SAND2017-8883R

EPA/600/R-17/264
August 2017

Water Network Tool for Resilience (WNTR) User Manual

by

Katherine A. Klise, David Hart and Dylan Moriarty
Sandia National Laboratories
Geoscience Research and Applications

Michael L. Bynum
Purdue University
Davidson School of Chemical Engineering

Regan Murray, Jonathan Burkhardt, and Terra Haxton
U.S. Environmental Protection Agency
Office of Research and Development

Disclaimer

The United States Environmental Protection Agency through its Office of Research and Development funded and collaborated in the research described here under an Interagency Agreement # DW89924502 with the Department of Energy's Sandia National Laboratories. It has been subjected to the Agency's review and has been approved for publication. Note that approval does not signify that the contents necessarily reflect the views of the Agency. Mention of trade names products, or services does not convey official EPA approval, endorsement, or recommendation. The contractor role did not include establishing Agency policy.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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-NA-0003525.

Contents

List of Tables

List of Figures

Abbreviations

API: Application programming interface

EPA: Environmental Protection Agency

IDE: Integrated development environment

SI: International System of Units

US: United States

WNTR: Water Network Tool for Resilience

Acknowledgements

The U.S. Environmental Protection Agency acknowledges the technical review of the WNTR software and user manual and/or technical editing provided by the following individuals:

- Jonathan Burkhardt, EPA, Office of Research and Development
- Eun Jeong Cha, University of Illinois
- Terra Haxton, EPA, Office of Research and Development
- Sudhir Kshirsagar, Global Quality Corp
- Marti Sinclair, Alion Science and Technology, for Attain

1 Overview

Drinking water systems face multiple challenges, including aging infrastructure, water quality concerns, uncertainty in supply and demand, natural disasters, environmental emergencies, and cyber and terrorist attacks. All of these have the potential to disrupt a large portion of a water system causing damage to infrastructure and outages to customers. Increasing resilience to these types of hazards is essential to improving water security.

As one of the United States (US) sixteen critical infrastructure sectors, drinking water is a national priority. The National Infrastructure Advisory Council defined infrastructure resilience as "the ability to reduce the magnitude and/or duration of disruptive events. The effectiveness of a resilient infrastructure or enterprise depends upon its ability to anticipate, absorb, adapt to, and/or rapidly recover from a potentially disruptive event" [11].

Being able to predict how drinking water systems will perform during disruptive incidents and understanding how to best absorb, recover from, and more successfully adapt to such incidents can help enhance resilience. Simulation and analysis tools can help water utilities to explore the capacity of their systems to handle disruptive incidents and guide the planning necessary to make systems more resilient over time [17].

The Water Network Tool for Resilience (WNTR, pronounced *winter*) is a Python package designed to simulate and analyze resilience of water distribution networks. Here, a network refers to the collection of pipes, pumps, nodes, and valves that make up a water distribution system. WNTR has an application programming interface (API) that is flexible and allows for changes to the network structure and operations, along with simulation of disruptive incidents and recovery actions. WNTR can be installed through the United States Environmental Protection Agency (US EPA) GitHub organization at https://github.com/USEPA/WNTR. An integrated development environment (IDE), like Spyder, is recommended for users and developers. Figure 1 shows the GitHub webpage, Spyder IDE, and sample graphics generated by WNTR.

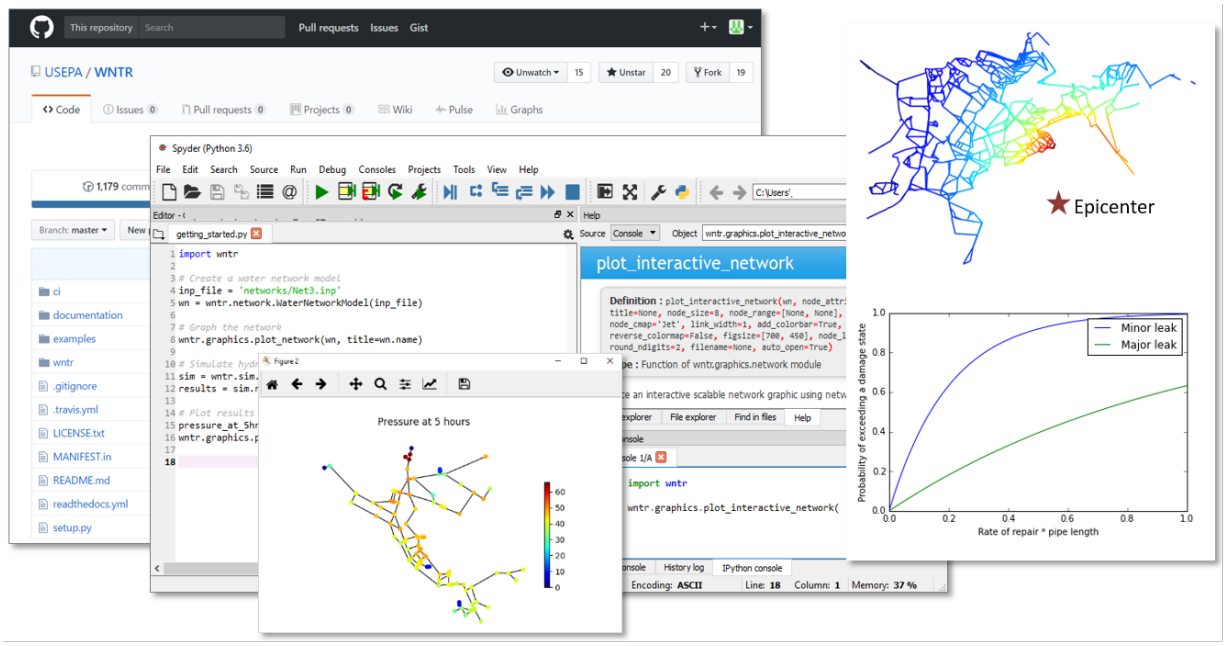

Figure 1: WNTR code repository on GitHub, integrated development environment using Spyder, and sample graphics generated by WNTR.

WNTR includes capabilities to:

- **Generate water network models** from scratch or from existing EPANET-formatted water network model input (EPANET INP) files [13]
- **Modify network structure** by adding/removing components and changing component characteristics
- **Modify network operation** by changing initial conditions, component settings, and time-based and conditional controls
- **Add disruptive incidents** including damage to tanks, valves, and pumps, pipe leaks, power outages, contaminant injection, and changes to supply and demand

- **Add response/repair/mitigation strategies** including leak repair, retrofitted pipes, power restoration, and backup generation
- **Simulate network hydraulics and water quality** using pressure dependent demand or demand-driven hydraulic simulation, and the ability to pause and restart simulations
- **Run probabilistic simulations** using fragility curves for component failure
- **Compute resilience** using topographic, hydraulic, water quality/security, and economic metrics
- **Analyze results and generate graphics** including state transition plots, network graphics, and network animation

These capabilities can be linked together in many different ways. Figure 2 illustrates four example use cases, from simple to complex.

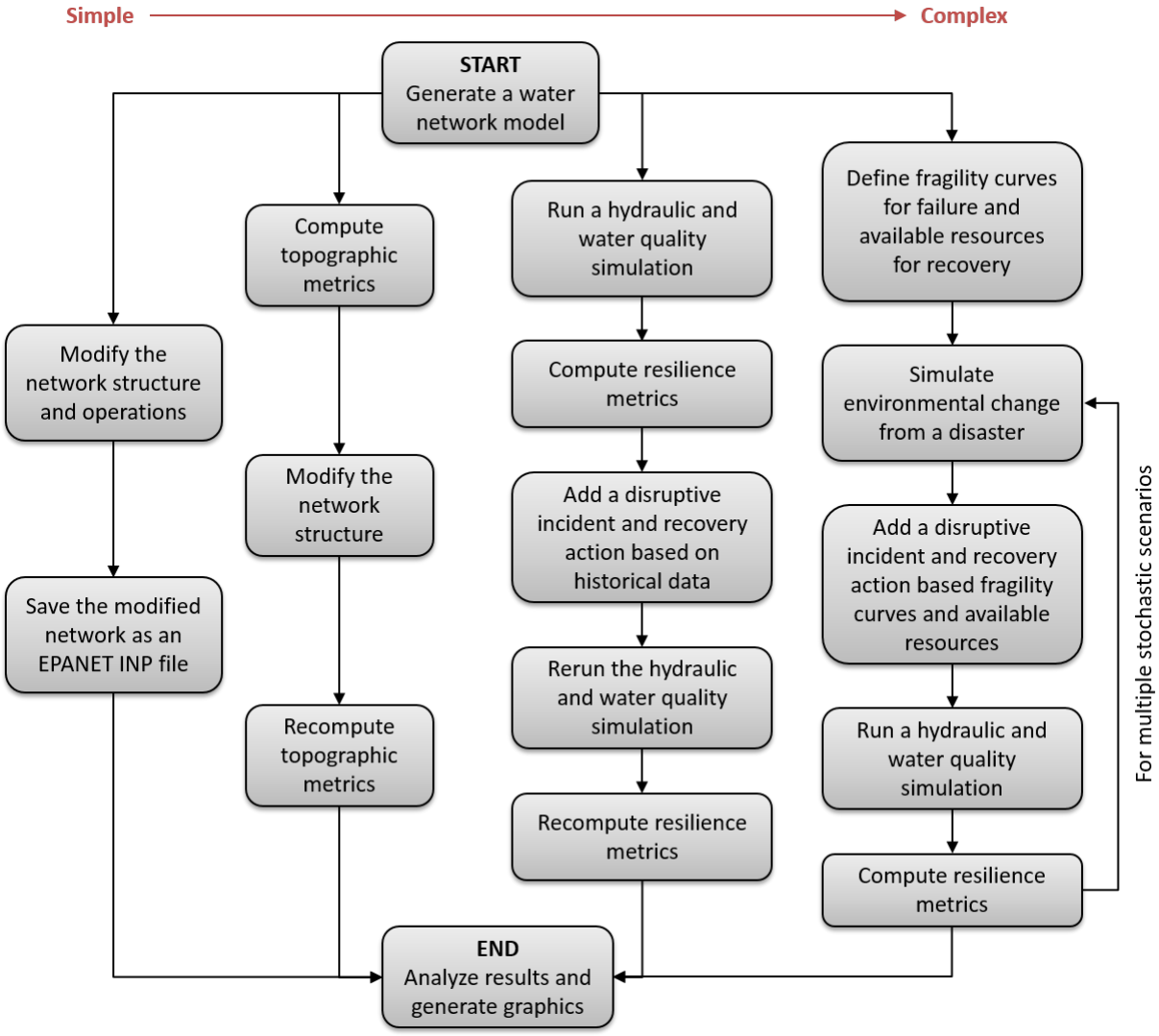

Figure 2: Flowchart illustrating four example use cases.

While EPANET includes some features to model and analyze water distribution system resilience, WNTR was developed to greatly extend these capabilities. WNTR provides a flexible platform for modeling a wide range of disruptive incidents and repair strategies, and pressure dependent demand hydraulic simulation is included to model the system during low pressure conditions. Furthermore, WNTR is compatible with widely used scientific computing packages for Python, including NetworkX [6], Pandas [10], Numpy [19], Scipy [19], and Matplotlib [7]. These packages allow the user to build custom analysis directly in Python, and gain access to tools that analyze the structure of complex water distribution networks, analyze time-series data from simulation results, run simulations efficiently, and create high-quality graphics and animations.

2 Installation

WNTR can be installed as a Python package using standard open source software tools.

Step 1: Setup your Python environment

Python can be installed on Windows, Linux, and Mac OS X operating systems. WNTR requires Python (versions 2.7, 3.4, or 3.5) along with several Python package dependencies. Python distributions, such as Anaconda, are recommended to manage the Python environment. Anaconda can be downloaded from https://www.continuum.io/downloads. General information on Python can be found at https://www.python.org/.

