EXPLORATION AND GRAPHICS FOR RIVER TRENDS (EGRET):
AN R-PACKAGE FOR THE ANALYSIS OF LONG-TERM CHANGES IN WATER QUALITY AND STREAMFLOW, INCLUDING THE WATER-QUALITY METHOD WEIGHTED REGRESSIONS ON TIME, DISCHARGE, AND SEASON (WRTDS) DRAFT MANUAL

Authors: Robert M. Hirsch and Laura A. De Cicco, U.S. Geological Survey

This package is available for beta testing and is not yet an approved USGS product. Because it is important that the package and manual get extensive user testing before being released, users are encouraged to provide feedback to the developers about aspects of the package that: 1) don't work, 2) aren't well explained, or 3) could be improved. Feedback and questions should be directed to egret_comments@usgs.gov. Users are encouraged to sign up for e-mail notices by sending a statement of interest to egret_comments@usgs.gov. Also, users should regularly check the web site for updates: https://github.com/USGS-R/EGRET/wiki

The following are 4 major features of EGRET.

1. It is designed to obtain its water quality sample data, streamflow data, and metadata directly from the USGS NWIS (National Water Information System), but it allows for user-supplied text files as inputs. The program is designed to ingest the data directly into R and structure them into file structures suited to the analysis. For those familiar with WRTDS_4c, the text file inputs used in that system will also work in EGRET.

2. It has all of the existing **WRTDS** functionality - computing concentrations, fluxes, flow normalized versions of those, trends in those and graphics to show results and to explore the behavior of the data (by season, by flow class...). Many graph and table outputs are possible and all are clearly labeled and suitable for presentation or publication. It is designed for both batch and interactive processing. It is very much oriented to graphics and should be thought of as an exploratory tool. It is intended for use with data sets of about 200 or more samples, over a time period of about 20 or more years. Some testing with smaller data sets has been done, and no significant problems have been identified in cases with sample sizes slightly larger than 100 but extensive testing with smaller data sets has not taken place yet.

3. It has additional statistics and graphics to help evaluate the possibility that flux estimates may be biased (it is known that in certain cases, regression-based methods can produce severely biased flux estimates). It can also accept results from other estimation methods like LOADEST and produce the same types of graphics and statistics for them (this part is not yet documented).

4. It has a streamflow history component, not related to water quality, that is not a part of WRTDS, but uses some similar concepts and shares some of the basic software and data structures. This component, called **flowHistory** provides a variety of table and graphical outputs looking only at flow statistics (like annual mean, annual 7-day low flow, annual 1-day maximum, or seasonal versions of these) all based on time-series

smoothing. It was designed for studies of long-term streamflow change (associated with climate or land use or water use change) and works best for daily streamflow data sets of 50 years or longer. It has been packaged together with the WRTDS method because it uses the same data retrieval method as WRTDS and the same data structure.

This initial version of the manual will only cover fairly standard implementation of the system. Although EGRET is designed to be very exploratory and facilitate looking at other methods of estimation (e.g. comparing WRTDS and LOADEST estimates, or comparing smoothed time series versus linear regression estimates of change), this early draft of the manual doesn't explain how to do this. Those sections will be added in later versions of the manual.

The best way to learn about the WRTDS approach and to see examples of its application to multiple large data sets is to read two journal articles. Both are available, for free, from the journals in which they were published.

The first relates to nitrate and total phosphorus data for 9 rivers draining to Chesapeake Bay. The URL is:
http://onlinelibrary.wiley.com/doi/10.1111/j.1752-1688.2010.00482.x/full

The second is an application to nitrate data for 8 monitoring sites on the Mississippi River or its major tributaries. The URL is:
http://pubs.acs.org/doi/abs/10.1021/es201221s

The manual available here assumes that the user understands the concepts underlying WRTDS. Thus, reading at least the first of these papers is necessary to understanding the manual. The method has been enhanced beyond what was published there. The enhancement is that it now properly handles censored data by using survival regression rather than ordinary regression. The details of that are in a manuscript currently in process by Doug Moyer and Bob Hirsch.

Table of Contents

Setup of software:

This manual assumes that the user has a basic knowledge of R (an open-source statistical programming language). General information and software can be downloaded from: http://www.r-project.org/. The software is platform-independent and free. You should have R 2.13.0 or higher installed on your computer. [Just a note, the EGRET package does not operate within the "R Commander" environment]. You should also make sure that you have installed up-to-date versions of the other packages that are called by the EGRET system. These are: zoo, survival, splines, reshape, fields, spam, and plyr. The code, help pages, and example data sets for the full EGRET system can be obtained from: https://github.com/USGS-R/EGRET.

dataRetrieval_x.x.x.tar.gz, and EGRET_x.x.x.tar.gz (x.x.x release numbers change over time).

How to get started:

Start R on your computer. Then install these two packages. Note that you only have to do this once. For each subsequent use of the packages you don't need to install them again. However, you should check back to the website to see if there is an updated version. If so, follow this same procedure again with the new version.

A. Package installation

First install the package called `dataRetrieval`, the command (on any system) is:

```
install.packages("../dataRetrieval_1.xx.tar.gz",repos=NULL,type="source")
```

Note that the ".../" signifies that you need to give it the full pathname to the file. The style of designating a file name is different on a Windows computer, the "\" is used rather than the "/". So, for example on my Macintosh computer the full command is

```
install.packages("/Users/rhirsch/Desktop/WRTDS_CIDA/MainRCodes/EGRET_1.0.tar.
gz",repos=NULL,type="source")
```

Also note that on a Macintosh, in the R GUI all you have to do to execute this command is type

```
install.packages("",repos=NULL,type="source")
```

then drag the icon for this file into the space between the quotes and the full path name will drop in and you have the full command.

Then you install the second package called `EGRET`

```
install.packages("../EGRET_1.xx.tar.gz",repos=NULL,type="source")
```

Once you have done this installation you don't have to do it again on subsequent uses of the packages until a new release comes out. When that new release comes out you just install it as instructed above. You don't need to "uninstall" the old one.

B. Loading the library

The next commands need to be used each time you start up. They load the packages and identify all the other packages that R will need to run these. If you get messages that these other packages (like survival, zoo, and several others) are not found, then you will need to install them in the way you install other R packages off of the web (this is machine dependent).

```
library(dataRetrieval)
```

```
library(EGRET)
```

You can get a list of all the functions and objects in the packages the same way you can get them for any R package, using the help system. For any function, for example `plotConcHist`, just enter

```
?plotConcHist
```

and you will get the information on that function. At the bottom of the help page you will see a hot link to the Index, you can go there to see a full list of the items in the package.

Appendices A and B provide lists of all of the functions that are designed to be called by the user and some brief information about their purpose and their arguments. The contents of A and B only differ in the way they are sorted (alphabetical or by purpose). Sections 2, 3, and 4 will all assume a “standard” implementation of WRTDS and flowHistory that does not include evaluating alternatives (like alternative window widths or even alternative methods). Looking through the argument lists you will see things like `localSample = Sample`. For now you can totally ignore those arguments that involve the word “local” and you don’t need to specify them in calling the functions. One note for those less familiar with R: When a function call shows an argument name, followed by an equals sign, the value after the equal sign is the default. The function will work without you specifying any value for that argument. If you want some value other than the default, then it needs to be entered in the form argumentname = value.

Arguments with no defaults listed **must** be entered by the user. The easiest thing is to simply enter them in the order they are listed in the call. But, they can also be entered with an equal sign. When an equal sign is used the arguments can appear in any order. You will see that for many basic implementations many of the arguments never need to have anything entered, the defaults will give you what you want in many cases.

1. Data input:

This section is written in a manner that applies to using EGRET for a WRTDS application (water quality study). It describes how to go about populating 3 data frames: `Sample`, `Daily`, and `INFO`. If you are doing a flowHistory application and not a WRTDS (water quality) application, you can ignore all of the information about creating the `Sample` data frame. You will only need to create the `Daily` and `INFO` data frames. *Inserted into the section in italic print are descriptions of this process for a flowHistory application (although most of the steps are identical or nearly so).*

For WRTDS applications you will need to create 3 data frames. One for the water quality data, which goes in the data frame called "`Sample`". One for the daily streamflow data, which goes in the data frame called "`Daily`". And one for the meta-data regarding the site and the constituent, and that is called "`INFO`". Throughout sections 2-4, they will always have exactly these names. For more advanced uses you will need to give them alternative names (for example: "DailyWRTDS" and "DailyLOADEST" if we wanted to consider a comparison of these two methods). This will be explained in section 6. One other note: for some analyses we want to consider a water quality variable that is the sum of two or more analytes (say dissolved and suspended phosphorus). The methods of data retrieval are all designed to handle that, but in the interest of simplicity this section will assume that the variable of interest is a single analyte. Chapter 7 considers input data that is the sum of multiple analytes.

A. Input of water quality sample data

There are two approaches that can be used to bring in the water quality data. One is to bring in USGS NWISWeb data directly from the Internet. It uses web services located at: http://qwwebservices.usgs.gov/. The other is to bring in the data from a file, assumed here to be a "csv" file (comma separated values, a standard output format from Excel).

a) Water quality data retrieval from USGS web services

To obtain the data from the Internet, you need to know the station number (eight digit) and the parameter code (five digit) and have a general idea of the time span you are interested in.

The command is this:

```
Sample <- getSampleData(siteNumber, ParameterCd, StartDate, EndDate,
interactive = TRUE)
```

A simple example of its application could look like this:

```
Sample<-getSampleData("06934500","00631","1970-10-01","2011-09-30")
```

What this command is doing is the following. It is obtaining from NWISweb services, all of the sample data on parameter code 00631, for site 06934500, for dates starting with October 1, 1970 and ending with September 30, 2011, and it is placing that information (and other derivative information about these samples) into a data frame called `Sample`. Note that the dates are always in yyyy-mm-dd format and all the entries in the call are in quotes. You can, if you like, make each of these arguments a variable and enter them into the call that way. It is just a matter of personal preference. It may make things simpler because you are likely going to want to use these entries more than once. In that style it would be this.

```
siteNumber<-"06934500"
ParameterCd<-"00631"
StartDate<-"1970-10-01"
EndDate<-"2011-09-30"
Sample<-getSampleData(siteNumber,ParameterCd,StartDate,EndDate)
```

The start and end dates can be any day of any year, but it may be handy to cut things off at the start and end of calendar years or water years.

Also, if you simply want to retrieve all the data, regardless of date, you can do this:

```
Sample<-getSampleData(siteNumber,ParameterCd,"","")
```

The pair of quotes with nothing inside them denotes, that you are not pre-specifying a start or end date to the data set you want.

When you execute this command you get some information back about values that the system had a problem with. You may want to check these out by going into NWISWeb and trying to determine what the issue is. Ultimately, you will have the ability to go into the `Sample` data frame and modify or delete samples. This kind of troubleshooting the data is described in section 5. If you find that there is a part of the record (say, before “1981-10-01”) that are problematic, you can always run the retrieval again with the start of end date modified so it doesn’t retrieve earlier or later data. You don’t need to delete the `Sample` data frame you have created, you can just run the command again and you will have a refreshed version of `Sample`.

To get an initial impression about how the data look, do the command

```
summary(Sample)
```

From this you will learn the date of the first and last sample, you can get an idea of the range of concentrations, and find out how common censoring is. A few explanations are in order. There are two variables here, one called `ConcLow` and the other `ConcHigh`. If the data are uncensored, then the two will be equal. If a value is censored (say at a reporting limit of <0.5) then `ConcLow` would be set to `NA` (which stands for Not Available) and `ConcHigh` would be at the reporting limit (0.5). The variable called `Uncen` is always 1.0 for an uncensored value and always 0.0 for a censored one. If you look at the mean

value of `Uncen` it will tell you the frequency of uncensored values in the data set. For example if the mean of `Uncen` is 0.95, then 5% of the values are censored. The variable `ConcAve` is there for convenience and is not used in any computation. For censored values it is just half the reporting limit. In more complex cases, where the data are the sum of multiple analyte concentrations, a censored value can have a lower bound that is greater than zero, in which case `ConcLow` is that lower bound and `ConcAve` is mid-way between the lower bound and the reporting limit (more information on this is provided in section 7).

You can see how many observations you have by giving the command

```
length(Sample$ConcHigh)
```

You can also make a very simple time series plot of the data with the command:

```
plot(Sample$DecYear,Sample$ConcAve)
```

It won't look particularly fancy, but it gives the basics. It is very useful for determining if there are any really odd values (which you might want to explore and verify). It can also tell you where, in time, the bulk of the data are. Are there big time gaps? Are there just one or two stray values in the early years, but the regular data collection doesn't begin till later? If there are big gaps, take note of them and you can come back later and blank that time out from the analysis using a function called `blankTime`. Looking at the data set, you will want to decide what the start and end of your real period of study should be. If you want to trim it down, this is the time to do so. For example, if we found that there was very limited data prior to water year 1980 and for 2011 and beyond, we might re-run our retrieval with this command:

```
Sample<-getSampleData("06934500","00631","1979-10-01","2010-09-30")
```

b) Water quality data from text file:

What if you want data that comes from a file and not from NWISWeb? The commands for doing this are:

`filePath<-` inside of quotes give the full path name of the file on your computer except for the final part that comes after the last slash

`fileName<-` inside of quotes the local file name (the part that comes after the last slash).

So, for example it might look like:

```
filePath<-"/Users/rhirsch/Desktop/waterqualitydata/"
fileName<-"MISS_HERM_NO3"
```

The call for the function is:

```
Sample<- getSampleDataFromFile(filePath, filename,  hasHeader = TRUE,
separator = ",", interactive = TRUE)
```

A typical call would look like this (assuming that we have already created the variables filePath and filename.

```
Sample<-getSampleDataFromFile(filePath,fileName)
```

The file has to conform to a strict structure. It should be a csv file. Each column needs a header in the first row, you can call them something like "date" "remark" and "value". It actually doesn't matter what you call them, but they must be in the right order. The first is a date, which should be in the form mm/dd/yyyy or yyyy-mm-dd. A common error is the use of a two-digit year (e.g. 04/25/12). Spreadsheet programs tend to revert to these two-digit years but clearly they are ambiguous. Make sure the input files are formatted with 4-digit years (like 04/25/2012). The second column is for remarks, and these should show "<" for all less-than values, and be blank otherwise. The third column is the concentration. In the case where there is a < in the remark column the concentration column should contain the reporting limit. It is assumed that all concentrations are in mg/L. There should never be a zero value in the concentration column. The program will look for that, and if it exists it will give a warning and discard it. Based on your knowledge you may recognize that it is really a "less than" value, in which case you should modify your file and enter a < code and change the concentration to some appropriately low reporting limit. [There are actually a number of other formats in which you can structure the input file. You can read about them in the on-line help for the function `getSampleDataFromFile`.]

B. Retrieval of streamflow data:

For a WRTDS application the program as presently designed requires that there be a continuous record of daily streamflow data that completely spans the period of record for the water quality data. In other words, the streamflow record being used must start on, or before, the first sample day, and end on or after the last sample day. It is important to make sure that you NOT use a record that extends many years beyond the range of the water quality data. A good rule of thumb is that you select a starting date that is the beginning of the first water year in which you have water quality data and an ending date that is the last day of the last water year in which you have data. But, it is perfectly acceptable to have starting and ending dates that are not the start or end of the water year.

Just as with the water quality sample data, there are two ways to obtain the streamflow data, either from USGS web services or from a data file.

a) Streamflow data from USGS web service

The source of the streamflow data from the web service is: http://waterservices.usgs.gov/ Take note. In its present implementation, this step can take a couple of minutes. We will be looking into ways to speed it up.

If you are doing a flowHistory application, the data set should typically be as long as possible. Other than that, the instructions are identical to those given here for the WRTDS application.

The call is similar to the one for the water quality sample data.

```
Daily <- getDVData(siteNumber, ParameterCd, StartDate, EndDate, interactive = TRUE)
```

A typical example might be:

```
Daily <- getDVData("06934500","00060","1979-10-01","2010-09-30")
```

The `ParameterCd` value is always going to be 00060 (which is discharge) but the way this function is designed it could be used to retrieve other parameters that are stored in a daily values format. The EGRET application doesn't make use of any other daily values, but this code can be used to retrieve them for uses not a part of EGRET.

Two things that the code is looking out for are these.

1) Are there any gaps in the record? It will tell you how many days there are between the first and last days of the record, and then it will tell you how many daily values there are. If the number of daily values is smaller than this time span it will tell you between what pair(s) of dates the gap is. The current implementation of the code simply doesn't allow for data gaps, so using this information you can change your `StartDate` and/or `EndDate` so you have a complete record, but it might mean that you need to go back to the water quality data retrieval and shorten it up correspondingly.

2) The other thing it will look for is days of zero flow. Because many of the computations in WRTDS and in flowHistory use log(discharge) we need to make sure that there are no zero flow days in the record. The function will report how many zero flow days there are. In order to proceed, it will modify the discharge record by adding a tiny constant to all of the streamflow data. This constant is set to 0.1% of the mean streamflow in the record. If there are many zero flow days, then it is not be a good idea to use EGRET. But EGRET should work fine if the number zero days is small (a few percent of the days). Make sure to take note of the constant that was added.

Once the data are retrieved you can explore the data set, first with the command

```
summary(Daily)
```

It will tell you about the time span of the data set and the range of the data values. Note that the discharge values you will see in the summary are in cubic meters per second. You can see how many daily discharge values you have by doing the command:

```
length(Daily$Q)
```

And you can make a very simple plot with the command:

```
plot(Daily$DecYear,Daily$Q,log="y",type="l")
```

Note, in type="l" that is a lower case letter L. It stands for "line". When type is not listed the plot function assumes you want points rather than a line.
Again, you should take a look and see if anything looks strange about the data and try to get that taken care of before you proceed.

b) Streamflow data from text file

If the streamflow data are not available from NWISWeb, then you can enter them from a csv file (although other delimiters are possible). It must consist of two columns. The first is the date expressed as mm/dd/yyyy or yyyy-mm-dd, the other column is daily discharge. A common error is the use of a two digit year (e.g. 04/25/12). Spreadsheet programs tend to revert to these two-digit years but clearly they are ambiguous. Make sure the input files are formatted with 4-digit years (like 04/25/2012). The first row should contain headings such as "date" and "Qdaily". [There are actually a variety of ways to structure the input file. You can read about them in the on-line help for the function `getDailyDataFromFile`.] The call to this function is:

```
getDailyDataFromFile(filePath, fileName, hasHeader = TRUE, separator = ",",
qUnit = 1, interactive = TRUE)
```

Make sure that the range of dates in the discharge data file are such that they start on or before the first water quality sample and end on or after the last water quality sample. The units of discharge can be either cubic feet per second or cubic meters per second. The default here is expressed by the argument `qUnit = 1`, which indicates that the data are in cubic feet per second. If the data were in cubic meters per second then we would set `qUnit = 2`. Assuming that the units are cubic feet per second the call is this:

```
Daily<-getDailyDataFromFile(filePath,fileName)
```

Where `filePath` and `fileName` follow the same conventions used above. If the units of discharge are cubic meters per second the call is:

```
Daily<-getDailyDataFromFile(filePath,filename,qUnit = 2)
```

The `qUnit` is just a code that specifies cubic meters per second (more below about unit codes). If the discharge data are in some other units, then it is up to the user to convert their data to cubic feet per second or cubic meters per second before they are used.

C. Entering metadata

In EGRET the metadata are important, not just to keep track of the origins of the data and key facts about it, but they also drive the labeling of all of the tables and figures and also help in labeling files for future processing. There is one program to use for entering the metadata, regardless of whether the data are from the USGS or not. In general, the call is this.

In a flowHistory application, `parameterCD` is always 00060. And, when a parameter abbreviation is called for the user can use something like "Q" or "flow" or whatever. Otherwise, these instructions work for both a WRTDS and flowHistory application.

```
INFO<-getMetaData(siteNumber,parameterCD)
```

Here are four specific examples that might arise:

```
INFO<-getMetaData(siteNumber="01594440", parameterCd="00665")
```

In this example the site is a USGS site and the parameter has a USGS parameter code, so the relevant information is provided over the Internet

```
INFO<-getMetaData()
```

In this case there is no on-line USGS data about the site or the parameter – the user will be prompted for this information by the program.

```
INFO<-getMetaData(parameterCd="00665")
```

In this example there is no USGS site information (user must supply it) but there is parameter information that will come from the USGS web site.

```
INFO<-getMetaData(siteNumber="01594440")
```

In this example it is a USGS site, but the parameter code is not known, in which case the user must supply information about the analyte.

