WDSA CCWI Ferrara 2024

Università degli Studi di Ferrara

WDSA/CCWI 2024 Short Course

WNTR and related open-source software for water distribution system analysis: Training and discussion on future capabilities

WNTR Water Network Tool for Resilience

July 1, 2024 Ferrara, Italy

Sandia National Laboratories

EPA

TEXAS The University of Texas at Austin

Agenda

- Introductions: 5 min
- Using the Tutorials: 5 min
- Overview: 15 min
- WNTR Tutorial: 60 min
- Break: 15 min
- MAGNets and VisWaterNet Tutorial: 45 min
- Break: 15 min
- Future Capabilities and Discussion: 45 min

Agenda

- **Introductions: 5 min**
- Using the Tutorials: 5 min
- Overview: 15 min
- WNTR Tutorial: 60 min
- Break: 15 min
- MAGNets and VisWaterNet Tutorial: 45 min
- Break: 15 min
- Future Capabilities and Discussion: 45 min

Introductions

Kate Klise
Research Scientist
Sandia National Laboratories

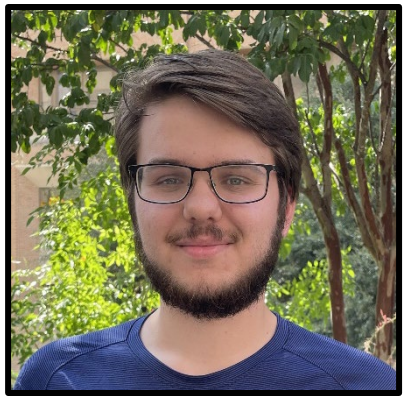
Terra Haxton
Environmental Engineer
US Environmental Protection Agency

Regan Murray
Director, Water Infrastructure Division
US Environmental Protection Agency

Lina Sela
Associate Professor
The University of Texas at Austin

Meghna Thomas
Ph.D. Candidate
The University of Texas at Austin

Other Team Members



Tyler Trimble
Undergraduate Researcher
The University of Texas at Austin

J'Lynn Estorga
Undergraduate Researcher
The University of Texas at Austin

Agenda

- Introductions: 5 min
- **Using the Tutorials: 5 min**
- Overview: 15 min
- WNTR Tutorial: 60 min
- Break: 15 min
- MAGNets and VisWaterNet Tutorial: 45 min
- Break: 15 min
- Future Capabilities and Discussion: 45 min

Using the Tutorials

- All scripts are implemented in **Python** using Jupyter Notebooks
- We will be using the **Binder** online platform
- To access the material for this workshop go to:
- tinyurl.com/wdsa-ccwi-shortcourse
- This Binder contains all of the Jupyter Notebook scripts and files you will need for this short course
- All codes can be run locally on your browser without having to install any packages
- **Note**: The Binder will take a few minutes to load the first time you open the link

Agenda

- Introductions: 5 min
- Using the Tutorials: 5 min
- **Overview: 15 min**
- WNTR Tutorial: 60 min
- Break: 15 min
- MAGNets and VisWaterNet Tutorial: 45 min
- Break: 15 min
- Future Capabilities and Discussion: 45 min

Background: Water Distribution System Disruptions

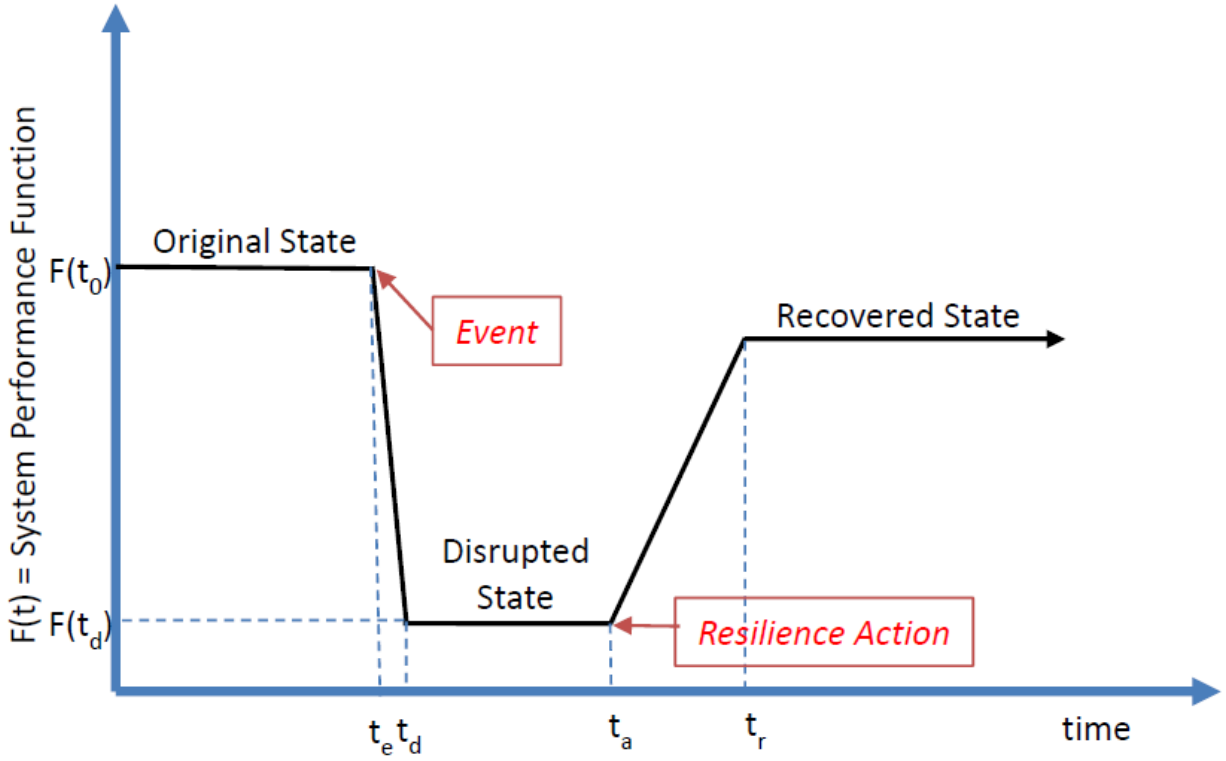
Water distribution systems are subject to a wide range of disruptions:

- Aging infrastructure
- Changes in supply and demand
- Damage from natural disasters (hurricanes, landslides, floods, wildfire, winter storms, earthquakes, land subsidence)
- Saltwater intrusion
- Contamination or water quality violations
- Cyber or terrorist attacks

Background: Water Distribution System Resilience

The goal of a resilient system is to minimize the magnitude and duration of disruption

- Resilience is influenced by the design, maintenance, operations, and dependence with other infrastructure



State transition or "resilience trapezoid"

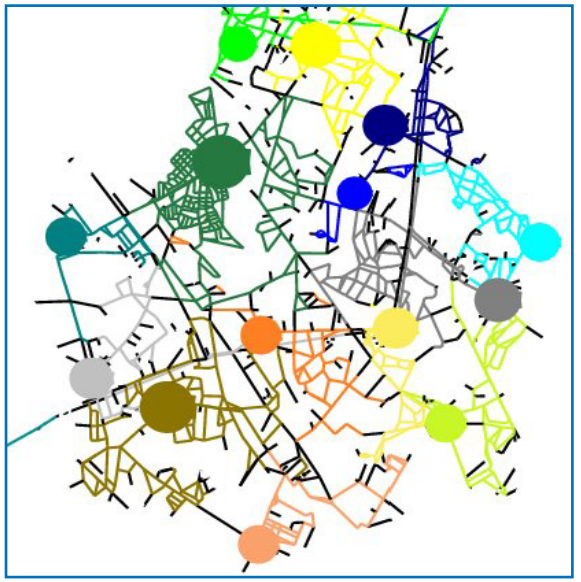
EPA Developed Relevant Tools to Address Resilience

Design of contamination warning systems

- EPANET for water distribution systems analysis
- EPANET-MSX for multi-species contaminant modeling
- TEVA-SPOT for sensor placement optimization
- CANARY for event detection data fusion
- Water Security Toolkit (WST) for response action optimization (flushing, booster chlorination, source inversion/backtracking)

Real-time modeling and response

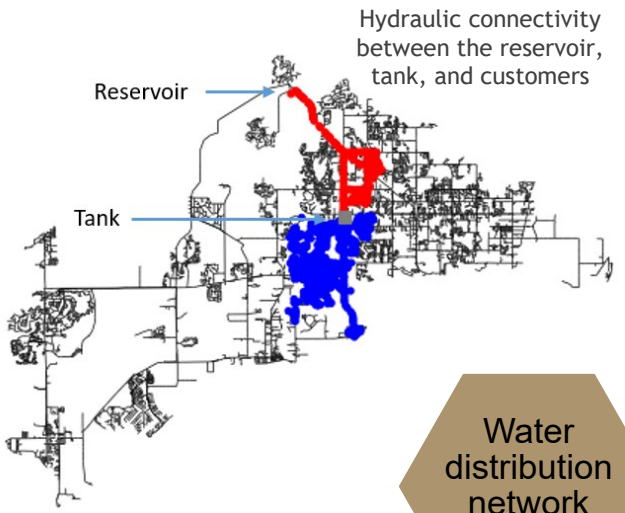
- EPANET-RTX libraries

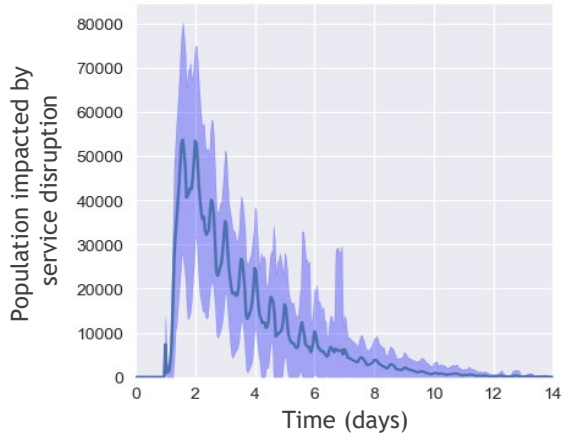


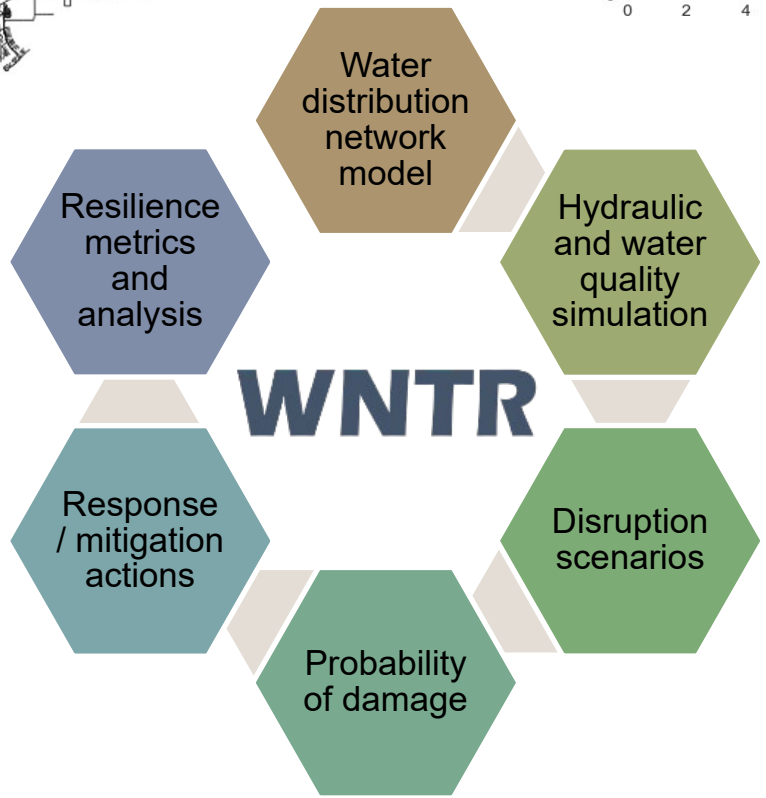
With a growing national interest in addressing all hazards, including natural disasters and climate resilience, we learned from our experience developing these tools to develop WNTR as an open-source Python package.

Water Network Tool for Resilience (WNTR)

WNTR is an open-source Python package designed to analyze water distribution system failure and recovery given a wide range of hazards, including pipe breaks, power outages, hurricanes, earthquakes, and cyber attacks.







Intended to answer utility specific questions:

- How long can the system continue to provide water to customers?
- How many people will be impacted?
- Which critical facilities/functions will be impacted?
- What is the best response in the immediate aftermath?
- Which components should be hardened to minimize future disruptions?

https://usepa.github.io/WNTR/

WNTR User and Developer Community: By the Numbers

Over 280,000 downloads through PyPI and conda-forge

20 software releases since 2016

Over 500 active user manual visitors each month. Top 50% of views from US, China, Italy, and Netherlands

Active user community posting issues, feedback, and suggestions to the GitHub repository

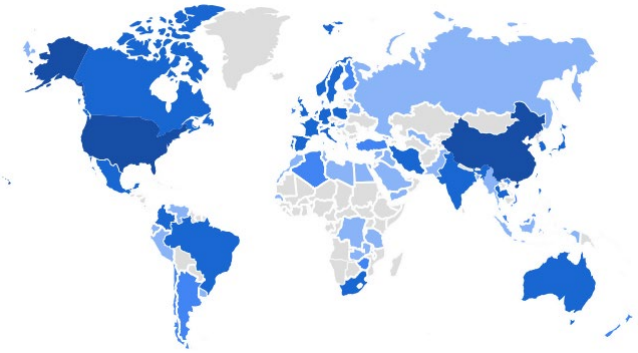
31 contributors from Sandia, EPA, and beyond

Over 400 publications reference WNTR

Over 90 software repositories include WNTR as a dependency

Check out the WNTR User Community webpage!

Active user manual visitors, last 90 days



WNTR Water Network Tool for Resilience

User Guide | API documentation | User community

Related software

- CriticalityMaps: pshassett/CriticalityMaps
- Digital HydrAuLic SIMulator (DHALSIM): Critical-Infrastructure-Systems-Lab/DHALSIM
- LeakDB: KIOS-Research/LeakDB
- MAGNets: meghnathomas/MAGNets
- MILPNet: meghnathomas/MILPNet
- PPMTools: USEPA/PPMtools
- PTSNet: gandresr/ptsnet
- TSNet: glorialulu/TSNet
- VisWaterNet: tylertrimble/viswaternet

See the WNTR GitHub Dependeny Graph to find additional repositories and packages that use WNTR.

Publications

- N. Abdel-Mottaleb, P. Ghasemi Saghand, H. Charkhgard, and Q. Zhang. An exact multiobjective optimization approach for evaluating water distribution infrastructure criticality and geospatial interdependence. *Water Resources Research*, 55(7):5255–5276, 2019. doi:10.1029/2018WR024063.
- Ariel Antonowicz, Alicja Bałut, Andrzej Urbaniak, and Przemysław Zakrzewski. Algorithm for early warning system for contamination in water network. In *2019 20th International Carpathian Control Conference (ICCC)*, 1–5. 2019. doi:10.1109/CarpathianCC.2019.8765966.

Who is using WNTR? And what are they using it for?

- **Arcadis** – Earthquake resilience for Seattle
- **Army Engineer Research and Development Center** – Military 14-day energy and water security initiative
- **Architecture and Building Systems, Switzerland** – Urban energy simulations
- **Case Western Reserve University** – Water/transportation infrastructure vulnerability
- **TU Delft, Singapore University of Technology and Design** – Cyber resilience
- **Global Quality Corp** – Sensor placement optimization
- **National Technical Univ. of Athens, KWR Netherlands** – Cyber resilience
- **Naval Postgraduate School** – Impact of term power outage for island communities
- **North Carolina University** – Leak detection
- **Rice University** – Distributed direct potable water reuse
- **University of California, Los Angeles** – Earthquake resilience
- **University of Cyprus, KIOS** – Benchmark dataset for leakage diagnosis
- **University of Texas, Austin** – Transient pressure analysis, graphics, reduced order models, pump scheduling

And many more…

Example WNTR Case Studies

- Earthquake: demonstrated infrastructure damage associated with network integrity and earthquake magnitude, as well as resources and repair strategies.
- Power Outage: evaluated resilience of water systems of St. Croix and the combined system of St. Thomas/St. John to four-week power outages.
- Contaminated Water Source: assessed effect of source water loss on water service availability due to saltwater intrusion in Poughkeepsie, NY.
- Environmental Justice: prioritized pipe replacement using social-economic data in Pittsburgh, PA.
- Landslide: identified pipes at risk for landslides and associated social vulnerability data for Pennsylvania utility (*accepted for publication*).

Summary of WNTR and case study applications available online.

Agenda

- Introductions: 5 min
- Using the Tutorials: 10 min
- Overview: 15 min
- **WNTR Tutorial: 60 min**
- Break: 15 min
- MAGNets and VisWaterNet Tutorial: 45 min
- Break: 15 min
- Future Capabilities and Discussion: 45 min

WNTR Tutorial

Installation and Getting Started

Software Framework

Basic Tutorial

Geospatial Tutorial

Installation and Examples

WNTR is a Python package that can be installing using pip or conda

```
> pip install wntr
> conda install -c conda-forge wntr
```

Required Dependencies (installed with WNTR)

- Numpy
- Scipy
- NetworkX
- Pandas
- Matplotlib
- Setuptools

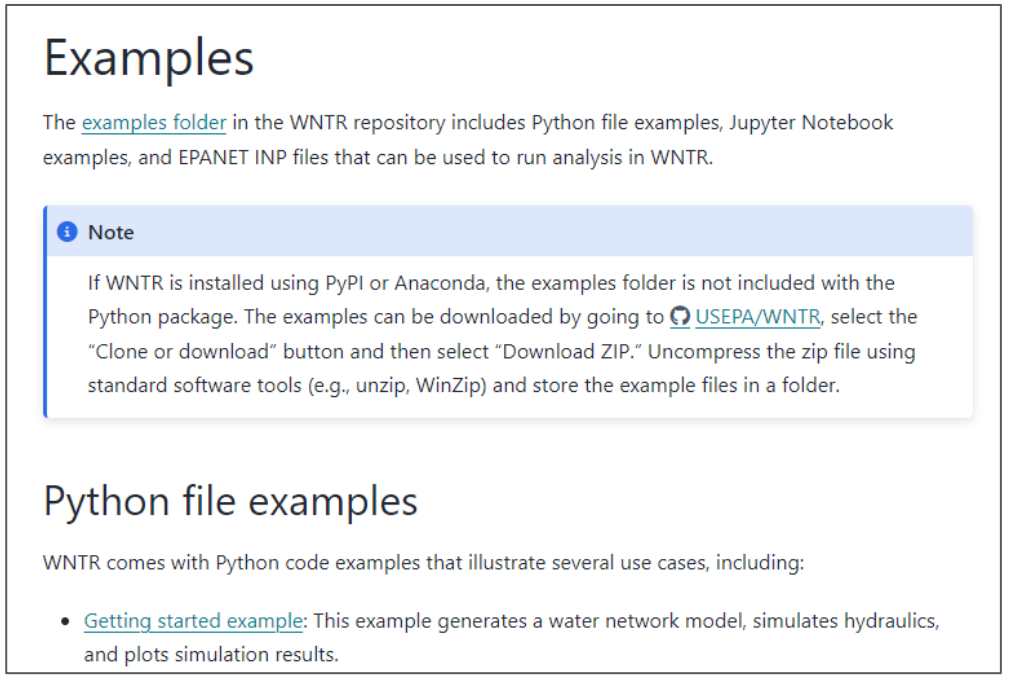
Optional dependencies include Geopandas, which can be installed using the requirements file

```
> pip install -r requirements.txt
```

https://usepa.github.io/WNTR/installation

WNTR comes with several examples to help get new users started

The examples folder in the WNTR repository includes Python file examples, Jupyter Notebook examples, and EPANET INP files that can be used to run analysis in WNTR.



https://usepa.github.io/WNTR/examples

Getting Started

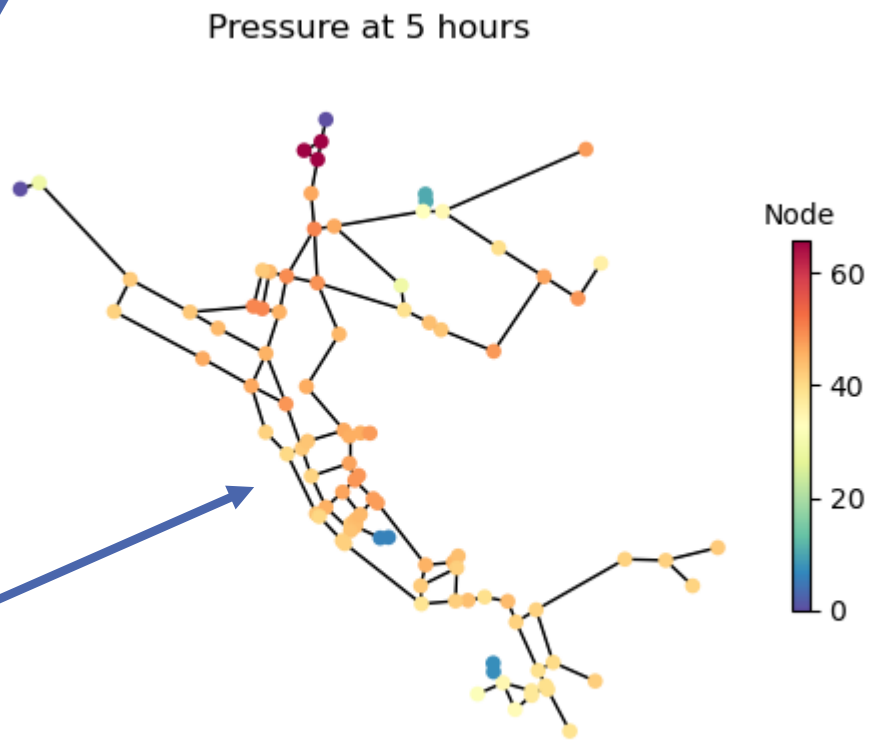
WNTR comes with a simple getting started example, which uses the EPANET Example Network 3 (Net3) INP file. This example demonstrates how to:

- Import WNTR
- Create a water network model from an EPANET INP file
- Simulate hydraulics using EPANET
- Plot simulation results on the network

```
"""
The following example demonstrates how to import WNTR, create a water
network model from an EPANET INP file, simulate hydraulics, and plot
simulation results on the network.
"""
# Import WNTR
import wntr

# Create a water network model
inp_file = 'networks/Net3.inp'
wn = wntr.network.WaterNetworkModel(inp_file)

# Simulate hydraulics
sim = wntr.sim.EpanetSimulator(wn)
results = sim.run_sim()

# Plot results on the network
pressure_at_5hr = results.node['pressure'].loc[5*3600, :]
wntr.graphics.plot_network(wn, node_attribute=pressure_at_5hr,
                           node_size=30, title='Pressure at 5 hours')
```

pressure_at_5hr in meters

```
10         29.004883
15         36.064419
20         10.455683
35         42.687435
40          6.064279
...
```

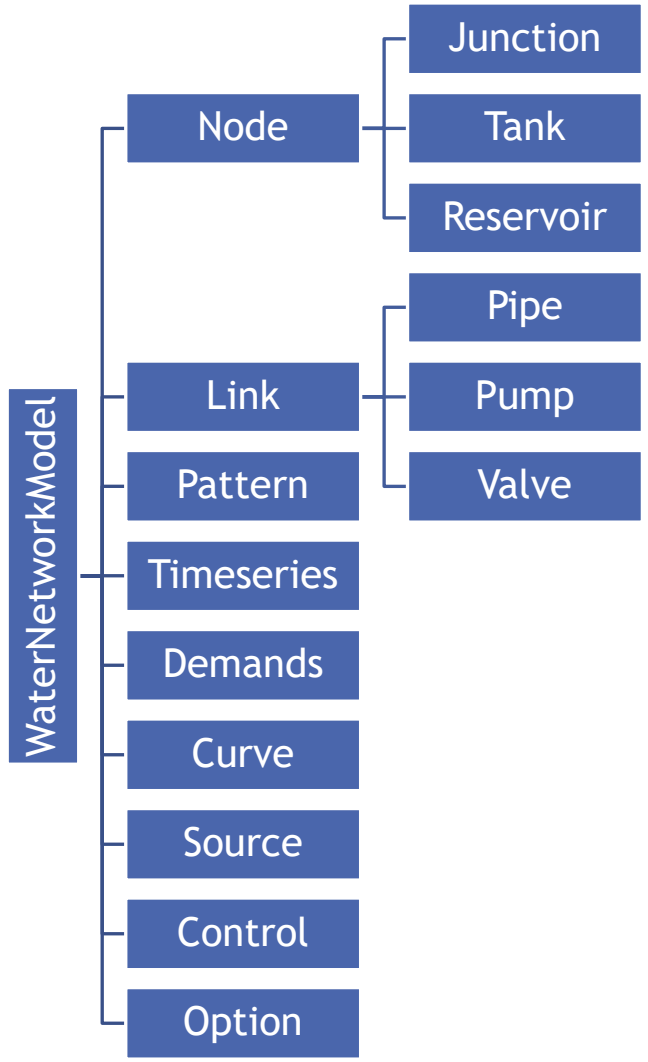


WNTR Software Framework

WNTR Subpackages

Subpackage	Description
`network`	Contains classes and methods to define a water network model, network controls, model options, and graph representation of the network.
`scenario`	Contains classes and methods to define disaster scenarios and fragility/survival curves.
`sim`	Contains classes and methods to run hydraulic and water quality simulations using the water network model.
`metrics`	Contains functions to compute resilience, including topographic, hydraulic, water quality, water security, and economic metrics.
`morph`	Contains methods to modify water network model morphology, including network skeletonization, modifying node coordinates, and splitting or breaking pipes.
`gis`	Contains geospatial capabilities, including a function to convert the water network model to GeoDataFrames.
`graphics`	Contains functions to generate graphics.
`epanet`	Contains EPANET 2.00.12 and EPANET 2.2.0 compatibility class and methods for WNTR.
`utils`	Contains helper functions.

Class Heirarchy of the WaterNetworkModel



WNTR Software Framework

	EpanetSimulator	WNTRSimulator
Driver	EPANET2.2 or EPANET2.00.12 DLLs	Customizable Python/C++
DD and PDD hydraulics	✓	✓
Water quality	✓	X
Leaks	✓ Emitters	✓ Leak model
Network components	✓ Standard	✓ Additional customization
Rules/Control options	✓ Standard	✓ Additional customization
Simulation options	- HW and DW	- HW - Simulate extreme low pressure conditions - Simulate disconnected networks
Runtime	Faster	Slower

Running a basic simulation

```
import wntr

inp_file = 'networks/Net3.inp'
wn = wntr.network.WaterNetworkModel(inp_file)

sim = wntr.sim.EpanetSimulator(wn)
results = sim.run_sim()
```

Results are stored as a dictionary of DataFrames.

