CriticalityMaps

Release 0.0.1

Patrick Hassett

Jun 11, 2020

CONTENTS

1 A word on units **3**

2 Overview **5**
- 2.1 Criticality Analysis . . . 5
- 2.2 Mapping . . . 7
- 2.3 API Documentation . . . 8

3 Indices and tables **17**

4 Funding Disclaimer **19**

Python Module Index **21**

Index **23**

CriticalityMaps is a WNTR-based utility for running large sets of fire and pipe criticality simulations and visualizing the results on interactive leaflet.js html maps.

Additionally, CriticalityMaps has mapping utilities that can be used to visualize any other attributes of the network on an interactive .html map.

Example Fire Criticality interactive map. Click on a highlited pipe to see the impact of its closure.

CHAPTER

ONE

A WORD ON UNITS

All values in CriticalityMaps are in SI for conformity with the WNTR units conventions. All unit conversions are left to the user.

CHAPTER

TWO

OVERVIEW

CriticalityMaps is composed of the following main components:

2.1 Criticality Analysis

The criticality analysis options currently supported in criticalityMaps are fire and pipe criticality analysis.

2.1.1 Fire Criticality

Fire criticality analysis is an assessment of the expected impact of firefighting demands at a given location. The analysis process consists of applying a fire fighting demand and measuring the impact on surrounding customers for all possible firefighting points in the system. Key parameters to customize this analysis are:

- firefighting demand (defaults to 1500 gpm)
- duration of the fire demand (defaults to 2 hr)
- diameter tresholds of pipes that will have fire demands applied (defaults to all pipes between 6 and 8in in diameter)

See *fire_criticality_analysis()* in the api documentation for more details on the customization options.

2.1.2 Pipe Criticality

Pipe criticality analysis provides insight on where the most critical pipes of the system are. To determine the criticality of a single pipe, the pipe is closed during a simulation and the impact on surrounding customers is measured. This process is then repeated for all pipes of interest in the system. Key parameters to customize this analysis are:

- duration of the pipe closure (defaults to 48 hr)
- diameter thresholds of pipes that will have pipe closures applied (defaults to all pipes greater than 12in in diameter)

See *pipe_criticality_analysis()* in the api documentation for more details on the customization options.

2.1.3 Segment Criticality

Segment criticality analysis provides insight on where the most critical segments of the system are. Segments are defined as groups of pipes that are connected within the same set of valves. In order to close one pipe in a segment, all the pipes in the segment must be closed. To determine the criticality of a single segment, all of the pipes in a segment are closed during a simulation and the impact on surrounding customers is measured. This process is then repeated for all segments of interest in the system. Key parameters to customize this analysis are:

- duration of the segment closure (defaults to 48 hr)
- diameter thresholds of pipes that will have pipe closures applied (defaults to all pipes greater than 12in in diameter)

See *segment_criticality_analysis()* in the api documentation for more details on the customization options.

2.1.4 Output and Post-processing

The core output of the criticality analyses is a [key:value] .yml file log where each key is the ID of a node/link tested and each value is the result of that test. Any nodes that fall below the minimum pressure threshold (a settable parameter `p_min`) of a pressure driven demand simulation recieved **none** of their requested demand during that period and are thus deemed impacted.

- If there were nodes impacted at a given test node/link, the value for that node/link will beanother set of [key:value] entries with the impacted node's ID as the key and its lowest observed pressure as the value.
- If there was no impact at a given test node/link, the value will be "NO AFFECTED NODES".
- Otherwise, if the simulation failed at a given test node/link, the value will be "failed:", followed by the exception message associated with the failure.

Below is an example of the .yml output demonstrating these three possible cases.

```
# a node/link with multiple impacted nodes
'123':
    '23': 11.33654
    '34': 5.345237
    '56': 10.21345
    '67': 9.234789
# a node/link with no impacted nodes
'35': NO AFFECTED NODES
# a node/link with failed simulation
'773': "failed: Simulation did not converge. Reached maximum number of iterations: 499
↪"
```

By default, the criticality analysis methods will additionally create the following outputs:

- a .csv file log of the population and nodes impacted at each node/link tested
- .pdf maps of the population and nodes impacted at each node/link tested

This behavior can be overridden by setting the post_process argument to False. The results summary .yml file will still be produced and can be then custom-processed with the *process_criticality()* function. See the api documentation on *process_criticality()* for more details.

