Documentation for Tree.py

1 Introduction

The tree.py file builds the structure (TreeModel) of a non-recombining binomial tree for the carbon pricing model. It is used in other files: analysis.py, bau.py, cost.py, damage_simulation.py, damage.py, forcing.py, optimization.py, storage_tree.py, utility.py. There are two different concepts: **node** and **state**.

- **Nodes** are path-dependent integers. They represent decision points throughout the whole time span in a given order. In Litterman's paper, the world can enter either a good state or a bad state, denoted by 'u' and 'd' respectively, at all but the last two decision times. Under this framework, the original single decision point is 0. For period 1, the 'u' state and the 'd' state are represented by node 1 and 2, respectively. For period 2, both node 1 and 2 give rise to 'u' and 'd' states, here we call them their child nodes. The 'uu', 'ud', 'du', 'dd' states are represented by node 3, 4, 5, 6 respectively (see Figure 1). The same logic applies for the entire model.

  The last period have no branching, and hence it has the same number of nodes as the previous period.

  Note that for an arbitrary node, we can treat it as a child node and find its unique parent node. For example, the parent node for 3 is 1, and that for 14 is 6. We set node 0 as its own parent node.

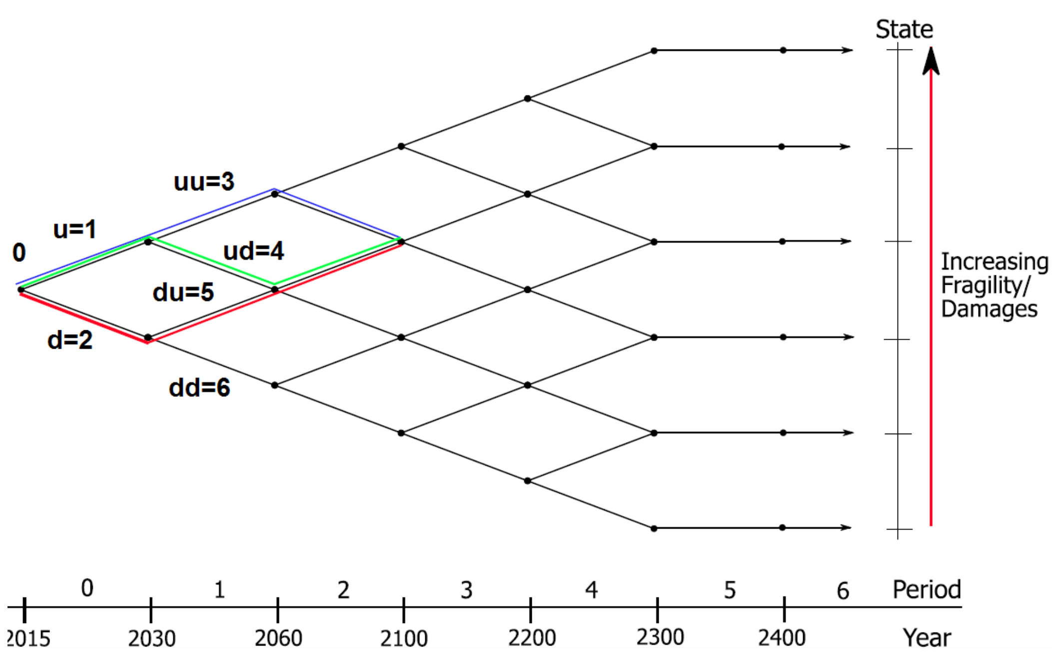

Figure 1: **Tree in carbon pricing model**

- **States** are path-independent integers. They represent ordered decision points within each period, so for an arbitrary period, we have states starting at 0. For example, in the second period, nodes 3, 4, 5, 6 correspond to states 0, 1, 2, 3. This is to say, the number of states in each period is exactly the number of nodes that represent decision points in this period.

The tree is stored in a 1-D array. What are the entries of this array? Particular nodes and states can be used via the methods defined in this file.

2 Inputs

The TreeModel requires the following input information:

- *Decision time.*

  decision_times: years in the future when decisions are to be made. For example, [2015,2030,2060,2100,2200,2300,2400]. Each of them represents the beginning of a period.