Each DataFrame is indexed by time (in seconds) and column are node names or link names

`results.node`

- Demand
- Head
- Pressure
- Quality

`results.link`

- Flowrate
- Friction factor
- Headloss
- Quality
- Reaction rate
- Setting
- Status
- Velocity

Working with Pandas DataFrames and GeoPandas GeoDataFrames

- Simulation results
- Network properties
- Resilience metrics
- Geospatial data

DataFrame/GeoDataFrame
= Labeled matrix

Series/GeoSeries
= Labeled vector

Simulation results for node pressure

index = time in seconds
columns = node names

```
> results.node['pressure']

          10         15   ...          2          3
0      -0.450070  28.593657  ...   7.162800   8.839199
3600   28.253721  28.887156  ...   6.752899   9.099170
7200   28.764309  30.003231  ...   6.369764   9.450771
10800  28.871420  30.415579  ...   6.534997   9.869695
14400  29.659281  31.970745  ...   6.740044  10.284711
...
```

Extract pressure at a specific time step

index = node names

```
> results.node['pressure'].loc[3600,:]

10       28.253721
15       28.887156
20        9.099179
35       41.513771
40        4.191203
...
```

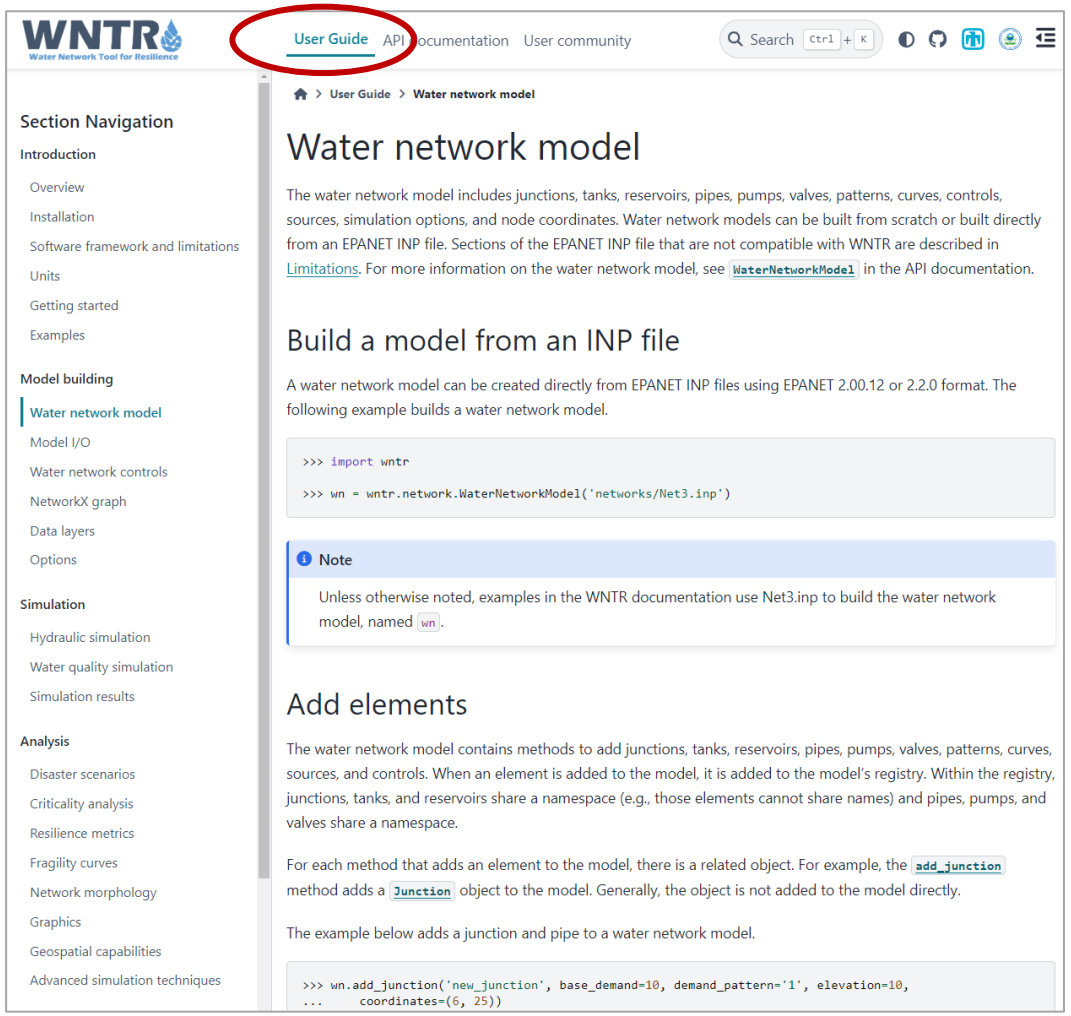
Extract pressure at a specific location

index = time in seconds

```
results.node['pressure'].loc[:,'123']

0         28.593657
3600      28.887156
7200      30.003231
10800     30.415579
14400     31.970745
...
```

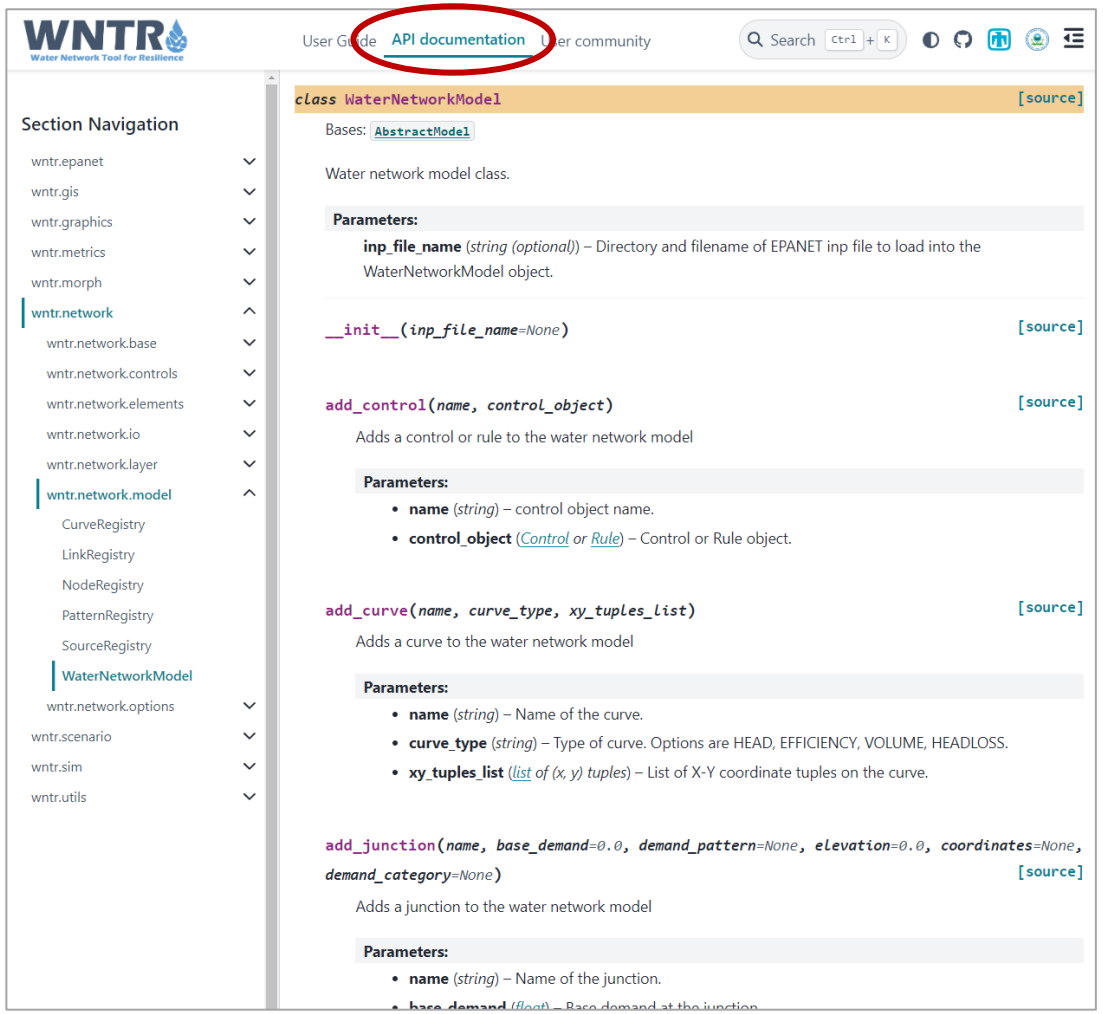
WNTR Documentation, https://usepa.github.io/WNTR

User Guide



https://usepa.github.io/WNTR/userguide

API Documentation



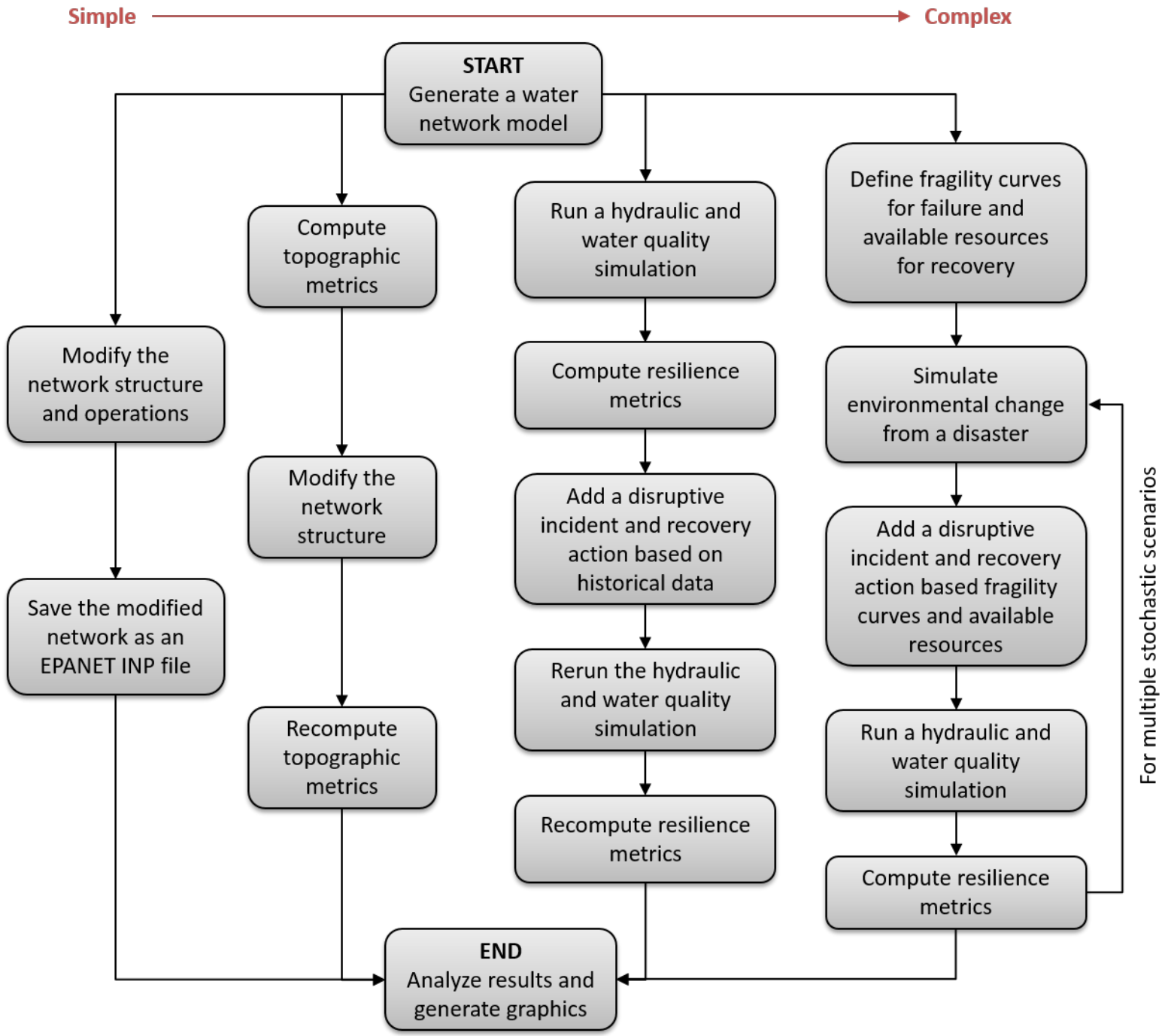
https://usepa.github.io/WNTR/wntr-api

Analysis Options

WNTR is modular and can be configured to run different types of analysis, from simple to complex

- Build and modify water network models
- Study network structure under different pipe configurations
- Compare hydraulics from normal and disrupted operations
- Run stochastic analysis on damage and recovery events

Most analysis starts with an EPANET INP model file, but models can be built from GIS data or from scratch



WNTR Data Formats

Easily translate water network models and simulation results between data formats for analysis and visualization

Store model, interface with external software packages

JSON file

EPANET INP file

CSV File

dictionary

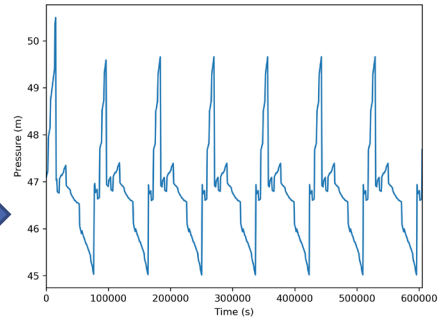
WNTR Water Network Model

Simulation and Analysis

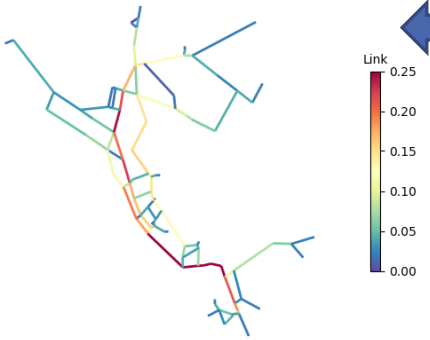
Pandas DataFrame

Append columns

Timeseries analysis and graphics



Topographic metrics and network graphics

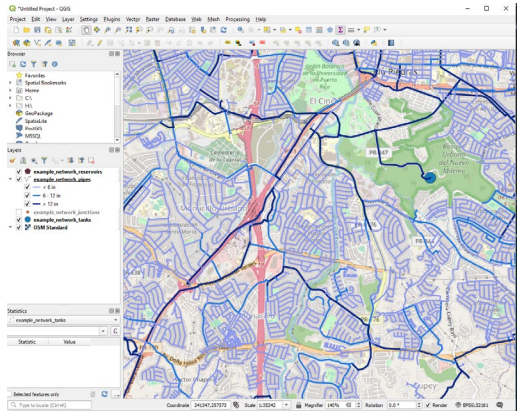


NetworkX Graph

GeoPandas GeoDataFrame

GIS GeoJSON and Shapefile files

Geospatial analysis and graphics in GIS



WNTR Basics Tutorial

Water Network Models

- Generate from EPANET INP file
- Add/delete/modify/query network elements

Model I/O

- Convert to/from data formats and files
- Create a WaterNetworkModel from GIS data

Hydraulic and Water Quality Simulation

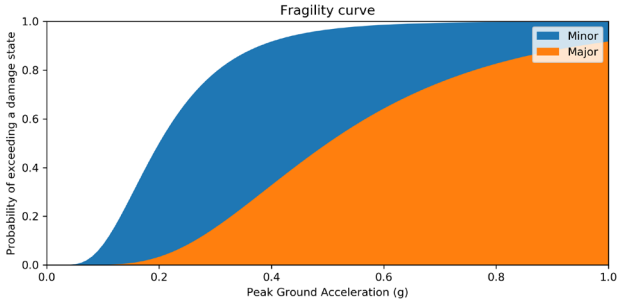
- EpanetSimulator
- WNTRSimulator
- Simulation results

Resilience Metrics

- Topographic
- Hydraulic
- Water quality

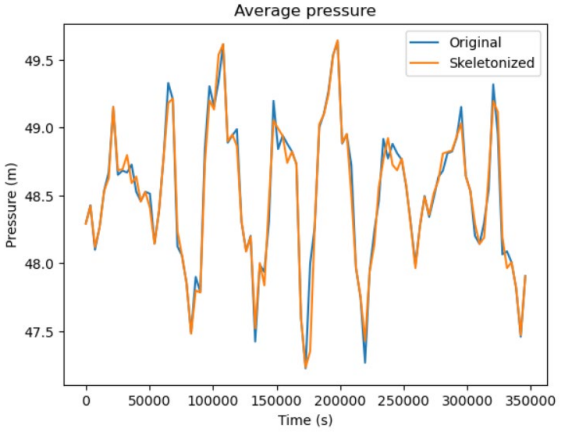
Fragility Curves

- Probability of damage from environmental condition



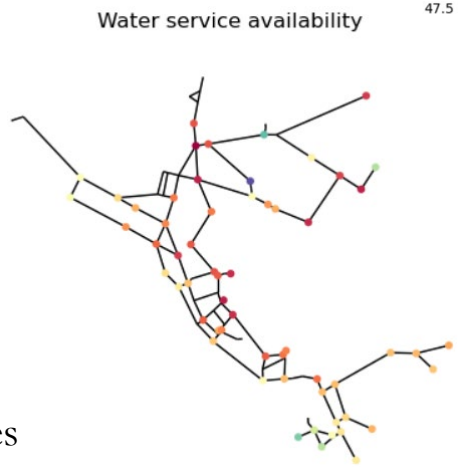
Network Skeletonization

- Reduce the size of a WaterNetworkModel



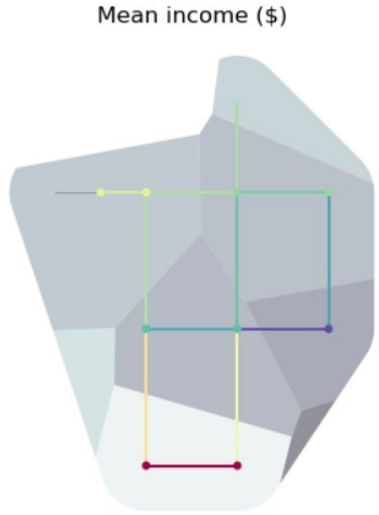
Valve Segmentation

- Group links and nodes into segments based on the location of isolation valves



Geospatial Capabilities

- Snap and intersect GIS data to WaterNetworkModel pipes and junctions

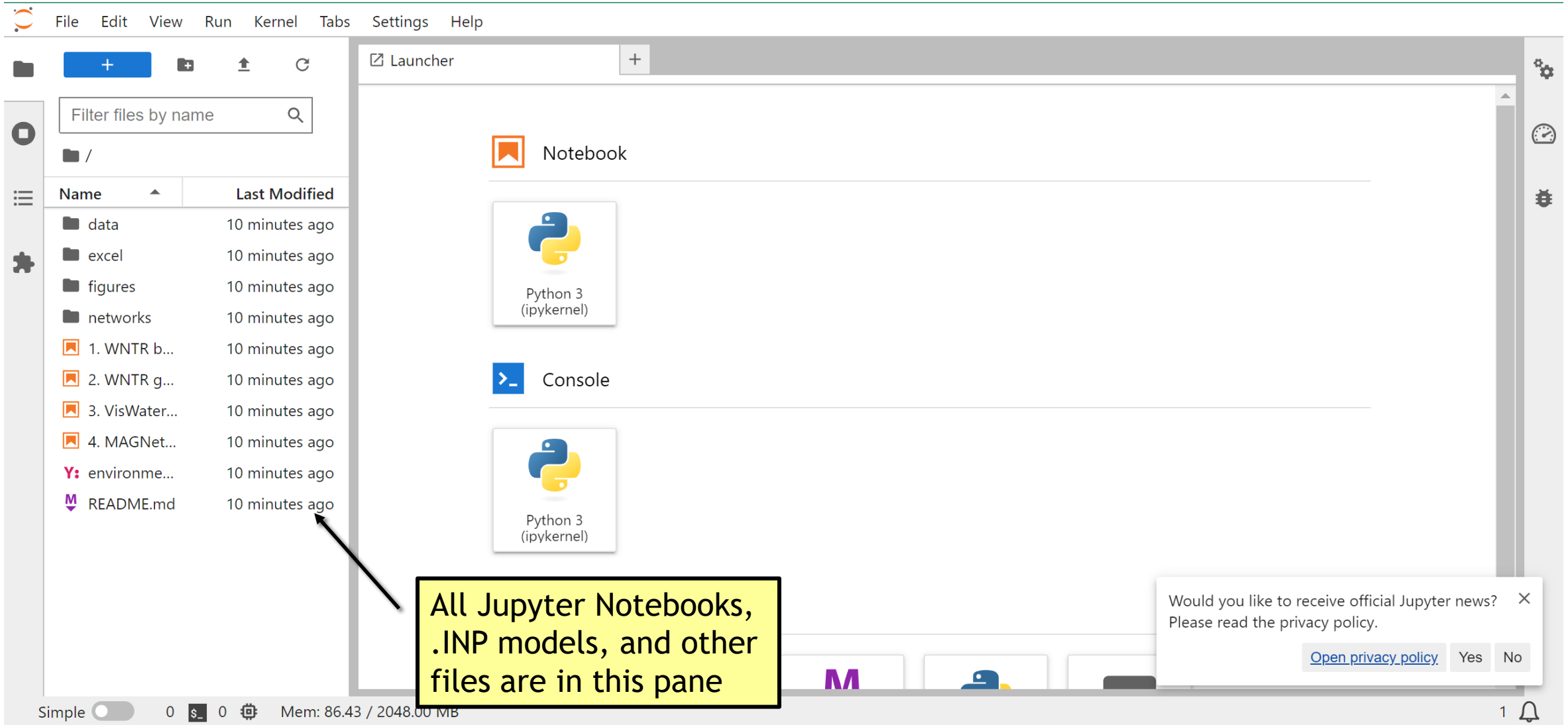


Using the Tutorials

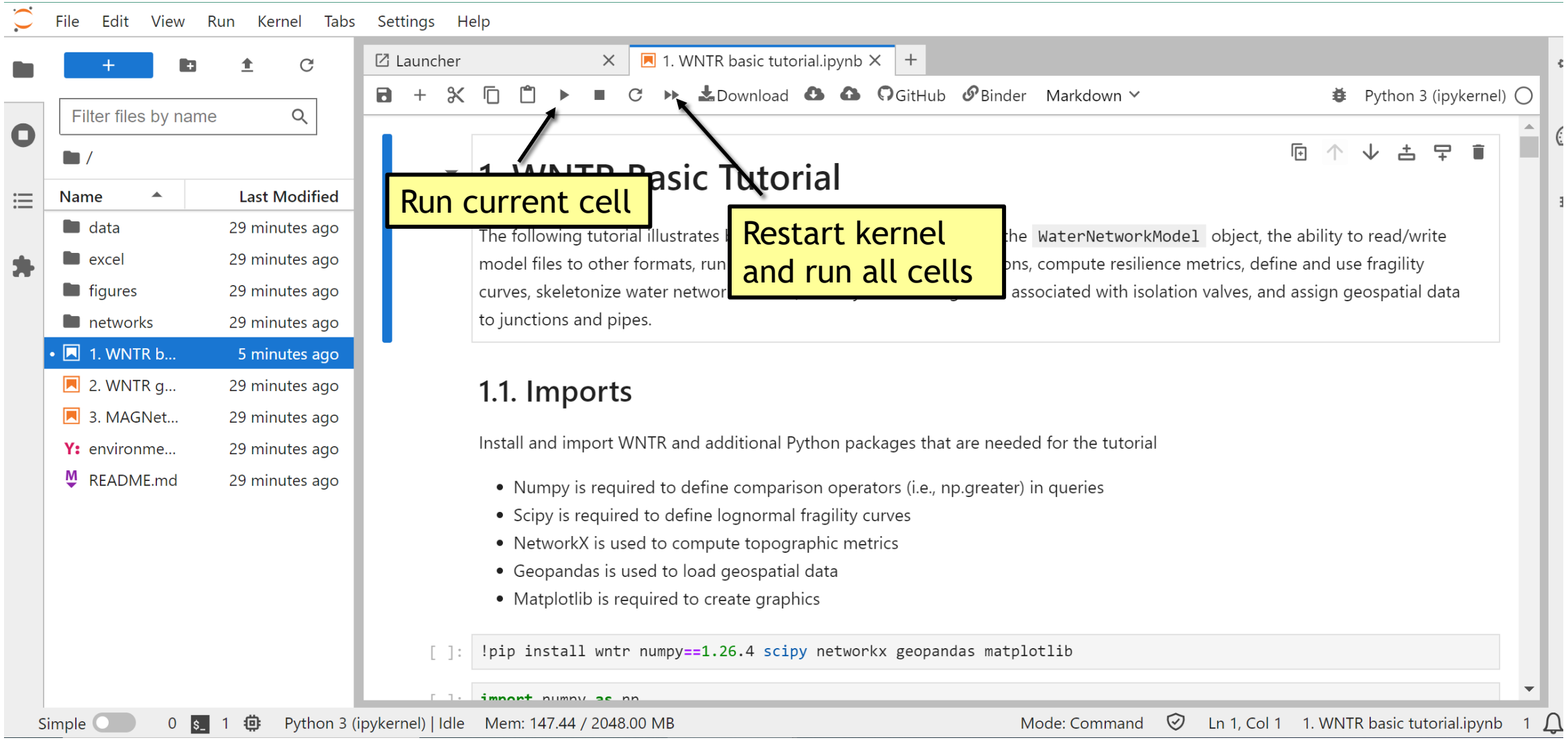
- The Binder might **time out** after 10 minutes of inactivity, and you can expect to see the following dialog box:

- To prevent this happening, we recommend scrolling through your screen or clicking on cells every few minutes
- If the Binder *does* time out, we recommend going to File → Download (if you want to download your modified Jupyter Notebook) and then **retyping the workshop link in the address bar** to restart the session

Using the Tutorials



Using the Tutorials



Using the Tutorials

- You can **make changes** to the Binder but changes are **not saved** locally to the browser
- To **save** your work, **download** the Jupyter Notebook files to your machine before closing the Binder tab
- These Binder files run on **Python 3.9**
- You can always re-open the Binder file in another tab to re-start the session

- You can **download** all the source files from the **Github** repository (tinyurl.com/wdsaccwi2024-github) and run the scripts locally on your machine. You will need to **install all packages** on your machine

WNTR Geospatial Tutorial

Objective:

Quantify water service disruptions that could occur from pipes damaged in landslides. Identify social vulnerability of populations impacted by the service disruptions.