The program provides a series of prompts that the user must respond to. A little explanation of these is in order. In some cases the metadata provided from the web is not particularly suitable for use in figure or table titles. For example some sites have names that are IN ALL CAPITAL LETTERS, which is very ugly and hard to read. The user can type in a version that uses upper and lower case letters. Sometimes the site

name is exceedingly long, and for purposes of graphics and table headings it might be appropriate to find a way of shortening it or using some abbreviations. Also, parameter names can often be very long and detailed and it is good to be able to shorten them to something that makes a clear title in a figure. Via these prompts, the user has control over these things, BUT the full metadata, as retrieved off of the Internet, is never lost. It is always carried along within the INFO data frame. If the user finds the site and parameter names perfectly ok to use in plot and table titles, then all that is needed is a carriage return and the original versions are used. All of this is spelled out in the prompts.

Something else that is in the prompts is a request for an abbreviation for a site and an abbreviation for the analyte. These are needed to help managing files (inputs, workspaces, and graphical and tabular output files). These abbreviations are always carried along and can be very useful in going back and looking at your work and also for structuring batch jobs to look at many sites and or many analytes. For example we might use suscon as the abbreviation for Susquehanna River at Conowingo, and potlf for Potomac at Little Falls. Similarly, we have abbreviations for analytes, say tp for total phosphorus or no3 for nitrate. So any kind of file that refers to the Potomac River at Little Falls and total phosphorus would have as part of its name: “potlf.tp”. These can be upper case or lower case (but case matters) and they can be short or long. The best advice is make them long enough to be clear to you and others you work with, but not too long.

The INFO data frame, can have a great number of variables in it (the user can even add additional ones), but the system really won’t work unless there is some kind of station name and parameter (analyte) name, a set of abbreviations for station and parameter, and also a drainage area for the site. If the site is a USGS site the drainage area comes in automatically. If there is no on-line information the user must supply it. If it is truly unknown, then the user can enter a zero value, but certain functions of the programs will not work (those related to runoff in mm/day or yields (flux per unit area per unit time)).

D. Review of data retrieval functions:

The `dataRetrieval` package contains quite a number of functions, but there are really only 5 of them that the user would use directly. They are:

Name of function	Source of data	For what application
`getDailyDataFromFile`	User supplied file	WRTDS and flowHistory
`getDVData`	USGS Web Services	WRTDS and flowHistory
`getSampleDataFromFile`	User supplied file	WRTDS
`getSampleData`	USGS Web Services	WRTDS
`getMetaData`	USGS Web Services and/or user responses to program	WRTDS and flowHistory

E. Final steps of data preparation for a WRTDS application:

There are three additional things that should be done once these retrieval steps are done.

First, there are times when sample values are replicated in the NWISWeb database. The duplicates should be removed before proceeding. The step is simple.

```
Sample <- removeDuplicates()
```

What this function does is look for samples taken on the same date that have the same concentration values. When it finds such a duplicate it removes it from the data set. It does not remove multiple observations on a given day if the concentration values are different.

The next step is one that brings the discharge data from the daily values record into the sample data set for the days on which there was a sample. Note, that WRTDS does not use the discharge at the time of sampling in its analysis. The only discharge used is the daily mean discharge from the daily values file. The reason for this is that WRTDS uses a regression model where daily mean discharge is an explanatory variable. Thus, to estimate that regression it needs to use daily mean discharge rather than the instantaneous value. (For those familiar with LOADEST or Fluxmaster, they use the same approach as is used here). This step is combined with another step that provides some summary overview of the information. It is good to look that over and see if anything looks out of line. The command for this two-part process is:

```
Sample <-mergeReport()
```

Note that you will not be able to do the subsequent analyses in WRTDS if you have not run the function `mergeReport`. An additional step that can be very helpful is to produce a four-panel graphic that gives a quick visual impression of the data. The command is.

```
multiPlotDataOverview()
```

There are things that you can do to modify the discharge units that are shown and to save the output for future use. These are described in the chapter on WRTDS graphics. But this simple command will give you a quick overview. In particular, it will show you if the shape of the flow versus concentration relationship is one that is likely to be captured well by a quadratic equation in the logs (like LOADEST uses). It will also help in the identification of outliers that should be checked. It will show the general pattern of seasonality in concentration, and will show how the distribution of daily values of discharge compares to the distribution of discharge on the sample days. We would like to see the samples cover much of the full range and be somewhat weighted towards the higher discharges.

Finally, it is a good idea to save the data you have gathered together into the three data frames. This is best done with the function `saveResults`. The call is:

```
saveResults(savePath, localINFO = INFO)
```

All you really need to do is this. The simplest way to do this might look like this:

```
savePath<-"/Users/rhirsch/Desktop/"

saveResults(savePath)
```

Let us say that when we ran `getMetaData` we provided the following abbreviations. For the station we gave it the abbreviation "Chop" (for Choptank River) and "TN" (for total nitrogen). Given that, what will happen is that it will create a file on your computer that is the entire workspace that has been created so far (which is basically just the three data frames (Sample, Daily, and INFO). In this example the file will have the name: /Users/rhirsch/Desktop/Chop.TN.RData

Once you have created that file, any time you want to restart R and load EGRET (the command is: `library(EGRET)`) you can give the command to load this workspace file and all of the data will be restored and you are ready to proceed.

Obviously, you will need to specify the first argument in a way that makes sense for your computer (and these will look different for Windows versus Macintosh computers). The key thing is that you do need to include the final "/" (or on Windows the final "\"). As you proceed along with your analysis, any time you would like you can resave the workspace by giving this same `saveResults` command again. It will overwrite the original file you saved with the new one. As you will see below, as you progress these three data frames will be augmented with a number of results that you will want to save

and some other matrices or data frames will be created. So, it is a good idea to do this repeatedly as you move forward with your work. Note that when you are working with multiple data sets you will be able to tell which is which because the file name contains the site and constituent abbreviations.

Note that if you use the approach shown above, `savePath` is now an object in your workspace and it will be saved, so when you want to use it again it will be there and you don't have to specify it again and again.

F. What to do when you get an error message.

A few comments on dealing with errors and getting help are in order before we move on. In general, when a function doesn't work and an error message is returned to you, it is unlikely that the error message will be particularly useful to you. However, in most cases, error messages come about because you have left out a key step. As a result of leaving out that key step some data element that the function needs doesn't exist. If you get an error message the first thing you should do is to look over the example workflow (appendix F) and see if you can identify a step you left out or did wrong. Three common errors in a WRTDS application are leaving out `getMetaData`, `mergeReport`, or `setupYears`. Any one of those will cause errors down the line. For a flowHistory application common errors would be leaving out `getMetaData`, `setPA`, or `makeAnnualSeries`. Another thing you should do when you find an error message that you don't understand is print summaries of the relevant data frames. For example `summary(Sample)`. Looking at the information here may reveal something strange. For example look at the dates. Are they in the right century? Section F (immediately below) gives a listing of all of the columns that should be present in the three basic data frames, you might look at these and compare them to the data frame summary lists and see if you see something missing or clearly in error.

When all else fails and you need to seek help from one of the code authors (or a colleague who has experience with R and EGRET) is to send them your workspace. A simple e-mail saying, I can't get such and such to work for me, can you tell me what's wrong here? Then include the relevant lines from your console (just copy and past into your email) and then attach the workspace file which you should have saved using the `saveResults` command. The recipient of the e-mail can quickly load the workspace and can likely find the problem very quickly and they may be able to fix it for you, re-save the workspace and send it back with an explanation of what the problem was. You can then load that new workspace and you should be off and running from there.

G. Overview of what's going on inside the applications

It is useful to have some understanding of what's going on and how the data are organized. In a WRTDS application, at this point, one has created 3 data frames that contain all of the information that will be used in the analysis. They are:

Data frame	What it contains	How it will be augmented later
`Daily`	The daily discharge values, and the 7-day and 30-day running averages of discharge, up-to and including that day, and a variety of indicators of time (such as Julian day or decimal year)	The WRTDS analysis will add to this a set of estimates for concentration, flux, flow-normalized concentration, and flow-normalized flux, and also the estimate of the log of concentration (without the bias correction) and the standard error for that day.
`Sample`	The dates of sampling, the daily mean discharge on that day, the observed concentration, plus a variety of indicators of time (such as Julian day or decimal year)	The WRTDS analysis will add to this the estimate of concentration on that day, the estimate of the log of concentration (without the bias correction) and the standard error for the day
`INFO`	The meta data for the site and analyte	It will be augmented with information about how the data were analyzed

Here is how units of measurement are handled. EGRET uses entirely SI units to store the data, but for purposes of output it can report results in a wide range of units (e.g. kg/day, tons/day, 10^6 kg/year…..). The units used in storing the data are these:

Variable	Units
Concentration	mg/L
Discharge	m^3/sec
Flux	kg/day
Drainage area	km^2

Here is some more detailed information about the data frames. Recognize that information about any column of the data frame is available to the user. For example, the Julian date of a sample in the `Sample` data from would be known as `Sample$Julian`.

Daily data frame. This data frame will contain a row for every day for which the WRTDS model is being used. The variables that will be added later in the process are shown in red:

Daily data frame

Name	Definition	Units or type
Date	The date	yyyy-mm-dd
Q	The discharge on that date	m^3/sec
Julian	The date expressed as days starting with Jan 1, 1850	days
Month	Month of the year, from 1 to 12	months
Day	Day of the year, from 1 to 366	days
DecYear	Year expressed as a decimal	years
MonthSeq	Month sequence: an index starting with 1 at Jan, 1850	months
LogQ	ln(Q)	numeric
i	index of days from the start of the data frame	days
Q7	Mean discharge for 7 days, up to day i[1]	m^3/sec
Q30	Mean discharge for 30 days, up to day i[2]	m^3/sec
yHat	The WRTDS estimate of the log of concentration	numeric
SE	The WRTDS estimate of the standard error of YHat	numeric
ConcDay	The WRTDS estimate of concentration	mg/L
FluxDay	The WRTDS estimate of flux	kg/day
FNConc	Flow Normalized estimate of concentration	mg/L
FNFlux	Flow Normalized estimate of flux	kg/day

Footnote 1: Q7 is the average of the 7 days ending with the present day (so if the day is September 21, 2010 this would be the average of September 15-21, 2010). At the start of the record before there is 7 days of history, the values will be NAs. This variable is not used in the current implementation of WRTDS, but it may be useful for diagnostic purposes. It is used in flowHistory.

Footnote 2: Q30 is the average of the 30 days ending with the present day (so if the day is September 29, 2010 this would be the average of August 31-September 29, 2010). At the start of the record before there is 30 days of history, the values will be NAs. This variable is not used in the current implementation of WRTDS, but it may be useful for diagnostic purposes. It is used in flowHistory.

Sample data frame. This data frame will contain a row for every day for which there is a water quality sample. The variables that will be added later in the process are shown in red:

Sample data frame

name	definition	units or type
Date	The date of the sample	yyyy-mm-dd
ConcLow	Lower bound for an observed concentration	mg/L
ConcHigh	Upper bound for an observed concentration	mg/L
Uncen	=1 if sample is uncensored, =0 if censored	integer
ConcAve	Average of ConcLow & ConcHigh	mg/L
Julian	The date expressed as days starting with Jan 1, 1850	days
Month	Month of the year, from 1 to 12	months
Day	Day of the year	days
DecYear	Year expressed as a decimal	years
MonthSeq	Month sequence: an index starting with 1 at Jan, 1850	months
SinDY	sin(2*π*DecYear)	numeric
CosDY	cos(2*π*DecYear)	numeric
Q	Daily mean discharge on the day of observation	m^3/sec
LogQ	ln(Q)	numeric
yHat	jack-knife estimate of the log of concentration	numeric
SE	jack-knife estimate of the standard error of yHat	numeric
ConcHat	jack-knife unbiased estimate of concentration	mg/L

INFO data frame. Depending on how it is retrieved, there are a wide range of possible elements in this data frame. What is shown here are the set of elements of INFO that are critical. The variables that will be added later in the process are shown in red:

INFO data frame.

Name	Definition	units or type
site.no	8-digit USGS site number	character
shortName	Name of site, suitable for use in headings	character
dec.lat.va	decimal latitude	numeric
dec.long.va	decimal longitude	numeric
drainSqKm	drainage area	km^2
staAbbrev	abbreviation for station name	character
param.nm	full name of the constituent (parameter code def.)	character
paramShortName	Name of constituent, suitable for use in headings	character
constitAbbrev	abbreviation for consituent name	character
paStart	Starting month of period of analysis[1]	integer (1-12)
paLong	Length of period of analysis in months[1]	integer (1-12)
bottomLogQ	lowest discharge in prediction surfaces[2]	numeric
stepLogQ	step size in discharge in prediction surfaces[2]	numeric
nVectorLogQ	number of steps in discharge, prediction surfaces[2]	numeric
bottomYear	starting year in prediction surfaces[2]	numeric
stepYear	step size in years in prediction surfaces[2]	numeric
nVectorYear	number of steps in years in prediction surfaces[2]	numeric

windowY	half-window width in the time dimension	years
windowQ	half-window width in the log discharge dimension	numeric
windowS	half-window width in the seasonal dimension	years
minNumObs	minimum number of observations for regression	integer
minNumUncen	minimum number of uncensored observations	integer

Footnote 1: paStart and paLong are more fully discussed below, but they are the two user-defined variables that determine the period of analysis (set of months and starting month of the portion of the record for which estimates are to be made).

Footnote 2: These six variables are used to define the grid over a space of log discharge and time in years, for which the yHat, SE, and ConcHat surfaces are evaluated. These are the basis for all of the interpolation of results.

2. Analysis and Graphics for flowHistory component

If your interest is only in the WRTDS applications, you can skip this section.

At this point it is assumed that you have loaded the daily discharge record and created the `Daily` data frame, and also entered the required meta-data into the `INFO` data frame.

The first choice you need to make is what "period of analysis" to use. What is the period of analysis? If we want to examine our data set as a time series of water years, then the period of analysis is the water year. If we want to examine the data set as calendar years then the period of analysis should be the calendar year. We might want to examine the winter season, which we might want to define as December, January and February, then those 3 months become the period of analysis. We might even want to examine September only then September becomes the period of analysis. The only constraints on the definition of a period of analysis are these: It must be defined in terms of whole months (it can't start or end mid-month). It must be a set of contiguous months (like March-April-May or November-December-January-February). And it must have a length that is no less than 1 month and no more than 12 months. It can be uniquely defined by two arguments. They are called, `paLong` and `paStart`, both of which must be integers no less than 1 and no greater than 12.

The argument `paLong` is the length of the period of analysis in months, and `paStart` is the first month of the period of analysis. The month numbers are the calendar months (January is 1, December is 12). So: If we are using calendar years, `paLong =12`, `paStart =1`. If water years, `paLong =12`, `paStart =10`. If we are leaving out the winter we might have `paLong = 9`, `paStart = 3`, (making the period of analysis March-November) or if we just want the months of March, April, May and June we would have `paLong =4`, `paStart = 3`. Or if we wanted the period of analysis to be September only, then `paLong =1` and `paStart = 9`.

There is a convention for numbering the "years". The year is known by the year in which it ends. That's the same approach we use with water years. The water year starting October 1, 1990 is called water year 1991. One other note, for time series graphs of annual averages the x-axis position of a value is always plotted at the mean date for the period, so for a calendar year, the data point shows up mid-way through the calendar year. For a water year it shows up mid-way through the water year (at ¼ the way through the calendar year). The x-axis on time series graphs of the data are always labeled such that the axis labels are for calendar years, so 1981.0 always means January 1, 1981, regardless of whether we are using calendar years or water years or some other period of analysis.

A. setPA

The period of analysis is selected by using the function `setPA`. The call to `setPA` is as follows. For the default case, which is water year it is just:

```
INFO<-setPA()
```

For a case where `paStart = 12` and `paLong=3` (a period running December through February) it is:

```
INFO<- setPA(paStart=12, paLong=3)
```

There are more arguments to this function, but for starting out, this is all that is needed. What is happening with this command is simply that the `paStart` and `paLong` values are being added to the INFO data frame making them part of the metadata. They can be changed at any point going forward, but they define how the user is evaluating the data.

B. makeAnnualSeries

The next step is to create the annual series of flow statistics. These will be stored in a matrix (called `annualSeries`) that contains a wide range of statistics for the data set. It contains, for every year in the data set, for the period of analysis the user has selected, a set of statistics about that year. These are, in order:

istat	statistic name
1	1-day minimum flow
2	7-day minimum flow
3	30-day minimum flow
4	median flow
5	mean flow
6	30-day maximum flow
7	7-day maximum flow
8	1-day maximum flow

`istat` is just an index of the statistics (something you may want to print out or commit to memory). A couple of details to know about these statistics: When the period of analysis is set to water year (the default) the 3 low-flow statistics are based on the climatic year (which runs from April through March) so that it minimizes the chance that a low flow period will be split across two different years. Another detail is that the 7-day and 30-day statistics are actually lagged time periods, so for example the 30-day maximum for water year 1990 would be the highest 30 day consecutive period ending on a day between October 1, 1989 and September 30, 1990. This means that the 30-day maximum for water year 1990 could possibly be made up mostly of days that were in water year 1989.

In addition to containing these annual results, the `annualSeries` matrix contains a smoothed time series version of these 8 flow statistics. The actual mathematics involved in computing this smoothed version is given in appendix E, but for here it will suffice to say that it is a lowess smooth, with a half-window width of 30 years (this can be adjusted by the user – see full definition of the arguments for `setPA`). The matrix also contains the decimal year value that corresponds to the middle of the data set evaluated for each year for the period of analysis (this establishes the time value at which the actual and smoothed data values are plotted on graphs). The algorithms used in creating these results depend on having a complete, or near complete, data set for any given period of analysis in any given year. The data set must be at least 90% complete – thus a result for a given water year could be computed if a single month of that year were missing, but not if two of them were missing. The function for creating this matrix is `makeAnnualSeries`. The command is just:

```
annualSeries<-makeAnnualSeries()
```

It can take a few seconds to run, be patient.

To understand what it contains, let's say that the data set was 83 years long, then `annualSeries` would be a matrix of dimension (3,8,83). The last dimension is simply an index of years. For the first dimension: 1 for the mean `decYear` value, 2 is the actual flow statistic for that year, and 3 is the smoothed value of the flow statistic for that year. The second dimension is the index of the flow statistics `istat`=1 through 8. Any values that can't be computed are stored as **`NA`**.

C. plotFlowSingle

The simplest way to look at these time series is with the function `plotFlowSingle`.

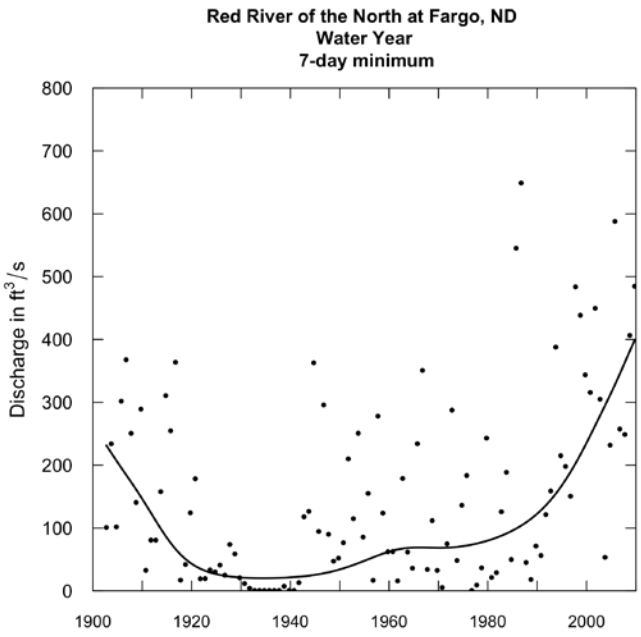
In the simplest case (let's say we want the mean flow for each year for the period of analysis) the command would look like this:

```
plotFlowSingle(istat=5)
```

If we wanted to switch to the 7-day minimum flow it would be

```
plotFlowSingle(istat=2)
```

Here is a reduced-size version of what the output looks like.



