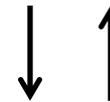
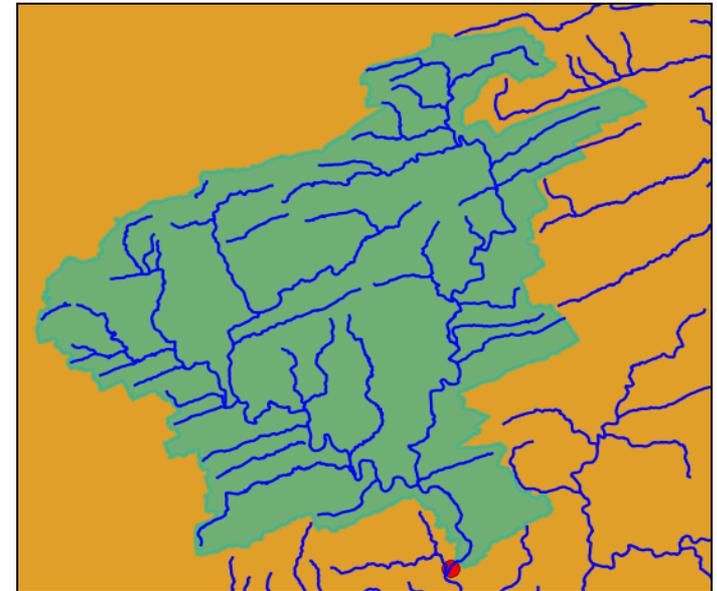
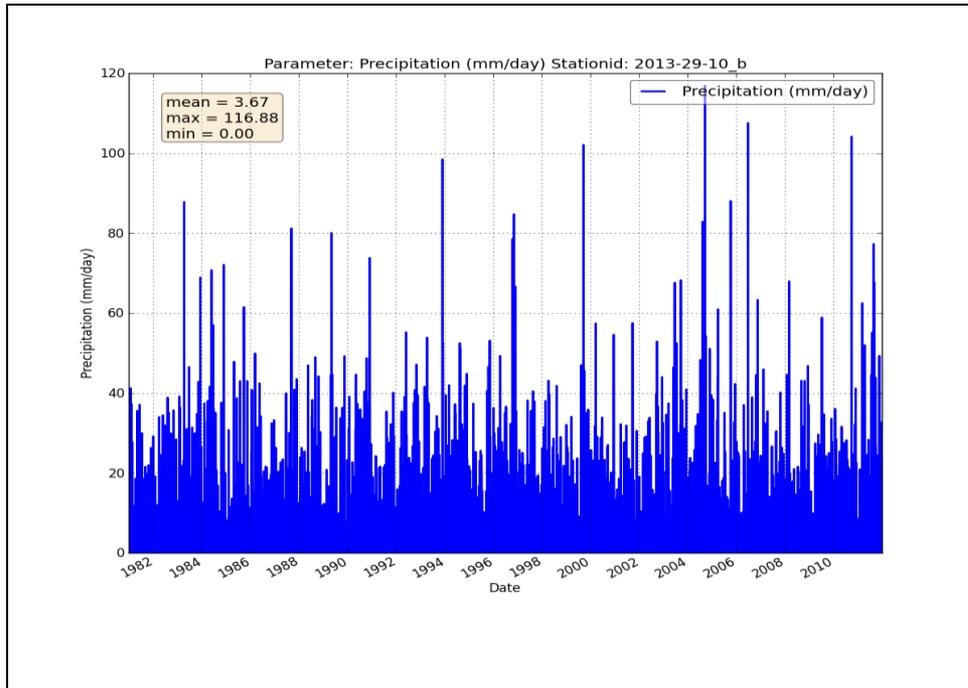
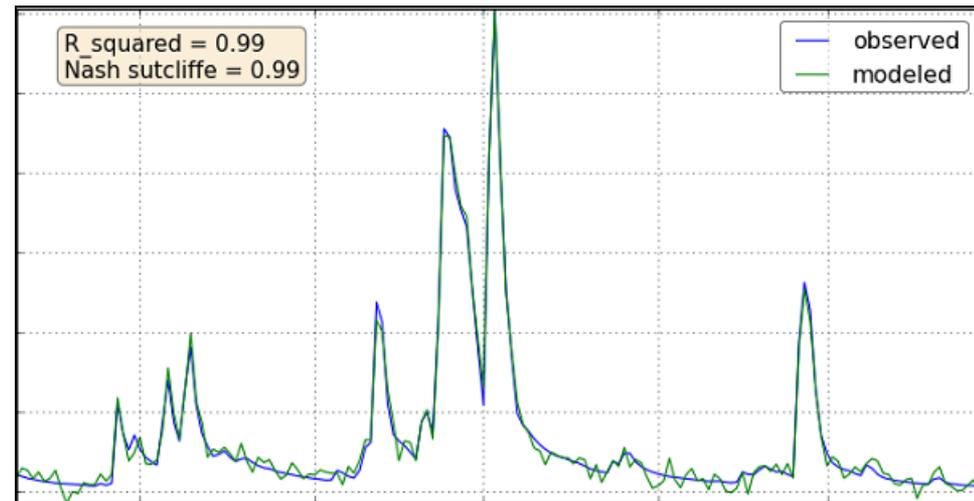


Current Hydrologic Modeling Projects and Tools



Jeremiah Lant, Hydrologist, USGS
Kentucky Water Science Center
jlant@usgs.gov



Outline

- Current hydrologic modeling projects
- Tools being developed
- Best practices for scientific computing

Acknowledgements

- The USGS Kentucky Water Science Center would like to thank the many cooperators that make this science possible.

Cumberland Gap National Historical Park, KY Project:

Ensuring Success and Management Effectiveness for the Imperiled Blackside Dace at Cumberland Gap National Historical Park: Sediment Acquisition and Modeling of Davis Branch



Cooperating Agencies and Period of Investigation

- Cooperating Agencies:
 - National Park Service, Cumberland Gap National Historical Park
 - University of Kentucky, Biosystems and Agricultural Engineering Department
- Period of Investigation:
 - FY 2014 (second quarter) – FY 2015 (fourth quarter)

Problem Description

- **Davis Branch**, a stream within the boundaries of **Cumberland Gap National Historical Park**, is a designated Warm Water Aquatic Habitat, Primary Contact Recreation, and Outstanding State Resource Water.
- **Davis Branch** provides a critical habitat to the **federally-threatened Blackside Dace**, a small cyprinid fish endemic to the upper Cumberland River basin.
 - In recent decades, the populations of the Blackside Dace have declined throughout the Cumberland River basin due to loss of habitat, stream corridor degradation, and water-quality changes as a consequence of beaver dams within the watershed.
- **Davis Branch has been classified by U.S. Fish and Wildlife Service as a critical stream to the continued existence of the Blackside Dace.** The National Park Service Organic Act and the Endangered Species Act are requiring resource managers at the Cumberland Gap National Historical Park (CUGA) to take actions needed to mitigate the potential loss of the federally threatened species.

Project Objectives

- **Collect** sufficient **baseline data** about sediment loads in Davis Branch.
- **Create a numerical modeling tool** to help predict sediment deposition and transport variability in Davis Branch under existing conditions and during and after the implementation of to-be determined stream restoration measures.
 - Provide field and modeled data needed to aid NPS resource managers to prepare an environmental assessment (ESA), plan stream restoration measures, and better assess the potential effectiveness of those restoration measures, thereby better ensuring that habitat conditions are suitable for the repopulation and long-term survivability of Blackside Dace in Davis Branch

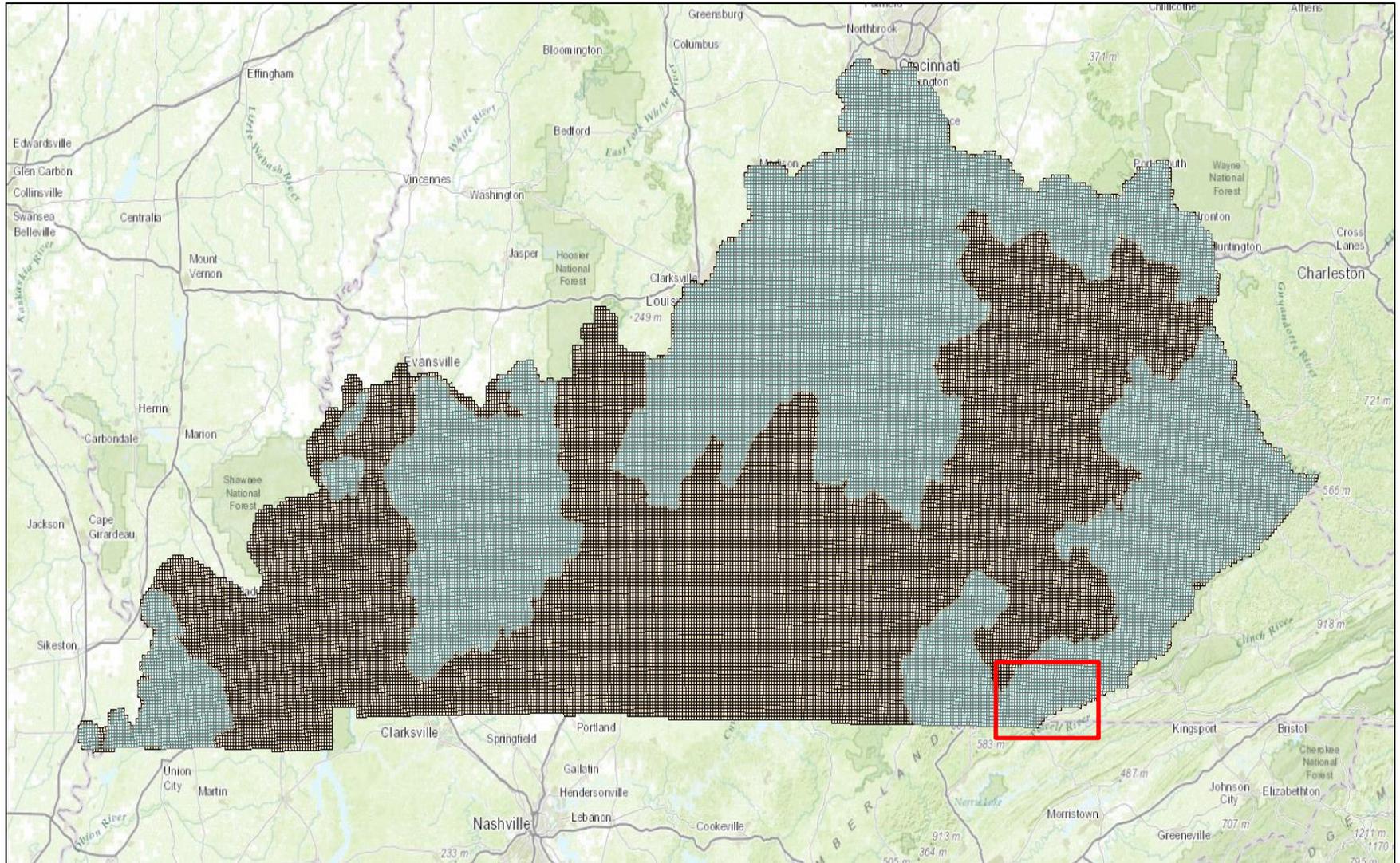
Approach

- **Collect data to characterize suspended sediment concentrations** and loads, determine sediment fractions, and to support modeling of sediment deposition and transport characteristics before, during, and after restoration measures
- **Apply a physically based catchment sediment deposition and transport model (SEDCAD), developed by Richard Warner (Professor, University of Kentucky),** needed to estimate sediment concentration, particle size distribution, sediment loads, and transport characteristics as part of the process of selection, design, and implementation of stream restoration measures.
- **Combine the functionality of the sediment and transport model (SEDCAD) with a rainfall-runoff model, Topmodel (Beven, 1995),** to develop new and innovative techniques for predicting and analyzing deposition and transport of suspended sediment in ungagged and flashy streams such as those within the Cumberland Gap National Historical Park and elsewhere in the Cumberland Gap River basin.

Products

- A Scientific Investigations Report and/or equivalent peer-reviewed journal article that summarize sediment load/transport data, and fully describes the procedures and findings of the hydrologic and sediment transport modeling conducted for the project.