Datasets:

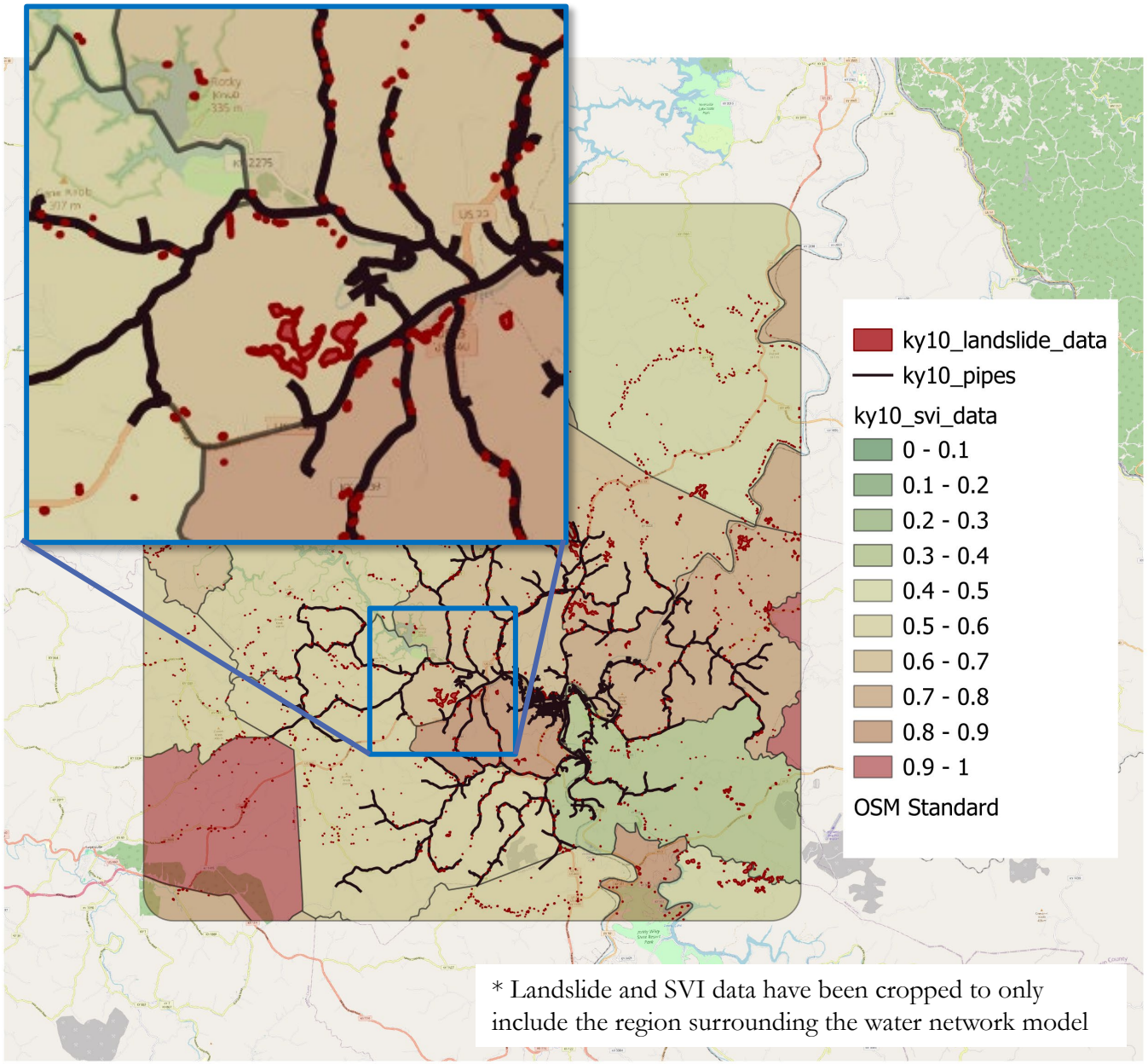
- **Water Network Model ky10.inp** from the UKnowledge Water Distribution Systems Research Database

- **Landslide inventory data** from the UKnowledge Kentucky Geological Survey Research Data

  Landslide inventory provides the locations of known landslides and areas susceptible to debris flows.

- **Social vulnerability index (SVI) data** from the Centers for Disease Control and Prevention/Agency for Toxic Substances and Disease Registry

  SVI includes socioeconomic status, household characteristics, racial and ethnic minority status, and housing type and transportation. The value ranges between 0 and 1, where higher values are associated with higher vulnerability.



* Landslide and SVI data have been cropped to only include the region surrounding the water network model

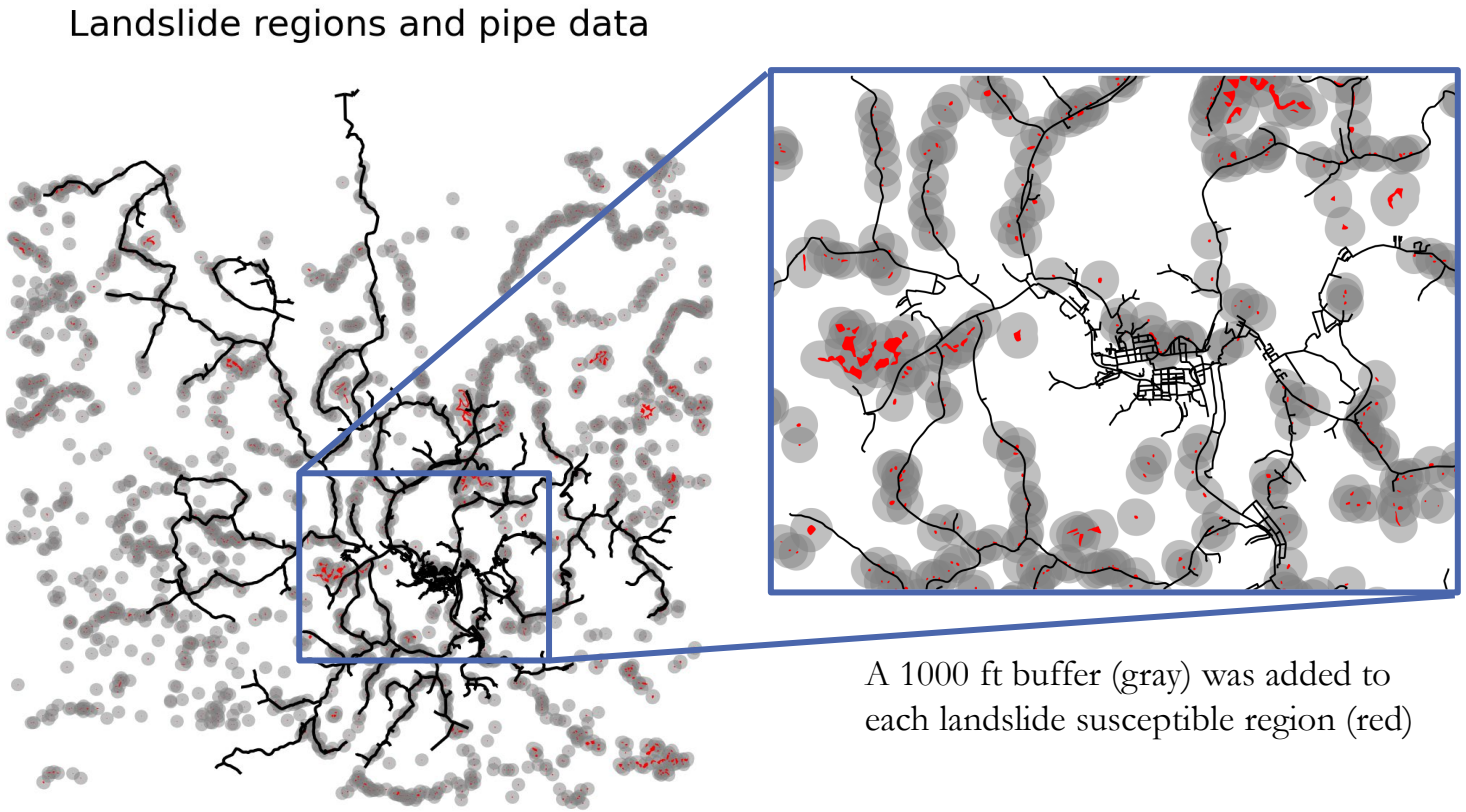
Simplifying Assumptions

To simplify the tutorial, we assume that pipes within a 1000 ft buffer (gray) of each landslide susceptible region (red) are damaged in that landslide.

This assumption could be replaced with results from a detailed landslide analysis that includes slope, soil type, weather conditions, and pipe material.

The probability that pipes are damaged by landslides could also be incorporated into the analysis.

Social vulnerability data could also be replaced by datasets that describe other attributes of the population and critical services.



A 1000 ft buffer (gray) was added to each landslide susceptible region (red)

Analysis Steps

1. Create a water network model object
2. Convert the water network model to a GIS object
3. Load landslide and social vulnerability data
4. Buffer landslides by 1000 ft
5. Intersect the landslide and social vulnerability data with pipes
6. Run a baseline hydraulic simulation
7. Run a hydraulic simulation for each landslide scenario
8. Compute water service availability for each scenario
9. Identify the social vulnerability of populations that experience water service below 50%

Step 1, Create a water network model object

A `WaterNetworkModel` object is created from the ky10.inp file

Network description:

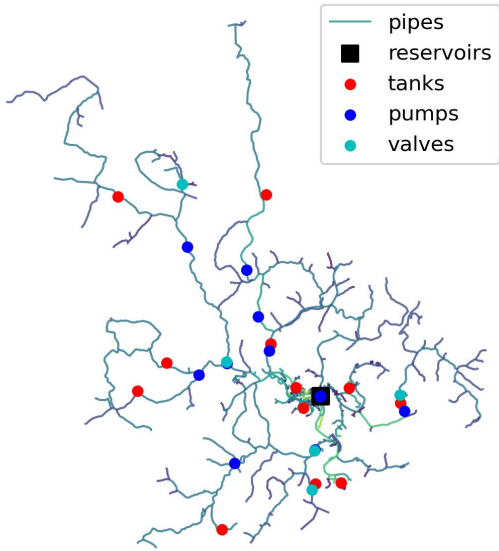
"KY 10 is primarily a branched system in Kentucky with the following assets: 13 Tanks, 13 Pumps, 1 Water Treatment Plant, and approximately 1,353,879 feet of pipe. KY 10 provides 2.26 million gallons of water per day to its 9,093 customers."

```
# Create a water network model
inp_file = 'ky10.inp'
wn = wntr.network.WaterNetworkModel(inp_file)
wn.describe(level=1)
```

```
{'Nodes': {'Junctions': 920, 'Tanks': 13, 'Reservoirs': 2},
 'Links': {'Pipes': 1043, 'Pumps': 13, 'Valves': 5},
 'Patterns': 4,
 'Curves': {'Pump': 0, 'Efficiency': 0, 'Headloss': 0, 'Volume': 0},
 'Sources': 0,
 'Controls': 6}
```

Tutorial includes WNTR functions to compute total pipe length, daily water use, and population.

- Total pipe length = 1,410,846 ft
- Total water use = 2.16 million gallons per day
- Total population = 10,799 assuming 200 gallons per person per day



Step 2, Convert the water network model to a GIS object

The GIS object is a dictionary of GeoDataFrames

- Junctions
- Tanks
- Reservoirs
- Pipes
- Pumps
- Valves

The GIS object does not include controls, patterns, curves, or options

The GIS object can be written to GeoJSON or Shapefiles for use in GIS software platforms

```
# Convert wn to GIS object
wn_gis = wn.to_gis()

# Explore individual GeoDataFrames
wn_gis.junctions
wn_gis.tanks
wn_gis.reservoirs
wn_gis.pipes
wn_gis.pumps
wn_gis.tanks

# Set the coordinate reference system
wn_gis.set_crs('EPSG:3089')

# Write GIS object to GeoJSON files
wn_gis.write_geojson('ky10')
```

The Coordinate Reference System (CRS) is set to geolocate the data. For more information on CRS, see https://epsg.io

First five rows of wn_gis.pipes

	link_type	start_node_name	end_node_name	check_valve	diameter	initial_status	length	minor_loss	roughness	geometry
P-1	Pipe	J-1	T-9	False	0.2032	Open	150.647400	0.0	150.0	LINESTRING (5765080.500 3843715.000, 5765082.1...
P-10	Pipe	J-19	T-3	False	0.1016	Open	508.278079	0.0	150.0	LINESTRING (5773843.490 3813613.000, 5774062.5...
P-100	Pipe	J-170	J-171	False	0.1524	Open	845.762088	0.0	150.0	LINESTRING (5741554.590 3884113.000, 5741551.8...
P-1000	Pipe	J-396	J-138	False	0.1524	Open	1015.611888	0.0	150.0	LINESTRING (5764266.350 3841465.000, 5764332.8...
P-1001	Pipe	J-661	J-793	False	0.0508	Open	65.303400	0.0	150.0	LINESTRING (5778890.430 3833727.000, 5779000.7...

Step 3, Load landslide and social vulnerability data

GeoPandas is used to load the additional GIS data

- GeoJSON format
- Shapefile format
- GeoPackage
- …

Check the CRS to ensure it matches the `wn_gis` object.

If the CRS needs to be changed, use the `to_crs` and `set_crs` methods on the GeoDataFrame.

```
import geopandas as gpd

# Load landslide data
landslide_file = 'ky10_landslide_data.geojson'
landslide_data = gpd.read_file(landslide_file).set_index('index')
print(landslide_data.crs)
```

```
EPSG:3089
```

```
# Load social vulnerability data
svi_file = 'ky10_svi_data.geojson'
svi_data = gpd.read_file(svi_file).set_index('index')
print(svi_data.crs)
```

```
EPSG:3089
```

First 5 rows of landslide_data

index	Type	QUADRANGLE	Confidence_Ranking	Shape_Length	Shape_Area	geometry
12158	Landslide	LANCER	3	600.093659	25776.421953	POLYGON ((5795711.138 3796524.413, 5795749.332...
12154	Landslide	LANCER	3	525.280081	18705.404676	POLYGON ((5793900.026 3796635.524, 5793945.165...
7509	Landslide	Prestonsburg	3	331.655229	7830.582786	POLYGON ((5769026.666 3797014.993, 5769050.971...
7511	Landslide	Prestonsburg	3	518.984567	19344.352333	POLYGON ((5776317.985 3797039.298, 5776324.930...
7507	Landslide	Prestonsburg	3	339.472695	8168.150433	POLYGON ((5767849.582 3797074.020, 5767866.943...

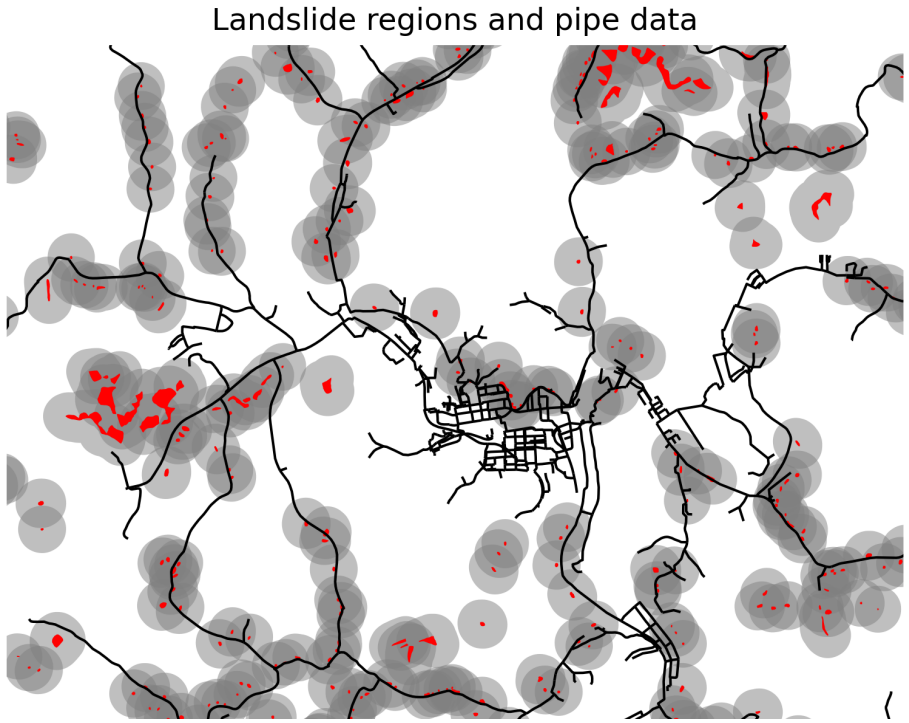
Step 4, Buffer landslides by 1000 ft

Each landslide is extended to include the surrounding 1000 ft, to create a region that might be impacted by an individual landslide.

The distance unit for buffering needs to match the distance unit of the CRS.

Other datasets or methods could be used to define landslide susceptibility or probability of pipe damage.

```
# Create landslide scenarios that include 1000 ft buffer
landslide_scenarios = landslide_data.copy()
landslide_scenarios['geometry'] = landslide_data.buffer(1000)

# Reindex scenarios to start with 'LS-'
landslide_scenarios.index = 'LS-' + landslide_scenarios.index.astype(str)
```



Step 5, Intersect the landslide and social vulnerability data with pipes

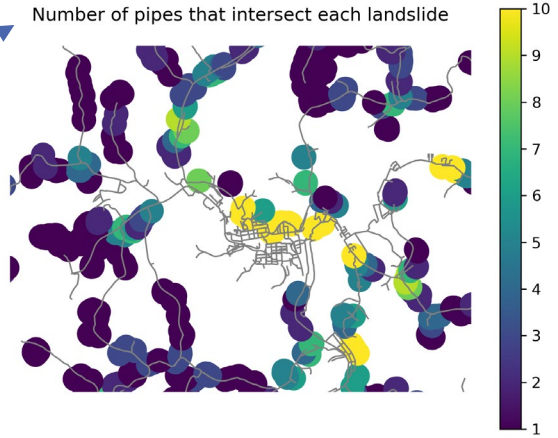
Identify pipes that intersect each landslide polygon

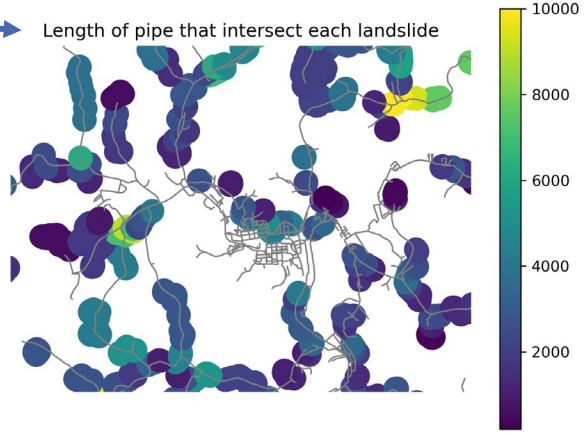
This results in a list of intersecting pipes and statistics on pipe attributes for each landslide.

This information is used to define the pipes that are closed in each landslide scenario

```
A = landslide_scenarios
B = wn_gis.pipes
B_value = 'length'

landslide_intersect = wntr.gis.intersect(A, B, B_value)
```

$A \cap B$ with results indexed by A





First 5 rows of landslide_intersect

	intersections	values	n	sum	min	max	mean
LS-6470	[P-145, P-190, P-389]	[6571.530672, 5059.2590712, 2437.31796]	3	14068.107703	2437.317960	6571.530672	4689.369234
LS-6471	[P-145, P-190, P-389]	[6571.530672, 5059.2590712, 2437.31796]	3	14068.107703	2437.317960	6571.530672	4689.369234
LS-6456	[P-145, P-190, P-389]	[6571.530672, 5059.2590712, 2437.31796]	3	14068.107703	2437.317960	6571.530672	4689.369234
LS-5063	[P-244, P-83]	[12708.2156808, 520.2234960000001]	2	13228.439177	520.223496	12708.215681	6614.219588
LS-5064	[P-244, P-83]	[12708.2156808, 520.2234960000001]	2	13228.439177	520.223496	12708.215681	6614.219588

Step 5, Intersect the landslide and social vulnerability data with pipes

Identify landslide polygons that intersect each pipe

This results in a list of landslides and statistics on landslide attributes for each pipe

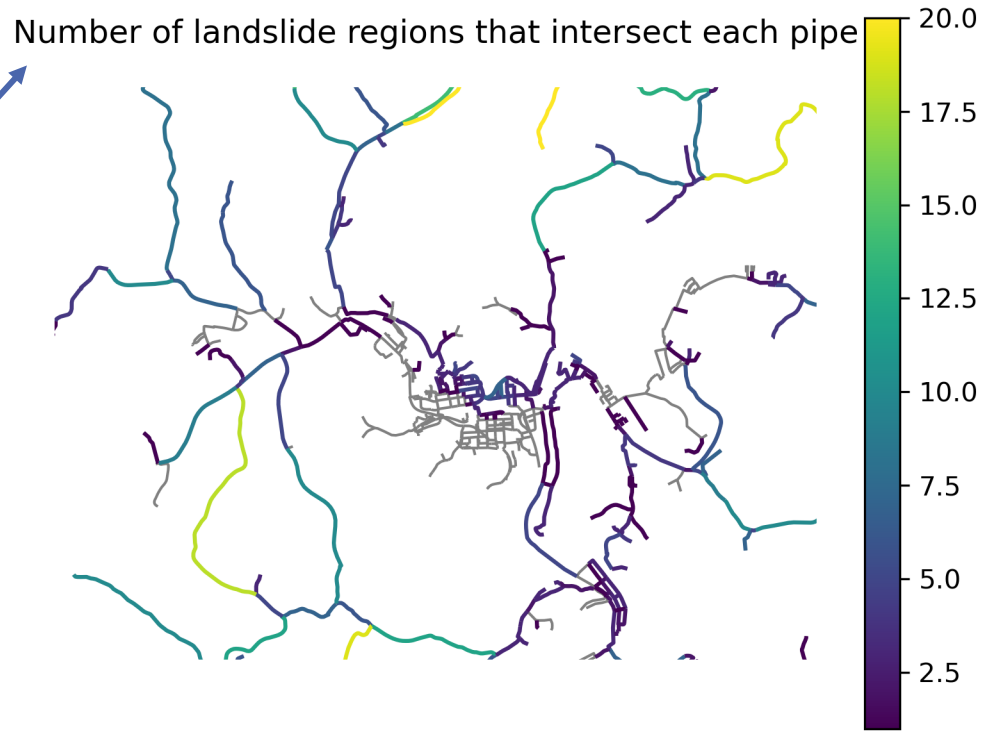
Within the study area, the confidence ranking values are all 3 - Landslide likely at or near the specified location

```
A = wn_gis.pipes
B = landslide_scenarios
B_value = 'Confidence_Ranking'

pipe_intersect = wntr.gis.intersect(A, B, B_value)
```

First 5 rows of pipe_intersect

	intersections	n
P-449	[LS-5086, LS-5087, LS-5088, LS-5125, LS-5126, ...	25
P-517	[LS-4553, LS-4554, LS-4555, LS-4556, LS-4557, ...	23
P-722	[LS-6831, LS-6832, LS-6833, LS-6834, LS-6835, ...	23
P-837	[LS-10295, LS-10307, LS-10308, LS-10309, LS-10...	23
P-442	[LS-6836, LS-6837, LS-6838, LS-6839, LS-6840, ...	20

* Columns related to Confidence_Ranking are not shown (all values are 3)



Step 5, Intersect the landslide and social vulnerability data with pipes

Assign social vulnerability to each junction

This results in a list of intersecting census block tracts for each junction along with the associated SVI value.

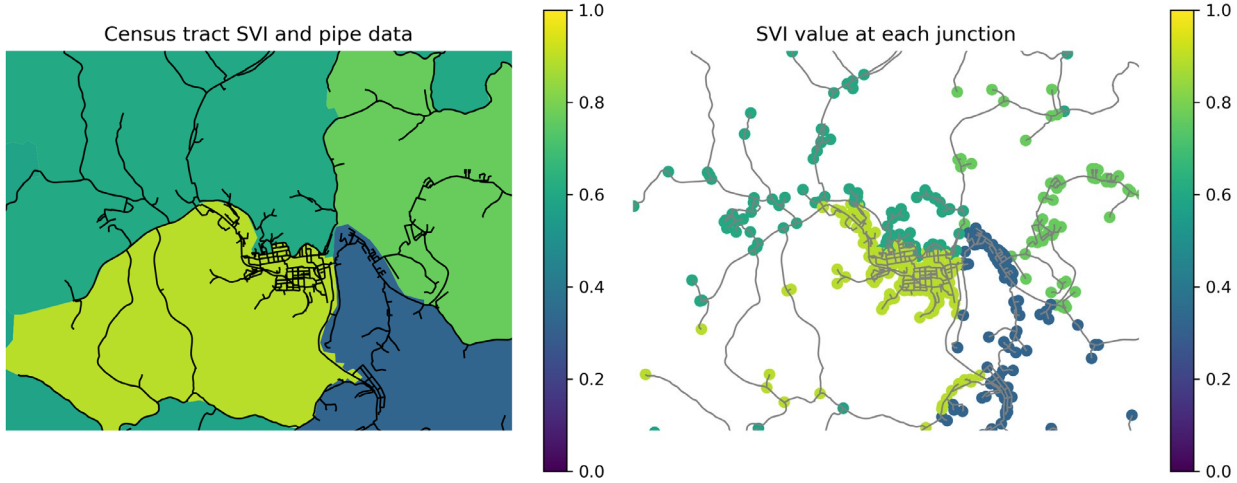
This information is used to determine the social vulnerability of populations that experience water service disruption

Note that each junction intersects 1 census tract

```
A = wn_gis.junctions
B = svi_data
B_value = 'RPL_THEMES'

junction_svi = wntr.gis.intersect(A, B, B_value)
```

First 5 rows of junction_svi

	intersections	values	n	sum	min	max	mean
J-1	[791]	[0.6031]	1	0.6031	0.6031	0.6031	0.6031
J-10	[787]	[0.6015]	1	0.6015	0.6015	0.6015	0.6015
J-100	[791]	[0.6031]	1	0.6031	0.6031	0.6031	0.6031
J-101	[791]	[0.6031]	1	0.6031	0.6031	0.6031	0.6031
J-102	[790]	[0.8942]	1	0.8942	0.8942	0.8942	0.8942



Step 6, Run a baseline hydraulic simulation

A function is created to setup the model for each simulation.

