TSNet Documentation

Release 0.1.0

Lu Xing, Lina Sela

Aug 16, 2019

Contents:

CHAPTER 1

Introduction to TSNet

TSNet performs transient simulation in water networks using Method of Characteristics (MOC).

- Free software: MIT license
- Github: https://github.com/glorialulu/TSNet.git
- Documentation: https://tsnet.readthedocs.io.

1.1 Overview

Hydraulic transients in water distribution networks (WDNs), typically induced by pipe bursts, valve operations, and pump operations, can disturb the steady-state flow conditions by introducing extreme pressure variability and imposing abrupt internal pressure force onto the pipeline systems [WOLB05]. These disturbances have been identified as one of the major contributing factors in the many pipe deterioration and catastrophic failure in WDNs [RERS15], thereby wasting a significant amount of treated water and creating unexpected possibilities of contamination intrusion [ASCE17]. Consequently, transient simulation, as a prominent approach to understand and predict the behavior of hydraulic transients, has become an essential requirement for ensuring the distribution safety and improving the efficiency in the process of design and operation of WDNs. In addition to improving design and operation of WDNs, various other transient-based applications, such as network calibration, leak detection, sensor placement, and condition assessment, has also enhanced the popularity and necessity of transient simulation

Acknowledgedly, a number of commercial software for transient simulation in water distribution systems is available in the market; however, the use of these software for research purposes is limited. The major restriction is due to the fact that the programs are packed as black boxes, and the source codes are not visible, thus prohibiting any changes, including modification of existing and implementation of new elements, in the source codes. Additionally, the commercial software is designed to perform only single transient simulations and do not have the capabilities to automate or run multiple transient simulations. Users are required to modify the boundary conditions using the GUI, perform the simulation, and manually record the hydraulic responses in the various conditions, which significantly complicated the research process.

There is a clear gap that currently available simulation software are not suitable for many research applications beyond the conventional design purposes. Hence, the motivation of this work is two-fold:

1. Provide users with open source and freely available python code and package for simulating transients in water distribution systems that can be integrated with other case specific applications, e.g. sensor placement and event detection; and

2. Encourage users and developers to further develop and extend the transient model.

1.2 Features

TSNet is a Python package designed to perform transient simulation in water distribution networks. The software includes capability to:

- Create transient models based on EPANET INP files
- Operating valves and pumps
- Add disruptive events including pipe bursts and leakages
- Perform transient simulation using Method of characteristics (MOC) techniques
- Visualize results

For more information, go to https://tsnet.readthedocs.io.

1.3 Version

TSNet is a ongoing research project in the University of Texas at Austin. The current version is 0.1.0, which is still a pre-release.

1.4 Contact

- Lu Xing, the University of Texas at Austin, xinglu@utexas.edu
- Lina Sela, the University of Texas at Austin, linasela@utexas.edu

1.5 Disclaimer

No warranty, expressed or implied, is made as to the correctness of the results or the suitability of the application.

1.6 Cite TSNet

To cite TSNet, use one of the following references:

1.7 License

CHAPTER 2

Installation

2.1 Setup Python Environment

TSNet is tested against Python versions 2.7, 3.5 and 3.6. It can be installed on Windows, Linux, and Mac OS X operating systems. Python distributions, such as Anaconda, are recommended to manage the Python environment as they already contain (or easily support installation of) many Python packages (e.g. SciPy, NumPy, pandas, pip, matplotlib, etc.) that are used in the TSNet package. For more information on Python package dependencies, see *Dependencies*.

2.2 Stable Release (for users)

To install TSNet, run this command in your terminal:

```
$ pip install tsnet
```

This is the preferred method to install tsnet, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.3 From Sources (for developers)

The sources for TSNet can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/glorialulu/tsnet
```

Or download the tarball:

```
$ curl  -OL https://github.com/glorialulu/tsnet/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

2.4 Dependencies

Requirements for TSNet include Python (2.7, 3.5, or 3.6) along with several Python packages. The following Python packages are required:

1. Numpy [VaCV11]: the fundamental package needed for scientific computing with Python included in Anaconda distribution http://www.numpy.org/
2. Matplotlib [Hunt07]: Python 2D plotting library included in Anaconda distribution http://matplotlib.org/
3. NetworkX [HaSS08]: Network creation and manipulation engine, install on a python-enabled command line with *pip install wntr* https://networkx.github.io/
4. WNTR [WNTRSi]: Water Network Tool for Resilience install on a python-enabled command line with *pip install wntr* http://wntr.readthedocs.io
5. pytest: Unit Tests engine install on a python-enabled command line with *pip install -U pytest* https://docs.pytest.org/en/latest/

CHAPTER 3

Software Conventions and Limitations

3.1 Units

All data in TSNet is stored in the following International System (SI) units:

- Length = m
- Diameter = m
- Water pressure = m (this assumes a fluid density of 1000 kg/m^3)
- Elevation = m
- Mass = kg
- Time = s
- Demand = m^3/s
- Velocity = m/s
- Acceleration = g (1 g = 9.8 m/s^2)
- Volume = m^3

If the unit system specified in .inp file is US units, it will be converted to SI unit in the simulation process. When setting up analysis in TSNet, all input values should be specified in SI units. All simulation results are also stored in SI units.

3.2 Modelling Assumptions and Limitations

TSNet is constantly under development. Current software limitations are as follows:

- Demands on the start and end nodes of pumps and valves are not supported. If demands are defined on these nodes in the .inp file, they will be ignored in transient simulation, and the simulation results may not be accurate due to discrepancies between the initial conditions and the first step in transient simulation. Warnings will be printed.

- Multi-branch junctions on the start and end nodes of pumps and valves are not supported. It is assumed that valves and pumps are connected by pipes in series.
- During transient simulation, demands are pressure dependent .
- Pipe Friction coefficients are converted to Darcy-Weisbach coefficients based on initial conditions.
- Pipe bursts and leaks occur only on the nodes.
- Transient simulation relies on a feasible steady state solution; hence, it is essential to verify that the steady state simulation succeeds without errors.

CHAPTER 4

Getting Started

To use tsnet in a project, open a Python console and import the package:

```
import tsnet
```

4.1 Simple example

A simple example, Tnet1_valve_closure.py is included in the examples folder. This example demonstrates how to:

- Import tsnet
- Generate a transient model
- Set wave speed
- Set time step and simulation period
- Perform initial condition calculation
- Define valve closure rule
- Run transient simulation and save results to .obj file
- Plot simulation results

```
# Open an example network and create a transient model
inp_file = 'examples/networks/Tnet1.inp'
tm = tsnet.network.TransientModel(inp_file)

# Set wavespeed
tm.set_wavespeed(1200.) # m/s
# Set time options
dt = 0.1  # time step [s], if not given, use the maximum allowed dt
tf = 60   # simulation period [s]
tm.set_time(tf,dt)
```

(continues on next page)

(continued from previous page)

```
# Set valve closure
tc = 0.6 # valve closure period [s]
ts = 0 # valve closure start time [s]
se = 0 # end open percentage [s]
m = 1 # closure constant [dimensionless]
valve_op = [tc,ts,se,m]
tm.valve_closure('VALVE',valve_op)

# Initialize steady state simulation
t0 = 0. # initialize the simulation at 0 [s]
engine = 'DD' # demand driven simulator
tm = tsnet.simulation.Initializer(tm, t0, engine)

# Transient simulation
results_obj = 'Tnet1' # name of the object for saving simulation results
tm = tsnet.simulation.MOCSimulator(tm, results_obj)

# report results
import matplotlib.pyplot as plt
node = 'N3'
node = tm.get_node(node)
fig1 = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.head)
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Pressure Head at Node %s '%node)
plt.xlabel("Time [s]")
plt.ylabel("Pressure Head [m]")
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

Three additional EPANET INP files and example files are also included in the TSNet examples repository in the examples folder. Example networks range from a simple 8-node network to a 126-node network.

CHAPTER 5

Transient Modeling Framework

The framework of performing transient simulation using TSNet is shown in Figure 5.1

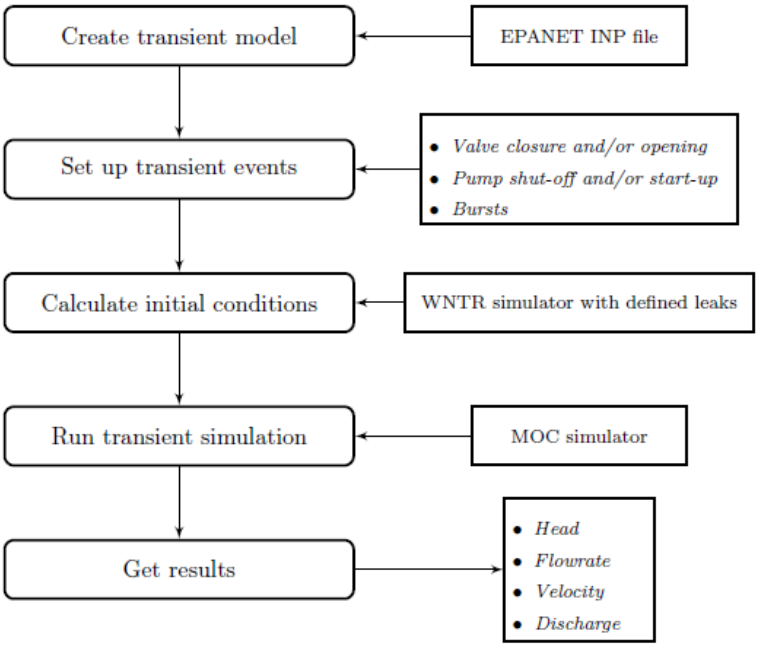

Figure 5.1: Flowchart of transient simulation in TSNet

The main steps of transient modelling and simulation in TSNet are described in subsequent sections.

5.1 Transient Model

The transient model inherits the WNTR water network model [WNTRSi], which includes junctions, tanks, reservoirs, pipes, pumps, valves, patterns, curves, controls, sources, simulation options, and node coordinates. It can be built directly from an EPANet INP file. Sections of EPANet INP file that are not compatible with WNTR are described in [WNTRSi].

Compared with WNTR water network model, TSNet transient model adds the features designed specifically for transient simulation, such as spatial discretization, temporal discretization, valve operation rules, pump operation rules, burst opening rules, and storage of time history results. For more information on the water network model, see *TransientModel* in the API documentation.

A transient model can be created directly from an EPANET INP file. The following example build a transient model.