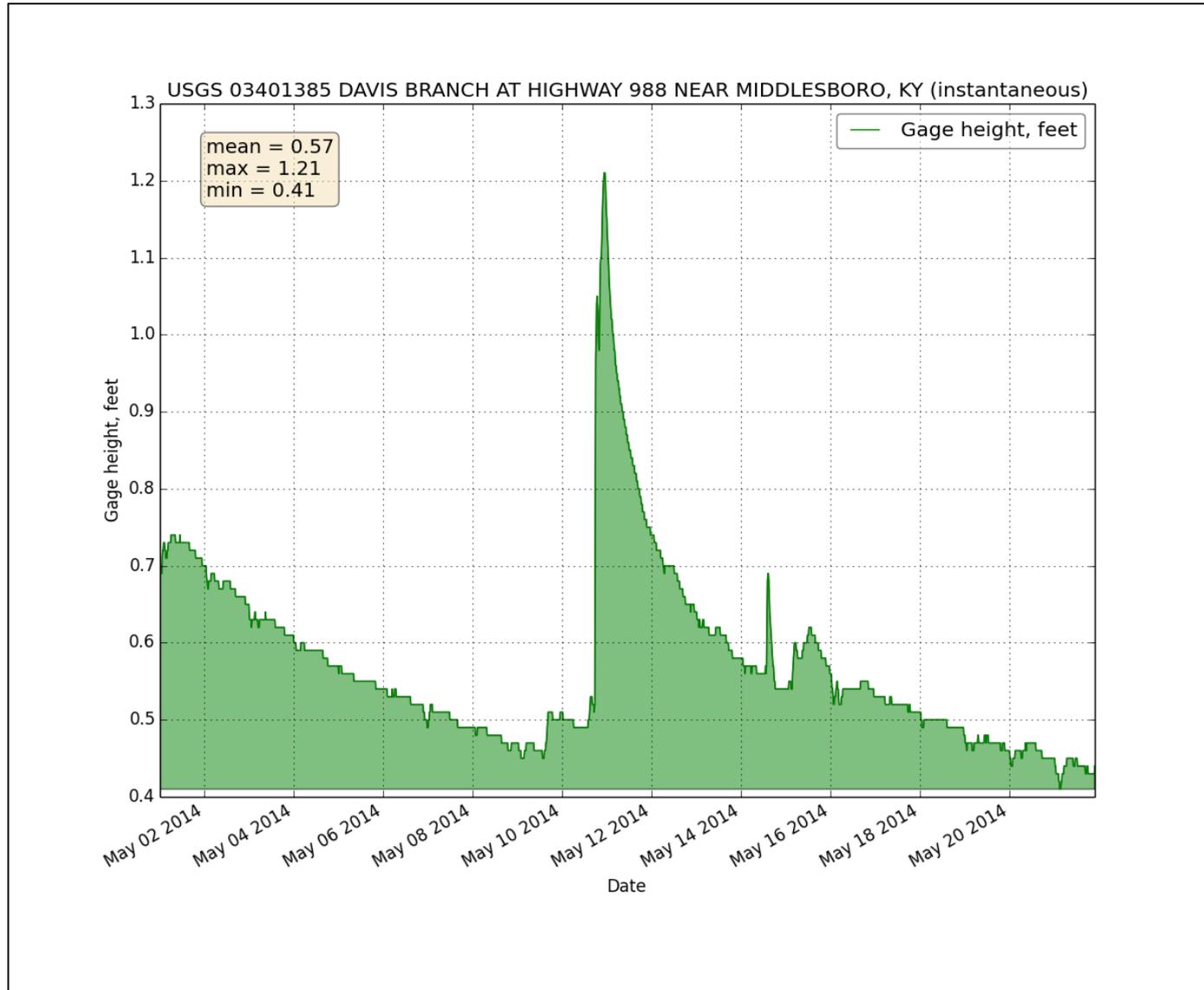
Datasets: Aerial imagery and DEM's



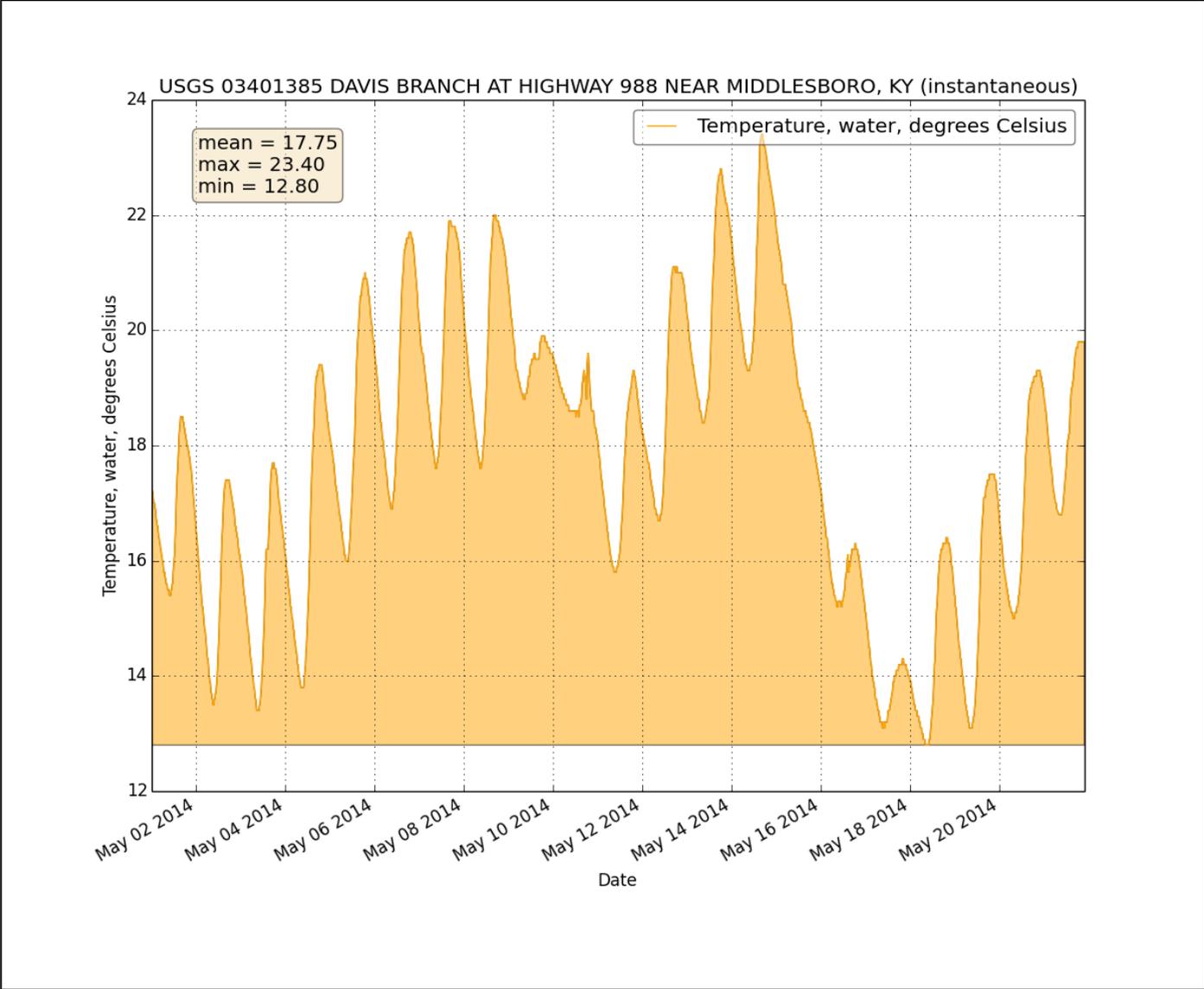
Datasets: USGS Gage 03401385

- Available Parameters:
 - Gage Height (feet)
 - Temperature (degrees Celsius)
 - Dissolved Oxygen (water, unfiltered, milligram)
 - pH (water, unfiltered, field, standard units)
 - Specific conductance (water, unfiltered, micro)
 - Turbidity (water, unfiltered, formazin nephelometric units, FNU)