Alternatively, changes to the model can added and reverted for each simulation, however a function call to rebuild the model is often easier and more reliable.

The baseline results are stored in a results object, which includes a node and link dictionary of DataFrames indexed by time (in seconds)

```
baseline_results.node
baseline_results.link
```

```
# Function to setup the model for hydraulic simulations
def model_setup(inp_file):
    wn = wntr.network.WaterNetworkModel(inp_file)
    wn.options.hydraulic.demand_model = 'PDD'
    wn.options.hydraulic.required_pressure = 20 # m
    wn.options.hydraulic.minimum_pressure  = 0 # m
    wn.options.time.duration = 48*3600 # seconds
    return wn

# Run a baseline simulation, with no landslide or damage
wn = model_setup(inp_file)
sim = wntr.sim.EpanetSimulator(wn)
baseline_results = sim.run_sim()
```

```
# View keys in node results dictionary
baseline_results.node.keys()

    dict_keys(['demand', 'head', 'pressure', 'quality'])

# View keys in link results dictionary
baseline_results.link.keys()

    dict_keys(['quality', 'flowrate', 'velocity',
               'headloss', 'status', 'setting',
               'friction_factor', 'reaction_rate'])
```

Step 7, Run a hydraulic simulation for each landslide scenario

- Downselect simulations to reduce duplicates
- Select 6 landslides to run for the tutorial
- Results are stored in a dictionary, with keys being the landslide scenario index (i.e. LS-7003)

Tips for running large case studies:

- Remove simulations results that are not needed for your analysis and/or compute metrics within the for loop to refine results that are saved
- Add `try:except` block to catch simulations that fail to converge or check simulation status in `results.error_code`

```
# Run hydraulic simulations for each landslide scenario
results = {}

for i, scenario in landslide_scenarios.iterrows():
    wn = model_setup(inp_file)

    for pipe_i in scenario['intersections']:
        pipe_object = wn.get_link(pipe_i)
        pipe_object.initial_status = 'CLOSED'

    sim = wntr.sim.EpanetSimulator(wn)
    results[i] = sim.run_sim()
```

First five rows of landslide_scenarios

Original data

	Type	QUADRANGLE	Confidence_Ranking	Shape_Length	Shape_Area
LS-6470	Landslide	Oil Springs	3	505.785629	17813.235857
LS-6471	Landslide	Oil Springs	3	885.366978	46428.920611
LS-6456	Landslide	Oil Springs	3	731.639387	40250.058481
LS-5063	Landslide	SITKA	3	432.545161	12300.359938
LS-5064	Landslide	SITKA	3	374.893710	8458.630241

Buffered geometry

	geometry
LS-6470	POLYGON ((5731632.245 3843127.579, 5731676.251...
LS-6471	POLYGON ((5731968.886 3842927.394, 5731968.570...
LS-6456	POLYGON ((5732906.128 3842944.427, 5732888.288...
LS-5063	POLYGON ((5762828.918 3877978.663, 5762925.073...
LS-5064	POLYGON ((5763551.762 3877868.449, 5763622.783...

Intersection results

	intersections	n	total pipe length
LS-6470	[P-145, P-190, P-389]	3	14068.107703
LS-6471	[P-145, P-190, P-389]	3	14068.107703
LS-6456	[P-145, P-190, P-389]	3	14068.107703
LS-5063	[P-244, P-83]	2	13228.439177
LS-5064	[P-244, P-83]	2	13228.439177

Step 8, Compute water service availability for each scenario

Water service availability (WSA) is the ratio of delivered demand to the expected demand

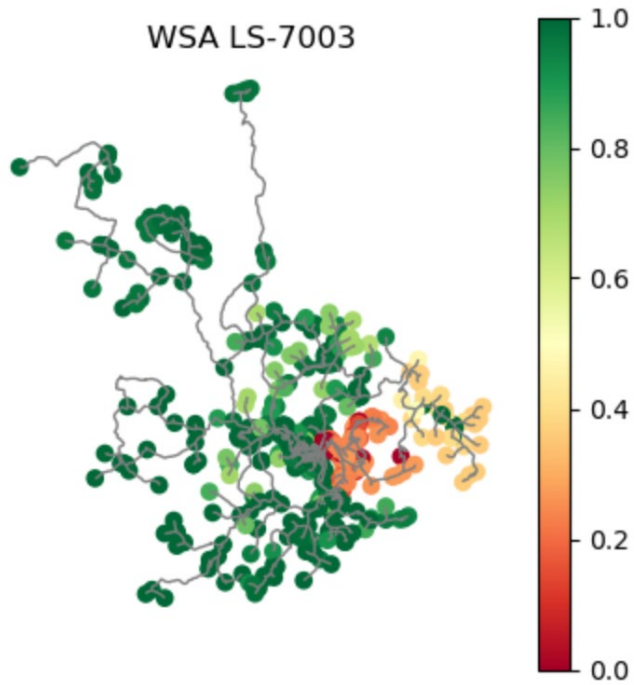
- WSA can be computed for each junction and timestep, or for each junction or each timestep using total demands
- Delivered demand is a result of a PDD hydraulic simulation
- Expected demand can be computed without running a hydraulic simulation and takes into account base demand, demand pattern, and demand multiplier

$$WSA_{j,t} = \frac{demand_{j,t}}{expected\_demand_{j,t}}$$

$$WSA_{j} = \frac{\sum_t demand_{j,t}}{\sum_t expected\_demand_{j,t}}$$

$$WSA_{t} = \frac{\sum_j demand_{j,t}}{\sum_j expected\_demand_{j,t}}$$

```
# Total expected demand at each junction
expected_demand = wntr.metrics.expected_demand(wn)
expected_demand_j = expected_demand.sum(axis=0)

# Total delivered demand at each junction (in this case, for scenario 7003)
demand = results['7003'].node['demand'].loc[:,wn.junction_name_list]
demand_j = demand.sum(axis=0)

# WSA for each junction
wsa_j = wntr.metrics.water_service_availability(expected_demand_j, demand_j)
```



Step 9, Identify the social vulnerability of impacted population

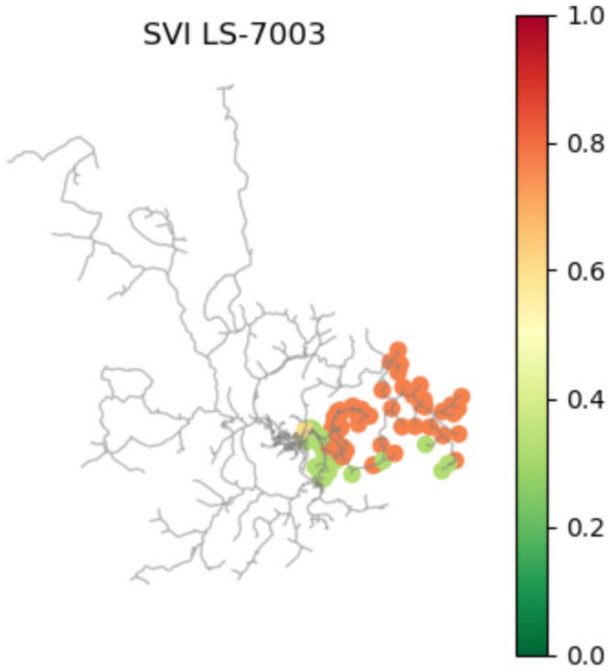
- Impacted junctions are defined as junctions where WSA falls below 0.5 (50% of the expected water was received) at any time during the simulation
- Create a filter which identifies junctions that have WSA below 0.5
- Extract SVI of impacted junctions

```
# Filter that identifies WSA < 0.5
impacted_junctions = {}
for scenario in wsa_results.columns:
    filter = wsa_results[scenario] < 0.5
    impacted_junctions[scenario] = wsa_results.index[filter]
```

First five rows of wsa_results

	LS-4495	LS-7003	LS-7111	LS-5086	LS-6966	LS-7058
J-1	1.000879	0.955301	1.000744	1.000750	0.222644	0.991877
J-10	1.000278	0.953388	0.986847	0.224127	0.746386	0.983880
J-100	1.000114	1.000079	1.000106	1.000099	0.271720	1.000080
J-101	1.000097	0.892301	1.000086	0.994608	0.139309	0.952506
J-102	1.003433	1.002577	1.003204	1.003148	1.003491	1.003377

Social vulnerability index (SVI) of impacted junctions



Agenda

- Introductions: 5 min
- Using the Tutorials: 10 min
- Overview: 15 min
- WNTR Tutorial: 60 min
- **Break: 15 min**
- MAGNets and VisWaterNet Tutorial: 45 min
- Break: 15 min
- Future Capabilities and Discussion: 45 min

Agenda

- Introductions: 5 min
- Using the Tutorials: 10 min
- Overview: 15 min
- WNTR Tutorial: 60 min
- Break: 15 min
- **MAGNets and VisWaterNet Tutorial: 45 min**
- Break: 15 min
- Future Capabilities and Discussion: 45 min

MAGNets and VisWaterNet Tutorial

Introducing the case study

Network visualizations with **VisWaterNet**

Network skeletonization with **MAGNets**

MAGNets and VisWaterNet Tutorial

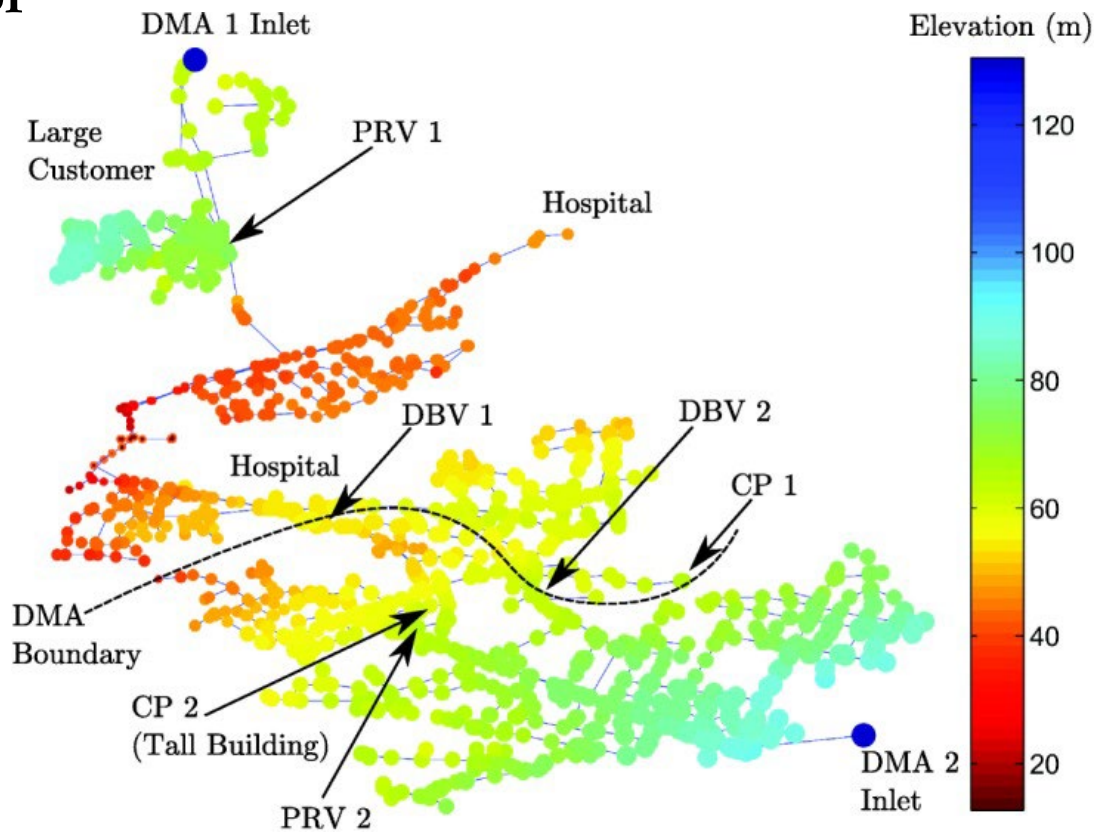
Introducing the case study

Network visualizations with **VisWaterNet**

Network skeletonization with **MAGNets**

Case study

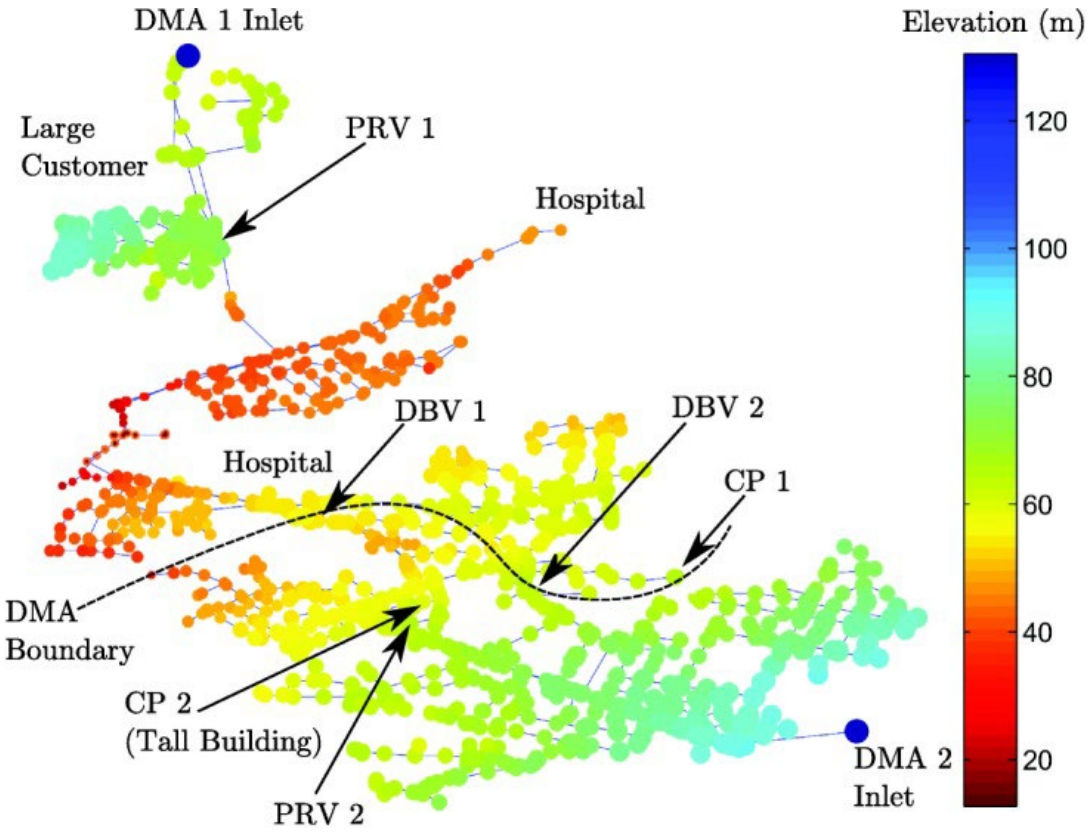
- **BWFLnet** is an operational network in Bristol, UK, operated by Bristol Water.
- Product of a long term partnership between **Bristol Water** and **InfraSense Labs** at Imperial College London.
- **Publicly** available data doi:10.17632/srt4vr5k38.1
- **BWFLnet hydraulic model:**
  - 2546 junctions
  - 2601 pipes
  - 3 pressure reducing valves (PRVs)
  - 2 pressure sustaining valves (PSVs)



Waldron, Alexander; Pecci, Filippo; Stoianov, Ivan (2019), "BWFLnet + data", Mendeley Data, V1, doi: 10.17632/srt4vr5k38.1

Case study

- **Additional info:**
  - 2 DMAs with different demand patterns
  - 15 junctions with unique demand patterns
  - Pressure data collected from 27 locations in the network
- **Changes to the original .INP:**
  - Identical demand patterns were merged
  - Replaced throttle control valves (TCVs) with equivalent pipes
- Modified .INP file is included in the Binder
- **Goal: visualize and simplify this model**



Waldron, Alexander; Pecci, Filippo; Stoianov, Ivan (2019), "BWFLnet + data", Mendeley Data, V1, doi: 10.17632/srt4vr5k38.1

MAGNets and VisWaterNet Tutorial

Introducing the case study

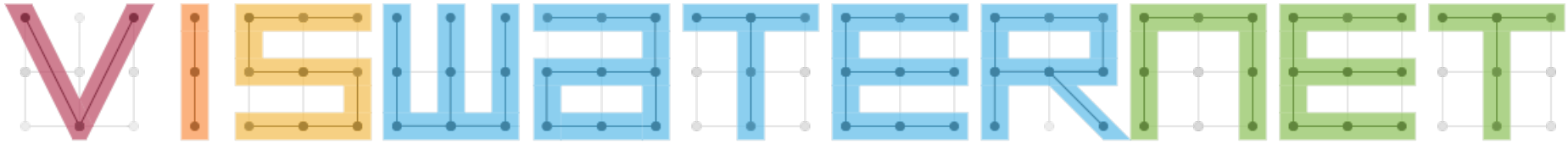
Network visualizations with **VisWaterNet**

Network skeletonization with **MAGNets**

Network Visualization

- We would like a wide range of visualization capabilities such as:
  - Visualizing **user-generated data** stored in Python lists or Excel files
  - Displaying **multiple metrics** on the same plot
  - Visualizing **categorical** data
  - Customizing **legends**
  - **Highlighting** and **labeling** specific nodes or links
  - Customizing node and link **styles** (e.g., node borders, link line styles)

Network Visualization with VisWaterNet



A Python package for Visualization of Water distribution Networks

tylertrimble/viswaternet

viswaternet.readthedocs.io

WNTR
Water Network Tool for Resilience

EPANET

NetworkX
Network Analysis in Python

matplotlib

python

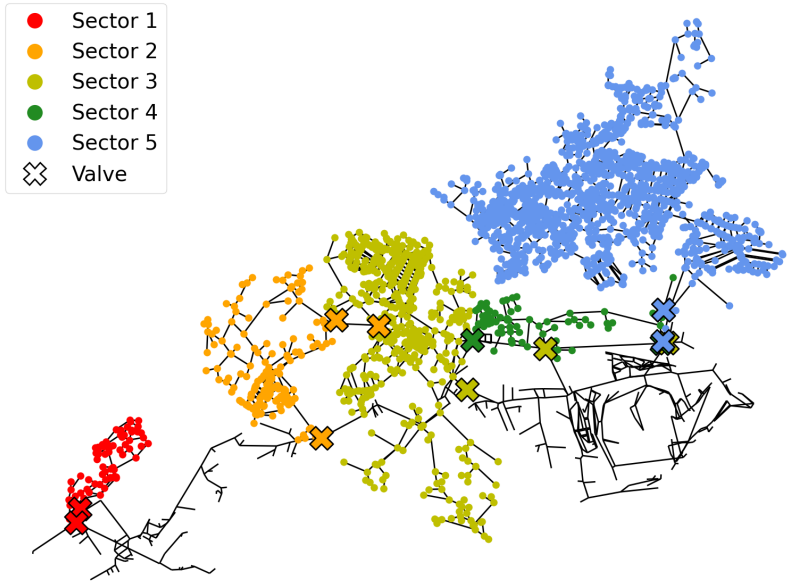
Thomas, M., Trimble, T., & Sela, L. (2023). VisWaterNet: A Python package for visualization of water distribution networks. *Journal of Open Source Software*, *8*(84), 5139. DOI: 10.21105/JOSS.05139.

Network Visualization with VisWaterNet

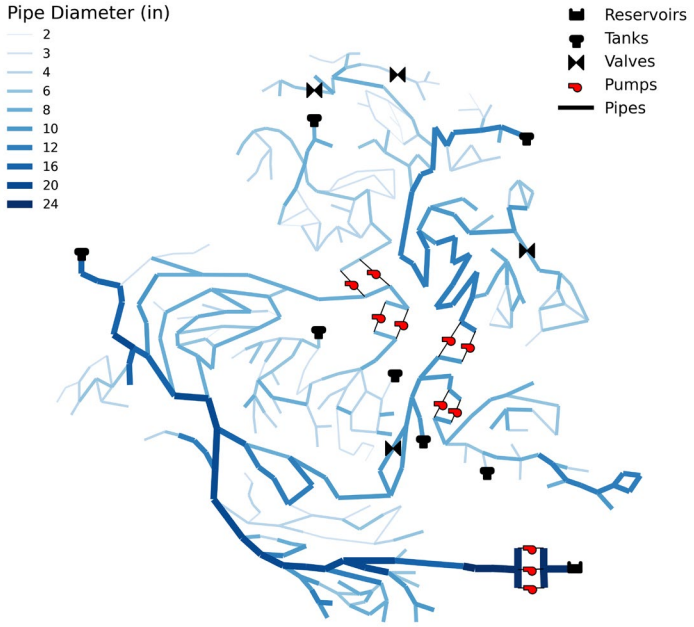
Category	Function
Initializing the VisWaterNet model	VisWNModel
Customize plot appearance	NetworkStyle
Comprehensive plotting functions	plot_basic_elements
	plot_continuous_nodes
	plot_continuous_links
	plot_discrete_nodes
	plot_discrete_links
	plot_unique_data
	animate_plot
Additional plotting functions	draw_labels
	draw_nodes
	draw_links

Example Visualizations

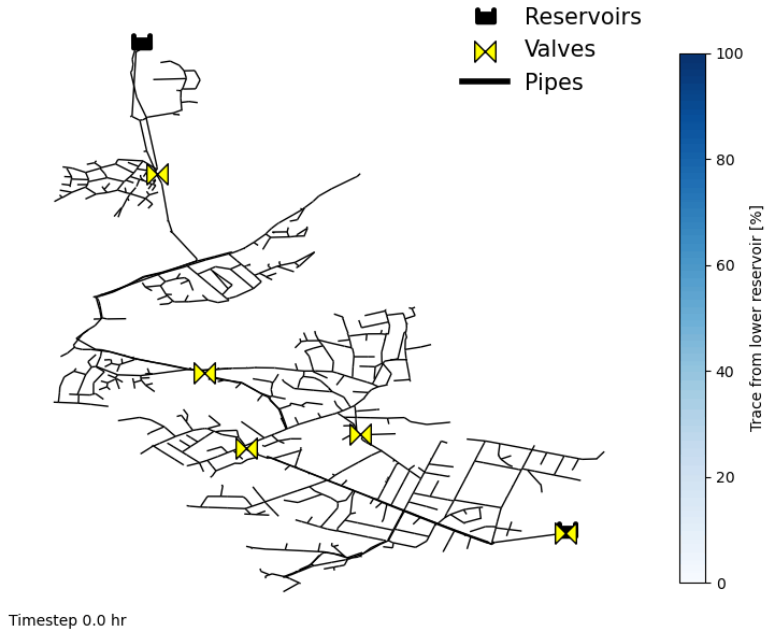
DMAs



Pipe diameters



Trace analysis



1. Getting Started

- The presentation mirrors the scripts in the Binder files **3. VisWaterNet Tutorial**
- **1.1 Install** packages

```
!pip install wntr matplotlib numpy magnets viswaternet
```

- **1.1 Import** packages into the Jupyter Notebook file

```
#import packages
import wntr
import matplotlib.pyplot as plt
import numpy as np
import time
```

- **1.2 Define** the .INP file and initialize WNTR water network model object

```
# import bwfl
inp = 'networks/BWFLnet_modified.inp'

wn = wntr.network.WaterNetworkModel(inp)
```

1. Getting Started

- **1.3** This network has several **critical nodes**. We define lists of junction IDs:
  - Locations of **pressure sensors**
  - Nodes with **unique demand patterns**
  - **Neighboring** nodes of nodes with unique demand patterns

```
#list of junctions where pressure sensors are installed
pressure_sensors_list = ['node_1925', 'node_1938', 'node_1961', 'node_1971',
                         'node_1983', 'node_0354', 'node_2012', 'node_2046',
```

```
#list of junctions with different demand patterns ...
diff_demand_pattern_list = ['node_0640', 'node_0870', 'node_0973', 'node_1122', 'node_1311',
                            'node_1363', 'node_1388', 'node_1685', 'node_1813', 'node_2125',
```

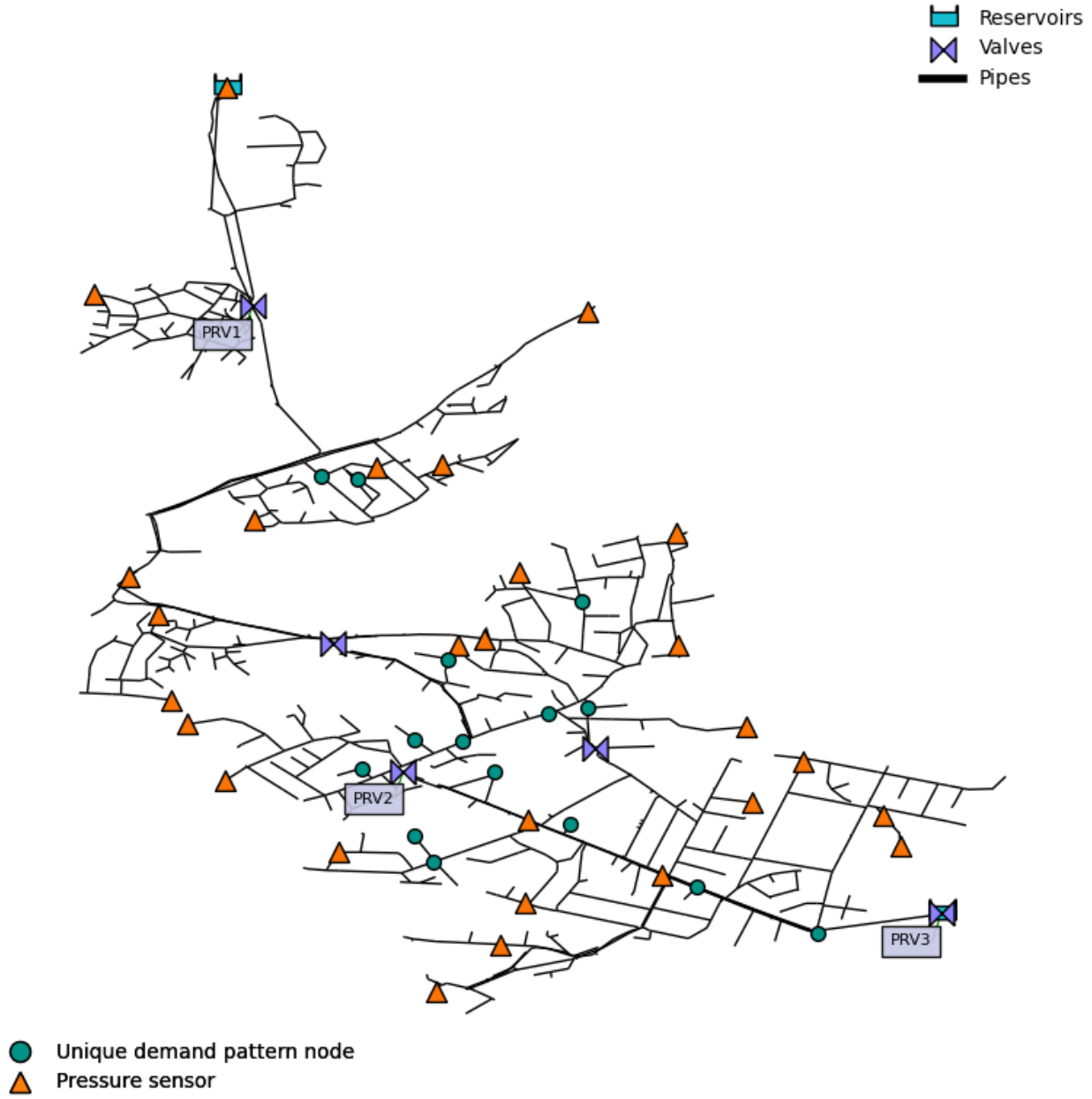
```
# ... and their neighbors
diff_demand_neighbor_list = ['node_0595', 'node_0647', 'node_0648', 'node_2168', 'node_2152',
                             'node_0907', 'node_0952', 'node_2199', 'node_2206', 'node_1121',
```

```
critical_nodes = pressure_sensors_list + diff_demand_pattern_list + diff_demand_neighbor_list
```