The results of criticality analyses can also be displayed on an interactive map as demonstrated in the *Criticality Maps* section.

2.1.5 Multiprocessing

CriticalityMaps has the built-in ability to execute criticality analysis with mulitiprocessing, enabling multiple processors to work on a set of simulations at once. This offers a significant speedup in execution time, especially in cases with a large number of simulations and extra computing capacity available.

To enable multiprocessing on your criticlaity analysis, in addition to setting the multiprocess keyword argument to True, the code the criticality analysis must be wrapped in a `if __name__ == "__main__":` block as shown below.

```
if __name__ == "__main__":
    cm.fire_criticality_analysis(wn, multiprocess=True)
    cm.pipe_criticality_analysis(wn, multiprocess=True)
```

By default criticalityMaps will use about 66.7% of the machine's cpu. The numbers of cpu's used can be increased or decreased used by assigning a value for `num_processors`. See the api documentation on *`fire_criticality_analysis()`* and *`pipe_criticality_analysis()`* for more details on the multiprocessing options.

2.2 Mapping

2.2.1 Criticality Maps

There are built in methods to create interactive maps from the results of completed criticality analyses. To create a criticality map, enter the wn and the criticality results file into the *`make_criticality_map()`* function:

```
# Fire criticality example
cm.make_criticality_map(wn, fire_criticality_summary.yml)
# Pipe criticality example
cm.make_criticality_map(wn, pipe_criticality_summary.yml)
```

See the api documentation on *`make_criticality_map()`* for all available function options.

2.2.2 Dataframe-based Maps

CriticalityMaps also has the ability to create more general network maps based on the *`wn_dataframe`* class. A *`wn_dataframe`* object is composed of the a wn object, a node_data pandas DataFrame indexed by the water network's node ID's, and a link_data pandas DataFrame indexed by the water network's link ID's.

Initializing a wn_dataframe object

The initialization of a wn_dataframe object requires a WNTR WaterNetworkModel (wn) object as input. The node_data and link_data DataFrames are automatically created and populated with a 'coordinates' column, containing the coordinates of those network components.:

```
# The most basic initialization of a wn_dataframe object
my_wn_dataframe = cm.wn_dataframe(wn)
```

Adding data to the wn_dataframe

To add data to the wn_dataframe, standard methods for adding columns to an existing pandas DataFrame can be used:

```
# collect some data indexed by node ID
elevations = {}
for name, node in wn.nodes:
    elevations[name] = node.elevations
# add the node data as a new column labeled "elevation"
my_wn_dataframe.node_data["elevation"] = elevations

# wn querys are another great way to collect data on the water network model
base_demands = wn.query_node_attribute("base_demand")
# add the node data as a new column labeled "base demand"
my_wn_dataframe.node_data["base demand"] = base_demands

# collect some other data indexed by link ID
diameters = wn.query_link_attribute("diameter")
# add the data as a new column labeled "diameter"
my_wn_dataframe.link_data["diameter"] = diameters
```

Optionally, initial node and link data indexed by component ID can also be added to the object at initialization:

```
my_wn_dataframe = cm.wn_dataframe(wn,
                                  node_data={"elevation": elevations,"base demand":␣
↪base_demands},
                                  link_data={"diameter": diameters})
```

The data entered at initialization can be a DataFrame, a dict of dicts/Series, or any other object that can be converted to a dataframe by pandas.DataFrame(), so long as it is indexed by node/link ID.

