OOPNET

Release 0.6.5

David B. Steffelbauer

Oct 05, 2023

CONTENTS

OOPNET (Object-Oriented Pipe Network Analyzer) is an easy-to-use programming interface of EPANET to students, researchers and scientists who are not blessed with sophisticated programming skills. It enables people to easily prototype Python code for different water distribution related tasks. Right now it is in an alpha stage, meaning that it is nothing more than a wrapper around EPANET's command line interface that is able to manipulate EPANET .inp files, run a EPANET simulation through the command line interface and parse the information contained in the .rpt and .bin files.

Although it is a little bit more, as you will see reading this documentation, since OOPNET

- is object-oriented
- has multi-dimensional xray support for simulation reports
- is easy to run in parallel
- makes beautiful, paper ready plots

The vision of OOPNET is to build an open-source community around it, connect programmers and wanna-be programmers in the water distribution related community and build a stand-alone hydraulic solver resulting in a totally in Python written water distribution system software making it nice and shiny from the same mould.

Warning: Be warned, that OOPNET is still changing a lot. Until it's marked as 1.0.0, you should assume that it is unstable and act accordingly. We are trying to avoid breaking changes but they can and will occur!

CHAPTER

ONE

CITING OOPNET

If you publish something that uses OOPNET, we greatly appreciate a citation of the following reference:

- Steffelbauer, D., Fuchs-Hanusch, D., 2015. OOPNET: an object-oriented EPANET in Python. Procedia Eng. 119, 710e718. https://doi.org/10.1016/j.proeng.2015.08.924.

CHAPTER

TWO

CONTENTS

2.1 Introduction

2.1.1 Why OOPNET?

Since the WDSA2010 conference, the water related community is waiting for EPANET3. Different attempts have been made in the past to convert EPANET[1] into an open-source project maintained by a bigger community instead of a single developer. To obtain this goal, the whole (or parts of the) software has to be rewritten in an object-oriented way, enabling many geographically separated programmers to contribute.

So far the provided solutions were often written in programming languages with a steep learning curve (e.g. C++, C#, Java). In contrary, Python is an easy to learn open-source language with a wide range of additional packages for multiple tasks. In 2022, Python was the most popular programming language referring to the PYPL (PopularitY of Programming Language; http://pypl.github.io/PYPL.html) index with a share of 28.38 % worldwide. Therefore, we chose Python as programming language to rewrite EPANET to an object-oriented EPANET (OOPNET).

So far the object-oriented structure of OOPNET is similar to CWSNET[2], OOTEN[3] or PORTEAU[4] with the focus on being as easy as possible to handle for end-users. EPANET input files can be translated into this structure and are manipulated and simulated with EPANET's command-line interface through Python. The reports are translated in Pandas[5], a Python data analysis library, to enable fast and easy-to-use data analysis of the results. Plotting the water distribution network as well as the results is performed either with MatplotLib[6], a plotting package similar to MATLAB, or directly in the user's browser or web with Bokeh[7].

[1] Rossman L., Woo H., Tryby M., Shang F., Janke R. Haxton T., "EPANET 2.2 User Manual" (Washington, D.C: U.S. Environmental Protection Agency, 2020).

[2] Guidolin M., Burovskiy P., Kapelan Z. Savić D., "CWSNET: An Object-Oriented Toolkit for Water Distribution System Simulations," in Water Distribution Systems Analysis 2010 (Tucson, Arizona, United States: American Society of Civil Engineers, 2011), pp.1–13.

[3] Van Zyl, J. E., Borthwick, J., Hardy, A., OOTEN: An object-oriented programmers toolkit for epanet. Advances in water supply management (CCWI 2003), CCWI, Sep 2003, London, United Kingdom. pp.1-8.

[4] Piller O., Gilbert D., Haddane K., Sabatié S., Porteau: An Object-Oriented programming hydraulic toolkit for water distribution system analysis. Eleventh International Conference on Computing and Control for the Water Industry (CCWI 2011), CCWI, Sep 2011, Exeter, United Kingdom. pp.27-32.

[5] Pandas, an open source data analysis and manipulation tool.

[6] Matplotlib, a visualization library for static, animated and interactive graphics.

[7] Bokeh, a library for interactive visualizations in web browsers.

2.1.2 Why Python?

Python is a widely used high level programming language, with a design philosophy that emphasizes code readability and is easy to learn compared to other programming languages with the same capabilities, like Java or C++.

The core philosophy of Python is summarized in the PEP 20 document, called the Zen of Python[8]. Here are the first lines of this document, to get an idea of the core philosophy:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

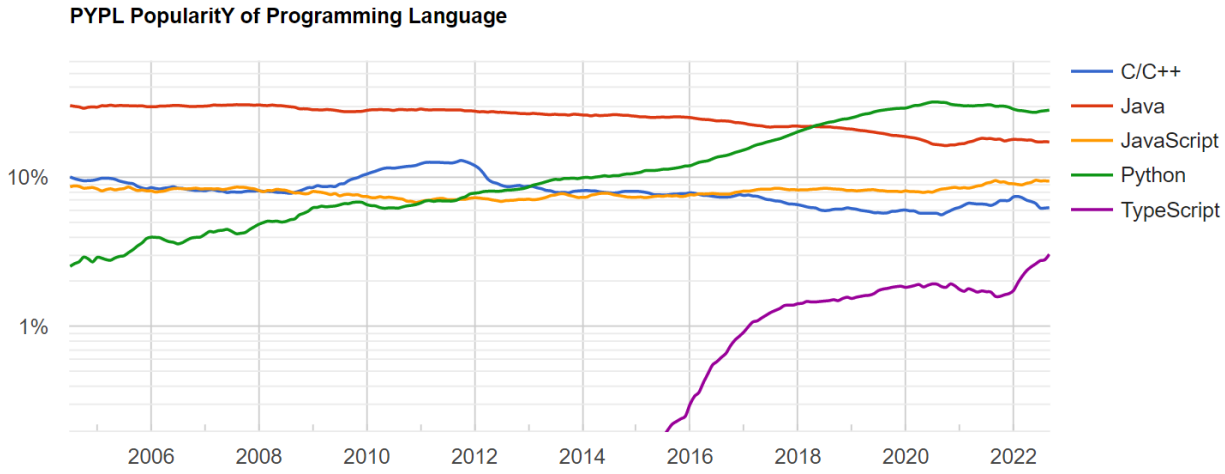


Fig. 1: PYPL (PopularitY of Programming Language) Index showing the growing popularity of Python worldwide. Taken from http://pypl.github.io/PYPL.html.

Because of its simple syntax and its various applications, Python is more and more used around the whole world and the user group is still growing (see Fig. 1), potentially replacing e.g. MATLAB, in the scientific community.

Furthermore, one of the main strengths of Python is that it has a large standard library providing tools suited to various tasks. This is described as its "batteries included" philosophy. By September 2022, the Python Package Index contained more than 400.000 Python projects offering a wide range of functionality.

2.1.3 Program Design

The design philosophy of OOPNET consists of two main objectives, usability and collaboration.

First, usability should be as good as possible, enabling water engineers to implement their ideas through readable and writeable code. Moreover, OOPNET should warn users, if they make programming mistakes. Mistakes, which may be hard to find later on if OOPNET is getting more complex due to further development, have to be reported in an early stage.

Second, OOPNET is designed in a way to make collaboration possible for various geographically separated contributors. Therefore, OOPNET consists of different packages containing modules with classes, having the object oriented design paradigm included in its foundation.

[8] PEP 20 - The Zen of Python.

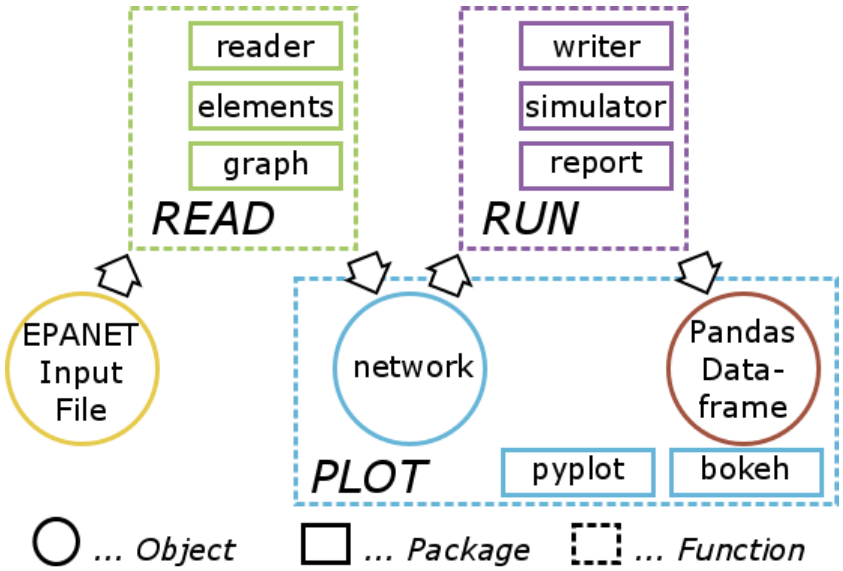


Fig. 2: OOPNET pacakage structure respectively workflow of the user

The overall package structure is described in Fig. 2, which is structured to represent the workflow of a potential user. Starting point is an EPANET Input file read in by the function Read. The Read function makes use of the three packages reader, elements and graph. Package reader contains the functions, which are necessary to translate arbitrary EPANET Input files in an object-oriented structure, which is implemented in the elements package.

Moreover, the graph package translates the water distribution network in a Python NetworkX graph object[9], enabling the application of advanced algorithms from graph theory. The output of the Read function is an OOPNET network object, which can be manipulated by the user in a simple Python syntax.

After the manipulation of the network by the user, the simulation of the network is started with the function Run. Run consists of three Python packages as well, namely writer, simulator and report. OOPNET makes use of the command line EPANET as no hydraulic solver has been implemented so far. Therefore, from the manipulated network object a new EPANET Input file has to be generated. This is done using the package writer. Afterwards, command line EPANET is called with the simulator package and the results of the simulation from EPANET's report file are read in by the package report. This translates the report file in a Python Pandas Dataframe. Pandas is a Python data analysis library enabling the user to get all information out of the report file with an easy to use syntax, containing fast statistical data analysis algorithms. Furthermore, the network respectively the simulation results can be plotted by the function Plot, which contains the packages pyplot and bokeh. The package pyplot makes use of Python's plotting library MatplotLib, which produces publication quality figures in an easy to use syntax similar to the programming language MATLAB. On the other hand, bokeh uses Python's library Bokeh, an interactive visualization library that targets modern web browsers for presentation.

Here is a code snippet of a simple example is presented, to make the use of OOPNET less abstract.

```
network = on.Network.read(filename)

for p in network.pipes:
    if p.diameter > 500:
        p.roughness = 2.0

report = network.run()
```

(continues on next page)

[9] NetworkX, a library for studying graphs and networks.

(continued from previous page)

```
print(report.pressure.mean())
```

Note: Imagine a user of OOPNET wants to change the roughness values of all pipes with a diameter greater than 500 mm to the value 2 mm. Subsequently, the user wants to analyze the mean pressure in the system for calculating e.g. the ILI (Infrastructure Leakage Index). Therefore, the network is loaded with the Read function in the first line. The next line leads to an iteration over all pipes in the system with Python's for-loop, asking for the pipes with a diameter greater than 500 mm with the if function and setting the pipe's roughness to the desired value of 2 mm. Subsequently, the network is simulated with the Run function and a report is generated. The last line leads to a print of the mean over all nodal pressures given in the generated report.

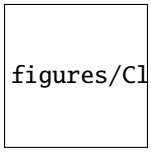


Fig. 3: Class structure of OOPNET implemented in the elements package

The object-oriented structure implemented in the elements package is represented in Fig. 4, which shows the object-oriented design paradigm with inheritance of classes and properties. Inheritance is depicted as a black arrow, e.g. the class Junction is a child of class Node, which is again the child of the class Network Component.

Additionally, Fig. 4 shows, that if a class has another class as one of its properties, it is depicted as dotted arrow. For example, a Link has always a start-node and an end-node. Therefore the Link class has an instance of the class Node as one of its properties.

The elements package is subdivided in several Python modules, to guarantee a higher level of modularity in the code. This increases the possibility of collaboration for more programmers, since the programmers are able to work on different files. The files are named according to the EPANET manual Input file structure (Network Components, System Operation, Water Quality, ...) and are shown in different colors in Fig. 4. This structure assists collaborators, which are new to OOPNET but used to EPANET, to quickly get familiar with OOPNET. In addition, the reader and writer packages are structured in a similar way.

In Fig. 5 the properties of the Junction class as an example for the property structure of all the other classes in OOPNET is shown. Only the properties emittercoefficient, demandpattern and demand are defined in the Junction class whereas all other properties are inherited from the Node or the Network Component class. Also the properties sourcepattern and demandpattern are stressed out in Fig. 5, since they refer to another instance of a class, namely Pattern, which has again properties.

On top of the elements class structure is the network object, which is again a class with properties consisting of Python lists of the classes of elements, describing the whole network and its physical properties respectively the simulation parameters (Fig. 6). An example of a bokeh plot of a network and its simulation results is shown in Fig. 7. The node pressures and the pipe flows are depicted in different colors. On top of the figure the bokeh's menu with different tools, like panning, zooming, refreshing or exporting, can be seen.

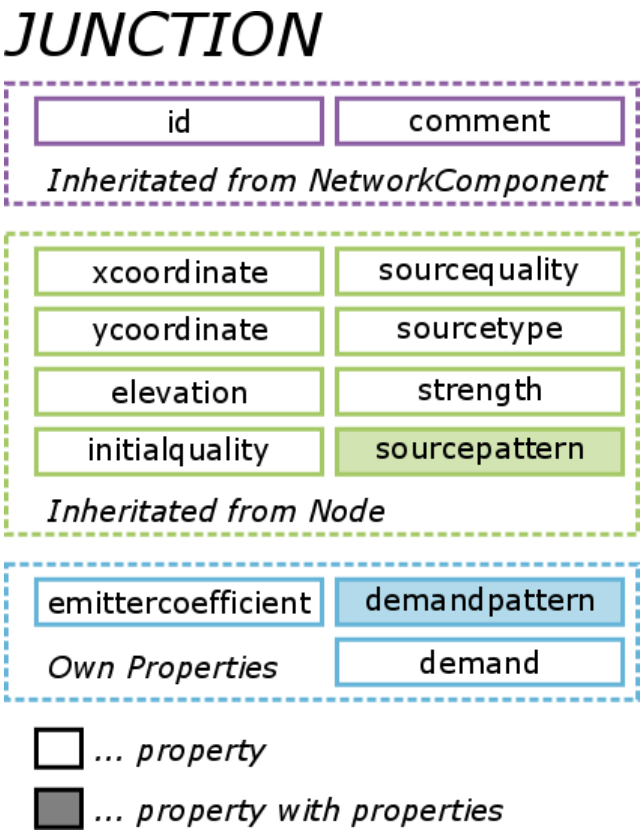


Fig. 4: Properties of OOPNET's Junction class

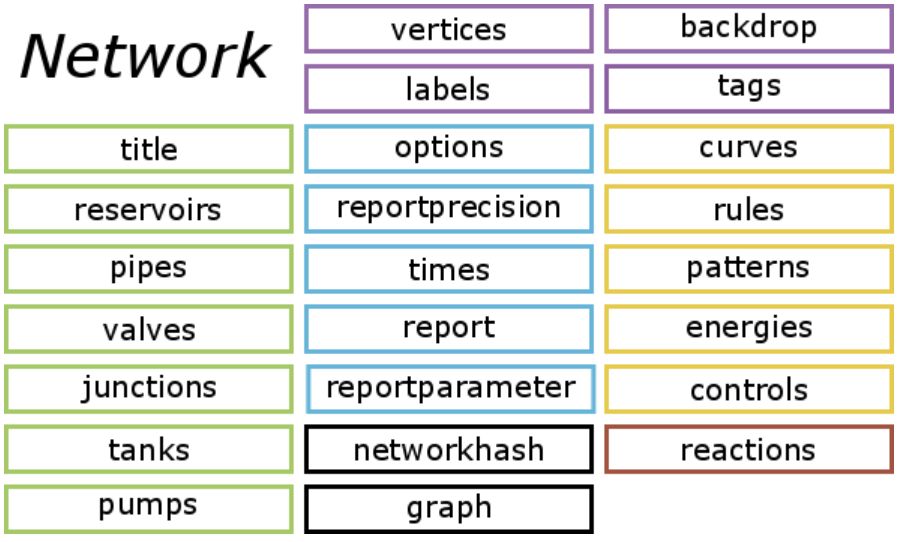


Fig. 5: Properties of the Network class object containing all the information from an EPANET input file

2.2 Getting started

2.2.1 Installation

OOPNET uses features only available in newer Python versions, which is why Python >= 3.9 is needed along with several Python package dependencies.

Note: If your are using Linux, EPANET has to be available in you command line interface.

To test, if EPANET is available, open a terminal and run:

```
epanet
```

OOPNET is available on PyPI and can be easily installed together with its dependencies using *pip*:

```
pip install oopnet
```

Alternatively, you can install OOPNET from its repository:

```
pip install git+https://github.com/oopnet/oopnet.git
```

2.2.2 Basic Usage

To use OOPNET, you first have to import it in your script:

```
import oopnet as on
```

In OOPNET, everything is about the *Network*. If you want to start with a new, empty Network, type the following:

```
network = on.Network()
```

If you want to read an existing EPANET model, you can *read* it as an input-file:

```
filename = "network.inp"
network = on.Network.read(filename)
```

To simulate the model, you can use the Network`s *run* method:

```
report = network.run()
```

If you want to create a basic Network plot, you can use its *plot* method:

```
network.plot()
```

2.3 User Guide

In this guide, we describe the most import parts of OOPNET and provide some further examples for its usage.

Note: The folder with the examples (and corresponding models) is not included, when you install OOPNET. However, you can download the examples from the GitHub repository.

2.3.1 Network and Network Components

In OOPNET, everything is about the *Network* class. A network is the object-oriented representation of a hydraulic water distribution system model. It contains all the information stored in a EPANET input file and can be easily manipulated, accessed, simulated and plotted.

In this guide, we will take a look at the network class, the individual components stored in it and how to interact with them.

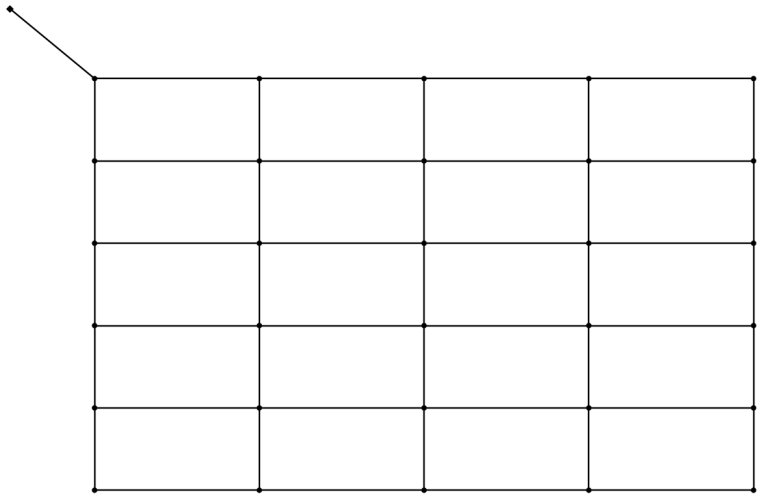
Creating a Network

The network acts as a container for all network components, settings etc. that you can find in an EPANET model. You can create a new, empty model:

```
import oopnet as on

blank_network = on.Network()
```

Alternatively, you can read an EPANET input file and create a network from it. OOPNET supports both EPANET 2.0 and 2.2 models. For instance, we can read a saved model - in this example the model by Poulakis et al. - using the network's *read()* method. We recommend using the `os` library for specifying the file path:

```
import os

import oopnet as on
filename = os.path.join('data', 'Poulakis.inp')
network = on.Network.read(filename)
```



Writing an Input File

Writing a network object to an EPANET input file is very easy. Just use the network's *write()* method:

```
network.write('new_model.inp')
```

Network Components

A network contains objects representing the different model components:

- ***Node***
  - *Junction*
  - *Tank*
  - *Reservoir*
- ***Link***
  - *Pipe*
  - *Pump*
  - *Valve* with dedicated subclass like *FCV* for the different valve types
- *Pattern*
- *Curve*
- *Rule*
- *Energy*

Getter Functions

To access the individual components in the model, OOPNET provides utility functions like *get_junction()* to get a single *Junction* by its ID, while functions like *get_pumps()* returns a list of all *Pump* objects. If you want a list of all IDs of objects of a certain type stored in a network, you can use functions like *get_node_ids()*. Take a look at the *getters* module for a complete list of all available getter functions.

As an example, we can iterate over all junctions in the network and print their demands and elevations in the console:

```
for j in on.get_junctions(network):
    print(j, j.demand, j.elevation)
```

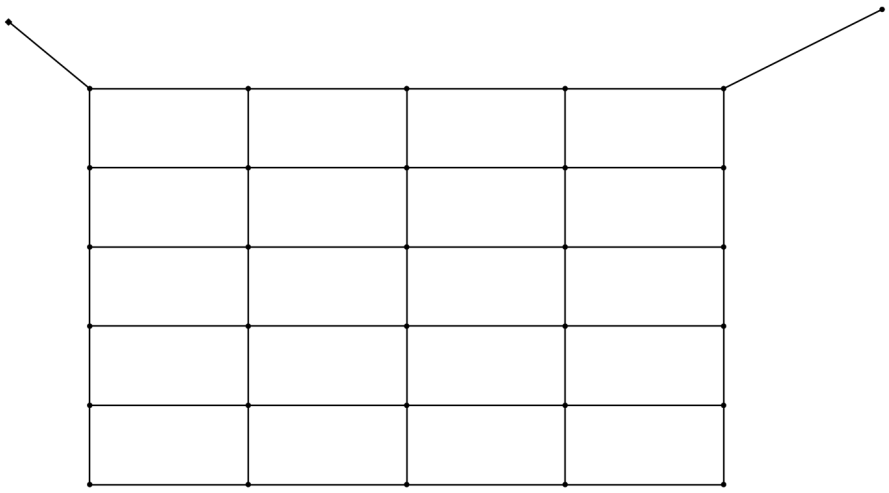
This results in an output like this:

```
Junction(id='J-02', comment=None, tag=None, xcoordinate=500.0, ycoordinate=7500.0,␣
↪elevation=0.0, initialquality=0.0, sourcequality=0.0, sourcetype=None, strength=0.0,␣
↪sourcepattern=None, emittercoefficient=0.0, demandpattern=None, demand=50.0) 50.0 0.0
Junction(id='J-03', comment=None, tag=None, xcoordinate=500.0, ycoordinate=7000.0,␣
↪elevation=0.0, initialquality=0.0, sourcequality=0.0, sourcetype=None, strength=0.0,␣
↪sourcepattern=None, emittercoefficient=0.0, demandpattern=None, demand=50.0) 50.0 0.0
Junction(id='J-04', comment=None, tag=None, xcoordinate=500.0, ycoordinate=6500.0,␣
↪elevation=0.0, initialquality=0.0, sourcequality=0.0, sourcetype=None, strength=0.0,␣
↪sourcepattern=None, emittercoefficient=0.0, demandpattern=None, demand=50.0) 50.0 0.0
Junction(id='J-05', comment=None, tag=None, xcoordinate=500.0, ycoordinate=6000.0,␣
↪elevation=0.0, initialquality=0.0, sourcequality=0.0, sourcetype=None, strength=0.0,␣
↪sourcepattern=None, emittercoefficient=0.0, demandpattern=None, demand=50.0) 50.0 0.0
Junction(id='J-06', comment=None, tag=None, xcoordinate=500.0, ycoordinate=5500.0,␣
↪elevation=0.0, initialquality=0.0, sourcequality=0.0, sourcetype=None, strength=0.0,␣
↪sourcepattern=None, emittercoefficient=0.0, demandpattern=None, demand=50.0) 50.0 0.0
Junction(id='J-07', comment=None, tag=None, xcoordinate=500.0, ycoordinate=5000.0,␣
↪elevation=0.0, initialquality=0.0, sourcequality=0.0, sourcetype=None, strength=0.0,␣
↪sourcepattern=None, emittercoefficient=0.0, demandpattern=None, demand=50.0) 50.0 0.0
...
```

Adding Components

If you want to add a new component, you can use the utility functions provided in the *adders* module. Here, we create a new junction and a new pipe to the system:

```
on.add_junction(network=network, junction=on.Junction(id='J-32', xcoordinate=5500,␣
↪ycoordinate=8000, demand=80))

on.add_pipe(network=network, pipe=on.Pipe(id='P-51', length=1000, diameter=400,␣
↪roughness=0.26,
                                          startnode=on.get_node(network, 'J-32'),␣
↪endnode=on.get_node(network, 'J-26')))
```



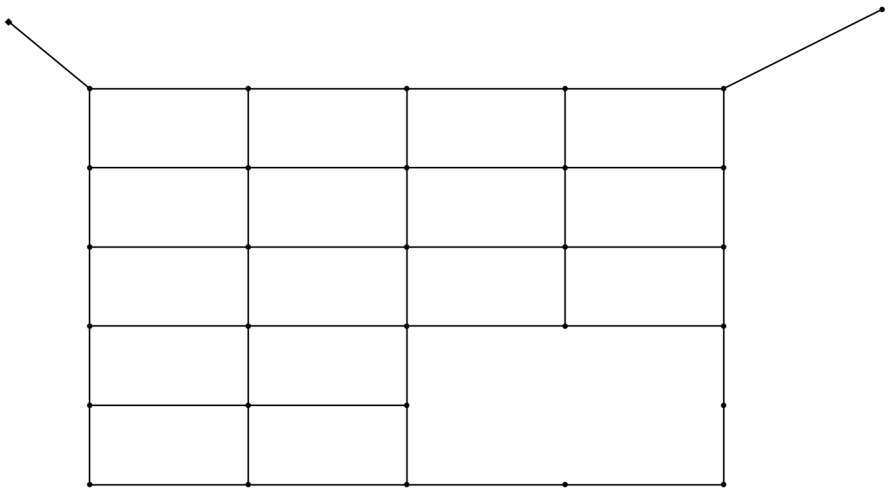
Removing Components

Now, we want to remove the junction with the ID `J-24` and all links connected to it. First, we get the junction object and use another utility function, *get_adjacent_links()*, to get a list of all links connected to the junction:

```
rjid = 'J-24'
rj = on.get_node(network, rjid)
neighbor_links = on.get_adjacent_links(network, rj)
```

Now, we can make use of the removing functions in the *removers* module. We first remove the links and then the junction from the model:

```
for neighbour in neighbor_links:
    on.remove_pipe(network=network, id=neighbour.id)

on.remove_junction(network=network, id=rjid)
```



Summary

```
import os

import oopnet as on

blank_network = on.Network()

filename = os.path.join('data', 'Poulakis.inp')
network = on.Network.read(filename)

network.write('new_model.inp')

for j in on.get_junctions(network):
    print(j, j.demand, j.elevation)

on.add_junction(network=network, junction=on.Junction(id='J-32', xcoordinate=5500,␣
↪ycoordinate=8000, demand=80))

on.add_pipe(network=network, pipe=on.Pipe(id='P-51', length=1000, diameter=400,␣
↪roughness=0.26,
                                          startnode=on.get_node(network, 'J-32'),␣
↪endnode=on.get_node(network, 'J-26')))

rjid = 'J-24'
rj = on.get_node(network, rjid)
neighbor_links = on.get_adjacent_links(network, rj)

for neighbour in neighbor_links:
```

(continues on next page)

(continued from previous page)

```
    on.remove_pipe(network=network, id=neighbour.id)

on.remove_junction(network=network, id=rjid)
```

2.3.2 Options

In this tutorial, we will take a look at the different modelling settings that users can modify.

Network objects have several attributes related to settings:

- *options* contains general model settings (e.g., units, headloss formula)
- *times* stores all time related settings (e.g., reporting time step, simulation duration)
- *report* configures general report settings (e.g., whether the simulation report should contain all the nodes)
- *reportparameter* is used to enable or disable the reporting on individual report parameters
- *reportprecision* allows for configuring the precision of the individual report parameters

Note: By default, OOPNET uses SI units and will convert all units in a model to SI units.

We start by importing all required packages and reading a model. We will again use the Poulakis model for demonstration.

```
import os

import oopnet as on

filename = os.path.join('data', 'Poulakis.inp')
network = on.Network.read(filename)
```

We can check what kind of demand model (demand driven or pressure driven) is used in the model:

```
print(network.options.demandmodel)
```

```
DDA
```

The model we loaded is configured for steady state analysis. We can check this by looking at the `duration` setting:

```
print(network.times.duration)
```

```
0:00:00
```

All times settings have `datetime.timedelta` as type. We will use this later in this guide, to set up an extended period simulation.

You can also enable or disable the reporting of certain parameters (pressure, flow, length, velocity, headloss etc). Here, we disable the reporting of the velocity and enable length reporting:

```
network.reportparameter.velocity = 'NO'
network.reportparameter.length = 'YES'
```

To change the reporting precision of a parameter, you can do something like this:

```
network.reportprecision.flow = 3
```

Summary

```
import os

import oopnet as on

filename = os.path.join('data', 'Poulakis.inp')
network = on.Network.read(filename)

print(network.options.demandmodel)
print(network.times.duration)

network.reportparameter.velocity = 'NO'
network.reportparameter.length = 'YES'

network.reportprecision.flow = 3
```

2.3.3 Network Simulation

OOPNET's network objects come with a simulation method *run()* that returns a *SimulationReport* object. This report object contains all simulation results as well as any errors raised by EPANET during the simulation. We will do both a steady state and an extended period simulation and take a look at handling simulations errors.

Steady State Analysis

For demonstrating a steady state analysis, we will once more use the Poulakis model:

```
import os
import oopnet as on

filename = os.path.join('data', 'Poulakis.inp')
network = on.Network.read(filename)
```

Let's run a simulation and look at the node and link results which are stored as `xarray.DataArray` objects:

```
report = network.run()
print(report.nodes)
```

```
<xarray.DataArray (id: 31, vars: 4)>
array([[    0.   ,    50.   ,    48.08 ,    48.076],
       [    0.   ,    50.   ,    36.26 ,    36.263],
       ...
       [    0.   ,    50.   ,     7.95 ,     7.952],
       [   52.   , -1500.   ,    52.   ,    -0.   ]])
Coordinates:
  * id       (id) object 'J-02' 'J-03' 'J-04' 'J-05' ... 'J-30' 'J-31' 'J-01'
  * vars     (vars) object 'Elevation' 'Demand' 'Head' 'Pressure'
```

```
print(report.links)
```

```
<xarray.DataArray (id: 50, vars: 8)>
array([[1.00000e+02, 6.00000e+02, 1.50000e+03, 5.31000e+00, 3.92400e+01,
        0.00000e+00, 0.00000e+00, 2.00000e-02],
       [1.00000e+03, 6.00000e+02, 8.18758e+02, 2.90000e+00, 1.18100e+01,
        0.00000e+00, 0.00000e+00, 2.00000e-02],
       ...

       [1.00000e+03, 3.00000e+02, 5.12740e+01, 7.30000e-01, 1.84000e+00,
        0.00000e+00, 0.00000e+00, 2.00000e-02],
       [1.00000e+03, 3.00000e+02, 2.68080e+01, 3.80000e-01, 5.30000e-01,
        0.00000e+00, 0.00000e+00, 2.00000e-02]])
Coordinates:
  * id       (id) object 'P-01' 'P-02' 'P-03' 'P-04' ... 'P-48' 'P-49' 'P-50'
  * vars     (vars) object 'Length' 'Diameter' 'Flow' ... 'Reaction' 'F-Factor'
```

There is another way of accessing the results, that is based on the data analysis and manipulation library `pandas`. You can get the simulation results as `pandas.Series` objects (for steady state analysis) or `pandas.DataFrame` (for extended period simulations) by accessing the different properties of the simulation report.