Anaconda includes the Python packages needed for WNTR, including Numpy, Scipy, NetworkX, Pandas, and Matplotlib. For more information on Python package dependencies, see *Requirements*.

Anaconda also comes with Spyder, an IDE, that includes enhanced editing and debug features along with a graphical user interface. Debugging options are available from the toolbar. Code documentation is displayed in the object inspection window. Pop-up information on class structure and functions is displayed in the editor and console windows.

To open a Python console, open a command prompt (cmd.exe on Windows, terminal window on Linux and Mac OS X) and run 'python', as shown in Figure 3, or open a Python console using an IDE, like Spyder, as shown in Figure 4.

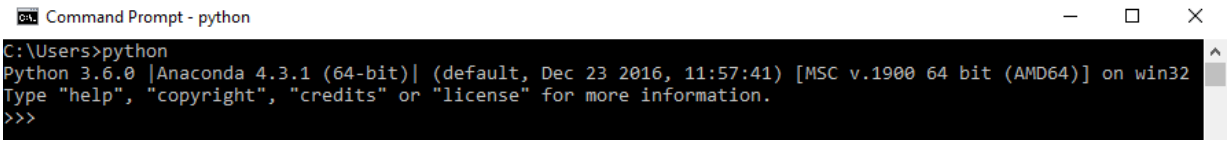

Figure 3: Opening a Python console from a command prompt.

Step 2: Install WNTR

The installation process differs for users and developers. Installation instructions for both types are described below.

For users: Users can install WNTR using pip, which is a command line software tool used to install and manage Python packages. It can be downloaded from https://pypi.python.org/pypi/pip.

To install WNTR using pip, open a command prompt and run:

```
pip install wntr
```

This will install the latest stable version of WNTR from https://pypi.python.org/pypi/wntr.

Note: A WNTR installation using pip will not include the examples folder, which is referenced throughout this manual.

Users can also download a zip file that includes source files and the examples folder from the US EPA GitHub organization. To download the master (development) branch, go to https://github.com/USEPA/WNTR, select the "Clone or download" button and then select "Download ZIP." This downloads a zip file called WNTR-master.zip. To download a specific release, go to https://github.com/USEPA/WNTR/releases and select a zip file. The software can then be installed by running a Python script, called setup.py, that is included in the zip file.

To build WNTR from the source files in the zip file, open a command prompt and run:

```
unzip WNTR-master.zip
cd WNTR-master
python setup.py install
```

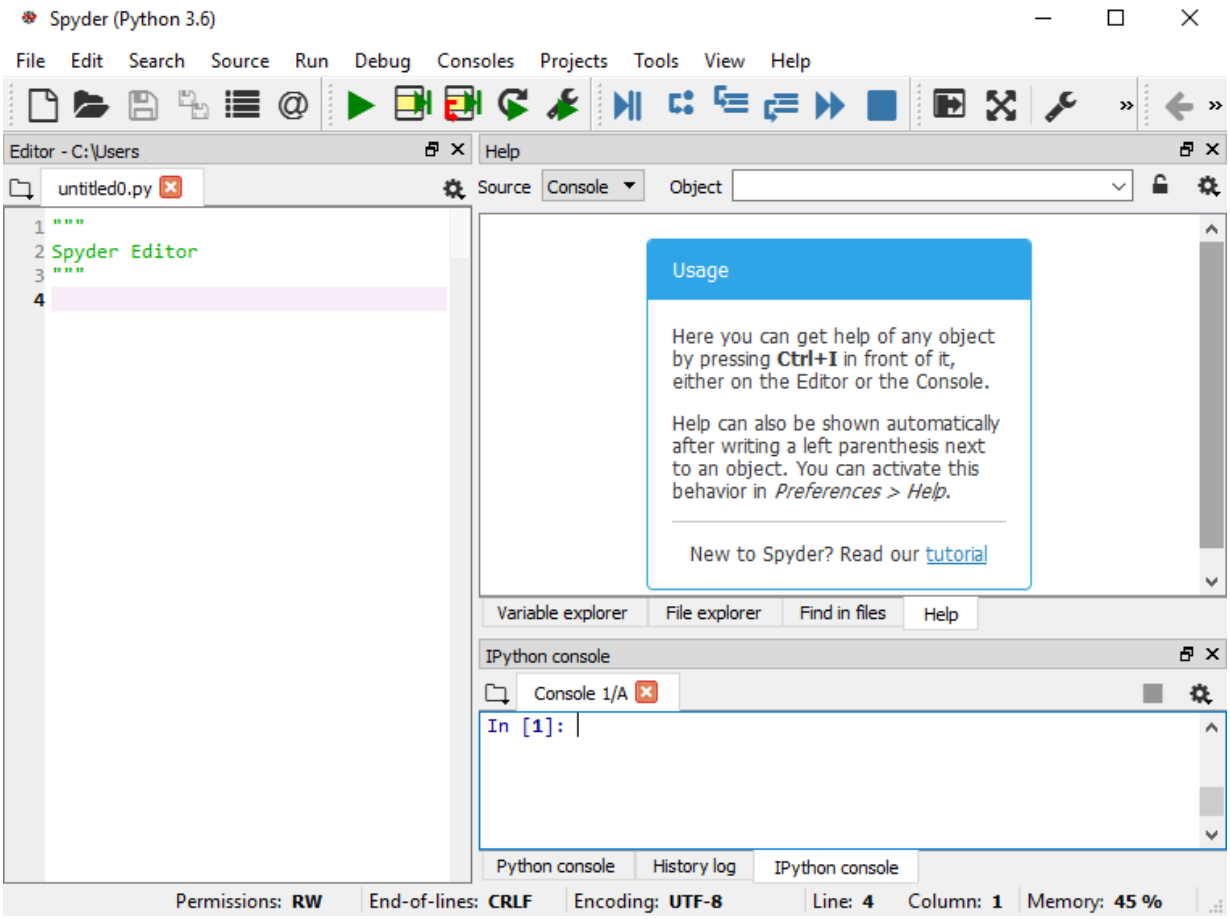

Figure 4: Opening a Python console using Spyder.

For developers: Developers can install and build WNTR from source using git, which is a command line software tool for version control and software development. It can be downloaded from http://git-scm.com.

To build WNTR from source using git, open a command prompt and run:

```
git clone https://github.com/USEPA/WNTR
cd wntr
python setup.py develop
```

This will install the master (development) branch of WNTR from https://github.com/USEPA/WNTR. More information for developers can be found in the *Software quality assurance* section.

Step 3: Test installation

To test that WNTR is installed, open a Python console and run:

```
import wntr
```

If WNTR is installed properly, Python proceeds to the next line. No other output is printed to the screen.

If WNTR is **not** installed properly, the user will see the following ImportError:

```
ImportError: No module named wntr
```

2.1 Requirements

Requirements for WNTR include Python (2.7, 3.4, or 3.5) along with several Python packages. The following Python packages are required:

- Numpy [19]: used to support large, multi-dimensional arrays and matrices, http://www.numpy.org/
- Scipy [19]: used to support efficient routines for numerical integration, http://www.scipy.org/
- NetworkX [6]: used to create and analyze complex networks, https://networkx.github.io/
- Pandas [10]: used to analyze and store time series data, http://pandas.pydata.org/
- enum34 (for Python 2.7): used to add enumerated type support for Python 2.7, https://pypi.python.org/pypi/enum34

These packages are included in the Anaconda Python distribution.

2.2 Optional dependencies

The following Python packages are optional:

- Matplotlib [7]: used to produce figures, http://matplotlib.org/
- Plotly [15]: used to produce interactive scalable figures, https://plot.ly/
- xlwt [22]: used to read/write to Microsoft® Excel® spreadsheets, http://xlwt.readthedocs.io
- Numpydoc [19]: used to build the user manual, https://github.com/numpy/numpydoc
- nose: used to run software tests, http://nose.readthedocs.io

These packages are included in the Anaconda Python distribution.

3 Software framework and limitations

Before using WNTR, it is helpful to understand the software framework. WNTR is a Python package, which contains several object-oriented subpackages, listed in Table 1. Each subpackage contains modules which contain classes, methods, and functions. See the online API documentation at https://wntr.readthedocs.io for more information on the code structure. The classes used to generate water network models and run simulations are described in more detail below, followed by a list of software limitations.

Table 1: WNTR Subpackages

Subpackage	Description
`epanet`	Contains EPANET 2 compatibility functions for WNTR.
`metrics`	Contains methods to compute resilience, including hydraulic, water quality, water security, and economic metrics. Methods to compute topographic metrics are included in the wntr.network.graph module.
`network`	Contains methods to define a water network model, network controls, and graph representation of the network.
`scenario`	Contains methods to define disaster scenarios and fragility/survival curves.
`sim`	Contains methods to run hydraulic and water quality simulations using the water network model.
`graphics`	Contains methods to generate graphics.
`utils`	Contains helper functions.

3.1 Water network model

The `network` subpackage contains classes to define the water network model, network controls, and graph representation of the network. These classes are listed in Table 2. Water network models can be built from scratch or built directly from EPANET INP files. Additionally, EPANET INP files can be generated from water network models.

Table 2: Classes in the *network* Subpackage

Class	Description
`WaterNetworkModel`	Contains methods to generate water network models, including methods to read and write INP files, and access/add/remove/modify network components. This class links to additional model classes (below) which define network components, controls, and model options.
`Junction`	Contains methods to define junctions. Junctions are nodes where links connect. Water can enter or leave the network at a junction.
`Reservoir`	Contains methods to define reservoirs. Reservoirs are nodes with an infinite external source or sink.
`Tank`	Contains methods to define tanks. Tanks are nodes with storage capacity.
`Pipe`	Contains methods to define pipes. Pipes are links that transport water.
`Pump`	Contains methods to define pumps. Pumps are links that increase hydraulic head.
`Energy`	Contains attributes for specifying global energy prices and global pump efficiencies.
`Valve`	Contains methods to define valves. Valves are links that limit pressure or flow.
`Curve`	Contains methods to define curves. Curves are data pairs representing a relationship between two quantities. Curves are used to define pump curves.
`Source`	Contains methods to define sources. Sources define the location and characteristics of a substance injected directly into the network.
`TimeControl`	Contains methods to define time controls. Time controls define actions that start or stop at a particular time.
`ConditionalControl`	Contains methods to define conditional controls. Conditional controls define actions that start or stop based on a particular condition in the network.
`WaterNetworkOptions`	Contains methods to define model options, including the simulation duration and time step.

3.2 Simulators

The `sim` subpackage contains classes to run hydraulic and water quality simulations using the water network model. WNTR contains two simulators: the EpanetSimulator and the WNTRSimulator. These classes are listed in Table 3.

Table 3: Classes in the *sim* Subpackage

Class	Description
`EpanetSimulator`	The EpanetSimulator uses the EPANET 2 Programmer's Toolkit [13] to run demand-driven hydraulic simulations and water quality simulations. When using the EpanetSimulator, the water network model is written to an EPANET INP file which is used to run an EPANET simulation. This allows the user to read in INP files, modify the model, run an EPANET simulation, and analyze results all within WNTR.
`WNTRSimulator`	The WNTRSimulator uses custom Python solvers to run demand-driven and pressure dependent demand hydraulic simulation and includes models to simulate pipe leaks. The WNTRSimulator does not perform water quality simulations.

3.3 Limitations

Current software limitations are noted:

- Certain EPANET INP model options are not supported in WNTR, as outlined below.
- Pressure dependent demand hydraulic simulation and leak models are only available using the WNTRSimulator.
- Water quality simulations are only available using the EpanetSimulator.

WNTR reads in and writes all sections of EPANET INP files. This includes the following sections: [BACKDROP], [CONTROLS], [COORDINATES], [CURVES], [DEMANDS], [EMITTERS], [ENERGY], [JUNCTIONS], [LABELS], [MIXING], [OPTIONS], [PATTERNS], [PIPES], [PUMPS], [QUALITY], [REACTIONS], [REPORT], [RESERVOIRS], [RULES], [SOURCES], [TAGS], [TANKS], [TIMES], [TITLE], [VALVES], and [VERTICES].