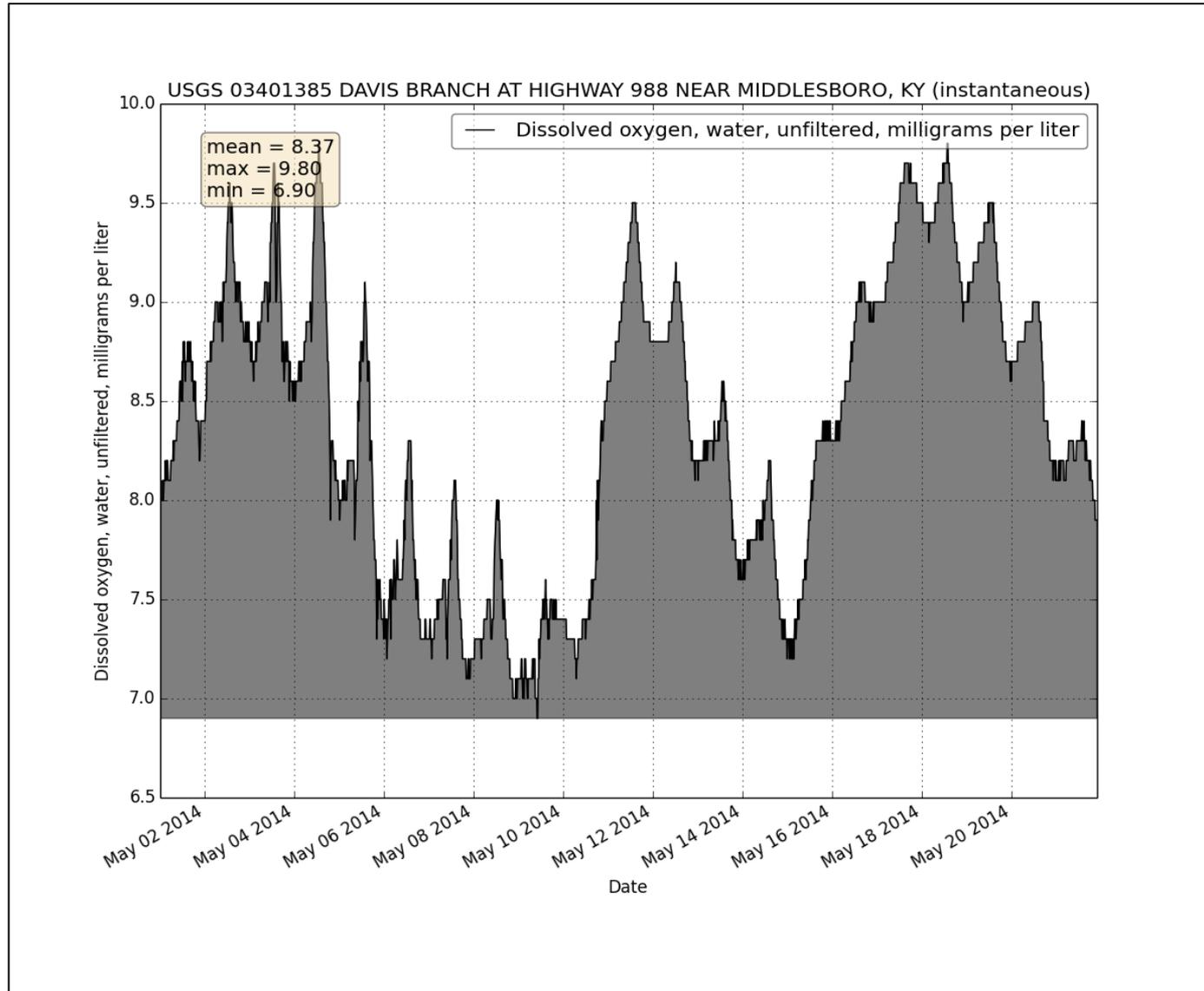
USGS Gage 03401385 – Gage Height



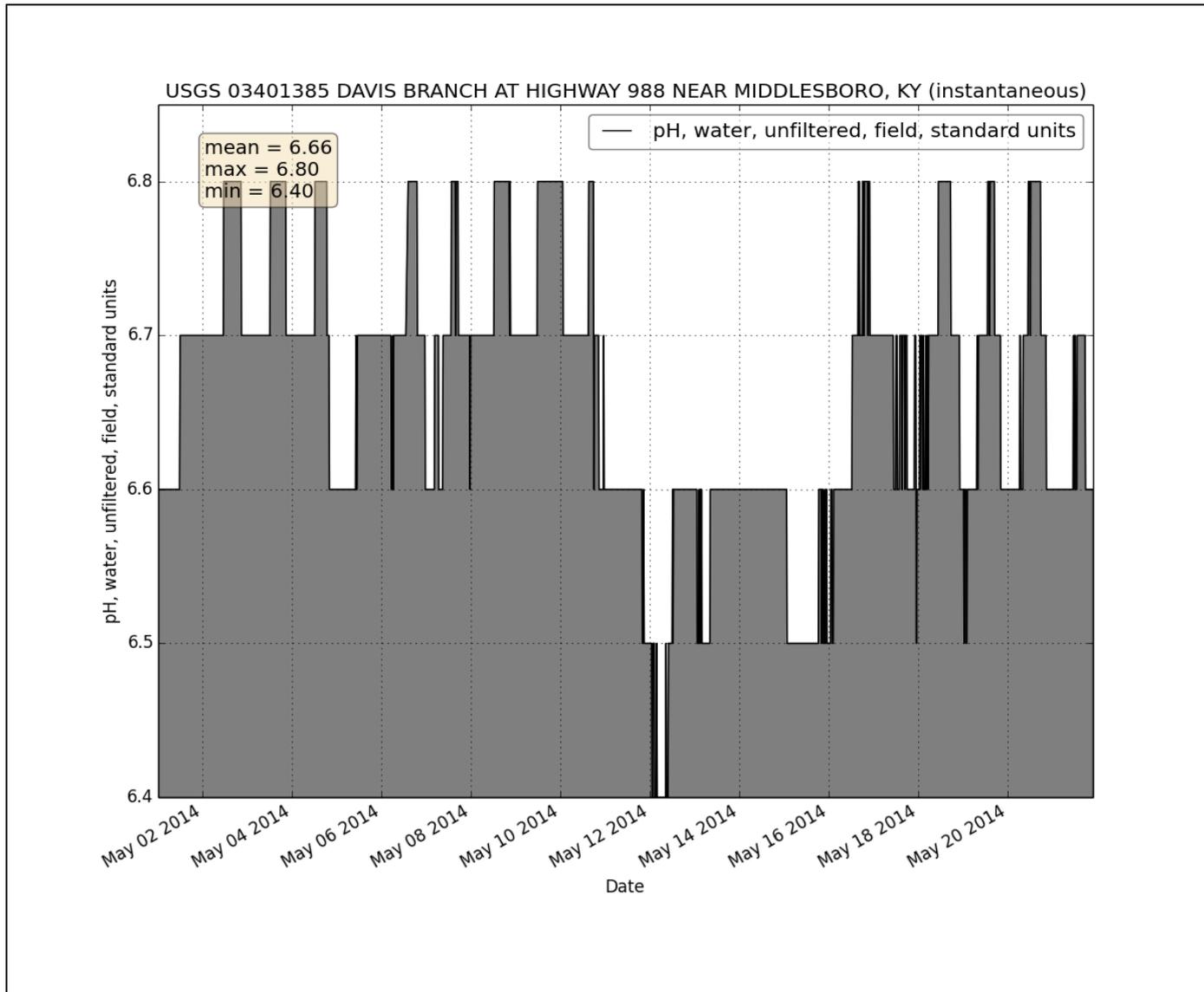
USGS Gage 03401385 – Temperature



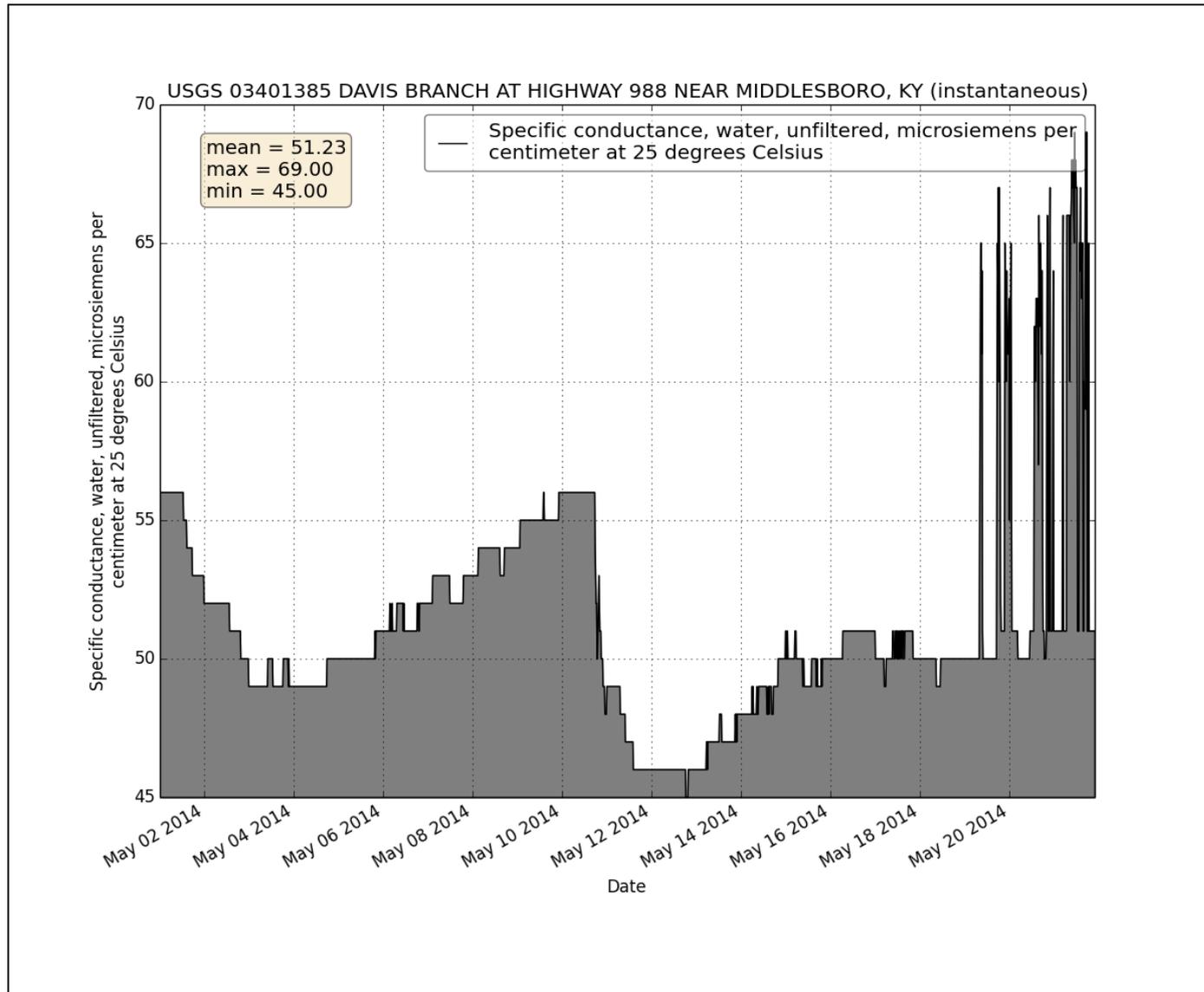
USGS Gage 03401385 – Dissolved Oxygen



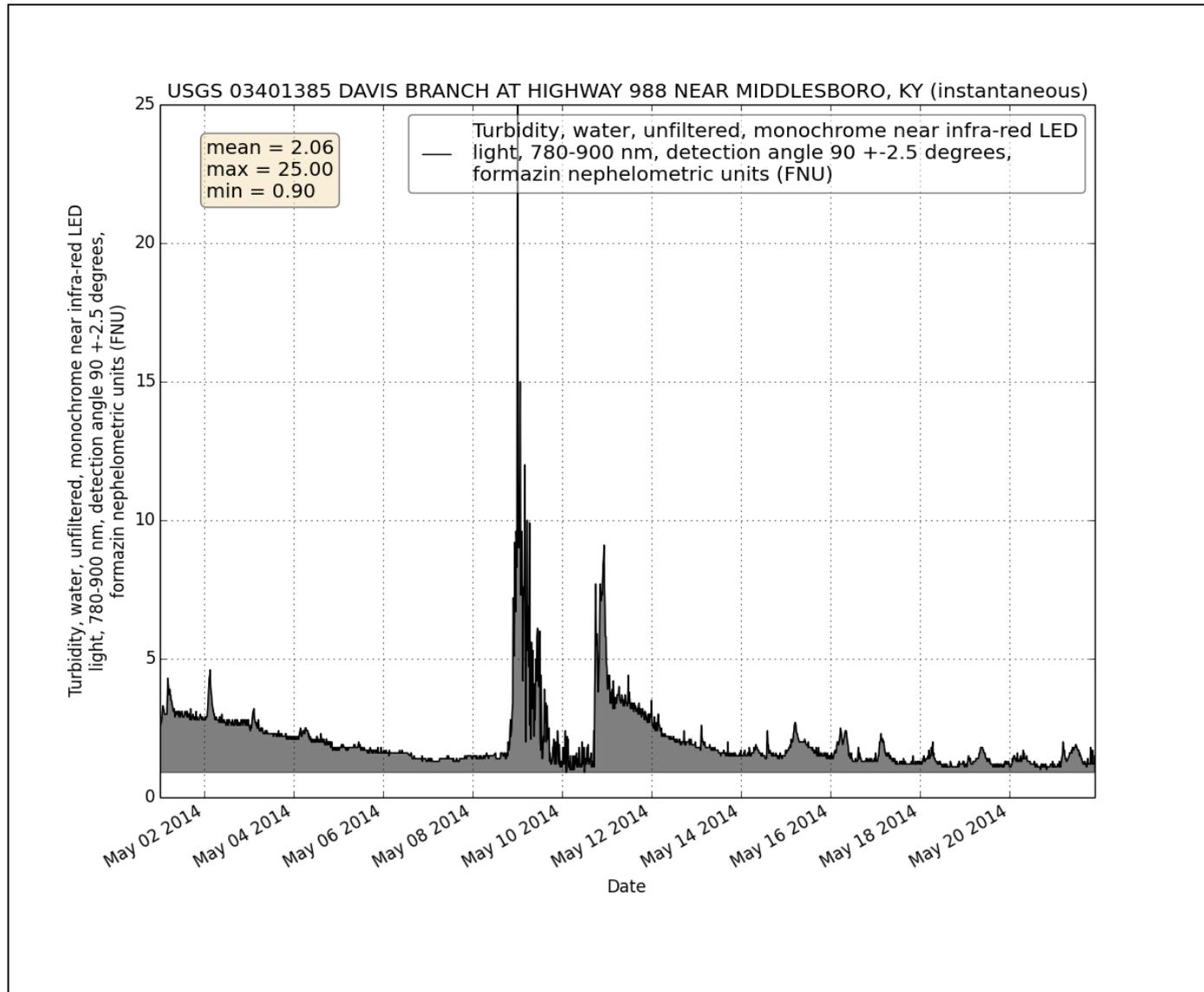
USGS Gage 03401385 – pH



USGS Gage 03401385 – Specific Conductance

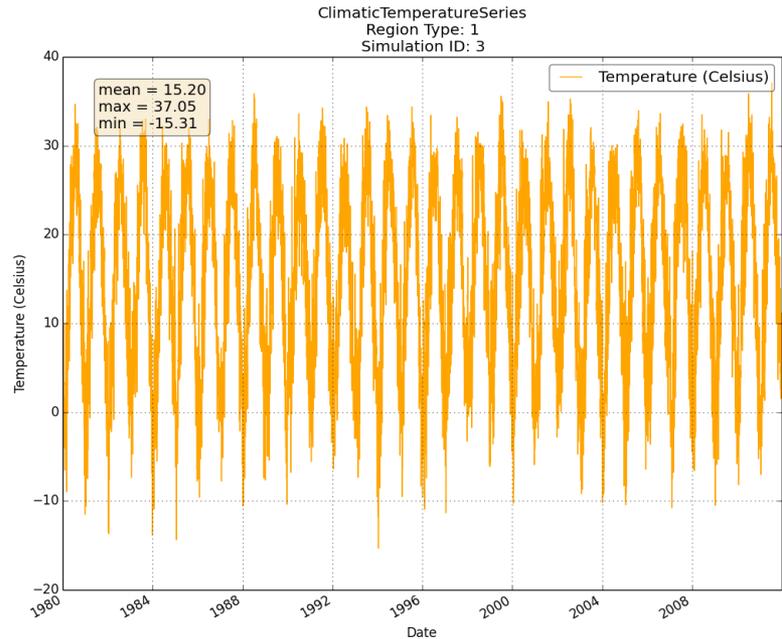
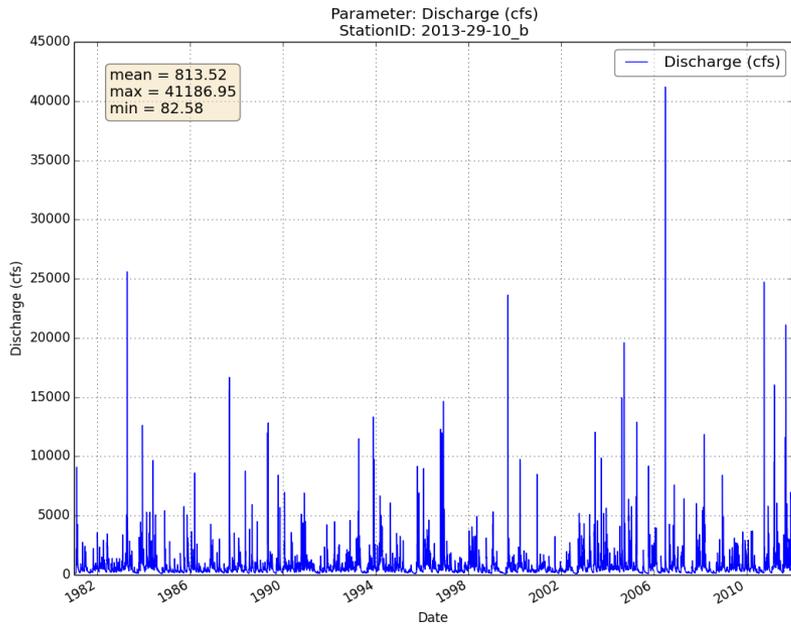
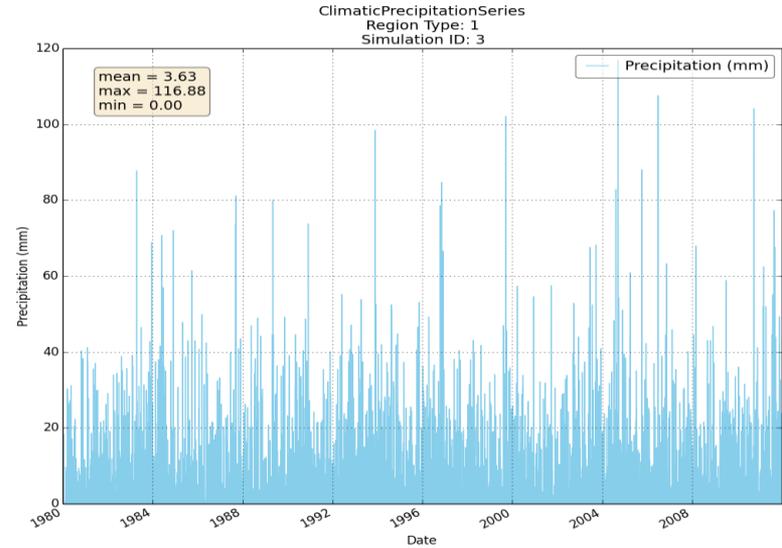
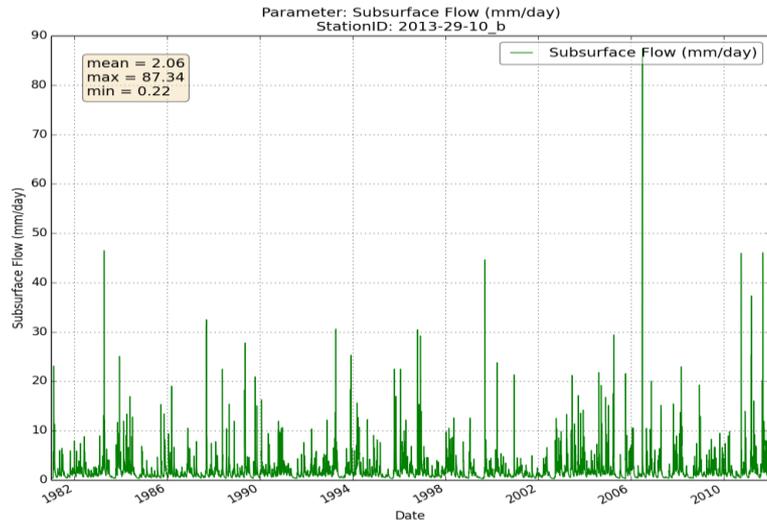