- *Probability that the world comes to a good state at a decision point.*

  prob_scale: scaling constant probability that the world is in a good state. What do you mean here? For example, at node 1 in Fig.1 there is 1/2 probability of going up (good state). But what are those scaling probabilities that you mention? It is assumed to be time-invariant. The default value is 1.0.

3 Python: TreeModel

In this section we explain different blocks of the python code in tree.py.

Import the packages.

```
from __future__ import division
import numpy as np
```

3.1 Attributes

The model has the following attributes which provide detailed information of the binomial tree structure that we can refer to.

- **decision_times**: years in the future when decisions will be made.
- **prob_scale**: scaling constant probability that the world is in a good state. It is assumed to be time-invariant. The default value is 1.0.
- **node_prob**: probability of reaching each node from period 0. For example,...

- **final_states_prob**: probability of reaching each node in the last period from period 0. For example,...

Define the tree model with decision times and an optional probability scale, modify the decision time input into array if it comes as a list.

```
class TreeModel(object):
    def __init__(self, decision_times, prob_scale=1.0):
        self.decision_times = decision_times
        if isinstance(self.decision_times, list):
            self.decision_times = np.array(self.decision_times)
        self.prob_scale = prob_scale
        self.node_prob = None
        self.final_states_prob = None
        self._create_probs()
```

From **decision_times**, we can calculate the number of periods we need to consider. Here the number of periods means the number of periods between two decision points. That is, we do not include the last period that extends to infinite future. We say these periods are within the decision time span. This is because the last decision point does not really influence the state of the world and no more branching needs to be considered.

number of periods $=$ number of decision times $-\ 1$.

For example, if we refer to Figure 1, given decision times [2015,2030,2060,2100,2200,2300,2400], we have 7 decision times and 7 periods in total. The number of periods is 6, however, takes only period 0 to 5 into consideration.

```
    @property
    def num_periods(self):
        """int: the number of periods in the tree"""
        return len(self.decision_times)-1
```

number of decision nodes $= 2^{\text{number of periods}} - 1$.

For example, for node 0 we have $63 = 2^6 - 1$ nodes where we have to make decisions. Hence, 63 variables for the optimization.

```
    @property
    def num_decision_nodes(self):
        """int: the number of nodes in tree"""
        return (2**self.num_periods) - 1
```

number of states in the final period $= 2^{\text{number of periods}-1}$.

For example, we have $2^{6-1} = 32$ states in period 5. This number is also true for any future period.

```
    @property
    def num_final_states(self):
```

```
        """int: the number of nodes in the last period"""
        return 2**(self.num_periods-1)
```

3.2 Methods

The TreeModel has many methods that can be helpful when we need to look into a specific period or decision point. For instance, each decision point has parameters including node, period, and state. Given any of them, we can determine the other two through the methods defined below.

Anything to say about this piece of code?

```
    def _create_probs(self):
        """Creates the probabilities of every nodes in the tree structure."""
        self.final_states_prob = np.zeros(self.num_final_states)
        self.node_prob = np.zeros(self.num_decision_nodes)
        self.final_states_prob[0] = 1.0 #init the prob of the first final state as 1 (a
        sum_probs = 1.0
        next_prob = 1.0

        for n in range(1, self.num_final_states):
            next_prob = next_prob * self.prob_scale**(1.0 / n)
            self.final_states_prob[n] = next_prob
        self.final_states_prob /= np.sum(self.final_states_prob) # normalize the prob a

        self.node_prob[self.num_final_states-1:] = self.final_states_prob
        for period in range(self.num_periods-2, -1, -1): #add up the prob of the child
            for state in range(0, 2**period):
                pos = self.get_node(period, state) #find the node from state
                self.node_prob[pos] = self.node_prob[2*pos + 1] + self.node_prob[2*pos +
```

get_num_nodes_period: get the number of nodes for a given period.

- For a period beyond the decision time span, its number of nodes $= 2^{\text{number of periods}-1}$. For example, we constantly have 32 nodes since period 5.
- For a period within the decision time span, its number of nodes $= 2^{\text{period}}$. For example, for period 2 we have 4 nodes (see Figure 1).

```
    def get_num_nodes_period(self, period):
        """Returns the number of nodes in the period.
        Parameters
        ----------
        period : int
            period
        Returns
        -------
        int
```

```
            number of nodes in period
        Examples
        --------
        >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
        >>> t.get_num_nodes_period(2)
        4
        >>> t.get_num_nodes_period(5)
        32
        """
        if period >= self.num_periods:
            return 2**(self.num_periods-1)
        return 2**period
```

get_node: get a node from given period and state.