For instance, we can get the pressure data from the simulation report like this:

```
p = report.pressure
print(p)
```

```
id
J-01    -0.000
J-02    48.076
...
J-30     8.481
J-31     7.952
Name: Pressure (m), dtype: float64
```

The nice thing about pandas is, that it already comes with a lot of useful features. Here, we get some basic statistical data from the pressure results:

```
print(p.describe())
```

```
count    31.000000
mean     20.189516
std      10.322198
min      -0.000000
25%      12.222000
50%      17.344000
75%      26.360500
max      48.076000
Name: Pressure (m), dtype: float64
```

Extended Period Simulations

We will use another model for this example. Let's read the "Micropolis" model and check the simulation duration and the reporting time step:

```
eps_model_path = os.path.join('data', 'MICROPOLIS_v1.inp')
eps_network = on.Network.read(eps_model_path)
print(eps_network.times.duration)
print(eps_network.times.reporttimestep)
```

```
10 days, 0:00:00
1:00:00
```

Right now, the simulation duration is 10 days and we get a result in the report for every hour. Let's change the simulation duration to one day and the reporting time step to 10 minutes. For this, we use `datetime.timedelta`:

```
from datetime import timedelta
eps_network.times.duration = timedelta(days=1)
eps_network.times.reporttimestep = timedelta(minutes=10)
```

Next, we will display the node and link results:

```
eps_report = eps_network.run()
print(eps_report.nodes)
```

```
<xarray.DataArray (time: 145, id: 1577, vars: 4)>
array([[[ 3.1577e+02,  0.0000e+00,  3.5171e+02,  3.5940e+01],
        [ 3.1577e+02,  2.1000e-01,  3.5171e+02,  3.5940e+01],
        [ 3.1638e+02,  0.0000e+00,  3.5152e+02,  3.5140e+01],
        ...
        [ 2.8346e+02, -3.0510e+01,  2.8346e+02,  0.0000e+00],
        [ 3.1394e+02,  0.0000e+00,  3.1394e+02,  0.0000e+00],
        [ 3.1699e+02, -5.4430e+01,  3.4643e+02,  2.9440e+01]]])
Coordinates:
  * id       (id) object 'IN0' 'TN1' 'IN2' ... 'Aquifer' 'SurfaceResrvr' 'Tank'
  * vars     (vars) object 'Elevation' 'Demand' 'Head' 'Pressure'
  * time     (time) datetime64[ns] 2016-01-01 ... 2016-01-01T09:50:00
```

```
print(eps_report.links)
```

```
<xarray.DataArray (time: 145, id: 1619, vars: 8)>
array([[[4.635e+01, 1.016e+02, 2.100e-01, ..., 0.000e+00, 0.000e+00,
         3.000e-02],
        [2.879e+01, 1.016e+02, 3.000e-01, ..., 0.000e+00, 0.000e+00,
         5.000e-02],
        [2.285e+01, 1.016e+02, 3.000e-02, ..., 0.000e+00, 0.000e+00,
         2.000e-01],
        ...
        [0.000e+00, 1.200e+01, 1.300e-01, ..., 0.000e+00, 0.000e+00,
         0.000e+00],
        [0.000e+00, 1.200e+01, 3.600e-01, ..., 0.000e+00, 0.000e+00,
         0.000e+00],
        [0.000e+00, 1.200e+01, 1.440e+00, ..., 0.000e+00, 0.000e+00,
```

(continues on next page)

(continued from previous page)

```
        0.000e+00]]])
Coordinates:
  * id       (id) object 'SC0' 'SC1' 'SC2' 'SC3' ... 'V201' 'V202' 'V1028'
  * vars     (vars) object 'Length' 'Diameter' 'Flow' ... 'Reaction' 'F-Factor'
  * time     (time) datetime64[ns] 2016-01-01 ... 2016-01-01T09:50:00
```

As you can see, the DataArrays now have a new dimension `time`.

If we access the pressure property of the simulation result, we now get a `pandas.DataFrame`:

```
eps_p = eps_report.pressure
print(eps_p)
```

```
    id                     IN0     TN1     IN2  ...  Aquifer  SurfaceResrvr   Tank
time                                            ...
2016-01-01 00:00:00   35.94   35.94   35.14  ...      0.0            0.0  35.05
2016-01-01 00:10:00   35.91   35.91   35.11  ...      0.0            0.0  35.03
2016-01-01 00:20:00   35.88   35.88   35.09  ...      0.0            0.0  35.00
2016-01-01 00:30:00   35.86   35.86   35.06  ...      0.0            0.0  34.97
2016-01-01 00:40:00   35.83   35.83   35.03  ...      0.0            0.0  34.95
...                     ...     ...     ...  ...      ...            ...    ...
2016-01-01 23:20:00 -290.15 -290.15 -290.90  ...      0.0            0.0  27.43
2016-01-01 23:30:00 -290.15 -290.15 -290.90  ...      0.0            0.0  27.43
2016-01-01 23:40:00 -290.15 -290.15 -290.90  ...      0.0            0.0  27.43
2016-01-01 23:50:00 -290.15 -290.15 -290.90  ...      0.0            0.0  27.43
2016-01-02 00:00:00 -205.97 -205.97 -206.71  ...      0.0            0.0  27.43

[145 rows x 1577 columns]
```

As you can see, the index of this DataFrame are `datetime.datetime` objects, starting with 01-01-2016. You can pass a custom `starttdateime` to your simulation to change this. This can come in handy when comparing simulation results with measurement data from a certain day. Here, we set the start time of our simulation to 03-01-2022:

```
new_start = datetime(year=2022, month=3, day=1)
new_eps_report = eps_network.run(startdatetime=new_start)
print(new_eps_report.pressure)
```

```
id                     IN0     TN1     IN2  ...  Aquifer  SurfaceResrvr   Tank
time                                        ...
2022-03-01 00:00:00   35.94   35.94   35.14  ...      0.0            0.0  35.05
2022-03-01 00:10:00   35.91   35.91   35.11  ...      0.0            0.0  35.03
2022-03-01 00:20:00   35.88   35.88   35.09  ...      0.0            0.0  35.00
2022-03-01 00:30:00   35.86   35.86   35.06  ...      0.0            0.0  34.97
2022-03-01 00:40:00   35.83   35.83   35.03  ...      0.0            0.0  34.95
...                     ...     ...     ...  ...      ...            ...    ...
2022-03-01 23:20:00 -290.15 -290.15 -290.90  ...      0.0            0.0  27.43
2022-03-01 23:30:00 -290.15 -290.15 -290.90  ...      0.0            0.0  27.43
2022-03-01 23:40:00 -290.15 -290.15 -290.90  ...      0.0            0.0  27.43
2022-03-01 23:50:00 -290.15 -290.15 -290.90  ...      0.0            0.0  27.43
2022-03-02 00:00:00 -205.97 -205.97 -206.71  ...      0.0            0.0  27.43

[145 rows x 1577 columns]
```

Handling errors

Now, we will take a look at handling simulation errors. You can find a list of all simulation errors that EPANET provides in the EPANET docs. OOPNET implements dedicated Exceptions for all of these errors in the *simulation_errors* module.

We will again use the Poulakis model for this. To cause an error, we will add a Junction without connecting it to the rest of the model. This will lead to an error during the simulation.

```
on.add_junction(net, on.Junction(id='unconnected'))
```

If we were to run the simulation now, an *EPANETSimulationError* would be raised by OOPNET. These exceptions act as containers for all the errors raised by EPANET. To catch those exceptions, we can use a try-except clause:

```
try:
    rpt = net.run()
except on.EPANETSimulationError as e:
    print(e)
```

We first try to simulate the model and wait for the exception. When it is caught, we can print it, which results in this:

```
[UnconnectedNodeError('Error 233 - Error  Error 200: one or more errors in input file'),␣
↪InputDataError('Error 200 - one or more errors in input file')]
```

Since an *EPANETSimulationError* is a container, we can check for specific errors using its *check_contained_errors()* method. You can check either be providing a single Exception class

```
    if e.check_contained_errors(on.UnconnectedNodeError):
        print('Caught UnconnectedNodeError')
    if e.check_contained_errors(on.InputDataError):
        print('Caught InputDataError')
```

or using a list of Exception classes:

```
    if any(e.check_contained_errors([on.InputDataError, on.UnconnectedNodeError])):
        print('Caught UnconnectedNodeError and InputDataError')
```

If EPANET provides a more detailed error description (e.g., if a value in the model is invalid, EPANET tells us which part is faulty), OOPNET also outputs these details.

Let's test this by rereading the model and setting a Pipe's diameter to a negative value:

```
net = on.Network.read(filename)
p = on.get_pipe(net, 'P-01')
p.length = -100.0

try:
    rpt = net.run()
except on.EPANETSimulationError as e:
    print(e)

    if e.check_contained_errors(on.IllegalLinkPropertyError):
        print('Illegal property encountered')
```

Here, outputting `e` results in this:

```
[IllegalLinkPropertyError('Error 211 - illegal link property value -100.0 in [PIPES]␣
↪section: P-01 J-01 J-02 -100.0 600.0 0.26 0.0'), InputDataError('Error 200 - one or␣
↪more errors in input file')]
```

Further Examples

Time machine - Extended period simulations with OOPNET

In this example we will discuss performing extended period simulations with OOPNET.

As always, we have to import all required packages. This time, we will also need `datetime.timedelta` to specify some time related settings in the model.

```
import os
from datetime import timedelta

from matplotlib import pyplot as plt
import oopnet as on
```

Next, we read the "C-Town" model:

```
filename = os.path.join('data', 'C-town.inp')
net = on.Network.read(filename)
```

Now, we change the simulation duration to 6 hours and the reporting time step to 5 minutes. For this, we use `datetime.timedelta`:

```
net.times.duration = timedelta(hours=6)
net.times.reporttimestep = timedelta(minutes=5)
```

Wen can now simulate the model and interact with the `oopnet.SimulationReport`. When we access a specific property of this report, we now get a `pandas.DataFrame` object instead of a `pandas.Series` like we get when doing steady-state simulations:

```
rpt = net.run()
print(rpt.pressure)
```

```
id                          J511   J411   J414  ...    T5    T2    T4
time                                             ...
2016-01-01 00:00:00        28.83  64.42  39.00  ...  2.25  2.95  2.35
2016-01-01 00:05:00        28.71  64.40  38.97  ...  2.20  2.95  2.23
2016-01-01 00:10:00        28.59  64.37  38.95  ...  2.15  2.95  2.12
2016-01-01 00:15:00        28.48  64.35  38.93  ...  2.10  2.95  2.00
2016-01-01 00:20:00        28.36  64.32  38.90  ...  2.05  2.95  1.88
...                          ...    ...    ...  ...   ...   ...   ...
2016-01-01 05:40:00 -7520000.00  63.65  38.24  ... -0.00  2.31 -0.00
2016-01-01 05:45:00 -7530000.00  63.63  38.22  ... -0.00  2.30 -0.00
2016-01-01 05:50:00 -7530000.00  63.62  38.20  ... -0.00  2.29 -0.00
2016-01-01 05:55:00 -7530000.00  63.60  38.18  ... -0.00  2.28 -0.00
2016-01-01 06:00:00 -9960000.00  63.29  37.88  ... -0.00  2.27 -0.00

[73 rows x 396 columns]
```

Using pandas, we can also get some basic statistics:

```
print(rpt.pressure.describe())
```

```
id             J511       J411       J414  ...         T5         T2         T4
count  7.300000e+01  73.000000  73.000000  ...  73.000000  73.000000  73.000000
mean  -5.704922e+06  64.220137  38.803562  ...   0.863288   2.698767   0.359315
std    4.155719e+06   0.268388   0.266418  ...   0.696533   0.237843   0.659588
min   -1.090000e+07  63.290000  37.880000  ...  -0.000000   2.270000  -0.000000
25%   -8.130000e+06  64.170000  38.760000  ...   0.240000   2.490000   0.000000
50%   -7.530000e+06  64.300000  38.880000  ...   0.730000   2.720000   0.000000
75%    2.732000e+01  64.390000  38.970000  ...   1.420000   2.950000   0.430000
max    2.883000e+01  64.560000  39.150000  ...   2.250000   2.950000   2.350000

[8 rows x 396 columns]
```

Now, we want to create a plot using this data. In the plot, we want to show the flow through pipe *P1000* on one subplot, while showing the pressure at junctions *J10* and *J1058* on a second subplot. For this, we first have to create a new matplotlib `matplotlib.Figure`.

```
fig = plt.figure(1)
```

We then create an `matplotlib.Axes` object for the first plot, plot the data, set labels for the axes and add a legend.

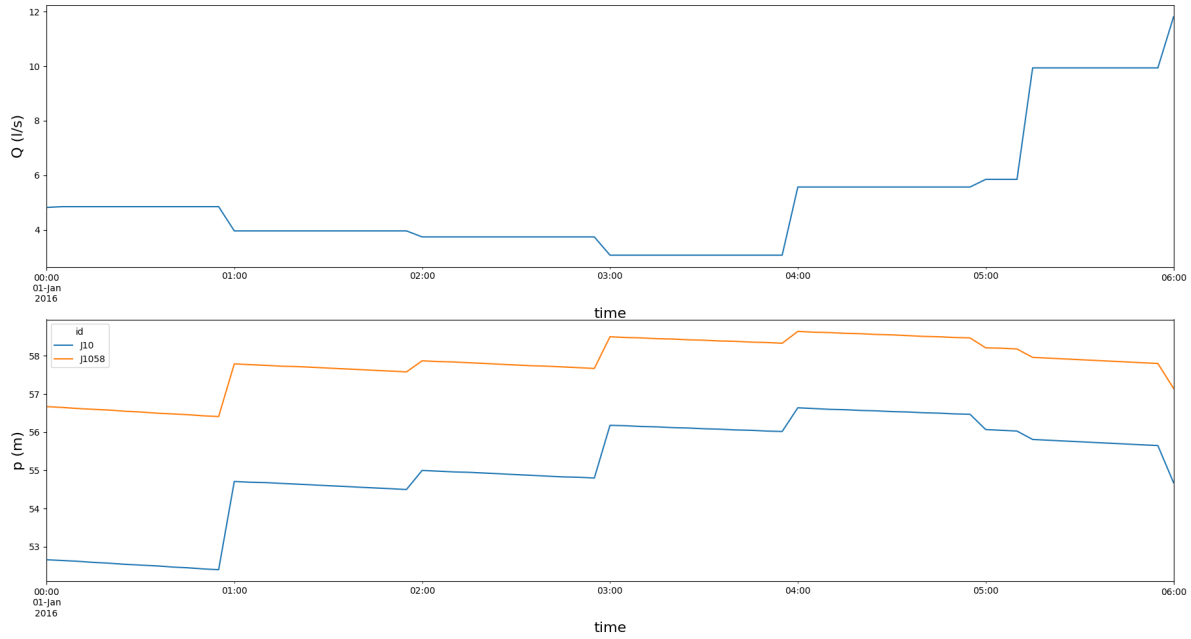
```
rpt = net.run()
print(rpt.pressure)
print(rpt.pressure.describe())
```

Next, we do the same with the pressure data. Note, that we don't have to create a legend because pandas does it automatically when plotting several lines.

```
ax2 = fig.add_subplot(212)
rpt.pressure[['J10', 'J1058']].plot(ax=ax2)
plt.xlabel('time', fontsize=16)
plt.ylabel('p (m)', fontsize=16)
```

Finally, we can show our plots:

```
plt.show()
```



Summary

```
import os
from datetime import timedelta

from matplotlib import pyplot as plt
import oopnet as on

filename = os.path.join('data', 'C-town.inp')
net = on.Network.read(filename)

net.times.duration = timedelta(hours=6)
net.times.reporttimestep = timedelta(minutes=5)

rpt = net.run()
print(rpt.pressure)
print(rpt.pressure.describe())

fig = plt.figure(1)

ax1 = fig.add_subplot(211)
rpt.flow['P1000'].plot(ax=ax1)
plt.xlabel('time', fontsize=16)
plt.ylabel('Q (l/s)', fontsize=16)
plt.legend()

ax2 = fig.add_subplot(212)
rpt.pressure[['J10', 'J1058']].plot(ax=ax2)
plt.xlabel('time', fontsize=16)
plt.ylabel('p (m)', fontsize=16)
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

MC, make some noise!

Example of a simple Monte Carlo simulation with OOPNET

Summary

```
import os

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import oopnet as on

filename = os.path.join('data', 'Poulakis.inp')

net = on.Network.read(filename)
net.reportprecision.flow = 3
net.reportprecision.pressure = 3
mcruns = 1000
p = []

for _ in range(mcruns):
    cnet = on.Copy(net)
    for j in on.get_junctions(cnet):
        j.demand += np.random.normal(0.0, 1.0)
    rpt = net.run()
    p.append(rpt.pressure)

p = pd.DataFrame(p, index=list(range(len(p))))
print(p)

pmean = p.mean()
print(pmean)

psub = p.sub(pmean, axis=1)

x = np.linspace(-1.5, 1.5, 40)
psub[['J-03', 'J-31']].hist(bins=x, layout=(2, 1))
plt.show()
```

All simulations

```
id       J-02      J-03      J-04      J-05      J-06      J-07      J-08  \
0    48.05263  36.20626  32.09023  25.57547  22.36982  16.98668  33.86014
1    48.05536  36.16504  32.02279  25.50081  22.22193  16.79289  33.85170
2    48.07595  36.27081  32.13451  25.59465  22.37663  16.98995  33.95707
3    48.03927  36.10299  31.94703  25.32881  22.07879  16.63252  33.78149
```

(continues on next page)

(continued from previous page)

```
4    48.06370  36.20726  32.07417  25.50348  22.25249  16.83488  33.88490
5    48.04335  36.11147  31.94106  25.32247  22.05440  16.66107  33.78643
6    48.06145  36.20734  32.05158  25.44507  22.11934  16.55697  33.89405
7    48.06710  36.23970  32.10273  25.54541  22.28316  16.94171  33.92965
8    48.07666  36.26645  32.13058  25.60568  22.37519  16.91885  33.97862
9    48.10617  36.38207  32.28330  25.78794  22.53194  17.20967  34.08455
10   48.07971  36.27354  32.13645  25.55922  22.32138  17.01659  33.95674
11   48.06863  36.24666  32.11699  25.57224  22.33344  16.91153  33.91496
12   48.10598  36.39598  32.32642  25.85967  22.66669  17.25910  34.09404
13   48.03654  36.09021  31.90990  25.30857  22.02899  16.54935  33.74682
14   48.12745  36.48903  32.45071  26.06793  22.92767  17.58916  34.21410
15   48.08218  36.28799  32.15750  25.62647  22.42363  17.05583  33.97613
16   48.08147  36.29768  32.19301  25.70372  22.44553  17.05880  33.96402
17   48.11480  36.40691  32.30753  25.82820  22.59041  17.17222  34.12262
18   48.07418  36.28747  32.16848  25.64538  22.41353  17.03652  33.95995
19   48.07574  36.25684  32.11346  25.55870  22.28025  16.78620  33.96915
20   48.09552  36.35032  32.23259  25.78288  22.61438  17.29116  34.03664
21   48.09579  36.33810  32.24323  25.77134  22.53391  17.18315  34.02614
22   48.05028  36.14321  31.97496  25.38715  22.10423  16.60662  33.83548
23   48.06408  36.19558  32.06313  25.54452  22.32975  16.98651  33.89101
24   48.09536  36.35620  32.26151  25.79544  22.56889  17.35523  34.05053
25   48.05047  36.13840  31.98923  25.41736  22.17638  16.76101  33.80855
26   48.08620  36.29636  32.19616  25.68926  22.46867  17.09886  33.99624
27   48.04121  36.13400  31.97240  25.35273  22.07393  16.68850  33.76446
28   48.06730  36.19647  32.04176  25.49324  22.24962  16.69672  33.87530
29   48.02549  36.06430  31.88877  25.26326  22.05229  16.67207  33.70544
..        ...       ...       ...       ...       ...       ...       ...
970  48.07579  36.23956  32.11909  25.57333  22.32073  16.75916  33.90469
971  48.07568  36.24693  32.09726  25.54302  22.28023  16.90422  33.94631
972  48.06403  36.21048  32.05608  25.51338  22.25726  16.85144  33.92192
973  48.06808  36.21251  32.08449  25.49238  22.20456  16.84911  33.89600
974  48.05397  36.16566  32.02773  25.45019  22.21239  16.94181  33.82619
975  48.06212  36.21580  32.06981  25.51364  22.24534  16.79800  33.90304
976  48.07601  36.27822  32.17314  25.69534  22.48069  17.17877  33.93767
977  48.10380  36.39755  32.31929  25.88289  22.68892  17.31169  34.11094
978  48.11771  36.45352  32.37920  25.97898  22.79887  17.42808  34.19733
979  48.12217  36.43320  32.35101  25.90001  22.69481  17.38090  34.13457
980  48.11197  36.37006  32.23886  25.71607  22.46581  17.04320  34.07144
981  48.02599  36.05515  31.86634  25.20280  21.90401  16.40769  33.70277
982  48.07099  36.23217  32.10479  25.61222  22.43352  17.06845  33.91563
983  48.02738  36.10115  31.96286  25.37524  22.13148  16.67139  33.72991
984  48.09159  36.32709  32.21476  25.70711  22.45411  16.97367  34.03801
985  48.09814  36.35109  32.26931  25.80882  22.59034  17.26244  34.05997
986  48.03797  36.11663  31.95249  25.39286  22.14025  16.67498  33.77147
987  48.01366  36.02291  31.85334  25.22548  21.95377  16.58075  33.62440
988  48.06314  36.19137  32.05089  25.52244  22.27832  16.92956  33.85867
989  48.05946  36.19695  32.05098  25.49600  22.22557  16.70917  33.88520
990  48.06960  36.21960  32.09010  25.56104  22.32839  16.90228  33.88485
991  48.08207  36.27187  32.17778  25.64262  22.40542  17.03619  33.93113
992  48.06675  36.21551  32.08350  25.55669  22.35908  17.06221  33.88421
993  48.06462  36.21929  32.07928  25.56847  22.31351  16.92476  33.89386
994  48.06660  36.20448  32.04678  25.46493  22.20528  16.83645  33.89636
```

(continues on next page)

(continued from previous page)

```
995  48.09550  36.32858  32.23270  25.69344  22.44268  17.07356  34.03864
996  48.02863  36.08414  31.90683  25.30478  21.98253  16.47938  33.75841
997  48.09834  36.34233  32.22326  25.69591  22.46196  17.03092  34.00829
998  48.10823  36.41499  32.32069  25.86593  22.68708  17.38987  34.14548
999  48.12778  36.48055  32.41754  25.97745  22.79677  17.39008  34.19878

id       J-09      J-10      J-11  ...       J-23      J-24     J-25  \
0    33.14309  28.57448  25.32328  ...   11.64737   9.20805  8.54955
1    33.13254  28.54361  25.27065  ...   11.82155   9.41415  8.72061
2    33.23581  28.65408  25.35044  ...   11.84946   9.42276  8.71845
3    33.05351  28.41055  25.06907  ...   11.53692   9.05867  8.32324
4    33.16064  28.56866  25.23994  ...   11.48818   9.00868  8.34509
5    33.05571  28.42649  25.08778  ...   11.59442   9.15463  8.45303
6    33.17048  28.57503  25.21403  ...   11.42540   8.94835  8.24148
7    33.21638  28.64712  25.32061  ...   12.09856   9.64923  8.92652
8    33.25213  28.66316  25.35448  ...   11.83829   9.46432  8.79167
9    33.37270  28.85130  25.55383  ...   12.06136   9.62416  8.94667
10   33.23439  28.66747  25.32994  ...   11.76787   9.39475  8.69047
11   33.19735  28.63541  25.33759  ...   11.61997   9.17759  8.47110
12   33.38242  28.84867  25.61284  ...   11.96496   9.66041  8.97429
13   33.01753  28.39240  25.07344  ...   11.57807   9.12906  8.42255
14   33.50227  29.02380  25.82987  ...   12.54798  10.13286  9.41448
15   33.25582  28.66578  25.38712  ...   11.86873   9.41032  8.71232
16   33.25035  28.69136  25.46610  ...   11.87598   9.44342  8.76123
17   33.40392  28.87685  25.59793  ...   12.35576   9.92918  9.23191
18   33.24575  28.69760  25.41076  ...   12.04371   9.64789  8.96806
19   33.24142  28.64711  25.33449  ...   11.90205   9.33406  8.63396
20   33.32619  28.78677  25.54384  ...   12.22296   9.84364  9.14210
21   33.30582  28.78679  25.53559  ...   12.04129   9.69533  9.04111
22   33.10546  28.49492  25.15934  ...   11.82118   9.35287  8.65717
23   33.17092  28.57731  25.29521  ...   11.88287   9.51844  8.85868
24   33.33958  28.80444  25.55190  ...   12.20332   9.82014  9.14480
25   33.08853  28.50545  25.18151  ...   11.49176   9.06489  8.32891
26   33.28095  28.74529  25.44574  ...   12.07874   9.72389  9.06365
27   33.04796  28.43761  25.10983  ...   11.45326   8.98505  8.36786
28   33.15872  28.56129  25.25121  ...   11.72401   9.18975  8.47434
29   32.97587  28.36515  25.00962  ...   11.34613   8.89754  8.15702
..        ...       ...       ...  ...        ...       ...      ...
970  33.18879  28.62428  25.33602  ...   11.81226   9.39272  8.65831
971  33.22563  28.64328  25.31116  ...   11.81554   9.40539  8.71420
972  33.19715  28.61410  25.28117  ...   11.66684   9.15503  8.44030
973  33.18218  28.60455  25.27159  ...   11.89090   9.43920  8.78492
974  33.11131  28.50665  25.20599  ...   11.65467   9.23945  8.53006
975  33.18261  28.60628  25.27737  ...   11.81141   9.42321  8.75554
976  33.22698  28.72639  25.44876  ...   12.16211   9.81266  9.16283
977  33.39751  28.87844  25.63053  ...   12.45737  10.05442  9.42039
978  33.47413  28.96060  25.73393  ...   12.34445   9.98866  9.29549
979  33.42332  28.89957  25.66801  ...   12.31217   9.92048  9.27591
980  33.35538  28.76871  25.48128  ...   11.88887   9.42383  8.75700
981  32.98268  28.33866  24.96757  ...   11.38320   8.92765  8.23181
982  33.18754  28.63535  25.35621  ...   11.70856   9.30805  8.64011
983  33.00915  28.42150  25.12285  ...   11.43427   8.93933  8.22755
```

(continues on next page)

(continued from previous page)

```
984  33.31590  28.75856  25.47295  ...   12.11868   9.71156  9.02241
985  33.34555  28.83504  25.57218  ...   12.21248   9.78691  9.07942
986  33.05609  28.45790  25.15143  ...   11.46326   9.00989  8.27913
987  32.90555  28.30417  24.98887  ...   11.49461   9.07458  8.37240
988  33.14406  28.56149  25.26867  ...   11.70384   9.28249  8.56323
989  33.16803  28.59267  25.27841  ...   11.73819   9.28232  8.57675
990  33.17555  28.62538  25.31975  ...   11.64945   9.21427  8.50140
991  33.21550  28.68331  25.40696  ...   11.95806   9.53935  8.85231
992  33.16695  28.60622  25.31792  ...   11.81663   9.39616  8.72345
993  33.17517  28.61468  25.33008  ...   11.69255   9.21436  8.50655
994  33.17433  28.58000  25.23847  ...   11.63271   9.12381  8.44343
995  33.31799  28.76972  25.46102  ...   12.08674   9.64129  8.94362
996  33.03205  28.41608  25.07021  ...   11.52320   8.95318  8.26925
997  33.28922  28.72377  25.43878  ...   11.64047   9.14227  8.40091
998  33.42528  28.91228  25.63077  ...   12.11646   9.70493  9.02932
999  33.49283  28.98942  25.71894  ...   12.31651   9.90564  9.25407

id       J-26      J-27      J-28      J-29     J-30     J-31  J-01
0    19.97942  16.95705  13.18969  10.02094  8.20003  7.67507    -0
1    20.14923  17.19084  13.46871  10.27905  8.43603  7.91152    -0
2    20.30230  17.29921  13.55919  10.33163  8.49349  7.98729    -0
3    20.09658  17.08447  13.24859  10.00572  8.08431  7.52487    -0
4    20.18385  17.04738  13.09673   9.84796  8.02043  7.50068    -0
5    20.05120  17.05044  13.26686  10.02552  8.19719  7.67153    -0
6    20.15705  17.13006  13.19764   9.85542  7.90013  7.40277    -0
7    20.31562  17.40267  13.70813  10.54702  8.69221  8.17527    -0
8    20.38574  17.39032  13.52757  10.26389  8.46984  7.98296    -0
9    20.45914  17.44146  13.75905  10.52069  8.70701  8.17143    -0
10   20.30441  17.34626  13.49866  10.21959  8.44183  7.91482    -0
11   20.29622  17.26673  13.37562  10.03857  8.18465  7.66012    -0
12   20.41252  17.38129  13.55883  10.36507  8.64277  8.12538    -0
13   19.93433  16.92132  13.17464   9.95668  8.14346  7.61011    -0
14   20.85484  17.90489  14.17237  10.98870  9.17334  8.61735    -0
15   20.29648  17.36255  13.56543  10.32651  8.44971  7.93362    -0
16   20.15505  17.20065  13.47107  10.31390  8.50364  7.99981    -0
17   20.47946  17.60337  13.96268  10.82617  8.98621  8.45989    -0
18   20.55746  17.52939  13.79314  10.57892  8.76665  8.22393    -0
19   20.43836  17.46011  13.65347  10.34997  8.41807  7.84739    -0
20   20.45984  17.51266  13.80446  10.65113  8.86326  8.33016    -0
21   20.42750  17.47705  13.67429  10.51805  8.78788  8.27465    -0
22   20.35529  17.34912  13.62486  10.32226  8.42120  7.90151    -0
23   20.24014  17.28358  13.55478  10.37765  8.53280  8.03115    -0
24   20.58209  17.63167  13.93109  10.68314  8.89247  8.37444    -0
25   20.11345  17.10120  13.27150   9.92937  8.06558  7.56087    -0
26   20.45310  17.42400  13.63974  10.51719  8.77967  8.28537    -0
27   19.91498  16.88747  13.07271   9.86220  8.05304  7.54676    -0
28   20.07524  17.12048  13.40796  10.12951  8.29335  7.70409    -0
29   19.98756  16.99474  13.07850   9.81304  7.94361  7.35528    -0
..        ...       ...       ...       ...      ...      ...   ...
970  20.26238  17.27621  13.45587  10.23681  8.37709  7.86934    -0
971  20.29805  17.30681  13.46209  10.23867  8.39157  7.88897    -0
972  20.26253  17.21900  13.42646  10.11736  8.14116  7.54671    -0
```

(continues on next page)

(continued from previous page)

```
973  20.23767  17.28996  13.65784  10.44850  8.52700  7.97129    -0
974  19.86391  16.84772  13.22378  10.04968  8.23651  7.71119    -0
975  20.21059  17.16162  13.43661  10.28934  8.46271  7.95376    -0
976  20.34590  17.40224  13.73516  10.62687  8.84939  8.35817    -0
977  20.72603  17.80926  14.21146  10.98996  9.15238  8.64899    -0
978  20.73532  17.75473  13.94221  10.70746  8.98678  8.47529    -0
979  20.57038  17.61937  13.94876  10.77348  8.93108  8.43617    -0
980  20.53898  17.52637  13.67651  10.29599  8.40124  7.87621    -0
981  19.89542  16.86917  13.11078   9.84585  7.98138  7.45399    -0
982  20.15851  17.11550  13.31865  10.12449  8.34390  7.81171    -0
983  19.78379  16.75719  13.05892   9.84000  7.97342  7.42735    -0
984  20.43250  17.54058  13.89122  10.58141  8.71999  8.18185    -0
985  20.54068  17.63651  13.94413  10.71919  8.84834  8.33123    -0
986  19.90912  16.90398  13.12616   9.88536  8.04341  7.47972    -0
987  19.82273  16.86367  13.11191   9.89710  8.04896  7.52300    -0
988  20.19112  17.16837  13.34088  10.12805  8.24555  7.71251    -0
989  20.07142  17.06293  13.29113  10.14353  8.30086  7.78775    -0
990  20.19978  17.20276  13.37569  10.12369  8.26043  7.73646    -0
991  20.30854  17.31426  13.52227  10.36112  8.57181  8.06349    -0
992  20.13706  17.09434  13.42495  10.26147  8.46915  7.97820    -0
993  20.24476  17.18448  13.41660  10.13289  8.21888  7.70470    -0
994  20.15512  17.12984  13.33682  10.11737  8.18892  7.65090    -0
995  20.49251  17.49895  13.72787  10.46304  8.65257  8.13649    -0
996  20.10684  17.06703  13.27101   9.94482  8.05414  7.52158    -0
997  20.28470  17.17357  13.25146  10.01272  8.09447  7.55685    -0
998  20.74576  17.73716  13.85601  10.57541  8.71258  8.19670    -0
999  20.73716  17.81979  14.02732  10.78508  8.97153  8.45809    -0