USGS Gage 03401385 – Turbidity



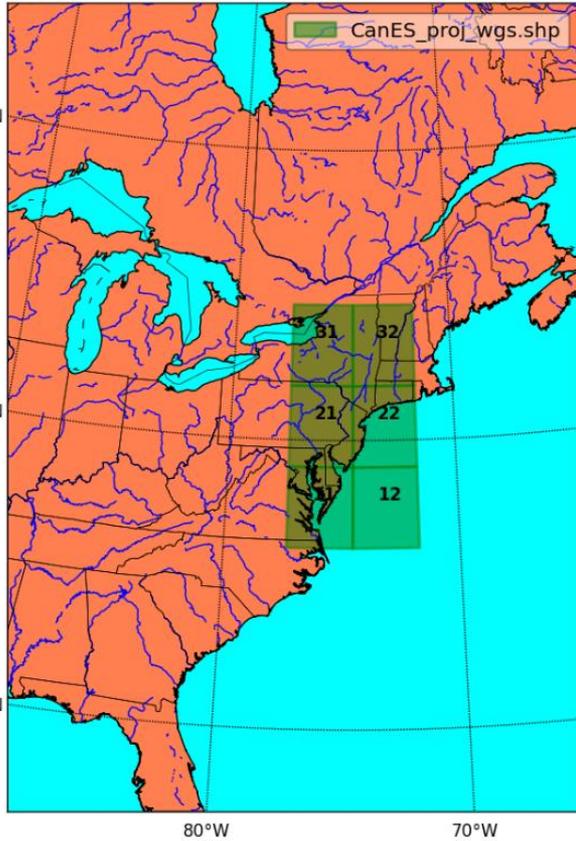
Delaware River Basin Hydrologic Modeling:

WATER Application *.txt, *.xml, editing capabilities

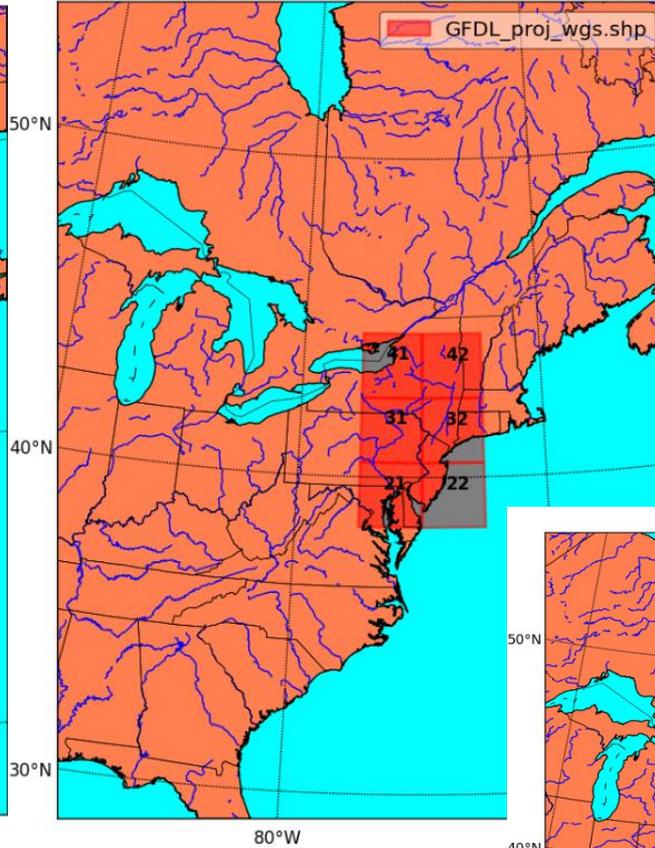


Delaware River Basin Hydrologic Modeling: Incorporating Global Climate Model Data

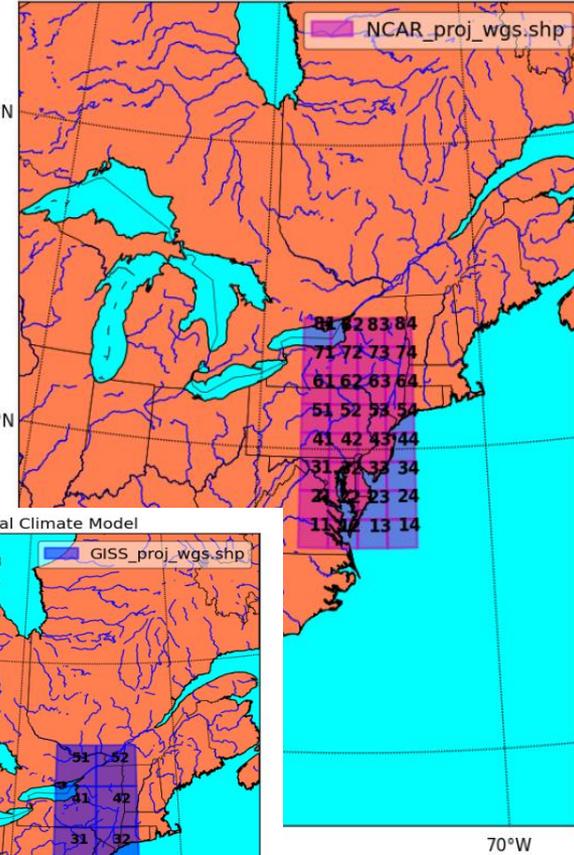
Global Climate Model



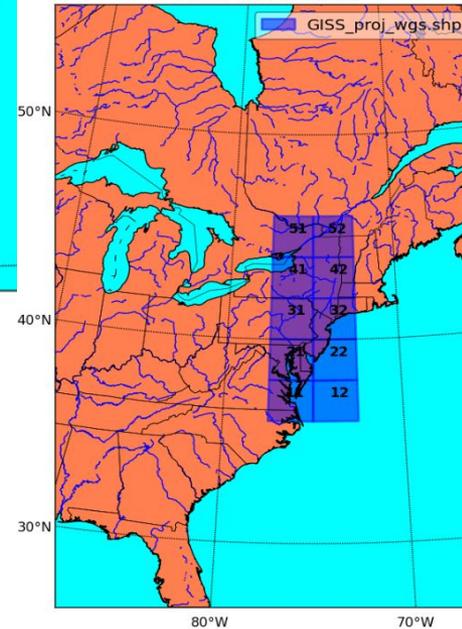
Global Climate Model



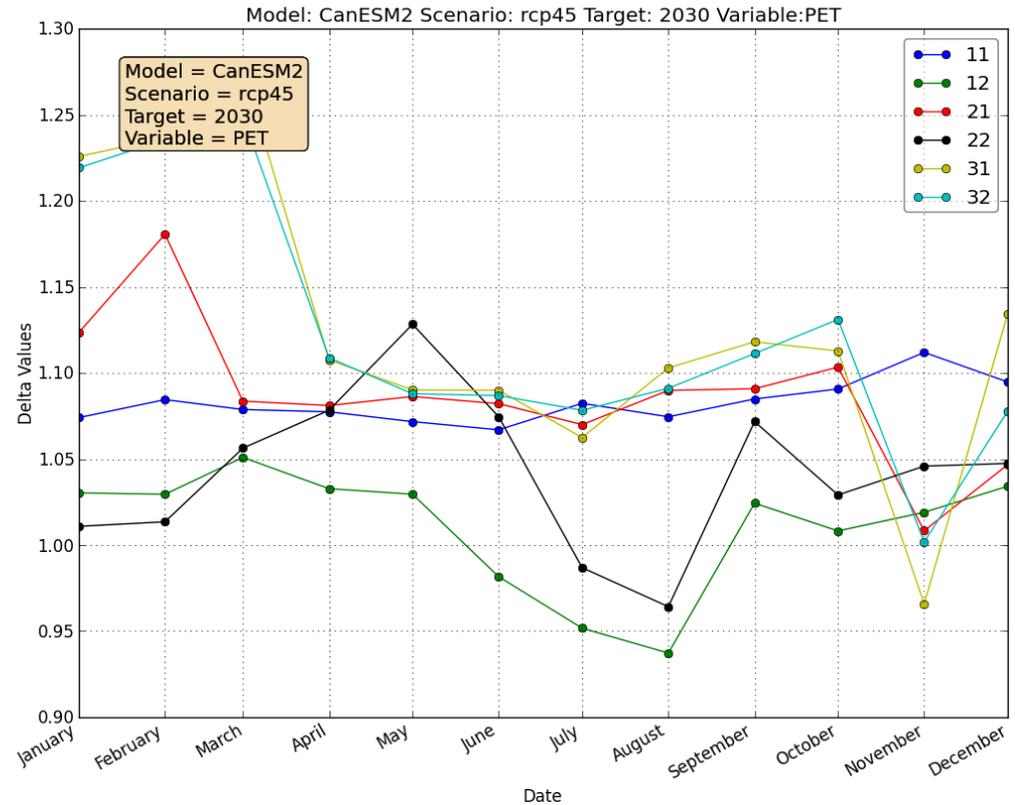
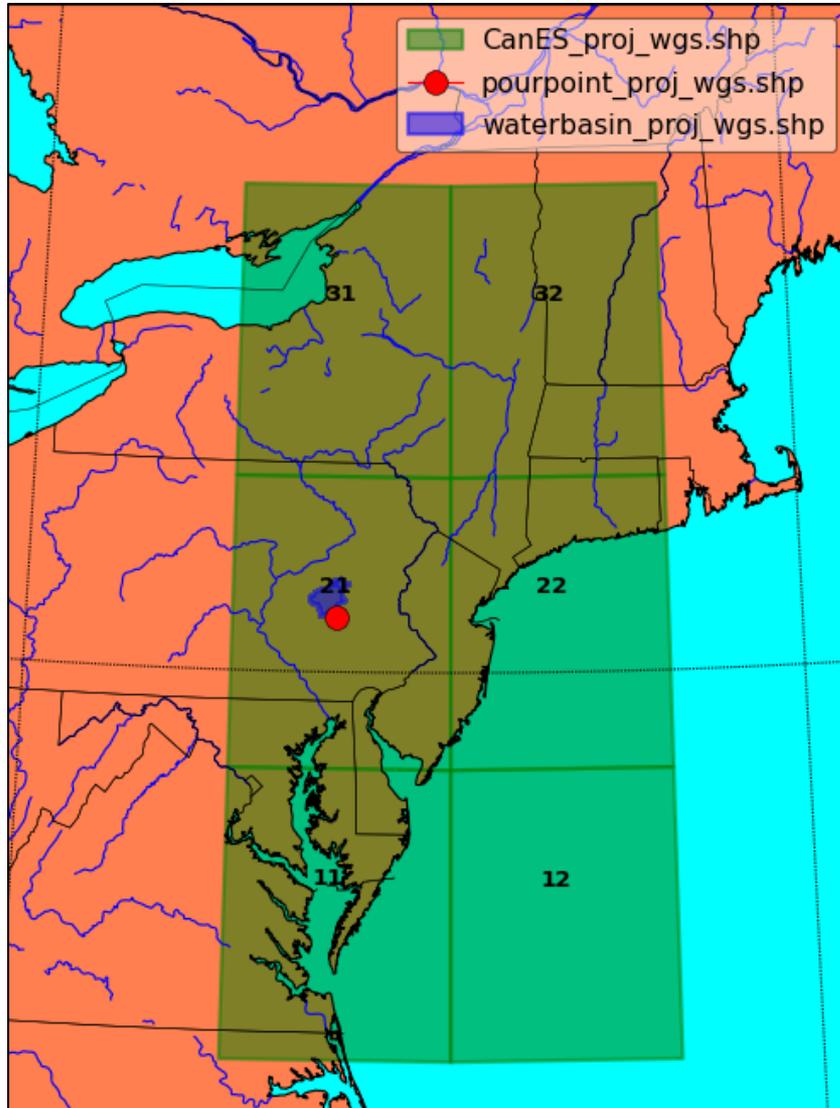
Global Climate Model