2. Visualizing the Network Model with VisWaterNet

Here, we:

1. **Plot** the basic network layout
2. **Highlight** the location of critical nodes
3. **Label** the PRVs
4. Build a **custom legend**



2.1. Visualizing the Network Model with VisWaterNet

- Use the `plot_basic_elements` function to **plot** the network layout
- Use the `draw_nodes` function to **highlight** the location of critical nodes
- Use the `draw_label` function to **label** the PRVs

Step 0: Use pip to install VisWaterNet

```
pip install viswaternet
```

Step 1: Import VisWaterNet into script

```
import viswaternet as vis
```

Step 2: Initialize VisWaterNet model object from .INP file

```
model=vis.VisWNModel(inp)
```

Step 3: Initialize a matplotlib figure (optional)

```
fig, ax = plt.subplots(figsize=(8,8))
ax.set_frame_on(False)
```

2.2. Visualizing the Network Model with VisWaterNet

Step 4: Define the plot **style**

```
style1 = vis.NetworkStyle(base_node_size=0, reservoir_size = 200, valve_size = 150,
                          base_legend_label_font_size = 10,
                          reservoir_border_width = 1, reservoir_color='tab:cyan',
                          valve_color='xkcd:periwinkle')
```

Change size of elements

Adjust legend

Customize markers

Step 5: **Plot** the network layout

Step 6: Highlight pressure sensors and nodes with unique demand patterns

Step 7: Add labels to PRVs

Step 8: Build a custom legend

2.2. Visualizing the Network Model with VisWaterNet

Step 4: Define the plot **style**

```
style1 = vis.NetworkStyle(base_node_size=0, reservoir_size = 200, valve_size = 150,
                          base_legend_label_font_size = 10,
                          reservoir_border_width = 1, reservoir_color='tab:cyan',
                          valve_color='xkcd:periwinkle')
```

Step 5: **Plot** the network layout

```
model.plot_basic_elements(ax=ax, style=style1)
```

Specify matplotlib axis

Specify plot style

Step 6: Highlight pressure sensors and nodes with unique demand patterns

Step 7: Add labels to PRVs

Step 8: Build a custom legend

2.2. Visualizing the Network Model with VisWaterNet

Step 4: Define the plot **style**

```
style1 = vis.NetworkStyle(base_node_size=0, reservoir_size = 200, valve_size = 150,
                          base_legend_label_font_size = 10,
                          reservoir_border_width = 1, reservoir_color='tab:cyan',
                          valve_color='xkcd:periwinkle')
```

Step 5: **Plot** the network layout

```
model.plot_basic_elements(ax=ax, style=style1)
```

Step 6: **Highlight** pressure sensors and nodes with unique demand patterns

```
model.draw_nodes(ax=ax, node_list=pressure_sensors_list, node_color='xkcd:orange',
                 node_shape='^', node_border_width=1, node_border_color='k')
```

Specify list of nodes to highlight

Customize markers

Step 7: Add labels to PRVs

Step 8: Build a custom legend

2.2. Visualizing the Network Model with VisWaterNet

Step 4: Define the plot **style**

```
style1 = vis.NetworkStyle(base_node_size=0, reservoir_size = 200, valve_size = 150,
                          base_legend_label_font_size = 10,
                          reservoir_border_width = 1, reservoir_color='tab:cyan',
                          valve_color='xkcd:periwinkle')
```

Step 5: **Plot** the network layout

```
model.plot_basic_elements(ax=ax, style=style1)
```

Step 6: **Highlight** pressure sensors and nodes with unique demand patterns

```
model.draw_nodes(ax=ax, node_list=pressure_sensors_list, node_color='xkcd:orange',
                 node_shape='^', node_border_width=1, node_border_color='k')
```

Step 7: Add **labels** to PRVs

```
model.draw_label(ax=ax, draw_nodes=prv_start_node_list, labels=['PRV1', 'PRV2', 'PRV3'],
                 x_coords=[-1000]*len(prv_list), y_coords=[-3000]*len(prv_list),
                 label_font_size = 8, label_face_color = '#c6c8e4ff')
```

Specify elements to label

Specify labels

Define offsets from element location

Customize label

Step 8: Build a custom legend

2.2. Visualizing the Network Model with VisWaterNet

Step 4: Define the plot **style**

```
style1 = vis.NetworkStyle(base_node_size=0, reservoir_size = 200, valve_size = 150,
                          base_legend_label_font_size = 10,
                          reservoir_border_width = 1, reservoir_color='tab:cyan',
                          valve_color='xkcd:periwinkle')
```

Step 5: **Plot** the network layout

```
model.plot_basic_elements(ax=ax, style=style1)
```

Step 6: **Highlight** pressure sensors and nodes with unique demand patterns

```
model.draw_nodes(ax=ax, node_list=pressure_sensors_list, node_color='xkcd:orange',
                 node_shape='^', node_border_width=1, node_border_color='k')
```

Step 7: Add **labels** to PRVs

```
model.draw_label(ax=ax, draw_nodes=prv_start_node_list, labels=['PRV1', 'PRV2', 'PRV3'],
                 x_coords=[-1000]*len(prv_list), y_coords=[-3000]*len(prv_list),
                 label_font_size = 8, label_face_color = '#c6c8e4ff')
```

Step 8: Build a **custom legend**

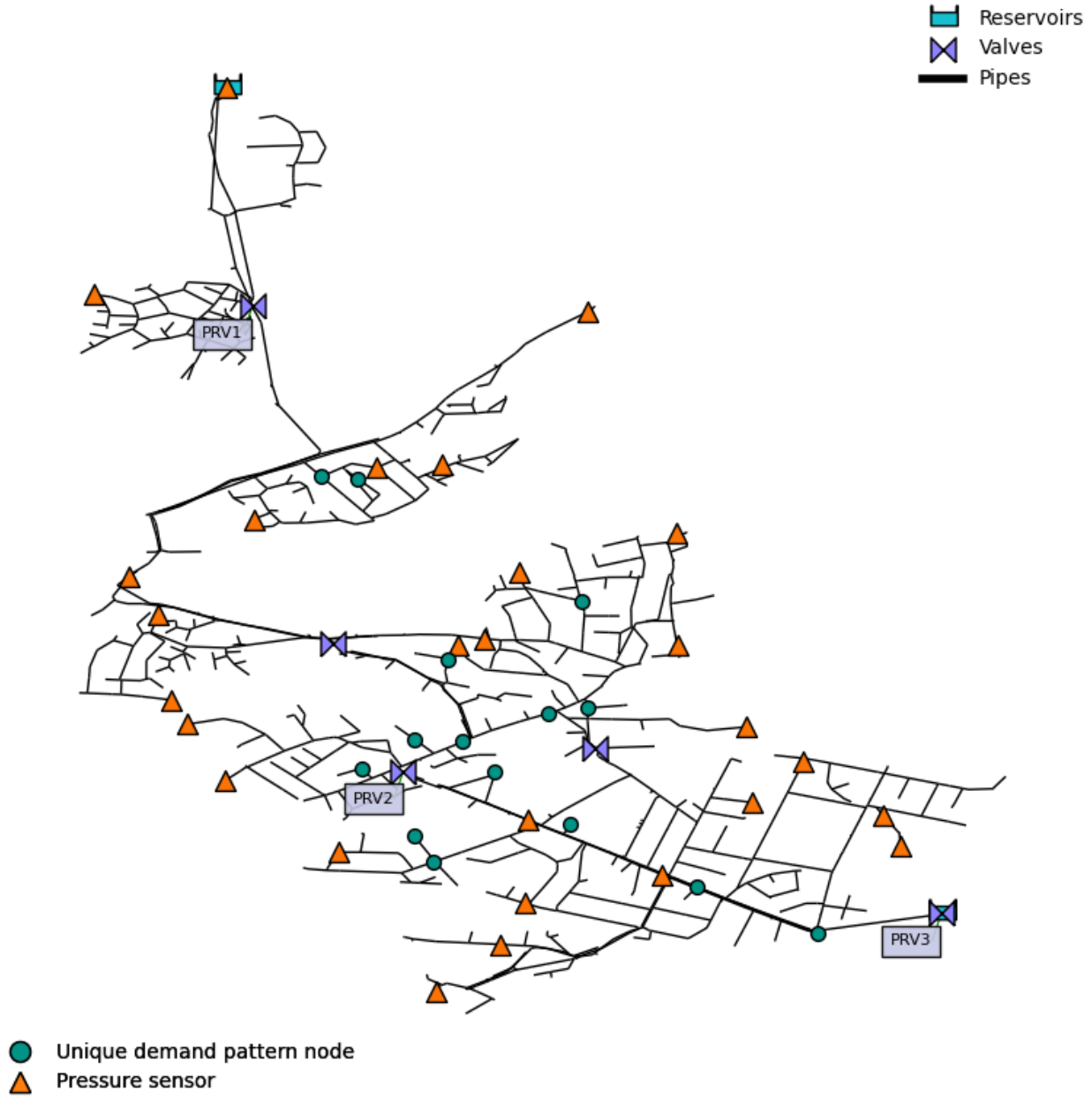
```
from matplotlib.lines import Line2D
custom_lines = [Line2D([0], [0], color='tab:orange',
                       marker='*', markersize=10, lw=0),
                Line2D([0], [0], color='tab:green',
                       marker='^', markersize=10, lw=0)]
leg_labels = ['Pressure sensor', 'Different demand pattern node']
legend = ax.legend(custom_lines, leg_labels, fontsize=10,  loc=3, frameon=False)
ax.add_artist(legend)
```

Use matplotlib to build custom legend for highlighted nodes

2.2. Visualizing the Network Model with VisWaterNet

Here, we:

1. **Plot** the basic network layout
2. **Highlight** the location of critical nodes
3. **Label** the PRVs
4. Build a **custom legend** using matplotlib



2.3. Visualizing Pressure at Nodes

- Use the `plot_discrete_nodes` function to plot **nodal pressures** from WNTR simulation results.

Define plot style

```
style2 = vis.NetworkStyle(discrete_legend_loc = 'upper right',
                          discrete_legend_label_font_size = 9,
                          discrete_legend_title_font_size = 11,
                          base_legend_loc = 'lower left',
                          base_legend_label_font_size=10)

model.plot_discrete_nodes(ax=ax,
                          parameter = 'pressure', value = 'mean',
                          style = style2)
```

Customize data legend

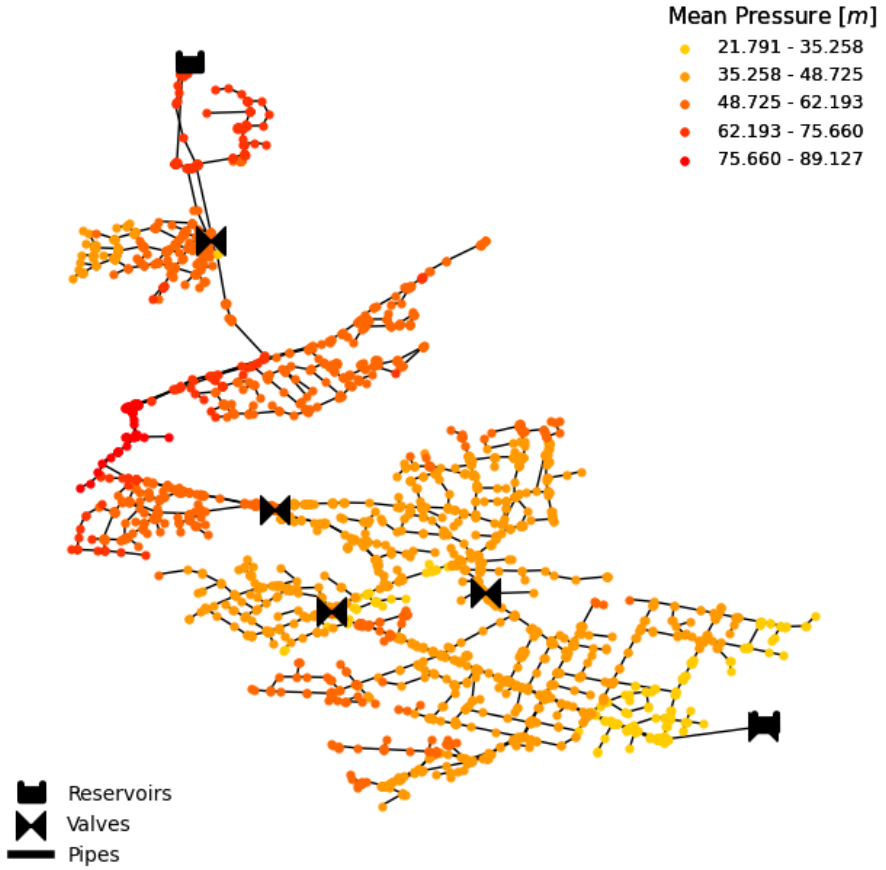
Customize base legend

Plot mean pressure

2.3. Visualizing Pressure at Nodes

- Use the `plot_discrete_nodes` function to plot **nodal pressures** from WNTR simulation results. Default: four equal intervals.

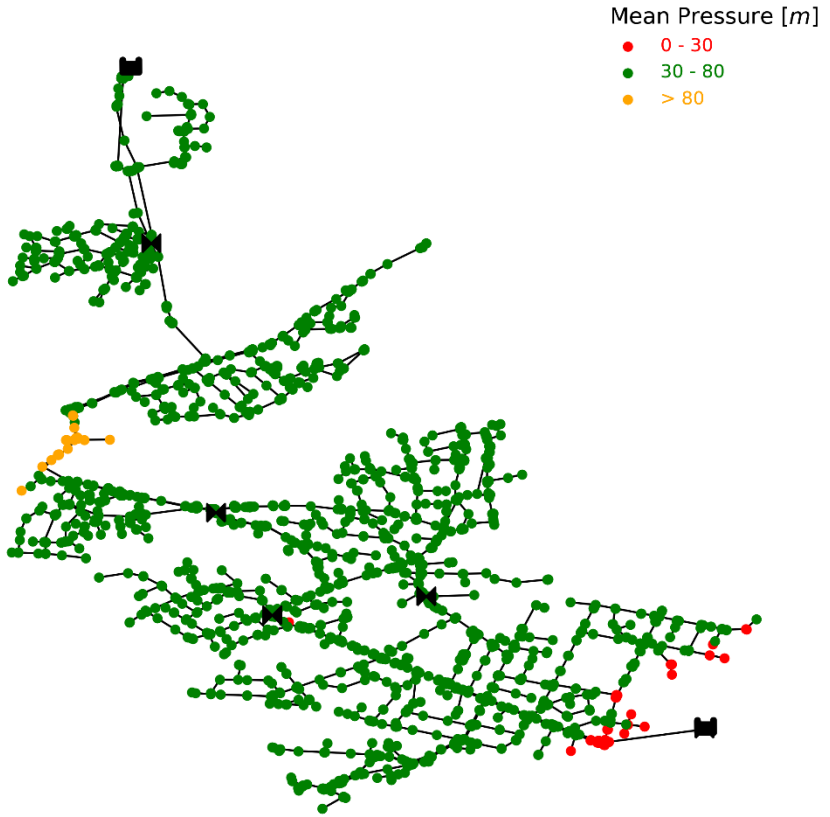
```
style2 = vis.NetworkStyle(discrete_legend_loc = 'upper right',
                          discrete_legend_label_font_size = 9,
                          discrete_legend_title_font_size = 11,
                          base_legend_loc = 'lower left',
                          base_legend_label_font_size=10)

model.plot_discrete_nodes(ax=ax,
                          parameter = 'pressure', value = 'mean',
                          style = style2)
```



2.3. Visualizing Pressure at Nodes

- **Customize** `plot_discrete_nodes` arguments to use specified intervals.

```
style3 = vis.NetworkStyle(color_list = ['red', 'green', 'orange'],
                          discrete_legend_loc = 'upper right',
                          discrete_legend_label_color = 'interval_color',
                          discrete_legend_label_font_size = 9,
                          discrete_legend_title_font_size = 11,
                          legend_decimal_places = 0,
                          draw_base_legend = False,
                          valve_size=80, reservoir_size=80, node_size=50)

model.plot_discrete_nodes(ax=ax, parameter='pressure', value = 'mean',
                          intervals = [0,30,80], style = style3)
```



2.4. Visualizing User-Generated Data

- Use the `plot_unique_data` function to:
  - Represent **user-generated data**
  - Example: plot pipe material (**categorical data)** from **Excel**

```
style4 = vis.NetworkStyle(cmap='tab20', draw_base_legend = False,
                          discrete_legend_loc = 'upper right',
                          discrete_legend_label_font_size=12,
                          link_width=2)

model.plot_unique_data(parameter = 'excel_data', parameter_type = 'link',
                          data_file = 'excel/bwflnet_materials.xlsx',
                          excel_columns = [0,1],
                          data_type = 'unique',
                          style = style4)
```

Extract data from Excel file

The data corresponds to links

Path to Excel file

Categorical data is ‘unique’

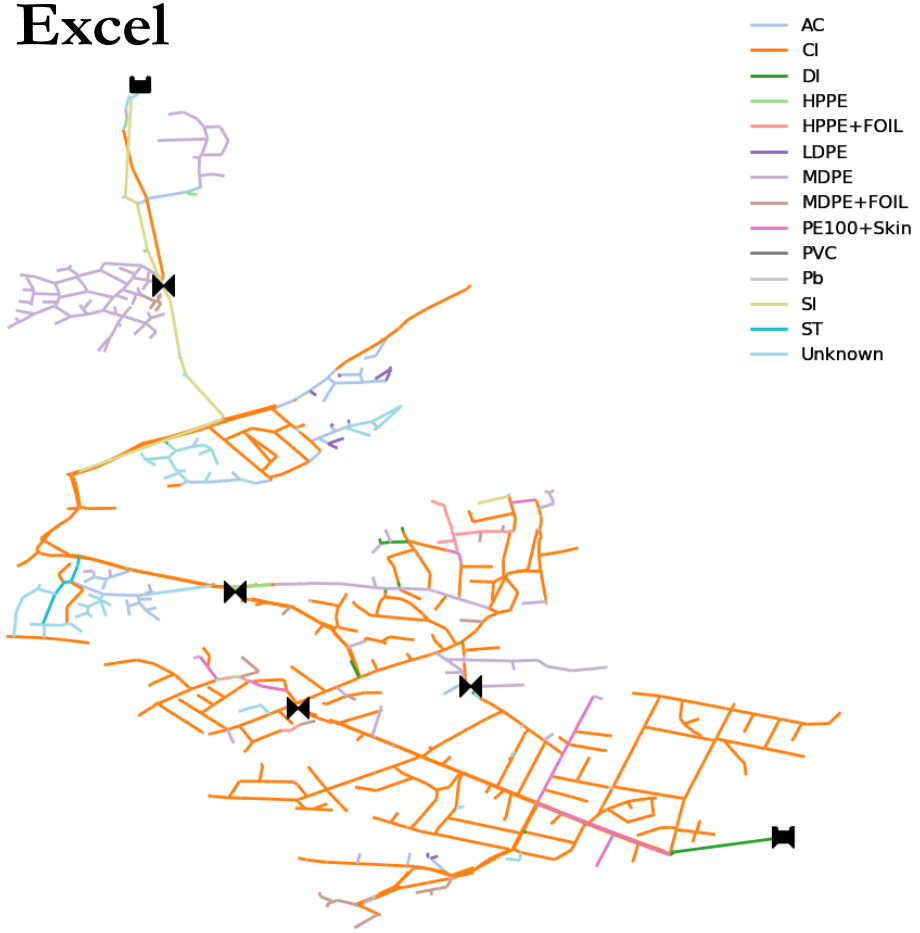
Excel column indices for
link names (0)
corresponding data (1)

	A	B
1	link_name	material
2	link_0001	CI
3	link_0002	Unknown
4	link_0003	CI
5	link_0004	Unknown
6	link_0005	Unknown
7	link_0006	MDPE

2.4. Visualizing User-Generated Data

- Use the `plot_unique_data` function to:
  - Represent **user-generated data**
  - Example: plot pipe material (**categorical data)** from **Excel**

```
style4 = vis.NetworkStyle(cmap='tab20', draw_base_legend = False,
                          discrete_legend_loc = 'upper right',
                          discrete_legend_label_font_size=12,
                          link_width=2)

model.plot_unique_data(parameter = 'excel_data', parameter_type = 'link',
                          data_file = 'excel/bwflnet_materials.xlsx',
                          excel_columns = [0,1],
                          data_type = 'unique',
                          style = style4)
```



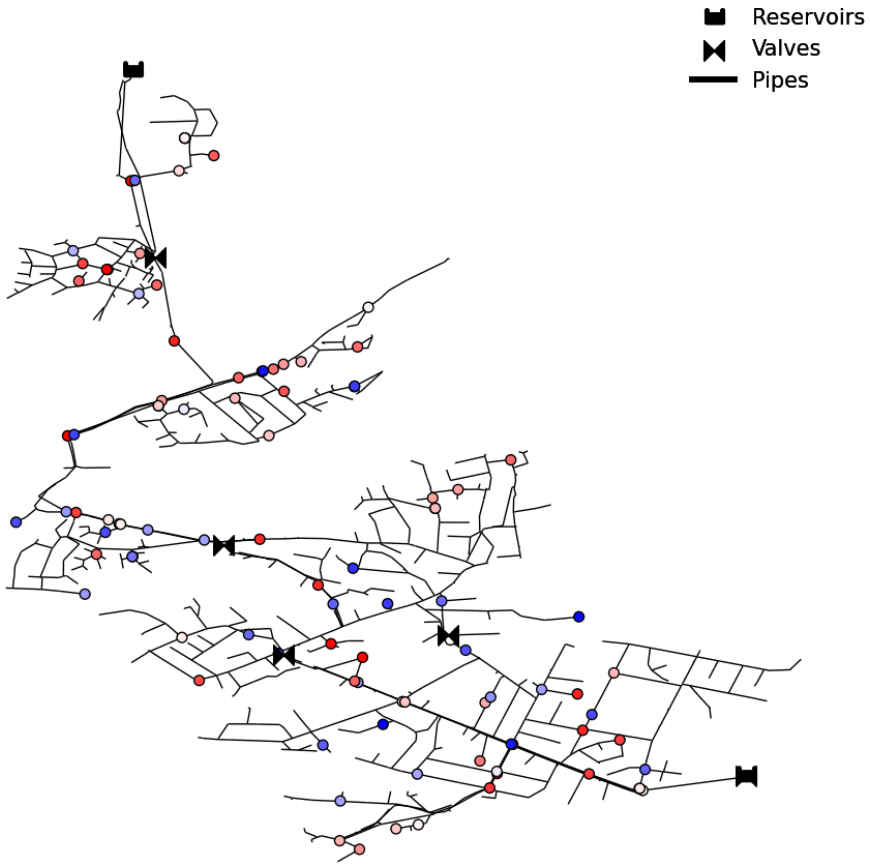
2.5. Visualizing User-Generated Data

- Use the `plot_unique_data` function to represent user-supplied data. Here, we
  - Access data from a **list**
  - Show estimate errors at a **subset of junctions**

```
style5 = vis.NetworkStyle(node_size = 200,
                          node_border_width = 1, node_border_color = "k",
                          color_bar_title = "Error (%)", cmap = "bwr")

model.plot_unique_data(parameter = "custom_data",
                       parameter_type = "node", data_type = "continuous",
                       custom_data_values = [element_list, diff_list],
                       style = style5)
```