[1000 rows x 31 columns]
```

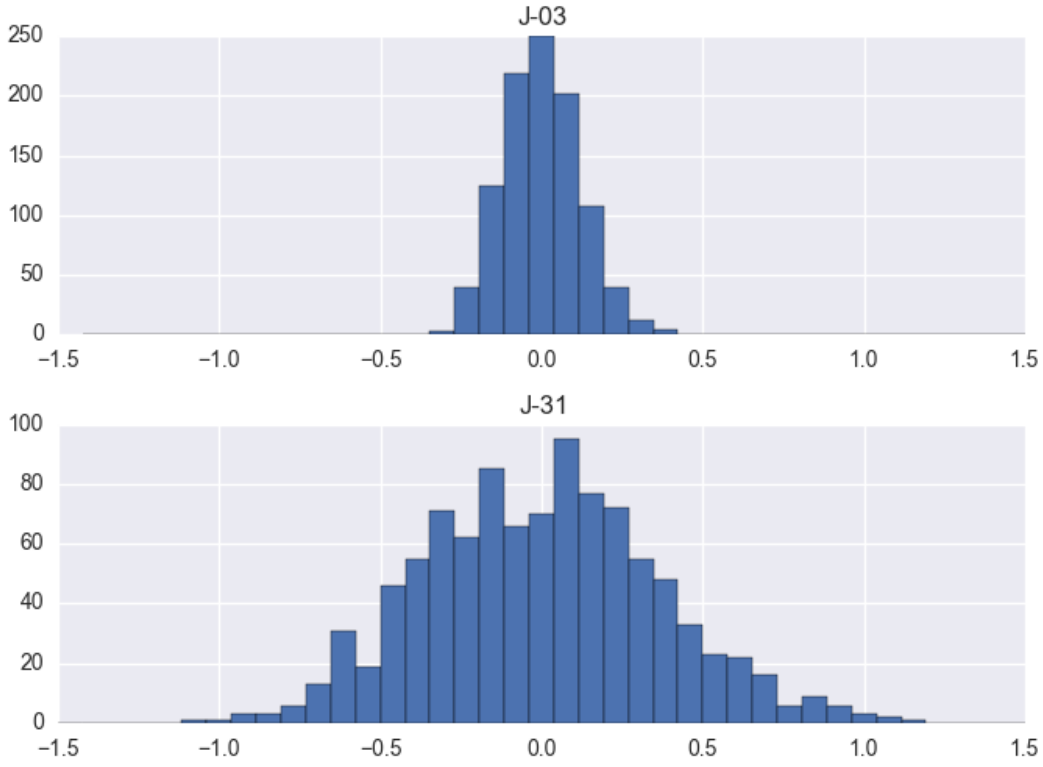
Mean of pressure

```
id
J-02    48.076265
J-03    36.264631
J-04    32.139784
J-05    25.609925
J-06    22.377730
J-07    16.983732
J-08    33.945294
J-09    33.228194
J-10    28.665426
J-11    25.371254
J-12    17.351544
J-13    14.796450
J-14    28.599816
J-15    26.925095
J-16    25.798810
J-17    17.050349
J-18    12.578846
J-19    11.301319
J-20    22.549633
```

(continues on next page)

(continued from previous page)

```
J-21    22.021013
J-22    16.323417
J-23    11.874678
J-24     9.448641
J-25     8.758459
J-26    20.303097
J-27    17.313045
J-28    13.548864
J-29    10.324250
J-30     8.486396
J-31     7.957737
J-01     0.000000
dtype: float64
```



MC in stereo!

Make the Monte Carlo simulations run in parallel (multiprocessing 17 seconds and scoop 21 seconds instead of 40 seconds)

Scoop

```
import os

from scoop import futures
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import oopnet as on


def roll_the_dice(network: on.Network) -> pd.Series:
    cnet = on.Copy(network)
    for j in on.get_junctions(cnet):
        j.demand += np.random.normal(0.0, 1.0)
    rpt = cnet.run()
    return rpt.pressure


if __name__ == '__main__':
    filename = os.path.join('data', 'Poulakis.inp')

    net = on.Network.read(filename)
    net.reportprecision.flow = 3
    net.reportprecision.pressure = 3
    mcruns = 1_000
    p = list(futures.map(roll_the_dice, [net] * mcruns))

    p = pd.DataFrame(p, index=list(range(len(p))))
    print(p)

    p_mean = p.mean()
    print(p_mean)

    p_sub = p.sub(p_mean, axis=1)

    x = np.linspace(-1.5, 1.5, 40)
    p_sub[['J-03', 'J-31']].hist(bins=x, layout=(2, 1))
    plt.show()
```

Multiprocessing

```
import os
from multiprocessing import Pool

import numpy as np
import pandas as pd
import oopnet as on
from matplotlib import pyplot as plt


def roll_the_dice(network: on.Network) -> pd.Series:
    cnet = on.Copy(network)
    for j in on.get_junctions(cnet):
        j.demand += np.random.normal(0.0, 1.0)
    rpt = cnet.run()
    return rpt.pressure


if __name__ == '__main__':
    filename = os.path.join('data', 'Poulakis.inp')

    net = on.Network.read(filename)
    mcruns = 1_000
    networks = [net] * mcruns

    p = Pool().map(roll_the_dice, networks)

    p = pd.DataFrame(p, index=list(range(len(p))))
    print(p)

    p_mean = p.mean()
    print(p_mean)

    p_sub = p.sub(p_mean, axis=1)

    x = np.linspace(-1.5, 1.5, 40)
    p_sub[['J-03', 'J-31']].hist(bins=x, layout=(2, 1))
    plt.show()
```

Summary

Simulation Example

```
import os
from datetime import timedelta
from datetime import datetime
import oopnet as on

filename = os.path.join('data', 'Poulakis.inp')
network = on.Network.read(filename)
```

(continues on next page)

(continued from previous page)

```
report = network.run()
print(report.nodes)
print(report.links)

p = report.pressure
print(p)
print(p.describe())

eps_model_path = os.path.join('data', 'MICROPOLIS_v1.inp')
eps_network = on.Network.read(eps_model_path)
print(eps_network.times.duration)
print(eps_network.times.reporttimestep)

eps_network.times.duration = timedelta(days=1)
eps_network.times.reporttimestep = timedelta(minutes=10)

eps_report = eps_network.run()
print(eps_report.nodes)
print(eps_report.links)

eps_p = eps_report.pressure
print(eps_p)

new_start = datetime(year=2022, month=3, day=1)
new_eps_report = eps_network.run(startdatetime=new_start)
print(new_eps_report.pressure)
```

Error Handling Example

```
import os

import oopnet as on

filename = os.path.join('data', 'Poulakis.inp')

net = on.Network.read(filename)

on.add_junction(net, on.Junction(id='unconnected'))

try:
    rpt = net.run()
except on.EPANETSimulationError as e:
    print(e)

    if e.check_contained_errors(on.UnconnectedNodeError):
        print('Caught UnconnectedNodeError')
    if e.check_contained_errors(on.InputDataError):
        print('Caught InputDataError')
    if any(e.check_contained_errors([on.InputDataError, on.UnconnectedNodeError])):
```

(continues on next page)

(continued from previous page)

```
        print('Caught UnconnectedNodeError and InputDataError')

net = on.Network.read(filename)
p = on.get_pipe(net, 'P-01')
p.length = -100.0

try:
    rpt = net.run()
except on.EPANETSimulationError as e:
    print(e)

    if e.check_contained_errors(on.IllegalLinkPropertyError):
        print('Illegal property encountered')
```

2.3.4 Plotting

Now, we will do some basic plots with OOPNET. OOPNET comes with several plotting options:

- static plotting based on `matplotlib`
- animations based on `matplotlib`
- interactive plots based on `bokeh`

First, we will create a static matplotlib plot.

Matplotlib

Static Plotting

First, we declare our imports and read the "C-Town" model:

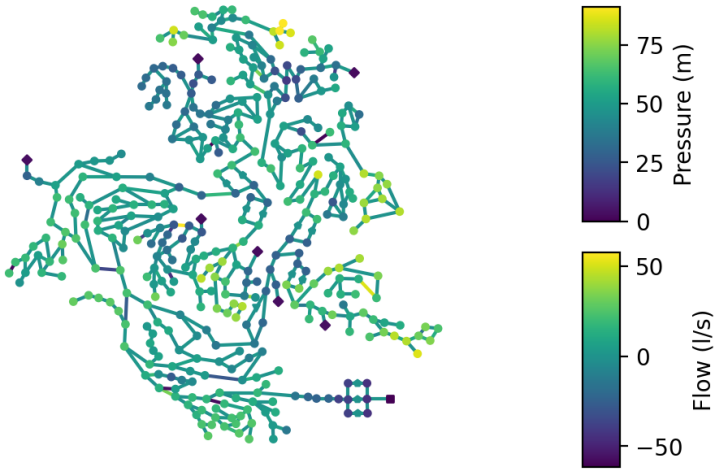
```
import os

from matplotlib import pyplot as plt
import oopnet as on

filename = os.path.join('data', 'C-town.inp')

net = on.Network.read(filename)
```

We can now plot the model as-is:

```
net.plot()
```

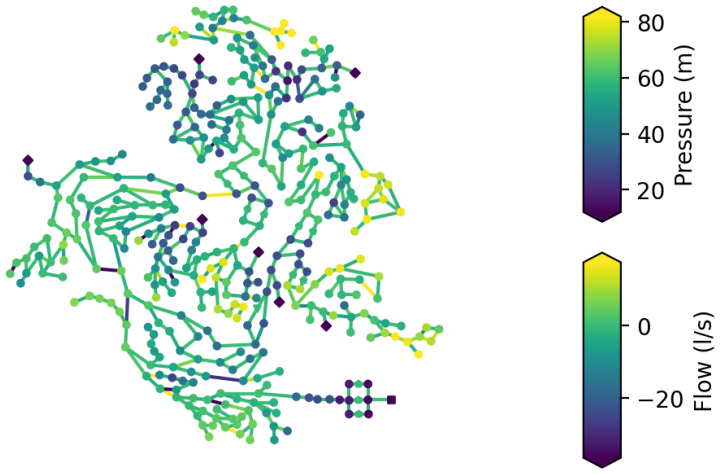
We can also run a simulation and plot the pressure and flow (or any other `pandas.Series` with link or node IDs as index) by passing them to the `nodes` and `links` arguments.

```
rpt = net.run()
p = rpt.pressure
f = rpt.flow

net.plot(nodes=p, links=f)
```



There is also the possibility to limit the color bar to values between the 2nd and 98th percentile using the `robust` parameter. If it is set to `True`, the colors in the plot will be more finely graduated because the minima and maxima values will not be used for calculating the value range of the color bar.

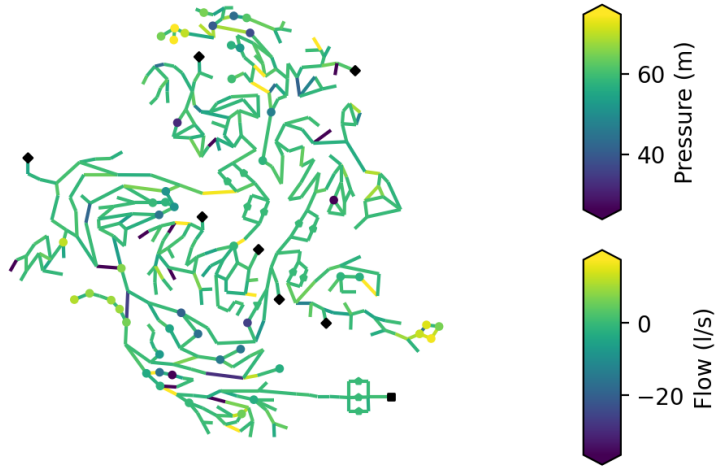
```
net.plot(nodes=p, links=f, robust=True)
```



But what if you don't want to plot all junctions? For instance, if you calculate the difference in pressure between measurement and simulation data, you might not have a value to plot for every junction. By default, if a junction doesn't have a value assigned, the junction will be plotted in black.

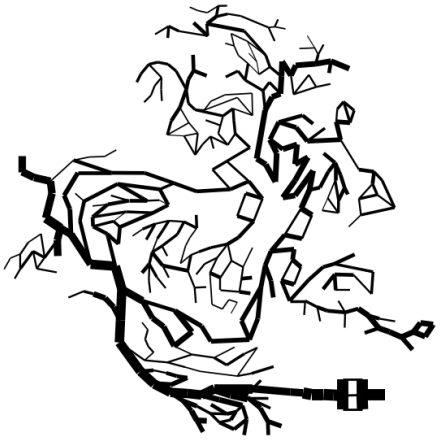
Alternatively, you can prevent OOPNET from plotting junctions without a value assigned and therefore simplify the plot. In this example we only plot the first 50 pressure values from the report by reducing the pressure dataset and passing the `truncate_nodes` argument to the plotting function:

```
p_reduced = p.iloc[:50]
net.plot(nodes=p_reduced, links=f, robust=True, truncate_nodes=True)
```



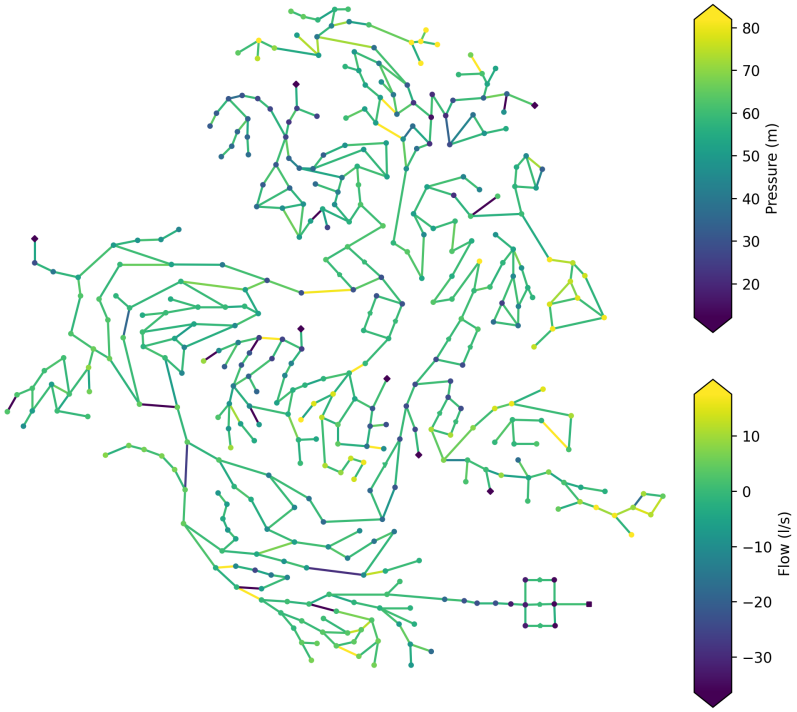
You can also pass values for the link width in the plot. Here, we use this to show the diameters in the network while hiding all the nodes.

```
diameters = on.get_diameter(net)
net.plot(linkwidth=diameters, markersize=0)
```



The figure can also be customized by using the `ax` argument. For instance, here we create a plot with a certain size and DPI count:

```
fig, ax = plt.subplots(figsize=(12, 9), dpi=250)
net.plot(ax=ax, nodes=p, links=f, robust=True)
```



Don't forget to show the plots:

```
plt.show()
```

Plotting Animations

In this example we want to create a matplotlib animation of the model where we plot the flow and pressure results from an extended period simulation.

For this, we first have to import the packages we require:

- `os` for specifying the path to the EPANET input file
- `matplotlib.pyplot` for creating an animation of a certain size
- `matplotlib.animation.PillowWriter` for writing the animation to a file
- and of course `oopnet` itself

In this example we read a part of the *L-Town* network (Area C) with slight modifications. This model already comes with included patterns and can be used for extended period simulations.

```
import os

from matplotlib import pyplot as plt
```

(continues on next page)

(continued from previous page)

```
from matplotlib.animation import PillowWriter

import oopnet as on

filename = os.path.join('data', 'L-TOWN_AreaC.inp')
net = on.Network.read(filename)
```

Then, we can simulate the model with its `.run()` method and save the simulation results to the variable `rpt`.

```
rpt = net.run()
```

If we want to take a closer look at the simulation results, we can access the report's different properties. Since we want to use the flow and pressure data in the animation, we assign them to variables. We also limit the data to a single day and take a look at a few data points.

```
p = rpt.pressure.loc['2016-01-01']
f = rpt.flow.loc['2016-01-01']
print(p)
print(f)
```

Now, we create an animation using the Network's `.animate()` method. First, we create matplotlib *Figure* and *Axes* objects and pass a desired figure size:

```
fig, ax = plt.subplots(figsize=(7.5, 5))
```

We then pass the `ax` object to the `animate` method along with the simulation data. We call the flow data's `.abs()` method, to use the absolute flow values in the animation. The labels for the node and link color bars have to be passed as well. You can also specify how long the interval between the reporting time steps should be. The model uses a reporting time step of 5 minutes, so we choose an interval of 50 ms. The `robust` argument limits the color bar to values between the 2nd and the 98th percentile of the passed data's value range.

```
anim = net.animate(ax=ax, nodes=p, links=f.abs(), node_label="Pressure (m)", link_label=
↪"Flow (m)", robust=True, interval=50)
```

Finally, we can save the animation. Using the `dpi` and `fps` attributes helps you control the animation quality and file size:

```
anim.save("simple_animation.gif", dpi=200, writer=PillowWriter(fps=20))
```

Bokeh

Now, we will create an interactive plot using `bokeh`. Let's start with our imports and model reading (we will use the "C-Town" model):

```
import os

from bokeh.plotting import output_file, show
import oopnet as on

filename = os.path.join('data', 'C-town.inp')
net = on.Network.read(filename)
```

Next, we simulate the model and get the pressure and flow results for our plot:

```
rpt = net.run()
p = rpt.pressure
f = rpt.flow
```

Bokeh creates a HTML file that will be locally stored on your device. This file will contain our plot. We can set the file name using `bokeh.plotting.output_file()`:

```
output_file('bokehexample.html')
```

Finally, we create and show the plot:

```
plot = net.bokehplot(nodes=p, links=f, colormap=dict(node='viridis', link='cool'))
show(plot)
```

Summary

Static Matplotlib Plotting

```
import os

from matplotlib import pyplot as plt
import oopnet as on

filename = os.path.join('data', 'C-town.inp')

net = on.Network.read(filename)

net.plot()

rpt = net.run()
p = rpt.pressure
f = rpt.flow

net.plot(nodes=p, links=f)
net.plot(nodes=p, links=f, robust=True)

p_reduced = p.iloc[:50]
net.plot(nodes=p_reduced, links=f, robust=True, truncate_nodes=True)

diameters = on.get_diameter(net)
net.plot(linkwidth=diameters, markersize=0)

fig, ax = plt.subplots(figsize=(12, 9), dpi=250)
net.plot(ax=ax, nodes=p, links=f, robust=True)

plt.show()
```

Animated Matplotlib Plotting

```
import os

from matplotlib import pyplot as plt
from matplotlib.animation import PillowWriter

import oopnet as on

filename = os.path.join('data', 'L-TOWN_AreaC.inp')
net = on.Network.read(filename)

rpt = net.run()

p = rpt.pressure.loc['2016-01-01']
f = rpt.flow.loc['2016-01-01']
print(p)
print(f)

fig, ax = plt.subplots(figsize=(7.5, 5))
anim = net.animate(ax=ax, nodes=p, links=f.abs(), node_label="Pressure (m)", link_label=
↪"Flow (m)", robust=True, interval=50)
anim.save("simple_animation.gif", dpi=200, writer=PillowWriter(fps=20))
```

Interactive Bokeh Plot

```
import os

from bokeh.plotting import output_file, show
import oopnet as on

filename = os.path.join('data', 'C-town.inp')
net = on.Network.read(filename)

rpt = net.run()
p = rpt.pressure
f = rpt.flow

output_file('bokehexample.html')
plot = net.bokehplot(nodes=p, links=f, colormap=dict(node='viridis', link='cool'))
show(plot)
```

2.3.5 Graphs

OOPNET can be used together with the library `networkx` for graph theoretic use cases.

First, we import all dependencies and read the "Anytown" model. We need the following libraries:

- `os` is used for specifying file paths
- `networkx` provides graph theoretical functionalities
- `matplotlib.pyplot` is used for plotting
- `seaborn` is based on `matplotlib` and is also used for plotting

```
import os

import networkx as nx
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import oopnet as on

filename = os.path.join('data', 'anytown.inp')
network = on.Network.read(filename)
```

To use NetworkX, we have to create a graph object from our model. NetworkX supports different graph types:

- `networkx.Graph`
- `networkx.DiGraph`
- `networkx.MultiGraph`
- `networkx.MultiDiGraph`

Choose a graph type that is suitable for the analysis you want to do. OOPNET provides factories to convert a *Network* to the classes listed above.

> **Warning:** `networkx.Graph` and `networkx.DiGraph` do not support multiple connections between two vertices! This means, that if you have several pipes in parallel connecting two junctions, only one of those pipes will be kept in graph. If you want to keep all connections, use `networkx.MultiGraph` or `networkx.MultiDiGraph` instead.

To convert a network into a graph object, use one of the factory classes available:

- *Graph*
- *DiGraph*
- *MultiGraph*
- *MultiDiGraph*

In this example, we create a `networkx.MultiGraph` from our model:

```
G = on.MultiGraph(network)
```

The package NetworkX offers various possibilities like for example calculate different graph measurements like computing the graph theoretical Center, Diameter or Radius of the graph:

```
print(f'Center: {nx.center(G)}')
```

```
Center: ['1', '4', '13', '19', '18']
```

```
print(f'Diameter: {nx.diameter(G)}')
```

```
Diameter: 7
```

```
print(f'Radius: {nx.radius(G)}')
```

```
Radius: 4
```

Now, we use Google's page rank algorithm on the network:

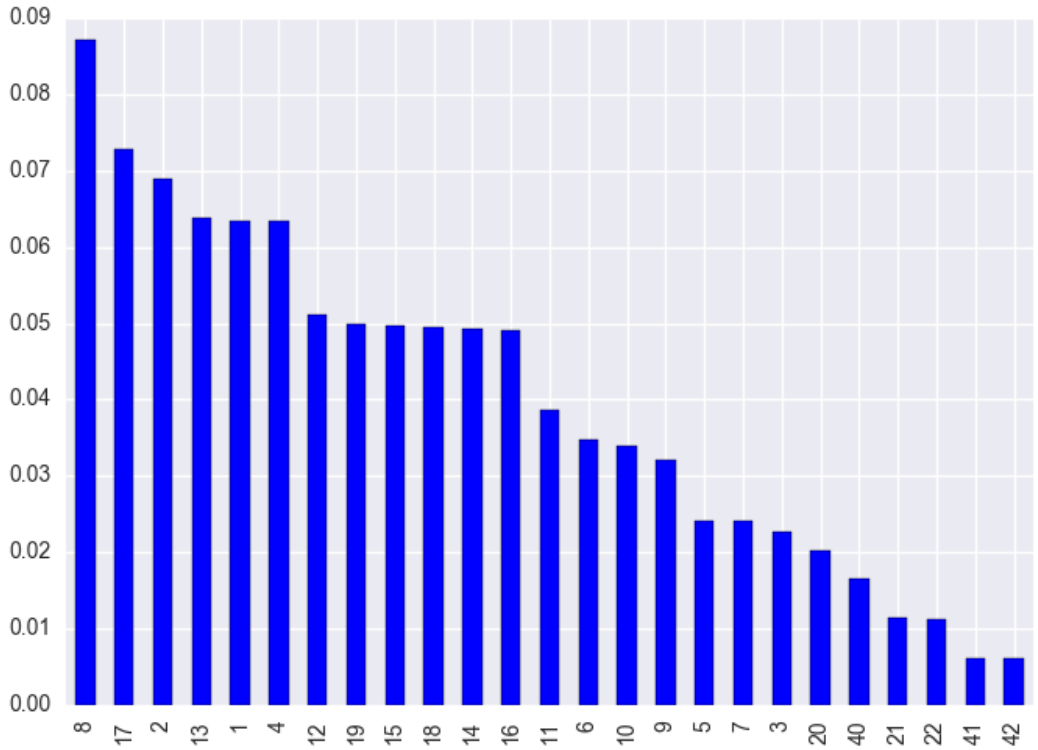
```
pr = nx.pagerank(G)
```

Let's create a `pandas.Series` out of the results for better data handling. We then sort this series in descending order and give it a name. This name which will serve as a label for the color bar in OOPNET's network plot:

```
pr = pd.Series(pr)
pr.sort_values(ascending=False, inplace=True)
pr.name = 'Page Rank'
```

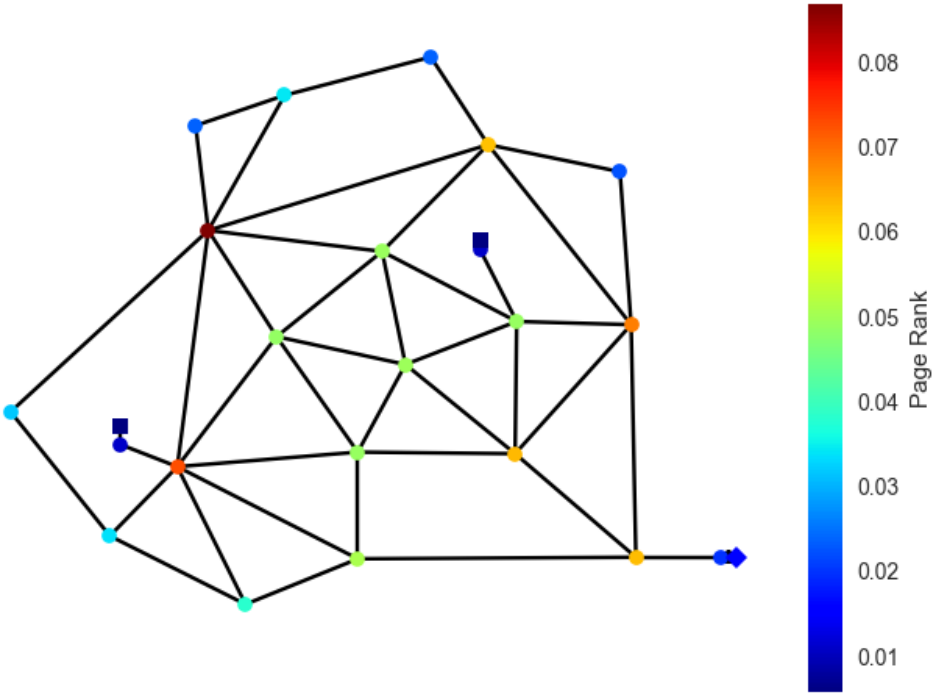
First, we plot the results as a bar plot:

```
f, ax = plt.subplots()
pr.plot(kind='bar', ax=ax)
```



We can also plot the page rank series directly on the network nodes with OOPNET's *plot()* function:

```
network.plot(nodes=pr)
```

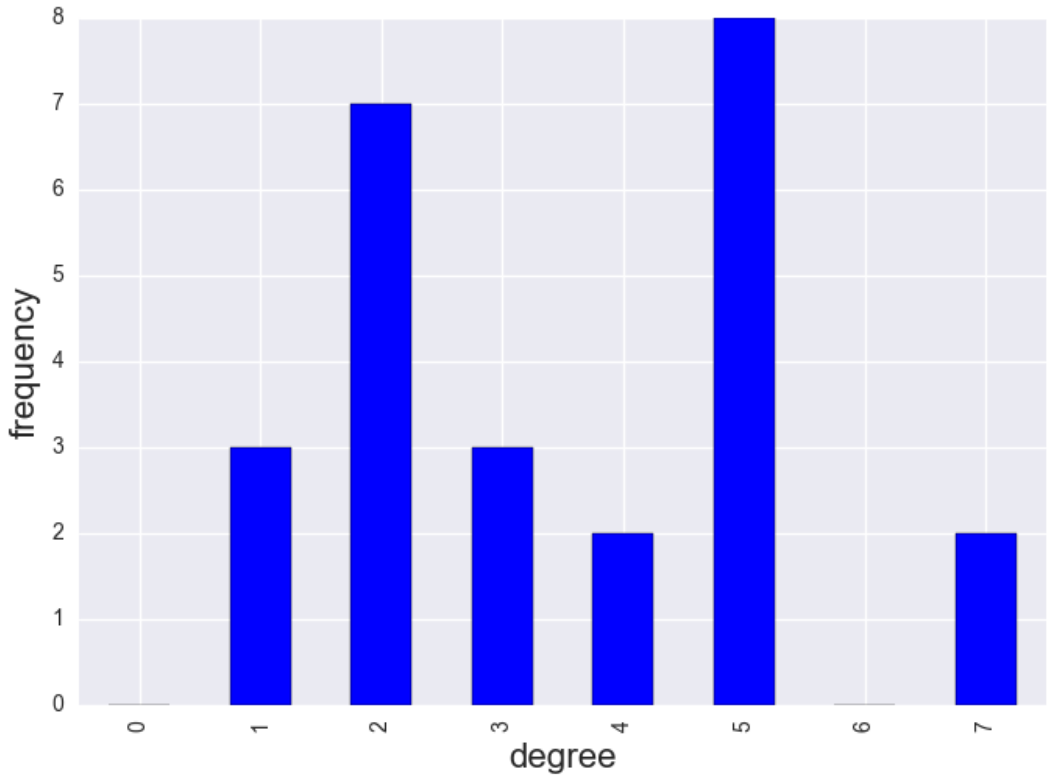


We can also calculate the degree of every node in the network and save it as a `pandas.Series`:

```
deg = nx.degree_histogram(G)
deg = pd.Series(deg)
```

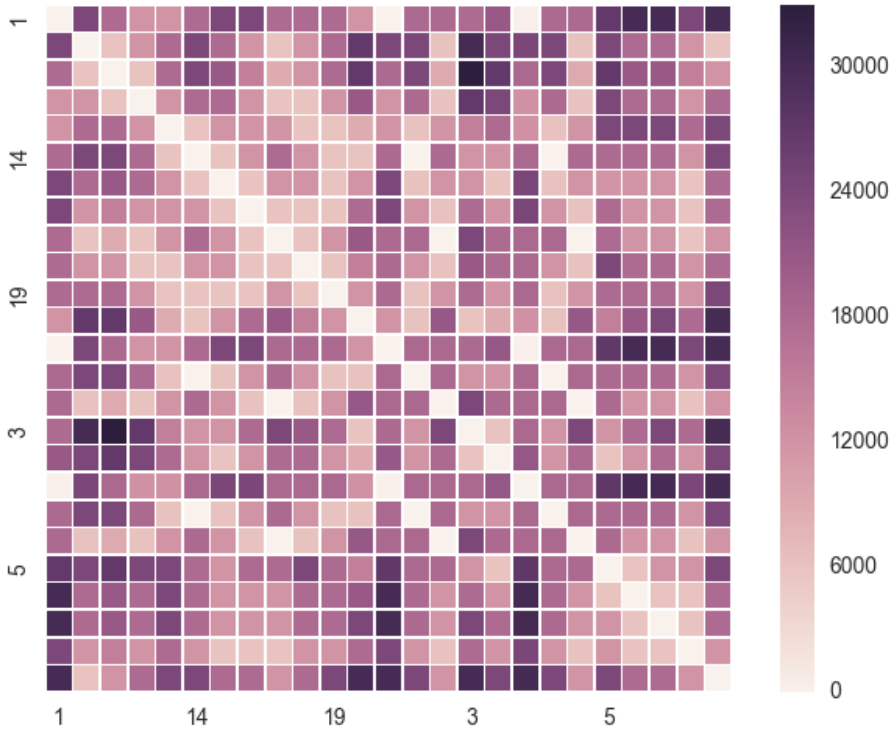
Of course, we can also plot this again as a bar plot:

```
f, ax = plt.subplots()
deg.plot(kind='bar', ax=ax)
plt.xlabel('degree', fontsize=16)
plt.ylabel('frequency', fontsize=16)
```



We can calculate all shortest paths in the network, save them as a `pandas.DataFrame` and plot them as a heatmap:

```
paths = dict(nx.all_pairs_dijkstra_path_length(G))
df = pd.DataFrame(paths)
f, ax = plt.subplots()

sns.heatmap(df, square=True, xticklabels=5, yticklabels=5, linewidths=.5)
plt.show()
```



Further Examples

Incidence and Adjacency Matrices

In this example, we take a look at calculation of incidence and adjacency matrices with OOPNET and NetworkX.

We first have to import the required packages:

- `os` is used for specifying the path to the EPANET input file
- `networkx` is responsible for the calculation of the matrices
- `matplotlib.pyplot` plots the matrices
- `oopnet` provides the means to use EPANET models for the matrix calculations

```
import os

import networkx as nx
from matplotlib import pyplot as plt
import oopnet as on
```

We are using the "Anytown" model in this example. We specify the path to the model and read it:

```
filename = os.path.join('data', 'anytown.inp')
net = on.Network.read(filename)
```

Next, we create a `networkx.Graph` object from our network by using the `oopnet.Graph` factory, get a list of all node IDs and convert the link objects in the model to `networkx` links:

```
G = on.Graph(net)
nodes = on.get_node_ids(net)
links = on.onlinks2nxlinks(net)
```

We can now calculate the incidence matrix for our network and show it in the console:

```
A = nx.incidence_matrix(G, nodelist=nodes, edgelist=links)
print('Incidence Matrix - not oriented')
print(A)
```

Next, we use the `oriented` argument to get the oriented incidence matrix:

```
B = nx.incidence_matrix(G, nodelist=nodes, edgelist=links, oriented=True)
print('Incidence matrix - oriented')
print(B)
```

Getting the adjacency matrix works very similar:

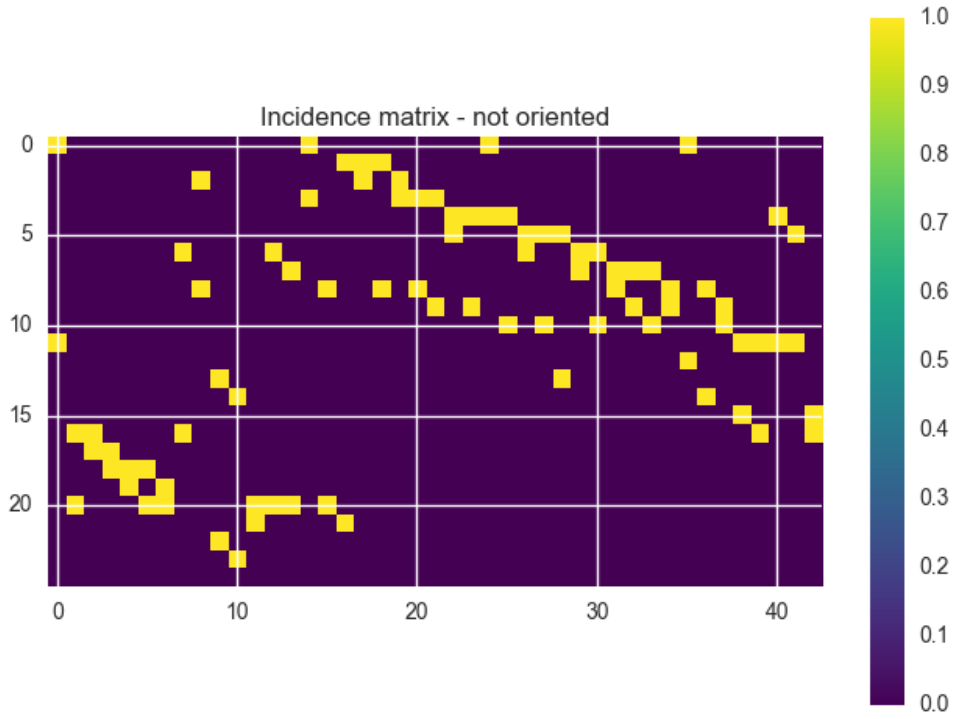
```
C = nx.adjacency_matrix(G, nodelist=nodes)
print('Adjacency matrix; undirected graph')
print(C)
```

We can not only create a simple `networkx.Graph` object from our network, we can also create `networkx.DiGraph`, `networkx.MultiGraph` and `networkx.MultiDiGraph` objects. OOPNET provides factories for all of these graph types (`oopnet.Graph`, `oopnet.DiGraph`, `oopnet.MultiGraph` and `oopnet.MultiDiGraph`). Here, we create a new `networkx.DiGraph` and calculate the adjacency matrix:

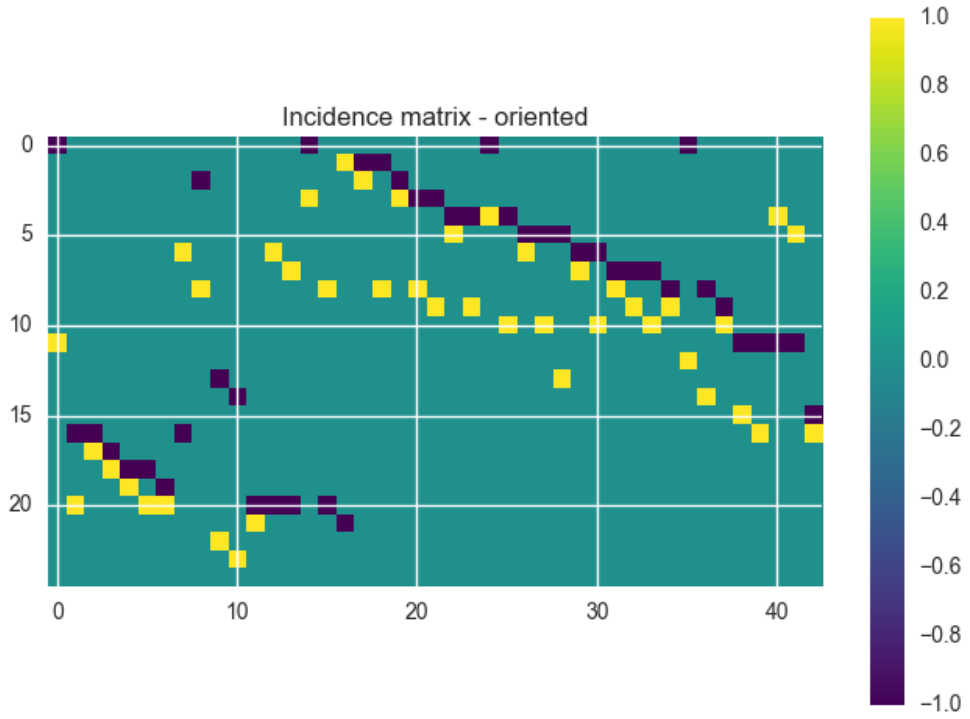
```
G = on.DiGraph(net)
D = nx.adjacency_matrix(G, nodelist=nodes)
print('Adjacency matrix; directed graph')
print(D)
```

Finally, we can use Matplotlib to plot the different matrices:

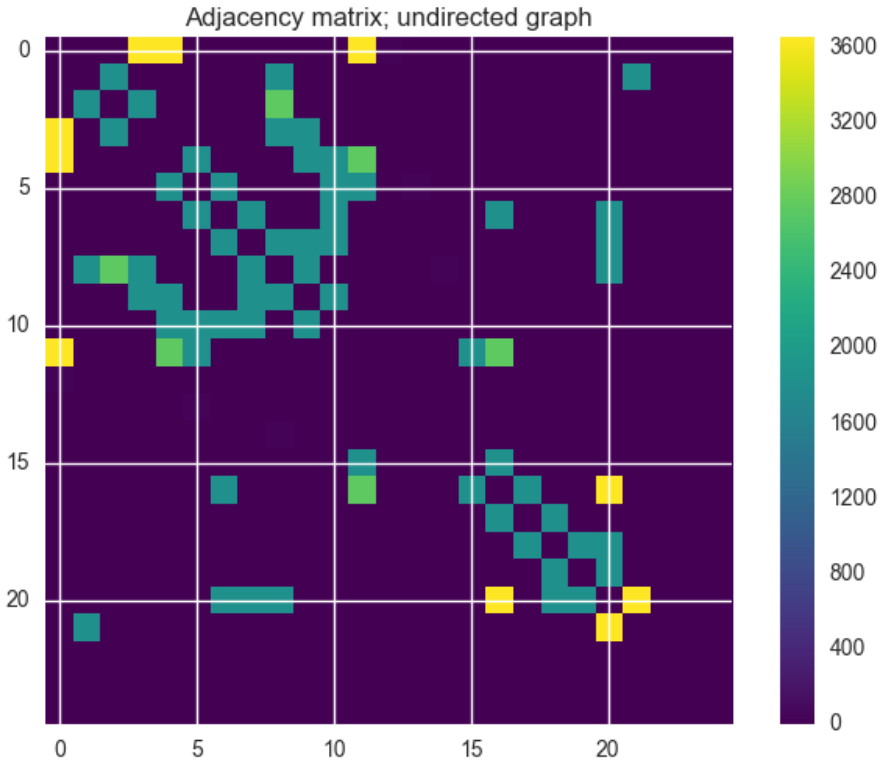
```
plt.figure(1)
plt.imshow(A.todense(), cmap='viridis', interpolation='nearest')
plt.title('Incidence matrix - not oriented')
plt.colorbar()
```



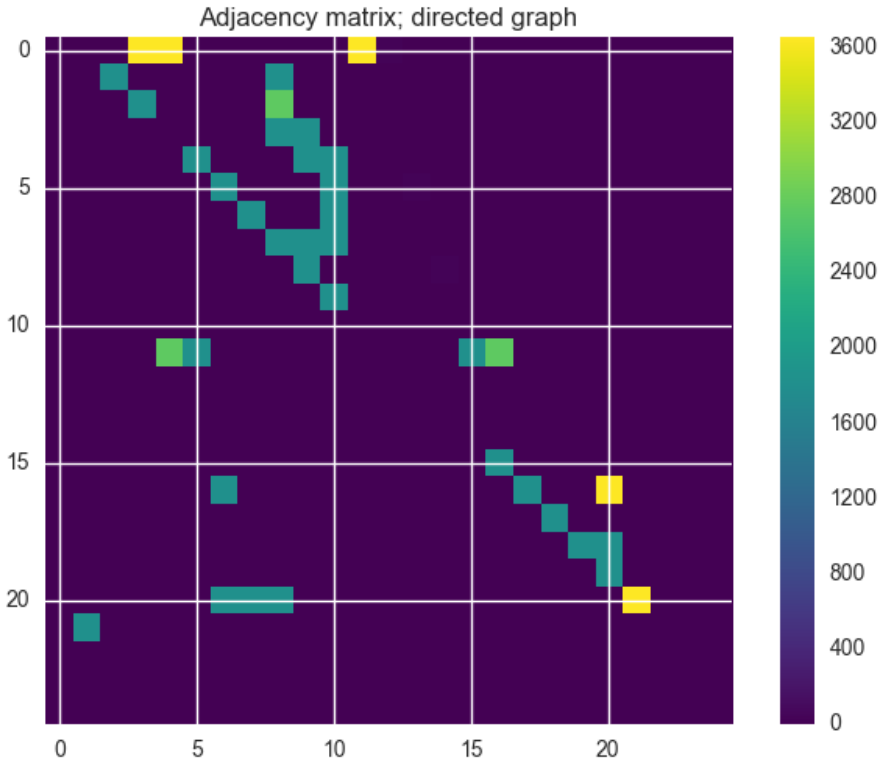

```
plt.figure(2)
plt.imshow(B.todense(), cmap='viridis', interpolation='nearest')
plt.title('Incidence matrix - oriented')
plt.colorbar()
```




```
plt.figure(3)
plt.imshow(C.todense(), cmap='viridis', interpolation='nearest')
plt.title('Adjacency matrix; undirected graph')
plt.colorbar()
```




```
plt.figure(4)
plt.imshow(D.todense(), cmap='viridis', interpolation='nearest')
plt.title('Adjacency matrix; directed graph')
plt.colorbar()
```



Use `plt.show()` to show the plots:

```
plt.show()
```

Summary

```
import os

import networkx as nx
from matplotlib import pyplot as plt
import oopnet as on

filename = os.path.join('data', 'anytown.inp')
net = on.Network.read(filename)

G = on.Graph(net)
nodes = on.get_node_ids(net)
links = on.onlinks2nxlinks(net)

A = nx.incidence_matrix(G, nodelist=nodes, edgelist=links)
print('Incidence Matrix - not oriented')
print(A)

B = nx.incidence_matrix(G, nodelist=nodes, edgelist=links, oriented=True)
```

(continues on next page)

(continued from previous page)

```
print('Incidence matrix - oriented')
print(B)


C = nx.adjacency_matrix(G, nodelist=nodes)
print('Adjacency matrix; undirected graph')
print(C)

G = on.DiGraph(net)
D = nx.adjacency_matrix(G, nodelist=nodes)
print('Adjacency matrix; directed graph')
print(D)

plt.figure(1)
plt.imshow(A.todense(), cmap='viridis', interpolation='nearest')
plt.title('Incidence matrix - not oriented')
plt.colorbar()

plt.figure(2)
plt.imshow(B.todense(), cmap='viridis', interpolation='nearest')
plt.title('Incidence matrix - oriented')
plt.colorbar()

plt.figure(3)
plt.imshow(C.todense(), cmap='viridis', interpolation='nearest')
plt.title('Adjacency matrix; undirected graph')
plt.colorbar()

plt.figure(4)
plt.imshow(D.todense(), cmap='viridis', interpolation='nearest')
plt.title('Adjacency matrix; directed graph')
plt.colorbar()

plt.show()
```

OOPNET in the middle - Centrality calculations

Another graph theoretic example of what you can do together with OOPNET and NetworkX.

In this example, different centrality calculations are shown. NetworkX is again used for the matrix calculations.

First, the necessary packages are imported, a filename is declared and an OOPNET `oopnet.Network` is created from the EPANET input file.

```
import os

import networkx as nx
from matplotlib import pyplot as plt
import pandas as pd
import oopnet as on

filename = os.path.join('data', 'C-town.inp')
```

(continues on next page)

(continued from previous page)

```
net = on.Network.read(filename)
```

Then a `networkx.MultiGraph` is created based on the Network.

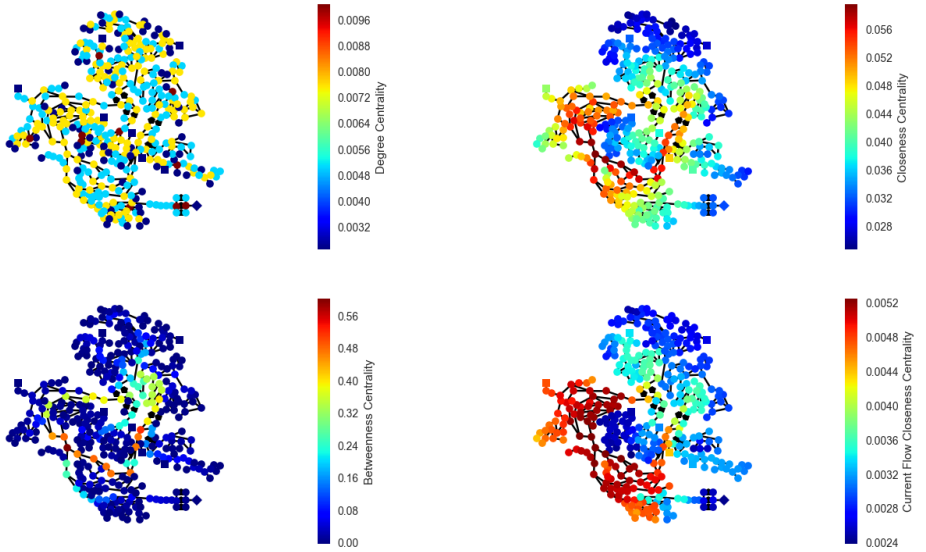
```
G = on.MultiGraph(net)
```

Next, a Matplotlib figure is instantiated. This object will hold a subplot for all of the four centralities we will calculate:

```
plt.figure()
```

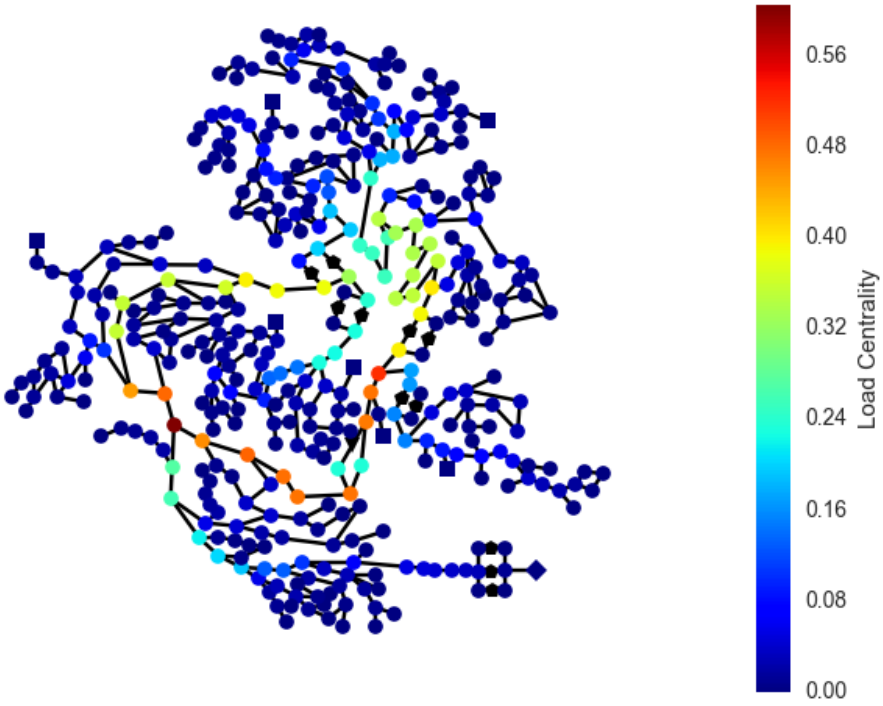
Now, the metrics are calculated. We pass the previously created `networkx.MultiGraph` to different NetworkX functions to calculate the metrics. Since we want to plot the centrality values per node, we need a pandas Series to pass it to the OOPNET `oopnet.Network.plot()` method. We change the series' names so that it is correctly shown in the plot. Then, we create Matplotlib axes objects and plot the network with these axes objects.

```
dc = nx.degree_centrality(G)
dc = pd.Series(dc)
dc.name = 'Degree Centrality'
ax = plt.subplot(221)
net.plot(nodes=dc, ax=ax)

cc = nx.closeness_centrality(G)
cc = pd.Series(cc)
cc.name = 'Closeness Centrality'
ax = plt.subplot(222)
net.plot(nodes=cc, ax=ax)

bc = nx.betweenness_centrality(G)
bc = pd.Series(bc)
bc.name = 'Betweenness Centrality'
ax = plt.subplot(223)
net.plot(nodes=bc, ax=ax)

cfcc = nx.current_flow_closeness_centrality(G)
cfcc = pd.Series(cfcc)
cfcc.name = 'Current Flow Closeness Centrality'
ax = plt.subplot(224)
net.plot(nodes=cfcc, ax=ax)
```



To create a single plot for a centrality, just omit the axes object creation and don't pass one to the plotting function:

```
lc = nx.load_centrality(G)
lc = pd.Series(lc)
lc.name = 'Load Centrality'
net.plot(nodes=lc)
```



Finally, show the plots:

```
plt.show()
```

Summary

```
import os

import networkx as nx
from matplotlib import pyplot as plt
import pandas as pd
import oopnet as on

filename = os.path.join('data', 'C-town.inp')

net = on.Network.read(filename)

G = on.MultiGraph(net)

plt.figure()

dc = nx.degree_centrality(G)
dc = pd.Series(dc)
dc.name = 'Degree Centrality'
```

(continues on next page)

(continued from previous page)

```
ax = plt.subplot(221)
net.plot(nodes=dc, ax=ax)

cc = nx.closeness_centrality(G)
cc = pd.Series(cc)
cc.name = 'Closeness Centrality'
ax = plt.subplot(222)
net.plot(nodes=cc, ax=ax)

bc = nx.betweenness_centrality(G)
bc = pd.Series(bc)
bc.name = 'Betweenness Centrality'
ax = plt.subplot(223)
net.plot(nodes=bc, ax=ax)

cfcc = nx.current_flow_closeness_centrality(G)
cfcc = pd.Series(cfcc)
cfcc.name = 'Current Flow Closeness Centrality'
ax = plt.subplot(224)
net.plot(nodes=cfcc, ax=ax)

lc = nx.load_centrality(G)
lc = pd.Series(lc)
lc.name = 'Load Centrality'
net.plot(nodes=lc)

plt.show()
```

Different Graph Weights

In this example, we will discuss how to specify different weights for edges, when creating a Graph from an OOPNET network.

First, we import `os`, `networkx` and `oopnet`.

```
import os

import networkx as nx
import oopnet as on

filename = os.path.join('data', 'C-town.inp')
```

We will use the "Anytown" model in this example, so we read it:

```
net = on.Network.read(filename)
```

When we create a `networkx.Graph` (or `networkx.DiGraph`, `networkx.MultiGraph` or `networkx.MultiDiGraph`) object using the corresponding factories in *oopnet.graph*, the pipe lengths are used as weights in the graph. We will use Dijkstra's shortest path to compare the different graphs.

```
G = on.MultiGraph(net)
avg_sp = nx.average_shortest_path_length(G, 'weight')
print(f'Average Shortest Path: {avg_sp}')
```

```
Average Shortest Path: 3494.6983087192225
```

You can specify a different weight, by passing a `weight` argument. You can specify any Link attribute you want (e.g. `roughness`). If a link doesn't have the attribute, a default value is used `0.00001`).

```
G = on.MultiGraph(net, weight='diameter')
avg_sp = nx.average_shortest_path_length(G, 'weight')
print(f'Average Shortest Path: {avg_sp}')
```

```
Average Shortest Path: 6363.355196667406
```

You can specify a different default value, if you want to:

```
G = on.MultiGraph(net, weight='diameter', default=0)
avg_sp = nx.average_shortest_path_length(G, 'weight')
print(f'Average Shortest Path: {avg_sp}')
```

```
Average Shortest Path: 6363.355182198411
```

Summary

```
import os

import networkx as nx
import oopnet as on

filename = os.path.join('data', 'C-town.inp')

net = on.Network.read(filename)

G = on.MultiGraph(net)
avg_sp = nx.average_shortest_path_length(G, 'weight')
print(f'Average Shortest Path: {avg_sp}')

G = on.MultiGraph(net, weight='diameter')
avg_sp = nx.average_shortest_path_length(G, 'weight')
print(f'Average Shortest Path: {avg_sp}')

G = on.MultiGraph(net, weight='diameter', default=0)
avg_sp = nx.average_shortest_path_length(G, 'weight')
print(f'Average Shortest Path: {avg_sp}')
```

Summary

```
import os

import networkx as nx
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import oopnet as on

filename = os.path.join('data', 'anytown.inp')
network = on.Network.read(filename)

G = on.MultiGraph(network)

# Some graph theoretic measurements:
print(f'Center: {nx.center(G)}')
print(f'Diameter: {nx.diameter(G)}')
print(f'Radius: {nx.radius(G)}')

# Page Rank algorithm
pr = nx.pagerank(G)
pr = pd.Series(pr)
pr.sort_values(ascending=False, inplace=True)
pr.name = 'Page Rank'

# Barplot
f, ax = plt.subplots()
pr.plot(kind='bar', ax=ax)

# Plot PageRank in network
network.plot(nodes=pr)

# Histogram of degrees in the network
deg = nx.degree_histogram(G)
deg = pd.Series(deg)

f, ax = plt.subplots()
deg.plot(kind='bar', ax=ax)
plt.xlabel('degree', fontsize=16)
plt.ylabel('frequency', fontsize=16)

# Calculate all shortest paths:
paths = dict(nx.all_pairs_dijkstra_path_length(G))
df = pd.DataFrame(paths)

# Plot shortest paths between all nodes
f, ax = plt.subplots()

sns.heatmap(df, square=True, xticklabels=5, yticklabels=5, linewidths=.5)
plt.show()
```

2.3.6 Controls

2.3.7 Examples

Here are some further examples of what you can do with OOPNET.

Logging

OOPNET allows for logging on different levels. *logging.INFO* provides basic information about what OOPNET is doing, while *logging.DEBUG* provides for detailed information.

To enable logging, you can either create your own logger and set the logging handlers as you please, or use logger provided by OOPNET. This logger uses a *RotatingFileHandler* and a *StreamHandler*. The *RotatingFileHandler* writes up to 5 MB of data to *oopnet.log* before rotating the logs. The default logging level is *logging.INFO* but this behaviour can be overruled.

To show logging functionality, we first have to import the necessary packages.

```
import os
import logging

import oopnet as on
```

Now, let's start the logger:

```
logger = on.start_logger()
```

Note: The logger only has to be started once! Don't put this function call in every part of your package/module! Below is also an example that describes, how to implement logging in the other parts of your program.

Next, we read the Poulakis model.

```
filename = os.path.join('data', 'Poulakis.inp')

net = on.Network.read(filename)
```

This leads to a log message in *oopnet.logs* and in the console:

```
Reading model from 'data/Poulakis.inp'
```

If you want more details, you can set the logging level to *logging.DEBUG*

```
logger.setLevel(logging.DEBUG)
```

Now, if we reread the Poulakis model, we will get way more information:

```
net = on.Network.read(filename)
```

And here are the log contents:

```
Reading model from 'data/Poulakis.inp'
Reading Curves
Reading Patterns
```

(continues on next page)

(continued from previous page)

```
Reading Junctions section
Added 30 Junctions
Reading Reservoirs section
Added 1 Reservoirs
Reading Tanks section
Added 0 Tanks
Reading Pipes section
Added 50 Pipes
Reading Pumps section
Added 0 Pumps
Reading Valve section
Added 0 Valves
Reading demand section
Reading Options
Reading report settings
Reading times settings
Reading Controls
Reading Energy
Reading Rules
Reading status section
Reading mixing section
Reading quality section
Reading reactions
Reading sources section
Reading Emitters section
Reading title
Reading Coordinates section
```

Now, what if you had a function, that you think might fail and that you want to log? OOPNET provides a decorator for this purpose, *logging_decorator()*, that needs a logger passed to it.

First, let's pretend that the logger has already been started in another part of you program and you want to add logging, to a different part of your program (i.e., you don't have to declare logging handlers, that's already taken care of). To do this, we simply have to get the logger:

```
logger = logging.getLogger('oopnet')
```

We will now use this logger to log a custom function. We want to log a function that tries to add a float and a string. After declaring the function, we call it and of course an exception is raised:

```
@on.logging_decorator(logger)
def do_some_crazy_things():
    a = 0
    for b in [0, 1, 2, 'a']:
        a += b
    return a
```

This results in the following logs:

```
Error raised by 'do_some_crazy_things'
Traceback (most recent call last):
  File "/home/***/oopnet/oopnet/utils/oopnet_logging.py", line 46, in wrapper
        return func(*args, **kwargs)
```

(continues on next page)

(continued from previous page)

```
  File "/home/***/oopnet_refactor/examples/logs.py", line 21, in do_some_crazy_things
        a += b
TypeError: unsupported operand type(s) for +=: 'int' and 'str'
Traceback (most recent call last):
  File "/home/***/oopnet/examples/logs.py", line 25, in <module>
        do_some_crazy_things()
  File "/home/***/oopnet/oopnet/utils/oopnet_logging.py", line 46, in wrapper
        return func(*args, **kwargs)
  File "/home/***/oopnet/examples/logs.py", line 21, in do_some_crazy_things
        a += b
TypeError: unsupported operand type(s) for +=: 'int' and 'str'
```

Summary

```
import os
import logging

import oopnet as on

logger = on.start_logger()

filename = os.path.join('data', 'Poulakis.inp')

net = on.Network.read(filename)

logger.setLevel(logging.DEBUG)

net = on.Network.read(filename)

logger = logging.getLogger('oopnet')


@on.logging_decorator(logger)
def do_some_crazy_things():
    a = 0
    for b in [0, 1, 2, 'a']:
        a += b
    return a