```
inp_file = 'examples/networks/Tnet1.inp'
tm = tsnet.network.TransientModel(inp_file)
```

5.2 Initial Conditions

TSNet employed WNTR [WNTRSi] for simulating the steady state in the network to establish the initial conditions for the upcoming transient simulations.

WNTRSimulators can be used to run demand-driven (DD) or pressure-dependent demand (PDD) hydraulics simulations, with the capacity of simulating leaks. The default simulation engine is DD. An initial condition simulation can be run using the following code:

```
t0 = 0. # initialize the simulation at 0 [s]
engine = 'DD' # demand driven simulator
tm = tsnet.simulation.Initializer(tm, t0, engine)
```

t_0 stands for the time when the initial condition will be calculated. More information on the initializer can be found in the API documentation, under *Initializer*.

5.3 Transient Simulation

After the initial conditions are obtained, TSNet adopts the Method of Characteristics (MOC) for solving governing transient flow equations. A transient simulation can be run using the following code:

```
results_obj = 'Tnet1' # name of the object for saving simulation results
tm = tsnet.simulation.MOCSimulator(tm, results_obj)
```

The results will be returned to the transient model (tm) object, and then stored in the 'Tnet1.obj' file for the easiness of retrieval.

In the following sections, an overview of the solution approaches and boundary conditions is presented, based on the following literature [LAJW99] , [MISI08], [WYSS93].

5.3.1 Governing Equations and Numerical Schemes

Mass and Momentum Conservation

The transient flow is governed by the mass and momentum conservation equation [WYSS93]:

$$\frac{\partial H}{\partial t} + \frac{a^2}{g}\frac{\partial V}{\partial x} - gV\sin\alpha = 0$$
$$\frac{\partial V}{\partial t} + g\frac{\partial H}{\partial x} + h_f = 0$$

where H is the head, V is the flow velocity in the pipe, t is time, a is the wave speed, g is the gravity acceleration, α is the angle from horizontal, and h_f represents the head loss (only quasi-steady friction head loss per unit length is modelled in current package).

Method of Characteristics (MOC)

The Method of Characteristics (MOC) method is used to solve the system of governing equations above. The essence of MOC is to transform the set of partial differential equations to an equivalent set of ordinary differential equations that apply along specific lines, i.e., characteristics lines (C+ and C-), as shown below [LAJW99]:

$$C+ : \frac{dV}{dt} + \frac{g}{a}\frac{dH}{dt} + h_f - gV\sin(\alpha) = 0 \text{ only when } \frac{dx}{dt} = a$$
$$C- : \frac{dV}{dt} - \frac{g}{a}\frac{dH}{dt} + h_f - gV\sin(\alpha) = 0 \text{ only when } \frac{dx}{dt} = -a$$

The explicit MOC technique is then adopted to solve the above systems of equations along the characteristics lines [LAJW99].

Headloss in Pipes

TSNet adopts Darcy-Weisbach equation to compute head loss, regardless of the friction method defined in the EPANET .inp file. This package computes Darcy-Weisbach coefficients (f) based on the head loss (h_{f_0}) and flow velocity (V_0) in initial condition, using the following equation:

$$f = \frac{h_{f_0}}{(L/D)(V_0^2/2g)}$$

where L is the pipe length, D is the pipe diameter, and g is gravity acceleration.

Subsequently, in transient simulation the headloss (h_f) is calculated based on the following equation:

$$h_f = f\frac{L}{D}\frac{V^2}{2g}$$

Pressure-driven Demand

During the transient simulation in TSNet, the demands are treated as pressure-dependent discharge; thus, the actual demands will vary from the demands defined in the INP file. The actual demands (d_{actual}) are modeled based on the instantaneous pressure head at the node and the demand discharge coefficients, using the following equation:

$$d_{actual} = k\sqrt{H_p}$$

where H_p is the pressure head and k is the demand discharge coefficient, which is calculated from the initial demand (d_0) and pressure head (H_{p0}):

$$k = \frac{d_0}{\sqrt{H_{p0}}}$$

It should be noted that if the pressure head is negative, the demand flow will be treated zero, assuming that a backflow preventer is installed on each node.

5.3.2 Choice of Time Step

The determination of time step in MOC is not a trivial task. There are two constraints that have to be satisfied simultaneously:

1. The Courant's criterion has to be satisfied for each pipe, indicating the maximum time step allowed in the network transient analysis will be:

$$\Delta t \leqslant \min\left(\frac{L_i}{N_i a_i}\right), i = 1, 2, ..., n_p$$

2. The time step has to be the same for all the pipes in the network, therefore restricting the wave travel time $\frac{L_i}{N_i a_i}$ to be the same for any computational unit in the network. Nevertheless, this is not realistic in a real network, because different pipe lengths and wave speeds usually cause different wave travel times. Moreover, the number of sections in the i^{th} pipe (N_i) has to be an even integer due to the grid configuration in MOC; however, the combination of time step and pipe length is likely to produce non-integer value of N_i, which then requires further adjustment.

This package adopted the wave speed adjustment scheme [WYSS93] to make sure the two criterion stated above are satisfied.

To begin with, the maximum allowed time step (Δt_{max}) is calculated, assuming there are two computational segments on the shortest pipe:

$$\Delta t_{max} = \min\left(\frac{L_i}{2a_i}\right), i = 1, 2, ..., n_p$$

If the user defined time step is greater than Δt_{max}, a fatal error will be raised and the program will be killed; if not, the user defined value will be used as the initial guess for the upcoming adjustment.

```
dt = 0.1  # time step [s], if not given, use the maximum allowed dt
tf = 60   # simulation period [s]
tm.set_time(tf,dt)
```

The determination of time step is not straightforward, especially in large networks. Thus, we allow the user to ignore the time step setting, in which case Δt_{max} will be used as the initial guess for the upcoming adjustment.

After setting the initial time step, the following adjustments will be performed. Firstly, the i^{th} pipes (p_i) with length (L_i) and wave speed (a_i) will be discretized into (N_i) segments:

It should be noted that even if the user defined time step satisfied the Courant's criterion, it will still be adjusted.

5.3.3 Boundary Conditions

Valve Operations

Valve operations, including closure and opening, are supported in TSNet. The default valve shape is gate valve, the valve characteristics curve of which is defined according to [STWV96]. The following examples illustrate how to perform valve operations.

Valve closure can be simulated by defining the valve closure start time (ts), the operating duration (t_c), the valve open percentage when the closure is completed (se), and the closure constant (m), which characterizes the shape of the closure curve. These parameters essentially define the valve closure curve. For example, the code below will yield the blue curve shown in Figure 5.2. If the closure constant (m) is instead set to 2, the valve curve will then correspond to the orange curve in Figure 5.2.

```
tc = 1 # valve closure period [s]
ts = 0 # valve closure start time [s]
se = 0 # end open percentage [s]
m = 1 # closure constant [dimensionless]
valve_op = [tc,ts,se,m]
tm.valve_closure('VALVE',valve_op)
```

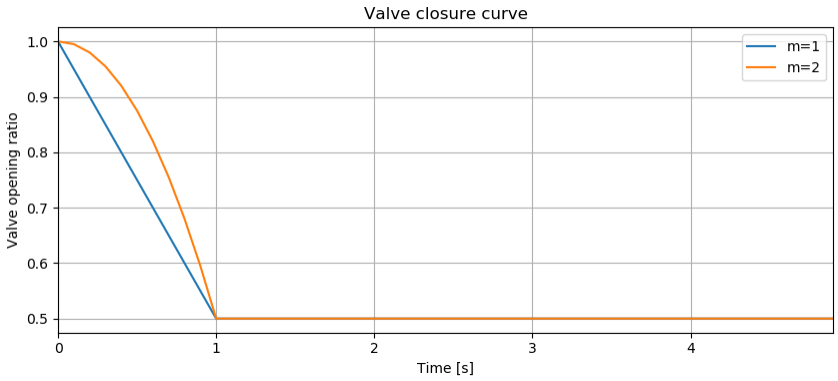

Figure 5.2: Valve closure operating rule

Furthermore, valve opening can be simulated by defining a similar set of parameters related to the valve opening curve. The valve opening curves with $m = 1$ and $m = 2$ are illustrated in Figure 5.3.

```
tc = 1 # valve opening period [s]
ts = 0 # valve opening start time [s]
se = 1 # end open percentage [s]
m = 1 # opening constant [dimensionless]
valve_op = [tc,ts,se,m]
tm.valve_opening('VALVE',valve_op)
```

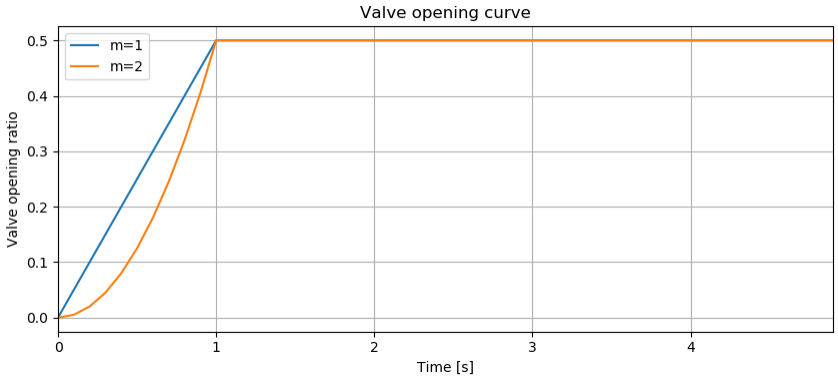

Figure 5.3: Valve opening operating rule

Pump Operations

The TSNet also includes the capability to perform controlled pump operations by specifying how the pump rotation speed changes over time. Explicitly, during pump start-up, the rotational speed of the pump is increased based on the user defined operating rule. The pump is then modeled using the two compatibility equations, a continuity equation,

the pump characteristic curve at given rotation speed, and the affinity laws [LAJW99], thus resulting in the rise of pump flowrate and the addition of mechanical energy. Conversely, during pump shut-off, as the rotational speed of the pump decreased according to the user defined operating rule, the pump flowrate and the addition of mechanical energy decline. However, pump shut-off due to power failure, when the reduction of pump rotation speed depends on the characteristics of the pump (such as the rotate moment of inertia), has not been included yet.

The following example shows how to add pump shut-off event to the network, where the parameters are defined in the same manner as in valve closure:

```
tc = 1 # pump closure period
ts = 0 # pump closure start time
se = 0 # end open percentage
m = 1 # closure constant
pump_op = [tc,ts,se,m]
tm.pump_shut_off('PUMP2', pump_op)
```

Correspondingly, the controlled pump opening can be simulated using:

```
tc = 1 # pump opening period [s]
ts = 0 # pump opening start time [s]
se = 1 # end open percentage [s]
m = 1 # opening constant [dimensionless]
pump_op = [tc,ts,se,m]
tm.pump_start_up('PUMP2',pump_op)
```

It should be noted that a check valve is assumed in each pump, indicating that the reverse flow will be prevented immediately.

Leaks

In TSNet, leaks and bursts are assigned to the network nodes. A leak is defined by specifying the leaking node name and the emitter coefficient (k_l):

