Getting started with pystorms

pystorms works on Windows, OSX, and Linux based operating systems. It requires Python 3.6+ and can be installed via pip

```
pip install pystorms
```

If you haven't installed Python packages before, please refer to https://packaging.python.org/en/latest/tutorials/installing-packages/

Simulating a stormwater control scenario

pystorms has seven scenarios that can used for evaluating and prototyping stormwater control algorithms. This example demonstrates how scenario theta for testing a rule-based control algorithm.

[1]:

```
import pystorms


def rule_based_controller(depths):
    """
    Determines control actions based on depths in the basin

    Parameters
    ----------
    depths : numpy.ndarray
        Depth in the basins of the stormwater network at current timestep.

    Returns
    -------
    actions : numpy.ndarray
        Gate positions to set at the outlets of basins in the stormwater␣
 ↪network at current timestep.


    Examples
    --------
    >>> depths = [1.5, 0.25]
    >>> rule_based_controller(depths)
    [0.5, 1.0]
    """
    actions = [1.0, 1.0]
    # gate positions in SWMM are between 0.0 to 1.0
    # 0.0 being completely closed and 1.0 is fully open
    for basin_index in range(0, len(depths)):
        if depths[basin_index] > 0.5:
            actions[basin_index] = 0.5
        else:
            actions[basin_index] = 0.0
```

```python
    return actions

scenario_theta_uncontrolled = pystorms.scenarios.theta()
done = False
while not done:
    # done gets set to True once the simulation ends else it is set to False
    # if no argument is passed to the step function, it sets the gate positions to completely open
    done = scenario_theta_uncontrolled.step()


scenario_theta_controlled = pystorms.scenarios.theta()
done = False
while not done:
    # get the current state in the stormwater network
    # in this scenario, state is the depth in the two controlled basins of the stormwater network
    state = scenario_theta_controlled.state()

    # determine the gate positions to set at the outlets of the two controlled basins
    actions = rule_based_controller(depths=state)

    # set the gate positions and progress the simulation
    # done gets set to True once the simulation ends else it is set to False
    done = scenario_theta_controlled.step(actions)

# performance of the control algorithm for scenario theta can be queried using the performance function call
print(f"\n\nPeformance of the uncontrolled scenario theta: {scenario_theta_uncontrolled.performance()}")
print(f"Peformance of the controller on scenario theta: {scenario_theta_controlled.performance()}")
```

```
 o  Retrieving project data
 o  Retrieving project data

Peformance of the uncontrolled scenario theta: 1630.3422288715237
Peformance of the controller on scenario theta: 1125.8162370076384
```

`pystorms` API explained

```
<scenario object> = pystorms.scenarios.<scenario name>()
```

`pystorms` treats each scenario as a class. The seven scenarios in pystorms can be invoked by replacing the scenario name by `theta`, `alpha`, `beta`, `gamma`, `delta`, and `epsilon`. Once the above statement is invoked, it will intialize the scenario, start the stormwater simulation, and hand the control over to the user. Users can then use this to class object to control the simulation.

```
state = <scenario object>.state()
```

`pystorms` scenario class object's `state` method can be used to query state of the stormwater network. This is a `numpy.ndarray`. The attributes in this array can be found in the scenario `.yaml` file. Scenario theta's configuration yaml is below.

```yaml
# Configuration file for scenario theta
# name of scearnio
name: theta
# state definitions
states:
        - !!python/tuple
          - P1
          - depthN
        - !!python/tuple
          - P2
          - depthN
# Action space
action_space:
        - "1"
        - "2"
# Performance Targets
performance_targets:
        - !!python/tuple
          - "8"
          - flow
        - !!python/tuple
          - P1
          - flooding
        - !!python/tuple
          - P2
          - flooding
```

`state` contains node depth in basin `P1` and node depth in basin `P2` as the first and second elements in the state array.

```
done = <scenario object>.step(actions)
```

`step(actions)` function implements the control actions, if actions are passed as an argument or else sets the controlled gates to completely open, and progresses the simulation one timestep. If the simulation ends it returns True or else it returns False.