2-Pynsim Trivial Example

Notebook:	Pynsim Examples		
Created:	8/13/2019 3:22 PM	**Updated:**	9/26/2019 10:53 AM
Author:	ajainf1@stanford.edu		
URL:	about:blank		

Pynsim Trivial Example Read-Me

The Pynsim trivial example is a simple example to introduce pynsim functionality. Pynsim is well-suited for node-link networks where decisions about the inflows/outflows to/from nodes and through links are complex and operated by different institutions/rules.

The pynsim trivial demo network is set up as shown in Figure 1 and is intended to calculate the irrigation deficit in the system during the 5 time step period. It could be considered a representation of a canal system in which decisions are simplified but typify how pynsim could be used with two decision makers.

The main objectives in this trivial example are:

- use the example network to introduce decision-makers, called institutions and introduce nodes and sub classes of nodes (irrigation, reservoir) and sub-subclasses (citrus, vegetable irrigatoin nodes)
- demonstrate the classes in pysnim (nodes, links, institutions, engine) and how the network maps to these
- walk-through the code to show how the components interact and calculate
- propose sensitivity tests to develop familiarity with pynsim

Defining the Network

The network has four **nodes:** One surface reservoir node (R1) and three irrigation nodes (I1, I2, I3). The irrigation nodes can be further classified as **Citrus** or **Vegetable** farms. The nodes are connected as shown by river section **links.**

The network has two **institutions** which represent the two decision makers: The **Water Department** decides the total water released from node R1 at each of the 5 time steps (i.e. the release). The **Irrigation Ministry** decides the water distribution to the irrigation nodes (i.e. the allocation) at each of the 5 time steps.

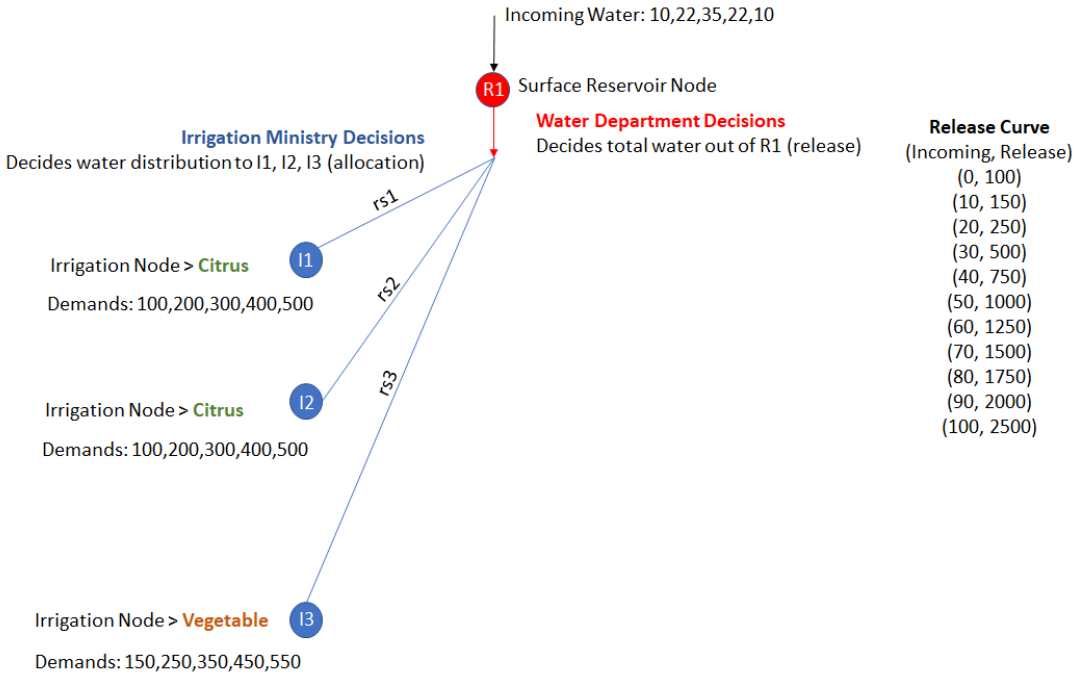

Figure 1: Trivial Example Setup. Note

The pynsim package makes use of object oriented programming, particularly using **classes** to store information (inputs) about the different components and to code the functions (called **method classes**) that are used to represent the processes or decision-making.

