardAnalyte==a] # Subset data to only relevant analyte

  # Start a PDF document of results
    pdf(paste0("/Users/stubbsrw/Documents/git_code/stubbs_repo/fe_problems/results/surface_water/",a,".p

     # Generate plot for each Analyte over time, for each station.
      for (s in unique(sw$StationName)){
        ts<-MakeAnalyteTSPlot(a=a,s=s) # Get inputs from UI, use function
        print(ts)  # Print it to the PDF
      }

  dev.off() # Close PDF for each
}
```