There are a number of optional arguments you can use to modify the appearance of your result. They are described in this table:

Argument	meaning and choices	default
`istat`	Which flow statistic to plot: 1-8. Must be specified.	No default
`yearStart`	What is the `decYear` value where you want the graph to start?	default causes it to start with the first year in the data set*
`yearEnd`	What is the `decYear` value where you want the graph to end?	default causes it to end with the last year in the data set.*
`qMax`	User specified upper limit on y axis (can be used when we want several graphs to all share the same scale) will be specified in the discharge units that the user selects (see `qUnit` or runoff below)	Default is that the graph will be self-scaled to have its maximum y value at least 10% higher than the highest value in the data set.
`printTitle`	can be TRUE or FALSE, you may want FALSE if it is going to be a figure with a caption or if it is a part of a multipanel plot	default is TRUE
`tinyPlot`	Can be TRUE or FALSE, the TRUE option assures that there will be a small number of tick marks, consistent with printing in a small space	default is FALSE
`runoff`	Can be TRUE or FALSE. If true then discharge values are reported as runoff in mm/day. This can be very useful in multi-site analyses	default is FALSE
`qUnit`	An index indicating what discharge units to use. Options run from 1 to 13 (see table D-1 for these values). The choice should be based on the units that are customary for the audience but also, the choice should be made so that the discharge values don't have too many digits to the right or left of the decimal point.	default is `qUnit=1` (which is cubic feet per second)

printStaName	Can be TRUE or FALSE, if TRUE the name of the streamgage is stated in the plot title	default = TRUE**
printPA	Can be TRUE or FALSE, if TRUE the period of analysis is stated in the plot title	default = TRUE**
printIstat	Can be TRUE or FALSE, if TRUE the name of the statistic (e.g. 7-day minimum flow) is stated in the plot title	default=TRUE**

Footnote *: Setting yearStart and yearEnd will determine where the graphs start and end, but they don't determine where the smoothing analysis starts and ends. There are situations, typically where many sites are be analyzed together, where you may want to run the smoothing on a consistent period of record across all sites. Doing this requires modifying the Daily data frame before running makeAnnualSeries – this is described in section 6.

Footnote **. If the printTitle argument is set to FALSE, then it really makes no difference what you do with printSta, printPA, or printIstat. They can all be left as their default values and thus there is no need to include them in the call for the function.

D. printSeries

Another way to consider the same set of results that is shown graphically by plotFlowSingle is to produce a table of these values using printSeries. The command, in the simplest case would be:

```
printSeries(istat=5)
```

The output looks like this:

```
Red River of the North at Fargo, ND
 Water Year
    mean daily
    Cubic Feet per Second
   year   annual   smoothed
           value    value

   1902    450.0      675
   1903    363.6      666
   1904    753.8      655
   1905    702.6      642
   1906   1064.5      628
.........etc. through    2011
```

There are a few options that can be used and they include using two of the optional arguments that were listed above for `plotFlowSingle`. These are `qUnit`, and runoff. They work the same for this function. The table will be printed to the console. You can copy it into a word processing document or Excel file. Note that it will generally look better if you format it in your word processor as Courier font (equal spacing).

E. tableFlowChange

Another way to look at the results is to consider how much the smoothed values change between various pairs of years. These changes can be represented in four different ways.

- As a change between the first and last year of the pair, expressed in the flow units selected.
- As a change between the first and last year of the pair, expressed as a percentage of the value in the first year
- As a slope between the first and last year of the pair, expressed in terms of the flow units per year.
- As a slope between the first and last year of the pair, expressed as a percentage change per year (a percentage based on the value in the first year).

The simplest version of this call (for mean discharge) is:

```
tableFlowChange(istat=5)
```

We can enhance it by using the arguments `qUnit` and runoff as described above for `plotFlowSingle`.

However, there is one other argument that can be very useful. It is called `yearPoints`. In the default case, the set of years that are compared are at 5-year intervals along the whole data set. If the data set was quite long this can be a daunting number of comparisons. For example, in an 80-year record starting in 1930 there would be 136 such pairs (e.g. 1930-1935, 1930-1940, 1930-1945…). Let's say, we want to look at changes between 1930 and 1970, 1970 and 2010 and the full period 1930 to 2010. We can set up a vector, let's call it `yearPoints`, consisting of 1930, 1970, and 2010. The commands would be:

```
yearPoints<-c(1930,1970,2010)

tableFlowChange(istat=5,yearPoints=yearPoints)
```

Another way to do the same thing would be:

```
tableFlowChange(istat=5,yearPoints=c(1930,1970,2010))
```

The output would look like this:

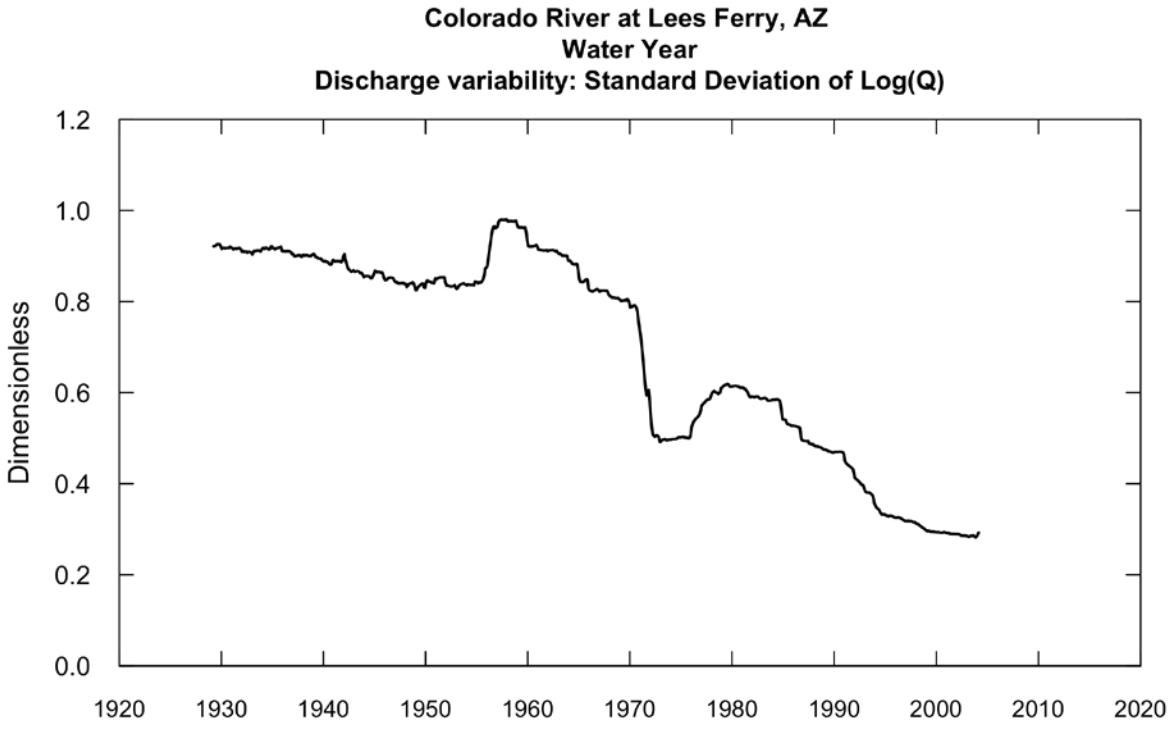
```
   Potomac River at Little Falls, DC
   Season Consisting of Mar Apr May Jun Jul Aug Sep Oct

              Streamflow Trends
   time span          change        slope       change        slope
                         cfs      cfs   /yr          %        %/yr
 1930  to  1970         1774           44           21         0.52
 1930  to  2010         2577           32           30         0.38
 1970  to  2010          803           20          7.8         0.19
```

F. plotSDLogQ

This function produces a graphic of the running standard deviation of the log of daily discharge over time. The idea is to get some idea of how variability of daily discharge is changing over time. By using the standard deviation of the log discharge the statistic becomes dimensionless. It also means that it is a way of looking at variability quite aside from average values, so, in the case of a system where discharge might be increasing over a period of years, this provides a way of looking at the variability relative to that changing mean value. It is much like a coefficient of variation, but it has sample properties that make it a smoother measure of variability. There are often comments about how things like urbanization or enhanced greenhouse gases in the atmosphere are bringing about an increase in variability, this is one way to explore that idea. In the simplest case the call is:

`plotSDLogQ()` and it will produce a figure like this:



This example is for the Colorado River, a short distance below Glen Canyon Dam, and it shows clearly the marked decrease in variability in streamflow since the time of the Dam's construction, and also some aspects of changing operations and extreme prolonged wet periods and dry periods.

The full call to the function is:

```
plotSDLogQ(yearStart = NA, yearEnd = NA, window = 15, localDaily = Daily,
 localINFO = INFO, sdMax = NA, printTitle = TRUE, tinyPlot = FALSE,
printStaName = TRUE, printPA = TRUE)
```

Most of these arguments are similar to those we have seen before. The different ones include these: 1) `window`, which is the full width of the window over which the standard deviation is computed (in years). Note that the standard deviation is plotted at the middle of the window. 2) `sdMax` is a maximum value for the vertical axis. This could be used if one is comparing one site to another and common scales are desired. 3) `printPA` is a logical variable and if TRUE it will print the period of analysis. These graphs always make their computations based on a specified period of analysis (determined by `INFO<-setPA(paStart, paLong)`), so, for example if we were interested in a site where winter conditions are of interest and very influenced by changes in frequency of thawed conditions we might want to restrict the analysis to January and February by running `INFO<-setPA(paStart=1,paLong=2)` before we run `plotSDLogQ`.

G. plotQTimeDaily

This graphic is simply a time series plot of discharge. But, it is most suited for showing events above some discharge threshold. In the simplest case, it can plot the entire record, but given the line weight and use of an arithmetic scale it will primarily provide a visual focus on the higher values.

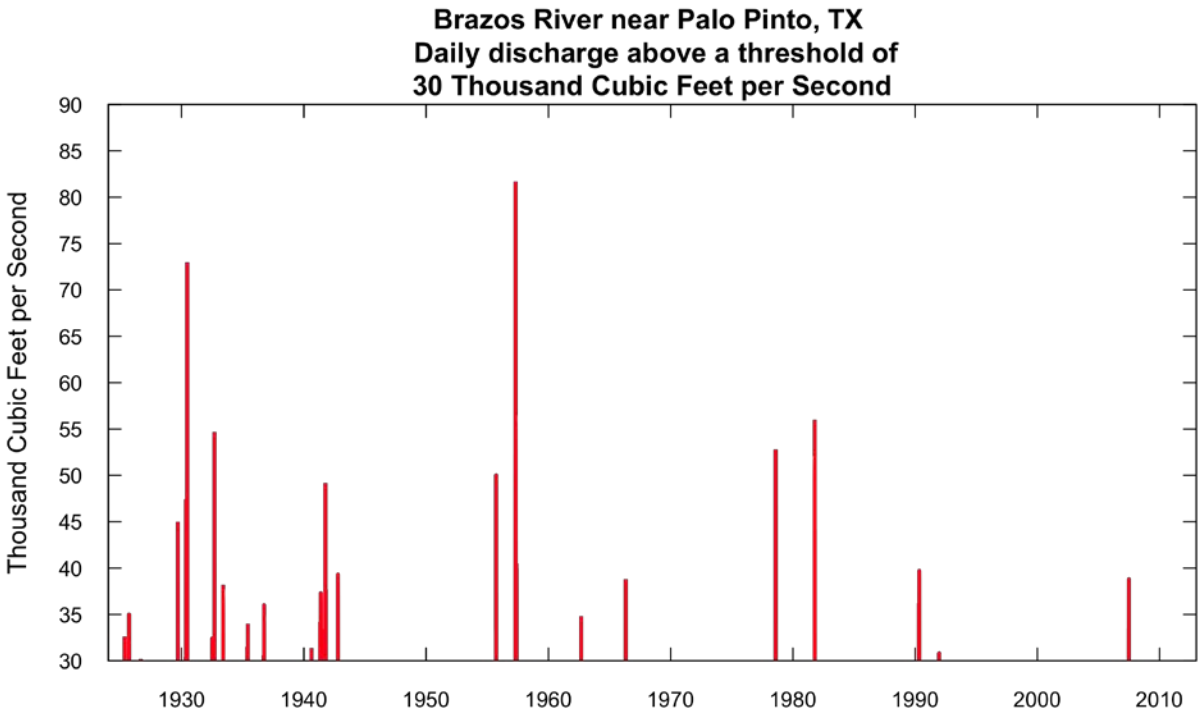
plotQTimeDaily(startYear,endYear)

The full call is:

```
plotQTimeDaily(startYear, endYear, localDaily = Daily, localINFO = INFO,
qLower = NA, qUnit = 1, tinyPlot = FALSE, printTitle = TRUE)
```

The argument, `startYear`, is the first year of record to be plotted, `endYear` specifies the end of the last year (so if we have data for all of calendar year 2011 we would set `endYear`=2012). The `qLower` is a lower bound on the flows that will be plotted, so that the graph can put its emphasis on the magnitude and frequency of the high flow events. `qLower` is expressed in the discharge units that are specified by `qUnit`. An example for a site that has had considerable regulation and changes in water use and land use over time is the following:

```
plotQTimeDaily(1924,2013,qUnit=3,qLower=30)
```



The plot shows us that exceedences of 30,000 have become much less frequent in recent decades and it appears that the magnitudes of these exceedences might also be decreasing over time.

H. plotFourStats, plotFour and plot15

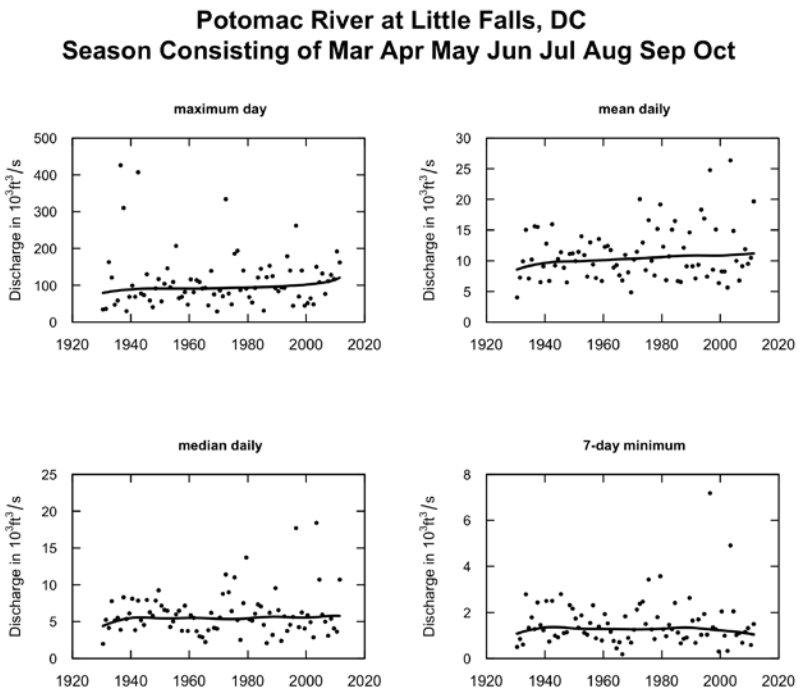
These three functions are all designed to plot several graphs from the other functions all in a single figure.

The first of these is a combined graphic of four plots. They are plots for the 7-day minimum, median, mean, and 1-day maximum (the other four flow statistics are not shown, but it is commonly the case that these look rather similar to these statistics).

In the simplest case, the call is

```
plotFourStats()
```

There are a variety of optional arguments available to modify the plot. These are: `yearStart`, `yearEnd`, `printTitle`, `runoff` , and `qUnit`. These follow all of the same definitions and options as described in the table above.

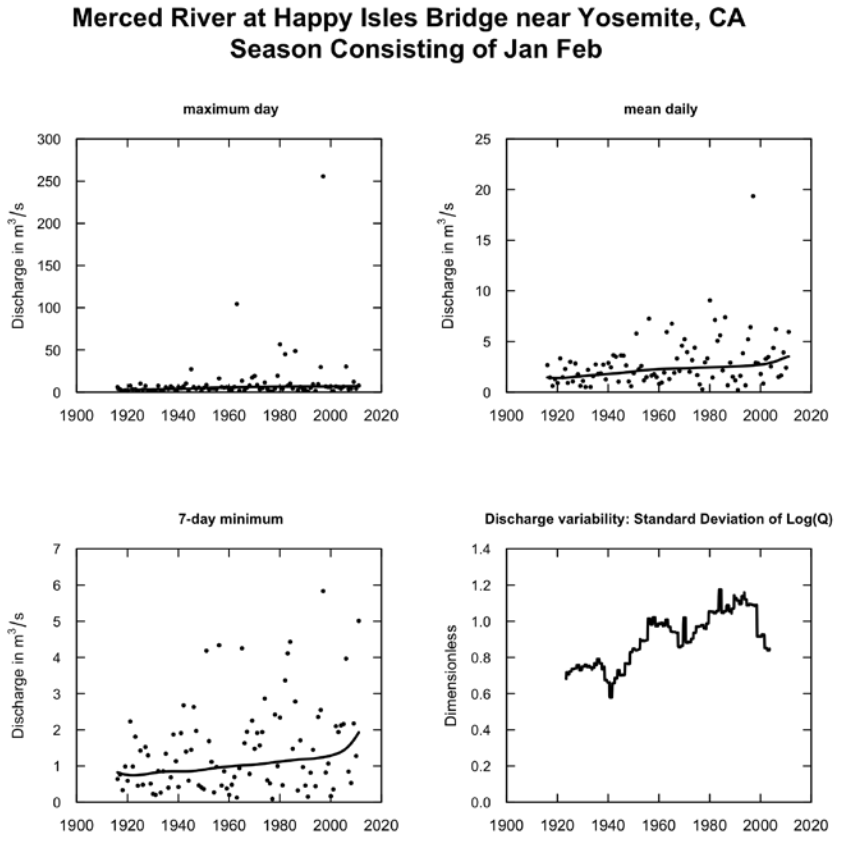


The second one is identical to this but deletes the median daily graph and puts in the running standard deviation. It is called plotFour. The call in the simplest case is just

```
plotFour()
```

Here is an interesting example from the Merced River high in the Sierra Nevada range in California, looking at the period of analysis January and February only. The full set of commands to create this after the initial steps of aquiring the data and meta data are:

```
INFO<-setPA(1,2)
annualSeries<-makeAnnualSeries()
plotFour(qUnit=2)
```



It shows a general increase in variability over time as well as an increase in average values of high flows, mean flows and low flows.