Components

- link.py defines the River Section links. They have properties of *min and max flow*. Currently these are set to none.
- node.py defines the Surface Reservoir node with properties of *release, capacity and max_release*. The property of interest is *release* (the other two do not play a role in the trivial example). This script also defines the Irrigation Nodes with properties of *demand, deficit and max_allowed*.
    - The child classes Citrus Farm and Vegetable Farm store the *seasonal_water_requirements* and the method class "setup" makes the irrigation demand at the node equal to the seasonal_water_requirement of the farm.
- institutions.py defines the 2 decision makers:
    - The Water Department has a hard-coded release curve and a method class "setup" that chooses the first point on the curve that is greater than the incoming water and sets the release to that. (i.e. If the incoming water is 10, the first value greater than 10 on the release curve - shown above - is 20, the release then is 250).
    - The Irrigation Decision Maker has a weighted allocation. The weights for the farms are hard-coded here (citrus = 0.8 and veg = 0.7). The maximum allowed value is set to the release from R1 times the weight for that farm.

Engine

- allocation.py has a method class "run" that calculates the deficit at a node as the difference between the demand and the max_allowed. It then takes the cumulative sum of the deficit to calculate the irrigation deficit at the farm for the full time-period.

Simulation

- dry_year_simulation.py puts the parts together to create Figure 1 (shown above). It creates an instance of the simulator, defines the timesteps, creates an instance of the network. It includes the Incoming Water as an input. The instances of the desired nodes are created to have 1 surface reservoir (sr1), one citrus farm (irr1), another citrus farm(irr2) and a vegetable farm (irr3). Notice that the variable name is different from the name property (for example variable name sr1, property name "R1"). The links, nodes and institutions are added to the network. Note the order of addition as the nodes are added to the institutions and to the network separately. The simulation is started here.

The network itself is created i in the dry_year_simulation. The inputs are hard-coded and then network pieces are added. The relevant code for creating the network is shown below:

```
# Create instances of the different pynsim classes with relevant attributes
s = Simulator()
n = Network(name="example network")

# Nodes
sr1 = SurfaceReservoir(x=1,  y=2,   name="R1", release=100.0)
irr1 = CitrusFarm(x=1,   y=2,   name="I1")
irr2 = CitrusFarm(x=10,  y=20,  name="I2")
irr3 = VegetableFarm(x=100, y=200, name="I3")
n.add_nodes(sr1, irr1, irr2, irr3)

# Links
n.add_link(RiverSection(name="rs1", start_node=sr1, end_node=irr1))
n.add_link(RiverSection(name="rs2", start_node=sr1, end_node=irr2))
n.add_link(RiverSection(name="rs3", start_node=sr1, end_node=irr3))

#Set the simulator's network to this network.
s.network = n

#Institutions
mow = WaterDepartment("Jordan Ministry of Water")
mow.add_nodes(sr1)
jva = IrrigationDecisionMaker("Jordan Valley Authority")
jva.add_nodes(irr1, irr2, irr3)
n.add_institutions(mow, jva)

#Engine
#Create an instance of the deficit allocation engine.
allocator = DeficitAllocation(n)
```

Walking through the trivial example code

The incoming water is hardcoded

incoming water = [10, 22, 35, 22, 10]

- **Water Department Decision**

The first decision occurs at the Surface Reservoir node **R1** by the Water Department. The release variable is initialized but this is overwritten. The hard coded _release_curve defines points based on incoming water.

Release Curve
(Incoming, Release)
(0, 100)
(10, 150)
(20, 250)
(30, 500)
(40, 750)
(50, 1000)
(60, 1250)

(70, 1500)
(80, 1750)
(90, 2000)
(100, 2500)

The decision does not interpolate. It simply chooses the first point on the curve that is greater than the incoming water. The section of the institutions code shown below represents the decision.

```
#find the amount to give to the reservoir and then give it (set the release
attribute)
for alloc in self._release_curve:
    if alloc[0] <= incoming_water:
        continue
    else:
        reservoir.release = alloc[1]
        break
```

With the incoming water and the release curve shown in Figure 1, the expected releases would be:

release = [250, 500, 750, 500, 250]

Run dry_year_simulation.py and check that the release matches the values above.

```
# To check after running, type the following in a console:
sr1._history['release']  #sr1 = surface reservoir R1

# or if you saved the network into a variable nwk
nwk._node_map['R1']._history['release']
nwk._get_node('R1')._history['release']
```

