The GSqwsr R package: Water Quality Surrogate Regressions

By Laura De Cicco and Steve Corsi

February 18, 2015

Contents

Figures

Tables

1 Introduction to GSqwsr package

The GSqwsr (USGS water quality surrogate regressions) package was designed to simplify the process of gathering water quality sample data and unit surrogate data, running a stepwise regression using the USGSwsQW censReg regression function, and analyzing those results. This vignette will first show a general overview workflow (2), then a more detailed description of the workflow with working examples (3).

2 General Workflow

```
library("GSqwsr")

#Sample data included with package:
DTComplete <- StLouisDT
UV <- StLouisUV
QWcodes <- StLouisQWcodes
siteINFO <- StLouisInfo

investigateResponse <- "Ammonia.N"
transformResponse <- "lognormal"

DT <- DTComplete[c(investigateResponse,
                   getPredictVariables(names(UV)),
                   "decYear","sinDY","cosDY","datetime")]
DT <- na.omit(DT)

predictVariables <- names(DT)[-which(names(DT)
                  %in% c(investigateResponse,"datetime","decYear"))]


#Check predictor variables
predictVariableScatterPlots(DT,investigateResponse)

# Create 'kitchen sink' formula:
kitchenSink <- createFullFormula(DT,investigateResponse)

#Run stepwise regression with "kitchen sink" as upper bound:
returnPrelim <- prelimModelDev(DT,investigateResponse,kitchenSink,
                               "BIC", #Other option is "AIC"
                               transformResponse)

steps <- returnPrelim$steps
```

```
modelResult <- returnPrelim$modelInformation
modelReturn <- returnPrelim$DT.mod

# Analyze steps found:
plotSteps(steps,DT,transformResponse)
analyzeSteps(steps, investigateResponse,siteINFO)

# Analyze model produced from stepwise regression:
resultPlots(DT,modelReturn,siteINFO)
resultResidPlots(DT,modelReturn,siteINFO)

# Create prediction plots
predictionPlot(UV,DT,modelReturn,siteINFO=siteINFO)
```

3 Workflow Details

In this section, we will step through the basic workflow.

3.1 Data Retrieval

Data retrieval is currently supported by web service calls to the National Water Information Service (NWIS). The first step is to get the discrete sample data that the regressions are modeling. In this example, we will look at the St Louis River at Scanlon (USGS site ID 04024000). If we don't know the sample data that is available, we can use the whatQW function to discover that information.

```
library(GSqwsr)
```

```
site <- "04024000"
QWcodes <- whatQW(site, minCount=20)
head(QWcodes)
```

```
  parameter_cd  startDate    endDate count service
1        00010 1964-10-28 2015-02-09   282      qw
2        00020 1974-10-30 2015-02-09   171      qw
3        00025 1981-10-20 2015-02-09   150      qw
4        00041 2010-10-07 2015-02-09    56      qw
5        00055 2010-10-07 2013-04-27    35      qw
6        00061 1960-07-28 2014-11-20   317      qw
```

Most likely, there will be a known set of parameters that are to be modeled. If the parameter codes for these analytes are known, the data from NWIS can be accessed directly with the function importNWISqw. The following example shows the process, and then lists the column names returned in the QW dataframe.

```
pCodeQW <- c("00608","00613","00618")
startDate <- "2011-04-22"
endDate <- ""
QW <- importNWISqw(site, params=pCodeQW,
                   begin.date=startDate, end.date=endDate)
```

```
names(QW)
```

```
 [1] "site_no"              "sample_dt"
 [3] "sample_tm"            "tzone_cd"
 [5] "medium_cd"            "Ammonia.N"
 [7] "Nitrite.N"            "Nitrate.N"
 [9] "NO2PlusNO3.N"         "Phosphorus_WW.P"
[11] "OrthoPhosphate.P"     "Chloride"
[13] "NitrogenTotal_WW.sum" "SuspSed"
[15] "datetime"
```

This brings the data in automatically as a "qw" object. This means that censoring information is embedded within each data point. If any processing needs to be done to the data, it might be easier to import the raw data first, then convert to "qw" objects with the makeQWObjects function.

```
QWRaw <- readNWISqw(site,pCodeQW,startDate,
                                endDate,expanded=TRUE)
QW <- makeQWObjects(QWRaw)
```

Next, the unit value data that will be used as surrogates for the analytes should be retrieved. If the parameters are not known, they can be discovered using the getDataAvailability function, filtering just the “uv” (unit value) data:

```
library(dataRetrieval)
UVcodes <- whatNWISdata(site, service = "uv")
names(UVcodes)
```

```
 [1] "parm_cd"            "agency_cd"
 [3] "site_no"            "station_nm"
 [5] "site_tp_cd"         "dec_lat_va"
 [7] "dec_long_va"        "coord_acy_cd"
 [9] "dec_coord_datum_cd" "alt_va"
[11] "alt_acy_va"         "alt_datum_cd"
[13] "huc_cd"             "data_type_cd"
[15] "stat_cd"            "dd_nu"
[17] "loc_web_ds"         "medium_grp_cd"
[19] "parm_grp_cd"        "srs_id"
[21] "access_cd"          "begin_date"
[23] "end_date"           "count_nu"
[25] "parameter_group_nm" "parameter_nm"
[27] "casrn"              "srsname"
```

```
[29] "parameter_units"

UVcodes$parm_cd

[1] "00010" "00021" "00060" "00065" "00095" "00300" "00301"
[8] "00400" "63680"
```

Finally, the unit value data can be retrieved with the getMultipleUV function. Because of the potentially large amount of data being returned, the web service call is automatically split into individual parameter codes.

```
UVpCodes <- c("00010","00060","00095","00300","00400","63680")
UV <- getMultipleUV(site, startDate, endDate, UVpCodes)
```

```
names(UV)

 [1] "agency_cd"    "site_no"      "datetime"     "tz_cd"
 [5] "Wtemp"        "Wtemp_cd"     "Flow"         "Flow_cd"
 [9] "SpecCond"     "SpecCond_cd" "DO"           "DO_cd"
[13] "pH"           "pH_cd"        "Turb"         "Turb_cd"
```

3.2 Data Merging

We now need to merge the sample and continuous data into one dataframe. This is accomplished using the mergeDatasets function. Both QW and UV dataframes need a column called 'datetime' that has the date and time in an POSIXct object. This may need to be done as shown below.

```
QW$datetime <- as.POSIXct(paste(QW$sample_dt," ",QW$sample_tm, ":00",sep=""))

# Make sure they are in consistant time zones:
QW$datetime <- setTZ(QW$datetime, QW$tzone_cd)
UV$datetime <- setTZ(UV$datetime, UV$tz_cd)

mergeReturn <- mergeDatasets(QW, UV, QWcodes)
DTComplete <- mergeReturn$DTComplete
QWcodes <- mergeReturn$QWcodes
```

The dataframe DTComplete contains a column of each of the discrete samples, and a column of the nearest (temporally) unit value data. The function mergeDatasets has an argument called 'max.diff'. The default is set to '2 hours', meaning that if the sample and continuous data timestamps do not match, the merge will take the closest continuous data within 2 hours. This value can be changed, see ?mergeNearest for more options.