Finally, there is a function for producing an array of 15 graphs on a single page. They are 3 statistics: 7-day low, mean, and maximum for each of 5 periods of analysis (full water year, and each of 4 seasons). It is not intended for use in a power point presentation but could work for exploratory purposes or in a report. It has a caption at the bottom that could be modified for use in a report. Unlike the other graphics in the

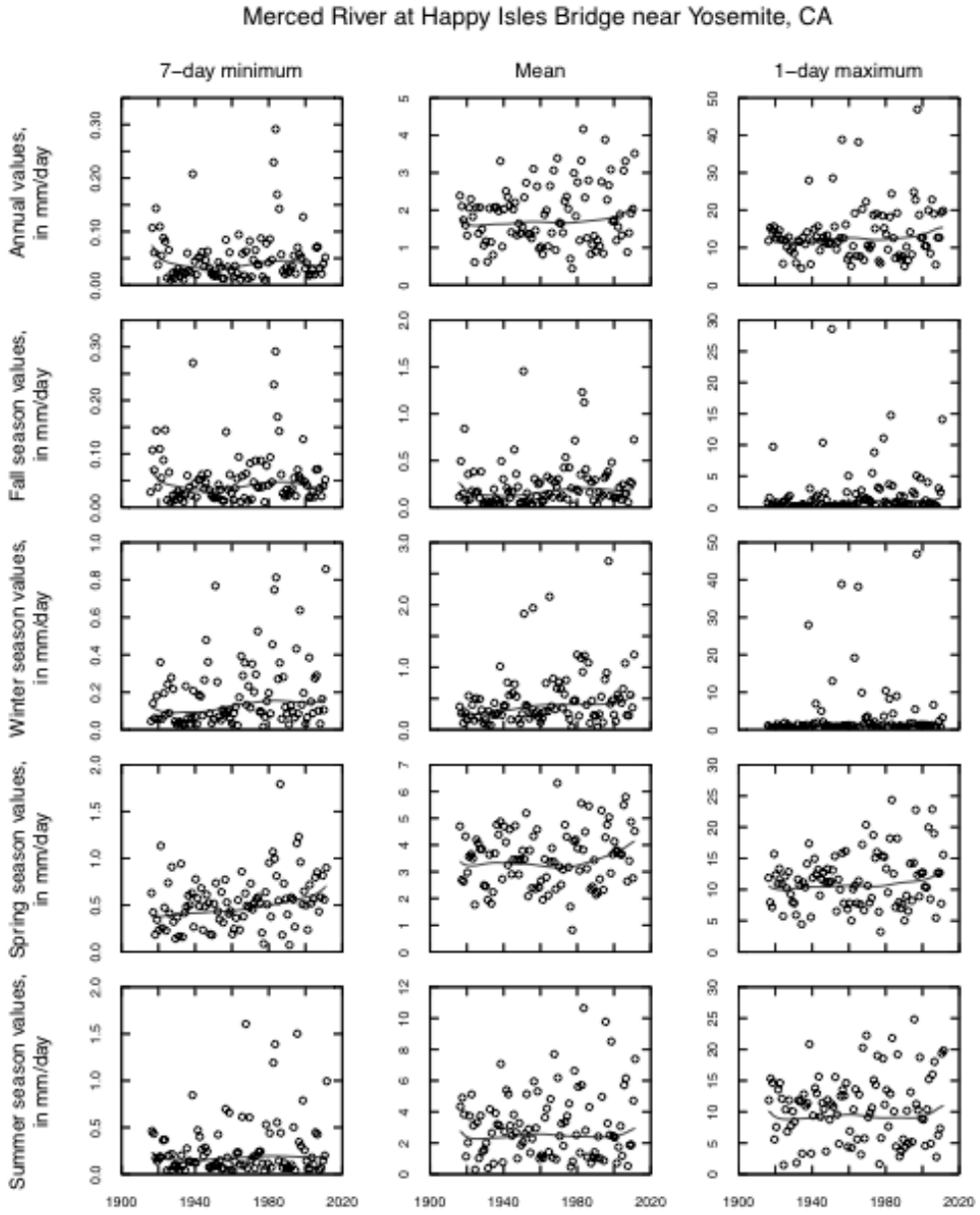
package, this one doesn't print to the screen but actually produces an output file (because it is very specific about the geometry of the collection of graphs). The call is:

```
plot15(savePath, yearStart, yearEnd, localDaily = Daily, localINFO = INFO)
```

Here is an example for the Merced River based on the command

```
plot15(savePath,1918,2012)
```

The postscript file is created and put in the directory specified by `savePath`. The file name is "plot15.Merced.ps", where "Merced" was the value of `INFO$staAbbrev`.



Streamflow statistics (circles) in units of millimeters per day, annual values and seasonal values
Fall (Sept., Oct., and Nov.), Winter (Dec., Jan., and Feb.), Spring (Mar., Apr., and May), and Summer (June, July, and Aug.)
and locally weighted scatterplot smooth (solid curve) for Merced River at Happy Isles Bridge near Yosemite, CA for 1918 – 2012.

3. Running the WRTDS system

This section will be organized into the following subsections.

a. Exploring the data, prior to running the model

b. Estimating the WRTDS model and looking at summary results

c. Exploring the fit of the WRTDS model (including flux bias)

d. Viewing the model behavior through contour and other types of plots.

a. Exploring the data, prior to running the model

As a general rule, it is very helpful to take a look at the data in a graphical way to understand its behavior and to identify things that might be errors in the data set or learn about the temporal distribution of the data (identify gaps) prior to running the model. It is always best to clear up these issues before moving forward.

The following is a table listing all of these functions, their purpose, and the arguments that you may want to set to run them. All of them can be run with a simple command such as

```
plotConcTime()
```

This will get the plot with all its default settings. But, if you want to customize it in some way, this table names the arguments that you may want to use to achieve the customization you want. Many of these arguments are the same across multiple functions and they are all defined in another table that follows this one.

Function name	Purpose	Arguments that you might want to specify, shown here set at their default value
`boxConcMonth`	A graphic of 12 box plots of the concentration values organized by month	None
`boxQTwice`	A graphic with two side-by-side box plots, one of the discharge values in the Sample data set, the other all discharge values in Daily (on a log scale)	`qUnit = 2` `printTitle = TRUE`

`plotConcTime`	A single plot showing concentration versus time. Concentration is on a linear scale. But, the function can be tailored to plot only a specified subset of the data specified by discharge or season.	`qUnit = 2`
		`qLower = NA`
		`qUpper = NA`
		`paLong = 12`
		`paStart = 10`
		`concMax = NA`
		`printTitle = TRUE`

`plotLogConcTime`	A single plot showing concentration versus time. Concentration is on a log scale. But, the function can be tailored to plot only a specified subset of the data specified by discharge or season.	`qUnit = 2`
		`qLower = NA`
		`qUpper = NA`
		`paLong = 12`
		`paStart = 10`
		`concMax = NA`
		`concMin = NA`
		`printTitle = TRUE`

`plotConcQ`	A single plot showing concentration versus discharge. Concentration is on a linear scale and discharge is on a log scale.	`qUnit = 2`
		`concMax = NA`
		`printTitle = TRUE`

`plotLogConcQ`	A single plot showing concentration versus discharge. Concentration is on a log scale and discharge is on a log scale.	`qUnit = 2`
		`concMax = NA`
		`concMin = NA`
		`printTitle = TRUE`

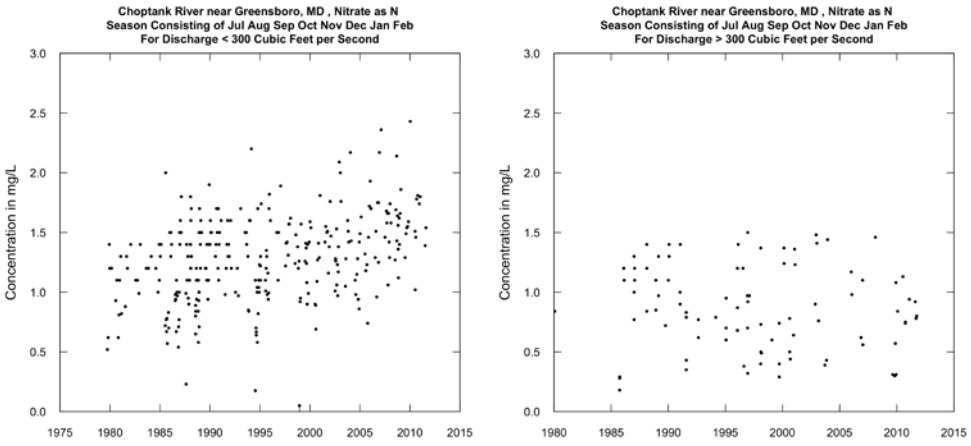
`plotLogFluxQ`	A single plot showing flux versus discharge. Flux and discharge are both on log scales.	`qUnit = 2`
		`fluxUnit = 3`
		`fluxMax = NA`
		`printTitle = TRUE`

`multiPlotDataOverview`	Produces a graphic containing four small plots. They are the same as: `plotLogConcQ`, `boxConcMonth`, `plotLogConcTime`, and `boxQTwice`	`qUnit = 2`

Arguments used in these functions:

Argument	Definition
qUnit	Determines what units will be used for discharge, use the command printqUnitCheatSheet() or Appendix D for a listing of the codes
printTitle	If TRUE the plot has a title. If FALSE no title (useful for publications where there will be a caption).
qLower	The lower bound on the discharge on the day of sampling that will be used in forming a subset of the sample data set that will be displayed in the graph. It is expressed in the units specified in qUnit. If qLower = NA, then the lower bound is set to zero.
qUpper	The upper bound on the discharge on the day of sampling that will be used in forming a subset of the sample data set that will be displayed in the graph. It is expressed in the units specified in qUnit. If qUpper = NA, then the upper bound is set to infinity.
paLong	The length of the time period that will be used in forming a subset of the sample data set that will be displayed in the graph. It is expressed in months.
paStart	The starting month for the time period that will be used in forming a subset of the sample data set that will be displayed in the graph. It is expressed in months (calendar months).
concMax	The upper limit on the vertical axis of graphs showing concentration values. In mg/L. If concMax = NA then the upper limit on the graph is set just above the maximum value. The reason this might be used is for making several similar graphs that show a comparison among data sets. Having the same vertical scale for all such graphs can be very helpful for that purpose.
concMin	The lower limit on the vertical axis of graphs showing concentration values. In mg/L. If concMin = NA then lower limit is just below the minimum value. The reason this might be used is for making several similar graphs that show a comparison among data sets. Having the same vertical scale for all such graphs can be very helpful for that purpose. (Note this argument is only used on log scales, for linear scales the minimum is always zero).
fluxUnit	Determines what units will be used for flux, see printFluxUnitCheatSheet() or Appendix D for a listing of the codes
fluxMax	The upper limit on the vertical axis of graphs showing flux values. fluxMax is expressed in the units specified by fluxUnit. If fluxMax = NA then the upper limit on the graph is set just above the maximum value. The reason this might be used is for making several similar graphs that show a comparison among data sets. Having the same vertical scale for all such graphs can be very helpful for that purpose.

The functions `plotConcTime` and `plotLogConcTime` may a new concept to some users. Let me comment on them and show some examples. As useful as WRTDS is, it is often valuable to go back to the sample data and look to see just how clear some of the patterns are when the data haven't been so heavily smoothed. In many cases, using a simpler scatter plot can be more convincing of something that is happening, than a rather abstract representation of the data that have been run through a complex smoother. These two functions are designed with that in mind. For example, we may want to visually examine the idea that nitrate concentrations below some threshold discharge value have been increasing over time, but concentrations at higher discharges might not be doing this. These graphs can help demonstrate that. It could also be that we think this is particularly true in the late Fall and Winter. Again we can narrow the criteria and plot those data (lower flows, `paLong =5`, `paStart =11` for example). Or, perhaps we think that there might only be a change in some more moderate flow range, not extreme low flow and not high flow either. Again these functions can provide this. Here are two examples:



The left hand figure was generated with the command:

```
plotConcTime(qUnit=1,qUpper=300,paLong=8,paStart=7,concMax=3)
```

The right hand figure was generated with the command:

```
plotConcTime(qUnit=1,qLower=300,paLong=8,paStart=7,concMax=3)
```

What they show is that for lower discharge values in the July-Feb period, nitrate concentrations have been increasing progressively over time. This does not appear to be the case for the higher discharge values.

One note about any of the plotting functions that show the sample data: If a value in the data set is a non-detect. Then it is displayed on a graph as a vertical line. The top of the line is the reporting limit and the bottom is either zero, or if the graph is plotting log concentration values, the minimum value on the y-axis. This line is an "honest"

representation of what we know about that observation and doesn't involve us using a statistical model to fill in what we don't know. If you find odd things in the data set that you want to check further, see Section 5 for suggestions on how to do that.

flowDuration

This is a utility function that can help define the flow ranges that we want to explore. It prints out key points on the flow duration curve. They are defined for a particular part of the year, although they can be done for the entire year. The flow duration curve is always expressed in whatever discharge units the user specifies (`qUnit`). The idea of this seasonal flow duration curve is that we may want to know about the distribution of discharges during a period, say, of March, April and May. To do that we identify an approximate middle day of that period (call it April 15) and we know that the number of days in that three-month period is 92 days, so we set a half-window width of 46 days (half of 92). The function brings together all of the days in this period centered around April 15 (of all the years in the `Daily` data frame) and constructs a flow duration curve from them and reports out a set of key values.

The call is:

```
flowDuration(centerDate = "09-30", localDaily = Daily, localINFO = INFO,
qUnit = 2, span = 365)
```

In the simplest case, where we wanted to use units of cubic meters per second and we wanted it to cover the full year, the call would simply be this.

```
flowDuration()
```

But, if we wanted it centered on April 15, and running for the months of March, April and May, and wanted the results in cubic feet per second, the call would be:

```
flowDuration(centerDate="04-15", qUnit= 1, span = 46)
```

The date must be in quotes and in must be in the format "mm-dd".

Here is an example using data from the Susquehanna River:

```
> flowDuration("04-15",qUnit=3,span=46)

Flow Duration for Susquehanna River at Conowingo, MD

Flow duration period is centered on April 15
And spans the period from February 28  To May 31

Discharge units are Thousand Cubic Feet per Second

       min        5%       10%       25%       50%       75%       90%       95%       max
     0.904    17.400    21.900    32.700    52.100    80.700   132.000   179.000   467.000
```

b. Estimating the WRTDS model and looking at summary results

a) Overview

There are two steps that are **required** and one that is **wise**. The required steps are to run the function `modelEstimation` and then to run `setupYears`. The step that is wise is to save your results after running these two functions, using `saveResults`.

b) Model estimation

The first stage is to estimate the full WRTDS model. This is organized into a single function.

```
modelEstimation(localDaily = Daily,localSample = Sample, localINFO = INFO,
windowY=10, windowQ=2, windowS=0.5,minNumObs=100,minNumUncen=50,
env=parent.frame())
```

In the simplest case, where the user is prepared to use the default values for the window widths, and also the default values for the minimum number of observations, and minimum number of uncensored observations in order to run each regression, the call is simply:

```
modelEstimation()
```

Here is a little information about each of the arguments that the user might want to modify:

argument	Definition	Default value
`windowY`	The half window width for the time weighting, measured in years. Values much shorter than 10 usually result in a good deal of oscillations in the system that are likely not very realistic	10
`windowQ`	The half window width for the weighting in terms of ln(Q). For very large rivers (average discharge values in the range of many tens of thousands of cfs) a smaller value than 2 may be appropriate, but probably not less than 1.	2
`windowS`	The half window width for the seasonal weighting, measured in years. Any value >0.5 will make data from all seasons have some weight. Values should probably not be lower than 0.3 and there is no need to go higher than 0.5	0.5
`minNumObs`	This is the minimum number of observations with non-zero weight that the individual regressions will require before they will	100

	be used. If there too few observations the program will iterate, making the windows wider until the number increases above this minimum. The only reason to lower this is in cases where the data set is rather small. It should always be set to a number at least slightly smaller than the sample size. Any value lower than about 60 is probably in the "dangerous" range, in terms of the reliability of the regression.	
`minNumUncen`	This is the minimum number of uncensored observations with non-zero weight that the individual regressions will require before they will be used. If there are too few uncensored observations the program will iterate, making the windows wider until the number increases above this minimum. The only reason to lower this is in cases where the number of uncensored values is rather small. The method has never been tested in situations where there are very few uncensored values.	50

So, an example of the call in the case where the user wants to modify, for example, `windowQ` to a value of 1, and `minNumObs` to 75 but use the defaults for the others would be:

```
modelEstimation(windowQ = 1, minNumObs =75)
```

The arguments `localSample`, `localDaily`, and `localINFO` will always be set to the defaults for applications discussed in this chapter (that will change in chapter 6 where we deal with comparisons of varying data sets or methods). The final argument `env=parent.frame()`is a bit of arcane R code that allows this function to return multiple arguments. You don't need to understand it and you shouldn't modify it.

Here's what's going on in `modelEstimation`, there are four steps involved. Each one is carried out by a specific function. You don't need to call them yourself, but I'm naming them here to help you understand the process flow:

1. `estCrossVal`: This function applies the WRTDS model estimation method to the sample data set in a cross validation "jack-knife" context. That means that for each observation in the data set the function is creating a subset of the full data set, with that one data point left out. From this subset it estimates the WRTDS regression model and makes a single estimate for the date and discharge associated with the deleted observation. This kind of estimate provides a more realistic picture of the accuracy of the model because it doesn't have the opportunity to attempt to fit the model to the data point in question. It is like having separate calibration and validation data sets. The results of these jack-knife estimates are added to the `Sample` data frame where they can be used later in a variety of graphics describing the quality of the model.

2. `surfaceIndexParameters`: This function stores additional information into the INFO data frame that define how the WRTDS analysis is going to be carried out.

These consist of the 5 parameters that are defined in the table above, as well as 6 more that define the grid over which the “surfaces” are defined in the next step.

3. `estSurfaces`: This function estimates 3 "surfaces" that are the heart of the WRTDS model. The WRTDS model is run at a large number of grid points defined in a time versus ln(Q) space (e.g. for a data set of 30 years the grid consists of about 6200 points). The time steps for this grid are always 1/16th of a year (about 23 days apart, but the definition is strictly 0.0625 years). The steps in ln(Q) space are 1/13 of a range from a value that is the log of a discharge that is about 95% of the smallest daily discharge in the `Daily` data frame, up to a value that is about 105% of the maximum daily discharge in the `Daily` data frame. At each of these grid points the function computes and stores 3 values: they are the estimated log concentration (known as `yHat`), the estimated standard error of this value (known as `SE`), and the estimated concentration in mg/L (known as `ConcHat`). These results are stored in an array called “surfaces” which is three-dimensional. Later we will see that we can actually view each of these surfaces as a contour plot called `plotContours`.

4. `estDailyFromSurfaces`: This function uses these surfaces in conjunction with the individual daily values stored in the `Daily` data frame, and for the given values of time (`DecYear`) and discharge (`LogQ`) it interpolates values of `yHat`, `SE`, and `ConcHat` from the three surfaces generated in the previous step. The value of `ConcHat` for a given day becomes `ConcDay`. Then it is multiplied by the discharge and a unit conversion in becomes `FluxDay`. This step also computes the flow-normalized version of daily concentration and daily flux (called `FNConc` and `FNFlux`) and all of these are stored in the `Daily` data frame. Note that this is the most time consuming step of all (it can take 5 to 7 minutes of computer time on 30 year data set). The program prints out some progress indicators so you can determine that it is really running and how long you have to take your coffee break! Some users may have special applications in which they don’t want to run the flow-normalized values. For this reason there is a separate version of this function called `estDailyWithoutNormaliztion`. If you have reason to do this, you can create a modified version of `modelEstimation` that substitutes `estDailyWithoutNormaliztion` in place of `estDailyFromSurfaces`, and run this modified code. It will be much faster and give the same results for `ConcDay` and `FluxDay` as in the regular version of `modelEstimation`.

The user doesn’t need to work directly with any of the functions named here, but it is important to understand the process that the function `modelEstimation` is using to create the results. If you look at the workspace before and after running `modelEstimation`, you will see that `Daily`, `Sample`, and `INFO` have all been augmented, and the object `surfaces` has been created.

c) Selecting the period of analysis and using it to run setupYears

The next step is for the user to select the period of analysis to use in looking at the summary results from WRTDS. [Note that the next three paragraphs are identical to material in section 3 – because the definitions are identical for WRTDS and flowHistory].

The first choice you need to make is what "period of analysis" to use. What is the period of analysis? If we want to examine our data set as a time series of water years, then the period of analysis is the water year. If we want to examine the data set as calendar years then the period of analysis should be the calendar year. We might want to examine the winter season, which we might want to define as December, January and February, then those 3 months become the period of analysis. We might even want to examine September only. Then September becomes the period of analysis. The only constraints on the definition of a period of analysis are these: It must be defined in terms of whole months (it can't start or end mid-month). It must be a set of contiguous months (like March-April-May or November-December-January). And it must have a length that is no less than 1 month and no more than 12 months. It can be uniquely defined by two arguments. They are called, `paLong` and `paStart`, both of which must be integers no less than 1 and no greater than 12.