```
emitter_coeff = 0.01 # [ m^3/s/(m H20)^(1/2)]
tm.add_leak('JUNCTION-22', emitter_coeff)
```

Existing leaks should be included in the initial condition solver (WNTR simulator); thus, it is necessary to define the leaks before calculating the initial conditions. For more information about the inclusion of leaks in steady state calculation, please refer to WNTR documentation [WNTRSi]. During the transient simulation, the leaking node is modeled using the two compatibility equations, a continuity equation, and an orifice equation which quantifies the leak discharge (Q_l):

$$Q_l = k_l \sqrt{H_{p_l}}$$

where H_{p_l} is the pressure head at the leaking node. Moreover, if the pressure head is negative, the leak discharge will be set to zero, assuming a backflow preventer is installed on the leaking node.

Bursts

The simulation of burst and leaks is very similar. They share similar set of governing equations. The only difference is that the burst opening is simulated only during the transient calculation and not included in the initial condition calculation. In other words, using burst, the user can model new and evolving condition, while the leak model simulates an existing leak in the system. In TSNet, the burst is assumed to be developed linearly, indicating that the burst area increases linearly from zero to a size specified by the user during the specified time period. Thus, a burst event can be modeled by defining the start and end time of the burst, and the final emitter coefficient when the burst is fully developed:

```
ts = 1 # burst start time
tc = 1 # time for burst to fully develop
final_burst_coeff = 0.01 # final burst coeff [ m^3/s/(m H20)^(1/2)]
tm.add_burst('JUNCTION-20', ts, tc, final_burst_coeff)
```

CHAPTER 6

Simulation Results

6.1 Results Structure

Simulation results are returned and saved in the *tsnet.network.model.TransientModel* object for each node and link in the networks.

Node results include the following attributes:

- Head [m]
- Emitter discharge (including leaks and bursts) [m^3/s]
- Actual demand discharge [m^3/s]

Link results include the following attributes:

- Head at start node [m]
- Flow velocity at start node [m^3/s]
- Flow rate at start node [m^3/s]
- Head at end node [m]
- Flow velocity at end node [m^3/s]
- Flow rate at end node [m^3/s]

The result for each attribute is a Numpy array, representing the time history of the simulation results, the length of which equals the total number of simulation time steps (tn).

For example, the results of head, emitter discharge and demand discharge at node 'JUNCTION-105' can be accessed by:

```
node = tm.get_node['JUNCTION-105']
head = node.head
emitter_discharge = node.emitter_discharge
demand_discharge = node.demand_discharge
```

To obtain the results on pipe 'LINK-40':

```
pipe = tm.get_link('LINK-40')
start_head = pipe.start_node_head
end_head = pipe.end_node_head
start_velocity = pipe.start_node_velocity
end_velocity = pipe.end_node_velocity
start_flowrate = pipe.start_node_flowrate
end_flowrate = pipe.end_node_flowrate
```

6.2 Time Step and Time Stamps

Additionally, the time step (in seconds) and the time stamps (in seconds from the start of the simulation) are also stored in the *tsnet.network.model.TransientModel* object. They can be retrieved by:

```
dt = tm.time_step
tt = tm.simulation_timestamps
```

The results can then be plotted with respect to the time stamps using **matplotlib** or any other preferred package, as shown in Figure 6.1:

```
import matplotlib.pyplot as plt
plt.plot(tt ,head)
```

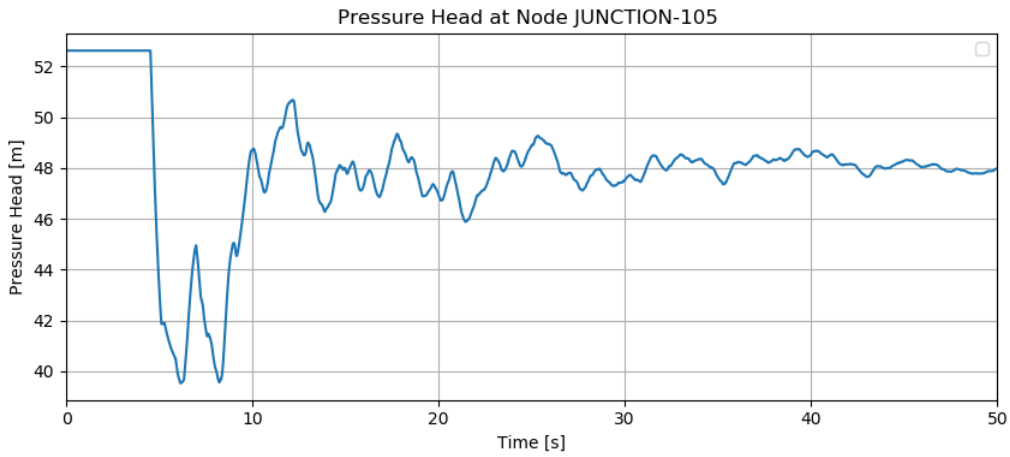

Figure 6.1: Head results at JUNCTION-105

6.3 Results Retrieval

The *tsnet.network.model.TransientModel* object, including the information of the network, operation rules, and the simulated results, is saved in the file **results_obj.obj**, located in the current folder. The name of the results file is defined by the input parameter *result_obj*. If *result_obj* is not given, the default results file is *results.obj*.

To retrieve the results from a previously completed simulation, one can read the *tsnet.network.model.TransientModel* object from the **results_obj.obj** file and access results from the objet by:

```
import pickle
file = open('results.obj', 'rb')
tm = pickle.load(file)
```

6.4 Runtime and Progress

At the beginning of transient simulation, TSNet will report the approximation simulation time based on the calculation time of first few time steps and the total number of time steps. Additionally, the computation progress will also printed on the screen as the simulation proceeds, as shown in Figure 6.2.

```
Simulation time step 0.14463 s
Total Time Step in this simulation 414
Estimated simulation time 0:00:01.245312
Transient simulation completed 9 %...
Transient simulation completed 19 %...
Transient simulation completed 29 %...
Transient simulation completed 39 %...
Transient simulation completed 49 %...
Transient simulation completed 59 %...
Transient simulation completed 69 %...
Transient simulation completed 79 %...
Transient simulation completed 89 %...
Transient simulation completed 99 %...
```

Figure 6.2: Runtime output about calculation time and process.

CHAPTER 7

Example Applications

7.1 Example 1 - End-valve closure

This example shows how to simulate the closure of a valve located at the boundary of a network. The first example network is shown below in Figure 7.1, adopted from [[STWY67],WOLB05]_. It comprises 9 pipes, 8 junctions, one reservoir, 3 closed loops, and one valve located at the downstream end of the system. There are five steps that the user needs to take to run the transient simulation using the TSNet package:

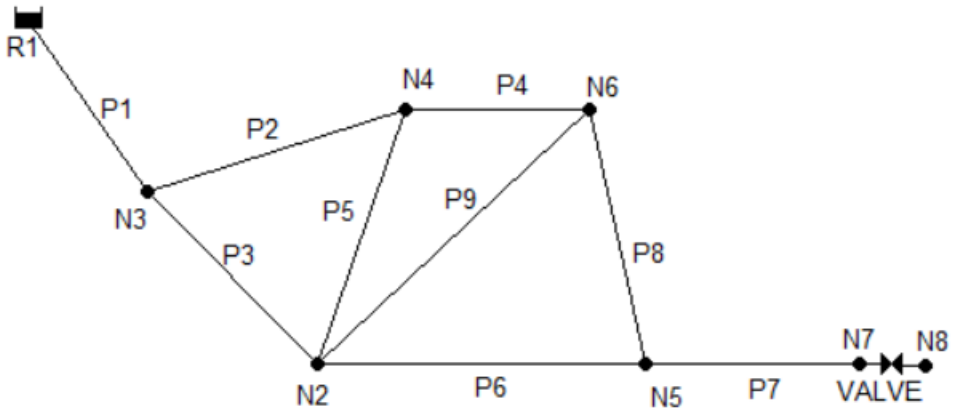

Figure 7.1: Tnet1 network graphics

1. Import TSNet package, read the EPANET INP file, and create transient model object.

```
import tsnet
# Open an example network and create a transient model
inp_file = 'examples/networks/Tnet1.inp'
tm = tsnet.network.TransientModel(inp_file)
```

2. Set the wave speed for all pipes to $1200m/s$, time step to $0.1s$, and simulation period to $60s$.

```
# Set wavespeed
tm.set_wavespeed(1200.) # m/s
# Set time options
dt = 0.1  # time step [s], if not given, use the maximum allowed dt
tf = 60   # simulation period [s]
tm.set_time(tf,dt)
```

3. Set valve operating rules, including how long it takes to close the valve (tc), when to start close the valve (ts), the opening percentage when the closure is completed (se), and the shape of the closure operating curve (m, 1 stands for linear closure, 2 stands for quadratic closure).

```
# Set valve closure
tc = 0.6 # valve closure period [s]
ts = 0 # valve closure start time [s]
se = 0 # end open percentage [s]
m = 1 # closure constant [dimensionless]
valve_op = [tc,ts,se,m]
tm.valve_closure('VALVE',valve_op)
```

4. Compute steady state results to establish the initial condition for transient simulation.

```
# Initialize steady state simulation
t0 = 0. # initialize the simulation at 0 [s]
engine = 'DD' # demand driven simulator
tm = tsnet.simulation.Initializer(tm, t0, engine)
```

5. Run transient simulation and specify the name of the results file.

```
# Transient simulation
results_obj = 'Tnet1' # name of the object for saving simulation results
tm = tsnet.simulation.MOCSimulator(tm, results_obj)
```

After the transient simulation, the results at nodes and links will be returned and stored in the transient model (tm) instance. The time history of flow rate on the start node of pipe P2 throughout the simulation can be retrieved by:

```
>>> print(tm.links['P2'].start_node_flowrate)
```

To plot the head results at N3:

```
import matplotlib.pyplot as plt
node = 'N3'
node = tm.get_node(node)
fig1 = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.head)
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Pressure Head at Node %s '%node)
plt.xlabel("Time [s]")
plt.ylabel("Pressure Head [m]")
plt.legend(loc='best')
```

(continues on next page)

(continued from previous page)

```
plt.grid(True)
plt.show()
```

yields Figure 7.2:

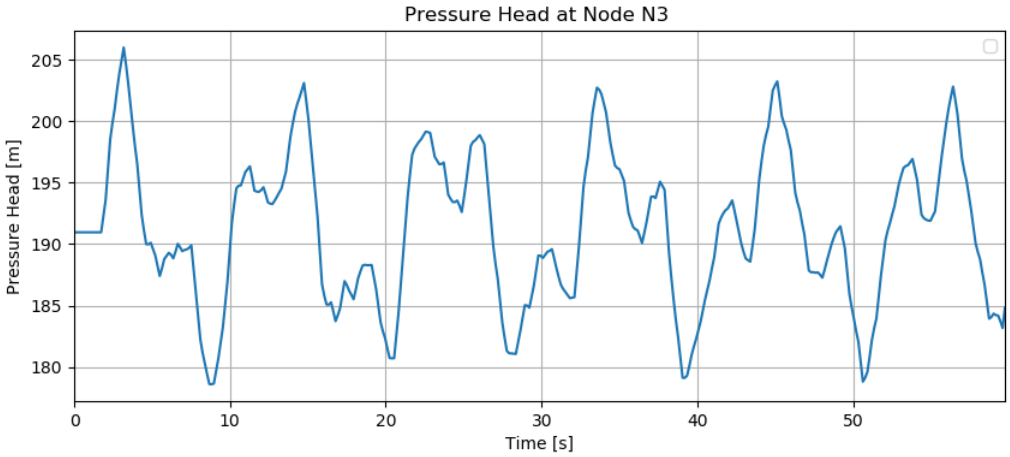

Figure 7.2: Tnet1 - Head at node N3.

Similarly, to plot the flow rate results in pipe P2:

```
pipe = 'P2'
pipe = tm.get_link(pipe)
fig2 = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,pipe.start_node_flowrate,
         label='Start Node')
plt.plot(tm.simulation_timestamps,pipe.end_node_flowrate,
         label='End Node')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Flowrate of Pipe %s '%pipe)
plt.xlabel("Time [s]")
plt.ylabel("Flow rate [m^3/s]")
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

yields Figure 7.3:

7.2 Example 2 - Pump operations

This example illustrates how the package models a transient event resulting from a controlled pump shut-off , i.e., the pump speed is ramped down. This example network, Tnet2, is shown below in Figure 7.4. Tnet2 comprises 113 pipes, 91 junctions, 2 pumps, 2 reservoir, 3 tanks, and one valve located in the middle of the network. A transient simulation of 50 seconds is generated by shutting off PUMP2. There are five steps user needs to take:

1. Import TSNet package, read the EPANET INP file, and create transient model object.

```
import tsnet
# open an example network and create a transient model
```

(continues on next page)

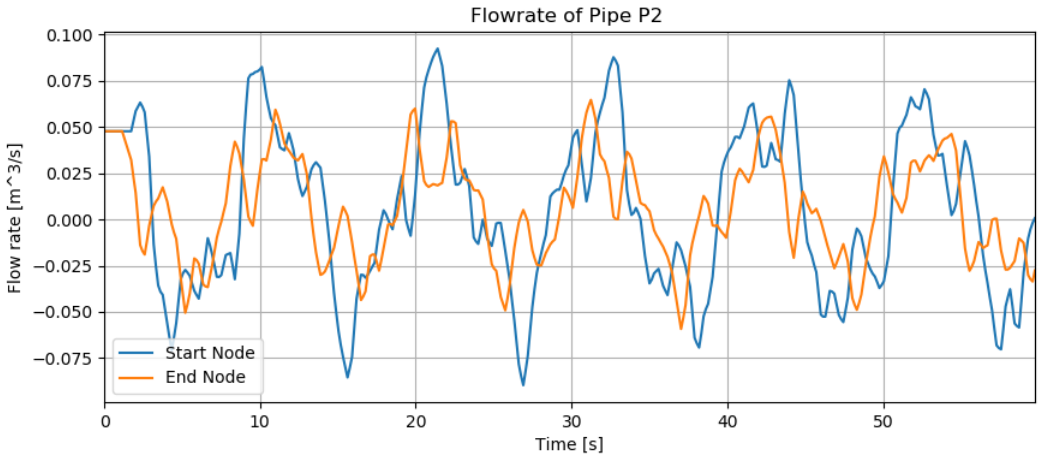

Figure 7.3: Tnet1 - Flow rate at the start and end node of pipe P2.

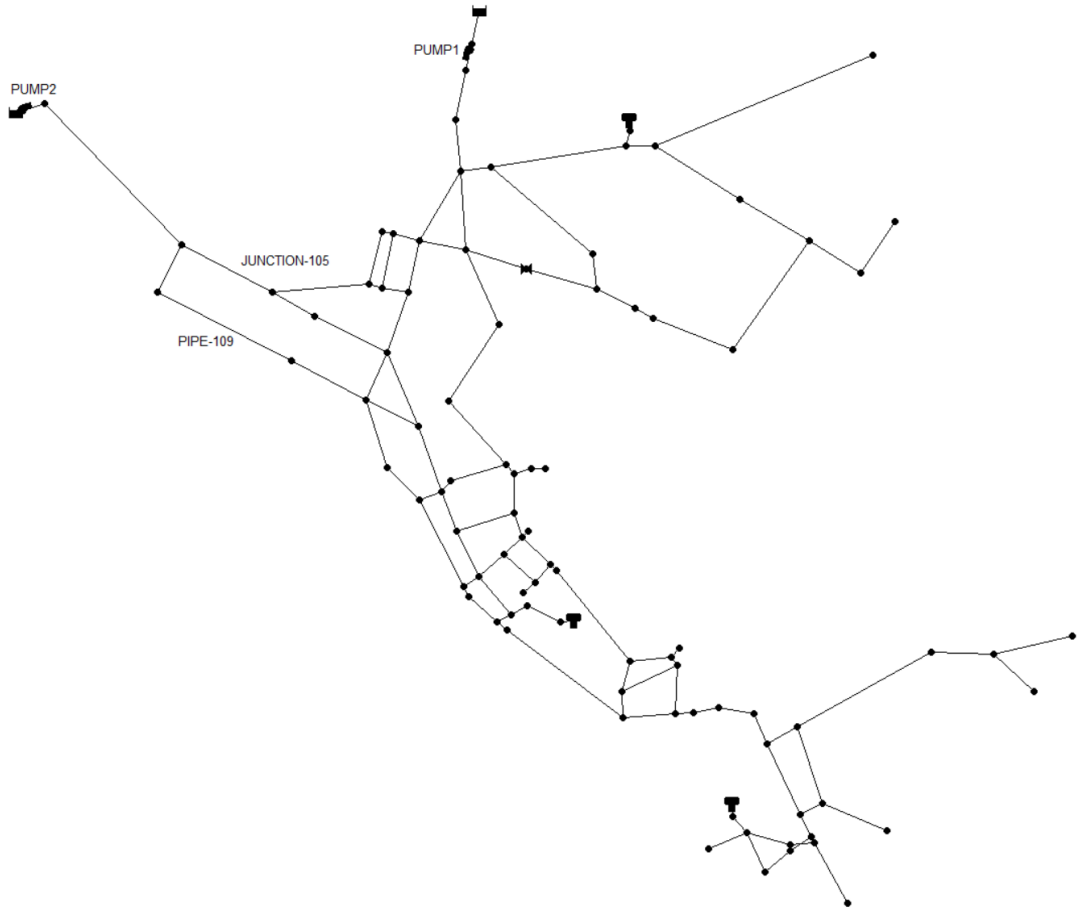

Figure 7.4: Tnet2 network graphics

(continued from previous page)

```
inp_file = 'examples/networks/Tnet2.inp'
tm = tsnet.network.TransientModel(inp_file)
```

2. Set the wave speed for all pipes to be $1200m/s$ and simulation period to be $50s$. Use suggested time step.

```
# Set wavespeed
tm.set_wavespeed(1200.)
# Set time step
tf = 50 # simulation period [s]
tm.set_time(tf)
```

3. Set pump operating rules, including how long it takes to shutdown the pump (tc), when to the shut-off starts (ts), the pump speed multiplier value when the shut-off is completed (se), and the shape of the shut-off operation curve (m, 1 stands for linear closure, 2 stands for quadratic closure).

```
# Set pump shut off
tc = 1 # pump closure period
ts = 0 # pump closure start time
se = 0 # end open percentage
m = 1 # closure constant
pump_op = [tc,ts,se,m]
tm.pump_shut_off('PUMP2', pump_op)
```

4. Compute steady state results to establish the initial condition for transient simulation.

```
# Initialize steady state simulation
t0 = 0. # initialize the simulation at 0s
engine = 'DD' # or PPD
tm = tsnet.simulation.Initializer(tm, t0, engine)
```

5. Run transient simulation and specify the name of the results file.

```
# Transient simulation
results_obj = 'Tnet2' # name of the object for saving simulation results
tm = tsnet.simulation.MOCSimulator(tm,results_obj)
```

After the transient simulation, the results at nodes and links will be returned to the transient model (tm) instance, which is then stored in **Tnet2.obj**. The actual demand discharge at JUNCTION-105 throughout the simulation can be retrieved by:

```
>>> print(tm.nodes['JUNCTION-105'].demand_discharge)
```

To plot the head results at JUNCTION-105:

```
import matplotlib.pyplot as plt
node = 'JUNCTION-105'
node = tm.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.head)
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Pressure Head at Node %s '%node)
plt.xlabel("Time [s]")
plt.ylabel("Pressure Head [m]")
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

yields Figure 6.1:

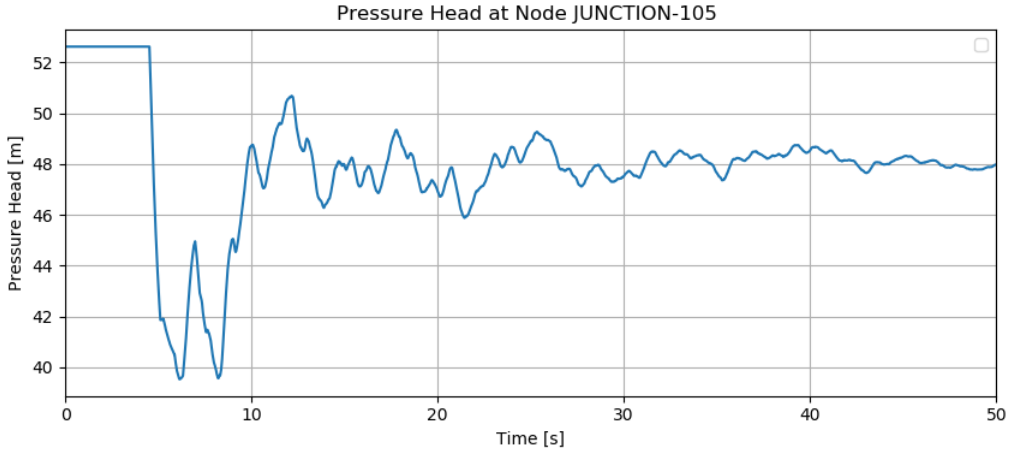

Figure 7.5: Tnet2 - Head at node JUNCTION-105.

Similarly, to plot the velocity results in PIPE-109:

```
pipe = 'PIPE-109'
pipe = tm.get_link(pipe)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,pipe.start_node_velocity,label='Start Node')
plt.plot(tm.simulation_timestamps,pipe.end_node_velocity,label='End Node')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Velocity of Pipe %s '%pipe)
plt.xlabel("Time [s]")
plt.ylabel("Velocity [m/s]")
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

yields Figure 7.6:

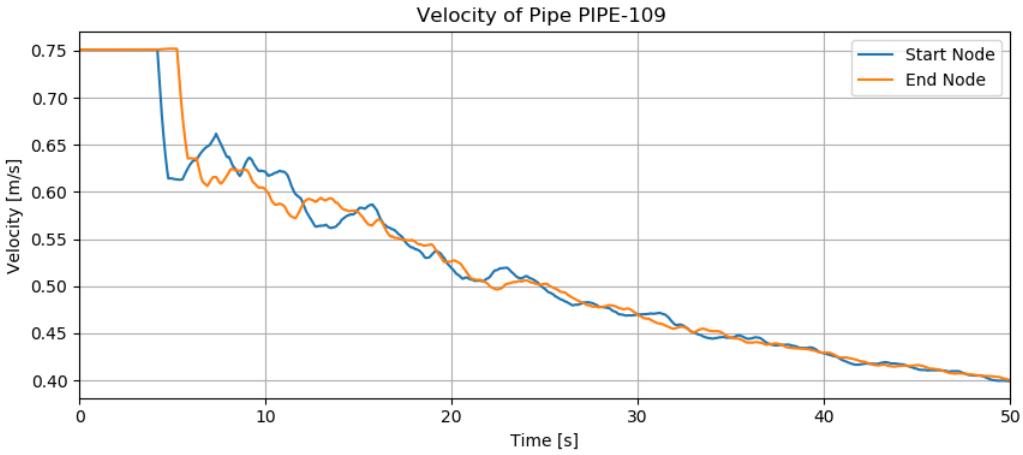

Figure 7.6: Tnet2 - Velocity at the start and end node of PIPE-109.