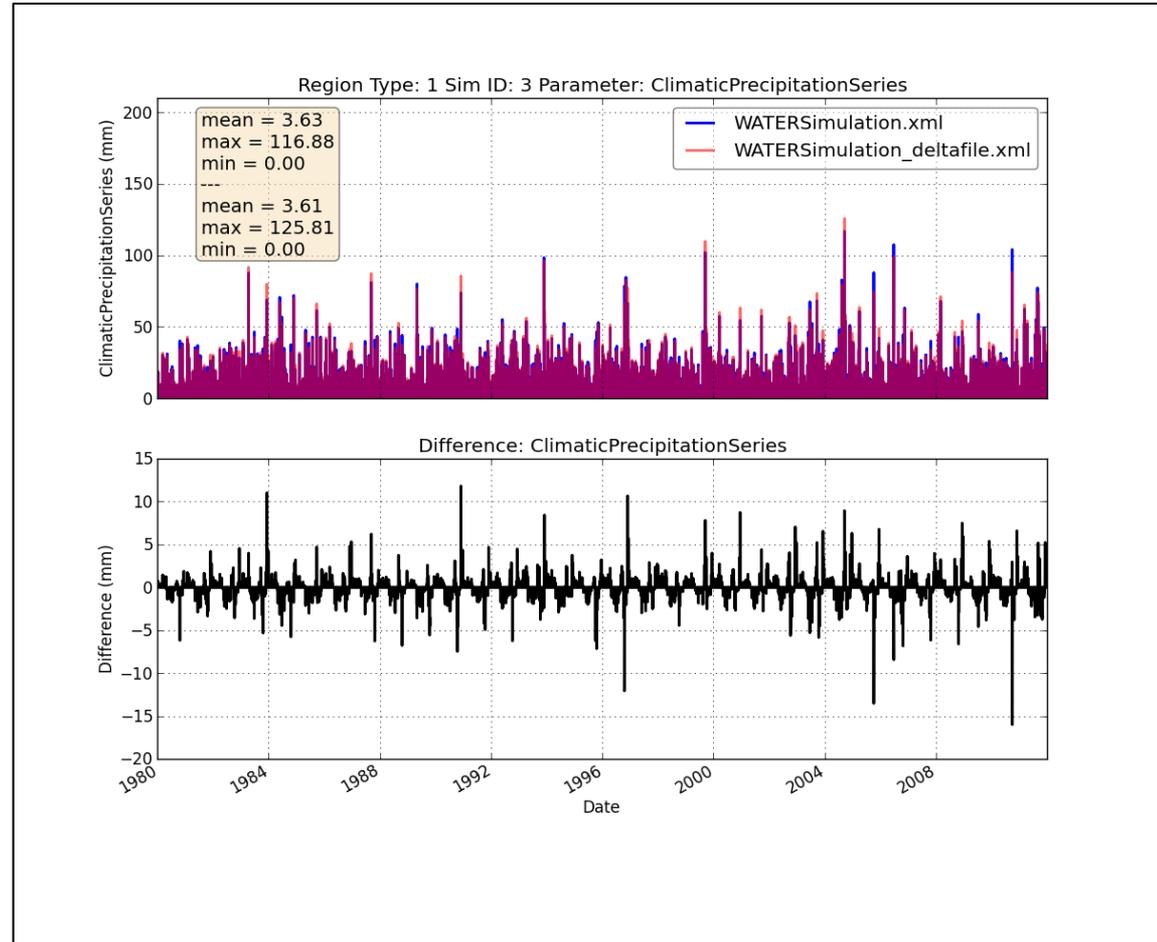
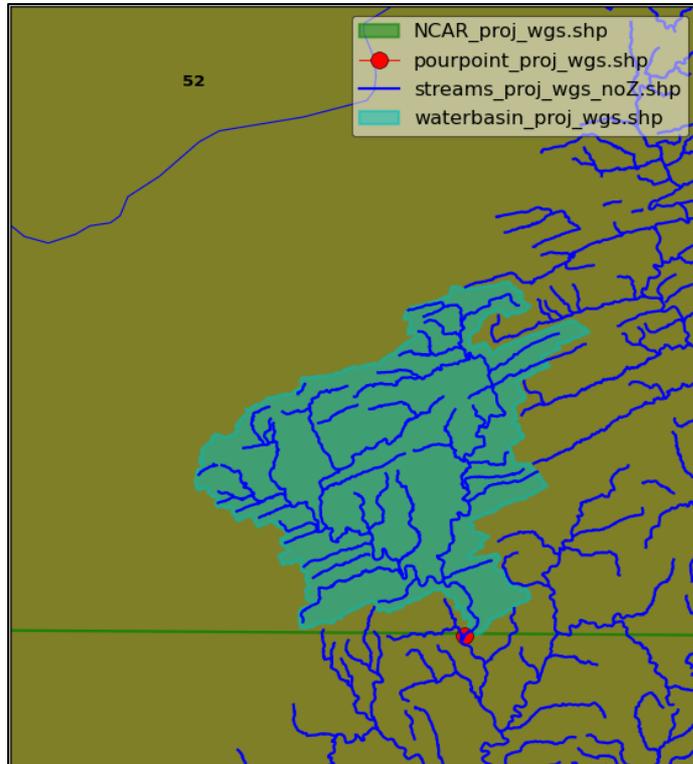
Global Climate Model



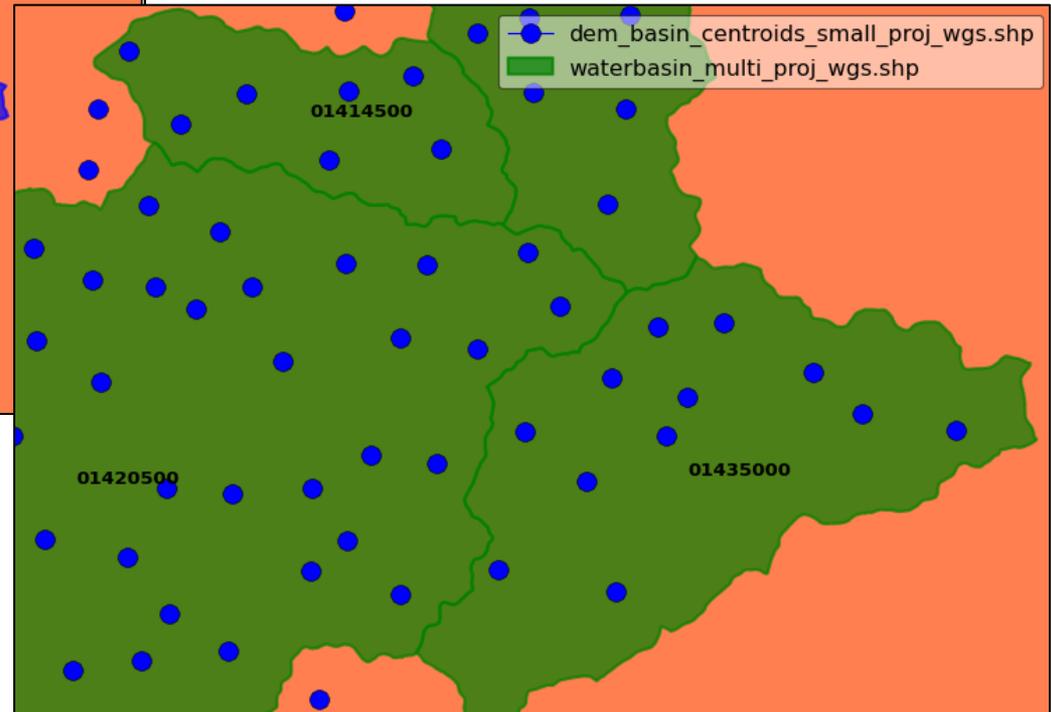
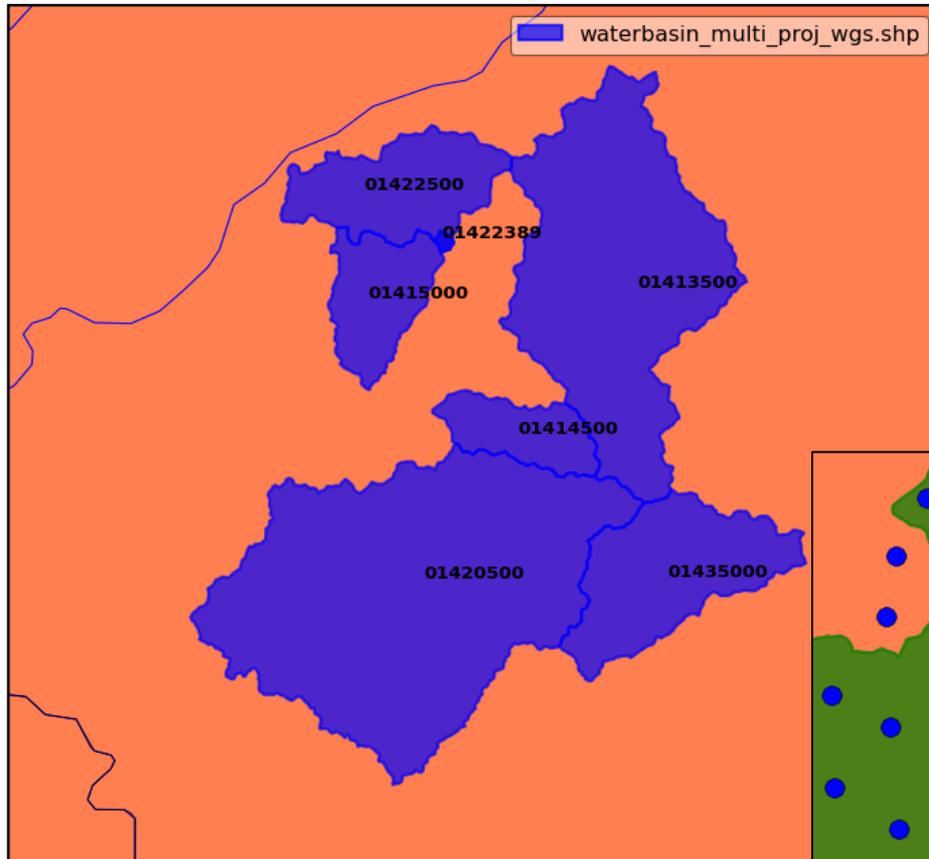
Delaware River Basin Hydrologic Modeling: Incorporating Global Climate Model Data



Delaware River Basin Hydrologic Modeling: Incorporating Global Climate Model Data