The argument `paLong` is the length of the period of analysis in months, and `paStart` is the first month of the period of analysis. The month numbers are the calendar months (January is 1, December is 12). So: If we are using calendar years, `paLong` =12, `paStart = 1`. If water years, `paLong =12`, `paStart = 10`. If we are leaving out the winter we might have `paLong = 9`, `paStart = 3`, (making the period of analysis March-November) or if we just want the months of March, April, May and June we would have `paLong = 4`, `paStart = 3`. Or if we wanted the period of analysis to be September only, then `paLong = 1` and `paStart = 9`.

There is a convention for numbering the "years". The year is known by the calendar year in which it ends. That's the same approach we use with water years. The water year starting October 1, 1990 is called water year 1991. One other note about time: For time series graphs of annual averages the x-axis position of a value is always plotted at the mean date for the period, so for a calendar year, the data point is plotted mid-way through the calendar year. For a water year it is plotted mid-way through the water year (or ¼ the way through the calendar year). The x-axis on time series graphs of the data are always labeled such that the axis labels are for calendar years, so 1981.0 always means January 1, 1981, regardless of whether we are using calendar years or water years or some other period of analysis.

Now run the function `setupYears`. The period of analysis is selected by using the function `setupYears`. The call to this function is this:

```
AnnualResults<- setupYears(paLong = 12, paStart = 10, localDaily = Daily)
```

In the simplest case, where you would like to do the analysis by water years, the call would be:

```
AnnualResults<-setupYears()
```

For a winter season of Dec, Jan, and Feb, then it would be

```
AnnualResults<-setupYears(paLong=3, paStart=12)
```

What is happening in this command? It is taking the daily records of estimated concentrations, flux, flow-normalized concentration, and flow-normalized flux, as well as the actual daily discharge values and summarizing them into time series of the annual averages of these quantities for the period of analysis.

There is a similar function for creating a data frame of monthly results. Unlike the setupYears command, which is a necessary part of the workflow, this one is entirely optional.

`calculateMonthlyResults` takes the results from the `Daily` data frame and computes results on a monthly time step. These are stored in a data frame in which the columns are: the average value for the month of `Q`, `ConcDay`, `FluxDay`, `FNConc`, `FNFlux`, as well as `DecYear`. It also contains the sequence number of the month (`MonthSeq`). The command for computing the monthly results is:

```
MonthlyResults <- calculateMonthlyResults()
```

The Egret code doesn't actually use these results, but they can be very handy to have for tables of results. They can easily be printed out. Just type the command, `MonthlyResults`.

d) saveResults

Because of the computational time that has been invested in producing these results, it can be very useful to save them in a file that you can return to at a later time for analysis. This was described above at the end of the data preparation section. Assuming that you have already created the object `savePath`, the command is just:

```
saveResults(savePath)
```

This will now save all of the objects in your workspace. So, this means that it will save not only the data but also the results that have now been generated. This can be reloaded later. In most environments this reloading can be done with a pull-down menu command "load workspace" and then you can search for and select this file to be loaded. The name will contain the abbreviations for site and constituent that you specified in the `getMetaData` step.

e) blankTime

This is a function that can be used with data sets that have a big time gap in them. If the data-gap is less than about two years, I would say that you don't need to use this function, but for gaps longer than that, it seems prudent to use it. It turns out that the WRTDS will run even if there are multi-year time gaps in the water quality data set. However, the results during the period are meaningless. They should not be used. They can even show strong oscillations in the absence of real information. The solution is this. Go ahead and run `modelEstimation` as described above, but after you have run it, you will need to substitute an **`NA`** value for all the daily estimates during this period of no data. The `blankTime` function accomplishes that for you.

You must determine the period for which you think that the results should be blanked out. It is a judgment call. I would suggest that the blanked out period should be a little longer (a few months longer) than the length of the period of no data. It is also useful to have the blanked out period be whole years in length and these should be water years if your results will typically be reported in water years. If there are multiple gaps you can run this function multiple times. There may be a situation where during a long gap (of several years) there is a period with a few observations. If there are less than about a dozen observations in that gap you might want to use `blankTime` for the whole period with very sparse data. The information contained in those samples will still be used to inform the model, but we can suppress the reporting of results through this period because the sampling is so sparse.

The command is:

```
Daily<- blankTime(startBlank, endBlank, localDaily = Daily)
```

Let's say that in your judgment the period of water year 1980 through water year 1989 should be blanked out. The blanked out period might be defined as 1979-10-01 to 1989-09-30. In this example your commands would be:

```
startBlank<-"1979-10-01"
endBlank<-"1989-09-30"
Daily<-blankTime(startBlank,endBlank)
```

Or

```
Daily<-blankTime("1979-10-01","1989-09-30")
```

What the function is doing is taking the `Daily` data frame, and for `yHat`, `SE`, `ConcDay`, `FluxDay`, `FNConc`, `FNFlux` the missing value indicator of **`NA`** is substituted for the value calculated in `modelEstimation`. This new version of `Daily` is simply substituted for the old version.

If you are interested in monthly results then you need to re-run the function `calculateMonthlyResults`. The call is just:

```
MonthlyResults<-calculateMonthlyResults()
```

If you have already run `setupYears` you will need to do that again as well. The command, if working with water years is this.

```
AnnualResults<-setupYears()
```

If the period of analysis is different then paStart and paLong need to be defined in the command.

Looking at the annual results:

At this point (after having run modelEstimation and setupYears) we can start considering how to view the annual averages for the variables that have been calculated. The functions for plotting or tabulating these are: `plotConcHist`, `plotFluxHist`, `tableResults`, and `tableChange`.

f) plotConcHist

This function produces a graphic of the annual average concentration (and annual average flow-normalized concentration) versus time. The full call to the function is:

```
plotConcHist(yearStart=NA, yearEnd=NA, localAnnualResults = AnnualResults,
localINFO = INFO, concMax = NA, printTitle = TRUE, plotFlowNorm = TRUE)
```

For purposes of this section of the manual, the only arguments of interest are these.

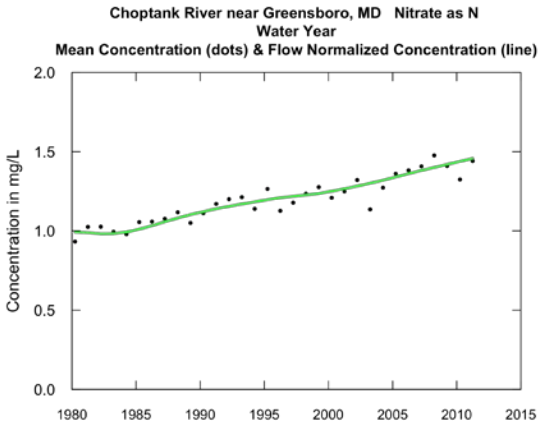
argument	meaning	default
`yearStart`	The starting year for the graph, typically expressed in whole year values, such as 1980.0, note that left margin of graph will probably be an earlier year, but the data will commence with the first yearly value after `yearStart`.	NA, which causes it to start at or before first year of record
`yearEnd`	The ending year for the graph. This should typically be the start of the calendar year **after** the last annual value you wish to plot. The right edge of the graph will typically be after `yearEnd`	NA, which causes it to end at or after last year of record
`concMax`	If specified, this defines the upper bound on the vertical axis of the graph. This can be very useful if there are going to be multiple graphs (for different sites) of the same constituent and for comparison purposes it would be nice to	NA, which causes the graph to be self-scaling.

	use the same vertical scale for all	
`printTitle`	If TRUE the title is printed above the graph. The reason you might not want to print it is for use in a publication illustration, where this information goes into a caption.	TRUE, title is printed
`plotFlowNorm`	If TRUE the graph shows the annual concentrations as circles and the flow-normalized values as a green curve. If FALSE, it only shows the annual concentrations.	TRUE, both are shown

A simple example of a call to this function would be:

```
plotConcHist(1980, 2012)
```

and the output might look like this:



This just means plot the concentration and flow-normalized concentrations that fall in the time range (1980-2012), and plot the title, and allow the graph to be scaled so the vertical axis ends slightly above the highest annual value.

One note about beginning and ending years: You can start the graph after the first annual value, but the data from prior to `startYear` will still have an influence on the data plotted, because with this (or any) smoothing procedure the data from a given year will have influence on the values computed for years before or after it. In some applications you might want to eliminate the influence of data from before a given date. The way to do that is to modify the data set prior to running `modelEstimation`. This will be described in section 6. In summary, the selection of `startYear` or `endYear` has no influence on the computation of the smoothed values. It only influences what part of the smoothed record is plotted.

g) plotFluxHist

The next function is `plotFluxHist`. It is very similar in its operation to `plotConcHist`. It produces a graph of the annual average flux values and annual average flow-normalized flux values for every year between `yearStart` and `yearEnd`. The full call to this function is:

```
plotFluxHist(yearStart = NA, yearEnd = NA, fluxUnit = 9, localAnnualResults =
    AnnualResults, localINFO = INFO, fluxMax = NA, printTitle = TRUE,
    plotFlowNorm = TRUE)
```

For purposes of this section of the manual, the only arguments that are of interest are these.

argument	meaning	default
`yearStart`	The starting year for the graph, typically expressed in whole year values, such as 1980.0, note that left margin of graph will probably be an earlier year, but the data will commence with the first yearly value after `yearStart`.	NA, which causes the graph to start at or before the first year of the record
`yearEnd`	The ending year for the graph. This should typically be the start of the calendar year after the last annual value you wish to plot. The right edge of the graph will typically be after `yearEnd`	NA, which causes the graph to start at after the last year of the record
`fluxUnit`	An index indicating what flux units to use (e.g. tons/day, kg/year, millions of kg/year). Options run from 1 to 12 (see table D-2 for these values). The choice should be based on the units that are customary for the audience but also, the choice should be made so that the flux values don't have too many digits to the right or left of the decimal point.	`fluxUnit=9` which means 10^6 kg/year
`fluxMax`	If specified, this defines the upper bound on the vertical axis of the graph. The reason that one may want this if there are going to be multiple graphs (for different sites) of the same constituent and for comparison purposes it would be nice to use the same vertical scale for all. It is expressed in the units specified by `fluxUnit`.	`NA`, which causes the graph to be self-scaling.
`printTitle`	If TRUE the title is printed above the graph. The reason you might not want to print it is for use in a publication illustration, where this information goes into a caption.	TRUE, title is printed
`plotFlowNorm`	If TRUE the graph shows the annual flux values as circles and the flow-normalized values as a green curve. If false, it only shows the annual flux values.	TRUE, both are shown

A simple example of a call to this function would be:

```
plotFluxHist(1980, 2012, fluxUnit=8, fluxMax=10)
```

This just means plot the flux and flow-normalized flux that fall in the time range (1980-2012), and print the title, and plot the fluxes in units of thousands of kg/year (that is what `fluxUnit=8` calls for) and have the axis scaled to run from 0 to 10,000 kg/year.

h) tableResults

This function produces a simple text table that contains the annual values for the results. Each row of the output represents a year and it prints: year, average discharge, average concentration, flow normalized concentration, average flux, and flow normalized flux.

The full call to the function is:

```
tableResults(localAnnualResults = AnnualResults, localINFO = INFO, qUnit = 2,
fluxUnit = 9)
```

The argument `qUnit` specifies what units to use to print the annual discharges, and `fluxUnit` specifies what units to use to print the flux values. Full lists of the unit options are provided in Tables D-1 and D-2. So, a simple example of this call would be:

```
tableResults(qUnit = 2, fluxUnit = 3)
```

The output would look like this:

```
Choptank River near Greensboro, MD
Nitrate as N
Water Year

Year   Discharge    Conc    FN_Conc     Flux    FN_Flux
          cms            mg/L             kg/day

1980      4.25     0.932     0.992       313       288
1981      2.22     1.025     0.989       183       295
1982      3.05     1.026     0.983       268       298
1983      4.99     0.995     0.983       357       302
1984      5.72     0.980     0.993       431       308
```

..... *etc*. through 2011

The table comes out printed on the console. If you want to save it as a file for future reference the simplest approach is to copy and paste it into a Word document. When you get it in the Word document you will see that the columns become somewhat irregular (because Word is probably using a proportionally spaced font). You can straighten it out by changing the font to Courier or Monaco. Later releases of this software will allow for this to be directly written to a file. In addition, you can have the function return the table as an object and this object can be used to create a file which can be input to Excel or other programs for further formatting (particularly useful in

applications involving many stations and/or analytes). The command would look something like this:

```
tableOut <- tableResults(qUnit = 2, fluxUnit = 3)
```

Hint: There is a simple way to be reminded of the unit codes, and that is to invoke these two commands.

```
printFluxUnitCheatSheet()
printqUnitCheatSheet()
```

Another suggestion is just to print out those cheat sheets and tape them to the edge of your computer or someplace handy.

i) tableChange

This is a function that provides for the computation of changes or slopes between any selected pairs of time points. These computations are made only on the flow-normalized results. For example, if our interest is in change over the time period 1980-2010 and also the sub-periods 1980-1995 and 1995-2010 this function would make those change-calculations for us.

The full call to the function is:

```
tableChange(localAnnualResults = AnnualResults, localINFO = INFO, fluxUnit =
9, yearPoints = NA)
```

`fluxUnit` is as defined above. The argument `yearPoints` requires some explanation. `yearPoints` is a vector of year values that are the break points for these calculations of change or slope. The calculation is made between every possible ordered pair of these values. So, for the example mentioned above we would like the vector `yearPoints` to have the following values: 1980, 1995, 2010. This will result in change-calculations for 1980-1995, 1980-2010, and 1985-2010. It is not required of the user to specify `yearPoints`. The default is `NA`. The result of leaving it out of the call to the function is that the program will use an algorithm to pick these change points and they will be every 5 years. Typically, this will not be very satisfactory. The process of specifying these change points is simple. In our example, we would type in this one command before calling `tableChange`. The command is this:

```
yearPoints<-c(1980,1995,2010)
```

This is how one creates a vector of values in R. Note here that these year values must be in ascending order. Also, they should always be integers. You can enter any number of values, as long as it is greater than 1. If you wanted to specify many values and they are equally spaced you can use the `seq` function in R (for sequence). For

example if you want to go from 1980 by steps of 5, up to 2010 you can say.

```
yearPoints<-seq(1980,2010,5)
```

You have yet another option, rather than creating a new object (this vector `yearPoints`) you can just specify it in the command. So, here are two options for how to make the call.

```
tableChange(fluxUnit=3,yearPoints=yearPoints)
tableChange(fluxUnit=3, yearPoints=c(1980,1995,2010))
```

The output from this function goes to the console and it provides, for every pair of years, eight results: The following example should illustrate the type of information produced. It includes concentration and flux, change magnitude and change slope, and presentation in real units and in percentage terms.

Here is an example of output from this function using the call stated above.

```
Choptank River near Greensboro, MD
   Nitrate as N
   Water Year

            Concentration trends
   time span       change     slope    change     slope
                     mg/L   mg/L/yr         %       %/yr

 1980  to  1995      0.21     0.014        21        1.4
 1980  to  2010      0.45     0.015        45        1.5
 1995  to  2010      0.24     0.016        20        1.3


                  Flux Trends
   time span           change        slope       change        slope
                     kg/day       kg/day   /yr      %         %/yr
 1980  to  1995           76          5.1           27          1.8
 1980  to  2010          121            4           42          1.4
 1995  to  2010           45            3           12         0.81
```

One particularly interesting result to look for in these tables is to look at change in % for concentration and for flux for any given pair of years. The flexible form that WRTDS uses allows for there being different percentage trends in concentration and in flux. For example, in this case, from 1995-2010 concentration increased 20% but flux only increased by 12%. This tells us that the increases in concentration are somewhat more focused in the low discharge range than the high discharge range. The discrepancies can be quite large and sometimes the two slopes can have opposite signs. Note that most other trend methods such as Estimator or LOADEST or Seasonal Kendall do not provide a way to discriminate between trends in concentration and trends in flux. In any of these other methods, concentration slope in percent will always be equal to flux slope in percent.

j) tableChangeSingle

This function operates exactly the same as `tableChange`, but it provides either concentration results or flux results, but not both. This can be useful when producing many output tables for a report that is entirely focused on concentration or one that is entirely focused on flux. The arguments are identical to those for tableChange, except that the final two arguments are these. The first is a logical argument to indicate if a data frame of output should be returned (for later manipulation and printing through other programs such as Excel), this argument is `returnDataFrame`, and its default is `FALSE`. The final argument is `flux`, and the default is `TRUE`. When `flux=TRUE` the output is only for flux, and when `flux=FALSE` the output is only for concentration. A call for this command might look like this (using the same example as above)

```
changeOutput <- tableChange(fluxUnit=3, yearPoints=c(1980,1995,2010),
returnDataFrame = TRUE)
```

this would create an object called `changeOutput` that contains only the flux trends. Or the call could be

```
changeOutput <- tableChange(fluxUnit=3, yearPoints=c(1980,1995,2010),
returnDataFrame = TRUE, flux=FALSE)
```

and it would only provide output for concentration trends.

A. Exploring the fit of the WRTDS model (including flux bias)

Once you have a fitted model it is valuable to check the quality of the fit. This can be done with a variety of graphical functions. These functions are described in the following table. Following that table is a table of the common arguments used within these functions.

Function name	Purpose	Arguments that you might want to specify, shown here set at their default value
`boxResidMonth`	A graphic of 12 box plots of the residual values organized by month	`stdResid = FALSE` `printTitle = TRUE`

`plotConcTimeDaily`	A plot of the time series of daily concentration estimates and the sample values for the days that were sampled. This works best when it covers no more than 4 years.	`startYear`
		`endYear`
		`printTitle = TRUE`
		`concMax = NA`

`plotFluxTimeDaily`	A plot of the time series of daily flux estimates and the sample values for the days that were sampled. This works best when it covers no more than 4 years.	`startYear`
		`endYear`
		`printTitle = TRUE`
		`fluxUnit = 3`
		`fluxMax = NA`

`plotConcPred`	A plot of observed concentration versus estimated concentration	`concMax = NA` `printTitle = TRUE`

`plotFluxPred`	A plot of observed flux versus estimated flux	`fluxUnit = 3`
		`fluxMax = NA`
		`printTitle = TRUE`

`plotLogConcPred`	A plot of observed concentration versus estimated concentration, as a log-log graph.	`concMax = NA` `printTitle = TRUE`

`plotLogFluxPred`	A plot of observed flux versus estimated flux, as a log-log graph.	`fluxUnit = 3`
		`fluxMax = NA`
		`printTitle = TRUE`

`plotResidPred`	A plot of residuals versus the estimate values (all in log concentration units).	`stdResid = FALSE` `printTitle = TRUE`

`plotResidQ`	A plot of residuals versus log discharge	`qUnit = 2`
		`stdResid = FALSE`
		`printTitle = TRUE`

`plotResidTime`	A plot of residuals versus time. Residuals are in log concentration units.	`stdResid = FALSE` `printTitle = TRUE`

`fluxBiasMulti`	Produces a graphic containing six small plots that are useful in exploring flux bias. They are the same as: `plotLogConcQ`, `plotResidQ`, `plotLogConcPred`, `plotResidPred`, `plotFluxPred`, and `plotLogFluxPred`. Plot title also gives estimate of the bias (est - obs)/obs.	`qUnit = 2`
		`fluxUnit = 3`
		`moreTitle = " "`

The arguments used in one or more of these functions are these:

Argument	Definition
`qUnit`	Determines what units will be used for discharge, see `printqUnitCheatSheet()` for a listing of the codes
`stdResid`	This is an option. If FALSE, it prints the regular residuals (they are in ln concentration units). If TRUE, it is the standardized residuals. These are the residuals divided by their estimated standard error (each residual has its own unique standard error). In theory, the standardized residuals should have mean zero and standard deviation of 1.
`printTitle`	If TRUE the plot has a title. If FALSE no title (useful for publications where there will be a caption).
`startYear`	The starting date for the graph, expressed as decimal years, for example, 1989.0.
`endYear`	The ending date for the graph, expressed as decimal years, for example, 1996.0.
`moreTitle`	A character variable that adds additional information to the graphic title. Typically used to indicate what the estimation method was (e.g. WRTDS or LOADEST). Default is " " which indicates that nothing is added to title.
`concMax`	The upper limit on the vertical axis of graphs showing concentration values. In mg/L. If `concMax = NA` then the upper limit on the graph is set just above the maximum value. Setting an upper limit can be very useful if there are going to be multiple graphs (for different sites) of the same constituent.
`fluxUnit`	Determines what units will be used for flux, see `printFluxUnitCheatSheet()` for a listing of the codes
`fluxMax`	The upper limit on the vertical axis of graphs showing flux values. `fluxMax` is expressed in the units specified by `fluxUnit`. If `fluxMax = NA` then the upper limit on the graph is set just above the maximum value. Setting an upper limit can be very useful if there are going to be multiple graphs (for different sites) of the same constituent.