Mapping the wn_dataframe

To map the data stored in the *wn_dataframe* on the water network, simply call the *make_map()* function of the wn_dataframe. Specify which fields will appear in tooltips and which fields are added as map overlays on the water network (Note: any fields added to map_columns will automatically be added to the tooltip when that layer is activated on the map).

```
my_wn_dataframe.make_map(map_columns=["base demand", "diameter"], tooltip_columns=[
↪"elevation"])
```

See the *wn_dataframe* class and its *make_map()* method in the api documentation for more details on implementation options.

2.3 API Documentation

2.3.1 criticalityMaps package

Subpackages

criticalityMaps.criticality package

Submodules

criticalityMaps.criticality.core module

Created on Tue Jun 4 11:09:02 2019

@author: PHassett

Modified: jhogge

criticalityMaps.criticality.core.**fire_criticality_analysis**(*wn*, *output_dir='./'*, *fire_demand=0.946*, *fire_start=86400*, *fire_duration=7200*, *min_pipe_diam=0.1524*, *max_pipe_diam=0.2032*, *p_nom=17.58*, *p_min=14.06*, *save_log=False*, *summary_file='fire_criticality_summary.yml'*, *post_process=True*, *pop=None*, *multiprocess=False*, *num_processors=None*)

A plug-and-play ready function for executing fire criticality analysis.

Parameters

- **wn** (*wntr WaterNetworkModel object*) – wntr wn for the water network of interest
- **output_dir** (*str/path-like object, optional*) – path to the directory to save the results of the analysis.

  Defaults to the working directory ("./").
- **fire_demand** (*float, optional*) – fire fighting demand(m^3/s).

  Defaults to 0.946 m^3/s (1500gpm).
- **fire_start** (*integer, optional*) – start time of the fire in seconds.

  Defaults to 86400 sec (24hr).
- **fire_duration** (*integer, optional*) – total duration of the fire demand in seconds.

  Defaults to 7200 sec (2hr).
- **min_pipe_diam** (*float, optional*) – minimum diameter pipe to perform fire criticality analysis on(meters).

  Defaults to 0.1524 m (6in).
- **max_pipe_diam** (*float, optional*) – maximum diameter pipe to perform fire criticality analysis on(meters).

  Defaults to 0.2032 m (8in).
- **p_nom** (*float, optional*) – nominal pressure for PDD (kPa). The minimun pressure to still recieve full expected demand.

  Defaults to 17.58 kPa (25psi).

- **p_min** (*float, optional*) – minimum pressure for PDD (kPa). The minimun pressure to still recieve any demand.

  Defaults to 14.06 kPa (20psi).

- **save_log** (*boolean, optional*) – option to save .json log files for each fire simulation. Otherwise, log files are still created but deleted after successful completion of all simulations. Serves as an effective back-up of the analysis results.

  Defaults to False.

- **summary_file** (*str, optional*) – file name for the yml summary file saved in output_dir

  Defaults to 'fire_criticality_summary.yml'.

- **post_process** (*boolean, optional*) – option to post process the analysis results with process_criticality. Saves pdf maps of the nodes and population impacted at each fire node, and corresponding csv files. To customize the post-processing output, set post_process to False and then run process_criticality() with the summary .yml file and any additional args as input.

  Defaults to True.

- **pop** (*dict or pandas DataFrame, optional*) – population estimate at each node. Used for post processing. If undefined, defaults to the result of wntr.metrics.population(wn).

  Defaults to None.

- **multiprocess** (*boolean, optional*) – option to run criticality across multiple processors.

  Defaults to False.

- **num_processors** (*int, optional*) – the number of processors to use if mp is True.

  Defaults to None if mp is False. Otherwise, defaults to int(mp.cpu_count() * 0.666), or 2/3 of the available processors.

criticalityMaps.criticality.core.**pipe_criticality_analysis**(*wn*, *output_dir='./'*, *break_start=86400*, *break_duration=172800*, *min_pipe_diam=0.3048*, *max_pipe_diam=None*, *p_nom=17.58*, *p_min=14.06*, *save_log=False*, *summary_file='pipe_criticality_summary.yml'*, *post_process=True*, *pop=None*, *multiprocess=False*, *num_processors=None*)

A plug-and-play ready function for executing fire criticality analysis.