do_some_crazy_things()
```

2.4 API

OOPNET consists of several dedicated subpackages. In this part of the documentation, you can find the description of these different parts of OOPNET.

2.4.1 Subpackages

oopnet.elements package

This section describes the various classes that OOPNET provides for hydraulic models. The figure below shows the classes and their relationships:

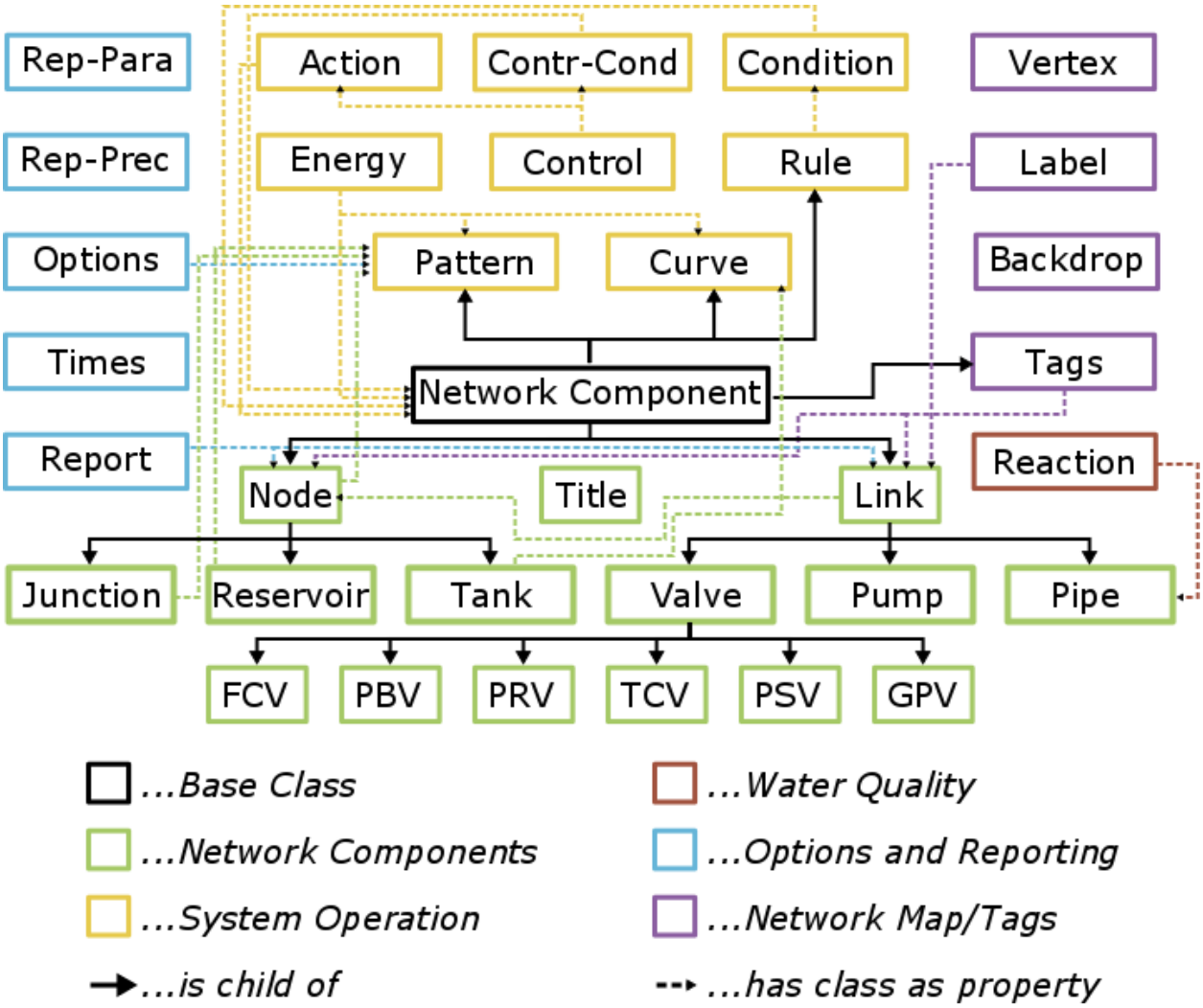


Fig. 6: Class structure of OOPNET

Submodules

oopnet.elements.base module

This module contains all the base classes of OOPNET

class oopnet.elements.base.**NetworkComponent**(*id=<property object>*, *comment=None*, *tag=None*)

Bases: ABC

This is OOPNET's base class for all objects having a name (id) in EPANET Input files

id

A unique label used to identify the Network Component. It can consist of a combination of up to 15 numerals or characters. It cannot be the same as the ID for any other node if it is a node, the same counts for links.

comment

Property containing a comment on the attribute if it is necessary. The comment is automatically read in from EPANET input files by parsing everything behind a semicolon (;), if a semicolon is not the first character in a line

tag

Associates category labels (tags) with specific nodes and links. An optional text string (with no spaces) used to assign e.g. the node to a category, such as a pressure zone.

comment: Optional[str] = None

abstract property id: str

tag: Optional[str] = None

oopnet.elements.component_registry module

exception oopnet.elements.component_registry.**ComponentNotExistingError**(*id*, *message=None*)

Bases: Exception

Raised when a no component with the ID exists in the network.

class oopnet.elements.component_registry.**ComponentRegistry**(*super_registry=None*)

Bases: dict

Class for storing NetworkComponents in a Network object or a SuperComponentRegistry.

Based on built-in dict but prevents overwriting an existing key and raises a ComponentNotExistingError error, when trying to look up a not exiting Component (instead of default KeyErrors).

exception oopnet.elements.component_registry.**IdenticalIDError**(*id*, *message=None*)

Bases: Exception

Raised when a component with the same ID already exists in the network.

class oopnet.elements.component_registry.**LinkRegistry**(**args*, ***kwargs*)

Bases: object

SuperComponentRegistry factory for Link components.

class oopnet.elements.component_registry.**NodeRegistry**(**args*, ***kwargs*)

Bases: object

SuperComponentRegistry factory for Node components.

class oopnet.elements.component_registry.**SuperComponentRegistry**(*classes*)

Bases: dict

Registry for Link and Node components.

Components are stored in ComponentRegistries for the individual subclasses (junctions, pipes, tanks, ...).

check_id_exists(*id*)

Checks if a component with the specified ID already exists in one of the ComponentRegistries.

Parameters
id – ID to check

Return type
bool

Returns
True, if the ID exists, False otherwise.

get_by_id(*id*)

Returns a component with a specified ID from the ComponentRegistries.

Args:
id: NetworkComponent ID

Raises
ComponentNotExistingError if no NetworkComponent with the specified ID is found. –

Return type
NetworkComponent

Returns
Requested NetworkComponent

oopnet.elements.network module

class oopnet.elements.network.**Network**(*title=None*, *labels=<factory>*, *backdrop=None*, *reactions=<factory>*, *options=<factory>*, *times=<factory>*, *report=<factory>*, *reportparameter=<factory>*, *reportprecision=<factory>*, *energies=<factory>*, *controls=<factory>*, *_nodes=<factory>*, *_links=<factory>*, *_curves=<factory>*, *_patterns=<factory>*, *_rules=<factory>*)

Bases: object

EPANET hydraulic model representation.

An OOPNET Network object contains all the information stored in an EPANET input file. This ranges from physical components like Junctions, Tanks or Pipes to non-physical components like Patterns, Curves or Controls. Furthermore, model settings and report parameter settings/precision settings are incorporated as well.

title

Network name

labels
List of all Labels in the network

backdrop
Contains the Backdrop object of the network

energies
List of all Energy curves in the network

controls
List of all Control objects in the network

_rules
List of all Rule objects in the network

reactions
List of all Reaction objects in the network

options
Option object representing model options

times
Time object representing time settings

report
SimulationReport object representing model report settings

reportparameter
Reportparameter object representing model report parameter settings

reportprecision
Reportprecision object representing model report parameter precision settings

_nodes
SuperComponentRegistry for all Node objects in the network

_links
SuperComponentRegistry for all Link objects in the network

_curves
ComponentRegistry of for Curve objects belonging to the network

_patterns
ComponentRegistry of for Pattern objects belonging to the network

animate(*fignum=None*, *nodes=None*, *node_label=None*, *links=None*, *link_label=None*, *linkwidth=None*, *colorbar=True*, *colormap='viridis'*, *ax=None*, *markersize=8.0*, *robust=False*, *nodes_vlim=None*, *links_vlim=None*, *truncate_nodes=None*, *interval=500*, *repeat=False*)

Animates the Network with simulation results as a network plot with Matplotlib.

Symbols for Nodes: Junctions are plotted as circles, Reservoirs as diamonds, Tanks as squares.

Symbols for Links: Pipes are plotted as lines with no markers, Valves are plotted as lines with triangulars in the middle, Pumps are plotted as lines with pentagons

Parameters

- **fignum** (*Optional[int]*) – figure number, where to plot the network

- **nodes** (*Optional[pd.Series]*) – Values related to the nodes as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Pressure(rpt)). If nodes is None or specific nodes do not have values, then the nodes are drawn as black circles
- **links** (*Optional[pd.Series]*) – Values related to the links as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)). If links is None or specific links do not have values, then the links are drawn as black lines
- **link_width** – Values describing the link width as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)).
- **colorbar** (*Union[bool, dict]*) – If True a colorbar is created, if False there is no colorbar in the plot. If one wants to set this setting for nodes and links seperatly, make use of a dictionary with key 'node' for nodes respectively key 'query_link' for links (e.g. colorbar = {'node':True, 'query_link':False} plots a colorbar for nodes but not for links)
- **colormap** (*Union[str, dict]*) – Colormap defining which colors are used for the simulation results (default is matplotlib's colormap viridis). colormap can either be a string for matplotlib colormaps, a matplotlib.colors.LinearSegmentedColormap object or a matplotlib.colors.ListedColormap object. If one wants to use different colormaps for nodes and links, then make use of a dictionary with key 'node' for nodes respectively key 'query_link' for links (e.g. colormaps = {'node':'jet', 'query_link':'cool'} plots nodes with colormap jet and links using colormap cool)
- **ax** (*Optional[Axes]*) – Matplotlib Axes object
- **markersize** (*float*) – size of markers
- **nodes_vlim** (*Optional[tuple[float, float]]*) – todo: add description
- **links_vlim** (*Optional[tuple[float, float]]*) –
- **robust** (*bool*) – If True, 2nd and 98th percentiles are used as limits for the colorbar, else the minima and maxima are used.
- **truncate_nodes** – If True, only junctions for which a value was submitted using the nodes parameter are plotted. If the nodes parameters isn't being used, all junctions are plotted. If not set True, junctions for which no value was submitted using the nodes parameters are plotted in black. This only applies to junctions and not to tanks and reservoirs, which are always plotted.

Return type
FuncAnimation

Returns
Matplotlib's figure handle

backdrop: Optional[*Backdrop*] = None

bokehplot(*tools=None*, *links=None*, *nodes=None*, *colormap='jet'*)

Plots the Network with simulation results as a network plot with Bokehplot.

Symbols for Nodes: Junctions are plotted as circles, Reservoirs as diamonds, Tanks as squares.

Symbols for Links: Pipes are plotted as lines with no markers, Valves are plotted as lines with triangulars standing on their top in the middle, Pumps are plotted as lines with triangulars standing on an edge

Parameters

- **tools** – tools used for the Bokeh plot (panning, zooming, ...)

- **nodes** – Values related to the nodes as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Pressure(rpt)). If nodes is None or specific nodes do not have values, then the nodes are drawn as black circles
- **links** – Values related to the links as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)). f links is None or specific links do not have values, then the links are drawn as black lines
- **colormap** – Colormap defining which colors are used for the simulation results (default is matplotlib's colormap jet). colormap can either be a string for matplotlib colormaps, a matplotlib.colors.LinearSegmentedColormap object or a matplotlib.colors.ListedColormap object. If one wants to use different colormaps for nodes and links, then make use of a dictionary with key 'node' for nodes respectively key 'query_link' for links (e.g. colormaps = {'node':'jet', 'query_link':'cool'} plots nodes with colormap jet and links using colormap cool)

Return type
BokehFigure

Returns
Bokehplot's figure handle

controls: *list[Control]*

energies: *list[Energy]*

labels: *dict[str, Label]*

options: *Options*

plot(*fignum=None*, *nodes=None*, *links=None*, *linkwidth=None*, *colorbar=True*, *colormap='viridis'*, *ax=None*, *markersize=8.0*, *robust=False*, *nodes_vlim=None*, *links_vlim=None*, *truncate_nodes=None*)

Plots the Network with simulation results as a network plot with Matplotlib.

Symbols for Nodes: Junctions are plotted as circles, Reservoirs as diamonds, Tanks as squares.

Symbols for Links: Pipes are plotted as lines with no markers, Valves are plotted as lines with triangulars in the middle, Pumps are plotted as lines with pentagons

Parameters

- **fignum** (*Optional[int]*) – figure number, where to plot the network
- **nodes** (*Optional[pd.Series]*) – Values related to the nodes as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Pressure(rpt)). If nodes is None or specific nodes do not have values, then the nodes are drawn as black circles
- **links** (*Optional[pd.Series]*) – Values related to the links as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)). If links is None or specific links do not have values, then the links are drawn as black lines
- **linkwidth** (*Optional[pd.Series]*) – Values describing the link width as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)).
- **colorbar** (*Union[bool, dict]*) – If True a colorbar is created, if False there is no colorbar in the plot. If one wants to set this setting for nodes and links seperatly, make use of a dictionary with key 'node' for nodes respectively key 'query_link' for links (e.g. colorbar = {'node':True, 'query_link':False} plots a colorbar for nodes but not for links)
- **colormap** (*Union[str, dict]*) – Colormap defining which colors are used for the simulation results (default is matplotlib's colormap viridis). colormap can either be a string

for matplotlib colormaps, a matplotlib.colors.LinearSegmentedColormap object or a matplotlib.colors.ListedColormap object. If one wants to use different colormaps for nodes and links, then make use of a dictionary with key 'node' for nodes respectively key 'query_link' for links (e.g. colormaps = {'node':'jet', 'query_link':'cool'} plots nodes with colormap jet and links using colormap cool)

- **ax** (*Optional[Axes]*) – Matplotlib Axes object
- **markersize** (*float*) – size of markers
- **nodes_vlim** – todo: add description
- **links_vlim** –
- **robust** (*bool*) – If True, 2nd and 98th percentiles are used as limits for the colorbar, else the minima and maxima are used.
- **truncate_nodes** – If True, only junctions for which a value was submitted using the nodes parameter are plotted. If the nodes parameters isn't being used, all junctions are plotted. If not set True, junctions for which no value was submitted using the nodes parameters are plotted in black. This only applies to junctions and not to tanks and reservoirs, which are always plotted.

Return type
PyPlotFigure

Returns
Matplotlib's figure handle

reactions: *Reaction*

classmethod read(*filename=typing.Optional[str]*, *content=typing.Optional[str]*)

Reads an EPANET input file.

Parameters

- **filename** – filename of the EPANET input file
- **content** – EPANET input file content as string

report: *Report*

reportparameter: *Reportparameter*

reportprecision: *Reportprecision*

run(*filename=None*, *delete=True*, *path=None*, *startdatetime=None*, *output=False*)

Runs an EPANET simulation by calling command line EPANET

filename

if thing is an OOPNET network, filename is an option to perform command line EPANET simulations with a specific filename. If filename is a Python None object then a file with a random UUID (universally unique identifier) is generated

delete

if delete is True the EPANET Input and SimulationReport file is deleted, if False then the simulation results won't be deleted and are stored in a folder named path

path

Path were to perform the simulations. If path is a Python None object then a tmp-folder is generated

output
If True, stdout and strerr will be printed to console and logged.

Return type
SimulationReport

Returns
OOPNET report object

times: *Times*

title: Optional[str] = None

write(*filename*)
Converts the Network to an EPANET input file and saves it with the desired filename.

Parameters
filename – desired filename/path were the user wants to store the file

Returns
0 if successful

oopnet.elements.network_components module

class oopnet.elements.network_components.**FCV**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *diameter=12.0*, *minorloss=0.0*, *maximum_flow=0.0*)

Bases: *Valve*

Flow Control Valve.

maximum_flow
maximum allow flow

maximum_flow: float = 0.0

property setting

class oopnet.elements.network_components.**GPV**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *diameter=12.0*, *minorloss=0.0*, *headloss_curve=None*)

Bases: *Valve*

General Purpose Valve.

headloss_curve
Curve representing flow-head loss relationship

headloss_curve: Curve = None

property setting

class oopnet.elements.network_components.**Junction**(*id=<property object>*, *comment=None*, *tag=None*, *xcoordinate=0.0*, *ycoordinate=0.0*, *elevation=0.0*, *initialquality=0.0*, *sourcequality=0.0*, *sourcetype=None*, *strength=0.0*, *sourcepattern=None*, *emittercoefficient=0.0*, *demandpattern=None*, *demand=0.0*)

Bases: *Node*

Junction node.

emittercoefficient

Discharge coefficient for emitter (sprinkler or nozzle) placed at junction. The coefficient represents the flow (in current flow units) that occurs at a pressure drop of 1 meter. Leave blank if no emitter is present. See the Emitters topic in the 'EPANET Manual Section 3.1 <https://epanet22.readthedocs.io/en/latest/3_network_model.html#physical-components>' for more details.

demandpattern

Pattern object used to characterize time variation in demand for the main category of consumer at the junction. The pattern provides multipliers that are applied to the Base Demand to determine actual demand in a given time period.

demand

The average or nominal demand for water by the main category of consumer at the junction, as measured in the current flow units. A negative value is used to indicate an external source of flow into the junction.

demand: Union[float, list[float]] = 0.0

demandpattern: Union[Pattern, list[Pattern], None] = None

emittercoefficient: float = 0.0

property id: str

class oopnet.elements.network_components.**Link**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*)

Bases: *NetworkComponent*

Base class for all Link like objects in OOPNET (Pipe, Pump, Valve).

startnode

Node-object at the start of the Link

endnode

Node-object at the end of the Link

status

Current status of the Link (OPEN, CLOSED, CV or ACTIVE)

property center

Returns the Link's center based on its start and end nodes as well as its vertices.

property coordinates: ndarray

Property returning start and end node coordinates

property coordinates_2d: ndarray

Property returning start and end node coordinates with the vertices in between them

endnode: Optional[Node] = None

revert()

Switches the link's start and end nodes and it's vertices.

startnode: Optional[Node] = None

status: str = 'OPEN'

vertices: list[Vertex]

class oopnet.elements.network_components.**Node**(*id=<property object>*, *comment=None*, *tag=None*, *xcoordinate=0.0*, *ycoordinate=0.0*, *elevation=0.0*, *initialquality=0.0*, *sourcequality=0.0*, *sourcetype=None*, *strength=0.0*, *sourcepattern=None*)

Bases: *NetworkComponent*

Base class for all Node like objects in OOPNET (Junction, Reservoir, Tank).

xcoordinate

The horizontal location of the junction on the map, measured in the map's distance units. If left blank, the node object will not appear on the network map.

ycoordinate

The vertical location of the junction on the map, measured in the map's distance units. If left blank, the node object will not appear on the network map.

elevation

The elevation in meters above some common reference of the node. This is a required property. Elevation is used only to compute pressure at the node. For tanks it is a required property and means Elevation above a common datum in meters of the bottom shell of the tank.

initialquality

Water quality level at the node at the start of the simulation period. Can be left blank if no water quality analysis is being made or if the level is zero.

sourcequality

Quality of any water entering the network at this location.

sourcetype

Source type (CONCEN, MASS, FLOWPACED, or SETPOINT)

strength

Baseline source strength

sourcepattern

Time Pattern object of source

property coordinates: ndarray

Property returning node coordinates.

Returns

x- and y-coordinate, and elevation

elevation: float = 0.0

initialquality: float = 0.0

sourcepattern: Optional[list[Pattern]] = None

sourcequality: float = 0.0

sourcetype: Optional[str] = None

strength: float = 0.0

xcoordinate: float = 0.0

ycoordinate: float = 0.0

class oopnet.elements.network_components.**PBV**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *diameter=12.0*, *minorloss=0.0*, *pressure_drop=0.0*)

Bases: *Valve*

Pressure Breaker Valve.

pressure_drop

pressure drop

pressure_drop: float = 0.0

property setting

class oopnet.elements.network_components.**PRV**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *diameter=12.0*, *minorloss=0.0*, *maximum_pressure=0.0*)

Bases: *Valve*

Pressure Reducing Valve.

maximum_pressure

pressure limit

maximum_pressure: float = 0.0

property setting

class oopnet.elements.network_components.**PSV**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *diameter=12.0*, *minorloss=0.0*, *pressure_limit=0.0*)

Bases: *Valve*

Pressure Sustaining Valve.

setting

pressure limit at upstream setting

pressure_limit: float = 0.0

property setting

class oopnet.elements.network_components.**Pipe**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *length=1000.0*, *diameter=12.0*, *roughness=100.0*, *minorloss=0.0*, *reactionbulk=None*, *reactionwall=None*)

Bases: *Link*

Pipe link.

length

The actual length of the pipe in meters.

diameter

The pipe diameter in mm.

roughness

The roughness coefficient of the pipe. It is unitless for Hazen-Williams or Chezy-Manning roughness and has units of mm for Darcy-Weisbach roughness.

minorloss

Unitless minor loss coefficient associated with bends, fittings, etc. Assumed 0 if left blank.

reactionbulk

The bulk reaction coefficient for the pipe. Time units are 1/days. Use a positive value for growth and a negative value for decay. Leave blank if the Global Bulk reaction coefficient from the project's Reaction Options will apply. See Water Quality Reactions in the 'EPANET manual Section 3.4<https://epanet22.readthedocs.io/en/latest/3_network_model.html#water-quality-simulation-model>' for moreinformation.

reactionwall

The wall reaction coefficient for the pipe. Time units are 1/days. Use a positive value for growth and a negative value for decay. Leave blank if the Global Wall reaction coefficient from the project's Reactions Options will apply. See Water Quality Reactions in the 'EPANET manual Section 3.4<https://epanet22.readthedocs.io/en/latest/3_network_model.html#water-quality-simulation-model>' for moreinformation.

diameter: float = 12.0

property id: str

length: float = 1000.0

minorloss: float = 0.0

reactionbulk: Optional[float] = None

reactionwall: Optional[float] = None

roughness: float = 100.0

split(*junction_id=None*, *pipe_id=None*, *split_ratio=0.5*)

Splits the pipe into two parts with respective lengths based on the passed split_ratio.

Creates a new Junction with the ID junction_id and a new Pipe with identical Pipe attributes except for xcoordinate, ycoordinate and elevation. These are defined by the split_ratio argument (measured from start to end node). If junction_id or pipe_id is not specified, the new Junction's/Pipe's ID will be derived from the old Junction and Pipe IDs.

> **Warning:** This will remove all vertices from the Pipe object.

Parameters

- **junction_id** (Optional[str]) – ID of newly created Junction
- **pipe_id** (Optional[str]) – ID of newly creation Pipe
- **split_ratio** (float) – ratio from start to end node along the Pipe

Return type
tuple[*Junction*, *Pipe*]

Returns
new Junction and Pipe objects

class oopnet.elements.network_components.**Pump**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *power=None*, *head=None*, *speed=1.0*, *pattern=None*, *setting=None*)

Bases: *Link*

Pump link.

todo: implement multiple keyword and value combinations .. attribute:: keyword

Can either be POWER (power value for constant energy pump, hp (kW)), HEAD (ID of curve that describeshead versus flow for the pump), SPEED (relative speed setting (normal speed is 1.0, 0 means pump is off)),PATTERN(ID of time pattern that describes how speed setting varies with time). Either POWER or HEAD must be supplied for each pump. The other keywords are optional.

value

Value according to the keyword attribute

status

head: Optional[Curve] = None

property id: str

pattern: Optional[Pattern] = None

power: Optional[float] = None

setting: Optional[float] = None

speed: float = 1.0

class oopnet.elements.network_components.**Reservoir**(*id=<property object>*, *comment=None*, *tag=None*, *xcoordinate=0.0*, *ycoordinate=0.0*, *elevation=0.0*, *initialquality=0.0*, *sourcequality=0.0*, *sourcetype=None*, *strength=0.0*, *sourcepattern=None*, *head=0.0*, *headpattern=None*)

Bases: *Node*

Reservoir nodes.

head

The hydraulic head (elevation + pressure head) of water in the reservoir in meters. This is a required property.

headpattern

Pattern object used to model time variation in the reservoir's head. Leave blank if none applies. This property is useful if the reservoir represents a tie-in to another system whose pressure varies with time.

mixingmodel

The type of water quality mixing that occurs within the tank. The choices include MIXED (fully mixed), 2COMP (two-compartment mixing), FIFO (first-in-first-out plug flow) and LIFO (last-in-first-out plug flow).

head: Union[float, list[float]] = 0.0

headpattern: Union[None, Pattern, list[Pattern]] = None

property id: str

class oopnet.elements.network_components.**TCV**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *diameter=12.0*, *minorloss=0.0*, *headloss_coefficient=0.0*)

Bases: *Valve*

Throttle Control Valve.

headloss_coefficient

head loss coefficient

headloss_coefficient: float = 0.0

property setting

class oopnet.elements.network_components.**Tank**(*id=<property object>*, *comment=None*, *tag=None*, *xcoordinate=0.0*, *ycoordinate=0.0*, *elevation=0.0*, *initialquality=0.0*, *sourcequality=0.0*, *sourcetype=None*, *strength=0.0*, *sourcepattern=None*, *initlevel=10.0*, *minlevel=0.0*, *maxlevel=20.0*, *diameter=50.0*, *minvolume=0.0*, *volumecurve=None*, *compartmentvolume=None*, *reactiontank=None*, *mixingmodel='MIXED'*)

Bases: *Node*

Tank node.

initlevel

Height in meters of the water surface above the bottom elevation of the tank at the start of the simulation.

minlevel

Minimum height in meters of the water surface above the bottom elevation that will be maintained. The water level in the tank will not be allowed to drop below this level.

maxlevel

Maximum height in meters of the water surface above the bottom elevation that will be maintained. The water level in the tank will not be allowed to rise above this level.

diameter

The diameter of the tank in meters. For cylindrical tanks this is the actual diameter. For square or rectangular tanks it can be an equivalent diameter equal to 1.128 times the square root of the cross-sectional area. For tanks whose geometry will be described by a curve (see below) it can be set to any value.

minvolume

The volume of water in the tank when it is at its minimum level, in cubic meter. This is an optional property, useful mainly for describing the bottom geometry of non-cylindrical tanks where a full volume versus depth curve will not be supplied (see below).

volumecurve

Curve object used to describe the relation between tank volume and water level. If no value is supplied then the tank is assumed to be cylindrical.

compartmentvolume

The fraction of the tank's total volume that comprises the inlet-outlet compartment of the two-compartment (2COMP) mixing model. Can be left blank if another type of mixing model is employed.

reactiontank

The bulk reaction coefficient for chemical reactions in the tank. Time units are 1/days. Use a positive value for growth reactions and a negative value for decay. Leave blank if the Global Bulk reaction coefficient specified in the project's Reactions Options will apply. See Water Quality Reactions in the 'EPANETmanual Section 3.4 <https://epanet22.readthedocs.io/en/latest/3_network_model.html#water-quality-simulation-model>' for moreinformation.

mixingmodel

The type of water quality mixing that occurs within the tank. The choices include MIXED (fully mixed), 2COMP (two-compartment mixing), FIFO (first-in-first-out plug flow) and LIFO (last-in-first-out plug flow).

compartmentvolume: Optional[float] = None

diameter: float = 50.0

property id: str

initlevel: float = 10.0

maxlevel: float = 20.0

minlevel: float = 0.0

minvolume: float = 0.0

mixingmodel: str = 'MIXED'

reactiontank: Optional[float] = None

volumecurve: Optional[Curve] = None

class oopnet.elements.network_components.**Valve**(*id=<property object>*, *comment=None*, *tag=None*, *startnode=None*, *endnode=None*, *status='OPEN'*, *vertices=<factory>*, *diameter=12.0*, *minorloss=0.0*)

Bases: *Link*

Valve link.

diameter

The valve diameter in mm.

minorloss

Unitless minor loss coefficient that applies when the valve is completely opened. Assumed 0 if left blank.

diameter: float = 12.0

property id: str

minorloss: float = 0.0

oopnet.elements.network_map_tags module

class oopnet.elements.network_map_tags.**Backdrop**(*dimensions*, *units*, *file*, *offset*)

Bases: object

Identifies a backdrop image and dimensions for the network map.

dimensions: list[float]

file: str

offset: list[float]

units: str

class oopnet.elements.network_map_tags.**Label**(*xcoordinate*, *ycoordinate*, *label*, *anchor*)

Bases: object

Assigns coordinates to map labels.

anchor: *Node*

label: str

xcoordinate: float

ycoordinate: float

class oopnet.elements.network_map_tags.**Tag**(*id*, *comment*, *object*, *tag*)

Bases: *object*

Associates category labels (tags) with specific nodes and links.

comment: str

id: str

object: *NetworkComponent*

tag: str

class oopnet.elements.network_map_tags.**Vertex**(*xcoordinate*, *ycoordinate*)

Bases: object

Vertex class.

xcoordinate

Vertex xcoordinate

ycoordinate

Vertex ycoordinate

property coordinates

Vertex x- and y-coordinate.

xcoordinate: float

ycoordinate: float

oopnet.elements.options_and_reporting module

class oopnet.elements.options_and_reporting.**Options**(*units='LPS'*, *headloss='H-W'*, *hydraulics=None*, *quality='NONE'*, *viscosity=1.0*, *diffusivity=1.0*, *specificgravity=1.0*, *trials=200*, *accuracy=0.001*, *unbalanced='STOP'*, *pattern=1.0*, *tolerance=0.01*, *map=None*, *demandmultiplier=1.0*, *emitterexponent=0.5*, *demandmodel='DDA'*, *minimumpressure=0.0*, *requiredpressure=0.1*, *pressureexponent=0.5*)

Bases: object

Defines various simulation options.

accuracy: float = 0.001

demandmodel: str = 'DDA'

demandmultiplier: float = 1.0

diffusivity: float = 1.0

emitterexponent: float = 0.5

headloss: str = 'H-W'

hydraulics: Optional[tuple[str, str]] = None

map: Optional[str] = None

minimumpressure: float = 0.0

pattern: Union[int, *Pattern*, None] = 1.0

pressureexponent: float = 0.5

quality: Union[str, tuple[str, str], tuple[str, str, str]] = 'NONE'

requiredpressure: float = 0.1

specificgravity: Optional[float] = 1.0

tolerance: float = 0.01

trials: int = 200

unbalanced: Union[str, tuple[str, int]] = 'STOP'

units: str = 'LPS'

viscosity: float = 1.0

class oopnet.elements.options_and_reporting.**Report**(*pagesize=0*, *file=None*, *status='NO'*, *summary='YES'*, *energy='NO'*, *nodes='ALL'*, *links='ALL'*, *parameter=None*, *value=None*)

Bases: object

Describes the contents of the output report produced from a simulation.

energy: str = 'NO'

file: Optional[str] = None

links: Union[str, *Link*, list[*Link*]] = 'ALL'

nodes: Union[str, *Node*, list[*Node*]] = 'ALL'

pagesize: int = 0

parameter: Union[str, tuple[str, float]] = None

status: str = 'NO'

summary: str = 'YES'

value: Optional[float] = None

class oopnet.elements.options_and_reporting.**Reportparameter**(*elevation='NO'*, *demand='YES'*, *head='YES'*, *pressure='YES'*, *quality='YES'*, *length='NO'*, *diameter='NO'*, *flow='YES'*, *velocity='YES'*, *headloss='YES'*, *setting='NO'*, *reaction='NO'*, *ffactor='NO'*)

Bases: object

The parameter reporting option is used to identify which quantities are reported on, how many decimal places are displayed, and what kind of filtering should be used to limit the output reporting.

Attributes:

demand: Union[str, tuple[str, float]] = 'YES'

diameter: Union[str, tuple[str, float]] = 'NO'

elevation: Union[str, tuple[str, float]] = 'NO'

ffactor: Union[str, tuple[str, float]] = 'NO'

flow: Union[str, tuple[str, float]] = 'YES'

head: Union[str, tuple[str, float]] = 'YES'

headloss: Union[str, tuple[str, float]] = 'YES'

length: Union[str, tuple[str, float]] = 'NO'

pressure: Union[str, tuple[str, float]] = 'YES'

quality: Union[str, tuple[str, float]] = 'YES'

reaction: Union[str, tuple[str, float]] = 'NO'

setting: Union[str, tuple[str, float]] = 'NO'

velocity: Union[str, tuple[str, float]] = 'YES'

class oopnet.elements.options_and_reporting.**Reportprecision**(*elevation=2*, *demand=2*, *head=2*, *pressure=2*, *quality=2*, *length=2*, *diameter=2*, *flow=2*, *velocity=2*, *headloss=2*, *setting=2*, *reaction=2*, *ffactor=2*)

Bases: object

Describes the precision per report parameter.

demand: int = 2

diameter: int = 2

elevation: int = 2

ffactor: int = 2

flow: int = 2

head: int = 2

headloss: int = 2

length: int = 2

pressure: int = 2

quality: int = 2

reaction: int = 2

setting: int = 2

velocity: int = 2

class oopnet.elements.options_and_reporting.**Times**(*duration=datetime.timedelta(0)*, *hydraulictimestep=datetime.timedelta(seconds=3600)*, *qualitytimestep=None*, *ruletimestep=None*, *patterntimestep=None*, *patternstart=datetime.timedelta(0)*, *reporttimestep=datetime.timedelta(seconds=3600)*, *reportstart=datetime.timedelta(0)*, *startclocktime=datetime.timedelta(0)*, *statistic='NONE'*)

Bases: object

Defines various time step parameters used in the simulation.

duration: timedelta = datetime.timedelta(0)

hydraulictimestep: timedelta = datetime.timedelta(seconds=3600)

patternstart: timedelta = datetime.timedelta(0)

patterntimestep: Optional[timedelta] = None

qualitytimestep: Optional[timedelta] = None

reportstart: timedelta = datetime.timedelta(0)

reporttimestep: timedelta = datetime.timedelta(seconds=3600)

ruletimestep: Optional[timedelta] = None

startclocktime: timedelta = datetime.timedelta(0)

statistic: str = 'NONE'

oopnet.elements.system_operation module

class oopnet.elements.system_operation.**Action**(*object=None*, *value=None*)

Bases: *object*

An action clause in a rule-based control

object: Union[*Node*, *Link*, str] = None

value: Union[float, str] = None

class oopnet.elements.system_operation.**Condition**(*object=None*, *logical=None*, *attribute=None*, *relation=None*, *value=None*)

Bases: *object*

A condition clause in a rule-based control

attribute: str = None

logical: Optional[str] = None

object: Union[*Link*, *Node*] = None

relation: str = None

value: Union[float, str, datetime, timedelta] = None

class oopnet.elements.system_operation.**Control**(*action=None*, *condition=None*)

Bases: object

Defines simple controls that modifiy links based on a single condition.

action: *Action* = None

condition: *Controlcondition* = None

class oopnet.elements.system_operation.**Controlcondition**(*object=None*, *relation=None*, *value=None*, *time=None*, *clocktime=None*)

Bases: *object*

clocktime: Optional[datetime] = None

object: Optional[*NetworkComponent*] = None

relation: Optional[str] = None

time: Union[None, float, timedelta] = None

value: Optional[float] = None

class oopnet.elements.system_operation.**Curve**(*id=<property object>*, *comment=None*, *tag=None*, *xvalues=<factory>*, *yvalues=<factory>*)

Bases: *NetworkComponent*

Defines data curves and their X,Y points.

property id: str

xvalues: list[float]

yvalues: list[float]

class oopnet.elements.system_operation.**Energy**(*keyword=None*, *pumpid=None*, *parameter=None*, *value=None*)

Bases: object

Defines parameters used to compute pumping energy and cost.