Delaware River Basin Hydrologic Modeling: Incorporating Water Use



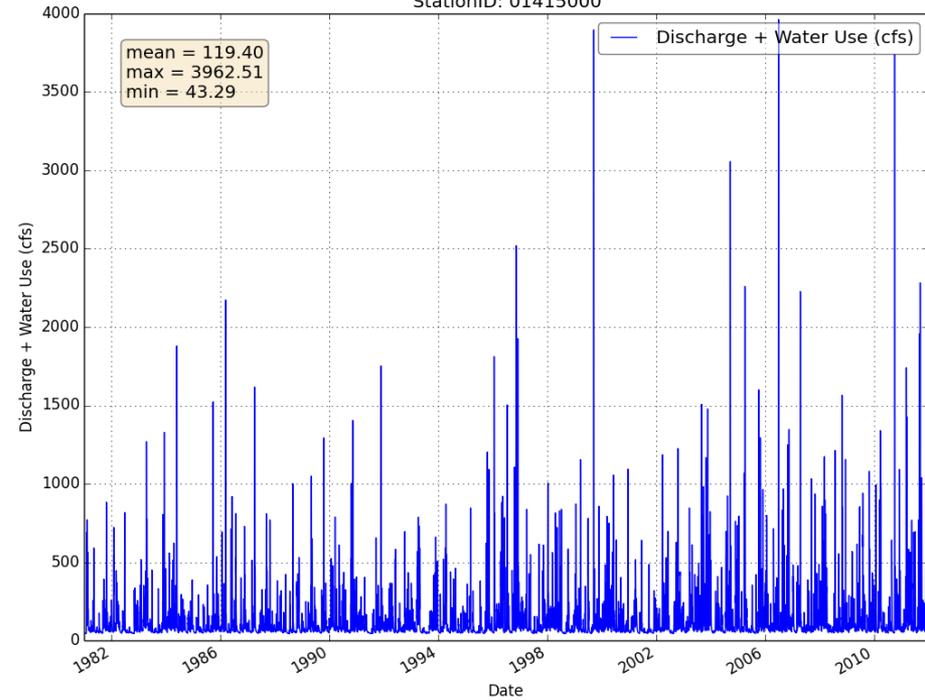
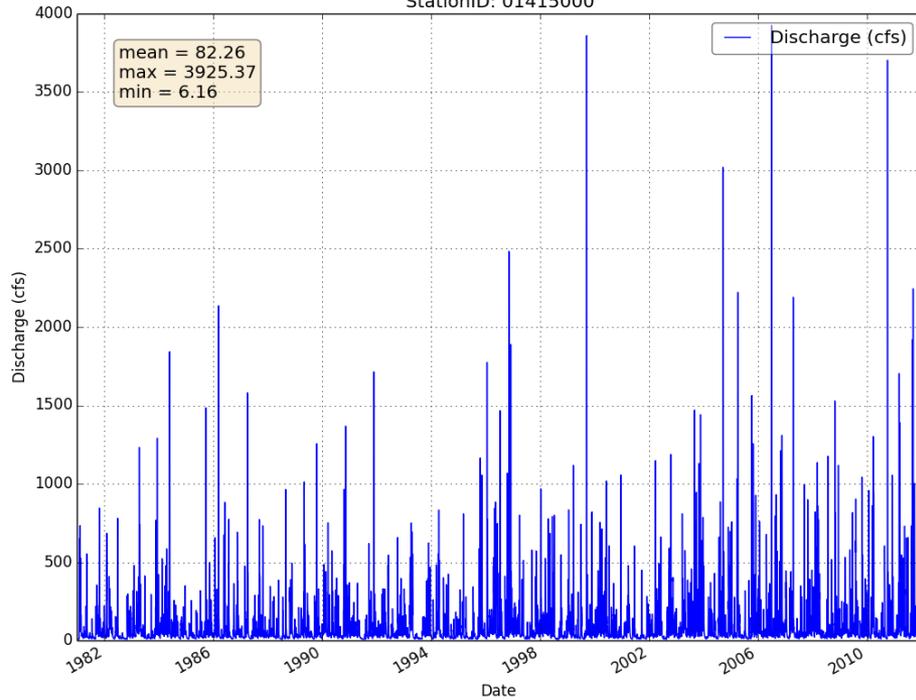
Delaware River Basin Hydrologic Modeling: Incorporating Water Use

Discharge

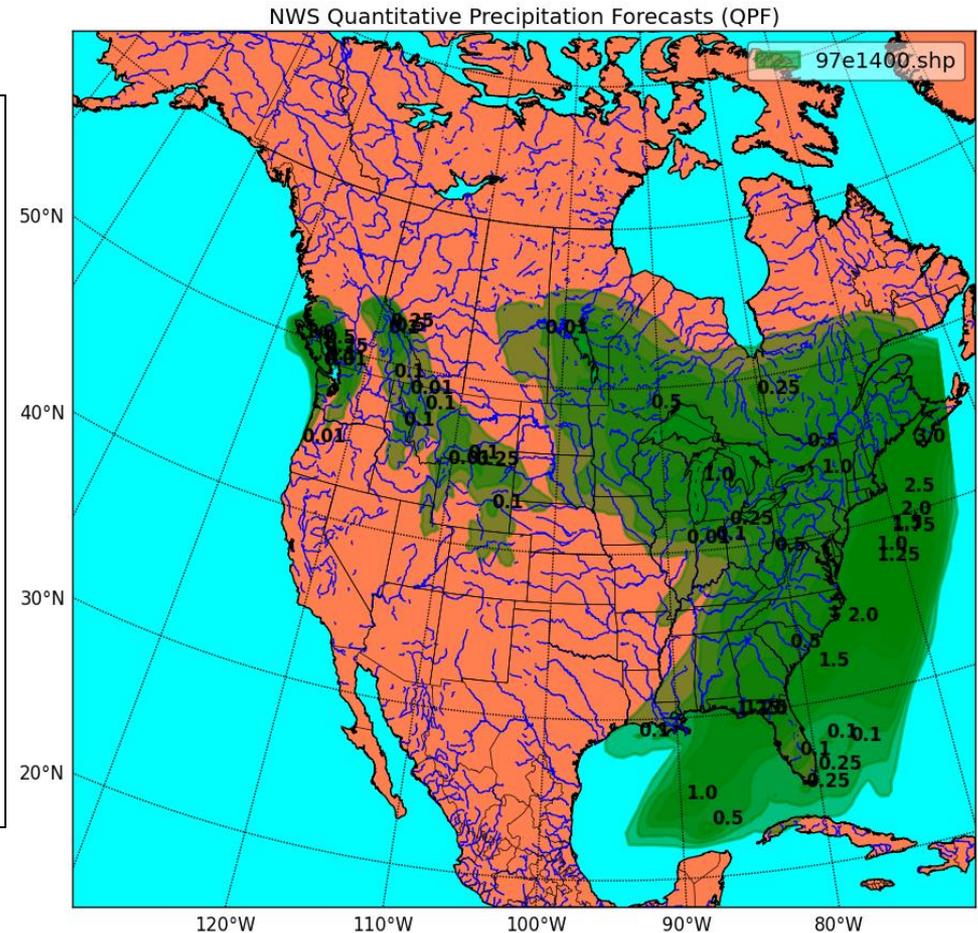
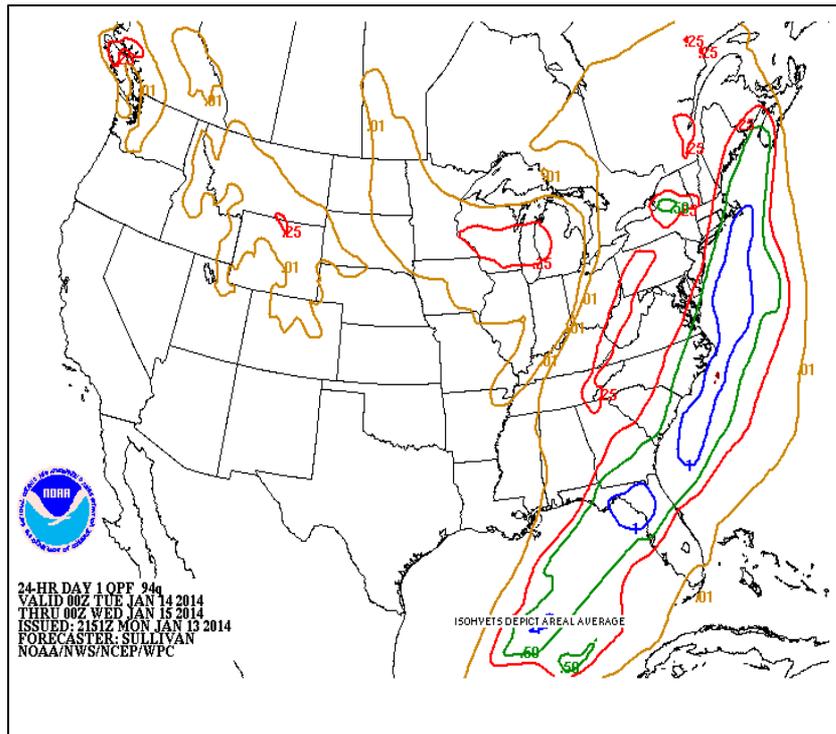
Discharge + Water Use

Parameter: Discharge (cfs)
StationID: 01415000

Parameter: Discharge + Water Use (cfs)
StationID: 01415000



Columbus, Indiana Flood Prediction: National Weather Service – Quantitative Precipitation Forecasts (QPF)



<http://www.hpc.ncep.noaa.gov/qpf/qpf2.shtml>
<ftp://ftp.hpc.ncep.noaa.gov/shapefiles/qpf/7day/>

Columbus, Indiana Flood Prediction: National Weather Service – Quantitative Precipitation Forecasts (QPF)

National Weather Service
Weather Prediction Center

Site Map News Organization
DOC NOAA NWS NCEP Centers: AWC CPC EMC NCO NHC OPC SPC SWPC WPC

Local forecast by "City, St" or Zip Code
City, St Go

Search WPC
Go

Facebook Twitter
NCEP Quarterly Newsletter

WPC Home
Analyses and Forecasts
National Forecast Charts
National High & Low
WPC Discussions
Surface Analysis
Days 1-2 CONUS
Days 3-7 CONUS
Days 4-8 Alaska
QPF
PQPF
Excessive Rainfall
Mesoscale Precip Discussion
Flood Outlook
Winter Weather
Storm Summaries
Heat Index
Air Quality
Tropical Products
Daily Weather Map
GIS Products

Current Watches/Warnings
Satellite and Radar Imagery
Satellite Images
National Radar
Product Archive
Verification
WPC Verification
NPVU
Model Diagnostics
Event Reviews
International Desks

Quantitative Precipitation Forecasts

Day 1	Days 1-2	5- and 7-day Totals
Day 2	Days 1-3	
Day 3	Days 4-5 and Days 6-7	

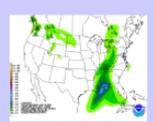
Loop of All [6-hourly](#) or [24-hourly](#) Forecasts for Days 1-3
[View 12-Hour QPFs for Days 1-3](#)
[WPC QPF Archive](#)

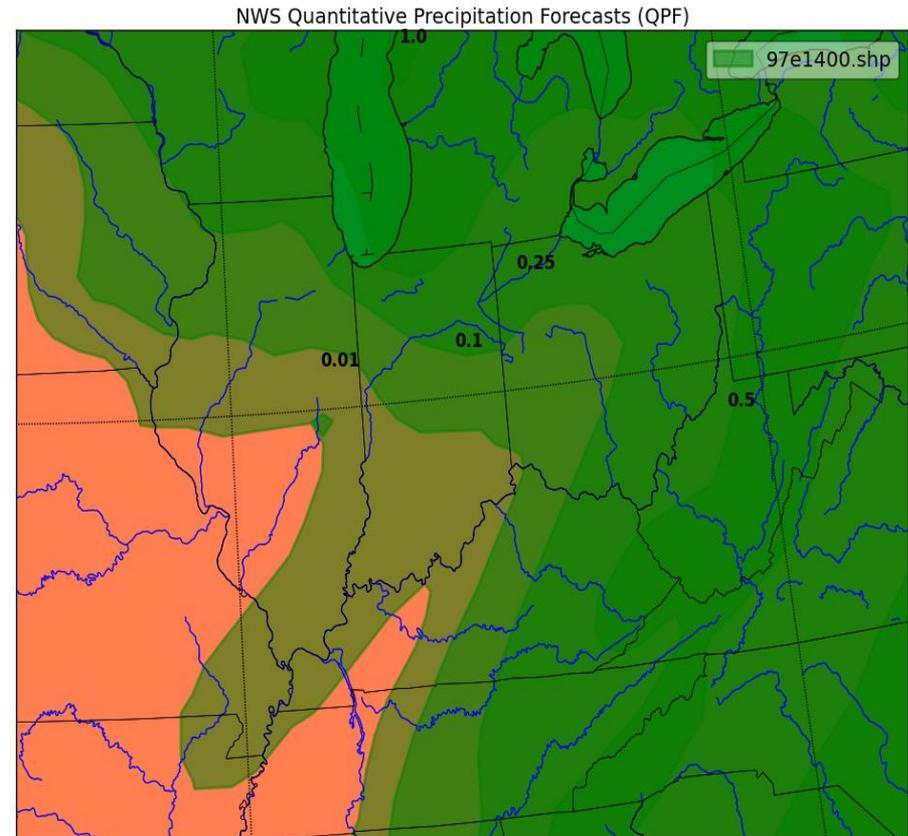
24 Hour Precipitation Total - Day 1



Day 1 QPF
[contours only]

6 Hourly Precipitation Amounts - Day 1

 <p>Update (00-06 hr) [contours only]</p>	 <p>06-12 hr. [contours only]</p>	 <p>12-18 hr. [contours only]</p>
--	--	--