3.3 Data Investigation

3.3.1 Narrow down investigation

We now want to narrow our investigation down to one analyte. Let us look at nitrate. First we will want a dataframe DT with just nitrate and the unit values. We will call these the 'prediction values' because they will eventually be used to predict nitrate.

```
investigateResponse <- "Nitrate.N"
predictionVariables <- getPredictVariables(names(UV))

DT <- DTComplete[c(investigateResponse,
                   predictionVariables,
                   "decYear","sinDY","cosDY","datetime")]

names(DT)
```

```
 [1] "Nitrate.N" "Wtemp"     "Flow"      "SpecCond"
 [5] "DO"        "pH"        "Turb"      "decYear"
 [9] "sinDY"     "cosDY"     "datetime"
```

For the regression, there can be no NA values in any of the columns. There are many ways in R to deal with this requirement. The easiest way to do it is remove any row that has any NA. This can be done as follows:

```
DT <- na.omit(DT)
```

There may be other situations where you want to remove a column that contains the majority of the missing data.

3.3.2 Plot variables

There are a few tools included in this package to explore the data before performing the regression.

```
plotQQTransforms(DT,investigateResponse)
```

```
predictVariableScatterPlots(DT,investigateResponse)
```

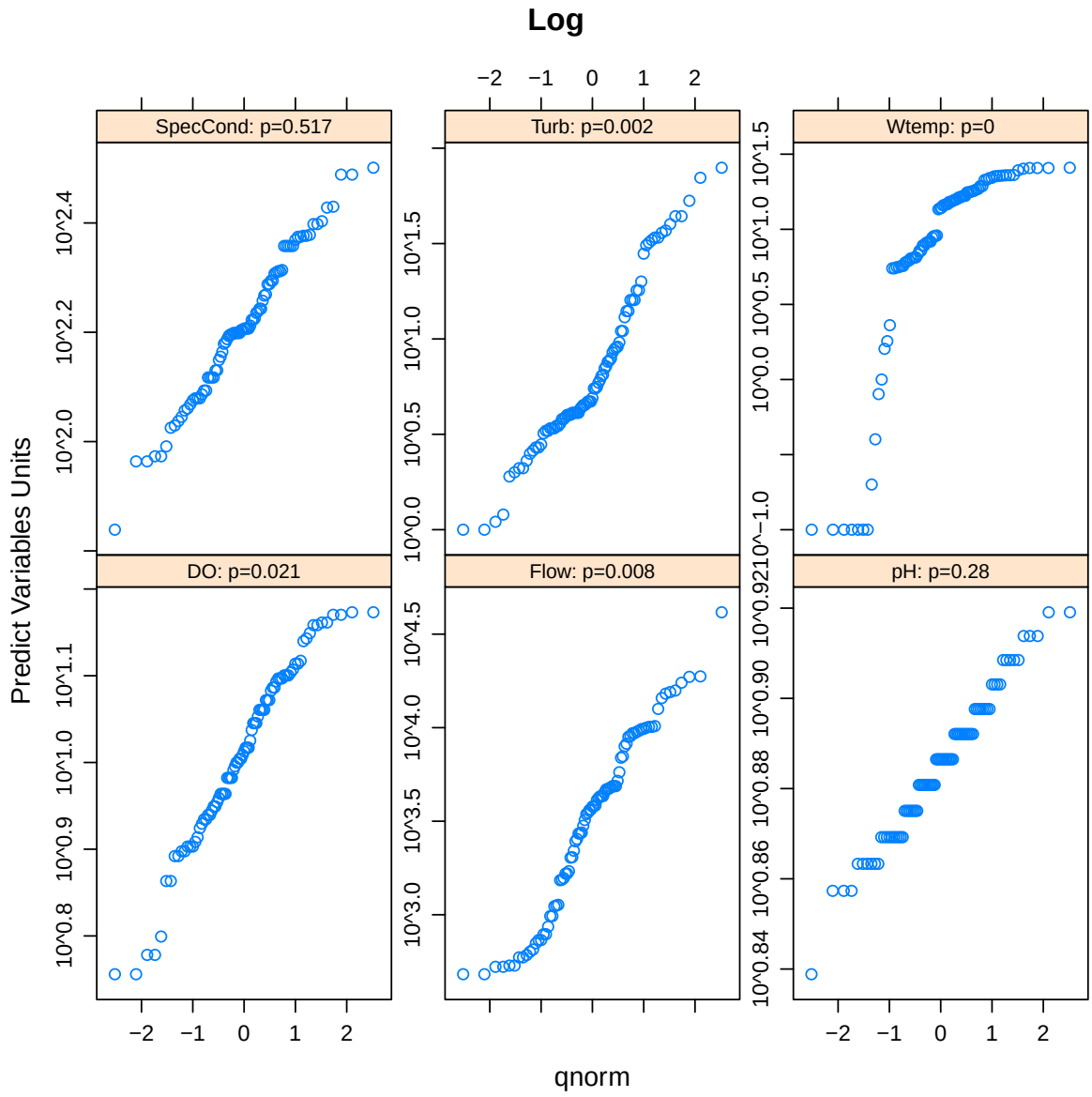


Figure 1. plotQQTransforms

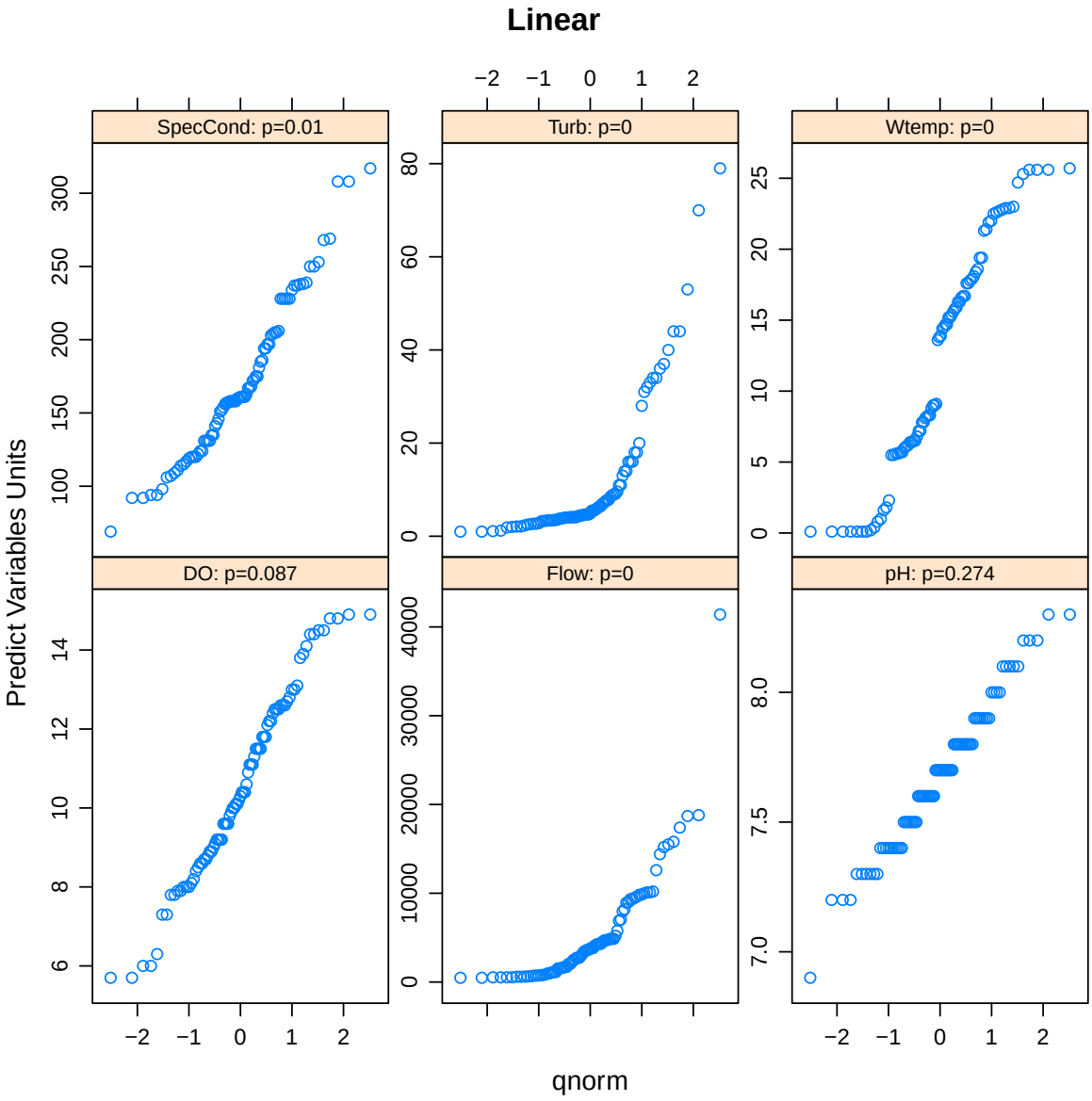


Figure 2. plotQQTransforms

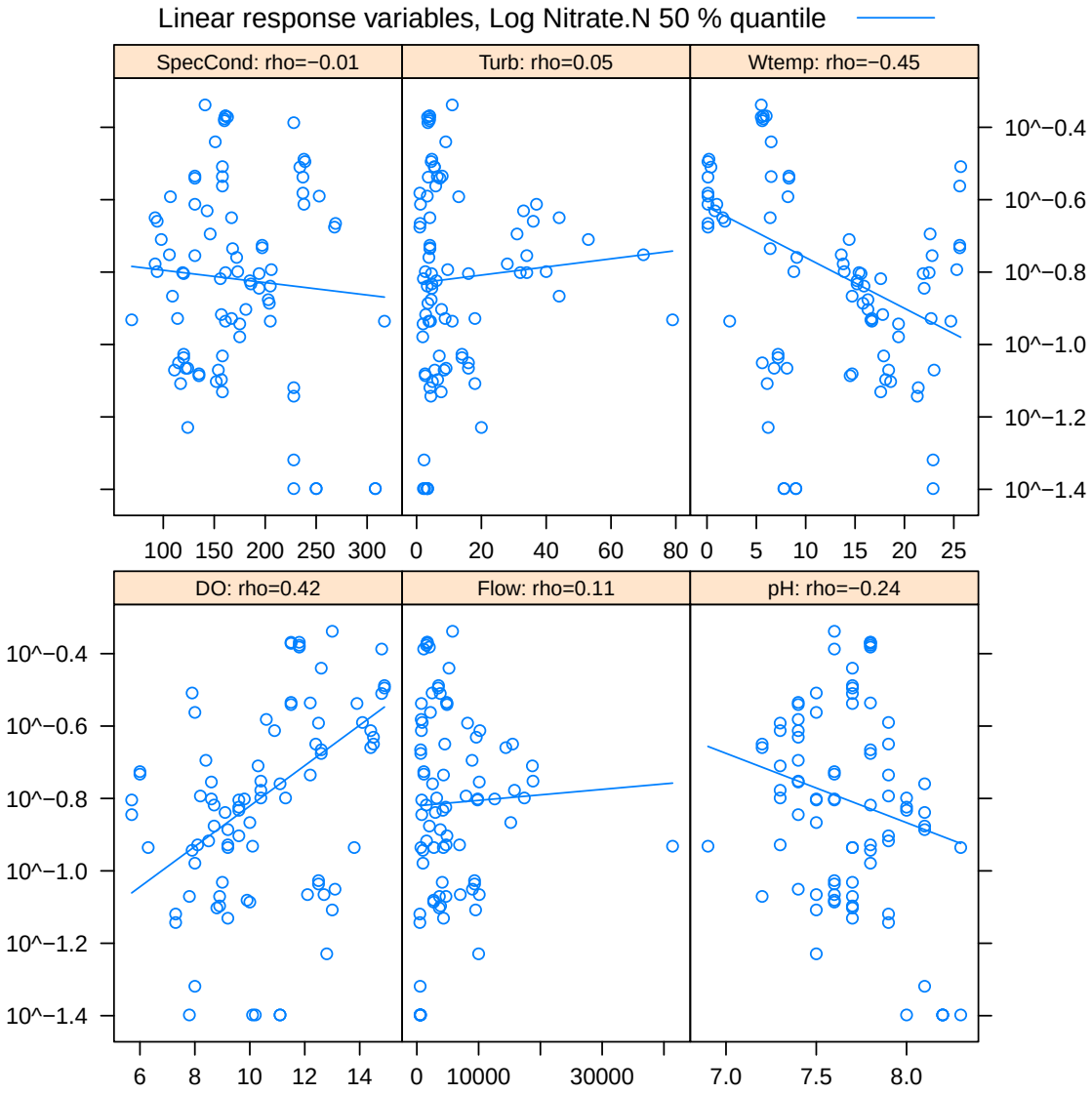


Figure 3. predictVariableScatterPlots

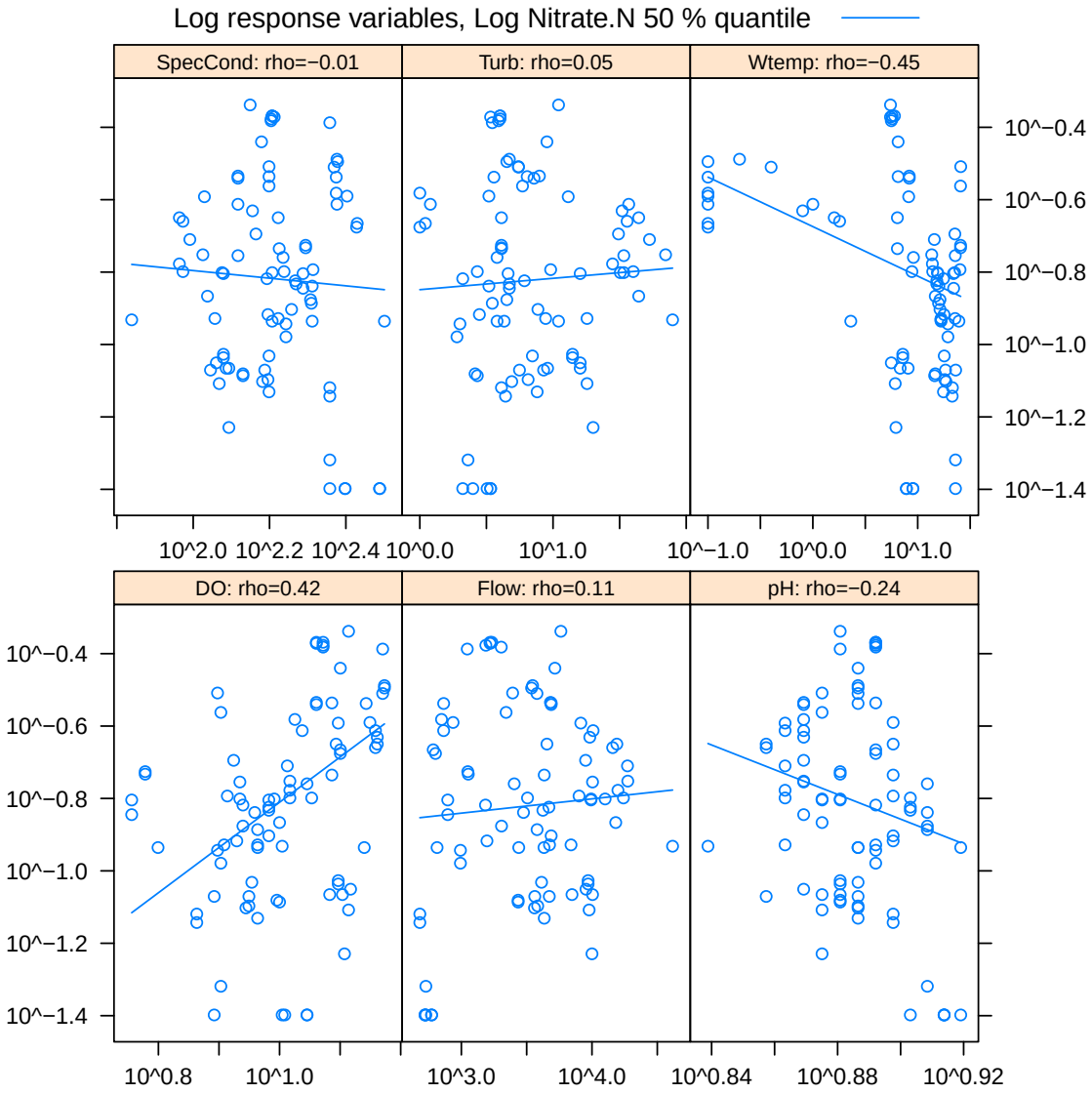


Figure 4. predictVariableScatterPlots

3.4 Stepwise Regression

We are ready to perform a stepwise regression of the data to find the most significant variables to use in the model. This is accomplished with the prelimModelDev function. There are several inputs to this function. DT is the dataframe with all the predictor variables as well as the response we are investigating. We also need to define an upper bound for the stepwise regression to test. This is an equation with all the possible predictor variables, along with their transforms that we are interested in testing. If we want to use all possible variables, and all available transforms, we can use the equation createFullFormula (continuing with our Chloride example):

```
upperBoundFormula <- createFullFormula(DT,investigateResponse)