keyword: Optional[str] = None

parameter: Optional[str] = None

pumpid: Optional[str] = None

value: Union[float, *Pattern*, *Curve*] = None

class oopnet.elements.system_operation.**Pattern**(*id=<property object>*, *comment=None*, *tag=None*, *multipliers=<factory>*)

Bases: *NetworkComponent*

Defines time patterns.

property id: str

multipliers: list[float]

class oopnet.elements.system_operation.**Rule**(*id*, *condition=<factory>*, *priority=None*)

Bases: object

Defines rule-based controls that modify links based on a combination of conditions.

condition: list[*Condition*]

id: str

priority: float = None

oopnet.elements.water_quality module

class oopnet.elements.water_quality.**Reaction**(*orderbulk=1.0*, *orderwall=1.0*, *ordertank=1.0*, *globalbulk=0.0*, *globalwall=0.0*, *bulk=None*, *wall=None*, *tank=None*, *limitingpotential=None*, *roughnesscorrelation=None*)

Bases: object

Defines parameters related to chemical reactions occurring in the network.

bulk: Optional[list[*Pipe*]] = None

globalbulk: float = 0.0

globalwall: float = 0.0

limitingpotential: Optional[float] = None

orderbulk: float = 1.0

ordertank: float = 1.0

orderwall: float = 1.0

roughnesscorrelation: Optional[float] = None

tank: Optional[list[*Pipe*]] = None

wall: Optional[list[*Pipe*]] = None

Module contents

oopnet.graph package

Submodules

oopnet.graph.graph module

class oopnet.graph.graph.**DiGraph**(*network:* Network, *weight: str | pd.Series = 'length'*, *default: float = 1e-05*, *switch_direction: bool = True*)

Bases: `object`

Generates a directed NetworkX DiGraph from an OOPNET network.

Note: NetworkX DiGraphs don't support parallel edges pointing the same direction between two Nodes. Only one of the parallel edges will be present in the Graph object in this case. To allow for parallel pipes, use `oopnet.graph.MultiDiGraph` instead.

Parameters

- **network** – OOPNET network object
- **weight** – name of pipe property as a string which is used as weight or a pandas Series with link IDs as index and weights as values.
- **default** – When set, the default value is returned as weight for objects that don't have the defined weight attribute or that are missing in the weight pandas Series. Without it, an exception is raised for those objects.
- **switch_direction** – If a Link's weight is <0 and switch_direction is True, the Links start and end nodes will be switched.

Returns

NetworkX DiGraph object containing all nodes and links in the passed Network.

Examples

The following will create a DiGraph with link lengths as edge weights (filename needs to be a valid EPANET input file): >>> network = Network(filename) >>> g = DiGraph(network, 'length') Using a simulation result as link weight: >>> rpt = Run(network) >>> flow = Flow(rpt) >>> g = DiGraph(network, flow)

class oopnet.graph.graph.**Graph**(*network: Network*, *weight: str | pd.Series = 'length'*, *default: float = 1e-05*, *switch_direction: bool = True*)

Bases: `object`

Generates an undirected NetworkX Graph from an OOPNET network.

Note: NetworkX Graphs don't support parallel edges between two Nodes. Only one of the parallel edges will be present in the Graph object. To allow for parallel pipes, use `oopnet.graph.MultiGraph` instead.

Parameters

- **network** – OOPNET network object
- **weight** – name of pipe property as a string which is used as weight or a pandas Series with link IDs as index and weights as values.
- **default** – When set, the default value is returned as weight for objects that don't have the defined weight attribute or that are missing in the weight pandas Series. Without it, an exception is raised for those objects.
- **switch_direction** – If a Link's weight is <0 and switch_direction is True, the Links start and end nodes will be switched.

Returns

NetworkX Graph object containing all nodes and links in the passed Network.

Examples

The following will create a Graph with link lengths as edge weights (filename needs to be a valid EPANET input file): >>> network = Network(filename) >>> g = Graph(network, 'length') Using a simulation result as link weight: >>> rpt = Run(network) >>> flow = Flow(rpt) >>> g = Graph(network, flow)

class oopnet.graph.graph.**MultiDiGraph**(*network: Network*, *weight: str | pd.Series = 'length'*, *default: float = 1e-05*, *switch_direction: bool = True*)

Bases: `object`

Generates a directed NetworkX MultiGraph from an OOPNET network.

Parameters

- **network** – OOPNET network object
- **weight** – name of pipe property as a string which is used as weight or a pandas Series with link IDs as index and weights as values.
- **default** – When set, the default value is returned as weight for objects that don't have the defined weight attribute or that are missing in the weight pandas Series. Without it, an exception is raised for those objects.
- **switch_direction** – If a Link's weight is <0 and switch_direction is True, the Links start and end nodes will be switched.

Returns
NetworkX MultiGraph object containing all nodes and links in the passed Network.

Examples

The following will create a MultiGraph with link lengths as edge weights (filename needs to be a valid EPANET input file): >>> network = Network(filename) >>> g = MultiDiGraph(network, 'length') Using a simulation result as link weight: >>> rpt = Run(network) >>> flow = Flow(rpt) >>> g = MultiGraph(network, flow)

class oopnet.graph.graph.**MultiGraph**(*network: Network*, *weight: str | pd.Series = 'length'*, *default: float = 1e-05*, *switch_direction: bool = True*)

Bases: `object`

Generates an undirected NetworkX MultiGraph from an OOPNET network.

Parameters

- **network** – OOPNET network object
- **weight** – name of pipe property as a string which is used as weight or a pandas Series with link IDs as index and weights as values.
- **default** – When set, the default value is returned as weight for objects that don't have the defined weight attribute or that are missing in the weight pandas Series. Without it, an exception is raised for those objects.
- **switch_direction** – If a Link's weight is <0 and switch_direction is True, the Links start and end nodes will be switched.

Returns
NetworkX MultiGraph object containing all nodes and links in the passed Network.

Examples

The following will create a MultiGraph with link lengths as edge weights (filename needs to be a valid EPANET input file): >>> network = Network(filename) >>> g = MultiGraph(network, 'length') Using a simulation result as link weight: >>> rpt = Run(network) >>> flow = Flow(rpt) >>> g = MultiGraph(network, flow)

oopnet.graph.graph.**edgeresult2pandas**(*graph*, *result*)

Transforms edge data retrieved e.g. from edge centric centrality measurements to a Pandas Series compatible with OOPNET.

Parameters

- **graph** (`Graph`) – networkx graph object
- **result** (`dict`) – dictionary with Link IDs as keys

Return type
`Series`

Returns
transformed result into a pandas Series

oopnet.graph.graph.**nxedge2onlink_id**(*graph*, *edge*)

Converts an NetworkX edge in a graph to an OOPNET Link ID.

Parameters

- **graph** (`Graph`) – NetworkX graph

- **edge** (tuple[str, str]) – NetworkX edge

Return type
Union[str, list[str]]

Returns
ID of corresponding OOPNET Link

oopnet.graph.graph.**nxlinks2onlinks**(*graph*)

Converts NetworkX graph edges to OOPNET Link IDs.

Parameters
graph (Graph) – NetworkX graph

Return type
list[str]

Returns
List of OOPNET Link IDs

oopnet.graph.graph.**onlinks2nxlinks**(*network*)

Converts OOPNET links to NetworkX graph edges.

Parameters
network (*Network*) – OOPNET network object

Return type
list[tuple[str, str]]

Returns
List of tuples in the format (link.startnode.id, link.endnode.id)

Module contents

oopnet.hydraulics package

Module contents

oopnet.plotter package

Submodules

oopnet.plotter.bokehplot module

class oopnet.plotter.bokehplot.**Plotsimulation**(*network*, *tools=None*, *links=None*, *nodes=None*, *colormap='jet'*)

Bases: object

This function plots OOPNET networks with simulation results as a network plot with Bokehplot.

Symbols for Nodes: Junctions are plotted as circles, Reservoirs as diamonds, Tanks as squares.

Symbols for Links: Pipes are plotted as lines with no markers, Valves are plotted as lines with triangulars standing on their top in the middle, Pumps are plotted as lines with triangulars standing on an edge

Parameters

- **network** – OOPNET network object one wants to plot

- **tools** – tools used for the Bokeh plot (panning, zooming, ...)
- **nodes** – Values related to the nodes as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Pressure(rpt)). If nodes is None or specific nodes do not have values, then the nodes are drawn as black circles
- **links** – Values related to the links as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)). f links is None or specific links do not have values, then the links are drawn as black lines
- **colormap** – Colormap defining which colors are used for the simulation results (default is matplotlib's colormap jet). colormap can either be a string for matplotlib colormaps, a matplotlib.colors.LinearSegmentedColormap object or a matplotlib.colors.ListedColormap object. If one wants to use different colormaps for nodes and links, then make use of a dictionary with key 'node' for nodes respectively key 'link' for links (e.g. colormaps = {'node':'jet', 'link':'cool'} plots nodes with colormap jet and links using colormap cool)

Returns
Bokehplot's figure handle

oopnet.plotter.bokehplot.**colorfun**(*series*, *colormap='jet'*)

Parameters

- **series** –
- **colormap** – (Default value = 'jet')

Returns:

oopnet.plotter.bokehplot.**convert_to_hex**(*rgba_color*)

Parameters
rgba_color –

Returns:

oopnet.plotter.bokehplot.**outsidelist**(*element*, *colorhash*)

Parameters

- **element** –
- **colorhash** –

Returns:

oopnet.plotter.bokehplot.**plotlink**(*f*, *elements*, *colors*, *marker='o'*)

Parameters

- **f** –
- **elements** –
- **colors** –
- **marker** – (Default value = 'o')

Returns:

oopnet.plotter.bokehplot.**plotnode**(*f*, *elements*, *colors*, *marker='o'*)

Parameters

- **f** –

- **elements** –
- **colors** –
- **marker** – (Default value = 'o')

Returns:

oopnet.plotter.bokehplot.**plotpipe**(*f*, *elements*, *colors*)

oopnet.plotter.pyplot module

class oopnet.plotter.pyplot.**NetworkPlotter**(*colorbar=True*, *colormap='viridis'*, *truncate_nodes=False*, *robust=False*, *markersize=5.0*)

Bases: object

Class for creating static matplotlib plots and matplotlib animations.

Parameters

- **colorbar** (Union[bool, dict]) – If True a colorbar is created, if False there is no colorbar in the plot. If one wants to set this setting for nodes and links seperatly, make use of a dictionary with key 'node' for nodes respectively key 'link' for links (e.g. colorbar = {'node':True, 'link':False} plots a colorbar for nodes but not for links)
- **colormap** (Union[str, dict]) – Colormap defining which colors are used for the simulation results (default is matplotlib's colormap viridis). colormap can either be a string for matplotlib colormaps, a matplotlib.colors.LinearSegmentedColormap object or a matplotlib.colors.ListedColormap object. If one wants to use different colormaps for nodes and links, then make use of a dictionary with key 'node' for nodes respectively key 'link' for links (e.g. colormaps = {'node':'jet', 'link':'cool'} plots nodes with colormap jet and links using colormap cool)
- **truncate_nodes** (bool) – If True, only junctions for which a value was submitted using the nodes parameter are plotted. If the nodes parameters isn't being used, all junctions are plotted. If not set True, junctions for which no value was submitted using the nodes parameters are plotted in black. This only applies to junctions and not to tanks and reservoirs, which are always plotted.
- **robust** (bool) – If True, 2nd and 98th percentiles are used as limits for the colorbar, else the minima and maxima are used.
- **markersize** (float) – size of markers

animate(*network*, *fignum=None*, *ax=None*, *nodes=None*, *node_label=None*, *links=None*, *link_label=None*, *link_width=None*, *interval=500*, *repeat=False*, *nodes_vlim=None*, *links_vlim=None*)

This function plots OOPNET networks with simulation results as a network plot with Matplotlib.

Symbols for Nodes: Junctions are plotted as circles, Reservoirs as diamonds, Tanks as squares.

Symbols for Links: Pipes are plotted as lines with no markers, Valves are plotted as lines with triangulars in the middle, Pumps are plotted as lines with pentagons

Parameters

- **network** (*Network*) – OOPNET network object one wants to plot
- **fignum** (Optional[int]) – figure number, where to plot the network
- **ax** (Optional[Axes]) – Matplotlib Axes object

- **nodes** (Optional[DataFrame]) – Values related to the nodes as Pandas Series generated e.g. by one of OOPNET's SimulationReport properties (e.g. rpt.pressure). If nodes is None or specific nodes do not have values, then the nodes are drawn as black circles
- **node_label** (Optional[str]) – label for the Node values colorbar
- **nodes_vlim** (Optional[tuple[float, float]]) – limits for the Node values colorbar as tuple (min, max)
- **links** (Optional[DataFrame]) – Values related to the links as Pandas Series generated e.g. by one of OOPNET's SimulationReport properties (e.g. rpt.flow). If links is None or specific links do not have values, then the links are drawn as black lines
- **link_label** (Optional[str]) – label for the Link values colorbar
- **links_vlim** (Optional[tuple[float, float]]) – limits for the Link values colorbar as tuple (min, max)
- **link_width** (Optional[DataFrame]) – Values describing the link width as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)).
- **interval** (int) – interval between the individual frames
- **repeat** (bool) – if True, the animation will be created as a recurring loop

Return type
FuncAnimation

Returns
Matplotlib's figure handle

plot(*network*, *fignum=None*, *nodes=None*, *links=None*, *link_width=None*, *ax=None*, *nodes_vlim=None*, *links_vlim=None*)

This function plots OOPNET networks with simulation results as a network plot with Matplotlib.

Symbols for Nodes: Junctions are plotted as circles, Reservoirs as diamonds, Tanks as squares.

Symbols for Links: Pipes are plotted as lines with no markers, Valves are plotted as lines with triangulars in the middle, Pumps are plotted as lines with pentagons

Parameters

- **network** (*Network*) – OOPNET network object one wants to plot
- **fignum** (Optional[int]) – figure number, where to plot the network
- **nodes** (Optional[Series]) – Values related to the nodes as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Pressure(rpt)). If nodes is None or specific nodes do not have values, then the nodes are drawn as black circles
- **nodes_vlim** (Optional[tuple[float, float]]) –
- **links** (Optional[Series]) – Values related to the links as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)). If links is None or specific links do not have values, then the links are drawn as black lines
- **links_vlim** (Optional[tuple[float, float]]) –
- **link_width** (Optional[Series]) – Values describing the link width as Pandas Series generated e.g. by one of OOPNET's SimulationReport functions (e.g. Flow(rpt)).
- **ax** (Optional[Axes]) – Matplotlib Axes object

Returns
Matplotlib's figure handle

Module contents

oopnet.reader package

Subpackages

oopnet.reader.factories package

Submodules

oopnet.reader.factories.base module

exception oopnet.reader.factories.base.**InvalidValveTypeError**(*received*)

Bases: Exception

Exception for invalid Valve types read from an EPANET input file.

A Valve can be one of six different types. This exception is raised, when an illegal Valve type is encountered in an EPANET input file.

exception oopnet.reader.factories.base.**LengthExceededError**(*actual_length*, *target_length*)

Bases: Exception

Exception for attribute lists that exceed their maximum specified length.

Since there is a predefined number of attributes for every NetworkComponent subclass, a list of attributes read from an EPANET input file must not exceed this predefined length.

class oopnet.reader.factories.base.**ReadFactory**

Bases: ABC

Abstract factory for EPANET input file reading.

oopnet.reader.factories.component_factory module

class oopnet.reader.factories.component_factory.**ComponentFactory**

Bases: *ReadFactory*

Base Factory for creating NetworkComponents and adding them to a Network.

oopnet.reader.factories.options_and_reporting module

class oopnet.reader.factories.options_and_reporting.**OptionsFactory**(*values: dict*, *options:* Options, *network:* Network)

Bases: *OptionsReportFactory*

class oopnet.reader.factories.options_and_reporting.**OptionsReportFactory**

Bases: *ReadFactory*

Module contents

oopnet.reader.reading_modules package

Submodules

oopnet.reader.reading_modules.read_network_components module

oopnet.reader.reading_modules.read_network_map_tags module

oopnet.reader.reading_modules.read_options_and_reporting module

class oopnet.reader.reading_modules.read_options_and_reporting.**OptionReportReader**(*network*, *block*)

Bases: object

class oopnet.reader.reading_modules.read_options_and_reporting.**TimesReader**(*network*, *block*)

Bases: *OptionReportReader*

oopnet.reader.reading_modules.read_options_and_reporting.**parameter2report**(*vals*)

Parameters
vals (list) –

Return type
Union[str, list[str, float]]

Returns:

oopnet.reader.reading_modules.read_options_and_reporting.**precision2report**(*vals*)

Parameters
vals (list) –

Return type
int

Returns:

oopnet.reader.reading_modules.read_options_and_reporting.**time2timedelta**(*vals*)

Parameters
vals (list) –

Return type
timedelta

Returns:

oopnet.reader.reading_modules.read_system_operation module

oopnet.reader.reading_modules.read_water_quality module

Module contents

oopnet.reader.unit_converter package

Submodules

oopnet.reader.unit_converter.convert module

Convert all units which are possible in the EPANET-Input file to LPS Possible: (This are the flow units) - CFS … cubic feet per second - GPM … gallons per minute - MGD … million gallons per day - IMGD … Imperial MGD - AFD … acre-feet per day - LPS … liters per second - LPM … liters per minute - MLD … million liters per day - CMH … cubic meters per hour - CMD … cubic meters per day

The Input of OOPNET is possible in all units, but OOPNET uses and returns SI-Units (LPS).

class oopnet.reader.unit_converter.convert.**Converter**(*network*)

Bases: `object`

f_demand: float = 1.0

f_diameter_pipes: float = 1.0

f_diameter_tanks: float = 1.0

f_elevation: float = 1.0

f_emitter_coefficient: float = 1.0

f_flow: float = 1.0

f_hydraulic_head: float = 1.0

f_length: float = 1.0

f_power: float = 1.0

f_pressure: float = 1.0

f_reaction_coeff_wall: float = 1.0

f_roughness_coeff: float = 1.0

f_velocity: float = 1.0

f_volume: float = 1.0

oopnet.reader.unit_converter.convert.**convert**(*network*)

Parameters
network –

Returns:

Module contents

Submodules

oopnet.reader.decorators module

class oopnet.reader.decorators.**ReaderDecorator**(*sectionname=None*, *functionname=None*, *priority=None*, *readerfunction=None*)

Bases: `object`

Class for saving the reader function properties in a proper way with the decorators.

functionname: Optional[str] = None

priority: Optional[int] = None

readerfunction: Optional[Callable] = None

sectionname: Optional[str] = None

oopnet.reader.decorators.**make_registering_decorator_factory**(*foreign_decorator_factory*)

Parameters
foreign_decorator_factory –

Returns:

oopnet.reader.decorators.**section_reader**(**args*, ***kw*)

Synchronization decorator.

Used to decorate factory functions and classes for the EPANET input file reader.The title marks the section in the input file and the priority is used for ordering the different factories.

Parameters

- **title** – section title
- **priority** – reading priority

Returns
section reader

oopnet.reader.module_reader module

oopnet.reader.module_reader.**list_section_reader_callables**(*modules*)

Lists all callables decorated with the section_reader decorator from a list of modules.

Parameters
modules – list of modules to be checked for callables decorated with section_reader.

Return type
`list[Type[Callable]]`

Returns
List of callables decorated with section_reader.

oopnet.reader.module_reader.**pred**(*c*)

Checks if a class or function is decorated with section_reader.

Parameters
c (Type[Callable]) – callable to be checked

Return type
bool

Returns
Returns True if the class or function is decorated with section_reader and False otherwise.

oopnet.reader.read module

oopnet.reader.read.**filesplitter**(*content*)

Reads an EPANET input file and splits the content into blocks.

Parameters
content (list[str]) – EPANET input file content as str

Return type
dict[str, list]

Returns
blocks

oopnet.reader.read.**read**(*network*, *filename=None*, *content=None*)

Function reads an EPANET input file and returns a network object.

Parameters

- **filename** (Optional[str]) – filename of the EPANET input file
- **content** (Optional[str]) – EPANET input file content as string

Return type
Network

Returns
network object

Module contents

oopnet.report package

Submodules

oopnet.report.report module

class oopnet.report.report.**SimulationReport**(*filename*, *precision*, *startdatetime=None*, *reader=<function ReportFileReader>*)

Bases: object

Class for storing stimulation results.

nodes

Node results

links
Link results

property demand: Series | DataFrame
Demands from the simulation report object.

Returns
Pandas Series containing the demands of the Nodes

property diameter: Series | DataFrame
Diameters from the simulation report object.

Returns
Pandas Series containing the diameters of the Links

property elevation: Series | DataFrame
Elevations from the simulation report object.

Returns
Pandas Series containing the elevations of the Nodes

property ffactor: Series | DataFrame
Ffactors from the simulation report object.

Returns
Pandas Series containing the ffactors of the Links

property flow: Series | DataFrame
Flows from the simulation report object.

Returns
Pandas Series containing the flows of the Links

get_link_info(*id*)
Gets the Link information from a simulation report object.

Parameters
id (str) – Link ID

Return type
Series

Returns
Pandas Series containing the information of the links

get_node_info(*id*)
Gets the Node information from a simulation report object.

Parameters
id (str) – Node ID

Return type
Series

Returns
Pandas Series containing the information of the specified Node

property head: Series | DataFrame
Heads from the simulation report object.

Returns
Pandas Series containing the heads of the Nodes

property headloss: Series | DataFrame

Headlosses from the simulation report object.

WARNING: If one wants to work with headloss, then the length reportparameter has to be set to 'YES' in the Network's Reportparameter settings.

Returns
Pandas Series containing the headlosses of the Links

property headlossper1000m: Series | DataFrame

Headlosses from the simulation report object as it is in the report (units in headloss per 1000m)

Returns
Pandas Series containing the headlosses of the Links

property length: Series | DataFrame

Lengths from the simulation report object.

Returns
Pandas Series containing the lengths of the Links

links: DataArray

nodes: DataArray

property position: Series | DataFrame

Positions from the simulation report object.

Returns
Pandas Series containing the positions of the Links

property pressure: Series | DataFrame

Pressures from the simulation report object.

Returns
Pandas Series containing the pressures of the Nodes

property quality: Series | DataFrame

Qualities from the simulation report object.

Returns
Pandas Series containing the qualities of the Nodes

property reaction: Series | DataFrame

Reactions from the simulation report object.

Returns
Pandas Series containing the reactions of the Links

property settings

Settings from the simulation report object.

Returns
Pandas Series containing the settings of the Links

property velocity: Series | DataFrame

Velocities from the simulation report object.

Returns
Pandas Series containing the velocities of the Links

Module contents

oopnet.simulator package

Submodules

oopnet.simulator.binaryfile_reader module

class oopnet.simulator.binaryfile_reader.**BinaryFileReader**(*filename: str*, **args*, ***kwargs*)

Bases: object

oopnet.simulator.epanet2 module

oopnet.simulator.epanet2.**ModelSimulator**(*thing:* Network | *str*, *filename: str* | *None* = *None*, *delete: bool* = *True*, *path: str* | *None* = *None*, *startdatetime: datetime.datetime* | *None* = *None*, *output: bool* = *False*)

Runs an EPANET simulation by calling command line EPANET

oopnet.simulator.epanet2.**thing**

either an OOPNET network object or the filename of an EPANET input file

oopnet.simulator.epanet2.**filename**

if thing is an OOPNET network, filename is an option to perform command line EPANET simulations with a specific filename. If filename is a Python None object then a file with a random UUID (universally unique identifier) is generated

oopnet.simulator.epanet2.**delete**

if delete is True the EPANET Input and Report file is deleted, if False then the simulation results won't be deleted and are stored in a folder named path

oopnet.simulator.epanet2.**path**

Path were to perform the simulations. If path is a Python None object then a tmp-folder is generated

Returns

OOPNET report object

oopnet.simulator.error_manager module

class oopnet.simulator.error_manager.**ErrorManager**

Bases: object

Class for managing errors encountered while simulating a hydraulic model.

This class checks the EPANET report file for errors, stores these errors (and if available the error details) and then raises an EPANETSimulationError that contains all the encountered errors.

found_errors

list of errors found in the report file

_error_exp

regular expression used for finding errors

_error_list

list of possible errors that might be found in a report file as tuples where the first item is the

exception,
the second one the general error message and an optional third item stores the error details.

append_error_details(*text_line*)

Appends the details of an error to the already found error.

EPANET report files use either one or two lines of text for errors. The first one contains the error code and a general error message, while the following line contains a detailed error description. This function adds this second line of text to the corresponding error.

check_line(*text_line*)

Checks a single line of text for error codes.

If an error is encountered, it is added to the found_errors list together with the error message.

Parameters
text_line (str) – text line to be checked

Return type
bool

Returns
Returns False if no errors were encountered and True if otherwise.

raise_errors()

Raises an EPANETSimulationError if any errors were encountered while simulating the model.

oopnet.simulator.reportfile_reader module

oopnet.simulator.reportfile_reader.**ReportFileReader**(*filename: str*, *precision:* Reportprecision, *startdatetime: datetime | None = None*) → tuple[DataArray | Dataset | None, DataArray | Dataset | None]

oopnet.simulator.reportfile_reader.**blockkey2typetime**(*blockkey*, *startdatetime=None*)

Parameters

- **blockkey** (str) –
- **startdatetime** (Optional[datetime]) – (Default value = None)

Return type
tuple[str, datetime]

Returns:

oopnet.simulator.reportfile_reader.**lst2xray**(*lst*, *precision*)

Parameters
lst (list) – list:

Return type
DataArray

Returns:

oopnet.simulator.reportfile_reader.**str2hms**(*timestring*)

Converts a string to a tuple containing hours, minutes and seconds.

Parameters
timestring (str) – string to be parsed

Return type
tuple[int, int, float]

Returns
parsed timestring as a tuple

oopnet.simulator.simulation_errors module

exception oopnet.simulator.simulation_errors.**BinaryOutputFileAccessError**(*description*, *details*)

Bases: *EPANETError*

code = 304

exception oopnet.simulator.simulation_errors.**CheckValveImmutableError**(*description*, *details*)

Bases: *EPANETError*

code = 207

exception oopnet.simulator.simulation_errors.**EPANETError**(*description*, *details*)

Bases: Exception

Base class for simulation errors.

This exception is not meant to be called manually but is only a super class for the specific EPANET errors listed in the 'EPANET manual appendix concerning error messages <https://epanet22.readthedocs.io/en/latest/back_matter.html#error-messages>'.

code

error code as listed in the 'EPANET manual appendix concerning error messages

<https

//epanet22.readthedocs.io/en/latest/back_matter.html#error-messages>'

abstract property code

Error code as described in the EPANET manual.

exception oopnet.simulator.simulation_errors.**EPANETSimulationError**(*message*)

Bases: Exception

Error raised when any errors were encountered while simulating a hydraulic model.

check_contained_errors(*errors*)

property errors

Property containing all raised errors.

This property can be used as a shortcut to the raised errors. Useful when looking for a specific kind of error.

exception oopnet.simulator.simulation_errors.**EPANETSyntaxError**(*description*, *details*)

Bases: *EPANETError*

code = 201

exception oopnet.simulator.simulation_errors.HydraulicEquationError(*description*, *details*)

Bases: *EPANETError*

code = 110

exception oopnet.simulator.simulation_errors.IllegalAnalysisOptionError(*description*, *details*)

Bases: *EPANETError*

code = 213

exception oopnet.simulator.simulation_errors.IllegalLinkPropertyError(*description*, *details*)

Bases: *EPANETError*

code = 211

exception oopnet.simulator.simulation_errors.IllegalNodePropertyError(*description*, *details*)

Bases: *EPANETError*

code = 209

exception oopnet.simulator.simulation_errors.IllegalNumericalValueError(*description*, *details*)

Bases: *EPANETError*

code = 202

exception oopnet.simulator.simulation_errors.IllegalValveSourceConnectionError(*description*, *details*)

Bases: *EPANETError*

code = 219

exception oopnet.simulator.simulation_errors.IllegalValveValveConnectionError(*description*, *details*)

Bases: *EPANETError*

code = 220

exception oopnet.simulator.simulation_errors.InputDataError(*description*, *details*)

Bases: *EPANETError*

code = 200

exception oopnet.simulator.simulation_errors.InsufficientMemoryError(*description*, *details*)

Bases: *EPANETError*

code = 101

exception oopnet.simulator.simulation_errors.InvalidCurveError(*description*, *details*)

Bases: *EPANETError*

code = 230

exception oopnet.simulator.simulation_errors.InvalidEnergyDataError(*description*, *details*)

Bases: *EPANETError*

code = 217

exception oopnet.simulator.simulation_errors.**InvalidIDError**(*description*, *details*)

Bases: *EPANETError*

code = 252

exception oopnet.simulator.simulation_errors.**InvalidPumpError**(*description*, *details*)

Bases: *EPANETError*

code = 227

exception oopnet.simulator.simulation_errors.**InvalidTankLevelError**(*description*, *details*)

Bases: *EPANETError*

code = 225

exception oopnet.simulator.simulation_errors.**LinkReferenceError**(*description*, *details*)

Bases: *EPANETError*

code = 210

exception oopnet.simulator.simulation_errors.**MisplacedRuleClause**(*description*, *details*)

Bases: *EPANETError*

code = 221

exception oopnet.simulator.simulation_errors.**NodeReferenceError**(*description*, *details*)

Bases: *EPANETError*

code = 208

exception oopnet.simulator.simulation_errors.**NotEnoughNodesError**(*description*, *details*)

Bases: *EPANETError*

code = 223

exception oopnet.simulator.simulation_errors.**NotEnoughSourcesError**(*description*, *details*)

Bases: *EPANETError*

code = 224

exception oopnet.simulator.simulation_errors.**ReportFileAccessError**(*description*, *details*)

Bases: *EPANETError*

code = 303

exception oopnet.simulator.simulation_errors.**ReportFileSavingError**(*description*, *details*)

Bases: *EPANETError*

code = 309

exception oopnet.simulator.simulation_errors.**ResultFileSavingError**(*description*, *details*)

Bases: *EPANETError*

code = 308

exception oopnet.simulator.simulation_errors.**SharedIDError**(*description*, *details*)

Bases: *EPANETError*

code = 215

exception oopnet.simulator.simulation_errors.**TempInputFileAccessError**(*description*, *details*)

Bases: *EPANETError*

code = 302

exception oopnet.simulator.simulation_errors.**UnconnectedNodeError**(*description*, *details*)

Bases: *EPANETError*

code = 233

exception oopnet.simulator.simulation_errors.**UndefinedCurveError**(*description*, *details*)

Bases: *EPANETError*

code = 206

exception oopnet.simulator.simulation_errors.**UndefinedLinkError**(*description*, *details*)

Bases: *EPANETError*

code = 204

exception oopnet.simulator.simulation_errors.**UndefinedNodeError**(*description*, *details*)

Bases: *EPANETError*

code = 203

exception oopnet.simulator.simulation_errors.**UndefinedPumpError**(*description*, *details*)

Bases: *EPANETError*

code = 216

exception oopnet.simulator.simulation_errors.**UndefinedTimePatternError**(*description*, *details*)

Bases: *EPANETError*

code = 205

exception oopnet.simulator.simulation_errors.**UndefinedTraceNodeError**(*description*, *details*)

Bases: *EPANETError*

code = 212

oopnet.simulator.simulation_errors.**get_error_list**()

Lists all errors implemented.

Return type
list[Type[*EPANETError*]]

Module contents

oopnet.utils package

Subpackages

oopnet.utils.adders package

Submodules

oopnet.utils.adders.add_element module

oopnet.utils.adders.add_element.**add_curve**(*network*, *curve*)

Adds a Curve to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **curve** (*Curve*) – Curve object to add to the network

oopnet.utils.adders.add_element.**add_junction**(*network*, *junction*)

Adds a Junction to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **junction** (*Junction*) – Junction object to add to the network

oopnet.utils.adders.add_element.**add_link**(*network*, *link*)

Adds a Link to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **link** (Union[*Pipe*, *Pump*, *Valve*]) – Link object to add to the network