However, **the following model options cannot be modified/created in WNTR**:

- [BACKDROP] section
- Efficiency curves in the [CURVES] section
- [DEMANDS] section (base demand and patterns from the [JUNCTIONS] section can be modified)
- [EMITTERS] section
- [LABELS] section
- [MIXING] section
- [REPORT] section
- [VERTICES] section

While the EpanetSimulator uses all EPANET model options, several model options are not used by the WNTRSimulator. Of the EPANET model options that directly apply to hydraulic simulations, **the following options are not supported by the WNTRSimualtor**:

- [DEMANDS] section (base demand and patterns from the [JUNCTIONS] section are used)
- [EMITTERS] section
- D-W and C-M headloss options in the [OPTIONS] section (H-W option is used)
- Accuracy, unbalanced, demand multiplier, and emitter exponent from the [OPTIONS] section
- Speed option and multipoint head curves in the [PUMPS] section (3-point head curves are supported)
- Head pattern option in the [RESERVOIRS] section

- Volume curves in the [TANKS] section
- Rule timestep, pattern start, report start, start clocktime, and statistics in the [TIMES] section
- PSV, FCV, PBV, and GPV values in the [VALVES] section

Future development of WNTR will address these limitations.

4 Units

All data in WNTR is stored in SI (International System) units:

- Length = m
- Diameter = m
- Water pressure = m (this assumes a fluid density of 1000 kg/m^3)
- Elevation = m
- Mass = kg
- Time = s
- Concentration = kg/m^3
- Demand = m^3/s
- Velocity = m/s
- Acceleration = g (1 g = 9.81 m/s^2)
- Energy = J
- Power = W
- Pressure = Pa
- Mass injection = kg/s
- Volume = m^3

WNTR is compatible with all EPANET unit conventions. When using an EPANET INP file to generate a water network model, WNTR converts model parameters using the units defined in the **Units** and **Quality** options of the EPANET INP file. These options define the mass and flow units for the model. Some units also depend on the equation used for pipe roughness headloss and on the reaction order specified. Table 4, Table 5, and Table 6 provide information on EPANET unit conventions (modified from [13]).

Table 4: EPANET Hydraulic Unit Conventions

Hydraulic parameter	US customary units	SI-based units
Flow	*flow* can be defined as: • CFS: ft ^{3} /s • GPM: gal/min • MGD: million gal/day • IMGD: million imperial gal/day • AFD: acre-feet/day	*flow* can be defined as: • LPS: L/s • LPM: L/min • MLD: million L/day • CMH: m ^{3} /hr • CMD: m ^{3} /day
Demand	*flow*	*flow*
Diameter: pipes	in	mm
Diameter: tanks	ft	m
Elevation	ft	m
Hydraulic head	ft	m
Length	ft	m
Emitter coefficient	*flow* / sqrt(psi)	*flow* / sqrt(m)
Friction factor	unitless	unitless
Minor loss coefficient	unitless	unitless
Pressure	psi	m or kPa
Roughness coeff: D-W	10 $^{-3}$ ft	mm
Roughness coeff: H-W, C-M	unitless	unitless
Velocity	ft/s	m/s
Volume	ft ^{3}	m ^{3}

Table 5: EPANET Water Quality Unit Conventions

Water quality parameter	US customary units	SI-based units
Concentration	*mass* /L where *mass* can be defined as mg or ug	*mass* /L where *mass* can be defined as mg or ug
Bulk reaction coefficient: order-1	1/day	1/day
Wall reaction coefficient: order-0	*mass* /ft ^{2} /day	*mass* /m ^{2} /day
Wall reaction coefficient: order-1	ft/day	m/day
Reaction rate	*mass* /L/day	*mass* /L/day
Source mass injection rate	*mass* /min	*mass* /min
Water age	hours	hours

Table 6: EPANET Energy Unit Conventions

Energy parameter	US customary units	SI-based units
Energy	kW-hours	kW-hours
Efficiency (pumps)	percent	percent
Power	hp (horse-power)	kW

When running analysis in WNTR, all input values (i.e., time, pressure threshold, node demand) should be specified in SI units. All simulation results are also stored in SI units and can be converted to other units if desired. The SymPy Python package can be used to convert between units [9].

5 Getting started

To start using WNTR, open a Python console and import the package:

```
import wntr
```

A simple script, **getting_started.py**, is included in the examples folder. This example demonstrates how to:

- Import WNTR
- Generate a water network model
- Simulate hydraulics
- Plot simulation results on the network

```
import wntr

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

# Graph the network
wntr.graphics.plot_network(wn, title=wn.name)

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

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

Additional examples, listed in Table 7, are included in the examples folder.

Table 7: Description of WNTR Example Files

Example file	Description
water_network_model.py	Generate and modify water network models
networkx_graph.py	Generate a NetworkX graph from a water network model
hydraulic_simulation.py	Simulate hydraulics using the EPANET and WNTR simulators
water_quality_simulation.py	Simulate water quality using EPANET
simulation_results.py	Extract information from simulation results
disaster_scenarios.py	Define disaster scenarios, including power outage, pipe leak, and changes to supply and demand
resilience_metrics.py	Compute resilience metrics, including topographic, hydraulic, water quality, water security, and economic metrics
stochastic_simulation.py	Run a stochastic simulation
fragility_curves.py	Define fragility curves
interactive_graphics.py	Create interactive network and time series graphics
animation.py	Animate network graphics

Several EPANET INP files are included in the examples/network folder. Example networks range from a simple 9 node network to a 3,000 node network. Additional network models can be downloaded from the University of Kentucky Water Distribution System Research Database at http://www.uky.edu/WDST/database.html.

6 Water network model

The water network model includes junctions, tanks, reservoirs, pipes, pumps, valves, demand patterns, pump curves, controls, sources, simulation options, and node coordinates. Water network models can be built from scratch or built directly from an EPANET INP file. Sections of EPANET INP file that are not compatible with WNTR are described in *Limitations*. The example **water_network_model.py** can be used to generate, save, and modify water network models.

A water network model can be created by adding components to an empty model:

```
wn = wntr.network.WaterNetworkModel()
wn.add_pattern('pat1', [1])
wn.add_pattern('pat2', [1,2,3,4,5,6,7,8,9,10])
wn.add_junction('node1', base_demand=0.01, demand_pattern_name='pat1',
        elevation=100.0, coordinates=(1,2))
wn.add_junction('node2', base_demand=0.02, demand_pattern_name='pat2',
        elevation=50.0, coordinates=(1,3))
wn.add_pipe('pipe1', 'node1', 'node2', length=304.8, diameter=0.3048, roughness=100,
        minor_loss=0.0, status='OPEN')
wn.add_reservoir('res', base_head=125, head_pattern_name='pat1', coordinates=(0,2))
wn.add_pipe('pipe2', 'node1', 'res', length=100, diameter=0.3048, roughness=100,
        minor_loss=0.0, status='OPEN')
wn.options.duration = 24*3600
wn.options.hydraulic_timestep = 15*60
wn.options.pattern_timestep = 60*60
```

A water network model can also be created directly from an EPANET INP file:

```
inp_file = 'networks/Net3.inp'
```

The water network model can be written to a file in EPANET INP format. By default, files are written in LPS units. The EPANET INP file will not include features not supported by EPANET (i.e., pressure dependent demand simulation options).:

```
wn.write_inpfile('filename.inp')
```

For more information on the water network model, see `WaterNetworkModel` in the API documentation.

7 Water network controls

One of the key features of water network models is the ability to control pipes, pumps, and valves using simple and complex conditions. EPANET uses "controls" and "rules" to define conditions [13]. A control is a single action (i.e., closing/opening a link or changing the setting) based on a single condition (i.e., time based or tank level based). A rule is more complex; rules take an IF-THEN-ELSE form and can have multiple conditions and multiple actions in each of the logical blocks. WNTR supports EPANET's rules and controls when generating a water network model from an INP file and simulating hydraulics using the EpanetSimulator. WNTR includes additional options to define controls that can be used by the WNTRSimulator.

The basic steps to define a control for a water network model are:

1. Define the control action
2. Define the control or rule using the control action
3. Add the control or rule to the network

These steps are defined below. Examples use the "Net3.inp" EPANET INP file to generate the water network model object, called *wn*.

7.1 Control actions

Control actions tell the simulator what to do when a condition becomes "true." Control actions are created using the `ControlAction` class. A control action is defined by a target link, the property to change, and the value to change it to. The following example creates a control action that opens pipe 330:

```
>>> import wntr.network.controls as controls
>>> l1 = wn.get_link('330')
>>> act1 = controls.ControlAction(l1, 'status', 1)
>>> print(act1)
set Pipe('330').status to Open
```

7.2 Simple controls

Control objects that emulate EPANET's [CONTROLS] section are defined in two classes: `ConditionalControl` and `TimeControl`. When generating a water network model from an EPANET INP file, a ConditionalControl or TimeControl will be created for each control.

Conditional controls: ConditionalControl objects define tank level and junction pressure based controls. Conditional controls require a source, operation, threshold, and a control action. The source is defined as tuple where the first value is a water network model component and the second value is the attribute of the object. The operation is defined using NumPy functions such as *np.greater* and *np.less*. The threshold is the value that triggers the condition to be true. The control action is defined above.

In the following example, a conditional control is defined that opens pipe 330 if the level of tank 1 goes above 46.0248 m. The source is the tank level and is defined as a tuple with the node object *n1* and the attribute *level*. To specify that the condition should be true when the level is greater than the threshold, the operation is set to *np.greater* and the threshold is set to 46.0248. The control action *act1* from above is used in the conditional control:

```
>>> n1 = wn.get_node('1')
>>> thresh1 = 46.0248
>>> ctrl1 = controls.ConditionalControl( (n1, 'level'), np.greater, thresh1, act1)
>>> ctrl1
<ConditionalControl: <Tank '1'>, 'level'), <ufunc 'greater'>, 46.0248,
 ↪<ControlAction: <Pipe '330'>, 'status', 'Open'>>
```

To get an EPANET-like description of this control, use the print function:

```
>>> print(ctrl1)
LINK 330 Open IF NODE 1 Above 46.0248
```

Time-based controls: TimeControl objects define time-based controls. Time-based controls require a water network model object, a time to start the condition, a control action, and additional flags to specify the time reference point and recurrence. The time flag is either *SIM_TIME* or *SHIFTED_TIME*; these indicate simulation or clock time, respectively. The daily flag is either True or False and indicates if the control should be repeated every 24 hours.

In the following example, a time-based control is defined that opens Pump 10 at hour 121. The time flag is set to *SIM_TIME* and the daily flag is set to False. A new control action is defined that opens the pump:

```
>>> time2 = 121 * 60 * 60
>>> timeflag2 = 'SIM_TIME'
>>> dailyflag2 = False
>>> pump2 = wn.get_link('10')
>>> act2 = controls.ControlAction(pump2, 'status', 1)
>>> ctrl2 = controls.TimeControl(wn, time2, timeflag2, dailyflag2, act2)
>>> print(ctrl2)
LINK 10 Open AT TIME 121:00:00
```

Note that the EpanetSimulator is limited to use the following pairs: time_flag='SIM_TIME' with daily_flag=False, and time_flag='SHIFTED_TIME' with daily_flag=True. The WNTRSimulator can use any combination of time flag and daily flag.

7.3 Complex rules

Control objects that emulate EPANET's [RULES] section are defined in the `IfThenElseControl` class. When generating a water network model from an EPANET INP file, an IfThenElseControl will be created for each rule. An IfThenElseControl is defined using a `ControlCondition` object and a `ControlAction` object. Condition classes are listed in Table 8.

Table 8: Condition Classes

Condition class	Description
`TimeOfDayCondition`	Time-of-day or "clocktime" based condition statement
`SimTimeCondition`	Condition based on time since start of the simulation
`ValueCondition`	Compare a network element attribute to a set value
`RelativeCondition`	Compare attributes of two different objects (e.g., levels from tanks 1 and 2)
`OrCondition`	Combine two WNTR Conditions with an OR
`AndCondition`	Combine two WNTR Conditions with an AND

All of the above conditions are valid EPANET conditions except RelativeCondition.

In the following example, the previous simple controls are recreated using the IfThenElseControl:

```
>>> cond1 = controls.ValueCondition(n1, 'level', '>', 46.0248)
>>> print(cond1)
Tank('1').level > 46.0248

>>> rule1 = controls.IfThenElseControl(cond1, [act1], name='control1')
>>> print(rule1)
Rule control1 := if Tank('1').level > 46.0248 then set Pipe('330').status to Open

>>> cond2 = controls.SimTimeCondition(wn, '=', '121:00:00')
>>> print(cond2)
sim_time = 435600 sec

>>> rule2 = controls.IfThenElseControl(cond2, [act2], name='control2')
>>> print(rule2)
Rule control2 := if sim_time = 435600 sec then set Pump('10').status to Open
```

More complex rules can be written using one of the Boolean logic condition classes. The following example creates a new rule that will open pipe 330 if both conditions are true, and otherwise it will open pipe 10. This rule will behave very differently from the rules above:

```
>>> cond3 = controls.AndCondition(cond1, cond2)
>>> print(cond3)
( Tank('1').level > 46.0248 && sim_time = 435600 sec )

>>> rule3 = controls.IfThenElseControl(cond3, [ act1 ], [ act2 ], priority=3, name=
↪'weird')
>>> print(rule3)
Rule weird := if ( Tank('1').level > 46.0248 && sim_time = 435600 sec ) then set␣
↪Pipe('330').status to Open else set Pump('10').status to Open with priority 3
```

Actions can also be combined, as shown in the following example:

```
>>> cond4 = controls.OrCondition(cond1, cond2)
>>> rule4 = controls.IfThenElseControl(cond4, [act1, act2])
>>> print(rule4)
Rule  := if ( Tank('1').level > 46.0248 || sim_time = 435600 sec ) then set Pipe('330
↪').status to Open and set Pump('10').status to Open
```

The flexibility of the IfThenElseControl combined with the different ControlCondition classes and ControlActions provides an extremely powerful tool for defining complex network operations.

7.4 Adding controls to a network

Once a control is created, they can be added to the network. This is accomplished using the `add_control` method of the water network model object. The control should be named so that it can be retrieved and modified if desired:

```
>>> wn.add_control('NewTimeControl', ctrl2)
>>> wn.get_control('NewTimeControl')
<TimeControl: model, 435600, 'SIM_TIME', False, <ControlAction: <Pump '10'>, 'status
↪', 'Open'>>
```

8 NetworkX graph

WNTR uses NetworkX data objects to store network connectivity as a graph. A **graph** is a collection of nodes that are connected by links. For water networks, nodes represent junctions, tanks, and reservoirs while links represent pipes, pumps, and valves.

Water networks are stored as directed multigraphs. A **directed multigraph** is a graph with direction associated with links and the graph can have multiple links with the same start and end node. A simple example is shown in Figure 5. For water networks, the link direction is from the start node to the end node. The link direction is used as a reference to track flow direction in the network. For example, positive flow indicates that the flow direction is from the start node to the end node while negative flow indicates that the flow direction is from the end node to the start node. Multiple links with the same start and end node can be used to represent redundant pipes or backup pumps. In WNTR, the graph stores the start and end node of each link, node coordinates, and node and link types (i.e., tank, reservoir, valve). WNTR updates the graph as elements are added and removed from the water network model.

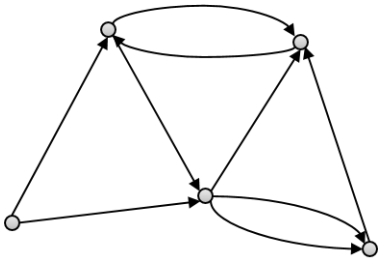

Figure 5: Example directed multigraph.

NetworkX includes numerous methods to analyze the structure of complex networks. For more information on NetworkX, see https://networkx.github.io/. WNTR includes a custom Graph Class, `WntrMultiDiGraph`. This class inherits from NetworkX MultiDigraph and includes additional methods that are specific to WNTR. The example **networkx_graph.py** can be used to generate a graph from a water network model.

A copy of the graph can an be obtained using the following function:

```
G = wn.get_graph_deep_copy()
```

The graph is stored as a nested dictionary. The nodes and links (note that links are called *edges* in NetworkX) can be accessed using the following:

```
node_name = '123'
print(G.node[node_name])
print(G.edge[node_name])
```

The graph can be used to access NetworkX methods, for example:

```
import networkx as nx
node_degree = G.degree()
bet_cen = nx.betweenness_centrality(G)
wntr.graphics.plot_network(wn, node_attribute=bet_cen, node_size=30,
                           title='Betweenness Centrality')
```

Some methods in NetworkX require that networks are undirected. An **undirected graph** is a graph with no direction associated with links. The following NetworkX method can be used to convert a directed graph to an undirected graph:

```
uG = G.to_undirected()
```

Some methods in NetworkX require that networks are connected. A **connected graph** is a graph where a path exists between every node in the network (i.e., no node is disconnected). The following NetworkX method can be used to check if a graph is connected:

```
print(nx.is_connected(uG))
```

Some methods in NetworkX can use weighted graphs. A **weighted graph** is a graph in which each link is given a weight. The WNTR method `weight_graph` can be used to weight the graph by any attribute. In the following example, the graph is weighted by length. This graph can then be used to compute path lengths:

```
length = wn.query_link_attribute('length')
G.weight_graph(link_attribute = length)
```

9 Hydraulic simulation

WNTR contains two simulators: the **WNTRSimulator** and the **EpanetSimulator**. See *Software framework and limitations* for more information on features and limitations of these simulators.

The EpanetSimulator can be used to run demand-driven hydraulic simulations using the EPANET 2 Programmer's Toolkit. The simulator can also be used to run water quality simulations, as described in *Water quality simulation*. A hydraulic simulation using the EpanetSimulator is run using the following code:

```
epanet_sim = wntr.sim.EpanetSimulator(wn)
epanet_sim_results = epanet_sim.run_sim()
```

The WNTRSimulator is a pure Python hydraulics simulation engine based on the same equations as EPANET. The WNTRSimulator does not include equations to run water quality simulations. The WNTRSimulator includes the option to simulate leaks, and run hydraulic simulations in either demand-driven or pressure dependent demand mode. A hydraulic simulation using the WNTRSimulator is run using the following code:

```
wntr_sim = wntr.sim.WNTRSimulator(wn)
wntr_sim_results = wntr_sim.run_sim()
```

The example **hydraulic_simulation.py** can be used to run both simulators.

More information on the simulators can be found in the API documentation, under `EpanetSimulator` and `WNTRSimulator`.

9.1 Options

Hydraulic simulation options are defined in the `WaterNetworkOptions` class. These options include duration, hydraulic timestep, rule timestep, pattern timestep, pattern start, default pattern, report timestep, report start, start clocktime, headloss, trials, accuracy, unbalanced, demand multiplier, and emitter exponent. All options are used with the EpanetSimulator. Options that are not used with the WNTRSimulator are described in *Limitations*.

9.2 Mass balance at nodes

Both simulators use the mass balance equations from EPANET [13]:

$$\sum_{p \in P_n} q_{p,n} - D_n^{act} = 0 \qquad \forall n \in N$$

where P_n is the set of pipes connected to node n, $q_{p,n}$ is the flow rate of water into node n from pipe p (m^3/s), D_n^{act} is the actual demand out of node n (m^3/s), and N is the set of all nodes. If water is flowing out of node n and into pipe p, then $q_{p,n}$ is negative. Otherwise, it is positive.

9.3 Headloss in pipes

Both simulators use the Hazen-Williams headloss formula from EPANET [13]:

$$H_{n_j} - H_{n_i} = h_L = 10.667 C^{-1.852} d^{-4.871} L q^{1.852}$$

where h_L is the headloss in the pipe (m), C is the Hazen-Williams roughness coefficient (unitless), d is the pipe diameter (m), L is the pipe length (m), q is the flow rate of water in the pipe (m^3/s), H_{n_j} is the head at the starting node (m), and H_{n_i} is the head at the ending node (m).

The flow rate in a pipe is positive if water is flowing from the starting node to the ending node and negative if water is flowing from the ending node to the starting node.

The WNTRSimulator solves for pressures and flows throughout the network as a set of linear equations. However, the Hazen-Williams headloss formula is not valid for negative flow rates. Therefore, the WNTRSimulator uses a reformulation of this constraint.

For $q < 0$:

$$h_L = -10.667 C^{-1.852} d^{-4.871} L|q|^{1.852}$$

For $q \geq 0$:

$$h_L = 10.667 C^{-1.852} d^{-4.871} L|q|^{1.852}$$

These equations are symmetric across the origin and valid for any q. Thus, this equation can be used for flow in either direction. However, the derivative with respect to q at $q = 0$ is 0. In certain scenarios, this can cause the Jacobian of the set of hydraulic equations to become singular (when $q = 0$). To overcome this limitation, the WNTRSimulator splits the domain of q into six segments to create a piecewise smooth function.

9.4 Demand-driven simulation

In demand-driven simulation, the pressure in the system depends on the node demands. The mass balance and headloss equations described above are solved assuming that node demands are known and satisfied. This assumption is reasonable under normal operating conditions and for use in network design. Both simulators can run hydraulics using demand-driven simulation.

9.5 Pressure dependent demand simulation

In situations that lead to low pressure conditions (i.e., fire fighting, power outages, pipe leaks), consumers do not always receive their requested demand and a pressure dependent demand simulation is recommended. In a pressure dependent demand simulation, the delivered demand depends on the pressure. The mass balance and headloss equations described above are solved by simultaneously determining demand along with the network pressures and flow rates.

The WNTRSimulator can run hydraulics using a pressure dependent demand simulation according to the following pressure-demand relationship [20]:

$$d = \begin{cases} 0 & p \leq P_0 \\ D_f \left(\frac{p - P_0}{P_f - P_0}\right)^{\frac{1}{2}} & P_0 \leq p \leq P_f \\ D^f & p \geq P_f \end{cases}$$

where d is the actual demand (m^3/s), D_f is the desired demand (m^3/s), p is the pressure (Pa), P_f is the pressure above which the consumer should receive the desired demand (Pa), and P_0 is the pressure below which the consumer cannot receive any water (Pa). The set of nonlinear equations comprising the hydraulic model and the pressure-demand relationship is solved directly using a Newton-Raphson algorithm.

Figure 6 illustrates the pressure-demand relationship using both the demand-driven and pressure dependent demand simulations. In the example, D_f is 0.0025 m^3/s (39.6 GPM), P_f is 30 psi, and P_0 is 5 psi. Using the demand-driven simulation, the demand is equal to D_f regardless of pressure. Using the pressure dependent demand simulation, the demand starts to decrease when the pressure is below P_f and goes to 0 when pressure is below P_0.

9.6 Leak model

The WNTRSimulator includes the ability to add leaks to the network. The leak is modeled with a general form of the equation proposed by Crowl and Louvar [4] where the mass flow rate of fluid through the hole is expressed as:

$$d_{leak} = C_d A p^{\alpha} \sqrt{\frac{2}{\rho}}$$

where d_{leak} is the leak demand (m^3/s), C_d is the discharge coefficient (unitless), A is the area of the hole (m^2), p is the gauge pressure inside the pipe (Pa), α is the discharge coefficient, and ρ is the density of the fluid. The default discharge coefficient is 0.75 (assuming turbulent flow), but the user can specify other values if needed. The value of α is set to 0.5 (assuming large leaks out of steel pipes). Leaks can be added to junctions and tanks. A pipe break is modeled using a leak area large enough to drain the pipe. WNTR includes methods to add leaks to any location along a pipe by splitting the pipe into two sections and adding a node.

Figure 7 illustrates leak demand. In the example, the diameter of the leak is set to 0.5 cm, 1.0 cm, and 1.5 cm.

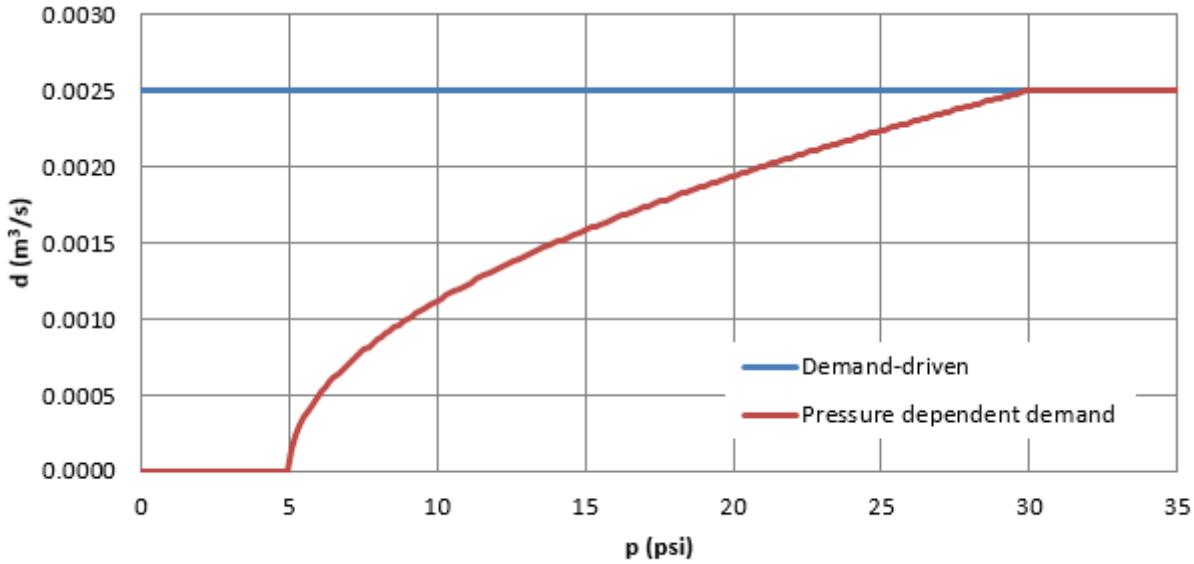

Figure 6: Example relationship between pressure (p) and demand (d) using both the demand-driven and pressure dependent demand simulations.

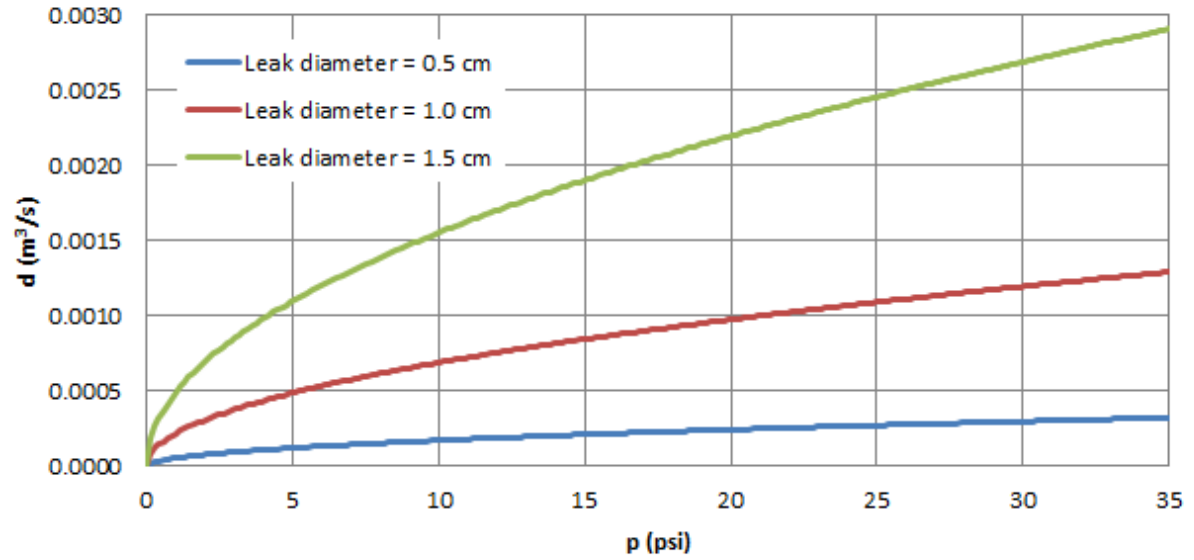

Figure 7: Example relationship between leak demand (d) and pressure (p).

9.7 Pause and restart

The WNTRSimulator includes the ability to

- Reset initial values and re-simulate using the same water network model. Initial values include tank head, reservoir head, pipe status, pump status, and valve status.
- Pause a hydraulic simulation, change network operations, and then restart the simulation
- Save the water network model and results to files and reload for future analysis

These features are helpful when evaluating various response action plans or when simulating long periods of time where the time resolution might vary. The file **hydraulic_simulation.py** includes examples of these features.

10 Water quality simulation

Water quality simulations can only be run using the **EpanetSimulator**. As listed in the *Software framework and limitations* section, this means that the hydraulic simulation must use demand-driven simulation. Note that the WNTRSimulator can be used to compute demands under pressure dependent demand conditions and those demands can be used in the EpanetSimulator. The following code illustrates how to reset demands in a water network model using a pressure dependent demand simulation:

```
sim = wntr.sim.WNTRSimulator(wn)
results = sim.run_sim()
wn.reset_demand(results.node['demand'], 'PDD')
sim = wntr.sim.EpanetSimulator(wn)
results_withPDdemands = sim.run_sim()
```

After defining water quality options and sources (described in the *Options* and *Sources* sections below), a hydraulic and water quality simulation using the EpanetSimualtor is run using the following code:

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

The example **water_quality_simulation.py** can be used to run water quality simulations and plot results.

10.1 Options

Water quality simulation options are defined in the `WaterNetworkOptions` class. Three types of water quality analysis are supported. These options include water age, tracer, and chemical concentration.

- **Water age**: Water quality simulation can be used to compute water age at every node. To compute water age, set the 'quality' option as follows:

```
wn.options.quality = 'AGE'
```

- **Tracer**: Water quality simulation can be used to compute the percent of flow originating from a specific location. The results include tracer percent values at each node. For example, to track a tracer from node '111', set the 'quality' and 'tracer_node' options as follows:

```
wn.options.quality = 'TRACE'
wn.options.quality_value = '111'
```

- **Chemical concentration**: Water quality simulation can be used to compute chemical concentration given a set of source injections. The results include chemical concentration values at each node. To compute chemical concentration, define sources (described in the *Sources* section below) and set the 'quality' options as follows:

```
wn.options.quality = 'CHEMICAL'
```

- To skip the water quality simulation, set the 'quality' options as follows:

```
wn.options.quality = 'NONE'
```

Additional water quality options include viscosity, diffusivity, specific gravity, tolerance, bulk reaction order, wall reaction order, tank reaction order, bulk reaction coefficient, wall reaction coefficient, limiting potential, and roughness correlation. These parameters are defined in the `WaterNetworkOptions` API documentation.

When creating a water network model from an EPANET INP file, water quality options are populated from the [OPTIONS] and [REACTIONS] sections of EPANET INP file. All of these options can be modified in WNTR and then written to an EPANET INP file.

10.2 Sources

Sources are required for CHEMICAL water quality analysis. Sources can still be defined, but *will not* be used if AGE, TRACE, or NONE water quality analysis is selected. Sources are added to the water network model using the `add_source` method. Sources include the following information:

- **Source name**: A unique source name used to reference the source in the water network model.
- **Node name**: The injection node.
- **Source type**: Options include 'CONCEN,' 'MASS,' 'FLOWPACED,' or 'SETPOINT.'
  - CONCEN source represents injection of a specific concentration.
  - MASS source represents a booster source with a fixed mass flow rate.
  - FLOWPACED source represents a booster source with a fixed concentration at the inflow of the node.
  - SETPOINT source represents a booster source with a fixed concentration at the outflow of the node.
- **Strength**: Baseline source strength (in mass/time for MASS and mass/volume for CONCEN, FLOWPACED, and SETPOINT).
- **Pattern**: The pattern name associated with the injection.

For example, the following code can be used to add a source, and associated pattern, to the water network model:

```
wn.add_pattern('SourcePattern', start_time=2*3600, end_time=15*3600)
wn.add_source('Source1', '121', 'SETPOINT', 1000, 'SourcePattern')
```

In the above example, the pattern is given a value of 1 between 2 and 15 hours, and 0 otherwise. The method `remove_source` can be used to remove sources from the water network model.

When creating a water network model from an EPANET INP file, the sources that are defined in the [SOURCES] section are added to the water network model. These sources are given the name 'INP#' where # is an integer related to the number of sources in the INP file.

11 Simulation results

WNTR uses Pandas data objects to store simulation results. The use of Pandas facilitates a comprehensive set of time series analysis options that can be used to evaluate results. For more information on Pandas, see http://pandas.pydata.org/.

Results are stored in Pandas Panels. A Panel is a 3-dimensional database. One Panel is used to store nodes results and one Panel is used to store link results. The Panels are indexed by:

- Node or link attribute
- Time in seconds from the start of the simulation
- Node or link name

Conceptually, Panels can be visualized as blocks of data with 3 axis, as shown in Figure 8.

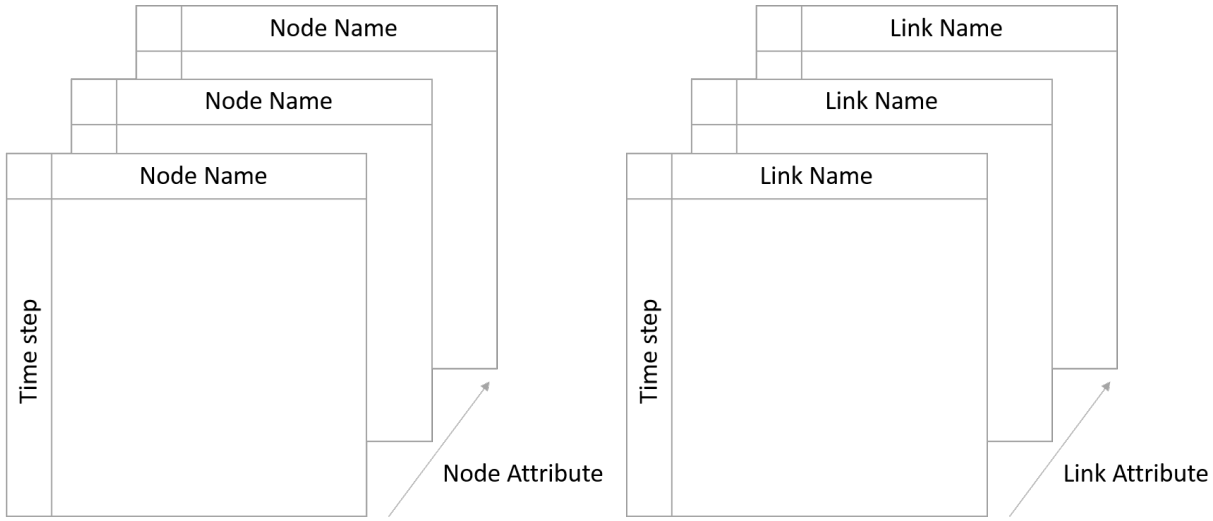

Figure 8: Conceptual representation of Panels used to store simulation results.

Node attributes include:

- Demand
- Expected demand
- Leak demand (only when the WNTRSimulator is used)
- Pressure
- Head
- Quality (only when the EpanetSimulator is used for a water quality simulation. Water age, tracer percent, or chemical concentration is stored, depending on the type of water quality analysis)
- Type (junction, tank, or reservoir)

Link attributes include:

- Velocity
- Flowrate
- Status (0 indicates closed, 1 indicates open)
- Type (pipe, pump, or valve)

The example **simulation_results.py** demonstrates use cases of simulation results. Node and link results are accessed using:

```
print(results.node)
print(results.link)
```

The indices can be used to extract specific information from the Panels. For example, to access the pressure and demand at node '123' at 1 hour:

```
print(results.node.loc[['pressure', 'demand'], 3600, '123'])
```

To access the pressure for all nodes and times (the ":" notation returns all variables along the specified axis):

```
print(results.node.loc['pressure', :, :])
```

Attributes can be plotted as a time-series, as shown in Figure 9:

```
pressure_at_node123 = results.node.loc['pressure', :, '123']
pressure_at_node123.plot()
```

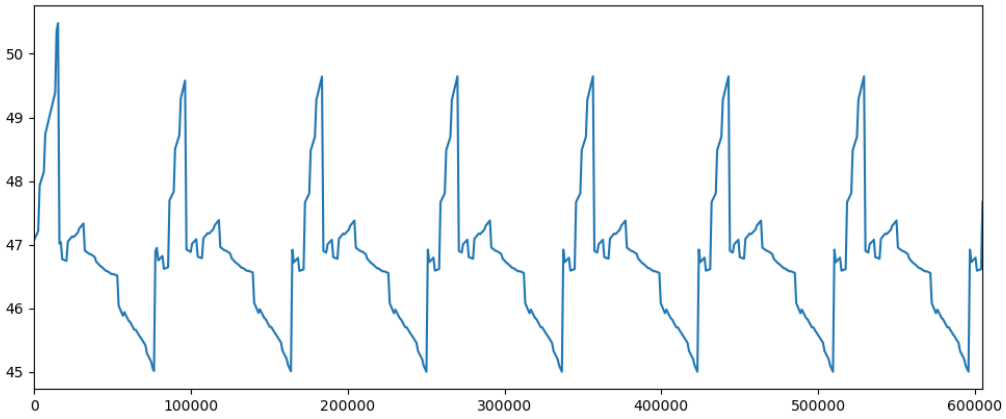

Figure 9: Example time-series graphic.

Attributes can be plotted on the water network model, as shown in Figure 10. In this figure, the node pressure at 1 hr and link flowrate at 1 hour are plotted on the network. A colorbar is included for both node and link attributes:

```
pressure_at_1hr = results.node.loc['pressure', 3600, :]
flowrate_at_1hr = results.link.loc['flowrate', 3600, :]
wntr.graphics.plot_network(wn, node_attribute=pressure_at_1hr,
                           link_attribute=flowrate_at_1hr)
```

Network and time-series graphics can be customized to add titles, legends, axis labels, etc.

Panels can be saved to Excel files using:

```
results.node.to_excel('node_results.xls')
results.link.to_excel('link_results.xls')
```

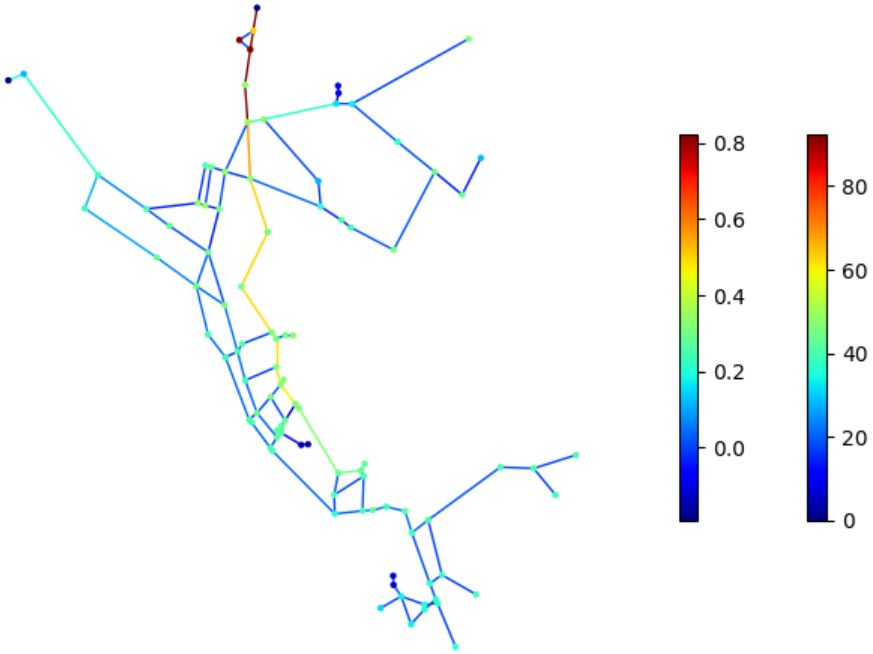

Figure 10: Example network graphic.

12 Disaster scenarios

Drinking water utilities might be interested in examining many different disaster scenarios. They could be acute incidents like power outages and earthquakes or they could be long term issues like persistent pipe leaks, population fluctuation, and changes to supply and demand. The following section describes disaster scenarios that can be modeled in WNTR. The example **disaster_scenarios.py** demonstrates methods to define disaster scenarios.

12.1 Earthquake

Earthquakes can be some of the most sudden and impactful disasters that a water systems experiences. An earthquake can cause lasting damage to the system that could take weeks, if not months, to fully repair. Earthquakes can cause damage to pipes, tanks, pumps, and other infrastructure. Additionally, earthquakes can cause power outages and fires.

WNTR includes methods to add leaks to pipes and tanks, shut off power to pumps, and change demands for fire conditions, as described in the sections below. The `Earthquake` class includes methods to compute peak ground acceleration, peak ground velocity, and repair rate based on the earthquake location and magnitude. Alternatively, external earthquake models or databases (e.g., ShakeMap [21]) can be used to compute earthquake properties and those properties can be loaded into Python for analysis in WNTR.

When simulating the effects of an earthquake, fragility curves are commonly used to define the probability that a component is damaged with respect to peak ground acceleration, peak ground velocity, or repair rate. The American Lifelines Alliance report [1] includes seismic fragility curves for water system components. See *Stochastic simulation* for more information on fragility curves.

Since properties like peak ground acceleration, peak ground velocity, and repair rate are a function of the distance to the epicenter, node coordinates in the water network model must be in units of meters. Since some network models use other units for node coordinates, WNTR includes a method to change the coordinate scale. To change the node coordinate scale by a factor of 1000, for example, use the following code:

```
wn.scale_node_coordinates(1000)
```

The following code can be used to compute peak ground acceleration, peak ground velocity, and repair rate:

```
epicenter = (32000,15000) # x,y location
magnitude = 6.5 # Richter scale
depth = 10000 # m, shallow depth
earthquake = wntr.scenario.Earthquake(epicenter, magnitude, depth)
distance = earthquake.distance_to_epicenter(wn, element_type=wntr.network.Pipe)
pga = earthquake.pga_attenuation_model(distance)
pgv = earthquake.pgv_attenuation_model(distance)
repair_rate = earthquake.repair_rate_model(pgv)
```

12.2 Pipe breaks or leaks

Pipes are susceptible to leaks. Leaks can be caused by aging infrastructure, the freeze/thaw process, increased demand, or pressure changes. This type of damage is especially common in older cities where distribution systems were constructed from outdated materials like cast iron and even wood.

WNTR includes methods to add leaks to junctions and tanks. Leaks can be added to a pipe by splitting the pipe and adding a junction. To add a leak to a specific pipe:

```
wn.split_pipe_with_junction('123', '123_A', '123_B', '123_leak_node')
leak_node = wn.get_node('123_leak_node')
leak_node.add_leak(wn, area=0.05, start_time=2*3600, end_time=12*3600)
```

The method `add_leak` adds time controls to a junction which includes the start and stop time for the leak.

12.3 Power outage

Power outages can be small and brief, or they can also span over several days and effect whole regions as seen in the 2003 Northeast Blackout. While the Northeast Blackout was an extreme case, a 2012 Lawrence Berkeley National Laboratory study [5] showed the frequency and duration of power outages are increasing by a rate of two percent annually. In water distribution systems, a power outage can cause pump stations to shut down and result in reduced water pressure. This can lead to shortages in some areas of the system. Typically, no lasting damage in the system is associated with power outages.

WNTR can be used to simulate power outages by changing the pump status from ON to OFF and defining the duration of the outage. To model the impact of a power outage on a specific pump:

```
wn.add_pump_outage('335', 5*3600, 10*3600)
```

The method `add_pump_outage` adds time controls to a pump to start and stop a power outage. When simulating power outages, consider placing check bypasses around pumps and check valves next to reservoirs.

12.4 Fires

WNTR can be used to simulate damage caused to system components due to fire and/or to simulate water usage due to fighting fires. To fight fires, additional water is drawn from the system. Fire codes vary by state. Minimum required fire flow and duration are generally based on building area and purpose. While small residential fires might require 1500 gallons/minute for 2 hours, large commercial spaces might require 8000 gallons/minute for 4 hours [8]. This additional demand can have a large impact on water pressure in the system.

WNTR can be used to simulate fire fighting conditions in the system. WNTR simulates fire fighting conditions by specifying the demand, time, and duration of fire fighting. Pressure dependent demand simulation is recommended in cases where fire fighting might impact expected demand. To model the impact of fire conditions at a specific node:

```
fire_flow_demand = 0.252 # 4000 gal/min = 0.252 m3/s
time_of_fire = 10
duration_of_fire = 4
remainder = int(wn.options.duration/3600-time_of_fire-duration_of_fire)
fire_flow_pattern = [0]*time_of_fire + [1]*duration_of_fire + [0]*remainder
wn.add_pattern('fire_flow', fire_flow_pattern)
node = wn.get_node('197')
original_base_demand = node.base_demand
original_demand_pattern_name = node.demand_pattern_name
node.base_demand = original_base_demand+fire_flow_demand
node.demand_pattern_name = 'fire_flow'
```

12.5 Environmental change

Environmental change is a long term problem for water distribution systems. Changes in the environment could lead to reduced water availability, damage from weather incidents, or even damage from subsidence. For example, severe drought in California has forced lawmakers to reduce the state's water usage by 25 percent. Environmental change also leads to sea level rise which can inundate distribution systems. This is especially prevalent in cities built on unstable soils like New Orleans and Washington, DC which are experiencing land subsidence.

WNTR can be used to simulate the effects of environmental change on the water distribution system by changing supply and demand, adding disruptive conditions (i.e., power outages, pipe leaks) caused by severe weather, or by adding pipe leaks caused by subsidence. Power outages and pipe leaks are described above. Changes to supply and demand can be simple (i.e., changing all nodes by a certain percent), or complex (i.e., using external data or correlated statistical methods). To model simple changes in supply and demand:

```
for reservoir_name, reservoir in wn.reservoirs():
    reservoir.base_head = reservoir.base_head*0.9
for junction_name, junction in wn.junctions():
    junction.base_demand = junction.base_demand*1.15
```

12.6 Contamination

Water distribution systems are vulnerable to contamination by a variety of chemical, microbial, or radiological substances. During disasters, contamination can enter the system through reservoirs, tanks, and at other access points within the distribution system. Long term environmental change can lead to degradation of water sources. Contamination can be difficult to detect and is very expensive to clean up. Recent incidents, including the Elk River chemical spill and Flint lead contamination, highlight the need to minimize human health and economic impacts.

WNTR simulates contamination incidents by introducing contaminants into the distribution system and allowing them to propagate through the system. The example **water_quality_simulation.py** includes steps to define and simulate contamination incidents.

Future versions of WNTR will be able to simulate changes in source water quality due to disruptions.

12.7 Other disaster scenarios

Drinking water systems are also susceptible to other natural disasters including floods, droughts, hurricanes, tornadoes, extreme winter storms, and wind events. WNTR can be used to simulate these events by combining the disaster models already described above. For example, tornadoes might cause power outages, pipe breaks, other damage to infrastructure, and fires. Floods might cause power outages, changes to source water (because of treatment failures), and pipe breaks.

13 Resilience metrics

Resilience of water distribution systems refers to the design, maintenance, and operations of that system. All these aspects must work together to limit the effects of disasters and enable rapid return to normal delivery of safe water to customers. Numerous resilience metrics have been suggested [17]. These metrics generally fall into five categories: topographic, hydraulic, water quality, water security, and economic. When quantifying resilience, it is important to understand which metric best defines resilience for a particular scenario. WNTR includes many metrics to help users compare resilience using different methods.

The following sections outline metrics that can be computed using WNTR, including:

- Topographic metrics (Table 9)
- Hydraulic metrics (Table 10)
- Water quality metrics (Table 11)
- Water security metrics (Table 12)
- Economic metrics (Table 13)

While some metrics define resilience as a single system-wide quantity, other metrics define quantities that are a function of time, space, or both. For this reason, state transition plots [3] and network graphics are useful ways to visualize resilience and compare metrics, as shown in Figure 11. In the state transition plot, the x-axis represents time (before, during, and after a disruptive incident). The y-axis represents performance. This can be any time varying resilience metric that responds to the disruptive state. State transition plots are often generated to show time varying performance of the system, but they can also represent the time varying performance of individual components, like tanks or pipes. Network graphics are useful to visualize resilience metrics that vary with respect to location. For metrics that vary with respect to time and space, network animation can be used to illustrate resilience.

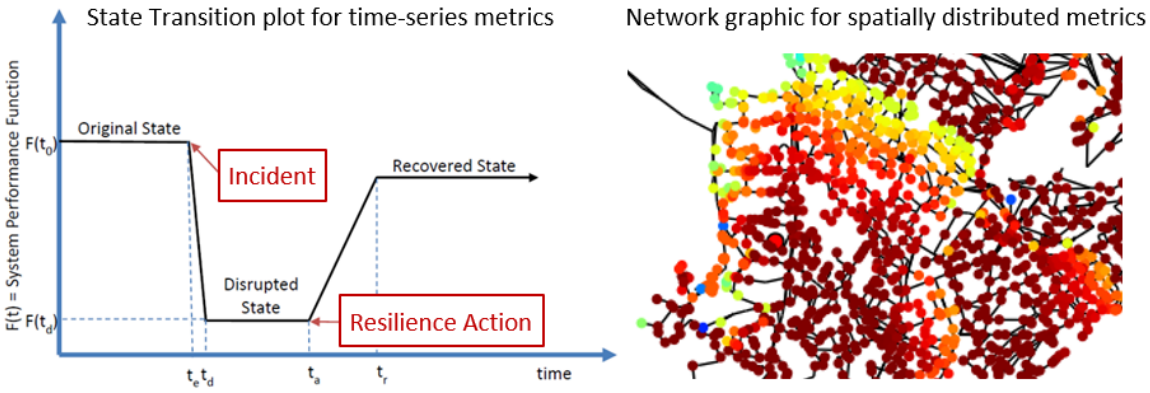

Figure 11: Example state transition plot and network graphic used to visualize resilience.

The example **resilience_metrics.py** demonstrates how to compute these metrics.

13.1 Topographic metrics

Topographic metrics, based on graph theory, can be used to assess the connectivity of water distribution networks. These metrics rely on the physical layout of the network components and can be used to understand how the underlying structure and connectivity constrains resilience. For example, a regular lattice, where each node has the same number of edges, is considered to be the most reliable graph structure. On the other hand, a random lattice has nodes and edges that are placed according to a random process. A real world water distribution system probably lies somewhere in between a regular lattice and a random lattice in terms of structure and reliability.

NetworkX includes a wide range of topographic metrics that can be computed using the WntrMultiDiGraph. WNTR includes additional methods/metrics to help compute resilience. These methods are in the `WntrMultiDiGraph` class. Commonly used topographic metrics are listed in Table 9. Information on additional topographic metrics supported by NetworkX can be found at https://networkx.github.io/.

Table 9: Topographic Resilience Metrics

Metric	Description
Node degree	Node degree is the number of links adjacent to a node. Node degree is a measure of the number of branches in a network. A node with degree 0 is not connected to the network. Terminal nodes have degree 1. A node connected to every node (including itself) has a degree equal to the number of nodes in the network. The average node degree is a system wide metric used to describe the number of connected links in a network. Node degree can be computed using the NetworkX method `degree`. Terminal nodes can be found using the method `terminal_nodes`.
Link density	Link density is the ratio between the total number of links and the maximum number of links in the network. If links are allowed to connect a node to itself, then the maximum number of links is n^2, where n is the number of nodes. Otherwise, the maximum number of nodes is $n(n-1)$. Link density is 0 for a graph without edges and 1 for a dense graph. The density of multigraphs can be higher than 1. Link density can be computed using the NetworkX method `density`.
Eccentricity and diameter	Eccentricity is the maximum number of links between a node and all other nodes in the graph. Eccentricity is a value between 0 and the number of links in the network. Diameter is the maximum eccentricity in the network. Eccentricity and diameter can only be computed using undirected, connected networks. Network X includes a method to convert directed graphs to undirected graphs, `to_undirected`, and to check if graphs are connected, `is_connected`. Eccentricity and diameter can be computed using the NetworkX methods `eccentricity` and `diameter`.
Simple paths	A simple path is a path between two nodes that does not repeat any nodes. NetworkX includes a method, `all_simple_paths`, to compute all simple paths between two nodes. The method `links_in_simple_paths` can be used to extract all links in a simple path along with the number of times each link was used in the paths. Paths can be time dependent, if related to simulated flow direction. The method `weight_graph` can be used to weight the graph by a specified attribute.
Shortest path lengths	Shortest path lengths is the minimum number of links between a source node and all other nodes in the network. Shortest path length is a value between 0 and the number of links in the network. The average shortest path length is a system wide metric used to describe the number of links between a node and all other nodes. Shortest path lengths and average shortest path lengths can be computed using the following NetworkX methods `shortest_path_length` and `average_shortest_path_length`.
Betweenness centrality	Betweenness centrality is the fraction of shortest paths that pass through each node. Betweenness coefficient is a value between 0 and 1. Central point dominance is the average difference in betweenness centrality of the most central point (having the maximum betweenness centrality) and all other nodes. These metrics can be computed using the NetworkX methods `betweenness_centrality` and the method `central_point_dominance`
Closeness centrality	Closeness centrality is the inverse of the sum of shortest path from one node to all other nodes. Closeness centrality can be computed using the NetworkX method `closeness_centrality`.
Articulation points	A node is considered an articulation point if the removal of that node (along with all its incident edges) increases the number of connected components of a network. Density of articulation points is the ratio of the number of articulation points and the total number of nodes. Density of articulation points is a value between 0 and 1. Articulation points can be computed using the NetworkX method `articulation_points`.
Bridges	A link is considered a bridge if the removal of that link increases the number of connected components in the network. The ratio of the number of bridges and the total number of links in the network is the bridge density. Bridge density is a value between 0 and 1. The method `bridges` can be used to find bridges in a network.

13.2 Hydraulic metrics

Hydraulic metrics are based upon variable flows and/or pressure. The calculation of these metrics requires simulation of network hydraulics that reflect how the system operates under normal or abnormal conditions. Hydraulic metrics included in WNTR are listed in Table 10.

Table 10: Hydraulic Resilience Metrics

Metric	Description
Pressure	To determine the number of node-time pairs above or below a specified pressure threshold, use the `query` method on results.node['pressure'].
Todini index	The Todini index [16] is related to the capability of a system to overcome failures while still meeting demands and pressures at the nodes. The Todini index defines resilience at a specific time as a measure of surplus power at each node and measures relative energy redundancy. The Todini index can be computed using the `todini` method.
Entropy	Entropy [2] is a measure of uncertainty in a random variable. In a water distribution network model, the random variable is flow in the pipes and entropy can be used to measure alternate flow paths when a network component fails. A network that carries maximum entropy flow is considered reliable with multiple alternate paths. Connectivity will change at each time step, depending on the flow direction. The method `weight_graph` method can be used to weight the graph by a specified attribute. Entropy can be computed using the `entropy` method.
Fraction of delivered volume	Fraction of delivered volume is the ratio of total volume delivered to the total volume requested [12]. This metric can be computed as a function of time or space using the `fdv` method.
Fraction of delivered demand	Fraction of delivered demand is the fraction of time periods where demand is met [12]. This metric can be computed as a function of time or space using the `fdd` method.
Population impacted	Population that is impacted by a specific quantity can be computed using the `population_impacted` method. For example, this method can be used to compute the population impacted by pressure below a specified threshold.

13.3 Water quality metrics

Water quality metrics are based on the concentration or water age. The calculation of these metrics require a water quality simulation. Water quality metrics included in WNTR are listed in Table 11.

Table 11: Water Quality Resilience Metrics

Metric	Description
Water age	To determine the number of node-time pairs above or below a specified water age threshold, use the `query` method on results.node['quality'] after a simulation using AGE.
Concentration	To determine the number of node-time pairs above or below a specified concentration threshold, use the `query` method on results.node['quality'] after a simulation using CHEM or TRACE.
Fraction of delivered quality	Fraction of delivered quality is the fraction of time periods where water quality standards are met [12]. This metric can be computed as a function of time or space using the `fdq` method
Average water consumed	Average water consumed is computed at each node, based on node demand and demand patterns [18]. The metric can be computed using the `average_water_consumed` method.
Population impacted	As stated above, population that is impacted by a specific quantity can be computed using the `population_impacted` method. This can be applied to water quality metrics.

13.4 Water security metrics

Water security metrics quantify potential consequences of contamination scenarios. These metrics are documented in [18]. Water security metrics included in WNTR are listed in Table 12.

Table 12: Water Security Resilience Metrics

Metric	Description
Mass consumed	Mass consumed is the mass of a contaminant that exits the network via node demand at each node-time pair [18]. The metric can be computed using the `mass_contaminant_consumed` method.
Volume consumed	Volume consumed is the volume of a contaminant that exits the network via node demand at each node-time pair [18]. A detection limit can be specified. The metric can be computed using the `volume_contaminant_consumed` method.
Extent of contamination	Extent of contamination is the length of contaminated pipe at each node-time pair [18]. A detection limit can be specified. The metric can be computed using the `extent_contaminant` method.
Population impacted	As stated above, population that is impacted by a specific quantity can be computed using the `population_impacted` method. This can be applied to water security metrics.

13.5 Economic metrics

Economic metrics include network cost and greenhouse gas emissions. Economic metrics included in WNTR are listed in Table 13.

Table 13: Economic Resilience Metrics

Metric	Description
Network cost	Network cost is the annual maintenance and operations cost of tanks, pipes, vales, and pumps based on the equations from the Battle of Water Networks II [14]. Default values can be included in the calculation. Network cost can be computed using the `cost` method.
Greenhouse gas emissions	Greenhouse gas emissions is the annual emissions associated with pipes based on equations from the Battle of Water Networks II [14]. Default values can be included in the calculation. Greenhouse gas emissions can be computed using the `ghg_emissions` method.
Pump operating energy and cost	The energy and cost required to operate a pump can be computed using the `pump_energy` method. This uses the flowrates and pressures from simulation results to compute pump energy and cost.

14 Stochastic simulation

Stochastic simulations can be used to evaluate an ensemble of hydraulic and/or water quality scenarios. For disaster scenarios, the location, duration, and severity of different types of incidents can be drawn from distributions and included in the simulation. Distributions can be a function of component properties (i.e., age, material) or based on engineering standards. The Python packages Numpy and Scipy include statistical distributions and random selection methods that can be used for stochastic simulations.

For example, the following code can be used to select N unique pipes based on the failure probability of each pipe:

```
N = 2
failure_probability = {'pipe1': 0.10, 'pipe2': 0.20, 'pipe3': 0.25, 'pipe4': 0.15,
        'pipe5': 0.30}
pipes_to_fail = np.random.choice(failure_probability.keys(), N, replace=False,
        p=failure_probability.values())
```

The example **stochastic_simulation.py** runs multiple realizations of a pipe leak scenario where the location and duration are drawn from probability distributions.

14.1 Fragility curves

Fragility curves are commonly used in disaster models to define the probability of exceeding a given damage state as a function environmental change. Fragility curves are closely related to survival curves, which are used to define the probability of component failure due to age. To estimate earthquake damage, fragility curves are defined as a function of peak ground acceleration, peak ground velocity, or repair rate. The American Lifelines Alliance report [1] includes seismic fragility curves for water network components. Fragility curves can also be defined as a function of flood stage, wind speed, and temperature for other types of disasters.

Fragility curves can have multiple damage states. Each state should correspond to specific changes to the network model that represent damage, for example, a major or minor leak. Each state is defined with a name (i.e., 'Major,' 'Minor'), priority (i.e., 1, 2, where higher numbers = higher priority), and distribution (using the Scipy Python package). The distribution can be defined for all elements using the keyword 'Default,' or can be defined for individual components. Each fragility curve includes a 'No damage' state with priority 0 (lowest priority).

The example **fragility_curves.py** uses fragility curves to determine probability of failure:

```
from scipy.stats import lognorm
FC = wntr.scenario.FragilityCurve()
FC.add_state('Minor', 1, {'Default': lognorm(0.5,scale=0.3)})
FC.add_state('Major', 2, {'Default': lognorm(0.5,scale=0.7)})
```

Figure 12 illustrates a fragility curve based on peak ground acceleration with two damage states: Minor damage and Major damage. For example, if the peak ground acceleration is 0.5 at a specific junction, the probability of exceeding a Major damage state 0.25 and the probability of exceeding the Minor damage state is 0.85. For each stochastic simulation, a random number is drawn between 0 and 1. If the random number is between 0 and 0.25, the junction is assigned Minor damage. If the random number is between 0.25 and 0.85, then the junction is assigned Major damage. If the random number is between 0.85 and 1, then the junction is assigned No damage. After selecting a damage state for the junction, the network should be changed to reflect the associated damage. For example, if the junction has Major damage, a large leak might be defined at that location.

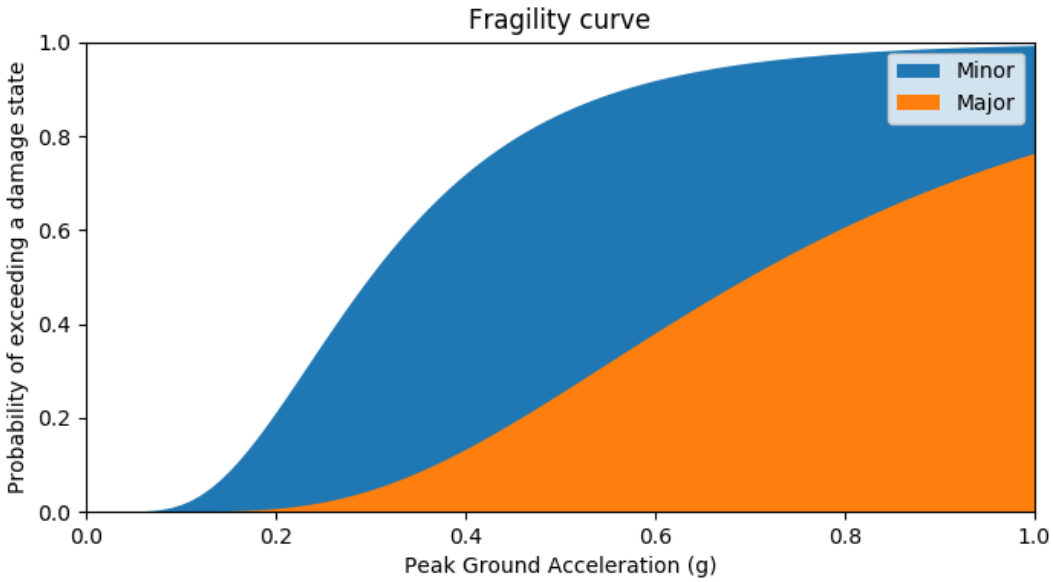

Figure 12: Example fragility curve.

15 Copyright and license

The WNTR Python package is copyright through Sandia National Laboratories. The software is distributed under the Revised BSD License. WNTR also leverages a variety of third-party software packages, which have separate licensing policies.

15.1 Copyright

15.2 Revised BSD license

16 Software quality assurance

The following section includes information about the WNTR software repository, software tests, documentation, examples, bug reports, feature requests, and ways to contribute.

16.1 GitHub repository

WNTR is maintained in a version controlled repository. WNTR is hosted on US EPA GitHub organization at https://github.com/USEPA/WNTR.

16.2 Software tests

WNTR includes continuous integration software tests that are run using Travis CI. The tests are run each time changes are made to the repository. The tests cover a wide range of unit and integration tests designed to ensure that the code is performing as expected. New tests are developed each time new functionality is added to the code. Testing status (passing/failed) and code coverage statistics are posted on the README section at https://github.com/USEPA/WNTR.

Tests can also be run locally using the Python package nose. For more information on nose, see http://nose.readthedocs.io/. nose comes with a command line software tool called nosetests. Tests can be run in the WNTR directory using the following command:

```
nosetests -v --with-coverage --cover-package=wntr wntr
```

16.3 Documentation

WNTR includes a user manual that is built using the Read the Docs service. The user manual is automatically rebuilt each time changes are made to the code. The documentation is publicly available at http://wntr.readthedocs.io/. The user manual includes an overview, installation instructions, simple examples, and information on the code structure and functions. WNTR includes documentation on the API for all public functions, methods, and classes. New content is marked *Draft*.

16.4 Examples

WNTR includes examples to help new users get started. These examples are intended to demonstrate high level features and use cases for WNTR. The examples are tested to ensure they stay current with the software project.

16.5 Bug reports and feature requests

Bug reports and feature requests can be submitted to https://github.com/USEPA/WNTR/issues. The core development team will prioritize and assign bug reports and feature requests to team members.

16.6 Contributing

Software developers, within the core development team and external collaborators, are expected to follow standard practices to document and test new code. Software developers interested in contributing to the project are encouraged to create a *Fork* of the project and submit a *Pull Request* using GitHub. Pull requests will be reviewed by the core development team.

Pull requests must meet the following minimum requirements to be included in WNTR:

- Code is expected to be documented using Read the Docs.
- Code is expected to be sufficiently tested using Travis CI. *Sufficient* is judged by the strength of the test and code coverage. 80% code coverage is recommended.

- Large files (> 1Mb) will not be committed to the repository without prior approval.
- Network model files will not be duplicated in the repository. Network files are stored in examples/network and wntr/tests/networks_for_testing only.

16.7 Development team

WNTR was developed as part of a collaboration between the United States Environmental Protection Agency National Homeland Security Research Center, Sandia National Laboratories, and Purdue University. See https://github.com/USEPA/WNTR/graphs/contributors for a list of contributors.

17 References

[1] American Lifelines Alliance. (2001). Seismic Fragility Formulations for Water Systems, Part 1 and 2. Report for the American Lifelines Alliance, ASCE (Ed.) Reston, VA: American Society of Civil Engineers. April 2001.

[2] Awumah, K., Goulter, I., and Bhatt, S.K. (1990). Assessment of reliability in water distribution networks using entropy based measures. Stochastic Hydrology and Hydraulics, 4(4), 309-320.

[3] Barker, K., Ramirez-Marquez, J.E., and Rocco, C.M. (2013). Resilience-based network component importance measures. Reliability Engineering and System Safety, 117, 89-97.

[4] Crowl, D.A., and Louvar, J.F. (2002). Chemical Process Safety: Fundamentals with Applications, 3 edition. Upper Saddle River, NJ: Prentice Hall, 720p.

[5] Eto, J.H., LaCommare, K.H., Larsen, P.H., Todd, A., and Fisher, E. (2012). An Examination of Temporal Trends in Electricity Reliability Based on Reports from U.S. Electric Utilities. Lawrence Berkeley National Laboratory Report Number 5268E. Berkeley, CA: Ernest Orlando Lawrence Berkeley National Laboratory, 68p.

[6] Hagberg, A.A., Schult, D.A., and Swart P.J. (2008). Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy2008), August 19-24, Pasadena, CA, USA.

[7] Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. Computing in Science and Engineering, 9(3), 90-95.

[8] International Code Council. (2011). 2012 International Fire Code, Appendix B - Fire-Flow Requirements for Buildings. Country Club Hills, IL: International Code Council, ISBN: 978-1-60983-046-5.

[9] Joyner, D., Certik, O., Meurer, A., and Granger, B.E. (2011). Open source computer algebra systems, SymPy. ACM Communications in Computer Algebra, 45(4), 225-234.

[10] McKinney W. (2013). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. Sebastopal, CA: O'Reilly Media, 1 edition, 466p.

[11] National Infrastructure Advisory Council (NIAC). (2009). Critical Infrastructure Resilience, Final Report and Recommendations, U.S. Department of Homeland Security, Washington, D.C., Accessed September 20, 2014. http://www.dhs.gov/xlibrary/assets/niac/niac_critical_infrastructure_resilience.pdf.

[12] Ostfeld, A., Kogan, D., and Shamir, U. (2002). Reliability simulation of water distribution systems - single and multiquality. Urban Water, 4(1), 53-61.

[13] Rossman, L.A. (2000). EPANET 2 Users Manual. Cincinnati, OH: U.S. Environmental Protection Agency. U.S. Environmental Protection Agency Technical Report, EPA/600/R–00/057, 200p.

[14] Salomons, E., Ostfeld, A., Kapelan, Z., Zecchin, A., Marchi, A., and Simpson, A. (2012). The battle of the water networks II - Problem description. Water Distribution Systems Analysis Conference 2012, September 24-27, Adelaide, South Australia, Australia. Retrieved on May 23, 2017 from https://emps.exeter.ac.uk/media/universityofexeter/emps/research/cws/downloads/WDSA2012-BWNII-ProblemDescription.pdf.

[15] Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., and Despouy, P. (2016). plotly: Create interactive web graphics via Plotly's JavaScript graphing library [Software].

[16] Todini, E. (2000). Looped water distribution networks design using a resilience index based heuristic approach. Urban Water, 2(2), 115-122.

[17] United States Environmental Protection Agency. (2014). Systems Measures of Water Distribution System Resilience. Washington DC: U.S. Environmental Protection Agency. U.S. Environmental Protection Agency Technical Report, EPA 600/R–14/383, 58p.

[18] United States Environmental Protection Agency. (2015). Water Security Toolkit User Manual. Washington DC: U.S. Environmental Protection Agency. U.S. Environmental Protection Agency Technical Report, EPA/600/R-14/338, 187p.

[19] van der Walt, S., Colbert, S.C., and Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. Computing in Science and Engineering, 13, 22-30.

[20] Wagner, J.M., Shamir, U., and Marks, D.H. (1998). Water distribution reliability: Simulation methods. Journal of Water Resources Planning and Management, 114(3), 276-294.

[21] Walk, D.J., Worden, B.C., Quitoriano, V., and Pankow, K.L. (2006). Shakemap manual, Technical manual, users guide, and software guide. United States Geologic Survey, Retrieved on April 25, 2017 from http://pubs.usgs.gov/tm/2005/12A01/

[22] xlwt contributors. (2016, November 18). xlwt documentation. Retrieved on April 25, 2017 from https://xlwt.readthedocs.io.