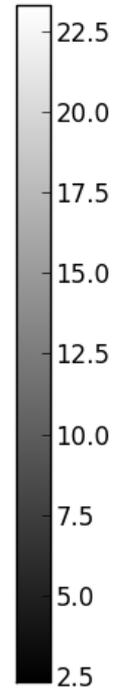
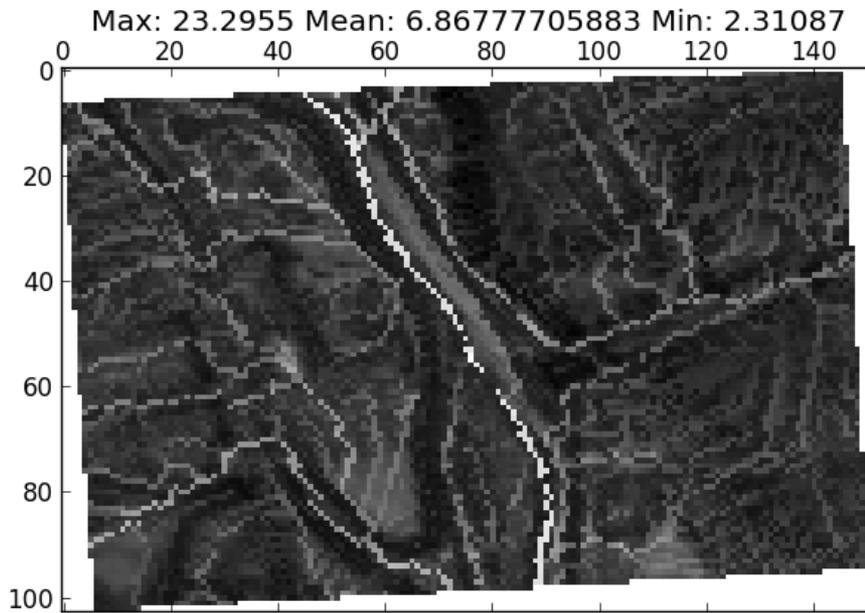


<http://www.hpc.ncep.noaa.gov/qpf/qpf2.shtml>
<ftp://ftp.hpc.ncep.noaa.gov/shapefiles/qpf/7day/>

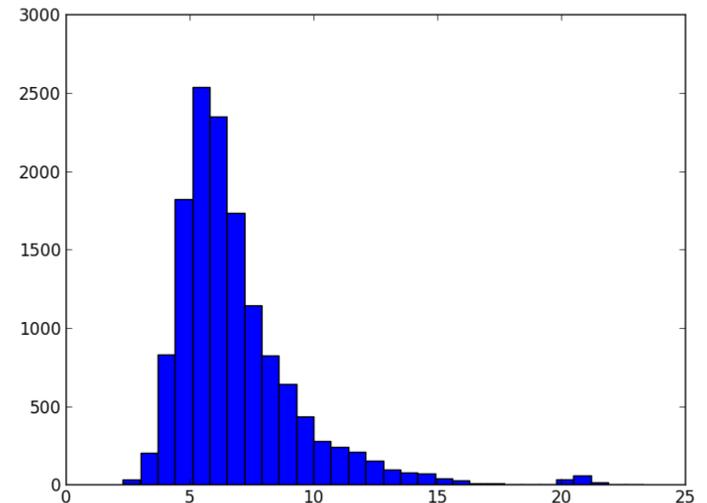
Preprocessing and ability to view/query data

Example: Topographic Wetness Index

TWI

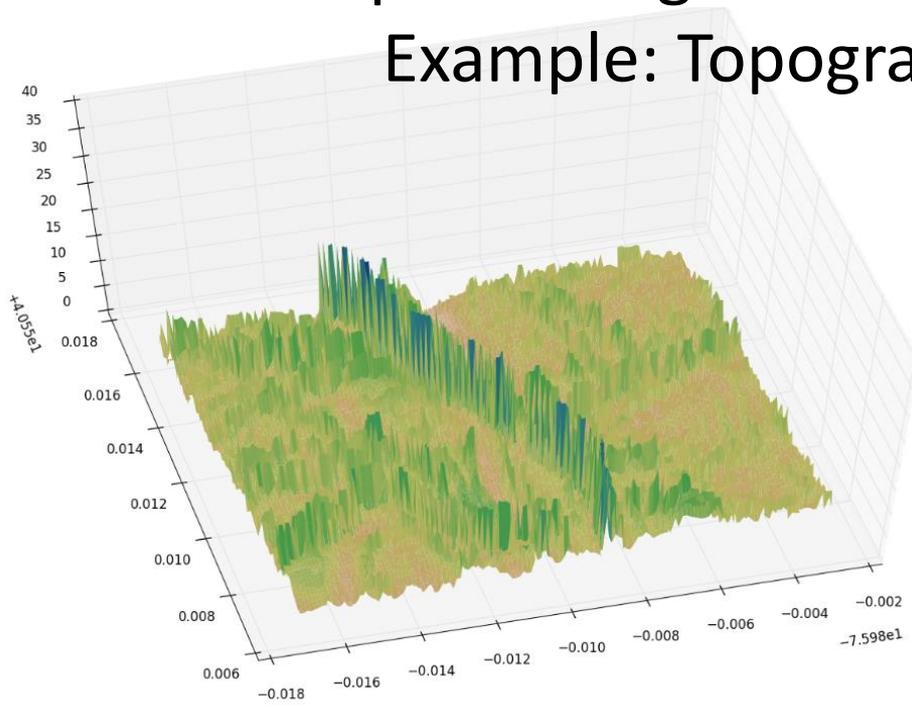


TWI histogram



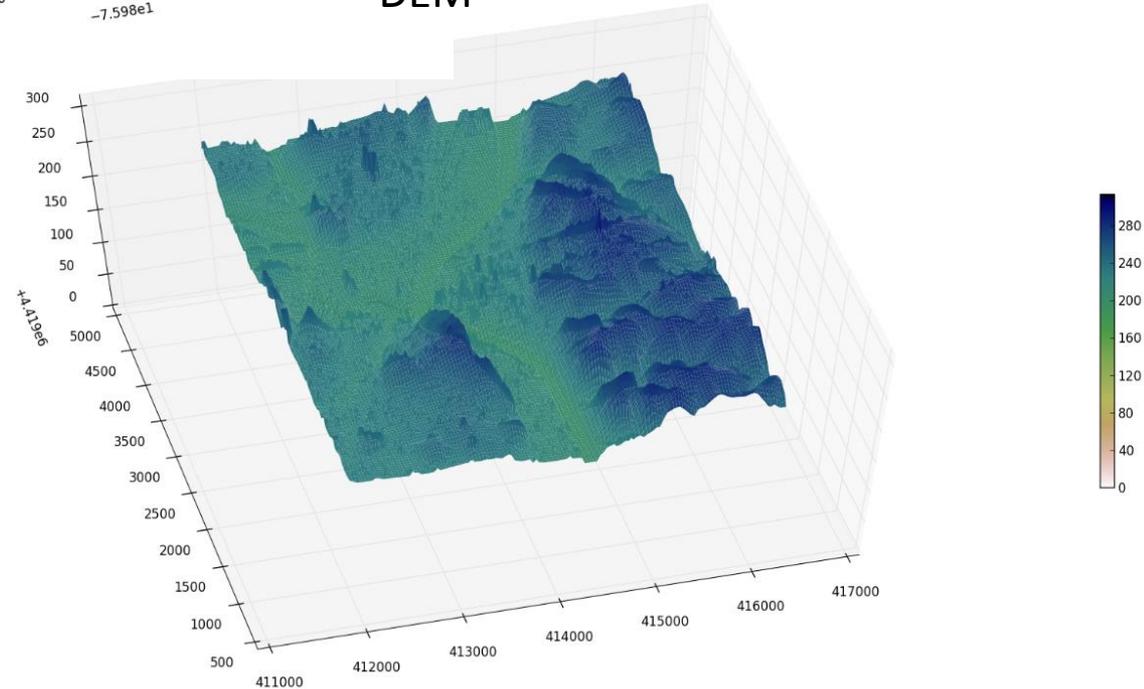
Preprocessing and ability to view/query data