oopnet.utils.adders.add_element.**add_node**(*network*, *node*)

Adds a Node to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **node** (Union[*Junction*, *Reservoir*, *Tank*]) – Node object to add to the network

oopnet.utils.adders.add_element.**add_pattern**(*network*, *pattern*)

Adds a Pattern to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network
- **pattern** (*Pattern*) – Pattern object to add to the network

oopnet.utils.adders.add_element.**add_pipe**(*network*, *pipe*)

Adds a Pipe to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **pipe** (*Pipe*) – Pipe object to add to the network

oopnet.utils.adders.add_element.**add_pump**(*network*, *pump*)

Adds a Pump to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **pump** (*Pump*) – Pump object to add to the network

oopnet.utils.adders.add_element.**add_reservoir**(*network*, *reservoir*)

Adds a Reservoir to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **reservoir** (*Reservoir*) – Reservoir object to add to the network

oopnet.utils.adders.add_element.**add_rule**(*network*, *rule*)

Adds a Rule to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **rule** (*Rule*) – Rule object to add to the network

oopnet.utils.adders.add_element.**add_tank**(*network*, *tank*)

Adds a Tank to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **tank** (*Tank*) – Tank object to add to the network

oopnet.utils.adders.add_element.**add_valve**(*network*, *valve*)

Adds a Valve to an OOPNET network object.

Parameters

- **network** (*Network*) – OOPNET network object
- **valve** (*Valve*) – Valve object to add to the network

Module contents

oopnet.utils.getters package

Submodules

oopnet.utils.getters.element_lists module

oopnet.utils.getters.element_lists.**get_controls**(*network*)

Gets all Controls in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Control*]

Returns
list of Controls

oopnet.utils.getters.element_lists.**get_curve_ids**(*network*)

Gets all Curve IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Curve IDs

oopnet.utils.getters.element_lists.**get_curves**(*network*)

Gets all Curves in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Curve*]

Returns
list of Curves

oopnet.utils.getters.element_lists.**get_energy_entries**(*network*)

Gets all Energy entries in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Energy*]

Returns
list of Energy entries

oopnet.utils.getters.element_lists.**get_inflow_node_ids**(*network*)

Gets all IDs of Nodes that act is inflows into the system.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
ID list of Tanks, Reservoirs and Junctions with a base demand < 0

oopnet.utils.getters.element_lists.**get_inflow_nodes**(*network*)

Gets all Nodes that act is inflows into the system.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Node*]

Returns
list of Tanks, Reservoirs and Junctions with either a base demand < 0 or if any demand category is < 0 if demand categories are used as demand.

oopnet.utils.getters.element_lists.**get_junction_ids**(*network*)

Gets all Junction IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Junction IDs

oopnet.utils.getters.element_lists.**get_junctions**(*network*)

Gets all Junctions in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Junction*]

Returns
list of Junctions

oopnet.utils.getters.element_lists.**get_link_ids**(*network*)

Gets all Link IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Link IDs

oopnet.utils.getters.element_lists.**get_links**(*network*)

Gets all Links (Pipes, Pumps and Valves) in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Link*]

Returns
list of Links

oopnet.utils.getters.element_lists.**get_node_ids**(*network*)

Gets all Node IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Node IDs

oopnet.utils.getters.element_lists.**get_nodes**(*network*)

Gets all Nodes (Junctions, Reservoirs and Tanks) in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Node*]

Returns
list of Nodes

oopnet.utils.getters.element_lists.**get_pattern_ids**(*network*)

Gets all Pattern IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Pattern IDs

oopnet.utils.getters.element_lists.**get_patterns**(*network*)

Gets all Patterns in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Pattern*]

Returns
list of Patterns

oopnet.utils.getters.element_lists.**get_pipe_ids**(*network*)

Gets all Pipe IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Pipe IDs

oopnet.utils.getters.element_lists.**get_pipes**(*network*)

Gets all Pipes in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Pipe*]

Returns
list of Pipes

oopnet.utils.getters.element_lists.**get_pump_ids**(*network*)

Gets all Pump IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Pump IDs

oopnet.utils.getters.element_lists.**get_pumps**(*network*)

Gets all Pumps in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Pump*]

Returns
list of Pumps

oopnet.utils.getters.element_lists.**get_reservoir_ids**(*network*)

Gets all Reservoir IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Reservoir IDs

oopnet.utils.getters.element_lists.**get_reservoirs**(*network*)

Gets all Reservoirs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Reservoir*]

Returns
list of Reservoirs

oopnet.utils.getters.element_lists.**get_rule_ids**(*network*)

Gets all Rule IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Rule IDs

oopnet.utils.getters.element_lists.**get_rules**(*network*)

Gets all Rules in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Rule*]

Returns
list of Rules

oopnet.utils.getters.element_lists.**get_tank_ids**(*network*)

Gets all Tank IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Tank IDs

oopnet.utils.getters.element_lists.**get_tanks**(*network*)

Gets all Tanks in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Tank*]

Returns
list of Tanks

oopnet.utils.getters.element_lists.**get_valve_ids**(*network*)

Gets all Valve IDs in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[str]

Returns
list of Valve IDs

oopnet.utils.getters.element_lists.**get_valves**(*network*)

Gets all Valves in a network.

Parameters
network (*Network*) – OOPNET network object

Return type
list[*Valve*]

Returns
list of Valves

oopnet.utils.getters.get_by_id module

oopnet.utils.getters.get_by_id.**get_curve**(*network*, *id*)

Gets a specific Curve from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Curve

Return type
Curve

Returns
Curve with property ID

oopnet.utils.getters.get_by_id.**get_junction**(*network*, *id*)

Gets a specific Junction from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Junction

Return type
Junction

Returns
Junction with property ID

oopnet.utils.getters.get_by_id.**get_link**(*network*, *id*)

Gets a specific Link from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Link

Return type
Link

Returns
Link with property ID

oopnet.utils.getters.get_by_id.**get_node**(*network*, *id*)

Gets a specific Node from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Node

Return type
Node

Returns
Node with property ID

oopnet.utils.getters.get_by_id.**get_pattern**(*network*, *id*)

Gets a specific Pattern from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Pattern

Return type
Pattern

Returns
Pattern with property ID

oopnet.utils.getters.get_by_id.**get_pipe**(*network*, *id*)

Gets a specific Pipe from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object

- **id** (str) – ID of the Pipe

Return type
Pipe

Returns
Pipe with property ID

oopnet.utils.getters.get_by_id.**get_pump**(*network*, *id*)

Gets a specific Pump from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Pump

Return type
Pump

Returns
Pump with property ID

oopnet.utils.getters.get_by_id.**get_reservoir**(*network*, *id*)

Gets a specific Reservoir from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Reservoir

Return type
Reservoir

Returns
Reservoir with property ID

oopnet.utils.getters.get_by_id.**get_rule**(*network*, *id*)

Gets a specific Rule from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Rule

Return type
Rule

Returns
Rule with property ID

oopnet.utils.getters.get_by_id.**get_tank**(*network*, *id*)

Gets a specific Tank from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Tank

Return type
Tank

Returns
Tank with property ID

oopnet.utils.getters.get_by_id.**get_valve**(*network*, *id*)

Gets a specific Valve from the network with a specific ID.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – ID of the Valve

Return type
Valve

Returns
Valve with property ID

oopnet.utils.getters.property_getters module

oopnet.utils.getters.property_getters.**get_basedemand**(*network*)

Gets all base demand values of all Junctions in the Network as a pandas Series.

Builds the sum if more than one base demand exists for a single junction.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Node IDs as index and base demands as values.

oopnet.utils.getters.property_getters.**get_coordinates**(*network*)

Gets all x and y coordinate values of all Nodes in the Network as a pandas Dataframe

Parameters
network (*Network*) – OOPNET Network object

Return type
DataFrame

Returns
Pandas DataFrame with Node IDs as index and x and y coordinates as columns.

oopnet.utils.getters.property_getters.**get_diameter**(*network*)

Gets all diameter values of all Pipes and Valves in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Pipe/Valve IDs as index and diameters as values.

oopnet.utils.getters.property_getters.**get_elevation**(*network*)

Gets all elevation values of all Nodes in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Node IDs as index and elevations as values.

oopnet.utils.getters.property_getters.**get_endnodes**(*network*)

Gets all end nodes of all Links in the Network as a pandas Series..

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Link IDs as index and end nodes as values.

oopnet.utils.getters.property_getters.**get_length**(*network*)

Gets all length values of all Pipes in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Pink IDs as index and lengths as values.

oopnet.utils.getters.property_getters.**get_link_comment**(*network*)

Gets all comments of all Links in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Link IDs as index and comments as values.

oopnet.utils.getters.property_getters.**get_linkcenter_coordinates**(*network*)

Get the center coordinates of all Links in the Network as a pandas Dataframe.

Parameters
network (*Network*) – OOPNET Network object

Return type
DataFrame

Returns
Pandas DataFrame with Link IDs as index and the Links' center x and y coordinates as columns.

oopnet.utils.getters.property_getters.**get_minorloss**(*network*)

Gets all minor loss coefficient values of all Pipes in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Pipe IDs as index and minor loss coefficients as values.

oopnet.utils.getters.property_getters.**get_node_comment**(*network*)

Gets all comments of all Nodes in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Node IDs as index and comments as values.

oopnet.utils.getters.property_getters.**get_roughness**(*network*)

Gets all roughness values of all Pipes in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Pipe IDs as index and roughness values as values.

oopnet.utils.getters.property_getters.**get_setting**(*network*)

Gets all settings of all Pumps and Valves in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Pump/Valve IDs as index and settings as values.

oopnet.utils.getters.property_getters.**get_startendcoordinates**(*network*)

Gets all start and end coordinates of all Links in the Network as a pandas DataFrame.

Parameters
network (*Network*) – OOPNET Network object

Return type
DataFrame

Returns
Pandas DataFrame with Link IDs as index and a column for start and end node x and y coordinates respectively.

oopnet.utils.getters.property_getters.**get_startendnodes**(*network*)

Gets all start and endnodes of all Links in the Network as a pandas DataFrame.

Parameters
network (*Network*) – OOPNET Network object

Return type
DataFrame

Returns
Pandas DataFrame with Link IDs as index and a column for start and end nodes respectively.

oopnet.utils.getters.property_getters.**get_startnodes**(*network*)

Gets all start nodes of all Links in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Link IDs as index and start nodes as values.

oopnet.utils.getters.property_getters.**get_status**(*network*)

Gets all status of all Links in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Link IDs as index and status as values.

oopnet.utils.getters.property_getters.**get_xcoordinate**(*network*)

Gets all x coordinate values of all Nodes in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Node IDs as index and x coordinates as values.

oopnet.utils.getters.property_getters.**get_ycoordinate**(*network*)

Gets all y coordinate values of all Nodes in the Network as a pandas Series.

Parameters
network (*Network*) – OOPNET Network object

Return type
Series

Returns
Pandas Series with Node IDs as index and y coordinates as values.

oopnet.utils.getters.topology_getters module

oopnet.utils.getters.topology_getters.**get_adjacent_links**(*network*, *query_node*)

Gets Links that are connected to query_node.

Parameters

- **network** (*Network*) – Network in which the query is executed
- **query_node** (*Node*) – Node that the query is based on

Return type
list[*Link*]

Returns
list of Links that are connected to the passed Node

oopnet.utils.getters.topology_getters.**get_inflow_neighbor_nodes**(*network*)

Gets Nodes that are connected to the inflow Nodes with any sort of Link.

Parameters
network (*Network*) – Network in which the query is executed

Return type
list[*Node*]

Returns
list of Nodes that are connected with Network's inflow Nodes

oopnet.utils.getters.topology_getters.**get_neighbor_links**(*network*, *query_link*)

Gets Links that share a Node with the passed Link.

Parameters

- **network** (*Network*) – Network in which the query is executed
- **query_link** (*Link*) – Link that the query is based on

Return type
list[*Link*]

Returns
list of Links that are connected to the passed Link

oopnet.utils.getters.topology_getters.**get_neighbor_nodes**(*network*, *query_node*)

Gets Nodes that are connected to query_node with any sort of Link.

Parameters

- **network** (*Network*) – Network in which the query is executed
- **query_node** (*Node*) – Node that the query is based on

Return type
list[*Node*]

Returns
list of Nodes that are connected to the passed Node

oopnet.utils.getters.topology_getters.**get_next_neighbor_links**(*network*, *query_link*)

Gets Links that share a Node with the neighbors of the passed Link.

Parameters

- **network** (*Network*) – Network in which the query is executed
- **query_link** (*Link*) – Link that the query is based on

Return type
list[*Link*]

Returns
list of Links that are connected to the passed Link

oopnet.utils.getters.topology_getters.**get_next_neighbor_nodes**(*network*, *query_node*)
Gets Nodes that are connected to neighbor Nodes of query_node with any sort of Link.

Parameters

- **network** (*Network*) – Network in which the query is executed
- **query_node** (*Node*) – Node that the query is based on

Return type
list[*Node*]

Returns
list of Nodes that are connected to the passed Node's neighbors

oopnet.utils.getters.vectors module

oopnet.utils.getters.vectors.**v_demand**(*network*)
Gets all emitter coefficients values of all junctions in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
demand as numpy.ndarray

oopnet.utils.getters.vectors.**v_diameter**(*network*)
Gets all diameter values of all Pipes and valves in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
diameter as numpy.ndarray

oopnet.utils.getters.vectors.**v_elevation**(*network*)
Gets all elevation values of all nodes in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
elevation as numpy.ndarray

oopnet.utils.getters.vectors.**v_emittercoefficient**(*network*)

Gets all emitter coefficients values of all junctions in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
elevation as numpy.ndarray

oopnet.utils.getters.vectors.**v_head**(*network*)

Gets all head values of all reservoir in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
head as numpy.ndarray

oopnet.utils.getters.vectors.**v_initlevel**(*network*)

Gets all initial levels of all tanks in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
initial levels as numpy.ndarray

oopnet.utils.getters.vectors.**v_length**(*network*)

Gets all length values of all Pipes in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
length as numpy.ndarray

oopnet.utils.getters.vectors.**v_maxlevel**(*network*)

Gets all maximum levels of all tanks in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
maximum levels as numpy.ndarray

oopnet.utils.getters.vectors.**v_minlevel**(*network*)

Gets all minimum levels of all tanks in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
minimum levels as numpy.ndarray

oopnet.utils.getters.vectors.**v_minorloss**(*network*)

Gets all minor loss coefficient values of all Pipes in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
minor loss coefficient as numpy.ndarray

oopnet.utils.getters.vectors.**v_minvolume**(*network*)

Gets all minimal volumes of all tanks in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
minimal volumes as numpy.ndarray

oopnet.utils.getters.vectors.**v_roughness**(*network*)

Gets all roughness values of all Pipes in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
roughness as numpy.ndarray

oopnet.utils.getters.vectors.**v_tankdiameter**(*network*)

Gets all diameters of all tanks in the network as a numpy array

Parameters
network (*Network*) – OOPNET network object

Return type
array

Returns
tank diameters as numpy.ndarray

Module contents

oopnet.utils.removers package

Submodules

oopnet.utils.removers.remove_element module

oopnet.utils.removers.remove_element.**remove_junction**(*network*, *id*)

This function removes a specific Junction from an OOPNET network.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – Junction ID

oopnet.utils.removers.remove_element.**remove_link**(*network*, *id*)

This function removes a specific Link from an OOPNET network.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – Link ID

oopnet.utils.removers.remove_element.**remove_node**(*network*, *id*)

This function removes a specific Node from an OOPNET network.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – Node ID

oopnet.utils.removers.remove_element.**remove_pipe**(*network*, *id*)

This function removes a specific Pipe from an OOPNET network.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – Pipe ID

oopnet.utils.removers.remove_element.**remove_pump**(*network*, *id*)

This function removes a specific Pump from an OOPNET network.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – Pump ID

oopnet.utils.removers.remove_element.**remove_reservoir**(*network*, *id*)

This function removes a specific Reservoir from an OOPNET network.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – Reservoir ID

oopnet.utils.removers.remove_element.**remove_tank**(*network*, *id*)

This function removes a specific Tank from an OOPNET network.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – Tank ID

oopnet.utils.removers.remove_element.**remove_valve**(*network*, *id*)

This function removes a specific Valve from an OOPNET network.

Parameters

- **network** (*Network*) – OOPNET network object
- **id** (str) – Valve ID

Module contents

Submodules

oopnet.utils.oopnet_logging module

oopnet.utils.oopnet_logging.**logging_decorator**(*logger*)

Decorates a function to log exceptions.

Parameters
logger (Logger) – logger to be used for logging

Returns
decorated function

oopnet.utils.oopnet_logging.**start_logger**(*level=20*)

Initializes a RotatingFileHanlder and a StreamHandler for logging.

This function creates a logger with two handlers (RotatingFileHandler and StreamHandler) that can come in handy, if no other logging handlers are being used. The RotatingFileHandler writes it's output to 'oopnet.log' and rotates the file when it reaches a size of 5 MB.

Parameters
level (Union[int, str]) – logging level (e.g., logging.DEBUG)

Return type
Logger

Returns
logger object

oopnet.utils.timer module

oopnet.utils.timer.**tic**()

Start stopwatch timer.

oopnet.utils.timer.**time_it**(*func*)

Decorates a function to measure its execution time.

Parameters
func – callable to time

Returns
decorated function

oopnet.utils.timer.**toc**()

Read elapsed time from stopwatch.

oopnet.utils.utils module

oopnet.utils.utils.**copy**(*network*)

This function makes a deepcopy of an OOPNET network object

Parameters
network – OOPNET network object

Returns
deepcopy of OOPNET network object

oopnet.utils.utils.**make_measurement**(*report*, *sensors*, *precision=None*)

This function simulates a measurement in the system at predefined sensorpositions and returns a measurement vector

Parameters

- **report** (*SimulationReport*) – OOPNET report object
- **sensors** (dict) – dict with keys 'Flow' and/or 'Pressure' containing the node- resp. linkids as list

-> {'Flow':['flowsensor1', 'flowsensor2], 'Pressure':['sensor1', 'sensor2', 'sensor3']}
precision: dict with keys 'Flow' and/or 'Pressure' and number of decimals -> {'Flow':3, 'Pressure':2} report: SimulationReport: sensors: dict: precision: Optional[dict]: (Default value = None)

Returns
numpy vector containing the measurements

oopnet.utils.utils.**mkdir**(*newdir*)

Creates a new directory.

- already exists, silently complete
- regular file in the way, raise an exception
- parent directory(ies) does not exist, make them as well

Parameters
newdir (str) – path to be created

Returns:

Module contents

oopnet.writer package

Subpackages

oopnet.writer.writing_modules package

Submodules

oopnet.writer.writing_modules.write_network_components module

oopnet.writer.writing_modules.write_network_map_tags module

oopnet.writer.writing_modules.write_options_and_reporting module

oopnet.writer.writing_modules.write_options_and_reporting.**reportparameter2str**(*rp*)

Parameters
rp (Union[str, list[str, float]]) –

Return type
str

Returns:

oopnet.writer.writing_modules.write_options_and_reporting.**reportprecision2str**(*rp*)

Parameters
rp (int) –

Return type
str

Returns:

oopnet.writer.writing_modules.write_options_and_reporting.**timedelta2hours**(*td*)

Parameters
td (timedelta) –

Return type
str

Returns:

oopnet.writer.writing_modules.write_options_and_reporting.**timedelta2startclocktime**(*t*)

Parameters
t (timedelta) –

Return type
str

Returns:

oopnet.writer.writing_modules.write_system_operation module

oopnet.writer.writing_modules.write_water_quality module

Module contents

Submodules

oopnet.writer.decorators module

class oopnet.writer.decorators.**WriterDecorator**(*sectionname=None*, *functionname=None*, *priority=None*, *writerfunction=None*)

Bases: object

Class for saving the writer function properties in a proper way with the decorators

sectionname

functionname

priority

writerfunction

oopnet.writer.decorators.**make_registering_decorator_factory**(*foreign_decorator_factory*)

Parameters
foreign_decorator_factory –

Returns:

oopnet.writer.decorators.**section_writer**(**args*, ***kw*)

Synchronization decorator

Parameters

- **title** – section title
- **priority** – write priority

Returns:

oopnet.writer.module_reader module

oopnet.writer.module_reader.**list_section_writer_callables**(*modules*)

Lists all callables decorated with the section_writer decorator from a list of modules.

Parameters
modules – list of modules to be checked for callables decorated with section_writer.

Returns
List of callables decorated with section_writer.

oopnet.writer.module_reader.**pred**(*c*)

Checks if a class or function is decorated with section_writer.

Parameters
c (Type[Callable]) – callable to be checked

Return type
`bool`

Returns
Returns True if the class or function is decorated with section_writer and False otherwise.

oopnet.writer.write module

oopnet.writer.write.**write**(*network*, *filename*)

Converts an OOPNET network to an EPANET input file and saves it with a desired filename.

Parameters

- **network** (*Network*) – OOPNET network object which one wants to be written to a file
- **filename** (str) – desired filename/path were the user wants to store the file

Return type
`int`

Returns
0 if successful

Module contents

2.5 Development

2.5.1 CHANGELOG

v0.6.3 (2023-10-03)

Fix

- fix: fix issue of merged values when parsing report file (`4e249d6 <https://github.com/oopnet/oopnet/commit/4e249d650de0af038b7a731b4737294de1de29da>`_)

Unknown

- Merge pull request #48 from oopnet/fix/report_parsing_decimals

Fix report parsing decimals and deprecated pandas methods (`266dd0c <https://github.com/oopnet/oopnet/commit/266dd0c266c479f4ff7c0dcf105f9f6b4c545631>`_)

- replace deprecated pandas methods iterrows and applymap (`5c4dcb1 <https://github.com/oopnet/oopnet/commit/5c4dcb107362bb1f9043080074ae1590b05ab652>`_)

v0.6.2 (2022-10-22)

Fix

- fix: fixed minor bug in ReportFileReader (`43a8fca <https://github.com/oopnet/oopnet/commit/43a8fcafb3024d25604343d26c79a36e869092bc>`_)

v0.6.1 (2022-10-15)

Fix

- fix: fixed a bug where Valves in controls resulted in an invalid EPANET model

EPANET does not accept FCV/PCV/PRV/... in the IF line in controls but requires VALVE instead. The THEN line accepts VALVE however. Added a corresponding rule to Rules_network and. (`1c20957 <https://github.com/oopnet/oopnet/commit/1c20957cb8a665997f7379a9af561e6ea46222e7>`_)

Unknown

- Merge remote-tracking branch 'origin/main' (`a82d833 <https://github.com/oopnet/oopnet/commit/a82d833a16b2c9644110fff22d0a621f2d218f3b>`_)
- [skip ci] docs: fixed figure in plotting userguide (`1d6a008 <https://github.com/oopnet/oopnet/commit/1d6a008c9454c8a499479d0e55b3551daa62fcd3>`_)

v0.6.0 (2022-10-07)

Documentation

- docs: recreated plots with new, stacked color bars and added example for ax argument (`60bc691 <https://github.com/oopnet/oopnet/commit/60bc69179ab0cdee4614464a0192a1d1f288cab7>`_)
- docs: added missing argument documentation to pyplot NetworkPlotter.animate() (`6b29827 <https://github.com/oopnet/oopnet/commit/6b298270d8b1fe2850e1ab5b99f00b37a3d6c5d9>`_)

Feature

- feat: matplotlib colorbars are now stacked if both `nodes` and `links` arguments are passed (`6523fd8 <https://github.com/oopnet/oopnet/commit/6523fd806ea849a1382f8829529069fcbfccf676>`_)

Unknown

- Merge remote-tracking branch 'origin/main' (`b213f72 <https://github.com/oopnet/oopnet/commit/b213f725ecf940d7a21bbb1226b170c5e0abf41a>`_)
- [skip ci] docs: added missing figure in plotting userguide (`d5fe95a <https://github.com/oopnet/oopnet/commit/d5fe95a2b612b0f41f2008fcc516c867d18329c8>`_)
- Merge remote-tracking branch 'origin/main' (`bf520f5 <https://github.com/oopnet/oopnet/commit/bf520f5137a029f477e0448ebbfcf79d92884a37>`_)

- [skip ci] docs: renamed OOPNET to Object-Oriented Pipe Network Analyzer (`7a5682d <https://github.com/oopnet/oopnet/commit/7a5682d7f3bc1e987df77cc045ab6d0c64274ac2>`_)
- Merge remote-tracking branch 'origin/main' into plt_colorbars (`b489351 <https://github.com/oopnet/oopnet/commit/b4893511fc0d7ddd7b2106384571e43acbe3d558>`_)

v0.5.3 (2022-10-05)

Fix

- fix: bugfix in ReportFileReader

fixed another bug, where concatenated results are not parsed correctly (`2dad842 <https://github.com/oopnet/oopnet/commit/2dad8422c260974d81b9da04a60e21475ae3c6fd>`_)

v0.5.2 (2022-09-26)

Fix

- fix: ReportFileReader bugfix

fixed a bug where concatenated large values are not split correctly if they appear in the first line of a result block (`127f1ce <https://github.com/oopnet/oopnet/commit/127f1cec0a47a7ab1b6932a5701085be961f0a86>`_)

v0.5.1 (2022-09-22)

Ci

- ci: deactivated black formatting (`a69d815 <https://github.com/oopnet/oopnet/commit/a69d815c63d11a5c8daaefa310318da209057e56>`_)

Fix

- fix: fixed demand category bug
- fixed demand category reading and writing
- added test for demand category reading and writing
- added demand categories to Junction J-02 in Poulakis_enhanced_PDA.inp and updated results file (`e150b00 <https://github.com/oopnet/oopnet/commit/e150b00c63e3c1e159a089731a2b77566befa380>`_)

Test

- test: merged animation plotting tests into one to speed up tests (`8307053 <https://github.com/oopnet/oopnet/commit/83070534e3843a20f058e93fc97926b408775d11>`_)

Unknown

- [ci skip] updated feature request template

added example code block (`e30a98d <https://github.com/oopnet/oopnet/commit/e30a98d5b3b12ec25fe0be36f9c775b94aec01ed>`_)

- [ci skip] bug report template update

added example code block (`49dc49b <https://github.com/oopnet/oopnet/commit/49dc49bf1c82279301f66eb817176cb64c0e0c4b>`_)

- started fixing colorbar positioning and sizing (`1e7d7ba <https://github.com/oopnet/oopnet/commit/1e7d7bab82e767c6090a5ce1946c3a5bec684c66>`_)
- [ci skip] rtd bugfix (`5493ae0 <https://github.com/oopnet/oopnet/commit/5493ae00c607c7bc36bffb120f1bf77ebab31e4f>`_)
- [ci skip] rtd bugfix (`8fa7909 <https://github.com/oopnet/oopnet/commit/8fa7909e6f331eb33655c3581820539c31602995>`_)
- [ci skip] rtd bugfix (`dbdf000 <https://github.com/oopnet/oopnet/commit/dbdf00078411e4bf85477e6191a62528306d991b>`_)
- [ci skip] rtd bugfix (`cdf9676 <https://github.com/oopnet/oopnet/commit/cdf96764dd2ea7a7adddb34cdcd7dba795381515>`_)
- [ci skip] rtd bugfix (`21497a9 <https://github.com/oopnet/oopnet/commit/21497a96c5379206351d652d68de58dd0d01ad1d>`_)
- [ci skip] rtd bugfix (`1d8c4d2 <https://github.com/oopnet/oopnet/commit/1d8c4d217a5a6af8eb56d2b2e294d8d8a4657da0>`_)
- [ci skip] rtd bugfix (`79ab631 <https://github.com/oopnet/oopnet/commit/79ab63198dc1c31b0d07bb0d9348556305e3cf40>`_)
- [ci skip] rtd bugfix (`b788ca0 <https://github.com/oopnet/oopnet/commit/b788ca0f314bf8d1b43fb21005052f8e2e5bb58d>`_)
- [ci skip] added pre_install job to install EPANET in the rtd environment (`6655b31 <https://github.com/oopnet/oopnet/commit/6655b31d7f01559d9ae4533aaff686a96254fdef>`_)
- [ci skip] fix bokeh plot in docs
- docs: updated examples
- docs: created user guide from old examples
- fixed minor errors related to docstrings and type hints for the documentation
- added some missing example script tests
- added a documentation status badge to README.md
- started revising about.rst
- added missing extended_period_simulation.png
- added plotting figures, fixed bokeh plot visualization in sphinx docs
- fixed path to license file in README.md
- added startdatetime argument passing in simulation example
- some more doc updates, typo corrections, citations, …

- created dedicated bokeh plot script for docs (`13a2c15 <https://github.com/oopnet/oopnet/commit/13a2c15d7f10a17b3de101928c8d88ad6ee04c99>`_)
- [ci skip] Documentation update from docs branch
- docs: updated examples
- docs: created user guide from old examples
  - fixed minor errors related to docstrings and type hints for the documentation
  - added some missing example script tests
  - added a documentation status badge to README.md
  - started revising about.rst
- added missing extended_period_simulation.png
- added plotting figures, fixed bokeh plot visualization in sphinx docs
- fixed path to license file in README.md
- added startdatetime argument passing in simulation example
- some more doc updates, typo corrections, citations, ... (`d1127d8 <https://github.com/oopnet/oopnet/commit/d1127d8387b5e506f612b6e28cd2cb1c70aeb6e6>`_)

v0.5.0 (2022-09-18)

Documentation

- docs: renamed LICENSE to LICENSE.md to use markdown (`08a1e69 <https://github.com/oopnet/oopnet/commit/08a1e6958e4ff9444ba9d9f209893cd6630bc932>`_)

Feature

- feat: added a new NetworkPlotter class that enables static plots and animations
- replaced PlotSimulation, Plotnodes etc. with a single class for plotting
- implemented a new animation function to visualize extended period simulations
- added `animate` method to Network class
- added documentation and a new example run_and_animate.py base on L-Town
- added tests (`3a3a028 <https://github.com/oopnet/oopnet/commit/3a3a02840443009cffb2d9d271402bb4c7123855>`_)
- feat: added `center` method to Link components to calculate a Link's geometric center in 2D

center coordinates are derived from start and end nodes, as well as vertices does not take component's length attribute into account useful for plotting (`d3f5dbf <https://github.com/oopnet/oopnet/commit/d3f5dbff5f5e7f2e175206fa7a6b2d7fb020711f>`_)

Fix

- fix: switched SimulationReport property type hints to Union[pd.Series, pd.DataFrame] to take extended period simulations into account (`2ffcccc <https://github.com/oopnet/oopnet/commit/2ffcccc635849a9528fb76e46f7a64dacef4e99f>`_)

Refactor

- refactor: reformatted by black (`1666128 <https://github.com/oopnet/oopnet/commit/1666128b59e04ce341f26f956dbe5a9a81cdec61>`_)

Test

- test: fixed simulation tests by adding sort_index call (`f10671f <https://github.com/oopnet/oopnet/commit/f10671f716147046ef8876f74d814f3fac8fe0b2>`_)
- test: added test for Link center calculation (`493cfe2 <https://github.com/oopnet/oopnet/commit/493cfe2cad0c98d39537000039f30255b690af5b>`_)

Unknown

- Merge pull request #41 from oopnet/animation_plotting

Animation plotting Fixes #39 (`7a5bb8a <https://github.com/oopnet/oopnet/commit/7a5bb8aed272e260c2e09fb605cbabf8a856d327>`_)

- added scyipy to requirements-dev.txt (`ab1e02c <https://github.com/oopnet/oopnet/commit/ab1e02c7ebc69c057e3cc50e3d16b3502760f39d>`_)

v0.4.1 (2022-09-02)

Fix

- fix: added sort_index() call to pandas results (`53545ca <https://github.com/oopnet/oopnet/commit/53545ca621afce3b6f36663100e9228e306c788f>`_)

Test

- test: fixed simulator test (`33dce6c <https://github.com/oopnet/oopnet/commit/33dce6cb762be27b4dc76c879e3275010e2f05ba>`_)

v0.4.0 (2022-09-01)

Documentation

- docs: updated information in setup.cfg for PyPI (`e6f23b5 <https://github.com/oopnet/oopnet/commit/e6f23b52a28b0b10475179221a91c550d6e9064b>`_)
- docs: added badges to README.md (`d131b8a <https://github.com/oopnet/oopnet/commit/d131b8adbac0e361ca12809c2ee856cf7b834bee>`_)

Feature

- feat: added vertex plotting to Bokeh plot (`4ebe022 <https://github.com/oopnet/oopnet/commit/4ebe02261672f06268f9d666f869e62ffcb5b73c>`_)

Fix