List of junctions

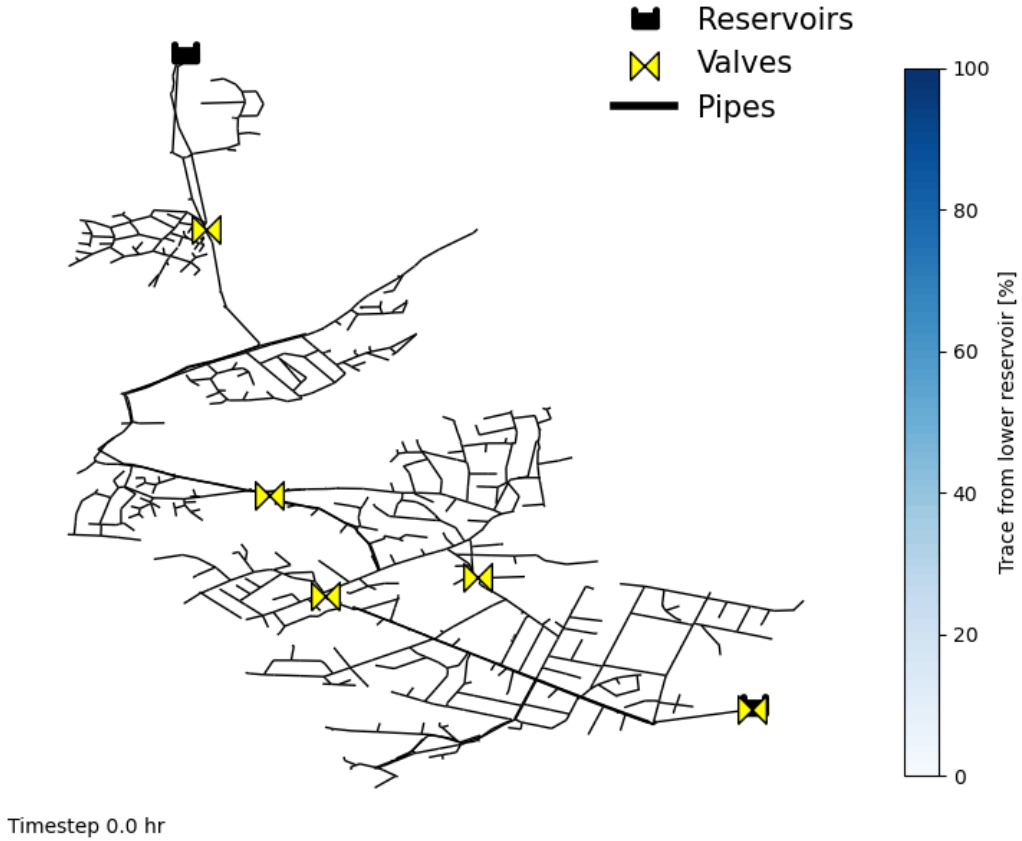
List of corresponding data



2.6. Visualizing Time-Varying Data

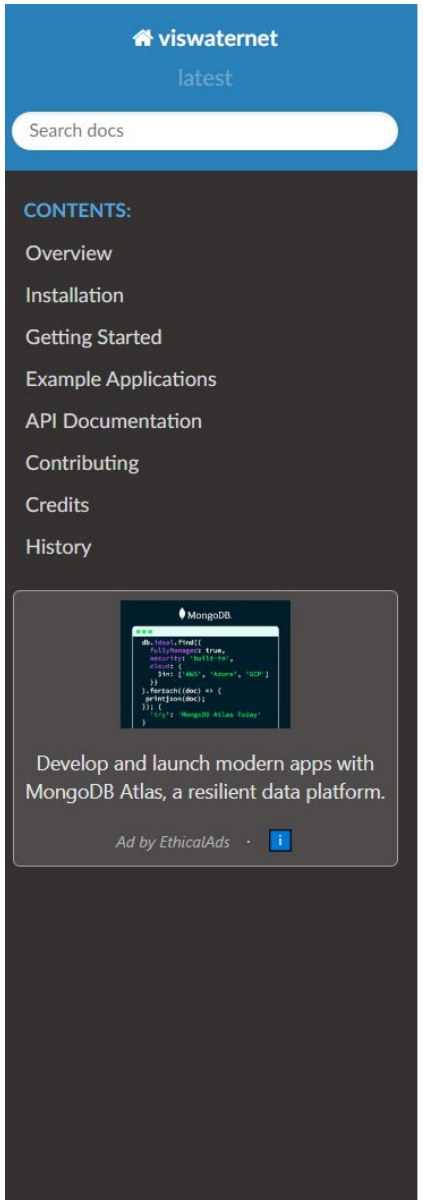
- Use the `animate_plot` function to
  - Show **time-varying data** (**trace** from the lower reservoir)

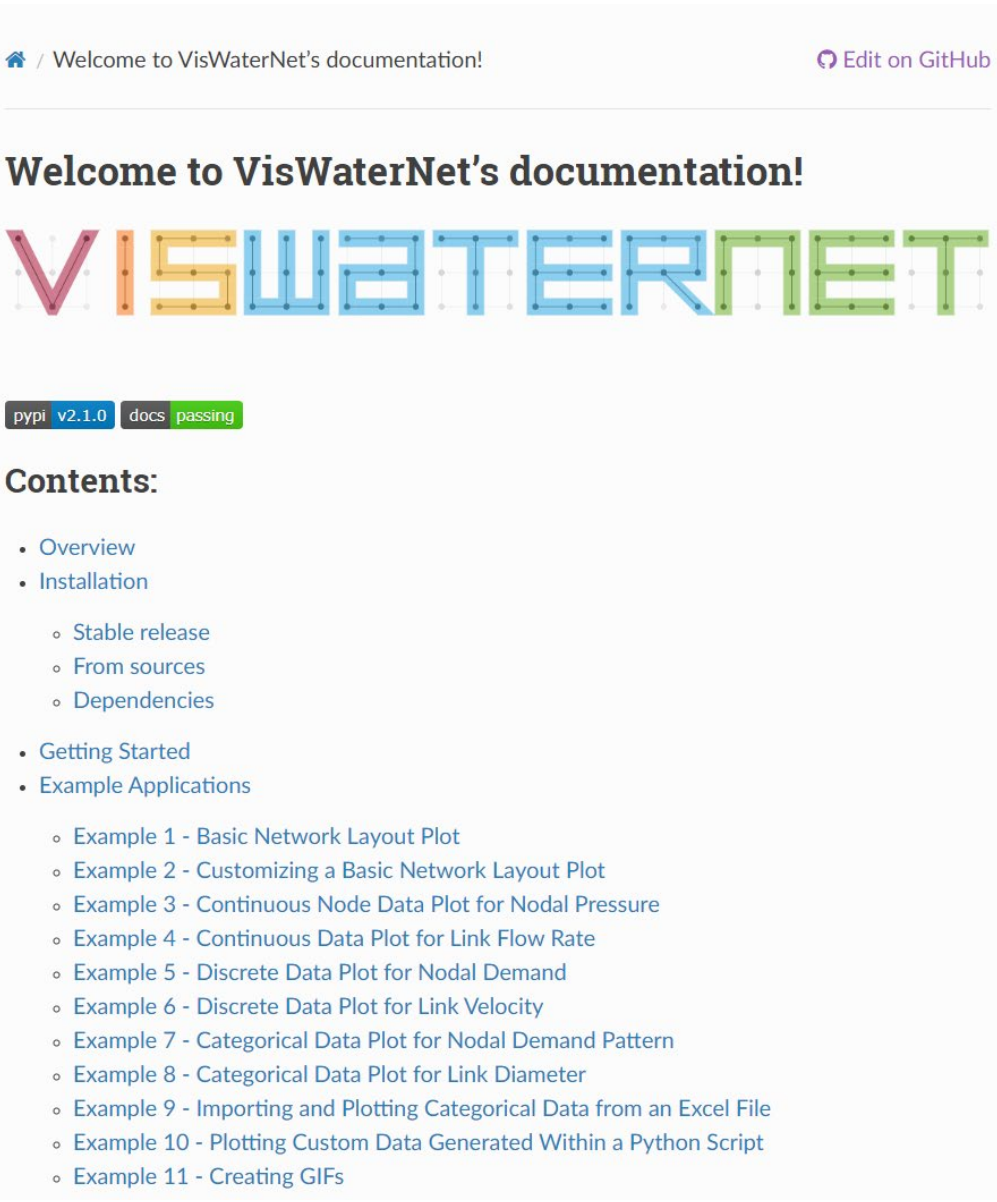
```
style6 = vis.NetworkStyle(cmap = 'Blues', valve_color = 'yellow',
                    node_size = (0,200),
                    node_border_width = 0.5, node_border_color = 'lightgray', dpi = 400)

fig, ax = plt.subplots(figsize=(8,8));
ax.set_frame_on(False);
model.animate_plot(function = model.plot_continuous_nodes,
                    ax=ax, parameter = 'quality',
                    style = style6,
                    time_unit = 'hr', fps = 1,
                    color_bar_title = 'Trace from lower reservoir [%]',
                    first_timestep=0, last_timestep=None,
                    save_name = 'figures/traceplot', save_format='gif')
```



Documentation

- Check out our docs for more examples: viswaternet.readthedocs.io





Ongoing work

- VisWaterNet is constantly under development
- Upcoming updates include:
  - Compatibility with Python 3.10, 3.11, and 3.12
  - Ability to specify alternate colorbar locations
  - Ability to define custom-generated colorbars

- **Feedback, suggestions, and contributions are welcome on our Github repository**

tylertrimble/viswaternet

Try it out:

- Run the codes
- Make changes:
  - ✓ Marker colors, sizes, and shapes
  - ✓ Link colors, widths, and line styles
  - ✓ Legend intervals
  - ✓ …

MAGNets and VisWaterNet Tutorial

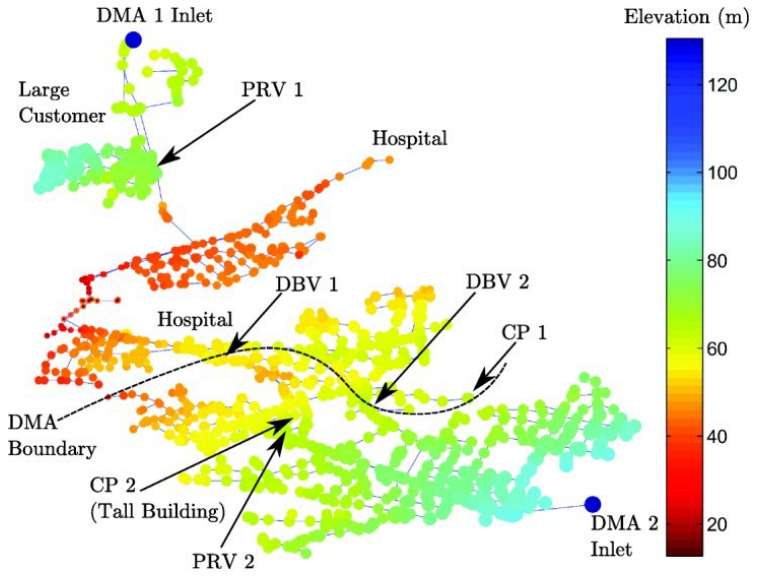
Introducing the case study

Network visualizations with **VisWaterNet**

Network skeletonization with **MAGNets**

Why Do We Need Reduced Models?

- Running simulations for **large** water distribution networks can be computationally demanding
- **Reduced models** can
  - Speed up running times
  - Allow real-time decision making
- **Applications of reduced models :**
  - Optimal real-time operation (Shamir and Salomons, 2008)
  - State estimation (Preis et al., 2011)
  - Leak detection (Moser et al., 2015)



Shamir, U., & Salomons, E. (2008). Optimal real-time operation of urban water distribution systems using reduced models. *Journal of Water Resources Planning and Management*, *134*(2), 181-185.

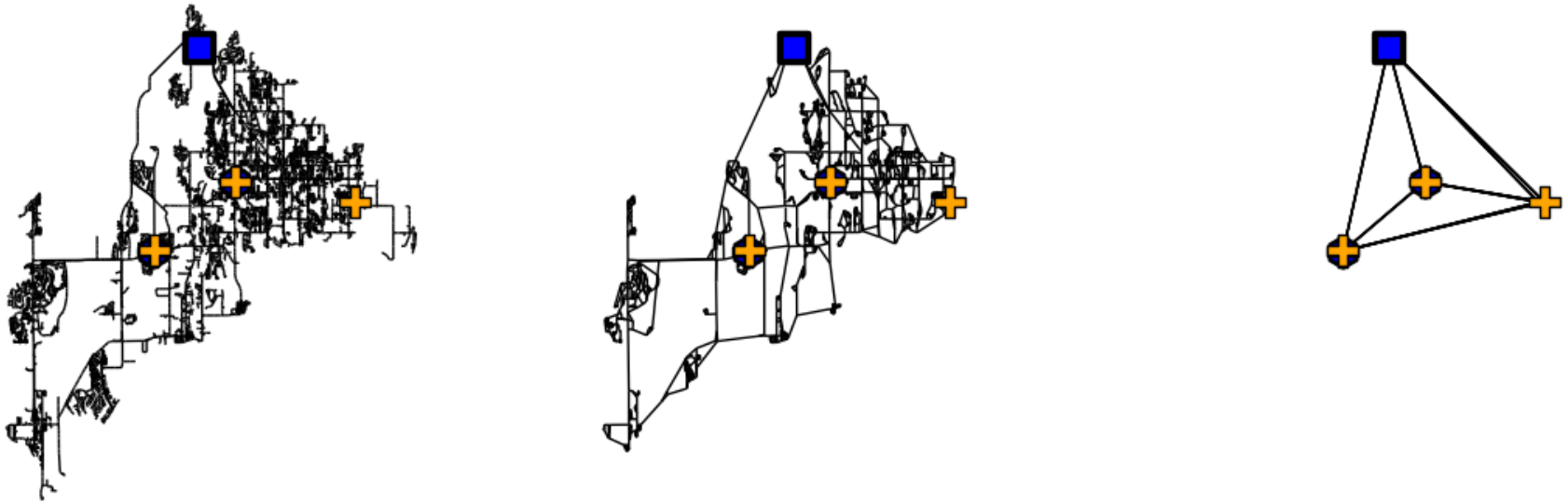
Preis, A., Whittle, A. J., Ostfeld, A., & Perelman, L. (2011). Efficient hydraulic state estimation technique using reduced models of urban water networks. *Journal of Water Resources Planning and Management*, *137*(4), 343-351.

Moser, G., Paal, S. G., & Smith, I. F. (2015). Performance comparison of reduced models for leak detection in water distribution networks. *Advanced Engineering Informatics*, *29*(3), 714-726.

Model Reduction with MAGNets

A Python package for **M**odel reduction and **Ag**gregation of Water distribution **Net**work**s**

meghnathomas/MAGNets



Ulanicki, B., A. Zehnpfund, and F. Martinez. (1996). Simplification of water distribution network models. In Proc., 2nd Int. Conf. on Hydroinformatics, Zurich, Switzerland, 493–500.

Thomas, M., & Sela, L. (2023). Magnets: Model reduction and aggregation of water networks. Journal of Water Resources Planning and Management, 149(2), 06022006. DOI: 10.1061/JWRMD5.WRENG-5486.

1. Model Reduction with MAGNets

- The presentation mirrors the scripts in the Binder files **4. MAGNets Tutorial**
- We will use the `reduce_model` function to **generate reduced models**

Step 0: Use pip to install MAGNets

```
pip install magnets
```

Step 1: Import MAGNets into script

```
import magnets as mg
```

1. Getting Started

- **1.2 Define** the .INP file and initialize WNTR water network model object

```
# import bwfl
inp = 'networks/BWFLnet_modified.inp'

wn = wntr.network.WaterNetworkModel(inp)
```

- **1.3 Run a hydraulic simulation** with WNTR (EPANET 2.2 engine), store run time, and number network components

```
#run wntr simulation
sim = wntr.sim.EpanetSimulator(wn)

t1 = time.time()
results = sim.run_sim()
t2 = time.time()

simtime = t2-t1
print('The simulation run time was ', simtime, 's')
```

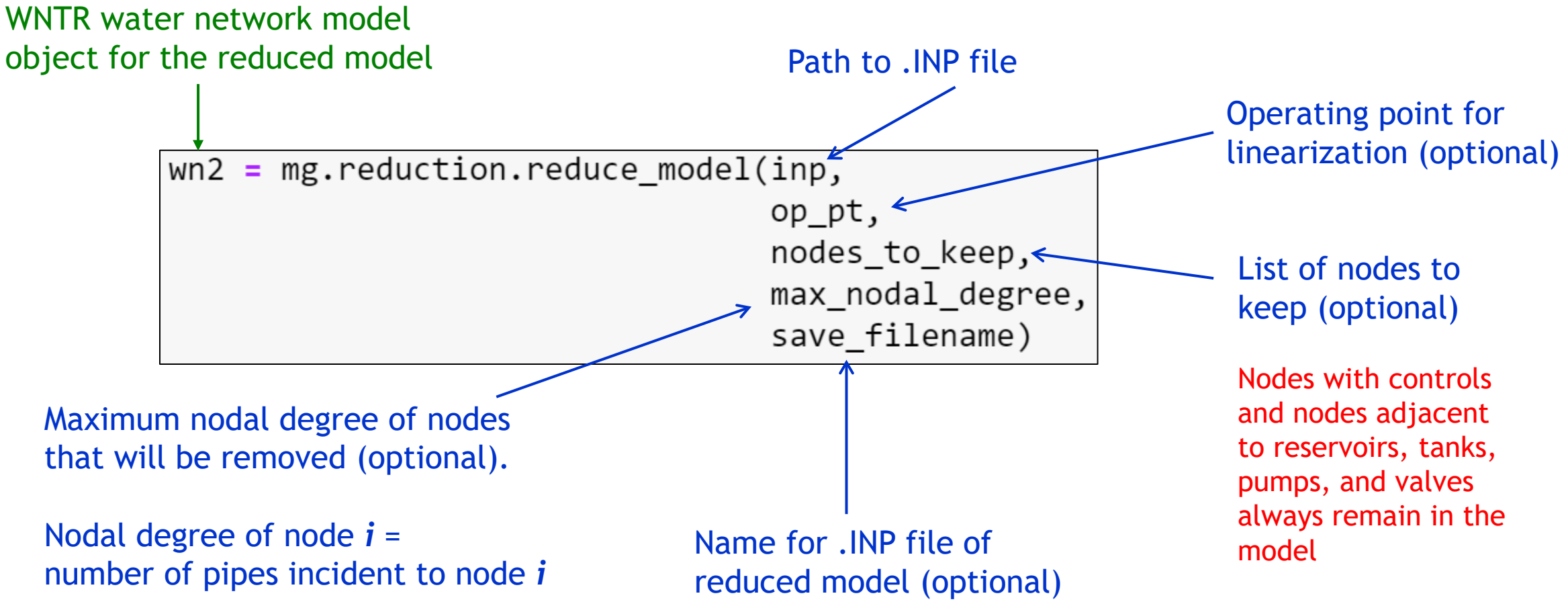
```
The simulation run time was  0.4101834297180176 s
```

```
#describe the number of components in the system
wn.describe()
```

```
{'Nodes': 2548,
 'Links': 2606,
 'Patterns': 9,
 'Curves': 0,
 'Sources': 0,
 'Controls': 0}
```

2.1. Model Reduction with MAGNets

Step 2: Use the `reduce_model` function to **generate reduced models**



2.2. Branch Trimming

- Use the `reduce_model` function to
  - **Trim branches** (remove nodes up to nodal degree 1)
  - Leave in **critical nodes**

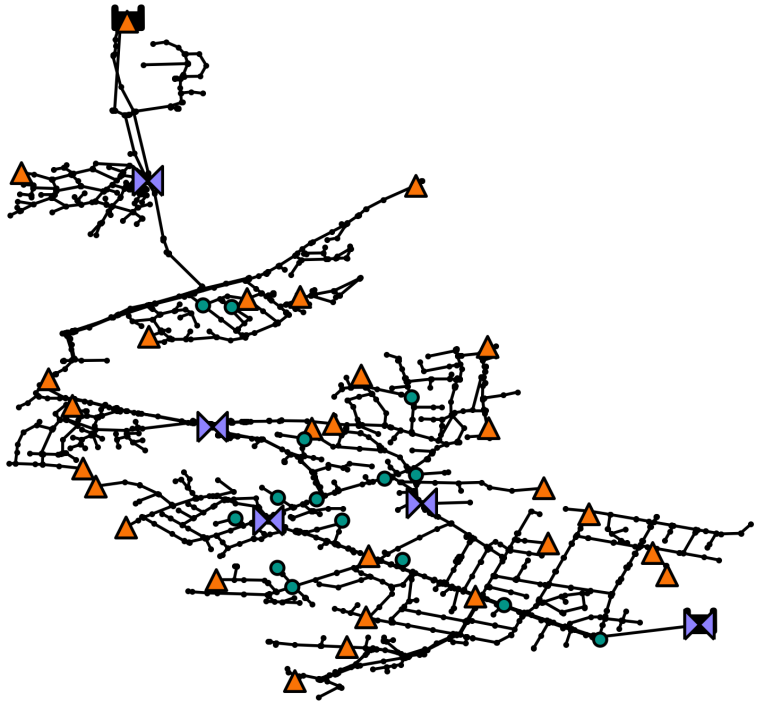
```
#define inp_file
inp = 'Networks/BWFLnet_modified.inp'

#specify operating point, nodes to keep
op_pt = 20

#max nodal degree = 1
max_nodal_degree1 = 1

#call model reduction
mg1 = mg.reduction.reduce_model(inp,
                                op_pt = op_pt,
                                nodes_to_keep = critical_nodes,
                                max_nodal_degree = max_nodal_degree1,
                                save_filename = 'Networks/BWFLnet_modified_level1')
```

Note: you will likely see this warning, which informs us that some of the nodes in critical nodes either do not exist in the model or were going to remain in the reduced model anyway (because they are assigned control rules or are connected to tanks, reservoirs, pumps, or valves).

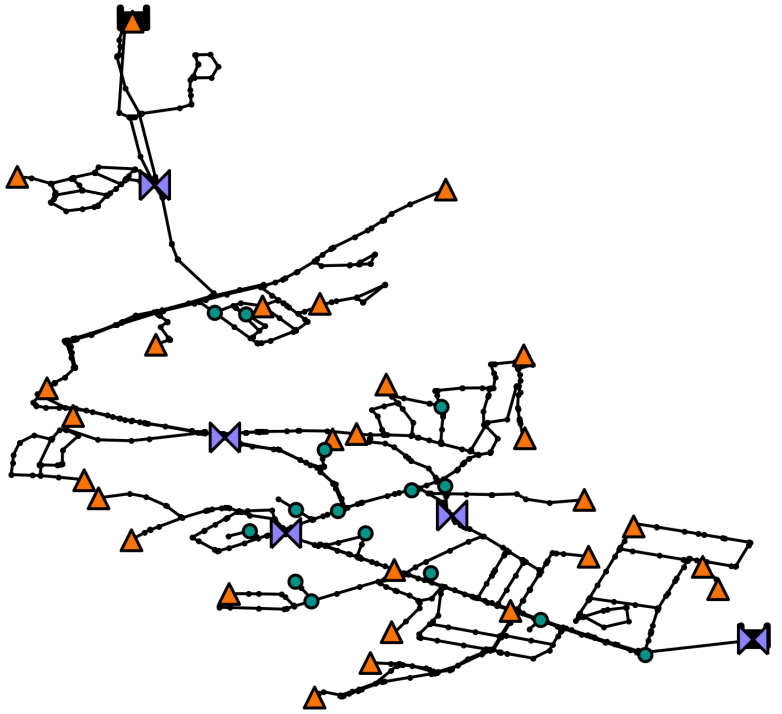
```
C:\Software\Anaconda\lib\site-packages\magnets\utils\characteristics.py:161: UserWarning: Some values in list of nodes to keep
provided by user do not exist in the model or have already been classified as special nodes. These values have been ignored.
  warnings.warn('Some values in list of nodes to keep provided by user do not exist in the model or have already been classifie
d as special nodes. These values have been ignored.')
```

2.2. Branch Trimming

- >1000 nodes removed
- All critical nodes remain



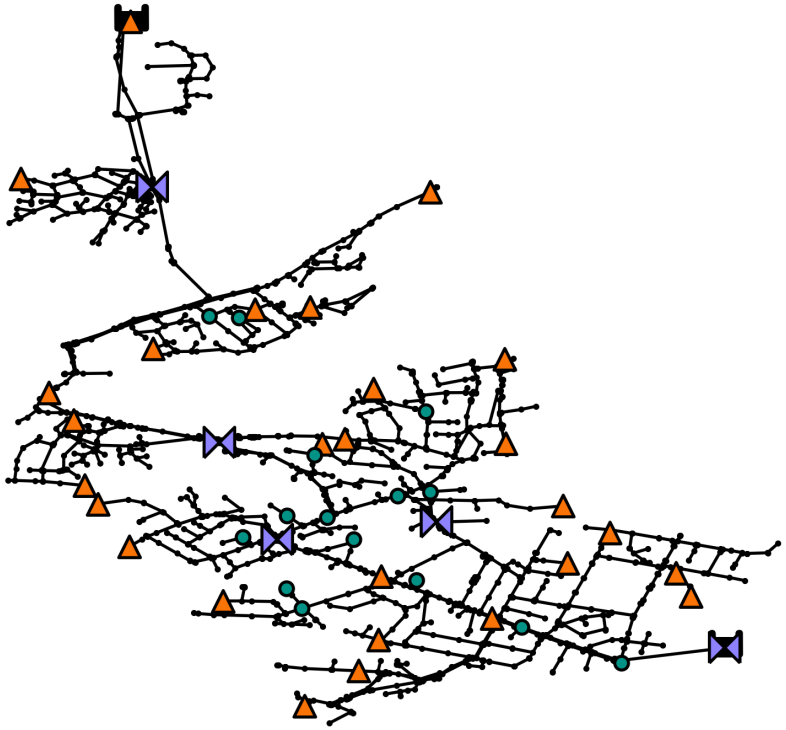
```
{'Nodes': 2548,
 'Links': 2606,
 'Patterns': 9,
 'Curves': 0,
 'Sources': 0,
 'Controls': 0}
```



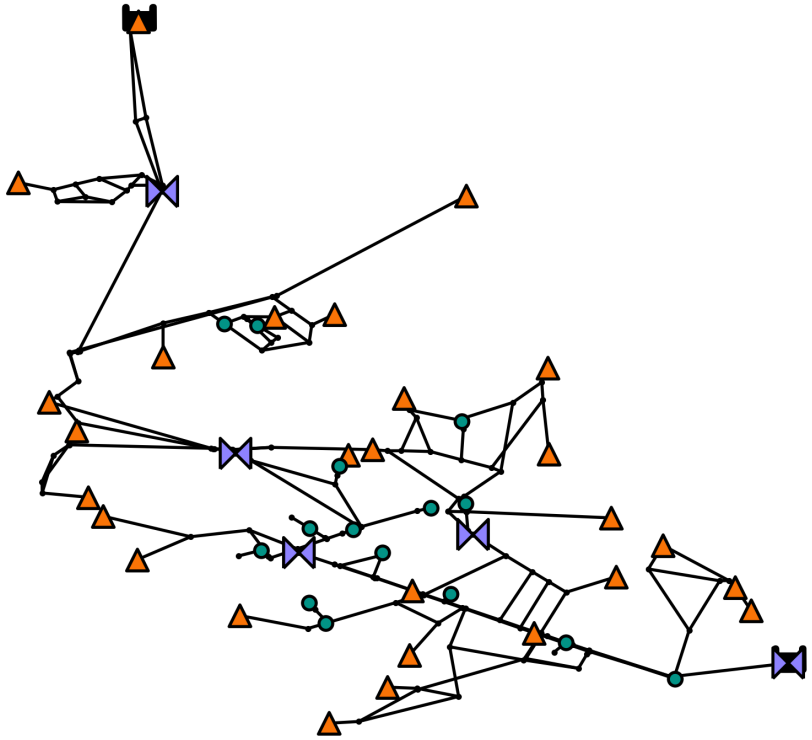
```
{'Nodes': 1427,
 'Links': 1485,
 'Patterns': 9,
 'Curves': 0,
 'Sources': 0,
 'Controls': 0}
```