```

```
[1] "Wtemp + Flow + SpecCond + DO + pH + Turb + sinDY + cosDY + "
[1] "log(Flow) + log(SpecCond) + log(DO) + log(Turb)"
```

The function will check if any data in DT has less than or equal to zero values. If so, a log transform is not included.

Now to use the stepwise regression within the prelimModelDev function. In this function, the DT dataframe is required, the column name of the response variable (in this example, investigateResponse), and the upper bound formula. The user can then choose a value for k which can be 'AIC' (akaike information criterion), 'BIC' (Bayesian information criterion), or a value of the multiple of the number of degrees of freedom used for the penalty. BIC has a harsher penalty for overfitting the model, which is typically seen as a benefit. The default is BIC. Finally, transformResponse can either be 'lognormal' or 'normal', which will define the transformation of the response variable.

Additionally, this function has an argument 'autoSinCos' which is a logical input. The default is set to TRUE, in this case - if the sine of decimal year (sinDY) is picked during the stepwise regression, the next step is forced to be cosDY. Likewise, if cosDY is picked, sinDY is forced on the next step. This feature can be turned off by setting autoSinCos to FALSE.

```
transformResponse <- "lognormal"

returnPrelim <- prelimModelDev(DT,
                 investigateResponse,
                 upperBoundFormula,
                 "BIC", #Other option is "AIC"
                 transformResponse)

Start:  AIC=182.33
Nitrate.N ~ 1

                Df    AIC
+ sinDY          1 154.61
+ DO             1 174.46
+ Wtemp          1 174.95
+ log(DO)        1 176.63
```

```
+ pH             1 177.29
<none>             182.33
+ log(Flow)      1 184.05
+ SpecCond       1 184.18
+ cosDY          1 184.32
+ log(SpecCond)  1 185.37
+ log(Turb)      1 186.42
+ Turb           1 186.43
+ Flow           1 186.64

Step:  AIC=154.61
Nitrate.N ~ sinDY

                Df    AIC
<none>             154.61
+ cosDY          1 155.86
+ pH             1 155.89
+ log(SpecCond)  1 156.03
+ SpecCond       1 157.00
+ Flow           1 157.58
+ log(Flow)      1 157.68
+ log(Turb)      1 158.80
+ log(DO)        1 158.85
+ Wtemp          1 158.90
+ Turb           1 158.99
+ DO             1 159.02
- sinDY          1 182.33
Start:  AIC=155.86