7.3 Example 3 - Burst and leak

This example reveals how TSNet simulates pipe bursts and leaks. This example network, adapted from [OSBH08], is shown below in Figure 7.7. Tnet3 comprises 168 pipes, 126 junctions, 8 valve, 2 pumps, one reservoir, and two tanks. The transient event is generated by a burst and a background leak. There are five steps that the user would need to take:

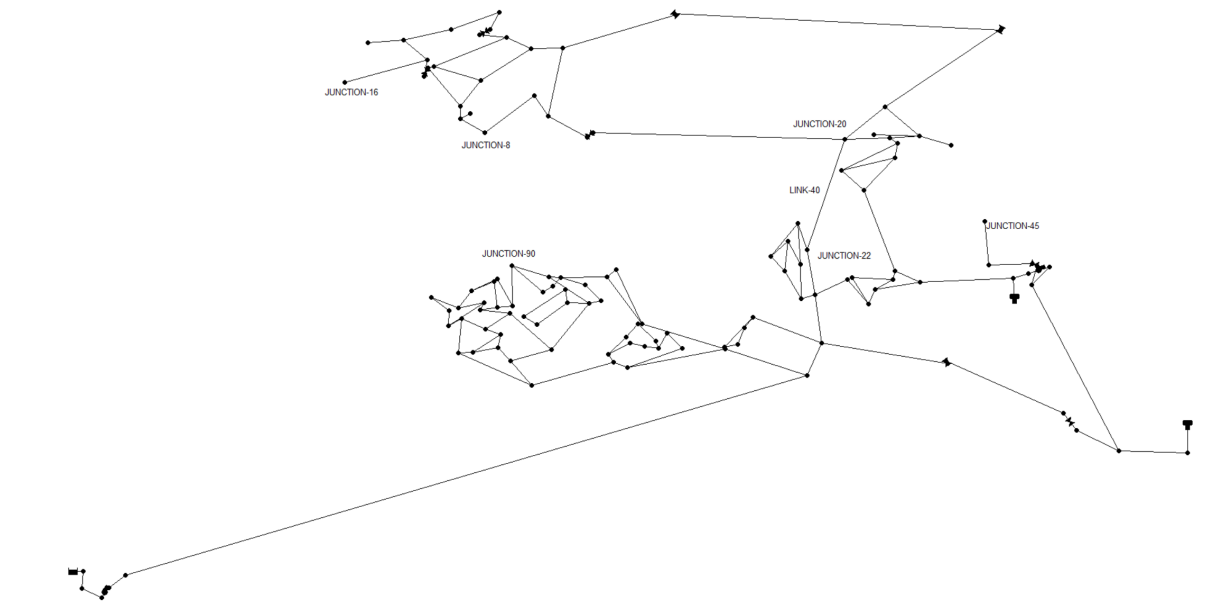

Figure 7.7: Tnet3 network graphics

1. Import TSNet package, read the EPANET INP file, and create transient model object.

```
import tsnet
# open an example network and create a transient model
inp_file = 'examples/networks/Tnet3.inp'
tm = tsnet.network.TransientModel(inp_file)
```

2. The user can import custom wave speeds for each pipe. To demonstrate how to assign different wave speed, we assume that the wave speed for the pipes is normally distributed with mean of $1200m/s$ and standard deviation of :math: *100m/s*. Then, assign the randomly generated wave speed to each pipe in the network according to the order the pipes defined in the INP file. Subsequently, set the simulation period as $20s$, and use suggested time step.

```
# Set wavespeed
import numpy as np
wavespeed = np.random.normal(1200., 100., size=tm.num_pipes)
tm.set_wavespeed(wavespeed)
# Set time step
tf = 20 # simulation period [s]
tm.set_time(tf)
```

3. Define background leak location, JUNCTION-22, and specify the emitter coefficient. The leak will be included in the initial condition calculation. See WNTR documentation [WNTRSi] for more info about leak simulation.

```
# Add leak
emitter_coeff = 0.01 # [ m^3/s/(m H20)^(1/2)]
tm.add_leak('JUNCTION-22', emitter_coeff)
```

4. Compute steady state results to establish the initial condition for transient simulation.

```
# Initialize steady state simulation
t0 = 0. # initialize the simulation at 0s
engine = 'DD' # or Epanet
tm = tsnet.simulation.Initializer(tm, t0, engine)
```

5. Set up burst event, including burst location, JUNCTION-20, burst start time (ts), time for burst to fully develop (tc), and the final emitter coefficient (final_burst_coeff).

```
# Add burst
ts = 1 # burst start time
tc = 1 # time for burst to fully develop
final_burst_coeff = 0.01 # final burst coeff [ m^3/s/(m H20)^(1/2)]
tm.add_burst('JUNCTION-20', ts, tc, final_burst_coeff)
```

6. Run transient simulation and specify the name of the results file.

```
# Transient simulation
result_obj = 'Tnet3' # name of the object for saving simulation results
tm = tsnet.simulation.MOCSimulator(tm,result_obj)
```

After the transient simulation, the results at nodes and links will be returned to the transient model (tm) instance, which is subsequently stored in **Tnet3.obj**.

To understand how much water has been lost through the leakage at JUNCTION-22, we can plot the leak discharge results at JUNCTION-22:

```
import matplotlib.pyplot as plt
node = 'JUNCTION-22'
node = tm.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.emitter_discharge)
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Leak discharge at Node %s '%node)
plt.xlabel("Time [s]")
plt.ylabel("Leak discharge [m^3/s]")
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

yields Figure 7.8:

Similarly, to reveal how much water has been wasted through the burst event at JUNCTION-20, we can plot the burst discharge results at JUNCTION-20:

```
node = 'JUNCTION-20'
node = tm.get_node(node)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node.emitter_discharge)
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Burst discharge at Node %s '%node)
plt.xlabel("Time [s]")
```

(continues on next page)

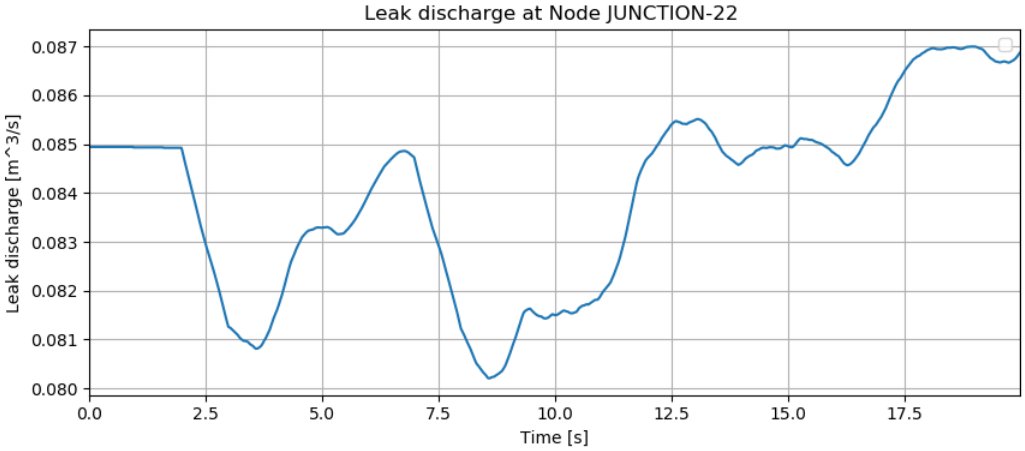

Figure 7.8: Tnet3 - Leak discharge at node JUNCTION-22.

(continued from previous page)

```
plt.ylabel("Burst discharge [m^3/s]")
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

yields Figure 7.9:

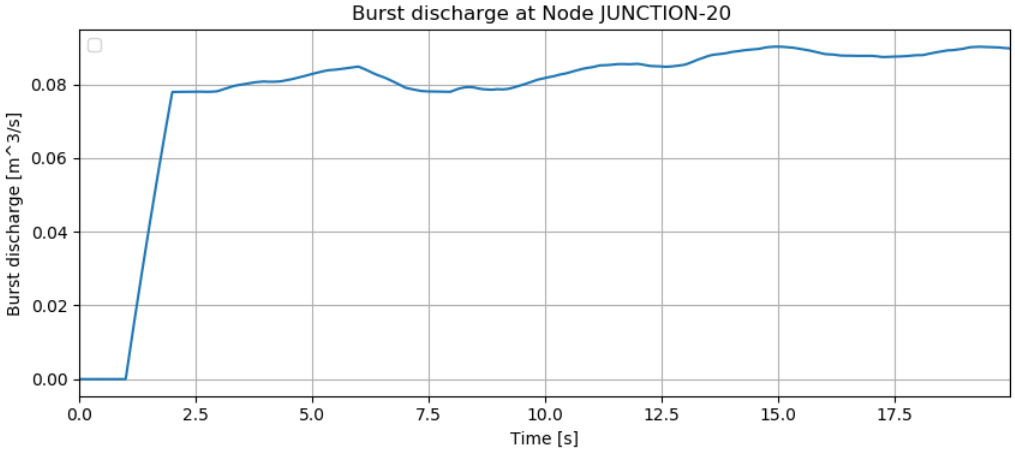

Figure 7.9: Tnet3 - Burst discharge at node JUNCTION-20.