Parameters

- **wn** (*wntr WaterNetworkModel object*) – wntr wn for the water network of interest
- **output_dir** (*str/path-like object, optional*) – path to the directory to save the results of the analysis.

  Defaults to the working directory ("./").

- **break_start** (*integer, optional*) – start time of the pipe break in seconds.

  Defaults to 86400 sec (24hr).

- **break_duration** (*integer, optional*) – total duration of the fire demand in seconds.

  Defaults to 172800 sec (48hr).

- **min_pipe_diam** (*float, optional*) – minimum diameter pipe to perform fire criticality analysis on(meters).

  Defaults to 0.3048 m (12in).

- **max_pipe_diam** (*float, optional*) – maximum diameter pipe to perform fire criticality analysis on(meters).

  Defaults to None.

- **p_nom** (*float, optional*) – nominal pressure for PDD (kPa). The minimun pressure to still recieve full expected demand.

  Defaults to 17.58 kPa (25psi).

- **p_min** (*float, optional*) – minimum pressure for PDD (kPa). The minimun pressure to still recieve any demand.

  Defaults to 14.06 kPa (20psi).

- **save_log** (*boolean, optional*) – option to save .json log files for each fire simulation. Otherwise, log files are still created but deleted after successful completion of all simulations. Serves as an effective back-up of the analysis results.

  Defaults to False.

- **summary_file** (*str, optional*) – file name for the yml summary file saved in output_dir.

  Defaults to 'pipe_criticality_summary.yml'.

- **post_process** (*boolean, optional*) – option to post process the analysis results with process_criticality. Saves pdf maps of the nodes and population impacted at each fire node, and corresponding csv files. To customize the post-processing output, set post_process to False and then run process_criticality() with the summary .yml file and any additional args as input.

  Defaults to True.

- **pop** (*dict or pandas DataFrame, optional*) – population estimate at each node. Used for post processing. If undefined, defaults to the result of wntr.metrics.population(_wn).

  Defaults to None.

- **multiprocess** (*boolean, optional*) – option to run criticality across multiple processors.

  Defaults to False.

- **num_processors** (*int, optional*) – the number of processors to use if mp is True.

  Defaults to None if mp is False. Otherwise, defaults to int(mp.cpu_count() * 0.666), or 2/3 of the available processors.

criticalityMaps.criticality.core.**segment_criticality_analysis**(*wn*, *link_segments*, *node_segments*, *valve_layer*, *output_dir='./'*, *break_start=86400*, *break_duration=172800*, *min_pipe_diam=0.3048*, *max_pipe_diam=None*, *p_nom=17.58*, *p_min=14.06*, *save_log=False*, *summary_file='segment_criticality_summary.y* *post_process=True*, *pop=None*, *multiprocess=False*, *num_processors=None*)

A plug-and-play ready function for executing segment criticality analysis.

Parameters

- **wn** (*wntr WaterNetworkModel object*) – wntr wn for the water network of interest
- **link_segments** (*Pandas series*) – results of valve_segments algorithm, listing links and their segment
- **node_segments** (*Pandas series*) – results of valve_segments algorithm, listing nodes and their segment
- **valve_layer** (*Pandas dataframe*) – list of node/segment combinations for the valves in the network
- **output_dir** (*str/path-like object, optional*) – path to the directory to save the results of the analysis.

  Defaults to the working directory ("./").
- **break_start** (*integer, optional*) – start time of the pipe break in seconds.

  Defaults to 86400 sec (24hr).
- **break_duration** (*integer, optional*) – total duration of the fire demand in seconds.

  Defaults to 172800 sec (48hr).
- **min_pipe_diam** (*float, optional*) – minimum diameter pipe to perform fire criticality analysis on(meters).

  Defaults to 0.3048 m (12in).
- **max_pipe_diam** (*float, optional*) – maximum diameter pipe to perform fire criticality analysis on(meters).

  Defaults to None.
- **p_nom** (*float, optional*) – nominal pressure for PDD (kPa). The minimun pressure to still recieve full expected demand.