It has been shown that under some circumstances, regression-based models such as LOADEST, Estimator, or WRTDS can end up computing average fluxes that are highly biased (positively or negatively). These kinds of problems can be readily identified by looking at a variety of scatter plots that provide insight on the quality of the model fit.

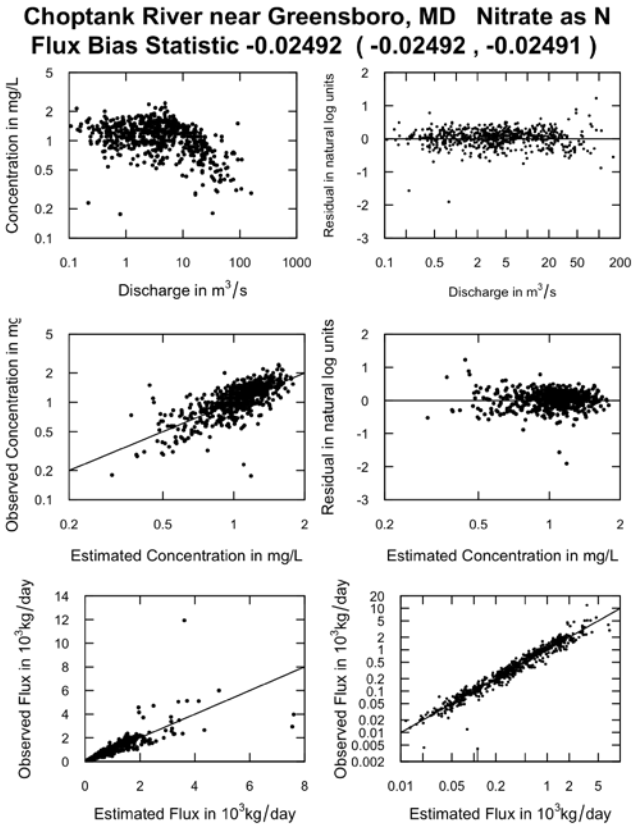
These problems most often arise because the flow versus concentration relationship fits very poorly at the highest discharges (where a large fraction of the annual flux takes place). This comes about because there are often relatively few data points in this very high discharge range and the model is being fit to all of the data (in the case of LOADEST or Estimator) or to a portion of the data that includes high and moderate flows (in the case of WRTDS). Recognizing that there is often great uncertainty about the "true" shape of the relationship in this very high range of discharges, exploring the scatter plots can commonly reveal those cases where the model is dramatically over- or under-estimating concentrations in this range. WRTDS offers substantial protection against this problem (because of its flexibility) as compared to LOADEST, but with any model looking at some diagnostic plots is a crucial step in helping the analyst protect against this problem. The evaluation of flux bias can be easily done in Egret by using from the graphics in `fluxBiasMulti` and `plotFluxTimeDaily`.

a) fluxBiasMulti

This function shows a total of 6 small graphics that can help identify problems in the fit. For example, with the call:

```
fluxBiasMulti(fluxUnit=4)
```

We get this graphical result:



The ones that show the residuals or the log of concentration on the vertical axis provide a way of looking at the fit in terms in log space, which is the space in which the fitting is done. Those that show flux values get more directly at flux bias, particularly the plot in the lower left corner which shows observed flux versus predicted flux. The analyst should note whether the observations are roughly evenly distributed above and below the 1:1 line for any given value of predicted flux. This graph also reports the flux bias statistic. This is computed as follows: (mean(predicted) - mean(observed)) / mean(observed). A perfect result would be 0.0. A value of 0.2 would mean that predictions are 20% higher than observed on average. A value of -0.5 would mean that predictions are 50% lower than observed on average. In this graphic we see that the mean of the predicted fluxes is about 2.5% lower than the mean of the observed fluxes. That's a very satisfactory result. Serious flux bias problems are those where the bias is well over 10% (positive or negative).

There are two values shown in parenthesis after the flux bias statistic. They are lower and upper bounds on its value. If there are no censored values in the data set then these numbers will be identical to the overall flux bias statistic. If there are censored values these numbers will bracket the overall flux bias statistic. The first number in parenthesis is computed by setting all the censored values to its detection limit. The second number is computed by setting all of the censored values to their lower bound (which is typically zero). The overall flux bias statistic is the average of these two values. In many cases all three numbers will be virtually equal, but in heavily censored cases they may show some differences.

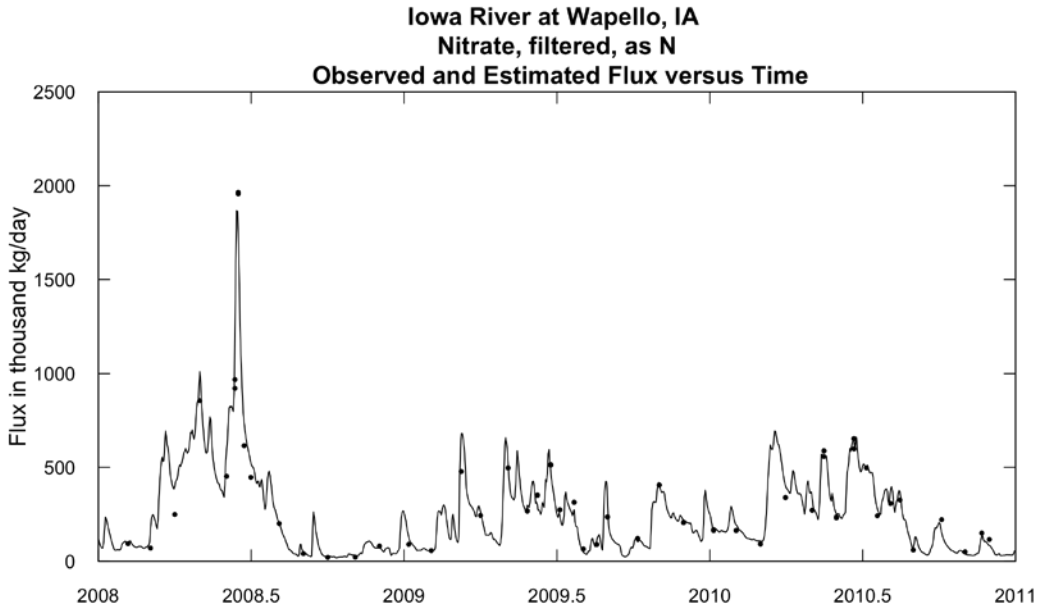
This graphic is not designed for publications or presentations, but it is a good guide to the flux bias problem and can point you to the single plot graphic might be best for displaying the quality of the fit. Any one of them can be produced as a stand-alone figure by calling the function directly.

b) plotFluxTimeDaily

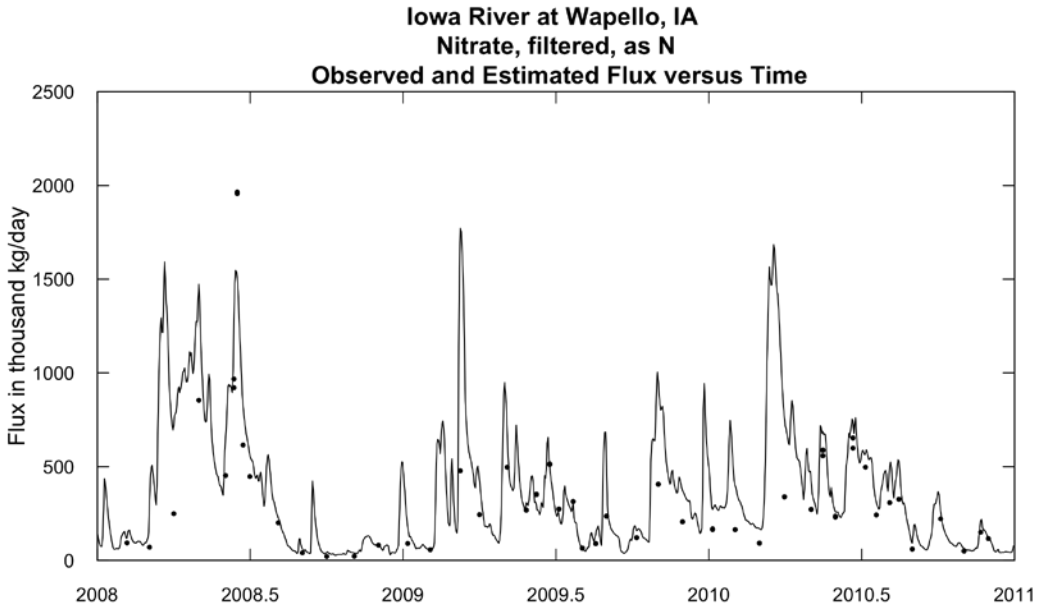
This is another other useful graphic for seeing the flux bias issue.

Here is a WRTDS result from a site where LOADEST has a serious bias problem. The call that produced this was:

```
plotFluxTimeDaily(2008, 2011, fluxUnit = 4, fluxMax = 2500)
```



This result looks very satisfactory. In contrast, if a 7-parameter LOADEST model were used to instead of WRTDS this graph would look as follows (Section 6 explains how we can make such a plot).



Note how the model estimates are 3 to 4 times higher than the sample values during most of the time when the fluxes were estimated to be above about 500,000 kg/day. In cases with serious flux bias problems there will be a substantial difference between the estimates and the sample values, and the difference will be consistently on the over-estimation or under-estimation side. This graph works best when it shows no more than about 4 years of data. For exploration of the results you can run it several times looking at different portions of the record.

D. Viewing the model behavior through contour and other types of plots.

The WRTDS approach is centered on the idea of creating a highly flexible model of the evolving behavior of the system. This is best illustrated in the form of a contour plot that shows the expected value of concentration as a function of time and discharge. There is also a plot that shows the difference in the contour plot between any two given years. There are also plots that represent vertical slices through that graph taken at as many as four different points in time. These are presented here.

a) plotContours

The full call for this function is this:

```
plotContours(yearStart, yearEnd, qBottom, qTop, whatSurface = 3,
    localsurfaces = surfaces, localINFO = INFO, localDaily = Daily, qUnit = 2,
    contourLevels = NA, span = 60, pval = 0.05, printTitle = TRUE, vert1 = NA,
    vert2 = NA, horiz = NA, flowDuration = TRUE)
```

The following table describes the meaning of all of the arguments (except localsurfaces, localINFO and localDaily which at this stage should all be left at their default values).

Argument	Definition
yearStart	The starting date for the graph, expressed as decimal years, for example, 1989.0.
yearEnd	The ending date for the graph, expressed as decimal years, for example, 1996.0.
qBottom	The discharge value that should form the bottom of the graph, expressed in the units specified by qUnit. The actual bottom could be lower (it will be set at values such as 1, 2, 5, 10, 20, 50, 100...). Using the function flowDuration can be very helpful for identifying reasonable levels for qBottom. A good starting point would be to set this slightly below the 5% level on the flow duration curve.
qTop	The discharge value that should form the top of the graph, expressed in the units specified by qUnit. The actual top could be higher (will be set at values such as 1, 2, 5, 10, 20, 50, 100...). Using the function flowDuration can be very helpful for identifying reasonable levels for qTop. A good starting point would be to set this slightly above the 95% level on the flow duration curve.
whatSurface	default = 3. This should generally be at its default value. At whatSurface = 3, the plotted surface shows the expected value of concentration. For whatSurface = 1, it shows the yHat surface (natural log of concentration). For whatSurface = 2, it shows the SE surface (the standard error in log concentration).
qUnit	Determines what units will be used for discharge. See printqUnitCheatSheet() for a listing of the codes
contourLevels	Default value is NA. With the default value the contour intervals are set automatically. These will generally NOT be a very good choice, but they may provide a starting point (discussed further below).
span	Default value = 60. Specifies the smoothness of the flow duration information that goes on this graph. A larger value will make it smoother. The default should work well in most cases.
pval	Default value = 0.05. The probability value for the flow frequency information shown on the plot. The plot has two black curves on it. In the default value case these are at the 5% and 95% levels on the

	seasonal flow duration curve. `pval = 0.01` would place these at the 1% and 99% points. `pval = 0.1` would place them at 10% and 90%.
`printTitle`	If TRUE the plot has a title. If FALSE no title (useful for publications where there will be a caption).
`vert1`	Default = **NA**. This simply plots a vertical black line on the graph at a particular time (defined in decimal years). It is used to illustrate the idea of a "vertical slice" through the contour plot, which might then be shown in a subsequent use of `plotConcQSmooth`.
`vert2`	Default = **NA**. This gives the location of a second vertical black line on the graph at a particular time (defined in decimal years).
`horiz`	Default = **NA**. This simply plots a horizontal black line on the graph at a particular discharge value (defined in the units specified by `qUnit`). It is used to illustrate the idea of the seasonal cycle in concentrations for a given discharge and the long-term change in this cycle.
`flowDuration`	Default = TRUE. If TRUE it draws the flow duration lines at the specified probabilities. If FALSE, the flow duration lines are left off.

Explanation of how to set the contour levels: It can take a couple of iterations to pick a good set of contour intervals. Having done an initial run of `plotContours`, consider first if the `qBottom` and `qTop` should be changed before you adjust the contour intervals. An ideal choice for these would be such that the 5% and 95% flow duration lines are entirely inside the plot, but the plot does not extend very far beyond them.

The initial plot will set up a scale for the contours which will likely extend to values that are too high and not provide much discrimination in the range of interest, but it can provide a guide for adjusting to a more reasonable range. For selecting the contour intervals, what you want is a set of about 6 to 8 evenly spaced contour levels that covers this range of values. For example, let's say that the estimated concentrations run from 0 to 10 mg/L, then you might want contours at 0, 2, 4, 6, 8 and 10. We can specify this by the command:

```
cLevel<-seq(0,10,2)
```

and then in the call to `plotContours`, we would specify within the call `contourLevels = cLevel`

For those not familiar with R, the command `cLevel<-seq(0,10,2)` means the following. Create a vector called `cLevel` and have it consist of the numbers from 0 to 10 in steps of 2.

As another example, suppose that the values run from 0 to 60, we might then give the command:

```
cLevel<-seq(0,60,10)
```
There may be cases where the expected value of concentration never approaches zero. For example, we may have expected concentrations that all fall between 4 and 9. In this case we might give the command:

```
cLevel<-seq(4,9)
```
Note that if the step size is 1, then there is no need for the third argument in the `seq` function.

In some cases one of the contour intervals may be shown in white. I find that this makes it harder to grasp the pattern. This typically happens when `cLevel` has 6 values (creating a total of 5 intervals). This can be avoided by setting up more intervals. So, instead of `seq(0,50,10)` we might use `seq(0,60,10)`or `seq(0,50,5)`. Also note that if some part of the surface is above the upper limit of these contour levels it will be shown in white.

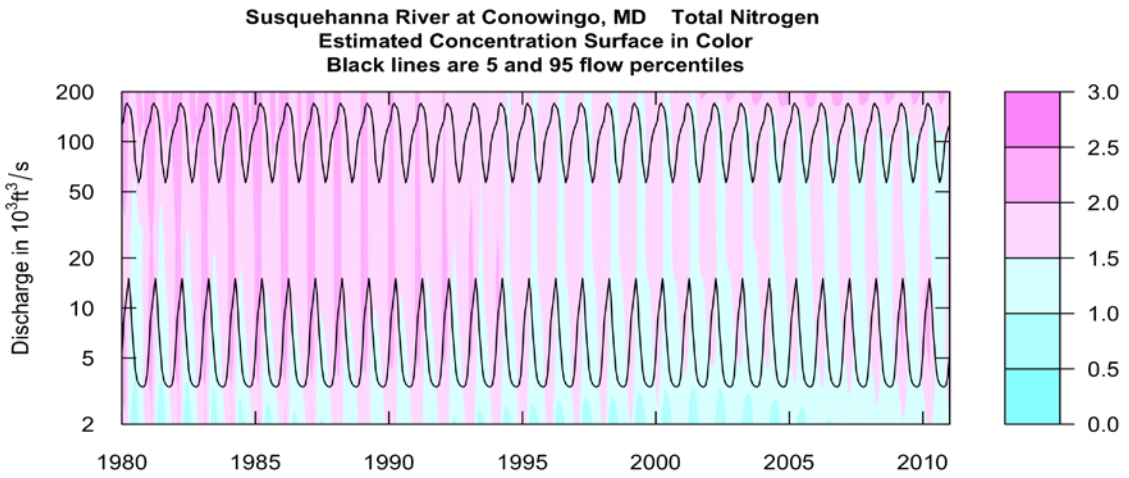
Because contour plots like this may be very unfamiliar to the user, here is an example. The commands to produce it were these:

```
cLevel<-seq(0,3,0.5)
plotContours(1980,2011,2,200,contourLevels=cLevel,qUnit=3)
```

This could also be produced all within a single command as:

```
plotContours(1980,2011,2,200,contourLevels=seq(0,3,0.5),qUnit=3)
```

The resulting output was this:



b) plotDiffContours

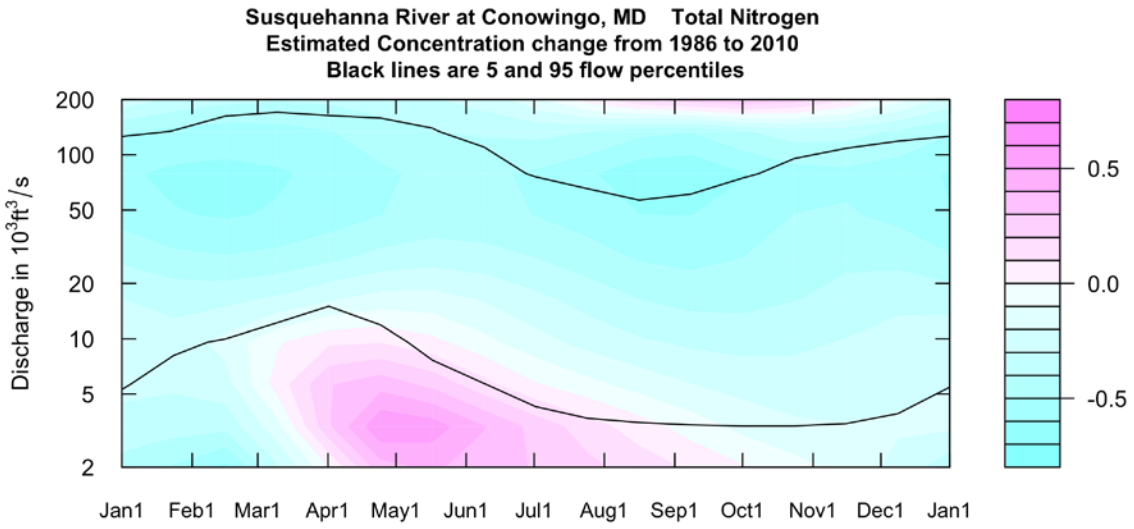
This function takes any two selected calendar years out of the entire contour plot and subtracts the earlier one from the later one and makes a contour plot of the differences between the two. This is a good way to visualize the changes represented by the contour plots and evaluate the extent and direction of the changes as a function of season and discharge level. The full call to the function is:

```
plotDiffContours(year0, year1, qBottom, qTop, maxDiff, whatSurface = 3,
    localsurfaces = surfaces, localINFO = INFO, localDaily = Daily,
    qUnit = 2, span = 60, pval = 0.05, printTitle = TRUE,
    vert1 = NA, vert2 = NA, horiz = NA, flowDuration = TRUE)
```

Most of these arguments are identical to those for `plotContours`. The only new ones are `year0`, `year1`, and `maxDiff`. Here is what they mean. The arguments `year0` and `year1` are the particular calendar years that are to be compared. So, if we want to compare 2010 to 1990 we would set `year0` to 1990 and `year1` to 2010. These two arguments should both be integers. Note that the calendar year specified by `year1` must be completely within the time period in the computed surfaces. So for example, in `plotContours` we may have the first two arguments be 1980 and 2011 as shown in the example above so that whole of 2010 is shown. But in `plotDiffContours` we would want the first two arguments to be 1980 and 2010, not 2011. The variable `maxDiff` is intended to represent the absolute value of the largest difference between the surface in `year0` and the surface in `year1`. Having a good version of `plotContours` can usually help you to pick a good starting value for this, but you are likely to need to iterate on it a few times. An example of a difference plot based on the same data set is shown here.

```
plotDiffContours(1986,2010,2,200,maxDiff=0.8,qUnit=3)
```

The output was this:



What we can say in words about the results shown in this figure might be the following:

- Over most of the flow range for all parts of the year, total nitrogen concentrations have declined over the time period 1986 through 2010.
- There is a small indication of increases at very low discharge during the March - July period, but these are focused on extremely low discharges, such as 3,000 to 5,000 cfs.
- There is also an indication of increases at the most extreme high discharges (above 150,000 cfs) particularly in the summer and fall (typically related to hurricane or tropical storm events).
- The largest declines in concentration are in the discharge range of 40,000 to 100,000 cfs, and these seem to cover most of the year, although the biggest improvements seem to be in the winter.

c) plotConcQSmooth and plotLogConcQSmooth

Another way to visualize a model of the change over time is to use `plotConcQSmooth` (or its alternative `plotLogConcQSmooth` with concentrations plotted on a log scale). This graphic is the equivalent of making a simple x-y plot of 1 or 2 or 3 vertical slices through the contour plot produced as output to `plotContours`. Unlike `plotContours` or `plotDiffContours`, this plot does not use the array of output from the stored WRTDS model (in `surfaces`), but rather it uses WRTDS "on the fly" to plot these results. The user has control over the window widths and thus can experiment with them to gain insight about what might be the best window widths to use in their application. Thus, these plots can be made before `modelEstimation` has been run, and can help determine what might be the best window widths to use for the data set. They can also be run after `modelEstimation` serving as another way to describe the change in system behavior.