Additionally, to plot the velocity results in LINK-40:

```
pipe = 'LINK-40'
pipe = tm.get_link(pipe)
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,pipe.start_node_velocity,label='Start Node')
plt.plot(tm.simulation_timestamps,pipe.end_node_velocity,label='End Node')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Velocity of Pipe %s '%pipe)
plt.xlabel("Time [s]")
```

(continues on next page)

(continued from previous page)

```
plt.ylabel("Velocity [m/s]")
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

yields Figure 7.10:

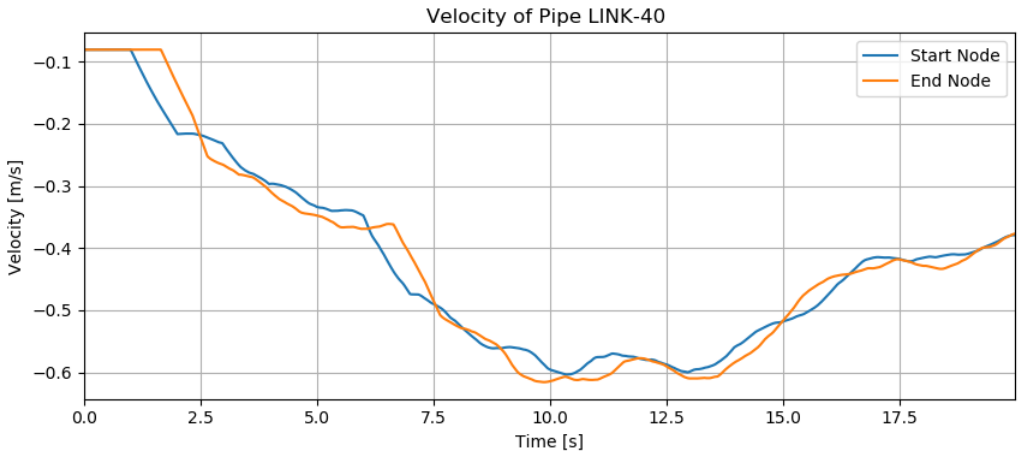

Figure 7.10: Tnet3 - Velocity at the start and end node of LINK-40.

Moreover, we can plot head results at some further nodes, such as JUNCTION-8, JUNCTION-16, JUNCTION-45, JUNCTION-90, by:

```
node1 = tm.get_node('JUNCTION-8')
node2 = tm.get_node('JUNCTION-16')
node3 = tm.get_node('JUNCTION-45')
node4 = tm.get_node('JUNCTION-90')
fig = plt.figure(figsize=(10,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(tm.simulation_timestamps,node1.head,label='JUNCTION-8')
plt.plot(tm.simulation_timestamps,node2.head,label='JUNCTION-16')
plt.plot(tm.simulation_timestamps,node3.head,label='JUNCTION-45')
plt.plot(tm.simulation_timestamps,node4.head,label='JUNCTION-90')
plt.xlim([tm.simulation_timestamps[0],tm.simulation_timestamps[-1]])
plt.title('Head on Junctions')
plt.xlabel("Time [s]")
plt.ylabel("Head [m]")
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

The results are demonstrated in Figure 7.11. It can be noticed that the amplitude of the pressure transient at JUNCTION-8 and JUNCTION-16 is greater than that at other two junctions which are further away from JUNCTION-20, where the burst occurred.

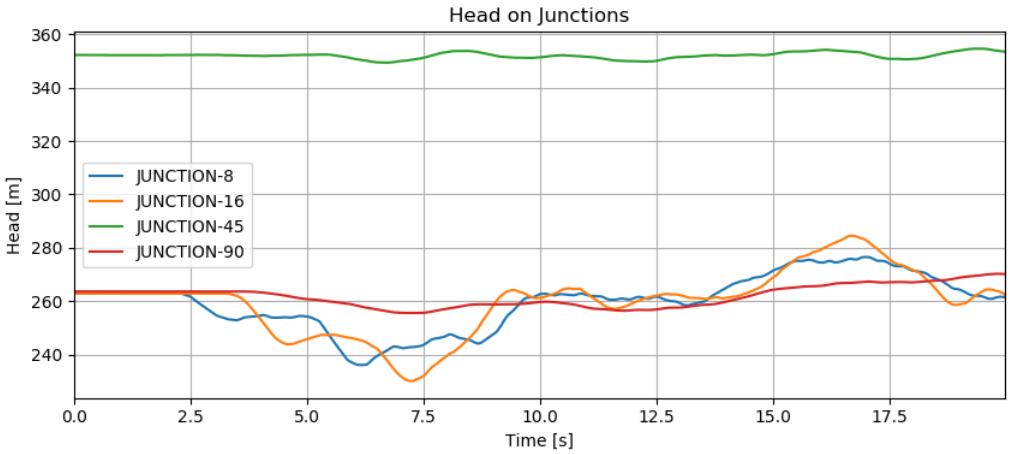

Figure 7.11: Tnet3 - Head at multiple junctions.

CHAPTER 8

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/glorialulu/TSNet/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

8.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

8.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

8.1.4 Write Documentation

TSNet could always use more documentation, whether as part of the official TSNet docs, in docstrings, or even on the web in blog posts, articles, and such.

8.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/glorialulu/TSNet/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

8.2 Get Started!

Ready to contribute? Here's how to set up *TSNet* for local development.

1. Fork the *TSNet* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/TSNet.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv TSNet
$ cd TSNet/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 TSNet tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

8.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/glorialulu/TSNet/pull_requests and make sure that the tests pass for all supported Python versions.

8.4 Tips

To run a subset of tests:

```
$ py.test .\tests\test_tsnet.py
```

8.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 9

Credits

9.1 Development Lead

- Lu Xing <xinglu@utexas.edu>
- Lina Sela <linasela@utexas.edu>

9.2 Contributors

- Gerardo Andres Riano Briceno <griano@utexas.edu>
- Ahmed A. Abokifa <ahmed.abokifa@utexas.edu>

CHAPTER 10

History

10.1 0.1.0 (2019-08-15)

- First release on PyPI.

CHAPTER 11

tsnet package

11.1 Subpackages

11.1.1 tsnet.network package

Submodules

tsnet.network.control module

The tsnet.network.control module includes method to define network controls of the pump and valve.These control modify parameters in the network during transient simulation.

tsnet.network.control.**valveclosing**(*dt*, *tf*, *valve_op*)
Define valve operation curve (percentage open v.s. time)

Parameters

- **dt** (*float*) – Time step
- **tf** (*float*) – Simulation Time
- **valve_op** (*list*) – Contains parameters to define valve operation rule valve_op = [tc,ts,se,m] tc : the duration takes to close the valve [s] ts : closure start time [s] se : final open percentage [s] m : closure constant [unitless]

Returns **s** – valve operation curve

Return type list

tsnet.network.control.**valveopening**(*dt*, *tf*, *valve_op*)
Define valve operation curve (percentage open v.s. time)

Parameters

- **dt** (*float*) – Time step
- **tf** (*float*) – Simulation Time

- **valve_op** (*list*) – Contains parameters to define valve operation rule valve_op = [tc,ts,se,m] tc : the duration takes to close the valve [s] ts : closure start time [s] se : final open percentage [s] m : closure constant [unitless]

Returns **s** – valve operation curve

Return type list

tsnet.network.control.**pumpclosing**(*dt*, *tf*, *pump_op*)
Define pump operation curve (percentage open v.s. time)

Parameters

- **dt** (*float*) – Time step
- **tf** (*float*) – Simulation Time
- **valve_op** (*list*) – Contains parameters to define valve operation rule valve_op = [tc,ts,se,m] tc : the duration takes to close the valve [s] ts : closure start time [s] se : final open percentage [s] m : closure constant [unitless]

Returns **s** – valve operation curve

Return type list

tsnet.network.control.**pumpopening**(*dt*, *tf*, *pump_op*)
Define pump operation curve (percentage open v.s. time)

Parameters

- **dt** (*float*) – Time step
- **tf** (*float*) – Simulation Time
- **pump_op** (*list*) – Contains parameters to define pump operation rule pump_op = [tc,ts,se,m] tc : the duration takes to start up the pump [s] ts : open start time [s] se : final open percentage [s] m : closure constant [unitless]

Returns **s** – valve operation curve

Return type list

tsnet.network.control.**burstsetting**(*dt*, *tf*, *ts*, *tc*, *final_burst_coeff*)
Calculate the burst area as a function of simulation time

Parameters

- **dt** (*float*) – Time step
- **tf** (*float*) – Simulation Time
- **ts** (*float*) – Burst start time
- **tc** (*float*) – Time for burst to fully develop
- **final_burst_coeff** (*list or float*) – Final emitter coefficient at the burst nodes

tsnet.network.discretize module

The tsnet.network.discretize contains methods to perform spatial and temporal discretization by adjusting wave speed and time step to solve compatibility equations in case of uneven wave travel time.

tsnet.network.discretize.**discretization**(*tm*, *dt*)
Discretize in temporal and spatial space using wave speed adjustment scheme.

Parameters

- **tm** (*tsnet.network.geometry.TransientModel*) – Network
- **dt** (*float*) – User defined time step

Returns **tm** – Network with updated parameters

Return type tsnet.network.geometry.TransientModel

tsnet.network.discretize.**max_time_step**(*tm*)
Determine the maximum time step based on Courant's criteria.

Parameters **tm** (*tsnet.network.geometry.TransientModel*) – Network

Returns **max_dt** – Maximum time step allowed for this network

Return type float

tsnet.network.discretize.**cal_N**(*tm*, *dt*)
Determine the number of computation unites (N_i) for each pipes.

Parameters

- **tm** (*tsnet.network.geometry.TransientModel*) – Network
- **dt** (*float*) – Time step for transient simulation

tsnet.network.discretize.**adjust_wavev**(*tm*)
Adjust wave speed and time step to solve compatibility equations.

Parameters **tm** (*tsnet.network.geometry.TransientModel*) – Network

Returns

- **tm** (*tsnet.network.geometry.TransientModel*) – Network with adjusted wave speed.
- **dt** (*float*) – Adjusted time step

tsnet.network.model module

The tsnet.network.geometry read in the geometry defined by EPANet .inp file, and assign additional parameters needed in transient simulation later in tsnet.

class tsnet.network.model.**TransientModel**(*inp_file*)
Bases: wntr.network.model.WaterNetworkModel

Transient model class. :param inp_file_name: Directory and filename of EPANET inp file to load into the

WaterNetworkModel object.

set_wavespeed(*wavespeed=1200.0*)
Set wave speed for pipes in the network

Parameters **wavespeed** (*float or int or list, optional*) – If given as float or int, set the value as wavespeed for all pipe; If given as list set the corresponding value to each pipe, by default 1200.

set_time(*tf*, *dt=None*)
Set time step and duration for the simulation.