  Defaults to 17.58 kPa (25psi).

- **p_min** (*float, optional*) – minimum pressure for PDD (kPa). The minimun pressure to still recieve any demand.

  Defaults to 14.06 kPa (20psi).

- **save_log** (*boolean, optional*) – option to save .json log files for each fire simulation. Otherwise, log files are still created but deleted after successful completion of all simulations. Serves as an effective back-up of the analysis results.

  Defaults to False.

- **summary_file** (*str, optional*) – file name for the yml summary file saved in output_dir.

  Defaults to 'segment_criticality_summary.txt'.

- **post_process** (*boolean, optional*) – option to post process the analysis results with process_criticality. Saves pdf maps of the nodes and population impacted at each fire node, and corresponding csv files. To customize the post-processing output, set post_process to False and then run process_criticality() with the summary .yml file and any additional args as input.

  Defaults to True.

- **pop** (*dict or pandas DataFrame, optional*) – population estimate at each node. Used for post processing. If undefined, defaults to the result of wntr.metrics.population(_wn).

  Defaults to None.

- **multiprocess** (*boolean, optional*) – option to run criticality across multiple processors.

  Defaults to False.

- **num_processors** (*int, optional*) – the number of processors to use if mp is True.

  Defaults to None if mp is False. Otherwise, defaults to int(mp.cpu_count() * 0.666), or 2/3 of the available processors.

criticalityMaps.criticality.core.**process_criticality**(*wn*, *summary_file*, *output_dir*, *pop=None*, *save_maps=True*, *save_csv=True*, *link_segments=None*, *node_segments=None*, *valve_layer=None*)

Process the results of a criticality analysis and produce some figures

Parameters

- **wn** (*wntr WaterNetworkModel object*) – the _wn that the analysis was performed on
- **summary_file** (*str/path-like object*) – path to the .yml summary file produced from a criticality analysis
- **pop** (*dict or pandas Series, optional*) – population estimate at each junction of the _wn. Output from *wntr.metrics.population* is suitable input format.
- **save_maps** (*bool, optional*) – option to save pdf maps of the population and nodes impacted at each node/link tested. Defaults to True.
- **save_csv** (*bool, optional*) – option to save a csv log of the population and nodes impacted at each node/link tested. Defaults to True.

criticalityMaps.criticality.criticality_functions module

Created on Tue Jun 4 11:33:38 2019

@author: PHassett

criticalityMaps.criticality.mp_queue_tools module

Created on Tue Jun 4 07:59:47 2019

@author: PHassett

criticalityMaps.criticality.mp_queue_tools.**runner**(*tasks*, *num_processors*)

Run the tasks specified across mutiple processors and return the results in a list.

Parameters

- **– list** (*tasks*) – task list of the form [(func,(arg1, arg2,. . . ,argN))]
- **– int** (*num_processors*) – the number of processors to use

Returns list of func return objects for each task

Return type results - list

Module contents

criticalityMaps.mapping package

Submodules

criticalityMaps.mapping.criticality_map module

Created on Wed Aug 7 18:41:45 2019

@author: PHassett

criticalityMaps.mapping.criticality_map.**make_criticality_map**(*wn*, *results_file*, *output_file=None*, *pop=None*)

Make a criticality map from a criticality results file.

Parameters

- **wn** (*wntr waternetwork model*) – the wntr waternetwork model of interest
- **results_file** (*str/path-like object*) – path to the .yml results file from a criticality analysis
- **output_file** (*str/path-like*) – path and .html file name for map output. Defaults to the path of the results file with '.yml' replaced with '_map.html'
- **pop** (*dict/Pandas Series, optional*) – population estimate at each node. If None, will use wntr.metrics.population(wn).