- fix: switched condition attribute default of Rules from None to an empty list (`aad8f00 <https://github.com/oopnet/oopnet/commit/aad8f00bb9095cee65967e5f6ba52a03c0e15273>`_)
- fix: fixed reading time options in hh:mm:ss format (`fa2abef <https://github.com/oopnet/oopnet/commit/fa2abef883af201a91924b4396a7221ca2389caa>`_)

Unknown

- [ci skip] ci: switched to dedicated user (OOPNET-bot) for CI pipelines (`c58173b <https://github.com/oopnet/oopnet/commit/c58173b742671f52b93faf34d50b77740a84dd30>`_)

v0.3.2 (2022-08-26)

Ci

- ci: added check to skip pipeline execution if a commit was triggered by OOPNET-bot (`b9a2f21 <https://github.com/oopnet/oopnet/commit/b9a2f21ac00a6f3e69d4c4b390bb23634b61edfe>`_)

Documentation

- docs: corrected Network attribute documentation (`3a71dd9 <https://github.com/oopnet/oopnet/commit/3a71dd9c26ee3aae277b80cc524dc313d662dd8c>`_)

Fix

- fix: fixed concatenated large number parsing error in report file reader

fixes #34 (`1685835 <https://github.com/oopnet/oopnet/commit/1685835e1d16bf7eaa605e829486ddde07e4d5de>`_)

- fix: moved parts of report module to simulator module for better consistency (`41ca790 <https://github.com/oopnet/oopnet/commit/41ca7907043fd718c40fc0c5bf060bdba37be7e9>`_)

Unknown

- ci bugfix (`16ec2d7 <https://github.com/oopnet/oopnet/commit/16ec2d77a18d0bb8134973449a5c90cc7f5dc337>`_)
- ci bugfix (`fc7aa0a <https://github.com/oopnet/oopnet/commit/fc7aa0ae6d600f76e28e7092826831fe0ac44586>`_)
- ci bugfix (`fe85d50 <https://github.com/oopnet/oopnet/commit/fe85d507afac61cd4c4962034ad617d156e8006e>`_)
- ci bugfix (`3bec184 <https://github.com/oopnet/oopnet/commit/3bec184484613dea6e9cc93490db6db94c888010>`_)
- Merge pull request #35 from oopnet/docs

Large number parsing fix (`9111738 <https://github.com/oopnet/oopnet/commit/91117381185c7c42cf6e2f876f2097b2ae2dbbbf>`_)

- [ci-skip] added ReadTheDocs configuration yml (`6df531c <https://github.com/oopnet/oopnet/commit/6df531cde00598b5e84fe30c6bfdfe50ef6cd10b>`_)
- [ci-skip] added pull request template (`d50c962 <https://github.com/oopnet/oopnet/commit/d50c96219c2c459b969d2675678c9c2452f952e4>`_)
- [ci-skip] added README.md prototype (`f0602c2 <https://github.com/oopnet/oopnet/commit/f0602c2e3a626045f67b9d8f2db3817d19389ffc>`_)
- [ci-skip] added checks to bug report template (`0890afd <https://github.com/oopnet/oopnet/commit/0890afdcb902188a6694937063d6390461a34bbe>`_)
- [ci-skip] Update issue templates (`fda2548 <https://github.com/oopnet/oopnet/commit/fda254804a113fe768b9c70dc7ac357a69711776>`_)
- Automated changes (`a4426f0 <https://github.com/oopnet/oopnet/commit/a4426f0788015f7175302c56926aee136c4842e1>`_)
- [ci-skip] docs: added doc building requirements to requirements-dev.txt (`3dfa65f <https://github.com/oopnet/oopnet/commit/3dfa65f0dfe9b0f62b5d2d1bbd4c04811966913e>`_)
- [ci-skip] doc refactoring (#29)
- [ci skip] added CONTRIBUTING.md and CODE_OF_CONDUCT.md
- [ci skip] added link to code of conduct to CONTRIBUTING.md
- [ci skip] updated CONTRIBUTING.md
- [ci skip] updated email addresses
- [ci skip] docs: documentation refactoring
- switched from readthedocs to mkdocs-like documentation
- restructured documentation layout
- restructured documentation files
- moved old into new docs

- corrected mistakes in a few examples
- switched LICENSE to LICENSE.md to include it in the docs
- removed old doc directory (`91f4ade <https://github.com/oopnet/oopnet/commit/91f4ade986010ad6eeb4f0748fad6e87dc175f09>`_)
- [ci skip] updated email address (`e2beb60 <https://github.com/oopnet/oopnet/commit/e2beb607d33281025fb921803ae329464a9602b3>`_)
- [ci skip] adapted code of conduct (`4bdcd41 <https://github.com/oopnet/oopnet/commit/4bdcd419a29ee58cf8c5de451da6f972b32374d8>`_)
- Dev (#27)
- [ci skip] added CONTRIBUTING.md and CODE_OF_CONDUCT.md
- [ci skip] added link to code of conduct to CONTRIBUTING.md
- [ci skip] updated CONTRIBUTING.md (`9881bd2 <https://github.com/oopnet/oopnet/commit/9881bd29b10d1e13ce584f1868981d4162935dcb>`_)
- [ci skip] added CONTRIBUTING.md and CODE_OF_CONDUCT.md (`763e086 <https://github.com/oopnet/oopnet/commit/763e086dbcfb9f4dd89a1b89ad40a5ace058d162>`_)
- [ci skip] Update issue templates (`72dc04d <https://github.com/oopnet/oopnet/commit/72dc04de16a41062128b53e43f4581263ab9281d>`_)

v0.3.1 (2022-04-14)

Fix

- fix: fixed a bug where networkx get_edge_data returning a dict instead of a list breaks nxedge2onlink_id (`8377d16 <https://github.com/oopnet/oopnet/commit/8377d16a09b0e9a23820db2af7820fa627242a1f>`_)

v0.3.0 (2022-02-28)

Ci

- ci: fixed pipeline (`635fe88 <https://github.com/oopnet/oopnet/commit/635fe88511e4c6fccdeab9ce9b18533096eecc2a>`_)

Feature

- feat: added get_by_id method to SuperComponentRegistry for NetworkComponent lookup

get_by_id is a utility function for iterating over all ComponentRegistries stored in a SuperComponentRegistry. It is implemented in get_nodes(network) and get_links(network). (`d89c8cc <https://github.com/oopnet/oopnet/commit/d89c8cc88bb7ab979ec73a0a7b8e3d18c1c40d49>`_)

Fix

- fix: renamed Tank attribute diam to diameter
- adapted tests, writer, reader … (`0f6801d <https://github.com/oopnet/oopnet/commit/0f6801d03d755ba2e5ef85773309a2e0af5fed03>`_)
- fix: fixed some type hints

Added lists to the appropriate class attributes (`91b3a6e <https://github.com/oopnet/oopnet/commit/91b3a6eda9fae12a07e36adcbb91b527cab3c9ed>`_)

Unknown

- Automated changes (`9283b91 <https://github.com/oopnet/oopnet/commit/9283b91964bd5bda6f8e30218010541cfd527be4>`_)

v0.2.3 (2022-02-22)

Fix

- fix: fixed ComponentRegistry pickling

quick fix to prevent pickling error of ComponentRegistries (object of type ComponentRegistry has no attribute super_registry) (`fbe2900 <https://github.com/oopnet/oopnet/commit/fbe290064dd353fdca969630cbfa6acac525c106>`_)

Test

- test: added tests for pickling Networks and SimulationReports (`dfbc9f2 <https://github.com/oopnet/oopnet/commit/dfbc9f27aa38ca83b123c1578fd1505cddb30d8e>`_)
- test: added tests for adding different component types with same ID

test for e.g. adding a Junction with ID "1" to a Network that already contains a Tank with the ID "1" (`3fd5818 <https://github.com/oopnet/oopnet/commit/3fd58185b369a21b6243e194cb734181eef1ba93>`_)

v0.2.2 (2022-02-22)

Ci

- ci: switched to OOPNET_SECRET for pushing changes (`851a087 <https://github.com/oopnet/oopnet/commit/851a08755b8ba0a22029849f957cab452791f7b0>`_)

Fix

- fix: disabled testing the mc stereo scoop example

SCOOP isn't working with Python 3.10 as described in this issue (`003f249 <https://github.com/oopnet/oopnet/commit/003f249099a40d6b5c647571c0b8ef48f3be8fca>`_)

- fix: fixed CI pipeline

replaced Python version 3.10 with "3.10" to prevent trimming the version to 3.1 (`50f2650 <https://github.com/oopnet/oopnet/commit/50f2650ea56249cb58d247210d41443c2d194283>`_)

- fix: fixed CI pipeline

CI pipeline now takes all commits since last release instead of latest only added tests for Python 3.10 minor changes to semantic release secrets (`b79726c <https://github.com/oopnet/oopnet/commit/b79726cbda8e37cd1cd9c675146f3b43cf67b15a>`_)

- fix: fixed ComponentRegistry initialization

removed dataclass decorator since it could lead to issues (`1cd7d64 <https://github.com/oopnet/oopnet/commit/1cd7d64184856de6b2633a1ceda4dbc2038e3ebe>`_)

Refactor

- refactor: minor changes to component registries and network annotations (`cdb7471 <https://github.com/oopnet/oopnet/commit/cdb74711c776e82f9f13cd1793a2b1fa7a9a79c5>`_)

Test

- test: added some new tests (patterns, curve, deepcopy)
- added model for curve and pattern testing and added some tests
- wrote some deepcopy tests (`1240cba <https://github.com/oopnet/oopnet/commit/1240cba5e0a652e13bfb07f0715dea8e41cc8ac9>`_)

Unknown

- Automated changes (`d00fb36 <https://github.com/oopnet/oopnet/commit/d00fb360d94fb6ed5d1b5a3a9dee922798fc9901>`_)

v0.2.1 (2022-02-17)

Fix

- fix: fixed writing Tank volumecurves (`960a232 <https://github.com/oopnet/oopnet/commit/960a23287042eb8f20012748c4fbbca38f959a7e>`_)

v0.2.0 (2022-02-14)

Feature

- feat: added linkwidth argument to Network.plot

Linkwidth takes a pandas Series with values describing the width of specific Pipes (!). Added tests for some other plotting arguments as well. (`f749a48 <https://github.com/oopnet/oopnet/commit/f749a4803e5583bea126791ed2c54e28a1059b6f>`_)

Fix

- fix: fixed pandas Series for links with missing values

Missing values from a link pandas Series don't lead to an error anymore when looking up the color of the missing Link. Black will be used instead. (`cc681f4 <https://github.com/oopnet/oopnet/commit/cc681f440ee9393691ab41ffcdbdebbdcc61907d>`_)

- fix: added ComponentExistsError to **init**.py and renamed to IdenticalIDError (`cde55a0 <https://github.com/oopnet/oopnet/commit/cde55a01e051b5cb62190dcb5fc6c33629d10290>`_)
- fix: fixed Pipe split function

Fixed pipe length calculation, added validation for split_ratio argument and added logging (DEBUG level). Added tests for invalid split_ratio arguments and enhanced existing tests. (`7e3b53a <https://github.com/oopnet/oopnet/commit/7e3b53a4b92a74b5cae46bc33a9cd0d11f143bb6>`_)

Refactor

- refactor: refactored benchmark.py to incorporate new Network API (`4f5d9bd <https://github.com/oopnet/oopnet/commit/4f5d9bde11134cf6a50b3bec1cd0439a3ae18544>`_)

v0.1.6 (2022-02-07)

Ci

- ci: added dedicated pull request workflow

Added dedicated PR workflow to prevent building OOPNET when adding a PR (`dc9549a <https://github.com/oopnet/oopnet/commit/dc9549a1adb3a939ab61cc3d21e1586be926a3bf>`_)

- ci: added dedicated pull request workflow

Added dedicated PR workflow to prevent building OOPNET when adding a PR (`2ae988c <https://github.com/oopnet/oopnet/commit/2ae988c02b893fd415e7c3d6b09997c4bdba9bb3>`_)

Fix

- fix: minor changes to setup.cfg to trigger release (`18b3df2 <https://github.com/oopnet/oopnet/commit/18b3df2a166bd0ae575b128d0f5753755428acc1>`_)
- fix: minor changes to setup.cfg to trigger release (`4456b9e <https://github.com/oopnet/oopnet/commit/4456b9e2b63d702f19f8bb44aadc3c0d1867f12f>`_)
- fix: fixed Network creation from strings

added tests for contentreading (`a5afc67 <https://github.com/oopnet/oopnet/commit/a5afc675b8901c3b3fe950982610162f00af026c>`_)

- fix: Headloss of Pumps is now correctly returned (`c53f22c <https://github.com/oopnet/oopnet/commit/c53f22c122ebce31d473771eab92f4dd1f678913>`_)

Refactor

- refactor: removed vertices from Network (`39c5fe1 <https://github.com/oopnet/oopnet/commit/39c5fe1550e74ef2992d53db0b22108b09c06266>`_)
- refactor: set explicit dtypes for pandas Series in property_getters.py

silencing pandas warning (`f19fb16 <https://github.com/oopnet/oopnet/commit/f19fb16113256810eba3e96c0d1f57437a100611>`_)

Test

- test: added missing Poulakis_enhanced_PDA.xlsx for SimulatorTests (`be5774a <https://github.com/oopnet/oopnet/commit/be5774a3fc902a2c93603a4bbfe2a4da3577a7c9>`_)
- test: added missing Poulakis_enhanced_PDA.xlsx for SimulatorTests (`50714a7 <https://github.com/oopnet/oopnet/commit/50714a791c6baf89f0d19bd18168418168b9de76>`_)
- test: added SimulatorTest for PoulakisEnhancedModel (`9c084a8 <https://github.com/oopnet/oopnet/commit/9c084a8d93460b309fecec46ba94ceaf93935538>`_)

Unknown

- Merge pull request #22 from oopnet/dev

Dev (`ac64888 <https://github.com/oopnet/oopnet/commit/ac64888bf5cb3b7673086272dab4b2153a0a9c16>`_)

- Automated changes (`bb6aecb <https://github.com/oopnet/oopnet/commit/bb6aecb3ad4d92e6425766eace0ffa2ee6fa389b>`_)
- Automated changes (`9f0dca0 <https://github.com/oopnet/oopnet/commit/9f0dca018b0025afa990e1ed47b2728acbf49c3f>`_)
- Merge pull request #21 from oopnet/dev

Dev (`0799437 <https://github.com/oopnet/oopnet/commit/0799437ab2b2792a23eaa016aadc2c92efc0dc18>`_)

v0.1.5 (2022-02-07)

Fix

- fix: fix CI (`98053e7 <https://github.com/oopnet/oopnet/commit/98053e7775ce0d87a239ea991e4905070f8a463f>`_)
- fix: fix CI (`460c190 <https://github.com/oopnet/oopnet/commit/460c190421c1ebb44e953d65fa3fcc322b7af565>`_)
- fix: fix CI (`3f655eb <https://github.com/oopnet/oopnet/commit/3f655eb0a92b15a967aa4272bd5e6037a9cd3718>`_)
- fix: fix CI (`4d38518 <https://github.com/oopnet/oopnet/commit/4d385188b43f84044f27a31914ffae31d6a943d9>`_)
- fix: another setup.cfg fix + slight CI changes (`8eb5f38 <https://github.com/oopnet/oopnet/commit/8eb5f3842c9e85051684c2ac6404940869ae32c5>`_)

v0.1.4 (2022-02-06)

Fix

- fix: another setup.cfg fix (`40e8eb8 <https://github.com/oopnet/oopnet/commit/40e8eb81d6a87839029de4566861539cd6d067c6>`_)

v0.1.3 (2022-02-06)

Fix

- fix: moved requirements from requirements.txt to setup.cfg to enable autoinstall of missing packages (`eb119cd <https://github.com/oopnet/oopnet/commit/eb119cd954d35f4ad49b1625651d3a2167e5c7a7>`_)

v0.1.2 (2022-02-06)

Fix

- fix: fixed setup.cfg package_data (`c3f7122 <https://github.com/oopnet/oopnet/commit/c3f71223641b51a31b4e6e9282436480190a598b>`_)

v0.1.1 (2022-02-06)

Fix

- fix: fixed setup.cfg license (`4accf2d <https://github.com/oopnet/oopnet/commit/4accf2d1038e4755bb5ec64b97c082f23376f2eb>`_)

v0.1.0 (2022-02-06)

Documentation

- docs: some fixes in various docstrings (`c8a02df <https://github.com/oopnet/oopnet/commit/c8a02df63cb53865fee6525ea7e5932c9b65e55d>`_)

Feature

- feat: nonsense to create new release (`617dee9 <https://github.com/oopnet/oopnet/commit/617dee993de458a42b374bc61a3eee834f699beb>`_)
- feat: added SimulatorTest for C-Town model
- Friction Factors and Status are not checked currently
- only using feat to trigger a new release for testing purposes (`9c410d9 <https://github.com/oopnet/oopnet/commit/9c410d926c7ef85b2b63fdf04d705ba5df2f9c7e>`_)

Refactor

- refactor: switched Valve settings to individual attributes like maximum pressure
- adapted reader, writer and tests (`d9853b2 <https://github.com/oopnet/oopnet/commit/d9853b22f845279cb07d50f30fd6d53587a188ec>`_)

Test

- test: fixed data reading for SimulatorTests (`c44d8b7 <https://github.com/oopnet/oopnet/commit/c44d8b7fab80f1fc5d8accb5f6c701f9f1af20fb>`_)
- test: fixed data reading for SimulatorTests (`9d1dae7 <https://github.com/oopnet/oopnet/commit/9d1dae7a06bd2e4b5d8c99c5c8edf549cb4f97a8>`_)
- test: fixed exception tests (`bd60b37 <https://github.com/oopnet/oopnet/commit/bd60b3737b97591d584ef85be1175256beb1e494>`_)
- test: fixed exception tests (`7df91e7 <https://github.com/oopnet/oopnet/commit/7df91e781685c31568dce8f62a7f3520bec9bdbf>`_)

Unknown

- fixed data reading for SimulatorTests (`114c945 <https://github.com/oopnet/oopnet/commit/114c945a4194a284db63a229a5955ac51efb7cac>`_)
- Automated changes (`63996ed <https://github.com/oopnet/oopnet/commit/63996eda2a4e53f7a22c607f6105ae302b78b4ab>`_)
- Update setup.cfg (`7d96993 <https://github.com/oopnet/oopnet/commit/7d96993a865814a11ae7995b426c6c791d115b64>`_)
- added output=True for model simulation tests for better debugging (`5bbca1d <https://github.com/oopnet/oopnet/commit/5bbca1df8b122d380ab7b66ae889dfded8fae1c1>`_)
- Update build.yml (`65708c5 <https://github.com/oopnet/oopnet/commit/65708c5ed772cb608174e26aade40aabdc3bde2f>`_)
- Update build.yml (`ead3dad <https://github.com/oopnet/oopnet/commit/ead3dade1276ab9f09557f853aff2857bed8711c>`_)

- added requirements-dev.txt (`b28d2d1 <https://github.com/oopnet/oopnet/commit/b28d2d16157a07817ff5da1f27de1c18694f932e>`_)
- Update build.yml (`1aa7441 <https://github.com/oopnet/oopnet/commit/1aa744133d6b9c802a5276d15da846f060ff348b>`_)
- Update build.yml (`2ce70e3 <https://github.com/oopnet/oopnet/commit/2ce70e3608d54a648d9dfb20e79cacac62bfa030>`_)
- Update and rename python-app.yml to build.yml (`1febe93 <https://github.com/oopnet/oopnet/commit/1febe93e14d093c9f184e1f0092245a273073616>`_)
- Automated changes (`c7e5f5e <https://github.com/oopnet/oopnet/commit/c7e5f5e64a9e77d7ab26c296d7a5f9ab5ec0b3c9>`_)
- refactor
- changed setup settings (`07741d5 <https://github.com/oopnet/oopnet/commit/07741d59b48bed7593e7348722a54b70202622c8>`_)
- refactor
- changed setup settings (`9117ddb <https://github.com/oopnet/oopnet/commit/9117ddba0edcb31e7dbed595a518dcb812d711da>`_)
- refactor
- changed setup settings (`3a5a197 <https://github.com/oopnet/oopnet/commit/3a5a197f7c9c919a3ce986ed01a517074a1697ee>`_)
- refactor
- changed setup settings (`2c3d732 <https://github.com/oopnet/oopnet/commit/2c3d732dac9e99bd541f5d183f544782fef451f1>`_)
- Automated changes (`7b748a5 <https://github.com/oopnet/oopnet/commit/7b748a59a5a83e096c9f0c23198b5a4e1a352fd9>`_)
- refactor
- changed setup settings (`4cdf32b <https://github.com/oopnet/oopnet/commit/4cdf32bce30484f0a771121f95c38f77a13d09ac>`_)
- refactor
- changed setup settings (`4d165f5 <https://github.com/oopnet/oopnet/commit/4d165f5455f707b1cb6625c29b48176c6fb299a1>`_)
- refactor
- added sematic_release section to setup.cfg
- added some classifiers to setup.cfg (`9511323 <https://github.com/oopnet/oopnet/commit/9511323ca9f9ca9034e65404b26dacbe36e52c70>`_)
- Automated changes (`9dbd67f <https://github.com/oopnet/oopnet/commit/9dbd67fc10e479ec869985dad25c95a5ec455564>`_)
- refactor
- moved build settings to setup.cfg (`f9f5bf6 <https://github.com/oopnet/oopnet/commit/f9f5bf6936d52c21052a0f88329e49cdb720c0aa>`_)
- Update python-app.yml (`ad0a46a <https://github.com/oopnet/oopnet/commit/ad0a46a8c0c9126ff83282c8f870181f91b207bb>`_)
- refactor

- moved **version** to setup.py (`7370a91 <https://github.com/oopnet/oopnet/commit/7370a91550cd8f472f9dac6aa9e08c773d84e8e9>`_)
- Update python-app.yml (`a24cd45 <https://github.com/oopnet/oopnet/commit/a24cd457185e22ca6690917995694dbf51f59220>`_)
- refactor
- disabled some setup.py settings for now
- added seaborn to requirements again
- fixed some tests
- removed graph attribute from Network (`6a1cffe <https://github.com/oopnet/oopnet/commit/6a1cffeefb5b848fcb90ca2fdcf8228479a69179>`_)
- Update python-app.yml (`62d1a92 <https://github.com/oopnet/oopnet/commit/62d1a92439a59bfe002bd80c976991507eeb1f8b>`_)
- Update python-app.yml (`f5c5e74 <https://github.com/oopnet/oopnet/commit/f5c5e749fab831fb4f83cd4e35cb88405c507af8>`_)
- Update python-app.yml (`d70a457 <https://github.com/oopnet/oopnet/commit/d70a4575376fb0a4f1665643b46e1d4086e624b6>`_)
- Update python-app.yml (`87904cb <https://github.com/oopnet/oopnet/commit/87904cb86342433af110c4217669ae35a5078376>`_)
- Create python-app.yml (`0d76bcb <https://github.com/oopnet/oopnet/commit/0d76bcb4bf3e00f777dced16d0ee7286fb7d411e>`_)
- Delete build_deploy.yml (`5b4fe80 <https://github.com/oopnet/oopnet/commit/5b4fe80a02b6d4e90d49eeafea72fd119547b863>`_)
- Create build_deploy.yml (`4595a59 <https://github.com/oopnet/oopnet/commit/4595a595ae914eee8fe34b6d3f462f761791fb32>`_)
- refactor
- refactored Pump attributes (dropped keyword value scheme)
- created plot() und bokehplot() instance methods for Network objects
- removed NetworkComponent hash method
- renamed Report to SimulationReport
- added some docs
- fixed helper functions in graph.py
- added get_inflow_nodes and get_inflow_node_ids functions
- refactored special_getters.py to topology_getters.py
- cleaned up utils.py
- some minor fixes
- removed traits and seaborn from requirements.txt (`0643495 <https://github.com/oopnet/oopnet/commit/06434957f30b4bafd1323880b494ef686568bcbd>`_)
- refactor
- added Vertices (reading, writing, plotting)
- added split function to Pipes

- added coordinates_2d property to Links
- fixed some imports
- finished Report class
- removed report_getter_functions.py with all functions
- added first (very basic) plotting test
- removed api.py
- removed pandasreport.py
- removed length function and adddummyjunction from utils.py (`ac9dd23 <https://github.com/oopnet/oopnet/commit/ac9dd238f05b9c661d969c49fff1dee1e73279eb>`_)
- refactoring
- implemented Report class with properties for flow, pressure, etc.
- adapted tests and examples accordingly (`ab6daf5 <https://github.com/oopnet/oopnet/commit/ab6daf5e02528de2b758726909da8b81825ac5fc>`_)
- refactor
- made `run`, `read` and `write` class/instance methods for `Network` class/objects and removed `Read`, `Write` and `Run` factories from `__init__.py` files
- refactored imports to prevent circular imports
- adapted examples and tests (`7933aa1 <https://github.com/oopnet/oopnet/commit/7933aa19fd87f82da246ce003835995a5fd43337>`_)
- refactor
- added setting description to Valve subclasses
- remove valve_type attribute (adjusted reader, writer and tests accordingly)
- enabled passing a string to read
- minor changes to benchmark stuff that we can also do in WNTR
- added a reset method to the benchmark (`a4d7027 <https://github.com/oopnet/oopnet/commit/a4d7027e6232cc0df5c1c3694bab4e3256fc3715>`_)
- Merge pull request #19 from oopnet/refactor

Refactor (`26cd804 <https://github.com/oopnet/oopnet/commit/26cd80401e771486555508d5a89461997596bf8e>`_)

- refactoring
- switched import statements in examples to `import oopnet as on` (issue #15) (`1842bdc <https://github.com/oopnet/oopnet/commit/1842bdcb86e77a5fbfa3acb92ac4ed49fee5f7bd>`_)
- refactoring
- added MultiDiGraph test (`1614c9e <https://github.com/oopnet/oopnet/commit/1614c9e348bbff80a47d0b3bd1df8f3a3d4abb24>`_)
- refactoring
- added MultiDiGraph to graph's module **init**.py (`1f55824 <https://github.com/oopnet/oopnet/commit/1f558242fec37f42b3f58842bc2295404cd105ed>`_)
- refactoring

- added ComponentRegistry and SuperComponentRegistry classes to handle NetworkComponent storage in Network objects
- moved check_id_exists functionality to ComponenRegistry
- adapted getters, adders, removers and tests
- removed check_exists argument from adders (everything will be checked
- added rename method to NetworkComponents
- (`ccce9e5 <https://github.com/oopnet/oopnet/commit/ccce9e5e96d46dfbead9cb4144a9592b3c9cf2a2>`_)
- refactoring

relates to issue #17

- added MultiDiGraph factory
- added warnings to Graph and DiGraph factories (`9a56229 <https://github.com/oopnet/oopnet/commit/9a5622924398349 5f2212f61696b3a5e1c137066>`_)
- refactoring
- added logging
- added timer decorator
- added tests for example scripts
- some docstring improvements
- added examples for error handling and logging
- removed inititialstatus attribute
- removed **deepcopy** from network
- renamed network hashtables to make them private attributes
- improved EPANET error handling
- moved some reader stuff around
- added some missing default values for component attributes
- fixed Network class with fields as attributes
- fixed some tests
- fixed types of TestModel component attributes
- fixed examples (`b90f5bc <https://github.com/oopnet/oopnet/commit/b90f5bc912fb270fe2db97f73e4f315a073d18d4>`_)
- refactoring
- removed enums and all traces of them
- switched _component_hash attr to not be included in equality checks
- bugfix for component IDs
- switched everything from `initialstatus` to `status`
- fixed status writing
- added very simple Writer test (`f76b875 <https://github.com/oopnet/oopnet/commit/f76b8757879935be8133569464ffd86b72a1b54f>`_)
- refactoring

- worked over utils.py (`752c1e3 <https://github.com/oopnet/oopnet/commit/752c1e31b5c17f570065205912a00b3183ab6c8e>`_)
- refactoring
- removed enums again
- redid some type hints (switched from e.g., Dict to dict)
- added some documentation
- updated some **init**.py
- worked over bokehplot.py
- fixed some testing bugs (`2d0cbda <https://github.com/oopnet/oopnet/commit/2d0cbdaa251a1389ad0bf01da50f1d42bd46d40d>`_)
- refactoring
- refactored examples
- added placeholder for report settings example
- added test for examples (`2b5eec1 <https://github.com/oopnet/oopnet/commit/2b5eec1d8a8f162870b578838f2c65c219ce9b59>`_)
- refactoring
- removed redundant import in utils.**init**.py
- added pandas Series weights to Graphs
- added Graphs to API docs
- small changes to network_components.py
- added docs to ComponentReader factory classes
- corrected type hints for pandas getters in report_getter_functions.py
- added Graph tests (`db21e40 <https://github.com/oopnet/oopnet/commit/db21e404f600866e2201a4616cf684bccf5d88c6>`_)
- refactoring
- started refactoring input file reader (`ad9e2ca <https://github.com/oopnet/oopnet/commit/ad9e2ca378609ab34ac42c72f4e645a9374f9b6e>`_)
- refactoring
- started implementing BinaryFileReader (`bfc9983 <https://github.com/oopnet/oopnet/commit/bfc9983f829f4dfe13c22a1871e9984d64dbd4ee>`_)
- refactoring
- removed NetworkComponent **str** method
- writer and simulator bugfixes (due to enums)
- added CTown TestingModel (`1354653 <https://github.com/oopnet/oopnet/commit/13546537ad909dceb9360b2d5e8bdc4d570014c4>`_)
- refactoring

added plotting to benchmark (`6bb4aea <https://github.com/oopnet/oopnet/commit/6bb4aeaf38d2807e49a3f60169bb4b894423338b>`_)

- refactoring

- flattened reader and writer directories
- redid all **init**.py files for proper structure + changed import statements accordingly
- used typing TYPE_CHECKING to circumvent circular imports due to type checking
- moved enums to enums.py (for now) (`d96beda <https://github.com/oopnet/oopnet/commit/d96beda3d427e0447315d9ac3af0db9ce8c97cca>`_)
- refactoring
- fixed bug that converted 12 am to 12 pm in time settings
- replaced try/except clause in read_options
- refactored read_system_operation.py
- refactored write_options_and_reporting.py a little
- added and removed some todos
- added new TestModel for rule testing
- added tests for rules, curves, options, report, times and patterns (`a5fb570 <https://github.com/oopnet/oopnet/commit/a5fb57041093cdec19704c1cf2a2220dd5096d41>`_)
- refactoring
- implemented Enums for strings like 'CLOSED', 'OPEN', 'YES', ...
- adapted tests, reader and writer accordingly (`dbef38b <https://github.com/oopnet/oopnet/commit/dbef38b6fa20a3bb16f1361796e73a4041b94a55>`_)
- refactoring
- made type hints for list getters for informative
- fixed simulation errors (`71c3cec <https://github.com/oopnet/oopnet/commit/71c3cec029d1071671fbb1d4e14ff40d23a6516f>`_)
- refactoring
- renamed ComponentExistsException to ComponentExistsError (`65d84b8 <https://github.com/oopnet/oopnet/commit/65d84b869ead6f1f306a668aa81a31ba62007e9a>`_)
- refactoring
- added first enum for testing (`f94e89c <https://github.com/oopnet/oopnet/commit/f94e89ce12a85101946a03f69c453e7056919cc5>`_)
- refactoring
- fixed EPANET simulation error catching (`a37ae3e <https://github.com/oopnet/oopnet/commit/a37ae3e37bf652d62e4de2d976c50811ac0a48a6>`_)
- refactoring
- renamed ComponentExistsException to ComponentExistsError
- added Errors and ErrorManager for failed Simulations + tests (`085af79 <https://github.com/oopnet/oopnet/commit/085af79c3af3d3438a49d15b1241d81070df7da1>`_)
- refactoring
- removed dataclass slots decorators (`7ac33ee <https://github.com/oopnet/oopnet/commit/7ac33ee3dd1bf2396b994d79e313f5dda2bccace>`_)
- refactoring

- added setting attribute to pumps (= pump speed)
- adapted get_initialstatus function
- adapted get_setting function
- adapted v_diameter function (+ some docstring corrections)
- adapted property_getter tests accordingly (`605b078 <https://github.com/oopnet/oopnet/commit/605b078d486c78e3d4124692c752fdbda20d8d19>`_)
- refactoring
- some more graph.py refactoring
- fixed ExistingModelTest unittests (`2bc6db3 <https://github.com/oopnet/oopnet/commit/2bc6db31e30604c89b653c1127a32de92085b8e0>`_)
- refactoring
- added adder function for rules
- removed keyword args for adders
- added getters for rule IDs, fixed get_rules
- switched to dictionaries for storing rules
- fixed converter
- added MicropolisModel for testing
- fixed type hints for Times
- fixed reader for patterns, rules
- added defaults for system operations class attributes
- some test refactoring (`352cfbc <https://github.com/oopnet/oopnet/commit/352cfbcc903ae9c1e53b55660b179334a0b1e871>`_)
- refactoring
- fixed component ID setter
- disabled slots for OOPNET to work with python 3.9 again (`70f62ce <https://github.com/oopnet/oopnet/commit/70f62ce0ed41cdd59828e8d5e227ec5615917efe>`_)
- refactoring
- added revert method to Links
- added Link test (reverting and renaming)
- fixed Graphs
- added Graph creation to benchmark.py (`d032fa2 <https://github.com/oopnet/oopnet/commit/d032fa2a6026806c99833761a7432ee504b96b5f>`_)
- refactoring
- fixed some bugs
- refactored graph.py
- created exceptions.py (`af08d00 <https://github.com/oopnet/oopnet/commit/af08d0010cc4da41152cfba45f864be8da8ce655>`_)
- refactoring

- made add_element.py more efficient and reduced boilerplate code
- switched (hopefully) all iterations in Read to element getter functions
- added slots=True to dataclasses (requires python 3.10)
- refactored NetworkComponents to simplify ID property setters
- added some element list getters
- minor refactoring of graph functions
- minor refactoring of pyplot.py
- minor fixes
- removed some checks like `if network.junctions` since junctions is always instantiated
- removed sortedcontainers from requirements.txt
- removed `is not None` where they don't make a difference
- disabled bulk, wall and tank writing for Reactions (`d9dd48e <https://github.com/oopnet/oopnet/commit/d9dd48e03b1ecbb3af0aaf80773fbd47c7ccbca8>`_)
- update (`a545441 <https://github.com/oopnet/oopnet/commit/a54544192d36c6992179148d9f61b07177d98cc5>`_)
- worked over tests and added a benchmark (`d39dbc7 <https://github.com/oopnet/oopnet/commit/d39dbc78f0caefa127d7874b3365808132ac1631>`_)
- worked over examples (`0c17cf3 <https://github.com/oopnet/oopnet/commit/0c17cf3801766d8418a910cff5a3dd29cfd974e4>`_)
- refactoring
- switched to Google Docstrings
- redid adders and removers
- used adder functions in reader
- added some docs
- created ModelSimulator class
- added type hints
- removed `if network.junctions` and related checks
- added some todo comments
- worked over imports
- add sortedcontainers to requirements
- worked over setup.py (slightly) (`aa2d292 <https://github.com/oopnet/oopnet/commit/aa2d292e4ac8c97ae72da4367e305d8a39e3b937>`_)
- moved part of the function imports to dedicated **init**.py files (`da94125 <https://github.com/oopnet/oopnet/commit/da94125501336012d1713b9020438b7dba30d5d9>`_)
- replace epanet2d.exe (EPANET 2.0) with runepanet.exe (EPANET 2.2) (`6baebdf <https://github.com/oopnet/oopnet/commit/6baebdfb9ff752a83980d5699c06e14d49a46e62>`_)
- remove traits, implement dataclasses and type hints instead, switched to Epanet 2.2, replace networkhash, add a few additional getters (`feea28d <https://github.com/oopnet/oopnet/commit/feea28d928f15aeebfcc23e8952616485f1e165f>`_)

- move part of tests from GitLab branch epanet2_2 (`6927d20 <https://github.com/oopnet/oopnet/commit/6927d20437d7d0604fc70886fed0659693a90631>`_)
- initial commit of oopnet code of brachn python3 from gitlab.com (`4d44a36 <https://github.com/oopnet/oopnet/commit/4d44a3669069137c6e1f1bdffd3f3a72f7996d43>`_)
- added mac specific files to gitignore (`e70ab2f <https://github.com/oopnet/oopnet/commit/e70ab2f44bb6892b47ecdb71a54e25d347f5b388>`_)
- Update README.md (`bed7434 <https://github.com/oopnet/oopnet/commit/bed7434aabadd70d881bfe82782ed0dfb897ab12>`_)
- Initial commit (`923eb1d <https://github.com/oopnet/oopnet/commit/923eb1d77063a6084a4a30fbb6e4885244e0080e>`_)

2.5.2 Contributor Covenant Code of Conduct

Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at oopnet.contact@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

2.5.3 Contributing to OOPNET

We love your input! We want to make contributing to this project as easy and transparent as possible, whether it's:

- Reporting a bug
- Discussing the current state of the code
- Submitting a fix
- Proposing new features
- Becoming a maintainer

Following these guidelines helps to communicate that you respect the time of the developers managing and developing this open source project. In return, they should reciprocate that respect in addressing your issue, assessing changes, and helping you finalize your pull requests.

Our Development Process

We use Github Actions, so all code changes happen through pull requests (PRs). Pull requests are the best way to propose changes to the codebase. We actively welcome your pull requests:

1. Fork the repo and create your branch from `main`.
2. If you've added code that should be tested, add tests.
3. If you've changed APIs, update the documentation.
4. Ensure the test suite passes.
5. Make sure your code lints.

6. Issue that pull request!

Report bugs

We use Github's issues to track public bugs. Report a bug by opening a new issue; it's that easy!

For us to be able to fix a bug, we require information about the bug. **Great Bug Reports** tend to have:

- A quick summary and/or background
- Steps to reproduce
  - Be specific!
  - Give sample code if you can
- What you expected would happen
- What actually happens
- Notes (possibly including why you think this might be happening, or stuff you tried that didn't work)

People *love* thorough bug reports. I'm not even kidding.

Request a feature

If you want a new feature in OOPNET, please provide us with a use case and a description of what is missing. If possible, provide a code snippet that describes your use case. If you already have a solution for the implementation, please add this to the issue as well.

Security issues

If you find a security vulnerability, do NOT open an issue. Email oopnet.contact@gmail.com instead.

Style and standards

OOPNET uses a few standards to simplify its development. Please adhere to them.

Coding Style

Codes will be formatted by black as part of the CI pipeline and check by Pylint.

Github issue labels

StandardIssueLabels are used for labeling issues. Please read the description to choose the correct label(s) for your issue.

Git commit messages

Semantic Versioning is used to calculate the next version number based of the commits and their corresponding commit message. We use the Angular Commit message format for this purpose. Please, adhere to this standard, if you want your PR to be merged.

Versions look like this: MAJOR.MINOR.PATCH

For now, MAJOR is set to 0 until OOPNET is deemed stable.

Code of Conduct

Please read our Code of Conduct before interacting with the OOPNET team or the community. Violations will be met with clear consequences.

License

In short, when you submit code changes, your submissions are understood to be under the same MIT License that covers the project.

References

This document was adapted from the open-source contribution guidelines by Brian Danielak which is based on the guidelines for Meta's Draft-JS

2.5.4 MIT License

2.6 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

Symbols

A

B

C

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y