The call for this function is:

```
plotConcQSmooth(date1, date2, date3, qLow, qHigh,
    qUnit = 2, legendLeft = 0, legendTop = 0, concMax = NA,
    bw = FALSE, printTitle = TRUE, printValues = FALSE,
    localSample = Sample, localINFO = INFO, windowY = 10,
    windowQ = 2, windowS = 0.5)
```

The call for `plotLogConcQSmooth` is identical except for the addition of the argument `concMin` (default value is `NA`). The following is a table explaining all of the arguments (except for `localSample` and `localINFO` which should just be left at their default values).

Arguments used in these functions are:

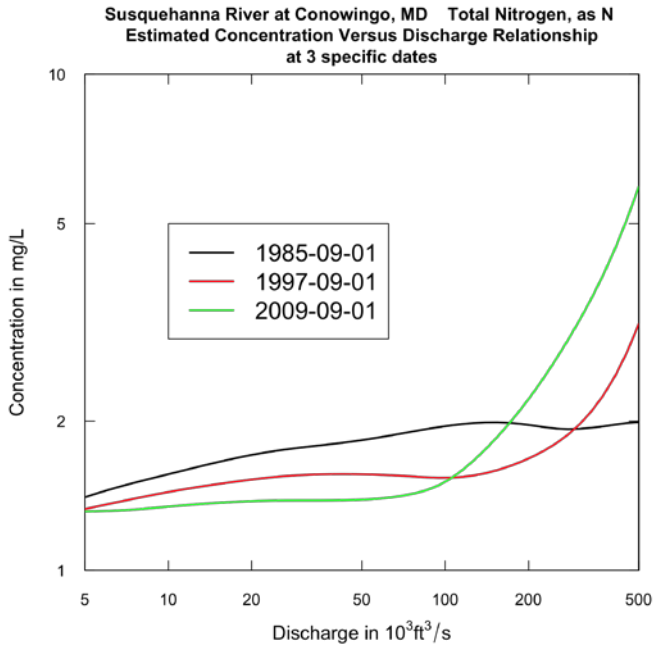
Argument	Definition

date1	This is the date for the first curve to be shown on the plot. It must be in the form "yyyy-mm-dd" (it must be in quotes)
date2	This is the date for the second curve to be shown on the plot. It must be in the form "yyyy-mm-dd" (it must be in quotes). If you don't want a second curve then the argument must be date2=NA
date3	This is the date for the third curve to be shown on the plot. It must be in the form "yyyy-mm-dd" (it must be in quotes). If you don't want a third curve then the argument must be date3=NA
qUnit	Determines what units will be used for discharge, see printqUnitCheatSheet() or Appendix D for a listing of the codes
qLow	The discharge value that should form the left edge of the graphic, expressed in the units specified by qUnit. Using the function flowDuration can be very helpful for identifying reasonable levels for qLow. A good starting point would be to set this near the 5% level on the flow duration curve.
qHigh	The discharge value that should form the right edge of the graphic, expressed in the units specified by qUnit. Using the function flowDuration can be very helpful for identifying reasonable levels for qHigh. A good starting point would be to set this near the 95% level on the flow duration curve.
legendLeft	This determines the placement of the legend on the graph. It establishes the left edge of the legend and is expressed in the flow units being used. The default (which is NA) will let it be placed automatically. The legend can end up conflicting with one or more of the curves. Once the location of the curves is established then this can be set in a way that avoids conflict.
legendTop	This determines the placement of the legend on the graph. It establishes the top edge of the legend and is expressed according to the concentration values on the y-axis. The default (which is NA) will let it be placed automatically. The legend can end up conflicting with one or more of the curves. Once the location of the curves is established then this can be set in a way that avoids conflict.
concMax	Maximum value for the vertical axis of the graph. The default is NA. The reason to set concMax is if you want to make several plots that have the same vertical axis.
concMin	[This one is only used in plotLogConcQSmooth]. Minimum value for the vertical axis of the graph. The default is NA. The reason to set

	concMin is if you want to make several plots that have the same vertical axis.
bw	Default is FALSE, which means we want a color plot. If bw=TRUE that means it should be black and white.
printTitle	If TRUE the plot has a title. If FALSE no title (useful for publications where there will be a caption).
printValues	If TRUE the estimated values that make up the plotted lines are printed on the console. If FALSE they are not printed. Default is FALSE. This could be useful if you wanted to compute various comparisons across time periods.
windowY	This is the half-window width for time in WRTDS. It has units of years. The default value is 10
windowQ	This is the half-window width for discharge in WRTDS. It has units of ln(discharge). The default value is 2.
windowS	This is the half-window width for seasons in WRTDS. It has units of years. The default value is 0.5.

Here is an example, with the same data set that is shown above:

```
plotLogConcQSmooth("1985-09-01","1997-09-01","2009-09-01", 5, 500,
     concMax=10, concMin=1, qUnit=3, legendLeft=10, legendTop=5)
```

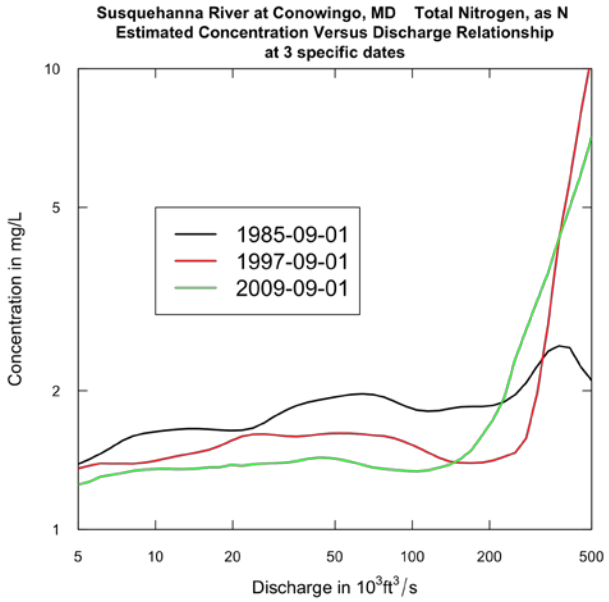


What this graph is telling us, for a period centered around September 1, is that as of 1985, TN concentrations didn't vary a great deal with discharge, ranging from an

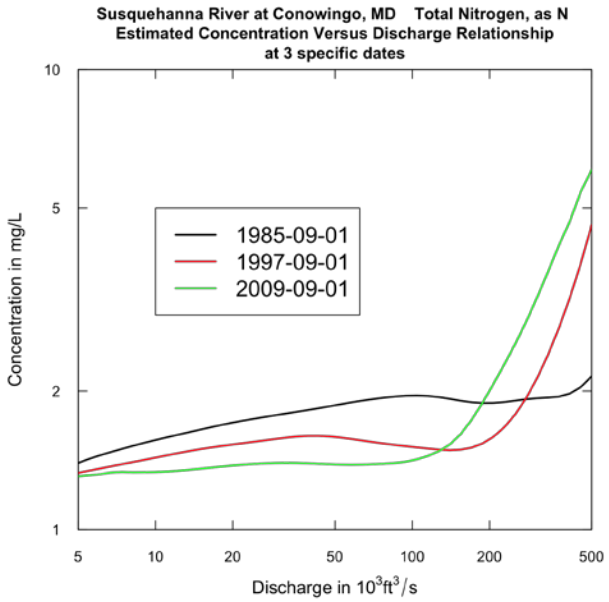
expected value of 1.5 mg/L at low flow to about 2.0 mg/L at very high flow. By 1997 that was starting to change, with concentrations being slightly lower at low to moderate discharges, but then turning upwards rather sharply above about 200,000 cfs and reaching an expected value of about 3 mg/L at 500,000 cfs. By 2009 the downward trend at low to moderate flows continues, but the upturn appears to come at around 100,000 cfs and the expected value of concentration appears to peak around 6 mg/L. This progression is quite reasonable because this sampling site is just downstream from a reservoir which has been filling with sediment, and the trapping effectiveness is decreasing over time and the discharge at which scour occurs has been decreasing over time. Also note how much the shape of the 1997 and 2009 curves depart from being a quadratic in log discharge. For an ESTIMATOR or LOADEST model these curves can only be linear or quadratic and must have the same shape at all times in the record.

We can experiment with decreasing the discharge window width, say from the default of 2.0 down to 0.8. This is the call and the result.

```
plotLogConcQSmooth("1985-09-01","1997-09-01","2009-09-01", 5, 500,
    concMax=10, concMin=1, qUnit=3, legendLeft=10, legendTop=5, windowQ=0.8)
```



This narrowing of the window has produced curves that are simply not credible. Why would they oscillate as they appear to here? After some experimenting, I concluded that a good choice of widths for Q is about 1.5. The resulting plot looks like this.



It is not much different from the one with `windowQ=2`, but the changes in slope is a bit more sharply defined but still credible.

There are times when these curves show a kind of "sawtooth" appearance. This can be an indication of a two possible problems. If the sawtooth is at the lowest or highest flows it may mean that the range of discharge values used in the plot is simply extending too far and it is trying to calculate these curves in a "region" where the data are very sparse. The sawtooth arises because of the process of incrementally extending the window widths to take in more data for the estimate. The other reason can be simply that the windows are too narrow. Selectively increasing them can help remove these artifacts and get to a more meaningful set of curves. These diagnostic curves can be repeated for other times of year and this can help refine the choice of window widths or help to describe the specific nature of the change that is happening.

5. Troubleshooting

This section describes a set of pointers about trouble-shooting with data sets, for example finding and fixing data errors, identifying individual data points to compare with other information sources, or shortening a data set.

There are many possibilities that one might consider for identifying an unusual observation and trying to track down information about it. We will look at a couple of possibilities here, and they should help to give users a better feel for some of the other things that the R-environment can facilitate. Let's say we use the function `plotConcTime()` and see one very extreme value. We might like to pin down what day it actually happened on and find out if there were other unusual things (in terms of discharge or other constituent concentrations) happening on or around that day.

Let's say there are one or two observations shown on this plot that are greater than 10 mg/L. We can use the R subset command to create a new data frame that is structured just like the `Sample` data frame but only contains those observations where the concentration value is >10. The command would be:

```
SampleHighVals<-subset(Sample,ConcHigh>10)
```

That just means, create a new data frame called `SampleHighVals`, and populate it with all the observations (rows) of data in which the variable `ConcHigh` is greater than 10 (we could have said `ConcLow` or `ConcAve`). Then we can say we just want to print out the content of this small data frame and see what we have. That command is just:

```
> SampleHighVals
           Date ConcLow ConcHigh Uncen ConcAve Julian Month Day  DecYear MonthSeq      SinDY
886  2004-09-20  20.065   20.065     1  20.065  56510     9 264 2004.720     1857 -0.9822229
1020 2011-09-08  10.397   10.397     1  10.397  59054     9 251 2011.684     1941 -0.9163169
          CosDY        Q     LogQ     yHat        SE  ConcHat
886  -0.1877186 15432.68 9.644243 1.235097 0.2689783 3.565382
1020 -0.4004539 16763.57 9.726964 1.456867 0.3616951 4.582656
```

What is this output telling us? There are two observations above 10 mg/L. They are observations 886 and 1020. Their dates are 2004-09-20 and 2011-09-08. Their concentration values are 20.065 and 10.397 respectively and their discharges are 15,432 and 16,763 cubic meters per second respectively. We can go to NWIS Web and try to learn about what was going on with flow on and around these days (these are

among the highest discharge values during the period of record). We can see what the concentration values were for other constituents.

Here are instructions for finding out how many provisional discharge data are in the record and to see the provisional data plotted against time.

```
zp<-ifelse(Daily$Qualifier=="P",1,0)
```

```
sum(zp)
```

```
[1] 246
```

```
plot(Daily$DecYear,zp,type="l")
```

{more to come in this section but this will do it for now,}

6. Working with multiple versions of a data frame

At times the user might want to compare results for the same site and variable of interest, where the comparison might be the use of somewhat different data sets or methods of analysis (including choices about window widths). This can be facilitated by making use of alternative data frames. In reality these could be alternative objects that aren't strictly data frames. For example they might be an array like surfaces, but for simplicity here they will be referred to as data frames. The key to this process is to have these alternative data frames structured like one of the standard data frames. For example we have the `Sample` data frame, and it has a standard set of variable names such as `Sample$Date, Sample$Q, Sample$ConcLow,` etc. The alternative data frame might be called `shortSample` (named so because we are curious about how our analysis results would behave if we left off the first few years or last few years of data). `shortSample` would have variables within it with names just like Sample: `shortSample$Date, shortSample$Q, shortSample$ConcLow`, etc.

Let's look at a real example. In the Choptank River example (seen in the standard workflow shown in Appendix F we retrieved all of the samples back to 1979-09-01. But, let us say that we are looking at this as a part of a larger study, and the other records in the study didn't start until Water Year 1985, and we want to consider the idea that we might want to leave out the Choptank data prior to 1985 and see if it substantially influences our results for the period starting in 1985. First, a comment about this question: It is certain that there will be some influence on the later results because the model for 1985 and beyond is influenced by data from before 1985 (because of the windows). The question is, how large is that influence? A second comment is that there is probably no right or wrong answer as to how to proceed with such a study. The argument for using the earlier data is that one should always work with the greatest amount of information available. The argument against is that if we used the earlier data then this one site would show behaviors around 1985 that are influenced by events (regional climatic events and/or regional changes in land use or cultivation practices) that preceded 1985 but they were excluded from the other data sets, because such data were not available. In any event, we may wish to make this long-record/short-record comparison. Let us assume here that we have already run the retrievals and fit the model and have created the `Sample, Daily`, and `INFO` data frames and the results in the form of surfaces, `MonthlyResults`, and `AnnualResults`.

Now, to have analyses for the short record we would modify our use of each of the functions as follows.

```
smallStartDate<-"1984-10-01"
smallSample<-getSampleData(sta,param,smallStartDate,EndDate)
summary(smallSample)
length(smallSample$Date)
smallSample<-removeDuplicates(localSample=smallSample)
```

Now, let's pause and consider what is going on here. We have created a new object for the starting date, and using it we have created a new data frame called `smallSample` which is structured like `Sample` but has a more limited set of data in it. Then when calling the function we need to use the argument `localSample`. Up to this point in the manual we have always allowed these arguments like `localSample` (or `localDaily`, etc.) to be equal to their default value (`localSample = Sample`). Now that we have created these new, alternative, data frames we need to specify them when calling a function and put in these arguments. First we need to make copies of the data frames we have already created and make sure we know what they are. For example:

```
fullDaily <- Daily
fullSample <- Sample
fullINFO <- INFO
fullsurfaces <- surfaces
fullMonthlyResults <- MonthlyResults
fullAnnualResults <- AnnualResults
```

Now we can give the command:

```
modelEstimation(localSample=smallSample)
```

When it does the calculations it will draw upon `smallSample` for the data and not on `Sample` or `fullSample`. When it is done with the calculations, the results will all have the standard names (`Sample, Daily, INFO, surfaces, MonthlyResults`) and you would create a new version of `AnnualResults` with the command:

```
AnnualResults <- setupYears()
```

Now, if we want to make a graph of the flux history: we could make the plot as follows.

```
plotFluxHist(1985,2012)
```

and compare that to the version that uses the older data

```
plotFluxHist(1985,2012,localAnnualResults=fullAnnualResults,localINFO=fullINFO)
```

and then we would have created the two versions of the same graph, based on the different data sets. This same idea applies to any one of the graphical or table outputs. They can be run on each of the several data sets and see what the differences look like.

There is an alternative to this approach and that is to save the results of one analysis (using `saveResults`) and starting all over with a fresh data retrieval and save those results (again, using `saveResults`). The caution here is that there needs to be a way of distinguishing between the two saved workspaces. This can easily be achieved during the use of `getMetaData`. When it calls for abbreviations for the data set, the user can specify something in the station or parameter abbreviation that will make it possible to distinguish between the two different workspaces.

7 Water quality data entry when the analyte of interest is the sum of multiple constituents.

As mentioned in Section 2Aa there are cases where the analyte of is interest is computed as the sum of multiple constituents. Where there is censoring (presence of less than values) this can create a good deal of complexity. This may be best understood through the use of an example.

Assume here that we have a rule that says the following: In 2004 and earlier we compute total phosphorus (call it tp) as the sum of dissolved phosphorus (call it dp) and particulate phosphorus (call it pp). For 2005 and after we have direct measurements of total phosphorus (call it tp).

Lets say we have a table of data that looks like this.

cdate	rdp	dp	rpp	pp	rtp	tp
2/15/2003		0.02		0.5		
6/30/2003	<	0.01		0.3		
9/15/2004	<	0.005	<	0.2		
1/30/2005						0.43
5/30/2005					<	0.05
10/30/2005					<	0.02

This table could be used as an input source for constructing the `Sample` data frame (called by using the function `getSampleDataFromFile` introduced in section 2Ab.). The EGRET software will "add up" all the values in a given row to form the total for that sample. Thus in creating this table you should only enter data that should be added together. For example we might know the dp value for 5/30/2005, but we don't want to put it on this table because under the rules for this data set we are not supposed to be adding it in to get a total value in 2005.

The total number of columns in the table will be equal to 2 times the total number of analytes that are ever used in our computations, plus one, for the date.

Understanding what's going on

It is useful to understand what EGRET is doing with these data. For every sample, the WRTDS method in EGRET is using a pair of numbers (they are called `Sample$ConcLow` and `Sample$ConcHigh`) that define an interval in which the true value lies. `ConcLow` is the lower bound and `ConcHigh` is the upper bound.

Let's start with the simplest of all cases (where there is only one analyte and no censoring) and that would be the 1/30/2005 case. What we see here is that tp is

reported as 0.43. In this case ConcLow=0.43 and ConcHigh=0.43 (the interval collapses down to a single value).

Now, let us consider the case of 5/30/2005. Here the reported value of tp is < 0.05. For this case EGRET will set ConcLow to NA and ConcHigh to 0.05. This represents an interval (0, 0.05). We use NA instead of 0 because EGRET is going to take logs of the concentration data and log(0) will create an error but if we take log(NA) that is still NA and the algorithm in EGRET understands that the interval for the log of concentration runs from minus infinity to log(0.05).