Nitrate.N ~ sinDY + cosDY

                Df    AIC
+ pH             1 154.42
+ Wtemp          1 155.19
+ log(DO)        1 155.83
<none>             155.86
+ DO             1 158.60
+ log(SpecCond)  1 159.53
+ Flow           1 159.90
+ log(Turb)      1 160.18
+ SpecCond       1 160.19
+ Turb           1 160.21
+ log(Flow)      1 160.29

Step:  AIC=154.42
```

```
Nitrate.N ~ sinDY + cosDY + pH

                Df    AIC
+ log(SpecCond)  1 143.44
+ SpecCond       1 150.56
+ Flow           1 152.37
+ Wtemp          1 153.60
+ log(DO)        1 154.31
<none>             154.42
- pH             1 155.86
+ log(Flow)      1 156.25
+ DO             1 156.73
+ Turb           1 156.77
+ log(Turb)      1 157.19

Step:  AIC=143.44
Nitrate.N ~ sinDY + cosDY + pH + log(SpecCond)

                Df    AIC
+ Wtemp          1 138.76
+ SpecCond       1 141.34
<none>             143.44
+ log(DO)        1 145.37
+ DO             1 146.12
+ log(Turb)      1 146.62
+ log(Flow)      1 147.51
+ Turb           1 147.55
+ Flow           1 147.68
- log(SpecCond)  1 154.42
- pH             1 159.53

Step:  AIC=138.76
Nitrate.N ~ sinDY + cosDY + pH + log(SpecCond) + Wtemp

                Df    AIC
+ log(Turb)      1 128.88
+ log(Flow)      1 135.16
+ DO             1 135.28
<none>             138.76
+ log(DO)        1 138.98
+ Turb           1 139.06
+ SpecCond       1 141.84
+ Flow           1 142.28
- Wtemp          1 143.44
```

```
- log(SpecCond)  1 153.60
- pH             1 158.40

Step:  AIC=128.88
Nitrate.N ~ sinDY + cosDY + pH + log(SpecCond) + Wtemp + log(Turb)

                Df    AIC
<none>             128.88
+ DO             1 130.45
+ SpecCond       1 131.88
+ log(DO)        1 132.31
+ Flow           1 132.59
+ Turb           1 132.85
+ log(Flow)      1 133.14
- log(Turb)      1 138.76
- Wtemp          1 146.62
- pH             1 149.28
- log(SpecCond)  1 157.87
Analyzing steps (total= 7 ): 2
3
4
5
6
7

steps <- returnPrelim$steps
modelResult <- returnPrelim$modelInformation
modelReturn <- returnPrelim$DT.mod
```

The output during the function shows the steps that the stepwise regression determined were ideal, information from the final model (modelInformation) such as the terms, their coefficients, standard error, p value as calculated by the censReg function, and standard coefficient (PARAML/STDDEV), and the raw data returned from censReg (DT.mod).