2.3. Remove Branches and Pipes in Series

- Use the `reduce_model` function with `max_nodal_degree` = 2:



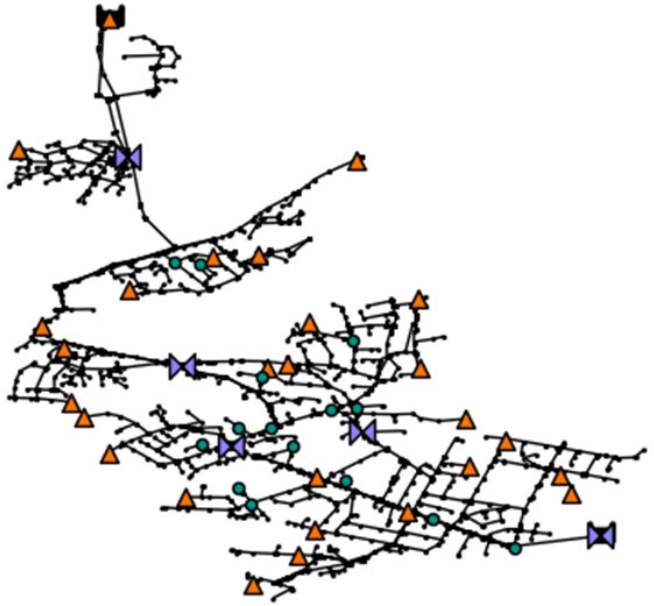
```
{'Nodes': 2548,
 'Links': 2606,
 'Patterns': 9,
 'Curves': 0,
 'Sources': 0,
 'Controls': 0}
```



```
{'Nodes': 211,
 'Links': 262,
 'Patterns': 9,
 'Curves': 0,
 'Sources': 0,
 'Controls': 0}
```

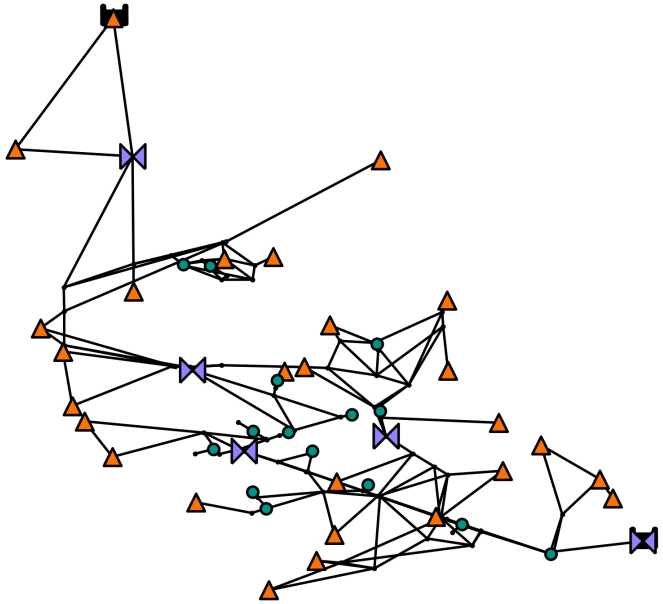
2.4., 2.5. Higher *Max Nodal Degree* Inputs

Original model



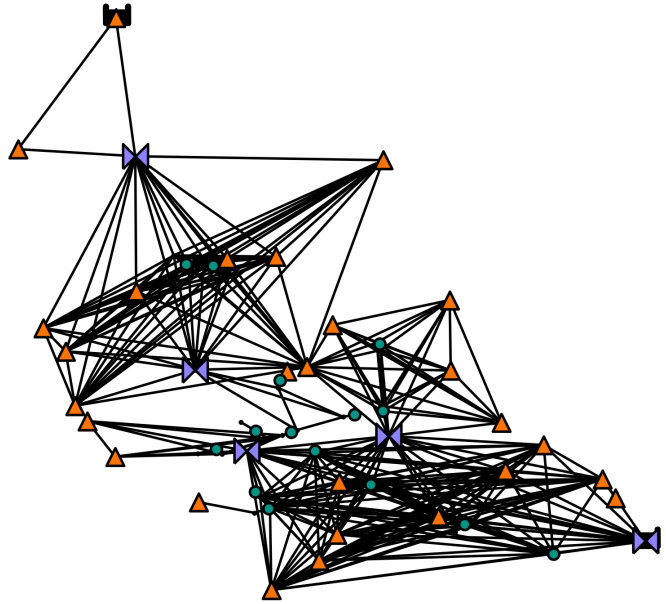
```
{'Nodes': 2548,
 'Links': 2606,
 'Patterns': 9,
 'Curves': 0,
 'Sources': 0,
 'Controls': 0}
```

max_nodal_degree = 3



```
{'Nodes': 140,
 'Links': 215,
 'Patterns': 9,
 'Curves': 0,
 'Sources': 0,
 'Controls': 0}
```

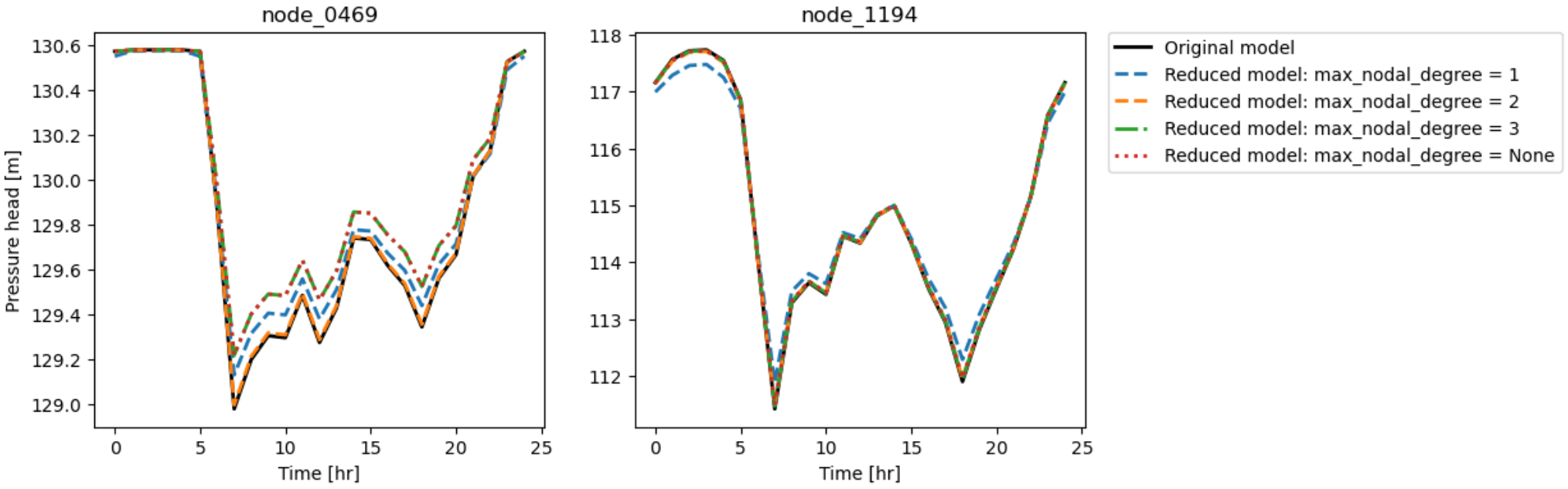
max_nodal_degree = None



```
{'Nodes': 86,
 'Links': 357,
 'Patterns': 9,
 'Curves': 0,
 'Sources': 0,
 'Controls': 0}
```

2.6. Time Series Comparison

- Plot **pressure head time series** for reduced and original models
- Pressures closely match for varying levels of reduction



2.7. Running Time Comparison

	Original model	Reduced model: degree 1	Reduced model: degree 2	Reduced model:degree 3	Reduced model: degree None
Running time [s]	0.51	0.29	0.07	0.06	0.08
Speedup	1.00	1.74	6.78	8.28	6.74

- Reduced models run **1.7-8.3** times faster than original models
- Reduced models can be used to speed up simulation running times while preserving pressures at the remaining nodes
- **Things to note:**
- Users should **carefully choose**:
  - Nodes to keep in model
  - Degree of reduction
  - Operating point

Comparison with WNTR Skeletonization

- WNTR offers a **skeletonization** function that performs
  - Branch trimming
  - Series pipe merge
  - Parallel pipe merge

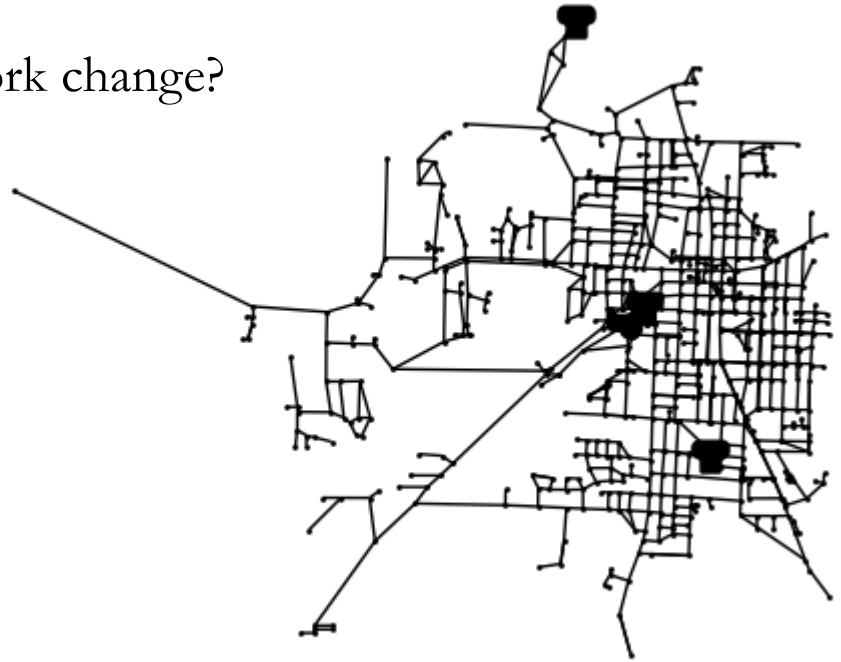
```
#skeletonize model using WNTR
wn2 = wntr.morph.skeletonize(wn, pipe_diameter_threshold = 0.254*3)
```

- WNTR skeletonization is performed using **equivalent pipe equations**
- MAGNets skeletonization uses **Gaussian elimination** (Ulanicki et al., 1996) and requires linearization around an **operating point**

Ulanicki, B., A. Zehnpfund, and F. Martinez. 1996. "Simplification of water distribution network models." In Proc., 2nd Int. Conf. on Hydroinformatics, Zurich, Switzerland, 493–500.

3. Exercise

- Use the `reduce_model` function to reduce network **ky2**
  - Change the **operating point**: how do the head results change?
  - Change the **max nodal degree**: how does the size of the network change?



- For more examples, refer to our Github repository and JWRPM paper

meghnathomas/MAGNets

Thomas, M., & Sela, L. (2023). Magnets: Model reduction and aggregation of water networks. Journal of Water Resources Planning and Management, 149(2), 06022006. DOI: 10.1061/JWRMD5.WRENG-5486.

Agenda

- Introductions: 5 min
- Using the Tutorials: 5 min
- Overview: 15 min
- WNTR Tutorial: 60 min
- Break: 15 min
- MAGNets and VisWaterNet Tutorial: 45 min
- **Break: 15 min**
- Future Capabilities and Discussion: 45 min

Agenda

- Introductions: 5 min
- Using the Tutorials: 5 min
- Overview: 15 min
- WNTR Tutorial: 60 min
- Break: 15 min
- MAGNets and VisWaterNet Tutorial: 45 min
- Break: 15 min
- **Future Capabilities and Discussion: 45 min**

Future Capabilities & Discussion

Current Modeling Capabilities & Future Directions

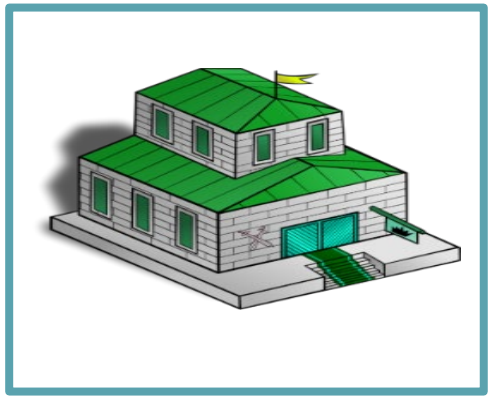
Brief overview of current EPA water infrastructure model development efforts

- Drinking water models
- Stormwater & collection system models
- Water reuse models

Current EPA priorities are focused on:

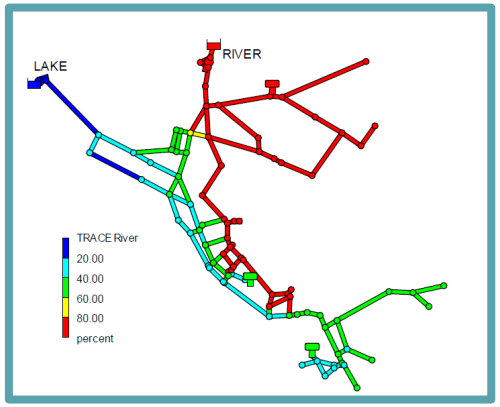
- Removing Pb plumbing materials from water systems
- Removing and destroying PFAS in water
- Assisting environmental justice and disadvantaged communities

EPA Drinking Water Infrastructure Models



Drinking Water Treatment Models & Tools

- Disinfection
- Treatment
- Treatability DB



Drinking Water Distribution System Models & Tools

- EPANET
- EPANET-MSX
- EPANET-RTX (Real-time)
- WNTR (Resilience)

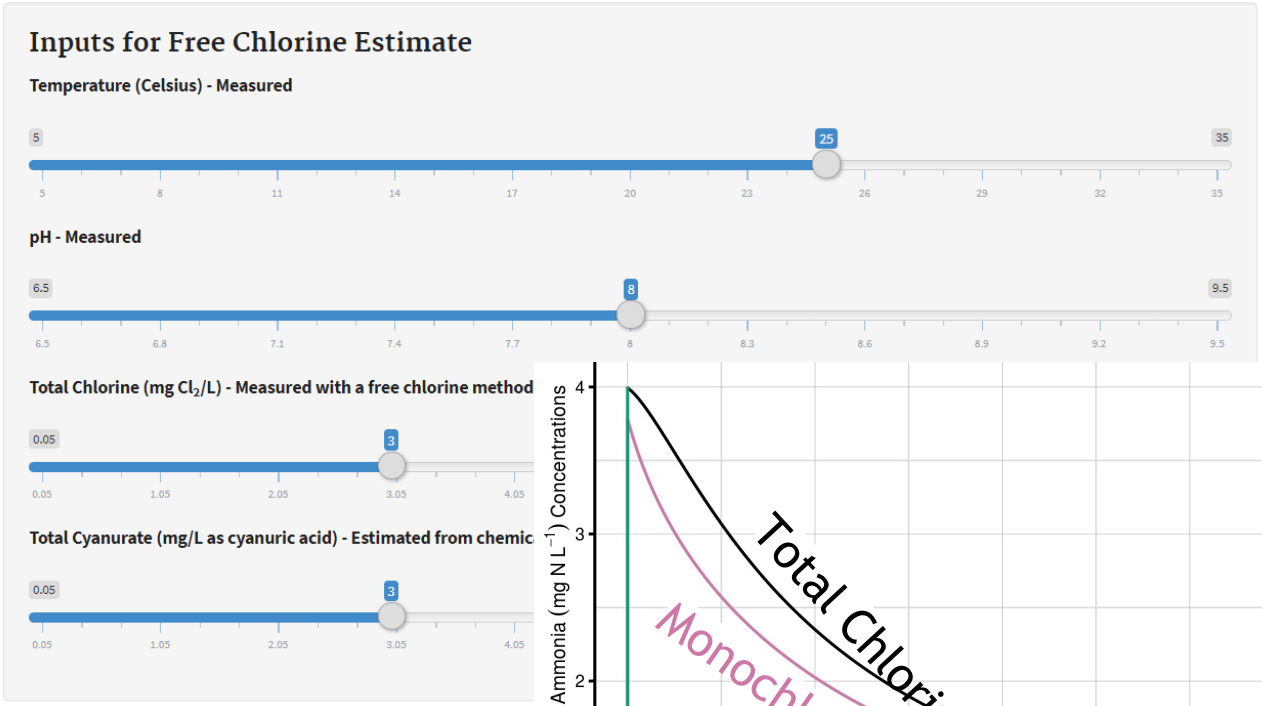
Drinking Water Premise Plumbing Models & Tools

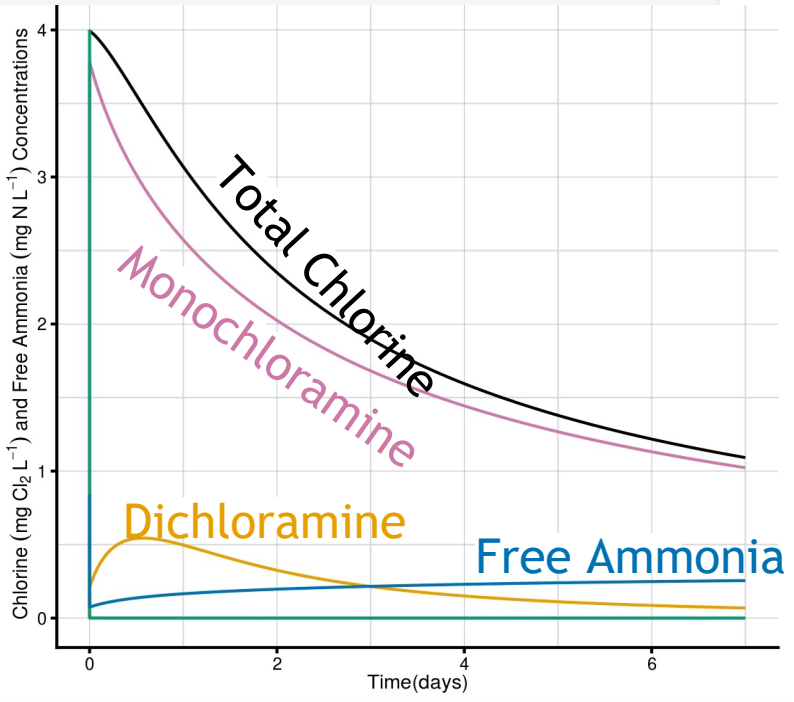
- PPM Tools

EPA Drinking Water Disinfection Models

Web-based calculators

- Chloramine formation and decay
  https://shiny.epa.gov/cfd/
- Chlorine breakpoint curve
  https://shiny.epa.gov/cbcs/
- Chlorine and cyanuric acid chemistry
  https://shiny.epa.gov/fccas/
- Free chlorine estimator for dichlor/trichlor disinfection
  https://shiny.epa.gov/fcedts





EPA Drinking Water Treatment Models

Environmental Technologies Design Option Tool (ETDOT)

- Suite of software for modeling a variety of treatment technologies by Michigan Technological University (MTU)
- Includes models for adsorption, advanced oxidation, aeration, biofilters, PAC adsorption, ion exchange

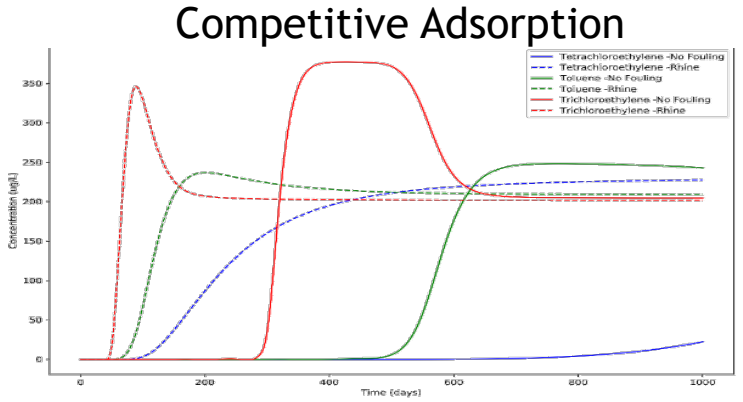
http://www.epa.gov/water-research/environmental-technologies-design-option-tool-etdot

Granular Activated Carbon Model

- Pore and Surface Diffusion Model (PSDM)
- Includes parameter estimation

Ion Exchange Media

- Supports gel-type, macroporous ion exchange resins & competition from divalent ions, such as sulfate



https://github.com/USEPA/Water_Treatment_Models

EPA Treatment Tools: Treatability Database

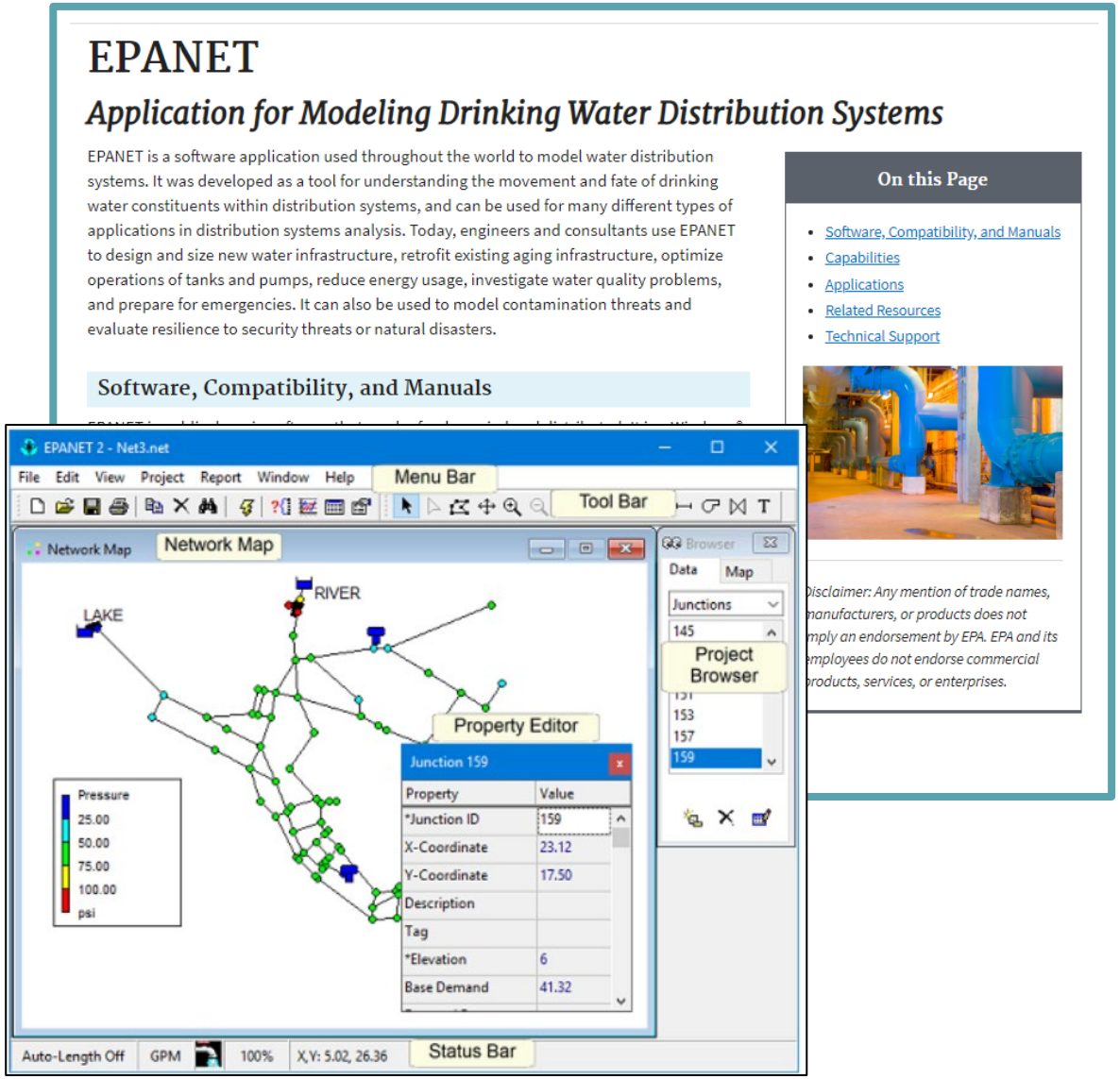
- Interactive searchable database
- 142 regulated and unregulated contaminants, including 37 PFAS
- 35 treatment processes commonly employed or known to be effective
- Referenced information gathered from thousands of literature sources

epa.gov/water-research/drinking-water-treatability-database-tdb

EPANET Water Distribution Model

- Simulates hydraulics within water distribution network
- Models decay/growth of single substance
- More than 50,000 downloads per year
- Components utilized for multiple commercial software packages
- Latest official release: version 2.2 in July 2020
- New engineer and new postdoc joining EPA
- Research directions include enhancing usability for small systems, optimization of operations, improving run time for real-time modeling, and incorporating consumer actions
- Planned stakeholder meeting next year at EWRI to seek input on new feature requests

https://www.epa.gov/water-research/epanet



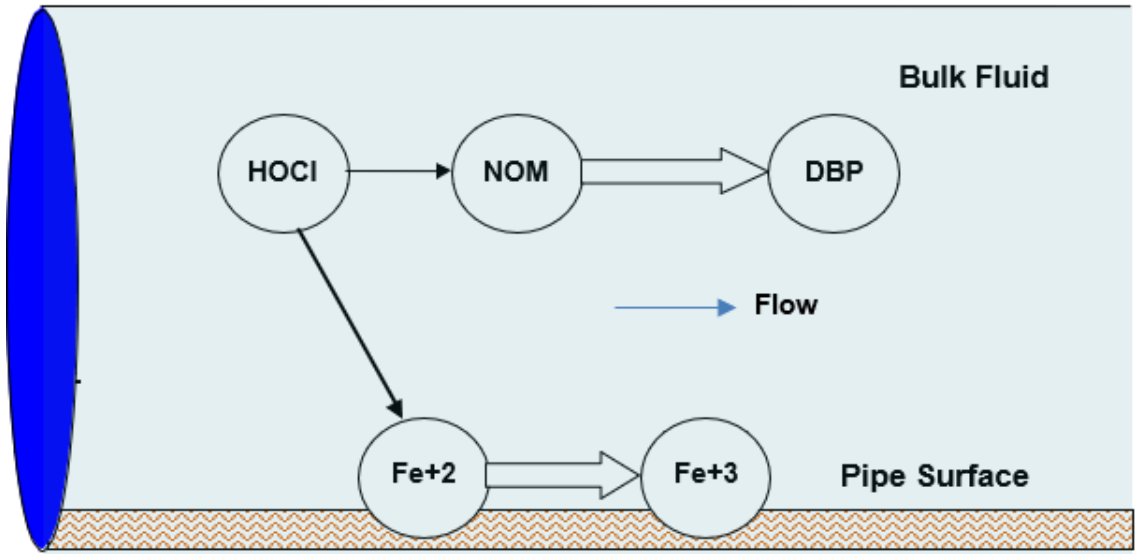
EPANET Extensions: EPANET-MSX

Used to model

- Adsorption/desorption on pipe walls
- Attachment to biofilms
- Chemical reactions
- Biological growth and decay