For the case of 2/15/2003 the problem is simple. EGRET knows that it should add dp and pp so it considers the concentration to be 0.52, and sets ConcLow = 0.52 and ConcHigh = 0.52, so the interval is the single value (0.52).

For the case of 6/30/2003 the problem is more complex. What do we know from the data? We want to "add" <0.01 and 0.3. What is that sum? It is clearly the interval (0.3, 0.31). So, in this case ConcLow = 0.3 and ConcHigh = 0.31. This is where EGRET departs from the convention used in other methods such as LOADEST or Estimator. They would consider the fact that there is some censoring in this observation and consider it to be a left-censored result, which is the interval (0.0, 0.31). This is because these systems do not allow for the use of interval censoring like EGRET does. There is a very substantial difference in these two ways to characterize this sample: EGRET recognizes it as a range, but really a very narrow one that is virtually indistinguishable statistically from this being consider a point value such as 0.31 or 0.3. This suggests (correctly) that we have very little uncertainty about the true value and the calculations treat it appropriately. The other systems see this as the interval (0, 0.31) which is a very broad range.

Finally consider the case of 9/15/2004 where the values are <0.005 and <0.2. In this case ConcLow = NA and ConcHigh = 0.205, or an interval (0, 0.205).

In summary, EGRET treats every water quality observation as if it were interval censored. In the uncensored cases the interval collapses to zero width (ConcLow = ConcHigh). In the conventional left censored case the interval runs from zero to the reporting limit and ConcLow is defined to be NA and ConcHigh is the reporting limit. And in the case where at least one of the summed analytes is a censored and at least one is uncensored, ConcLow is set to the sum of the uncensored analytes and ConcHigh is set to the sum of the reporting limits for the censored and reported values for the uncensored cases. The estimation algorithm always uses survival regression, with all of the observations treated as interval censored.

8 Guidance about running batch jobs.

When running a large number of analyses the EGRET code can be run in batch mode. The way this is done is to create a text file that contains the sequence of commands that are to be run and this file can then be run using the `source` function in R. I would suggest that the process of creating the initial workspace (the data frames `Sample`, `Daily`, and `INFO`) are best done interactively, although they can be done in batch. I say that because of the potential for surprising or erroneous data values or other data problem. Running the process through the step of `multiPlotDataOverview` gives the analyst several opportunities to spot problems with the data set that need to be explored and corrected.

Let's assume that we have created two workspaces: call them `CHOP.TN.RData` and `SUSQ.TN.RData` (the first being for Choptank River total nitrogen, the second being for Susquehanna River total nitrogen) and these data frames have the Sample, Daily, and INFO data frames in them and everything is ready for running model estimation and looking at results. I'm going to assume that the user has a directory where these workspaces are saved and the object `savePath` has been defined with a command like:

```
savePath<-"/Users/rhirsch/Desktop/examples/"
```

Here is what the file of commands might look like:

```
library(EGRET)
fileName<-paste(savePath,"CHOP.TN.RData",sep="")
load(fileName)
modelEstimation()
AnnualResults<-setupYears()
tableChange(fluxUnit=5,yearPoints=c(1980,1990,2000,2010)
tableResults(qUnit=1,fluxUnit=5)
saveResults(savePath)
fileName<-paste(savePath,"SUSQ.TN.RData",sep="")
load(fileName)
modelEstimation()
AnnualResults<-setupYears()
tableChange(fluxUnit=6,yearPoints=c(1980,1990,2000,2010)
tableResults(qUnit=3,fluxUnit=6)
saveResults(savePath)
```

This set of code will load the workspace, run the WRTDS analysis, provide tabular output (to the console) in the form of changes and listings of the data, and then re-save the workspaces with them augmented with the results of the analysis. Note that some of the options in the table commands are different because the rivers are of very different sizes so we might want to see flux in tons/year for the Choptank and thousands of tons per year for the Susquehanna. Later on, the workspaces that were saved can be reloaded and a variety of plots can be made. Note that most of the batch commands are just a repetition of steps and only a few differences need to be entered to make up a

long file of many analyses. Then, if we name this file "exampleCommands.txt" then all we need to do once we start up R is to give the one command:

```
source(" full pathname goes here/exampleCommands.txt")
```

In future versions of EGRET there will be a capability to produce all of the tables and graphics as named files so the batch job can even further automate the process.

Appendix A: An alphabetical list of all EGRET functions

blankTime	deletes the computed values during periods of time when there is no sample data
boxConcMonth	Box plot of the water quality data by month
boxQTwice	Two box plots side-by-side, discharge on sample days, and discharge on all days
boxResidMonth	A box plot of WRTDS residuals by month
calculateMonthlyResults	calculates monthly values of Discharge, Concentration, Flux, Flow Normalized Concentration and Flow Normalized Flux for the entire record
estCrossVal	Jack-Knife cross validation of the WRTDS (Weighted Regressions on Time, Discharge, and Season)
estDailyFromSurfaces	Estimates all daily values of Concentration, Flux, Flow Normalized Concentration, and Flow Normalized Flux
estDailyWithoutNormalization	Estimates all daily values of Concentration, and Flux
estSurfaces	Estimating three surfaces (`yHat`, `SE` and `ConcHat`) as functions of `DecYear` & `logQ` and store in "surfaces"
flowDuration	Computes several values of the flow duration curve for streamflow centered on a specific date of the year
fluxBiasMulti	Produces a 6-panel plot that is useful for determining if there is a flux bias problem
fluxBiasStat	Compute the flux bias statistic: (mean of estimated flux - mean of observed flux) / mean of observed flux
getJulian	A utility to determine the Julian day for any given date
logPretty1	Sets up tick marks for a log-scale axis, for a small graph
logPretty3	Sets up tick marks for log-scale axis
makeAnnualSeries	Produces annual series of 8 streamflow statistics (and a lowess smooth of them) from daily streamflow data
modelEstimation	estimation process for the WRTDS (Weighted Regressions on Time, Discharge, and Season)
multiPlotDataOverview	Produces a 4 panel plot that gives an overview of the data set prior to any processing
plot15	Makes 15 graphs of streamflow statistics on a single page
plot1of15	Plots one of the 15 graphs of streamflow statistics that will be plotted by plot15
plotConcHist	Annual concentration and flow normalized concentration versus year
plotConcPred	Plot of Observed Concentration versus Estimated Concentration

plotConcQ	Plot of Observed Concentration versus Discharge
plotConcQSmooth	plot up to three curves representing the concentration versus discharge relationship, each curve is a different point in time
plotConcTime	Plot of Observed Concentration versus Time
plotConcTimeDaily	plot of the time series of daily concentration estimates and the sample values for the days that were sampled
plotContours	color contour plot of the estimated surfaces as a function of discharge and time (surfaces include log concentration, standard error, and concentration)
plotDiffContours	Plots the difference between two years from a contour plot created by `plotContours`
plotFlowSingle	Creates a plot of a time series of a particular flow statistic and a lowess smooth of that flow statistic
plotFluxHist	Annual flux and flow normalized flux versus year
plotFluxPred	Observed versus estimated flux
plotFluxTimeDaily	plot of the time series of daily flux estimates and the sample values for the days that were sampled
plotFour	Makes four graphs of annual streamflow statistics on a single page (1-day max, mean, 7-day min, and running standard deviation of log(Q)
plotFourStats	Makes four graphs of annual streamflow statistics on a single page (1-day max, mean, median, and 7-day min)
plotLogConcPred	Observed versus estimated concentration as a log-log graph
plotLogConcQ	Sample data plot: concentration vs. discharge (log/log)
plotLogConcQSmooth	plot up to three curves representing the log concentration versus discharge relationship, each curve is a different point in time
plotLogConcTime	Sample data plot: time vs. log of concentration
plotLogFluxPred	Sample data plot: predicted log flux vs observed log flux
plotLogFluxQ	Sample data plot: observed log flux vs log discharge
plotQTimeDaily	Plot the discharge time series, particularly useful for looking at days above some threshold
plotResidPred	plot of the residuals from WRTDS versus the estimated values (all in log concentration units)
plotResidQ	plot of the residuals from WRTDS (in log concentration units) versus the discharge
plotResidTime	plot of the residuals from WRTDS (in log concentration units) versus time
plotSDLogQ	Plot of the standard deviation of the log of discharge versus time
printFluxUnitCheatSheet	Available Flux Unit properties
printqUnitCheatSheet	Available Flow Unit properties
printSeries	Prints annual results for a given streamflow statistic

runSurvReg	Run the weighted survival regression for a set of estimation points (defined by `DecYear` and Log(Q))
saveResults	A utility program for saving the contents of the workspace
setPA	Sets up the period of analysis to be used in `makeAnnualSeries`
setPAx	Special version of `setPA`, to be used in the function `plot15`
setSeasonLabel	creates a character string that describes the period of analysis, when period of analysis has already been set in `AnnualResults`
setSeasonLabelByUser	creates a character string that describes the period of analysis, when the period of analysis is being set by the user and not from `AnnualResults`
setupYears	creates the `AnnualResults` data frame from the Daily data frame
surfaceIndex	this function computes the 6 parameters needed to lay out the grid for the surfaces computed in `estSurfaces`
tableChange	create a table of the changes in flow-normalized values between various points in time in the record
tableFlowChange	Prints table of change metrics for a given streamflow statistic
tableResults	Table of annual results for discharge, concentration and flux
triCube	Tukey's Tricube weight function
yPretty	Sets up tick marks for an axis for a graph with an arithmetic scale which starts at zero

Appendix B: A list of all functions organized by functional groups

[not done yet]

Appendix C: A list of arguments that are frequently used across several functions, and explanation of their meaning.

[not done yet]

Appendix D: A list of the units that can be used for output.

These are discharge units, and flux units.

The following codes apply to the qUnit list:

1 = cfs (Cubic Feet per Second)
2 = cms (Cubic Meters per Second)
3 = thousandCfs (Thousand Cubic Feet per Second)
4 = thousandCms (Thousand Cubic Meters per Second)
5 = mmDay (mm per day)
6 = mmYear (mm per year)

The following codes apply to the fluxUnit list:

1 = poundsDay (pounds/day)
2 = tonsDay (tons/day)
3 = kgDay (kg/day)
4 = thousandKgDay (thousands of kg/day)
5 = tonsYear (tons/year)
6 = thousandTonsYear (thousands of tons/year)
7 = millionTonsYear (millions of tons/year)
8 = thousandKgYear (thousands of kg/year)
9 = millionKgYear (millions of kg/year)
10 = billionKgYear (billions of kg/year)
11 = thousandTonsDay (thousands of tons/day)
12 = millionKgDay (millions of kg/day)

Appendix E: The smoothing method used in flowHistory

{This will be slightly modified in the next version - this write up has a few specifics to a particular application and are not as general as what is in EGRET, but the principles are all here}. The analysis of long-term variation in streamflow characteristics used in this study builds on time-series smoothing methods that were pioneered by Cleveland (1979) and Cleveland and Devlin (1988). For any given time series, the graphs included in the appendix of this report show a scatter plot of runoff (Q_i) as a function of time (T_i), for i from one to n, where Q_i is the i^{th} annual value of the streamflow statistic, expressed in mm/day, and T_i is the time value at the mid-point of the time period over which the statistic is evaluated, expressed in years. For example, for a flow statistic computed for water year 1972, the T_i value is approximately 1972.25, which represents the decimal-year value of the half-way point in the 1972 water year.

In addition to showing the actual values of the annual or seasonal streamflow statistic, the graphs show a curve that represents a smoothed representation of the time series. The smoothing method used is based on locally weighted scatterplot smoothing (lowess) but with some particular features that are described below. The purpose of producing the smooth curves is an attempt to extract patterns of change that describe broad temporal-scale variations, at time spans of about a decade or more. Such curves are very resistant to the influence of one or two years with extremely high or low flows.

The term y_i is the log-transformed value of the flow statistic:

$$y_i = \ln(Q_i),$$

where ln is the natural logarithm function. If the flow statistic is equal to zero, Q_i is replaced with a constant, set equal to 0.1% of the long-term daily mean discharge for the station. The logarithm transformation was applied because streamflow data typically are highly skewed, approximating a log-normal distribution in many cases. The logarithm transformation results in weighted regressions in which the residuals are more nearly normal and thus, individual extreme values do not exert a large amount of influence on the estimates. This results in a more robust smoothing process. It also means that the $\hat{Q}_i$ values are more nearly an approximation of the median of the time series than they are an approximation of the mean [see Helsel and Hirsch (1992), pages 254-260 for a discussion of transformation issues].

In log-space, the smooth curve is defined by a series of n-weighted regressions on the data set. The estimate, $\hat{y}_i$, of y_i is defined as

$$\hat{y}_i = \beta_{0i} + \beta_{1i} \bullet T_i \quad \text{for } i = 1, \text{n}$$

where β_{0i} is the estimated regression intercept for the regression model fitted for year i, and β_{1i} is the estimated regression slope for the regression model fitted for year i.

The two regression coefficients, β_{0i} and β_{1i}, were computed from a weighted regression, where the weights are equal to one for the observation for the year in which the estimate is being made, and decay to zero at a time separation of 30 years between a given observation and the time of the estimate. The specific weights are computed with the Tukey tri-cubed weight

function (Tukey, 1977). The weight for the i^{th} streamflow value in the computation of the smoothed value for the j^{th} year is:

$$w_{i,j} = \begin{cases} \left(1-\left(d_{i,j}/30\right)^3\right)^3 & if \left|d_{i,j}\right| \le 30 \\ 0 & if \left|d_{i,j}\right| \ge 30 \end{cases}$$

where,

$$d_{i,j} = T_i - T_j$$

The shape of the weight function is such that all of the observations between zero and 15 years before or after the year for which the estimate is being made have weights that are at least 67% as large as the largest weight. The largest weight is for the observation that is in the year for which the estimate is being made. For observations that are 25 years before or after the year for which the estimate is being made, the weight is only 7% of the maximum weight. At 30 years, the weight is zero. The "half-window width," in this case, 30 years, was selected by visual examination of graphics for many alternatives. The half-window width was selected to be as narrow as possible, such that individual year-to-year oscillations are fully damped out. The smoothing process takes place on the full discharge record, not just the record shown in the graphs, i.e., data prior to 1930, if available, were used to calculate the lowess line.

The final step in producing the graphic of the smoothed annual values is the retransformation:

$$\hat{Q}_i = \exp(\hat{y}_i).$$

Graphs showing time series with lowess trends for the 7-day minimum, mean, and 1-day maximum for annual and seasonal data for each station are included the appendix.

Appendix F: Sample workflows

All of the commands given here are generic. They should work on any computer that has access to the Internet. The only exception is the second to last command (shown in red) in each sample workflow. That is entirely specific to the particular computer being used.

This is a sample workflow for using WRTDS on the Choptank River at Greensboro MD, for Nitrate.

```
library(dataRetrieval)
library(EGRET)
sta<-"01491000"
param<-"00631"
StartDate<-"1979-09-01"
EndDate<-"2011-09-30"
Sample<-getSampleData(sta,param,StartDate,EndDate)
summary(Sample)
length(Sample$Date)
Sample<-removeDuplicates()
length(Sample$Date)
Daily<-getDVData(sta,"00060",StartDate,EndDate)
summary(Daily)
Sample<-mergeReport()
INFO<-getMetaData(sta,param)
INFO
multiPlotDataOverview()
modelEstimation()
AnnualResults<-setupYears()
plotConcHist(1975,2012)
plotFluxHist(1975,2012,fluxUnit=5)
tableResults(qUnit=1,fluxUnit=5)
tableChange(fluxUnit=5,yearPoints=c(1980,1995,2011))
fluxBiasMulti(qUnit=1,fluxUnit=1,moreTitle="WRTDS")
plotFluxTimeDaily(1998,2005)
plotFluxTimeDaily(2010,2011.75)
plotContours(1980,2012,5,1000,qUnit=1,contourLevels=seq(0,2,0.25))
plotDiffContours(1980,2011,5,1000,1,qUnit=1)
plotConcQSmooth("1984-04-01","1995-04-01","2008-04-01",30,2000,qUnit=1)
plotContours(1980,2012,5,1000,qUnit=1,whatSurface=2)
plotResidPred()
plotResidQ()
savePath<-"/Users/rhirsch/Desktop/"
saveResults(savePath)
```

This is a sample workflow for a flowHistory application for the Mississippi River at Keokuk, Iowa for the entire record.

```
> library(dataRetrieval)
library(dataRetrieval)
library(EGRET)
sta<-"05474500"
param<-"00060"
Daily<-getDVData(sta,param,"","")
summary(Daily)
INFO<-getMetaData(sta,param)
INFO<-setPA()
annualSeries<-makeAnnualSeries()
plotFlowSingle(istat=5,qUnit=4)
printSeries(istat=5,qUnit=4)
tableFlowChange(istat=5,yearPoints=c(1880,1930,1970,2010))
plotFourStats(qUnit=4)
INFO<-setPA(paStart=12,paLong=3)
annualSeries<-makeAnnualSeries()
plotFlowSingle(istat=1,qUnit=4)
plotFlowSingle(istat=7,qUnit=4)
tableFlowChange(istat=5,yearPoints=c(1880,1930,1970,2010))
plotFourStats(qUnit=4)
savePath<-"/Users/rhirsch/Desktop/"
saveResults(savePath)
```

Appendix G. Minor changes to the code: December, 2012.

None of them change anything about the computation of results in EGRET. They are all changes designed to provide increased flexibility and/or to simplify use of code. A brief explanation of these changes is provided here along with an indication of the page number on which the change has been inserted into the text of the manual. These changes are shown in green in this document. Our intention is that this will be the last set of changes to this version of the code. The next update will involve some significant repackaging and improvements in documentation, although we anticipate that users will find that from a users perspective the next version will be very similar to the present version, but commands will offer more options than they presently do. The main changes will be in packaging and documentation.

These are the changes in the order in which they appear in the manual.

`modelEstimation` (pages 40 and 41). The change made here is that the step `calculateMonthlyResults` is no longer a part of the process, because the workflows and output routines don't depend on monthly results. The capability to compute monthly values still exists, but the user has to specifically call for it using the function `calculateMonthlyResults` (see page 44).

`plotConcHist` and `plotFluxHist` (pages 46 and 48) have both been changed so that `yearStart` and `yearEnd` can both be set automatically. The most common situation is that the user will want the graphs to run from at or just before the first year of results to at or just after the last year of results. If that is the case, then the defaults will be sufficient. Thus, the call would typically look like this:

```
plotConcHist()
```
or
```
plotFluxHist()
```

There are cases where one wants to set the minimum and maximum years on the graphs to some common values (typically if many graphs are going to be shown together, but the data sets have different lengths). So, for example if we wanted to assure that the graph ran from 1975 to 2015 then the call would look like this:

```
plotConcHist(yearStart=1975, yearEnd=2015)
```

Note, that the self-scaling rules in EGRET will typically break the axis at 5 or 10 year increments, so

```
plotConcHist(yearStart=1977, yearEnd=2012)
```

would actually start at 1975 and end at 2015.

`plotFluxTimeDaily` has simply been enhanced so that the user can select the flux units for the plot. The argument is fluxUnit and its default value is 3 (which signifies kg/day). See pages 49 and 59.

`tableResult`, has been modified to provide the means to produce an object which contains the contents of the table that is printed to the console. This can be helpful in a large set of analyses when many tables are being produced. The object could then be output for use in another program such as Excel. See page 50.

`tableChangeSingle`, is identical to `tableChange`, except that it only produces output for concentration or only for flux, not both. This can be helpful in applications that may be only focused on concentration or only on flux. It also has the capability to produce an object for output, similar to `tableResult`. See page 53.

`plotContours` and `plotDiffContours`, in both of these functions the option is provided to delete the flow duration lines being plotted along with the concentration contours. In some cases, these flow duration lines can be a distraction to the audience, so they can be deleted if the user chooses. The default, in both cases is `flowDuration=TRUE`, so if the command is given without any mention of this, it will plot exactly as the previous version would have plotted it (with the flow duration lines). See pages 60 and 64.