3.5 Stepwise Regression Analysis

It might be a good idea here to verify that the results from the stepwise regression are indeed what you want. The process can be observed using two functions: plotSteps and analyzeSteps.

analyzeSteps creates a plot with correlation, slope, RMSE, PRESS, and AIC statistics. Correlation and slope are values that should trend towards 1. Correlation is the correlation between observed and predicted, slope is the slope of observed and predicted. RMSE should trend towards zero, RMSE is the root mean squared error of the observed vs. predicted data. PRESS (predicted residual sums of squares) is calculated from the external studentized residuals, and should trend downward (for a better model

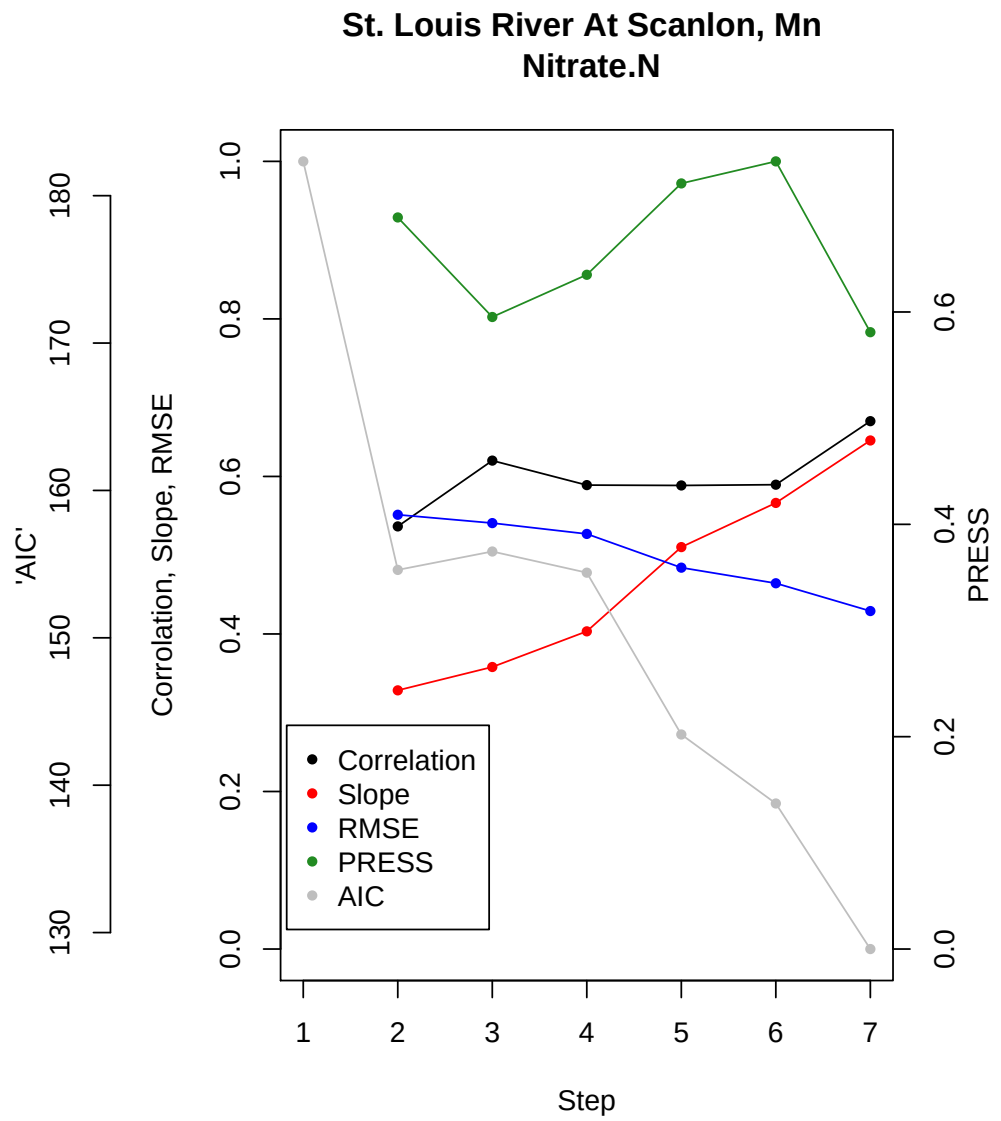


Figure 5. analyzeSteps

fit). Residuals are calculated for censored values using the detection limit. AIC (akaike information criterion) is returned from the ANOVA output of the stepwise regression. It is always called 'AIC' whether or not 'AIC' or 'BIC' was specified. AIC will also trend downward for better model fits.

```
siteINFO <- readNWISsite(site)
analyzeSteps(steps, investigateResponse,siteINFO,
             xCorner=0.01,yCorner=0.3)
```

In this case, it may seem strange that the AIC value goes up then down. This is because the first

parameter that was picked was sinDY (sine of decimal year). As mentioned earlier, if sinDY is picked, we automatically force cosine to be the next parameter.

plotSteps shows the observed versus predicted for each step along the way of the stepwise regression.There are two lines included, the blue line is a one-to-one line, the red line is the slope of the observed versus predicted values as calculated with a linear regression (lm). In this simple regression, censored values are taken as their detection limits. Red points indicate potential outliers as calculated based on external studentized residuals greater than 3 or less than -3. Censored values are represented with a line segment from the detection limit to zero (for left-censored data).

```
m <- t(matrix(c(1:6), nrow = 2, ncol = 3))
layout(m)
par(mar=c(2,2,2,2))
plotSteps(steps,DT,transformResponse)
```

```
Nitrate.N ~  sinDY
Nitrate.N ~  sinDY + cosDY
Nitrate.N ~  sinDY + cosDY + pH
Nitrate.N ~  sinDY + cosDY + pH + log(SpecCond)
Nitrate.N ~  sinDY + cosDY + pH + log(SpecCond) + Wtemp
Nitrate.N ~  sinDY + cosDY + pH + log(SpecCond) + Wtemp + log(Turb)
```