Parameters

- **tf** (*float*) – Simulation period

- **dt** (*float, optional*) – time step, by default maximum allowed dt

add_leak(*name*, *coeff*)
Add leak to the transient model

Parameters

- **name** (*str, optional*) – The name of the leak nodes, by default None
- **coeff** (*list or float, optional*) – Emitter coefficient at the leak nodes, by default None

add_burst(*name*, *ts*, *tc*, *final_burst_coeff*)
Add leak to the transient model

Parameters

- **name** (*str*) – The name of the leak nodes, by default None
- **ts** (*float*) – Burst start time
- **tc** (*float*) – Time for burst to fully develop
- **final_burst_coeff** (*list or float*) – Final emitter coefficient at the burst nodes

valve_closure(*name*, *rule*)
Set valve closure rule

Parameters

- **name** (*str*) – The name of the valve to close
- **rule** (*list*) – Contains paramters to define valve operation rule rule = [tc,ts,se,m] tc : the duration takes to close the valve [s] ts : closure start time [s] se : final open percentage [s] m : closure constant [unitless]

valve_opening(*name*, *rule*)
Set valve opening rule

Parameters

- **name** (*str*) – The name of the valve to close
- **rule** (*list*) – Contains paramters to define valve operation rule rule = [tc,ts,se,m] tc : the duration takes to open the valve [s] ts : opening start time [s] se : final open percentage [s] m : closure constant [unitless]

pump_shut_off(*name*, *rule*)
Set pump shut off rule

Parameters

- **name** (*str*) – The name of the pump to shut off
- **rule** (*list*) – Contains paramaters to define valve operation rule rule = [tc,ts,se,m] tc : the duration takes to close the pump [s] ts : closure start time [s] se : final open percentage [s] m : closure constant [unitless]

pump_start_up(*name*, *rule*)
Set pump start up rule

Parameters

- **name** (*str*) – The name of the pump to shut off

- **rule** (*list*) – Contains paramaters to define valve operation rule rule = [tc,ts,se,m] tc : the duration takes to close the valve [s] ts : closure start time [s] se : final open percentage [s] m : closure constant [unitless]

tsnet.network.topology module

The tsnet.network.topology figure out the topology, i.e., upstream and downstream adjacent links for each pipe, and store the information in lists.

tsnet.network.topology.**topology**(*wn*)
Figure out the topology of the network

Parameters

- **wn** (*wntr.network.model.WaterNetworkModel*) – .inp file used for EPAnet simulation
- **npipe** (*integer*) – Number of pipes

Returns

- **links1** (*list*) – The id of adjacent pipe on the start node. The sign represents the direction of the pipe. + : flowing into the junction - : flowing out from the junction
- **links2** (*list*) – The id of adjacent pipe on the end node. The sign represents the direction of the pipe. + : flowing into the junction - : flowing out from the junction
- **utype** (*list*) – The type of the upstream adjacent links. If the link is not pipe, the name of that link will also be included. If there is no upstream link, the type of the start node will be recorded.
- **dtype** (*list*) – The type of the downstream adjacent links. If the link is not pipe, the name of that link will also be included. If there is no downstream link, the type of the end node will be recorded.

Module contents

The tsnet.network package contains methods to define 1. a water network geometry, 2. network topology, 3. network control, and 4 .spatial and temporal discretization.

11.1.2 tsnet.postprocessing package

Submodules

tsnet.postprocessing.time_history module

The tsnet.postprocessing.time_history module contains functions to plot the time history of head and velocity at the start and end point of a pipe

tsnet.postprocessing.time_history.**plot_head_history**(*pipe*, *H*, *wn*, *tt*)
Plot Head history on the start and end node of a pipe

Parameters

- **pipe** (*str*) – Name of the pipe where you want to report the head
- **H** (*list*) – Head results

- **wn** (*wntr.network.model.WaterNetworkModel*) – Network
- **tt** (*list*) – Simulation timestamps

tsnet.postprocessing.time_history.**plot_velocity_history**(*pipe*, *V*, *wn*, *tt*)
Plot Velocity history on the start and end node of a pipe

Parameters

- **pipe** (*str*) – Name of the pipe where you want to report the head
- **V** (*list*) – velocity results
- **wn** (*wntr.network.model.WaterNetworkModel*) – Network
- **tt** (*list*) – Simulation timestamps

Module contents

The tsnet.postprocessing package contains functions to postprocess the simulation results.

11.1.3 tsnet.simulation package

Submodules

tsnet.simulation.initialize module

The tsnet.simulation.initialize contains functions to 1. Initialize the list containing numpy arrays for velocity and head. 2. Calculate initial conditions using Epanet engine. 3. Calculate D-W coefficients based on initial conditions. 4. Calculate demand coefficients based on initial conditions.

tsnet.simulation.initialize.**Initializer**(*tm*, *t0*, *engine='DD'*)
Initial Condition Calculation.

Initialize the list containing numpy arrays for velocity and head. Calculate initial conditions using Epanet engine. Calculate D-W coefficients based on initial conditions. Calculate demand coefficients based on initial conditions.

Parameters

- **tm** (*tsnet.network.geometry.TransientModel*) – Simulated network
- **t0** (*float*) – time to calculate initial condition
- **engine** (*string*) – steady state calculation engine: DD: demand driven; PDD: pressure dependent demand, by default DD

Returns **tm** – Network with updated parameters

Return type tsnet.network.geometry.TransientModel

tsnet.simulation.initialize.**cal_demand_coef**(*demand*, *pipe*, *Hs*, *He*, *t0=0.0*)
Calculate the demand coefficient for the start and end node of the pipe.

Parameters

- **demand** (*list*) – Demand at the start (demand[0]) and end demand[1] node
- **pipe** (*object*) – Pipe object
- **Hs** (*float*) – Head at the start node
- **He** (*float*) – Head at the end node

- **t0** (*float, optional*) – Time to start initial condition calculation, by default 0

Returns **pipe** – Pipe object with calculated demand coefficient

Return type object

tsnet.simulation.initialize.**cal_roughness_coef**(*pipe*, *V*, *hl*)

Calculate the D-W roughness coefficient based on initial conditions.

Parameters

- **pipe** (*object*) – Pipe object
- **V** (*float*) – Initial flow velocity in the pipe
- **hl** (*float*) – Initial head loss in the pipe

Returns **pipe** – Pipe object with calculated D-W roughness coefficient.

Return type object

tsnet.simulation.main module

The tsnet.simulation.main module contains function to perform the workflow of read, discretize, initial, and transient simulation for the given .inp file.

tsnet.simulation.single module

The tsnet.simulation.single contains methods to perform MOC transient simulation on a single pipe, including 1. inner pipe 2. left boundary pipe (without C- charateristic grid) 3. right boundary pipe (without C+ characteristic grid)

tsnet.simulation.single.**inner_pipe**(*linkp*, *pn*, *dt*, *links1*, *links2*, *utype*, *dtype*, *p*, *H0*, *V0*, *H*, *V*, *H10*, *V10*, *H20*, *V20*, *pump*, *valve*)

MOC solution for an individual inner pipe.

Parameters

- **linkp** (*object*) – Current pipe object
- **pn** (*int*) – Current pipe ID
- **dt** (*float*) – Time step
- **H** (*numpy.ndarray*) – Head of current pipe at current time step [m]
- **V** (*numpy.ndarray*) – Velocity of current pipe at current time step [m/s]
- **links1** (*list*) – Upstream adjacent pipes
- **links2** (*list*) – Downstream adjacent pipes
- **utype** (*list*) – Upstream adjacent link type, and if not pipe, their name
- **dtype** (*list*) – Downstream adjacent link type, and if not pipe, their name
- **p** (*list*) – pipe list
- **n** (*int*) – Number of discretization of current pipe
- **H10** (*list*) – Head of left adjacent nodes at previous time step [m]
- **V10** (*list*) – Velocity of left adjacent nodes at previous time step [m/s]
- **H20** (*list*) – Head of right adjacent nodes at previous time step [m]
- **V20** (*list*) – Velocity of right adjacent nodes at previous time step [m/s]

- **pump** (*list*) – Characteristics of the pump
- **valve** (*list*) – Characteristics of the valve

Returns

- **H** (*numpy.ndarray*) – Head results of the current pipe at current time step. [m]
- **V** (*numpy.ndarray*) – Velocity results of the current pipe at current time step. [m/s]

tsnet.simulation.single.**left_boundary**(*linkp*, *pn*, *H*, *V*, *H0*, *V0*, *links2*, *p*, *pump*, *valve*, *dt*, *H20*, *V20*, *utype*, *dtype*)

MOC solution for an individual left boundary pipe.

Parameters

- **linkp** (*object*) – Current pipe object
- **pn** (*int*) – Current pipe ID
- **H** (*numpy.ndarray*) – Head of current pipe at current time step [m]
- **V** (*numpy.ndarray*) – Velocity of current pipe at current time step [m/s]
- **links2** (*list*) – Downstream adjacent pipes
- **p** (*list*) – pipe list
- **pump** (*list*) – Characteristics of the pump
- **valve** (*list*) – Characteristics of the valve
- **n** (*int*) – Number of discretization of current pipe
- **dt** (*float*) – Time step
- **H0** (*numpy.ndarray*) – Head of current pipe at previous time step [m]
- **V0** (*numpy.ndarray*) – Velocity of current pipe at previous time step [m/s]
- **H20** (*list*) – Head of right adjacent nodes at previous time step [m]
- **V20** (*list*) – Velocity of right adjacent nodes at previous time step [m/s]
- **utype** (*list*) – Upstream adjacent link type, and if not pipe, their name
- **dtype** (*list*) – Downstream adjacent link type, and if not pipe, their name

Returns

- **H** (*numpy.ndarray*) – Head results of the current pipe at current time step. [m]
- **V** (*numpy.ndarray*) – Velocity results of the current pipe at current time step. [m/s]

tsnet.simulation.single.**right_boundary**(*linkp*, *pn*, *H0*, *V0*, *H*, *V*, *links1*, *p*, *pump*, *valve*, *dt*, *H10*, *V10*, *utype*, *dtype*)

MOC solution for an individual right boundary pipe.