EPANET-MSX 2.0 updates

- Compatible with EPANET 2.2
- Added dispersion
- Added parallelization
- Added mass balance reporting
- Added sub-second timesteps



https://github.com/USEPA/EPANETMSX

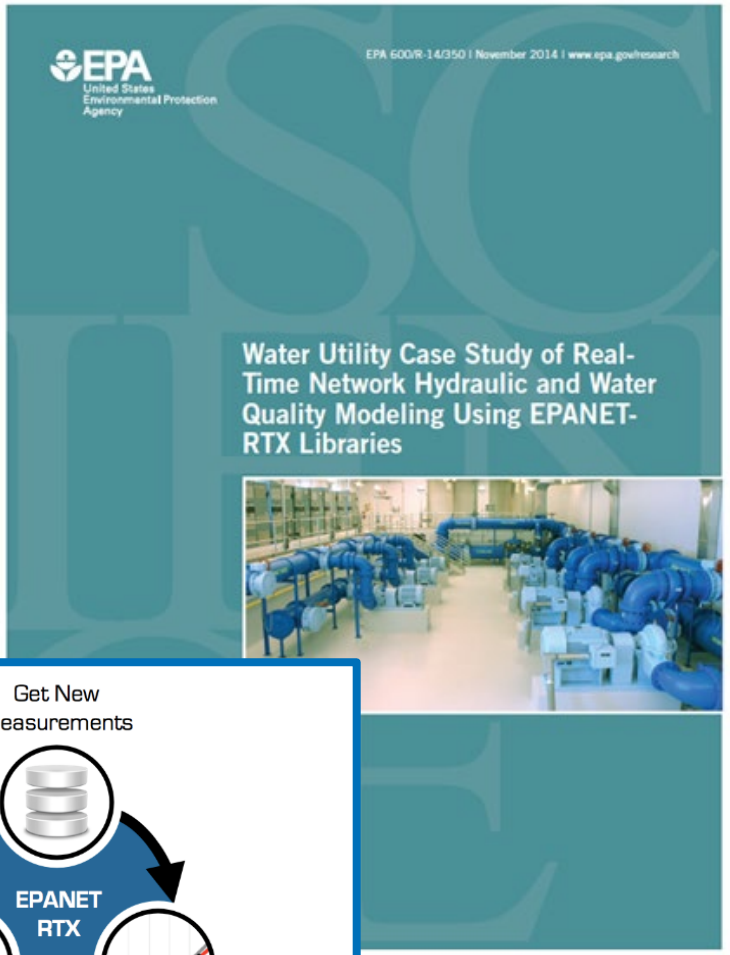
EPANET Extensions: Real-time Tools

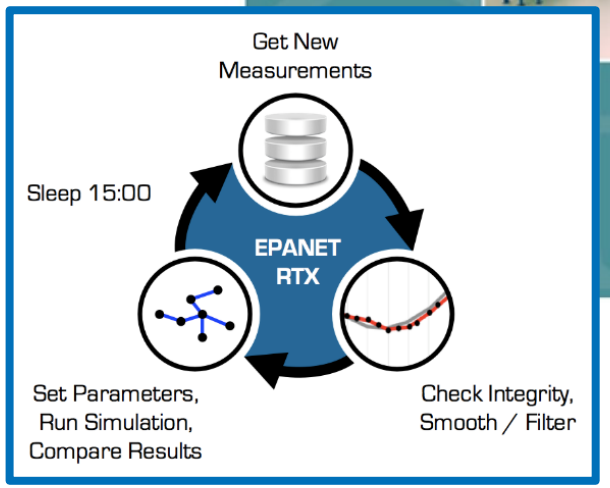
EPANET-RTX is suite of software libraries to integrate EPANET with SCADA operational data

RTX enables real-time analytics for automated:

- Forward and hind casting
- Model calibration
- Simulation & comparison of operational decisions
- Continuous comparison between model and sensor/data outputs, and analysis, allows for more accurate predictions

https://github.com/OpenWaterAnalytics/epanet-rtx





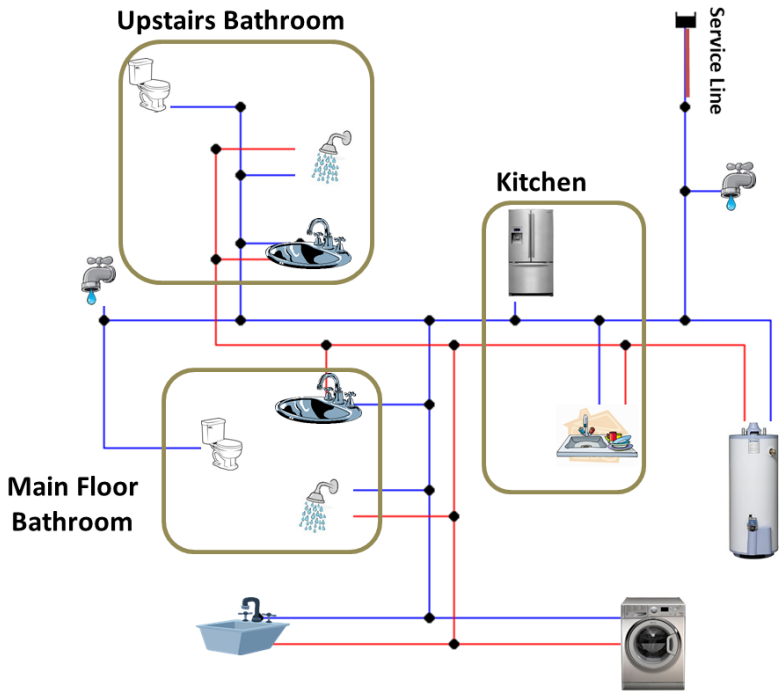
EPA Premise Plumbing Models

Premise Plumbing Modeling Tools (PPMtools)

- Leverages EPANET and WNTR
- Models real-world fixtures & usage patterns
- Generates and runs many scenarios
- Predicts water quality information over time
- Simulates flushing to remove contaminants
- Estimates exposure to contaminants
- Being validated through experiments

https://github.com/USEPA/PPMtools

EWRI Premise Plumbing Modeling Workgroup Webinar Series



Stormwater and Water Reuse Models and Tools

Stormwater Models & Tools

- SWMM
- Stormwater Calculator
- GI Toolkit

Water Reuse Models & Tools

- NEWR
- Microbial Risk Assessment

EPA Stormwater Management Model (SWMM)

SWMM simulates runoff quantity and quality and routes them through collection system infrastructure

SWMM has played pivotal role in flow and pollution control in collection systems since its inception

SWMM supports studies driven by regulatory imperatives including:

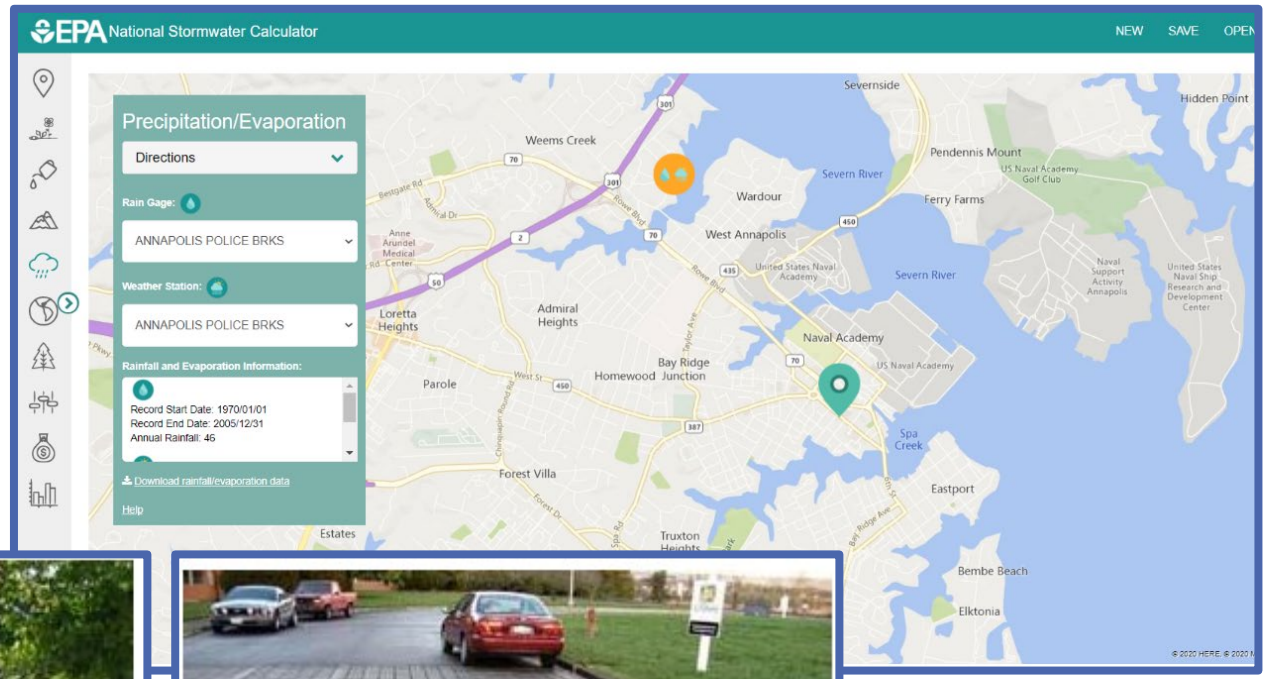
- Long Term Control Plan development (LTCP)
- Total Maximum Daily Allocation (TMDL)
- Municipal Separate Stormwater Systems (MS4)
- National Pollutant Discharge Elimination System (NPDES)

Future directions include:

1) Improving performance, runtime, real-time data integration to support digital twin applications,
2) Enhancing geospatial capabilities to support applications,
3) Advancing water quality modeling component of SWMM (similar to MSX)

EPA National Stormwater Calculator

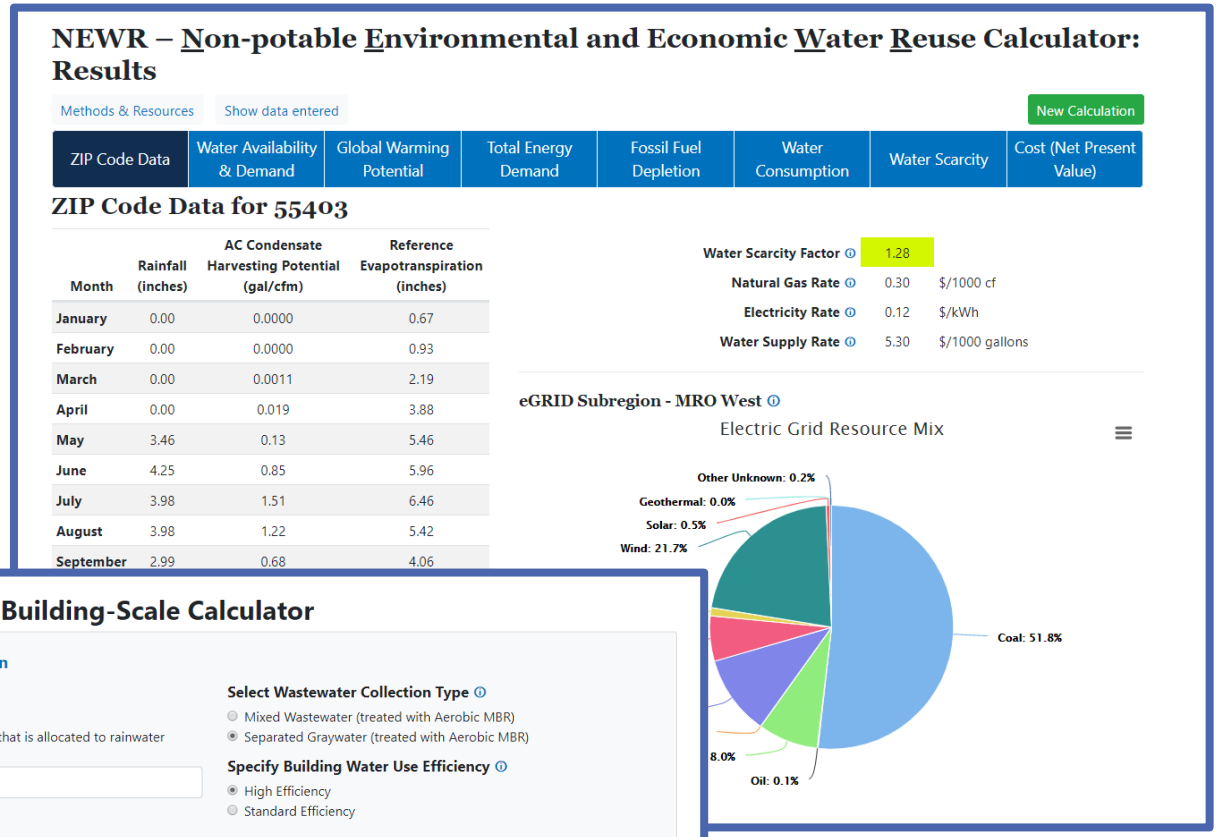
- Distributed, estimates annual rainwater and frequency of runoff
- Site specific
- Low impact development controls

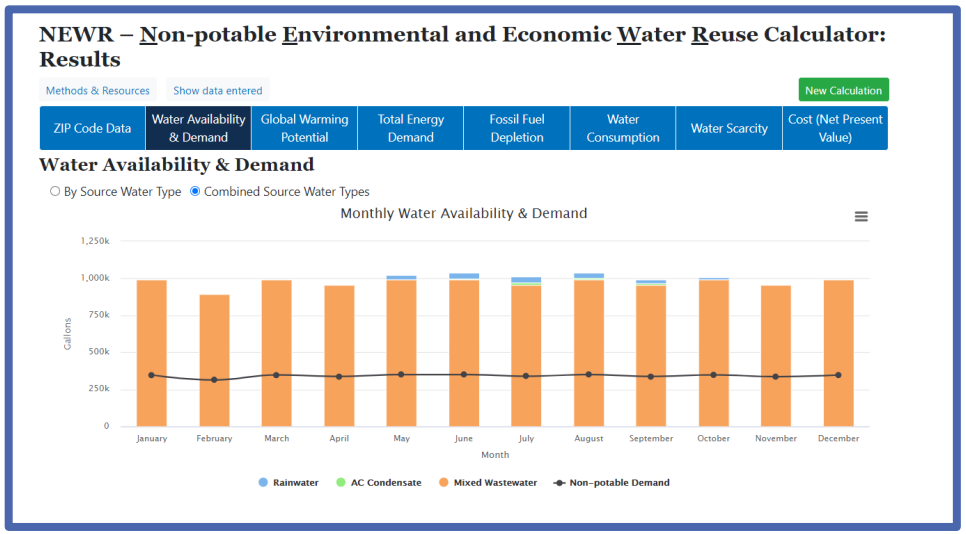


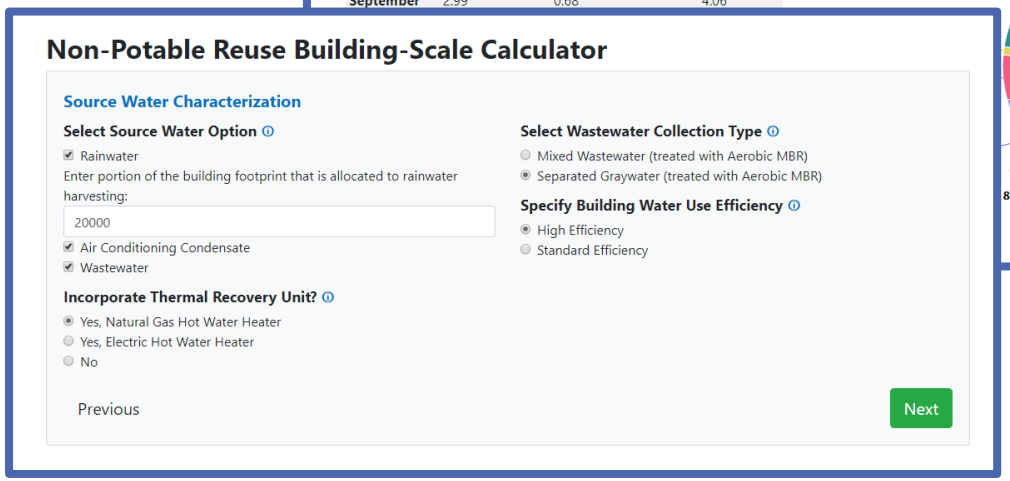
https://www.epa.gov/water-research/national-stormwater-calculator

EPA Building Water Reuse Calculator

- NEWR compares water reuse options within a single building
- Based on geography, end use, building specifications, and source water type
- Calculates costs as well as environmental benefits







New and Upcoming WNTR Capabilities

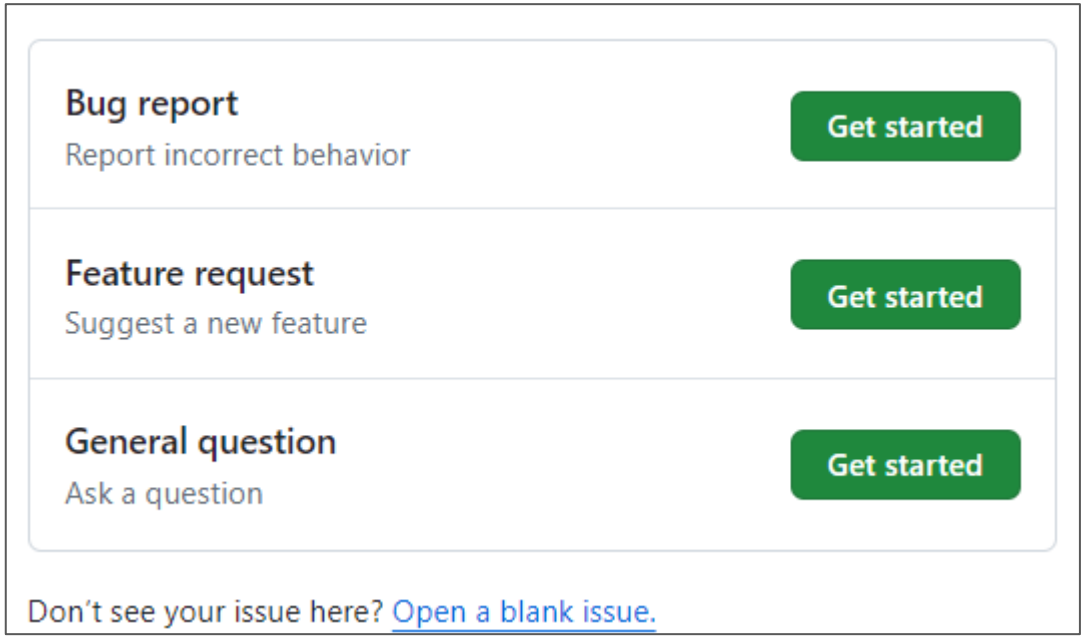
Capability	Module	Status
Geospatial analysis: Continue to improve geospatial analysis options, using GIS data for model development and resilience analysis	wntr.gis	First released Nov 2022
Stormwater and wastewater analysis: Couple WNTR with SWMM (S-WNTR)	wntr.stormwater	Coming soon
Multi-species water quality simulations: Couple WNTR with EPANET-MSX, load and modify MSX models	wntr.msx	Coming soon
Demand pattern library: Load and modify existing pattern timeseries to add to water network models	wntr.library	Coming soon
Response optimization: Couple WNTR with optimization solvers to determine response and mitigation actions	wntr.optimize	Prototype

Contributing to WNTR

WNTR welcomes contributions from the user community. There are numerous ways to contribute:

- Add new features
- Improve or extend existing features
- Post issues and feedback
- Add or update documentation
- Add or improve tests
- Contact the developers to discuss ideas
- Add journal articles that use WNTR to the User Community website

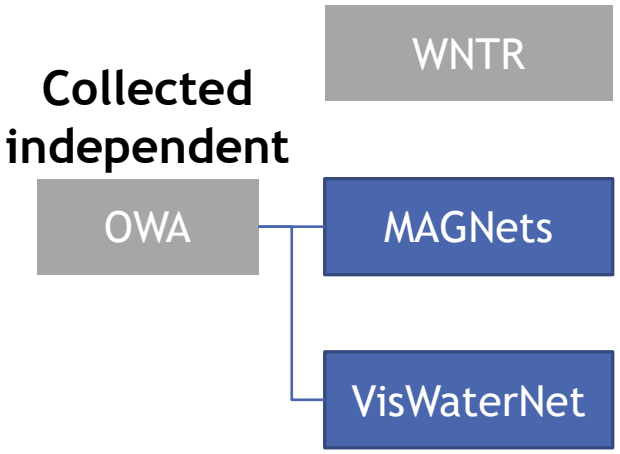
https://github.com/USEPA/WNTR/issues



Potential Co-development Frameworks with WNTR

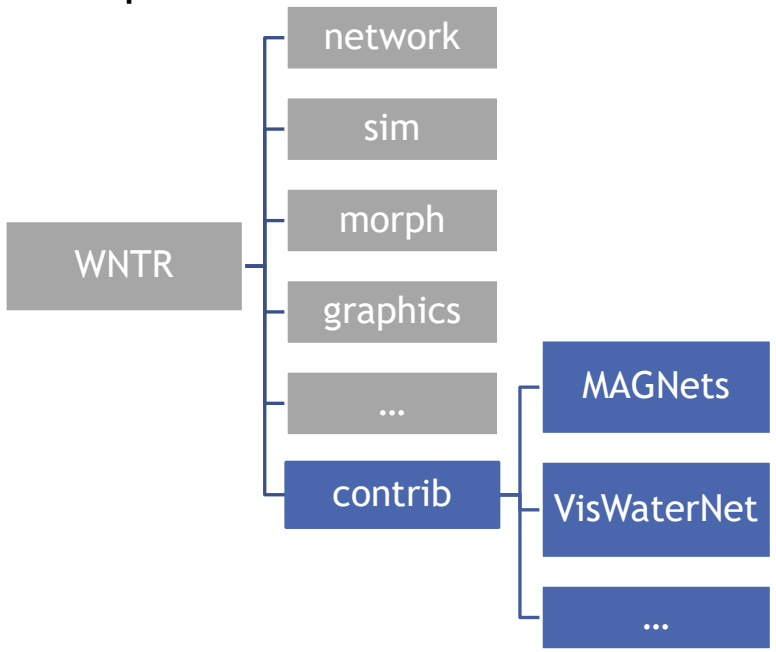
Independent

Third party software imports and uses WNTR. No WNTR requirements for documentation, or testing.



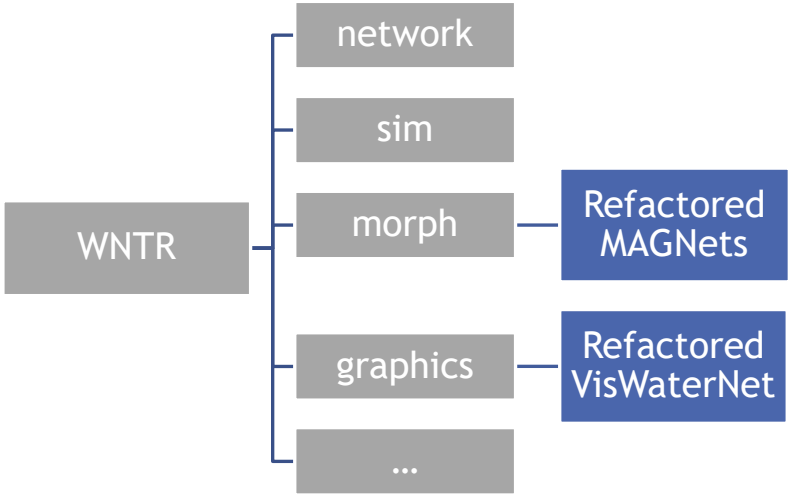
Contrib module

Third party software, documentation, and testing pulled into WNTR, but remains separate from the main code.



Fully integrated

Third party software, documentation, and testing are added directly to WNTR.



New capabilities and student research

Intended for community use and refinement

Widely used, fully tested and documented

Open Source Software and Datasets

Many GitHub Organizations and University websites support software and datasets for Water Distribution Systems Analysis

- USEPA: EPANET2.2, SWMM, WNTR, Water Security Toolkit, Canary, ...
- OpenWaterAnalytics: EPANET, epanet-dev, EPyT, ...
- Sandia National Laboratories: Chama, Pecos, ...
- Pyswmm: pyswmm, swmmio, ...
- KWR-Water: pysimdeum, ...
- Vitens: epynet, DBM, ...
- KIOS-Research: LeakDB, EPANET-Matlib-Toolbox, BattleDIM, ...
- Critical-Infrastructure-Systems-Lab: DHALSIM, epanetCPA, ...
- Graz University of Technology: oopnet, ...
- University of Innsbruck: DynaVIBe, ORONET, ...
- University of Kentucky: Water Distribution System Research Database, ...
- University student GitHub accounts: MAGNets, VisWaterNet, …
- And many more…

Open-Source Software Development Best Practices

- Host software on a **public repository** (e.g., Github, Gitlab) that allows **community involvement** through
  - Raising issues
  - Providing feedback
  - Accepting contributions
- Provide **installation instructions** and specify language versions and package dependencies
- Provide clear **documentation** of functions and arguments and provide usage **examples**
- Consider using **automated** tools for **testing** (e.g., Github Pages, Travis-CI) and **package creation and management** (e.g., Cookiecutter)
- **New versions of your dependencies will be released all the time so you will have to maintain and update your code to stay relevant!**

Discussion

- (1) What software packages you and your team are using?
- (2) What accessibility or usability roadblocks you experience? (software language, operating system, software design)
- (3) What new WNTR capabilities are needed to support your research and development?
- (4) How should we coordinate software development efforts? What are the pros and cons of hosting these capabilities in separate software packages or combing packages?
- (5) How can we encourage and preserve contributions from students?

Thank you!

This presentation has been reviewed in accordance with U.S. Environmental Protection Agency policy and approved for publication. The views expressed in this presentation are those of the author(s) and do not necessarily represent the views or policies of the U.S. Environmental Protection Agency.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
SAND2024-07839C