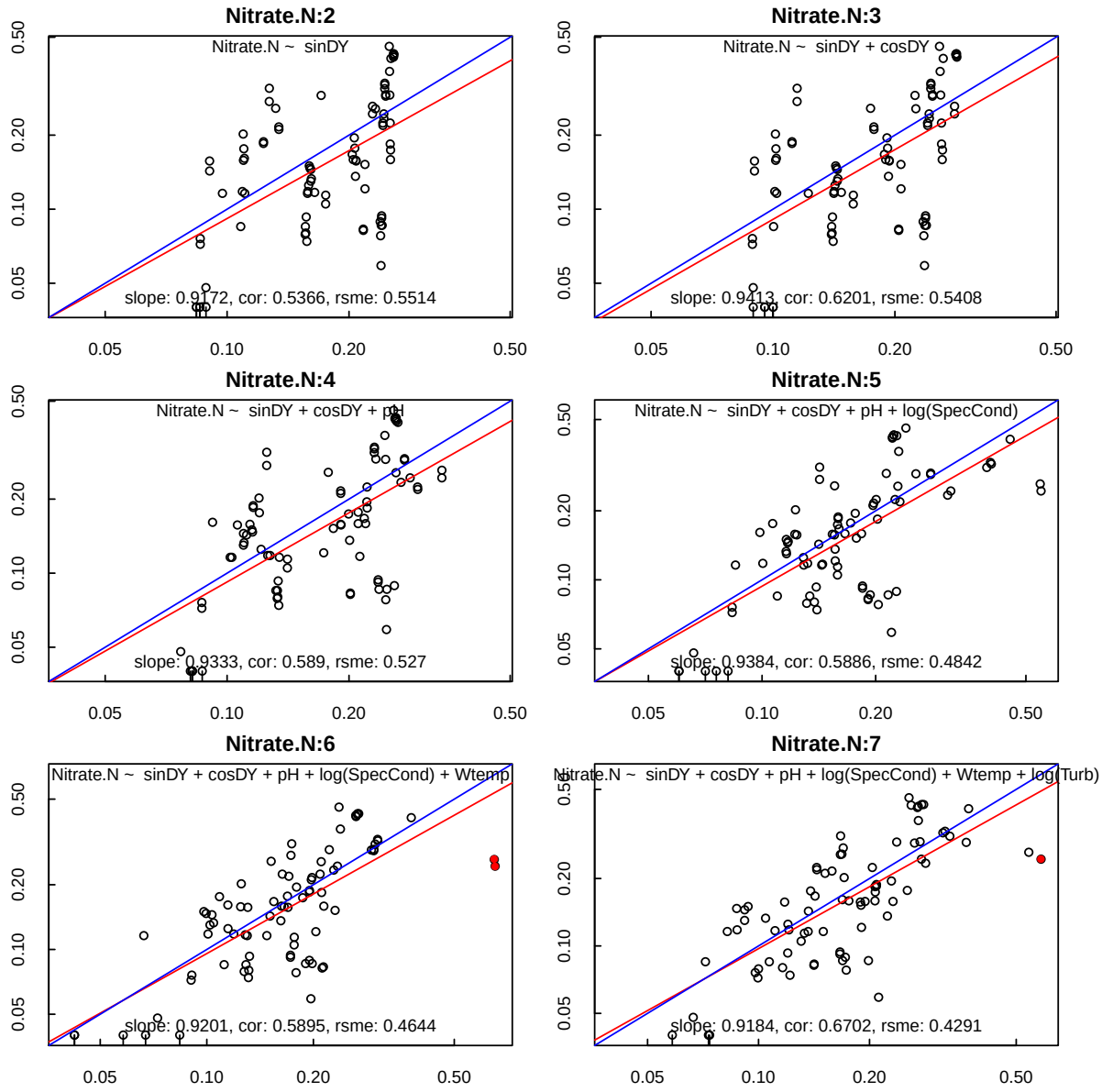


Figure 6. plotSteps

3.6 Model Adjustments

There may be situations in which the user wants to explore alternative models compared to the results of the stepwise regression. The first tool offered in the package for this type of work is the function generateParamChoices. This function will create a dataframe, and optionally save it to a csv file with all of the parameter choices.

```
#Change this to a relavent path:
pathToSave <- "C:/RData/"
choices <- generateParamChoices(predictionVariables,
                                modelReturn,pathToSave,save=TRUE)
```

This produces a file that can be opened in Microsoft Excel, or any text editor:

	A	B	C	D	E	F	G	H	I	J	K	L
1	variableNames	Scalar	Wtemp	Flow	SpecCond	DO	pH	Turb	log(Flow)	log(SpecCond)	log(DO)	log(Turb)
2	Wtemp	1	0	0	0	0	0	0	0	0	0	0
3	Flow	0	0	0	0	0	0	0	0	0	0	0
4	SpecCond	0	0	0	0	0	0	0	0	0	0	0
5	DO	0	0	0	0	0	0	0	0	0	0	0
6	pH	1	0	0	0	0	0	0	0	0	0	0
7	Turb	0	0	0	0	0	0	0	0	0	0	0
8	log(Flow)	0	0	0	0	0	0	0	0	0	0	0
9	log(SpecCond)	1	0	0	0	0	0	0	0	0	0	0
10	log(DO)	0	0	0	0	0	0	0	0	0	0	0
11	log(Turb)	1	0	0	0	0	0	0	0	0	0	0

Figure 7. Output of generateParamChoices shown in Excel

The first column, 'Scalar', is pre-populated with zeros and ones, where ones represent the variables picked in the stepwise regression. Changing the 1's and 0's in this column will allow the user to easily set which parameters should be modeled. So, adding a 1 to the Flow row in the Scalar column will tell the program to include Flow in the model (as well as any other parameters with 1's). The next set of columns are used to allow users to include interaction terms. For example, if the interaction between log(Flow) and Turbidity was thought to be useful, a 1 in row 8, column H (log(Flow):Turb) would be required.

Once the parameter choice file is adjusted, it can be read in using read.csv, and a new formula can be created using the createFormulaFromDF function.

```
choicesNew <- read.csv(pathToSave)
newFormula <-createFormulaFromDF(choicesNew)
```

```
newFormula

[1] "Wtemp + Flow + pH + log(SpecCond) + log(Turb) + Turb:log(Flow)"
```