Parameters

- **linkp** (*object*) – Current pipe object
- **pn** (*int*) – Current pipe ID
- **H** (*numpy.ndarray*) – Head of current pipe at current time step [m]
- **V** (*numpy.ndarray*) – Velocity of current pipe at current time step [m/s]
- **links1** (*list*) – Upstream adjacent pipes
- **p** (*list*) – pipe list

- **pump** (*list*) – Characteristics of the pump
- **valve** (*list*) – Characteristics of the valve
- **n** (*int*) – Number of discretization of current pipe
- **dt** (*float*) – Time step
- **H0** (*numpy.ndarray*) – Head of current pipe at previous time step [m]
- **V0** (*numpy.ndarray*) – Velocity of current pipe at previous time step [m/s]
- **H10** (*list*) – Head of left adjacent nodes at previous time step [m]
- **V10** (*list*) – Velocity of left adjacent nodes at previous time step [m/s]
- **utype** (*list*) – Upstream adjacent link type, and if not pipe, their name
- **dtype** (*list*) – Downstream adjacent link type, and if not pipe, their name

Returns

- **H** (*numpy.ndarray*) – Head results of the current pipe at current time step. [m]
- **V** (*numpy.ndarray*) – Velocity results of the current pipe at current time step. [m/s]

tsnet.simulation.solver module

The tsnet.simulation.solver module contains methods to solver MOC for different grid configurations, including: 1. inner_node 2. valve_node 3. pump_node 4. source_pump 5. valve_end 6. dead_end 7. rev_end 8. add_leakage

tsnet.simulation.solver.**inner_node**(*link1*, *link2*, *demand*, *H1*, *V1*, *H2*, *V2*, *dt*, *g*, *nn*, *s1*, *s2*)
Inner boundary MOC using C+ and C- characteristic curve

Parameters

- **link1** (*object*) – Pipe object of C+ charateristics curve
- **link2** (*object*) – Pipe object of C- charateristics curve
- **demand** (*float*) – demand at the junction
- **H1** (*list*) – List of the head of C+ charateristics curve
- **V1** (*list*) – List of the velocity of C+ charateristics curve
- **H2** (*list*) – List of the head of C- charateristics curve
- **V2** (*list*) – List of the velocity of C- charateristics curve
- **dt** (*float*) – Time step
- **g** (*float*) – Gravity acceleration
- **nn** (*int*) – The index of the calculation node
- **s1** (*list*) – List of signs that represent the direction of the flow in C+ charateristics curve
- **s2** (*list*) – List of signs that represent the direction of the flow in C- charateristics curve

Returns

- **HP** (*float*) – Head at current node at current time
- **VP** (*float*) – Velocity at current node at current time

tsnet.simulation.solver.**valve_node**(*KL_inv*, *link1*, *link2*, *H1*, *V1*, *H2*, *V2*, *dt*, *g*, *nn*, *s1*, *s2*)
Inline valve node MOC calculation

Parameters

- **KL_inv** (*int*) – Inverse of the valve loss coefficient at current time
- **link1** (*object*) – Pipe object of C+ charateristics curve
- **link2** (*object*) – Pipe object of C- charateristics curve
- **H1** (*list*) – List of the head of C+ charateristics curve
- **V1** (*list*) – List of the velocity of C+ charateristics curve
- **H2** (*list*) – List of the head of C- charateristics curve
- **V2** (*list*) – List of the velocity of C- charateristics curve
- **dt** (*float*) – Time step
- **g** (*float*) – Gravity acceleration
- **nn** (*int*) – The index of the calculation node
- **s1** (*list*) – List of signs that represent the direction of the flow in C+ charateristics curve
- **s2** (*list*) – List of signs that represent the direction of the flow in C- charateristics curve

tsnet.simulation.solver.**pump_node**(*pumpc*, *link1*, *link2*, *H1*, *V1*, *H2*, *V2*, *dt*, *g*, *nn*, *s1*, *s2*)

Inline pump node MOC calculation

Parameters

- **pumpc** (*list*) – Parameters (a, b,c) to define pump characteristic cure, so that .. math:: h_p = a*Q**2 + b*Q + c
- **link1** (*object*) – Pipe object of C+ charateristics curve
- **link2** (*object*) – Pipe object of C- charateristics curve
- **H1** (*list*) – List of the head of C+ charateristics curve
- **V1** (*list*) – List of the velocity of C+ charateristics curve
- **H2** (*list*) – List of the head of C- charateristics curve
- **V2** (*list*) – List of the velocity of C- charateristics curve
- **dt** (*float*) – Time step
- **g** (*float*) – Gravity acceleration
- **nn** (*int*) – The index of the calculation node
- **s1** (*list*) – List of signs that represent the direction of the flow in C+ charateristics curve
- **s2** (*list*) – List of signs that represent the direction of the flow in C- charateristics curve

tsnet.simulation.solver.**source_pump**(*pump*, *link2*, *H2*, *V2*, *dt*, *g*, *s2*)

Source Pump boundary MOC calculation

Parameters

- **pump** (*list*) – pump[0]: elevation of the reservoir/tank pump[1]: Parameters (a, b,c) to define pump characteristic cure, so that .. math:: h_p = a*Q**2 + b*Q + c
- **link2** (*object*) – Pipe object of C- charateristics curve
- **H2** (*list*) – List of the head of C- charateristics curve
- **V2** (*list*) – List of the velocity of C- charateristics curve

- **dt** (*float*) – Time step
- **g** (*float*) – Gravity acceleration
- **s2** (*list*) – List of signs that represent the direction of the flow in C- charateristics curve

tsnet.simulation.solver.**valve_end**(*H1*, *V1*, *V*, *nn*, *a*, *g*, *f*, *D*, *dt*)
End Valve boundary MOC calculation

Parameters

- **H1** (*float*) – Head of the C+ charateristics curve
- **V1** (*float*) – Velocity of the C+ charateristics curve
- **V** (*float*) – Velocity at the valve end at current time
- **nn** (*int*) – The index of the calculation node
- **a** (*float*) – Wave speed at the valve end
- **g** (*float*) – Gravity acceleration
- **f** (*float*) – friction factor of the current pipe
- **D** (*float*) – diameter of the current pipe
- **dt** (*float*) – Time step

tsnet.simulation.solver.**dead_end**(*linkp*, *H1*, *V1*, *nn*, *a*, *g*, *f*, *D*, *dt*)
Dead end boundary MOC calculation with pressure dependant demand

Parameters

- **link1** (*object*) – Current pipe
- **H1** (*float*) – Head of the C+ charateristics curve
- **V1** (*float*) – Velocity of the C+ charateristics curve
- **nn** (*int*) – The index of the calculation node
- **a** (*float*) – Wave speed at the valve end
- **g** (*float*) – Gravity acceleration
- **f** (*float*) – friction factor of the current pipe
- **D** (*float*) – diameter of the current pipe
- **dt** (*float*) – Time step

tsnet.simulation.solver.**rev_end**(*H2*, *V2*, *H*, *nn*, *a*, *g*, *f*, *D*, *dt*)
Reservoir/ Tank boundary MOC calculation

Parameters

- **H2** (*list*) – List of the head of C- charateristics curve
- **V2** (*list*) – List of the velocity of C- charateristics curve
- **H** (*float*) – Head of the reservoir/tank
- **nn** (*int*) – The index of the calculation node
- **a** (*float*) – Wave speed at the valve end
- **g** (*float*) – Gravity acceleration
- **f** (*float*) – friction factor of the current pipe

- **D** (*float*) – diameter of the current pipe
- **dt** (*float*) – Time step

tsnet.simulation.solver.**add_leakage**(*emitter_coef*, *link1*, *link2*, *elev*, *H1*, *V1*, *H2*, *V2*, *dt*, *g*, *nn*, *s1*, *s2*)

Leakage Node MOC calculation

Parameters

- **emitter_coef** (*float*) – float, optional Required if leak_loc is defined The leakage coefficient of the leakage .. math:: Q_leak = leak_A [m^3/s/(m H20)^(1/2)] * sqrt(H)
- **link1** (*object*) – Pipe object of C+ charateristics curve
- **link2** (*object*) – Pipe object of C- charateristics curve
- **H1** (*list*) – List of the head of C+ charateristics curve
- **V1** (*list*) – List of the velocity of C+ charateristics curve
- **H2** (*list*) – List of the head of C- charateristics curve
- **V2** (*list*) – List of the velocity of C- charateristics curve
- **dt** (*float*) – Time step
- **g** (*float*) – Gravity acceleration
- **nn** (*int*) – The index of the calculation node
- **s1** (*list*) – List of signs that represent the direction of the flow in C+ charateristics curve
- **s2** (*list*) – List of signs that represent the direction of the flow in C- charateristics curve

Module contents

The tsnet.simulation package contains methods to run transient simulation using MOC method

11.1.4 tsnet.utils package

Submodules

tsnet.utils.calc_parabola_vertex module

The tsnet.utils.calc_parabola_vertex contains function to calculate the parameters of a parabola based on the coordinated of three points on the curve.

tsnet.utils.calc_parabola_vertex.**calc_parabola_vertex**(*points*)

Adapted and modifed to get the unknowns for defining a parabola

Parameters **points** (*list*) – Three points on the pump characterisc curve.

tsnet.utils.memo module

tsnet.utils.memo.**decorator**(*d*)

Make function d a decorator: d wraps a function fn.

tsnet.utils.memo.**memo**(*f*)

Decorator that caches the return value for each call to f(args). Then when called again with same args, we can just look it up.

tsnet.utils.print_time_delta module

tsnet.utils.print_time_delta.**print_time_delta**(*seconds*)

tsnet.utils.valve_curve module

The tsnet.utils.valve_curve contains function to define valve characteristics curve, gate valve by default.

tsnet.utils.valve_curve.**valve_curve**(*s*, *valve='Gate'*)

Define valve curve

Parameters

- **s** (*float*) – open percentage
- **valve** (*str, optional*) – [description], by default 'Gate'

Returns **k** – Friction coeffient with given open percentage

Return type float

Module contents

The tsnet.utils package contains helper functions.

11.2 Module contents

Top-level package for tsnet.

CHAPTER 12

Abbreviations

API: Application programming interface

EPA: Environmental Protection Agency

IDE: Integrated development environment

SI: International System of Units

US: United States

MOC: Method of Characteristics

TSNET: Transient Simulation in water Networks

CHAPTER 13

Reference

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

Bibliography

[WYSS93] Wylie, E. B., Streeter, V. L., & Suo, L. (1993). Fluid transients in systems (Vol. 1, p. 464). Englewood Cliffs, NJ: Prentice Hall.

[WNTRSi] Klise, K. A., Hart, D., Moriarty, D., Bynum, M. L., Murray, R., Burkhardt, J., & Haxton, T. (2017). Water network tool for resilience (WNTR) user manual. US Environmental Protection Agency, EPA/600/R-17/264, Cincinnati, OH.

[LAJW99] Larock, B. E., Jeppson, R. W., & Watters, G. Z. (1999). Hydraulics of pipeline systems. CRC press.

[STWV96] Street, R. L., Watters, G. Z., & Vennard, J. K. (1996). Elementary fluid mechanics. J. Wiley.

[WOLB05] Wood, D. J., Lingireddy, S., Boulos, P. F., Karney, B. W., & McPherson, D. L. (2005). Numerical methods for modeling transient flow in distribution systems. Journal-American Water Works Association, 97(7), 104-115.

[RERS15] Rezaei, H., Ryan, B., & Stoianov, I. (2015). Pipe failure analysis and impact of dynamic hydraulic conditions in water supply networks. Procedia Engineering, 119, 253-262.

[ASCE17] ASCE. (2017). 2017 infrastructure report card. Reston, VA: ASCE.

[VaCV11] 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.

[HaSS08] 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.

[Hunt07] Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. Computing in Science and Engineering, 9(3), 90-95.

[MISI08] Misiūnas, D. (2008). Failure monitoring and asset condition assessment in water supply systems. Vilnius Gediminas Technical University.

[STWY67] Streeter, V. L., & Wylie, E. B. (1967). Hydraulic transients (No. BOOK). mcgraw-hill.

[OSBH08] Ostfeld, A., Uber, J. G., Salomons, E., Berry, J. W., Hart, W. E., Phillips, C. A., ... & di Pierro, F. (2008). The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. Journal of Water Resources Planning and Management, 134(6), 556-568.

Python Module Index

t

Index

V