  Defaults to None

criticalityMaps.mapping.df_map module

Created on Mon Sep 9 13:12:07 2019

@author: PHassett

class criticalityMaps.mapping.df_map.**wn_dataframe**(*wn*, *node_data=None*, *link_data=None*)

Bases: object

A WaterNetwork Dataframe class specifically designed for mapping network components and their attributes

wn: wntr WaterNetworkModel

WaterNetworkModel of interest

node_data: pandas DataFrame or other object than can be converted to a DataFrame by pd.DataFrame(node_data)indexed by node id

link_data: pandas DataFrame or other object than can be converted to a DataFrame by pd.DataFrame(node_data)indexed by link id

make_map(*output_file=None*, *map_columns=[]*, *tooltip_columns=[]*, *geojson_layers={}*)

Make a .html web map of the wn and any data contained in the wn_dataframe

Parameters

- **output_file** (*str/path-like*) – path and .html file name for map output. Defaults to the name of the wn .inp file in the working directory.
- **map_columns** (*list, optional*) – list of column names in the wn_dataframe to be added as map layers

  Defaults to an empty list: [].
- **tooltip_columns** (*list, optional*) – list of column names in the wn_dataframe to be added to the informational tooltip that appears when hovering over network components with mouse.

  Defaults to an empty list: [].

criticalityMaps.mapping.geojson_handler module

Created on Wed Sep 4 08:42:03 2019

@author: PHassett

criticalityMaps.mapping.geojson_handler.**inp_to_geojson**(*wn*, *to_file=True*)

Write a minimal geojson representation of the Water Network.

Parameters

- **wn** (*wntr WaterNetworkModel object*) – The network to be make the geojson from
- **to_file** (*Boolean, default=False*) – To save the geojson representation as a file in the directory of the inp file

Returns **wn_geojson** – geojson spatial representation of the water network

Return type dict in geojson format

Module contents

Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FOUR

FUNDING DISCLAIMER

The U.S. Environmental Protection Agency (EPA) through its Office of Research and Development funded and collaborated in the research described herein under Interagency Agreement (IA #92432901) with the Department of Energy's Oak Ridge Associated Universities (ORAU).

PYTHON MODULE INDEX

C

criticalityMaps, 16
criticalityMaps.criticality, 14
criticalityMaps.criticality.core, 9
criticalityMaps.criticality.criticality_functions, 14
criticalityMaps.criticality.mp_queue_tools, 14
criticalityMaps.mapping, 16
criticalityMaps.mapping.criticality_map, 14
criticalityMaps.mapping.df_map, 15
criticalityMaps.mapping.geojson_handler, 15

INDEX

C

criticalityMaps
    module, 16
criticalityMaps.criticality
    module, 14
criticalityMaps.criticality.core
    module, 9
criticalityMaps.criticality.criticality_functions
    module, 14
criticalityMaps.criticality.mp_queue_tools
    module, 14
criticalityMaps.mapping
    module, 16
criticalityMaps.mapping.criticality_map
    module, 14
criticalityMaps.mapping.df_map
    module, 15
criticalityMaps.mapping.geojson_handler
    module, 15

F

fire_criticality_analysis() (*in module criticalityMaps.criticality.core*), 9

I

inp_to_geojson() (*in module criticalityMaps.mapping.geojson_handler*), 15

M

make_criticality_map() (*in module criticalityMaps.mapping.criticality_map*), 14
make_map() (*criticalityMaps.mapping.df_map.wn_dataframe method*), 15
module
    criticalityMaps, 16
    criticalityMaps.criticality, 14
    criticalityMaps.criticality.core, 9
    criticalityMaps.criticality.criticality_functions, 14
    criticalityMaps.criticality.mp_queue_tools, 14
    criticalityMaps.mapping, 16
    criticalityMaps.mapping.criticality_map, 14
    criticalityMaps.mapping.df_map, 15
    criticalityMaps.mapping.geojson_handler, 15

P

pipe_criticality_analysis() (*in module criticalityMaps.criticality.core*), 10
process_criticality() (*in module criticalityMaps.criticality.core*), 13

R

runner() (*in module criticalityMaps.criticality.mp_queue_tools*), 14

S

segment_criticality_analysis() (*in module criticalityMaps.criticality.core*), 11

W

wn_dataframe (*class in criticalityMaps.mapping.df_map*), 15