Example: Topographic Wetness Index

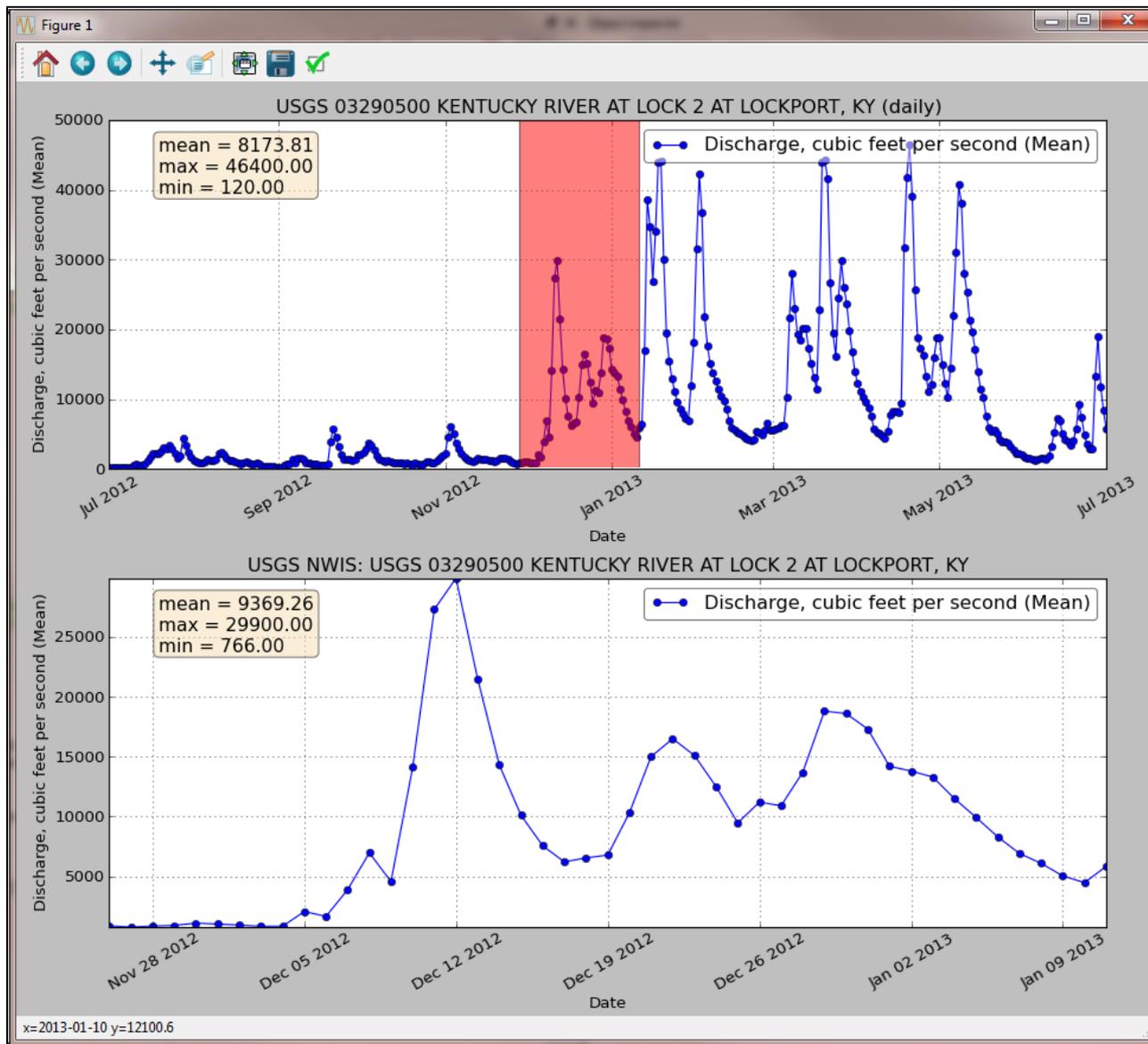


TWI

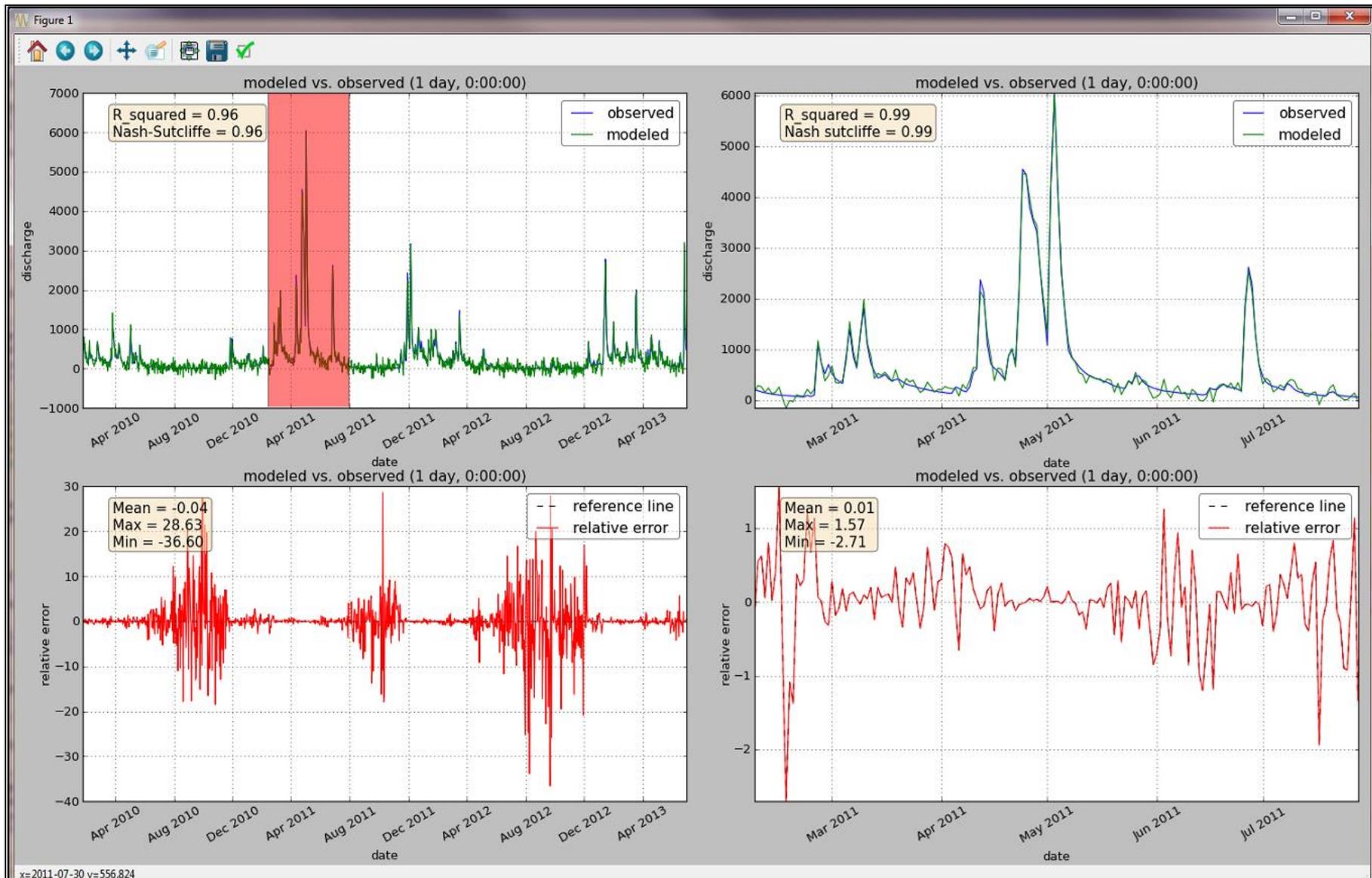
DEM



Modeling Preprocessing Tool - NWISPY



Modeling Post Processing Tool - HYDROCOMP



Best Practices for Scientific Computing

OPEN ACCESS Freely available online

PLOS BIOLOGY

Community Page

Best Practices for Scientific Computing

Greg Wilson^{1*}, D. A. Aruliah², C. Titus Brown³, Neil P. Chue Hong⁴, Matt Davis⁵, Richard T. Guy^{6*}, Steven H. D. Haddock⁷, Kathryn D. Huff⁸, Ian M. Mitchell⁹, Mark D. Plumbley¹⁰, Ben Waugh¹¹, Ethan P. White^{1,2}, Paul Wilson¹³

1 Mozilla Foundation, Toronto, Ontario, Canada, **2** University of Ontario Institute of Technology, Oshawa, Ontario, Canada, **3** Michigan State University, East Lansing, Michigan, United States of America, **4** Software Sustainability Institute, Edinburgh, United Kingdom, **5** Space Telescope Science Institute, Baltimore, Maryland, United States of America, **6** University of Toronto, Toronto, Ontario, Canada, **7** Monterey Bay Aquarium Research Institute, Moss Landing, California, United States of America, **8** University of California Berkeley, Berkeley, California, United States of America, **9** University of British Columbia, Vancouver, British Columbia, Canada, **10** Queen Mary University of London, London, United Kingdom, **11** University College London, London, United Kingdom, **12** Utah State University, Logan, Utah, United States of America, **13** University of Wisconsin, Madison, Wisconsin, United States of America

Introduction

Scientists spend an increasing amount of time building and using software. However, most scientists are never taught how to do this efficiently. As a result, many are unaware of tools and practices that would allow them to write more reliable and maintainable code with less effort. We describe a set of best practices for scientific software development that have solid foundations in research and experience, and that improve scientists' productivity and the reliability of their software.

Software is as important to modern scientific research as telescopes and test tubes. From groups that work exclusively on computational problems, to traditional laboratory and field scientists, more and more of the daily operation of science revolves around developing new algorithms, managing and analyzing the large amounts of data that are generated in single research projects, combining disparate datasets to assess synthetic problems, and other computational tasks.

Scientists typically develop their own software for these purposes because doing so requires substantial domain-specific knowledge. As a result, recent studies have found that scientists typically spend 30% or more of their time developing software [1,2]. However, 90% or more of them are primarily self-taught [1,2], and therefore lack exposure to basic software development practices such as writing maintainable code, using version control and issue trackers, code reviews, unit testing, and task automation.

We believe that software is just another kind of experimental apparatus [3] and should be built, checked, and used as carefully as any physical apparatus. However, while most scientists are careful to validate their laboratory and field equipment, most do not know how reliable their software is [4,5]. This can lead to serious errors impacting the central conclusions of published research [6]: recent high-profile retractions, technical comments, and corrections because of errors in computational methods include papers in *Science* [7,8], *PNAS* [9], the *Journal of Molecular Biology* [10], *Ecology Letters* [11,12], the *Journal of Neurology* [13], *Journal of the American College of Cardiology* [14], *Hypertension* [15], and *The American Economic Review* [16].

In addition, because software is often used for more than a single project, and is often reused by other scientists, computing errors can have disproportionate impacts on the scientific process. This type of cascading impact caused several prominent retractions when an

error from another group's code was not discovered until after publication [6]. As with bench experiments, not everything must be done to the most exacting standards; however, scientists need to be aware of best practices both to improve their own approaches and for reviewing computational work by others.

This paper describes a set of practices that are easy to adopt and have proven effective in many research settings. Our recommendations are based on several decades of collective experience both building scientific software and teaching computing to scientists [17,18], reports from many other groups [19–25], guidelines for commercial and open source software development [26,27], and on empirical studies of scientific computing [28–31] and software development in general [summarized in [32]]. None of these practices will guarantee efficient, error-free software development, but used in concert they will reduce the number of errors in scientific software, make it easier to reuse, and save the authors of the software time and effort that can be used for focusing on the underlying scientific questions.

Our practices are summarized in Box 1; labels in the main text such as “[1a]” refer to items in that summary. For reasons of space, we do not discuss the equally important (but independent) issues of reproducible research, publication and citation of code and data, and open science. We do believe, however, that all of these will be much easier to implement if scientists have the skills we describe.

Citation: Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, et al. (2014) Best Practices for Scientific Computing. *PLoS Biol* 12(1): e1001745. doi:10.1371/journal.pbio.1001745

Academic Editor: Jonathan A. Eisen, University of California Davis, United States of America

Published: January 7, 2014

Copyright: © 2014 Wilson et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Funding: Neil Chue Hong was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) Grant EP/H043160/1 for the UK Software Sustainability Institute. Ian M. Mitchell was supported by NSERC Discovery Grant #29211. Mark Plumbley was supported by EPSRC through a Leadership Fellowship (EP/G007144/1) and a grant (EP/M01011/1) for SoandSoftware.ac.uk. Ben Waugh was supported by a CAREER grant from the US National Science Foundation (OEB-0933694). Greg Wilson was supported by a grant from the Sloan Foundation. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The lead author (GW) is involved in a pilot study of code review in scientific computing with PLOS Computational Biology.

* E-mail: gwilson@software-carpentry.org

Current address: Microsoft, Inc, Seattle, Washington, United States of America

Box 1. Summary of Best Practices

1. Write programs for people, not computers.
 - (a) A program should not require its readers to hold more than a handful of facts in memory at once.
 - (b) Make names consistent, distinctive, and meaningful.
 - (c) Make code style and formatting consistent.
2. Let the computer do the work.
 - (a) Make the computer repeat tasks.
 - (b) Save recent commands in a file for re-use.
 - (c) Use a build tool to automate workflows.
3. Make incremental changes.
 - (a) Work in small steps with frequent feedback and course correction.
 - (b) Use a version control system.
 - (c) Put everything that has been created manually in version control.
4. Don't repeat yourself (or others).
 - (a) Every piece of data must have a single authoritative representation in the system.
 - (b) Modularize code rather than copying and pasting.
 - (c) Re-use code instead of rewriting it.
5. Plan for mistakes.
 - (a) Add assertions to programs to check their operation.
 - (b) Use an off-the-shelf unit testing library.
 - (c) Turn bugs into test cases.
 - (d) Use a symbolic debugger.
6. Optimize software only after it works correctly.
 - (a) Use a profiler to identify bottlenecks.
 - (b) Write code in the highest-level language possible.
7. Document design and purpose, not mechanics.
 - (a) Document interfaces and reasons, not implementations.
 - (b) Refactor code in preference to explaining how it works.
 - (c) Embed the documentation for a piece of software in that software.
8. Collaborate.
 - (a) Use pre-merge code reviews.
 - (b) Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems.
 - (c) Use an issue tracking tool.

Following Best Practices – Modular, Version Controlled, Unit Tested, Documented (accessible online), etc.

hydrocomp

[home](#) | [search](#) | [previous](#) | [next](#) | [modules](#) | [index](#)

Overview

hydrocomp is a project that contains python modules that compare timeseries output from a model of a particular parameter (i.e. discharge) with an observed timeseries of the same parameter and compute differences and statistics between modeled and observed hydrologic parameters. For example, the model timeseries can be a rainfall-runoff model output of estimated discharge and the observed timeseries can be an USGS NWIS data file that contains discharge as a hydrologic parameter.

The following are the statistics calculated:

- Nash-Sutcliffe
- R Squared Coefficient
- Mean Squared Error
- Absolute Error
- Relative Error
- Percent Error
- Percent Difference

Details

hydrocomp.py is a module that contains functions to calculate, print, and plot comparison data and statistics.

In the *hydrocomp.py* module, *main()* prompts user for observed and model files. Processes each file, prints information, and plots data and statistics. Information is printed to the screen. Plots are saved to a directory called 'figs' which is created in the same directory as the data file. Currently, if an NWIS data file is selected as the observed file a log file called 'nwis_error.log' is created if any errors are found in the data file.

nwispy.py is a module that contains functions to read, print, and plot data from an USGS NWIS data file. The parameters; i.e. discharge, gage height, temperature, sediment concentration, etc.

USGS NWIS data files can be found at:
<http://waterdata.usgs.gov/nwis/rt>

statistics.py is a module that contains functions to calculate all the statistics.

helpers.py is a module that currently contains functions to subset dates and find common date ranges between files.

water.py is a module that reads output from an application called WATER (gui wrapper around a Kentucky application).

Author

Jeremiah Lant
jlant@usgs.gov



GitHub

This repository | Search or type a command | Explore | Features | Enterprise | Blog | [Sign up](#) | [Sign in](#)

jlant-usgs / hydrocomp ★ Star 0 🍴 Fork 0

hydrocomp is a repository that contains code that compares timeseries output from a model of a particular parameter (i.e. discharge) with an observed timeseries of the same parameter.

10 commits | 1 branch | 0 releases | 1 contributor

branch: master | **hydrocomp** | [+](#)

updated functions and main() for the updated watertxt.py module
jlant-usgs authored a month ago | latest commit baf578a4ca

bin	initial commit	6 months ago
data	minor doc string edits	6 months ago
docs	added intended use statements to docs and README file	3 months ago
hydrocomp	updated functions and main() for the updated watertxt.py module	a month ago
tests	initial commit	6 months ago
.gitignore	added intended use statements to docs and README file	3 months ago
LICENSE.txt	initial commit	6 months ago
README.md	added intended use statements to docs and README file	3 months ago
setup.py	initial commit	6 months ago

[Code](#) | [Issues](#) 0 | [Pull Requests](#) 0 | [Pulse](#) | [Graphs](#) | [Network](#)

HTTPS clone URL
<https://github.com>

You can clone with HTTPS, or Subversion

[Clone in Desktop](#) | [Download ZIP](#)

<https://github.com/jlant-usgs>

Questions?

Jeremiah Lant
USGS Hydrologist
Kentucky Water Science Center
jlant@usgs.gov
(502) 493-1949

GitHub site:
<https://github.com/jlant-usgs>

Example online documentation:
http://ky.water.usgs.gov/usgs/projects/jlant_program_code/nwispy/html/index.html