- **Irrigation Ministry Decision**

The next decision is the allocation to **I1, I2, I3**, the different irrigation nodes managed by the Irrigation Ministry. This decision depends on the weight given to the citrus vs. vegetable farms as follows:
The weighted allocation is for 2 citrus farms (w_c=0.8) and 1 vegetable farm (w_v=0.7) so:
weighted sum, ws= 2 (0.8) + 1(0.7) = 2.3;
The citrus release decision = 0.7/2.3 * release = max_allowed
The veg release decision = 0.8/2.3 * release = max_allowed

Citrus Max allowed, A = (0.8/2.3)*[250, 500, 750, 500, 250]= [86.96, 173.91, 260.87, 173.91, 86.96]
Veg Max allowed, A = (0.7/2.3)*[250, 500, 750, 500, 250]= [76.1, 152.2, 228.3, 152.2, 76.1]

The relevant part of the institutions.py code that represents this is as follows:

```
#Add up the weights.
for n in self.nodes:
    if n.component_type == "CitrusFarm":
        weight_sum = weight_sum + citrus_weight
    elif n.component_type == "VegetableFarm":
        weight_sum = weight_sum + veg_weight


#Get the proportions of the release per farm based on a simple weighting
#calculation.
citrus_release = citrus_weight/weight_sum * release
veg_release    = veg_weight/weight_sum    * release


#The proportion of release per farm is the maximum each farm can have
```

```
#on this time step. The difference between this value and the farmer's
#demand is the deficit
for n in self.nodes:
    if n.component_type == "CitrusFarm":
        n.max_allowed = citrus_release
    elif n.component_type == "VegetableFarm":
        n.max_allowed = veg_release
```

Run dry_year_simulation.py and check that the irrigation max_allowed matches the values above.

```
# To check after running, type the following in a console:
irr1._history['max_allowed’] # for citrus
irr3._history['max_allowed’] # for veg

#or if you saved the network in variable nwk
nwk._node_map['I1']._history['max_allowed’]

# note in one we make use of the variables defined in the simulation while the
other uses the name property from the class definition
```

- **Node Deficit** calculated by the allocation.py (This is the engine)

I1, I2: Citrus Farm Irrigation Nodes
The calculation for the two citrus farms is identical because they have the same demands. These could have different demands and we could repeat calculations independently for each one. This is a "worked out" example for one citrus farm. For completion the result for veg farm is also included but the calculation is not shown.

Citrus Demands, D = [100,200,300,400,500] # Hardcoded/input

Max Allowed from above:
Citrus Max allowed, A = (0.8/2.3)*[250, 500, 750, 500, 250]
= [86.96, 173.91, 260.87, 173.91, 86.96]
Allocation Calculation:
Citrus Node Deficit, ND= D-A = [13.04, 26.09, 39.13, 226.09, 413.04]
Citrus Irrigation Deficit = cumulative sum of Node Deficit
= [13.04, 39.13, 78.26, 304.35, 717.39]
For completion:
Veg Irrigation Deficit = ([73.91, 171.74, 293.48, 591.3 , 1065.22])
Total annual deficit = 2*(717.39)+1065.22 = 2500
The relevant part of the allocation.py code that represents this is as follows:

```
#find all the irrigation nodes and calculate their deficits.
irrigation_nodes = []
deficits = {}
for n in self.target.nodes:
    if n.type == 'irrigation':
        irrigation_nodes.append(n)
        deficits[n.name] = n.deficit


for irr in irrigation_nodes:
    if irr.demand > irr.max_allowed:
        node_deficit = irr.demand - irr.max_allowed
        irr.deficit = irr.deficit + node_deficit
```

Run dry_year_simulation.py and check that the node deficit matches the calculated values above.

```
# To check
import numpy as np
```

```
node_def= np.array(irr1._history['demand']) -
np.array(irr1._history['max_allowed'])

irr_def = np.cumsum(node_def)
```

Sensitivity

With this general understanding you can do sensitivity analysis on the trivial example.

- What happens if you change the incoming water? >> run wet year simulation
- What happens if the relative weights of citrus and vegetables are changed?
- What happens if the farm seasonal water requirements (demands) change?
- Can you change the release decision rule to interpolate between values?
- Can you add another farm to the network?
- Can you change the irrigation decision rule to using an optimization based on the value of the water to each farm? **

Playing enough with this example should eventually make you comfortable writing your own trivial canal-like example with Pynsim.

Extra

Some other commands to try:
network.draw()