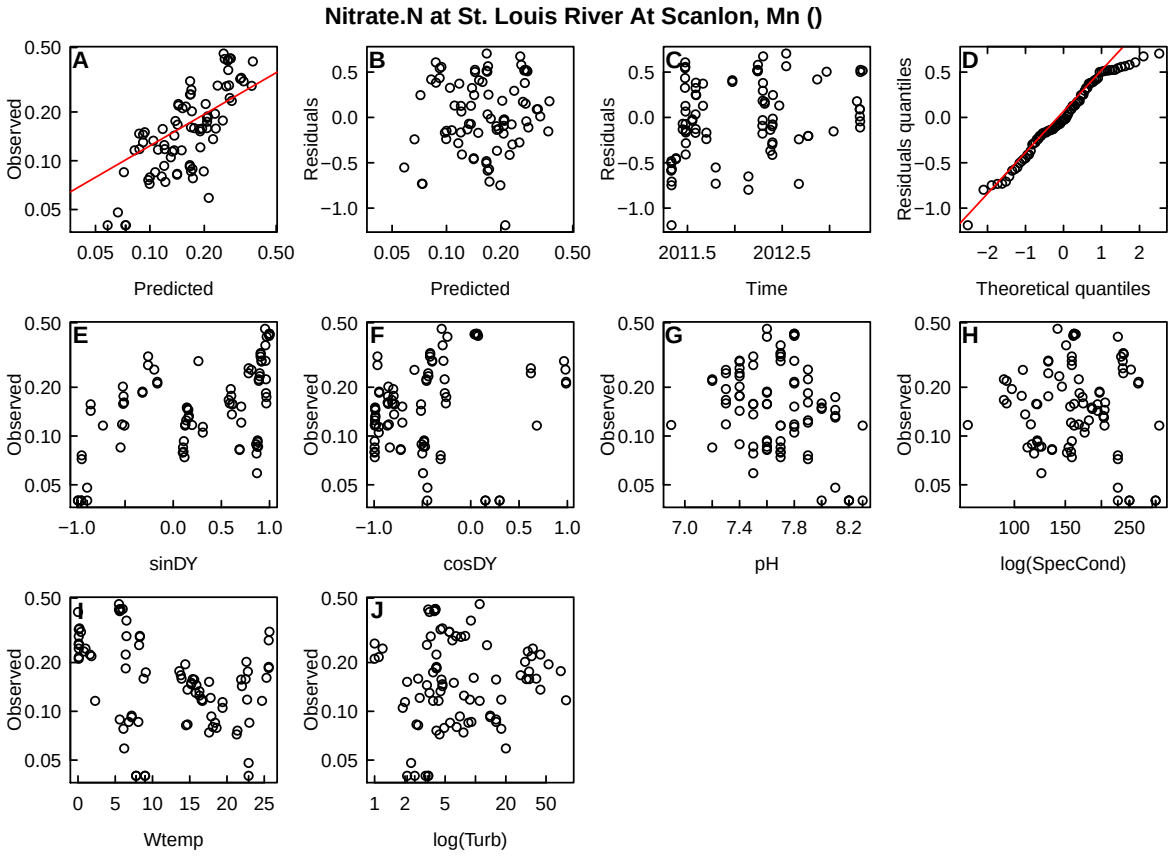
From this formula, the stepwise regression routine can be re-run (if certain parameters were deleted for example), or the model can be created:

```
newUpperFormula <- paste(investigateResponse," ~ ", newFormula, sep="")
modelReturnCustom <- censReg(newUpperFormula,
                        dist=transformResponse, data=DT)
modelReturnCustom

Call:
censReg(formula = Nitrate.N ~ Wtemp + Flow + pH + log(SpecCond) +
    log(Turb) + Turb:log(Flow), data = DT, dist = transformResponse)

Coefficients:
                 Estimate Std. Error  z-score p-value
(Intercept)     7.879e+00  2.538e+00  3.10485  0.0017
Wtemp          -3.029e-02  7.959e-03 -3.80579  0.0002
Flow           -4.829e-05  3.452e-05 -1.39901  0.1440
pH             -1.131e+00  3.494e-01 -3.23676  0.0011
log(SpecCond)  -1.023e-01  4.238e-01 -0.24147  0.7980
log(Turb)      -1.343e-02  1.416e-01 -0.09488  0.9235
Turb:log(Flow)  7.178e-04  1.385e-03  0.51844  0.5872

Estimated residual standard error (Unbiased) = 0.574
Distribution: lognormal
Percent standard error: 62.47
Positive percent error: 77.54
Negative percent error: -43.68

Number of observations = 85, number censored = 5 (5.9 percent)

Loglik(model) = -72.23 Loglik(intercept only) = -86.72
  Chi-square = 28.98, degrees of freedom = 6, p-value = <0.0001

Computation method: AMLE
```

3.7 Model Analysis

Finally, the package offers several ways to analyze and report on the model results. We will look at the results of the original model returned from the stepwise regression (not the custom model we created in the last section). Censored values are plotted as points with segments (from the detection limit towards zero for left-censored data). Also, left-censored residuals are calculated using the detection limit.

The function resultPlots plots a set of plots. All model results include observed vs. predicted (A), residuals vs. predicted (B), residuals vs. time (C), and residual quantiles vs. theoretical quantiles (D). After that, there is a plot for observed vs. each parameter in the model (E...).



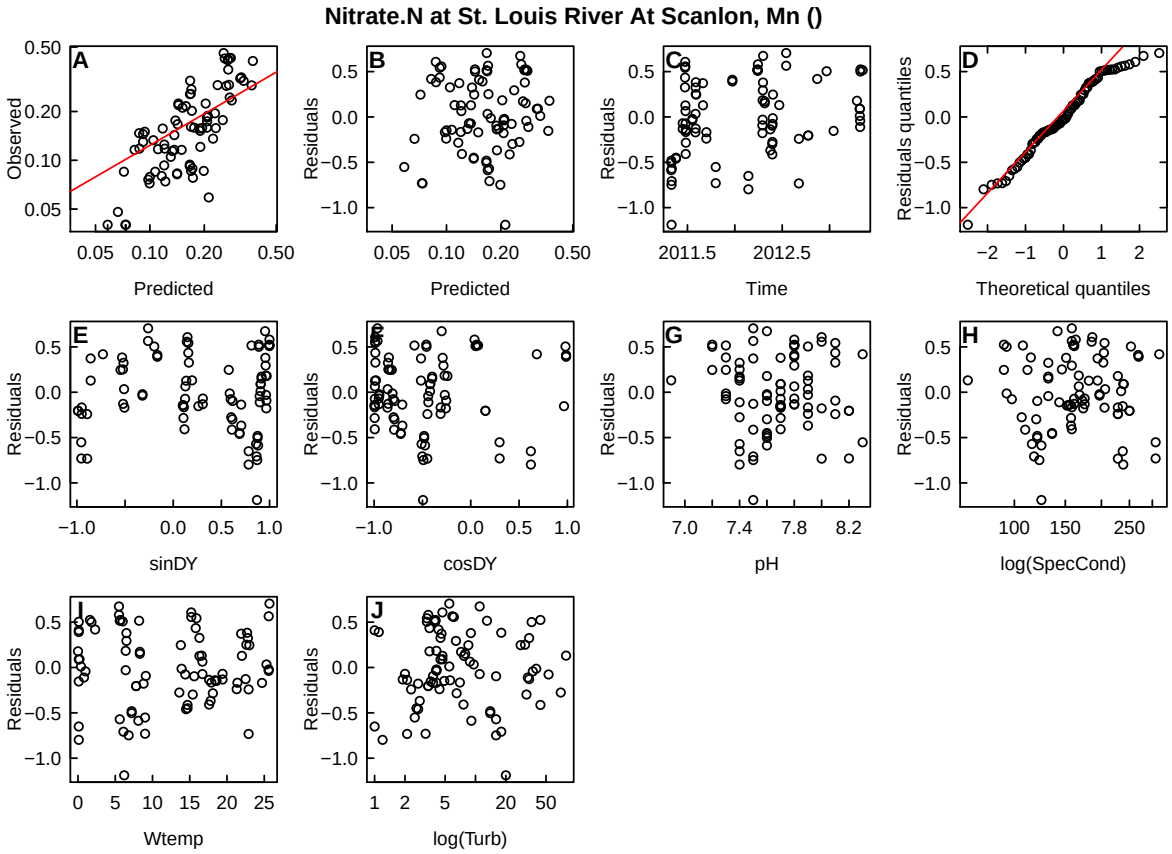
ln(Nitrate.N) = −5.844 (Intercept) + 1.423 sinDY + 0.8419 cosDY + −1.345 pH + 2.461 log(SpecCond) + 0.0841 Wtemp + 0.3604 log(Turb)

Figure 8. resultPlots

```
resultPlots(DT,modelReturn,siteINFO)
```

The function resultResidPlots plots a set of plots. All model results include observed vs. predicted (A), residuals vs. predicted (B), residuals vs. time (C), and residual quantiles vs. theoretical quantiles (D). After that, there is a plot for residuals vs. each parameter in the model (E...).

```
resultResidPlots(DT,modelReturn,siteINFO)
```



ln(Nitrate.N) = −5.844 (Intercept) + 1.423 sinDY + 0.8419 cosDY + −1.345 pH + 2.461 log(SpecCond) + 0.0841 Wtemp + 0.3604 log(Turb)

Figure 9. resultResidPlots

The predictionPlot function plots the predicted values based on all of the unit value data available (from the UV dataframe) in blue. Red dots representing the actual measured data are also included. Left-censored points are shown at their detetction limit, with a segment going towards zero.

```
predictionPlot(UV,DT,modelReturn,siteINFO=siteINFO)
```

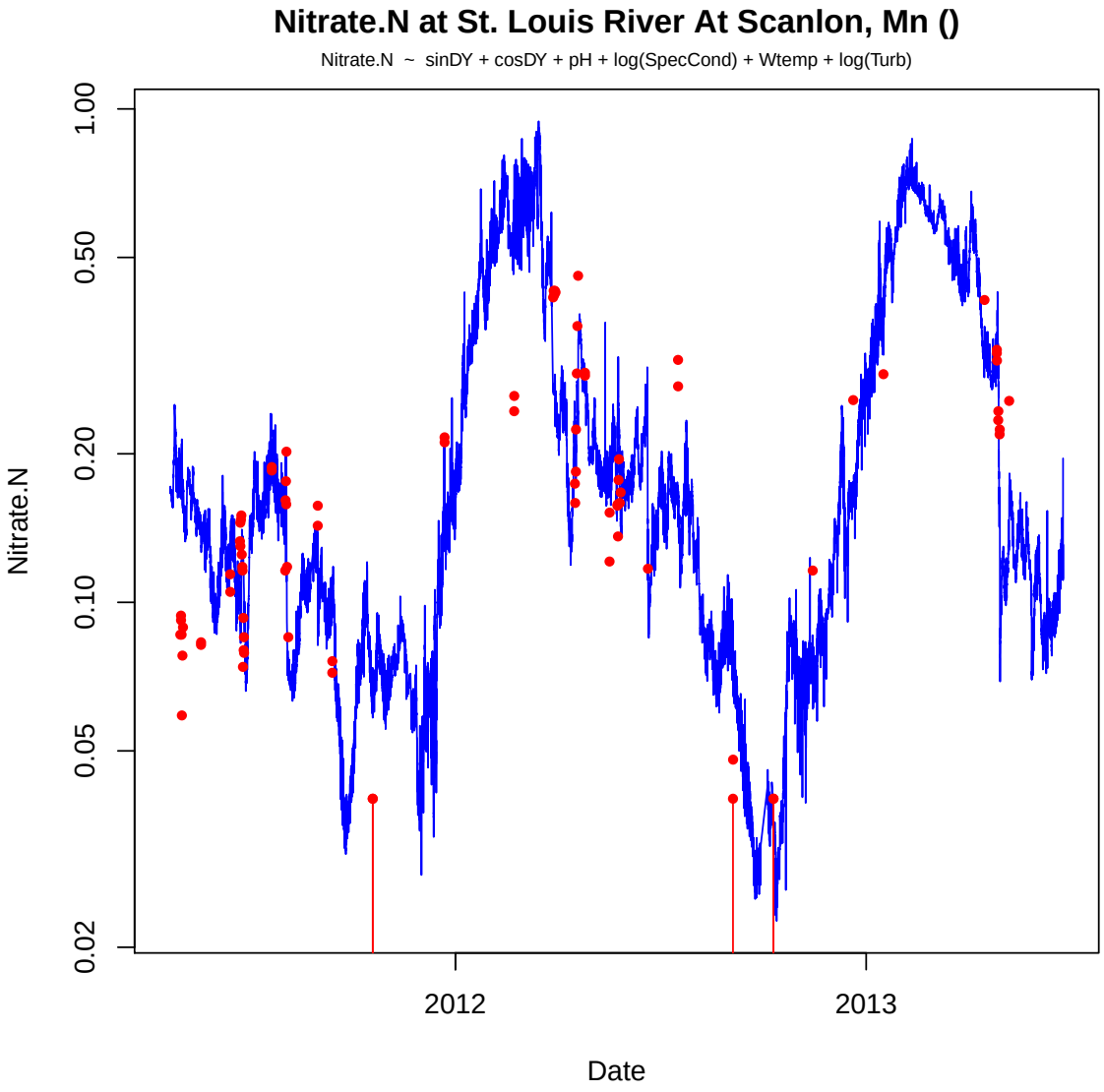


Figure 10. predictionPlot

Finally, a summary printout can be obtained with the function summaryPrintout in either the R console or saved to a file.

```
summaryPrintout(modelReturn, siteINFO)

Nitrate.N  at  St. Louis River At Scanlon, Mn  ( )
Number of observations:  85
Distribution:  lognormal
Method:  AMLE
Degrees of freedom:  14
RMSE:  0.4291243
RSQ:  61.5314
Number of censored values:  5
Nitrate.N ~ (Intercept) + sinDY + cosDY + pH + log(SpecCond) + Wtemp + log(Turb

           Term Coefficient StdError pValue StCoef
1   (Intercept)      -5.844    2.336   0.01 -2.502
2         sinDY       1.423    0.178   0.00  8.002
3         cosDY       0.842    0.211   0.00  3.996
4            pH      -1.345    0.264   0.00 -5.099
5 log(SpecCond)       2.461    0.405   0.00  6.074
6         Wtemp       0.084    0.018   0.00  4.768
7     log(Turb)       0.360    0.095   0.00  3.774
8      logSigma       0.184    0.028   0.00  6.481

Correlation matrix of coefficients:
              (Intercept)   sinDY   cosDY      pH
(Intercept)        1.0000 -0.6214 -0.3342 -0.4422
sinDY             -0.6214  1.0000  0.6207 -0.0938
cosDY             -0.3342  0.6207  1.0000  0.1125
pH                -0.4422 -0.0938  0.1125  1.0000
log(SpecCond)     -0.5749  0.6560  0.1692 -0.4760
Wtemp             -0.5114  0.8580  0.8391 -0.0300
log(Turb)         -0.7868  0.6155  0.4931  0.0740
logSigma          -0.0059  0.0376  0.0142 -0.0228
              log(SpecCond)   Wtemp log(Turb) logSigma
(Intercept)         -0.5749 -0.5114   -0.7868  -0.0059
sinDY                0.6560  0.8580    0.6155   0.0376
cosDY                0.1692  0.8391    0.4931   0.0142
pH                  -0.4760 -0.0300    0.0740  -0.0228
log(SpecCond)        1.0000  0.4712    0.6765   0.0242
Wtemp                0.4712  1.0000    0.5820   0.0283
log(Turb)            0.6765  0.5820    1.0000   0.0144
logSigma             0.0242  0.0283    0.0144   1.0000
```

A Getting Started in R

This section describes the options for downloading and installing the GSqwsr package.

A.1 New to R?

If you are new to R, you will need to first install the latest version of R, which can be found here: *http://www.r-project.org/*.

There are many options for running and editing R code, one nice environment to learn R is RStudio. RStudio can be downloaded here: *http://rstudio.org/*. Once R and RStudio are installed, the dataRetrieval package needs to be installed as described in the next section.

At any time, you can get information about any function in R by typing a question mark before the functions name. This will open a file (in RStudio, in the Help window) that describes the function, the required arguments, and provides working examples.

```
library(GSqwsr)
?plotSteps
```

To see the raw code for a particular code, type the name of the function:

```
plotSteps
```

A.2 R User: Installing QWSR

Before installing GSqwsr, the dependent packages must be first be installed:

```
install.packages(c("USGSwsBase","USGSwsStats",
                   "USGSwsData","USGSwsGraphs",
                   "USGSwsQW","dataRetrieval","GSqwsr"),
      repos=c("http://usgs-r.github.com","http://cran.us.r-project.org"),
      dependencies=TRUE)
```

After installing the package, you need to open the library each time you re-start R. This is done with the simple command:

```
library(GSqwsr)
```