- node $= 2^{\text{period}} + \text{state} - 1$.
- For example, the 11th node in the 5th period is 25, with period and state being 10 and 4 respectively.
- Value Error for period and state inputs beyond valid range.

```
    def get_node(self, period, state):
        """Returns the node in period and state provided.
        Parameters
        ----------
        period : int
            period
        state : int
            state of the node
        Returns
        -------
        int
            node number
        Examples
        --------
        >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
        >>> t.get_node(1, 1)
        2
        >>> t.get_node(4, 10)
        25
        >>> t.get_node(4, 20)
        ValueError: No such state in period 4
        Raises
        ------
        ValueError
            If period is too large or if the state is too large
```

```
            for the period.
        """
        if period > self.num_periods:
            raise ValueError("Given period is larger than number of periods")
        if state >= 2**period:
            raise ValueError("No such state in period {}".format(period))
        return 2**period + state - 1
```

get_nodes_in_period: get the range of nodes for a given period within the decision time span.

- For a period beyond the decision time span, modify it to the last period in the decision time span, period 5 in our case.
- Get the first node with **get_node** and the state being 0.
- Get the number of nodes in this period with **get_num_nodes_period**.
- Get the last node = the first node + number of nodes in this periods − 1.
- For example, we get the node range (0, 0) for period 0 and (15, 30) for period 4.

```
    def get_nodes_in_period(self, period):
        """Returns the first and last nodes in the period.
        Parameters
        ----------
        period : int
            period
        Returns
        -------
        int
            number of nodes in period
        Examples
        --------
        >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
        >>> t.get_nodes_in_period(0)
        (0, 0)
        >>> t.get_nodes_in_period(1)
        (1, 2)
        >>> t.get_nodes_in_period(4)
        (15, 30)
        """
        if period >= self.num_periods:
            period = self.num_periods-1
        nodes = self.get_num_nodes_period(period)
        first_node = self.get_node(period, 0)
        return (first_node, first_node+nodes-1)
```

get_period: get the period that a given node falls in.

- For a node beyond the decision nodes, let it be the last period that extends to infinite future, period 6 in our case,.
- For a node within our decision time span, we use a for loop to find the period it falls in. The target period i should satisfy $\text{int}\left(\frac{\text{node}+1}{2^i}\right) = 1$.
- For example, node 4 falls in period 2.

```
def get_period(self, node):
    """Returns what period the node is in.

    Parameters
    ----------
    node : int
        the node

    Returns
    -------
    int
        period

    Examples
    --------
    >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
    >>> t.get_period(0)
    0
    >>> t.get_period(4)
    2

    """
    if node >= self.num_decision_nodes:
        return self.num_periods

    for i in range(0, self.num_periods):
        if int((node+1) / 2**i ) == 1:
            return i
```

get_state: get the state of a given node.

- For a node beyond the decision time span, its state is node − number of decision nodes
- For a node within the decision time span, get the node's period, if not available, with **get_period**, then compute state = node $-\left(2^{\text{period}-1}\right)$
- For example, the state of node 4 (in period 2) is 1.

```
def get_state(self, node, period=None):
    """Returns the state the node represents.
```

```
        Parameters
        ----------
        node : int
            the node
        period : int, optional
            the period

        Returns
        -------
        int
            state

        Examples
        --------
        >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
        >>> t.get_state(0)
        0
        >>> t.get_state(4, 2)
        1

        """
        if node >= self.num_decision_nodes:
            return node - self.num_decision_nodes
        if not period:
            period = self.get_period(node)
        return node - (2**period - 1)
```

get_parent_node: get the parent node for a given node.

- For node 0, its parent node is 0.
- For node beyond the decision time span, we find its parent node by subtracting the number of nodes in the last period.
- For an even node within the decision time span, we get the parent node int $\left(\frac{\text{child}-2}{2}\right)$. For example, the parent node for node 4 is 1.
- For a odd node within the decision time span, we get the parent node int $\left(\frac{\text{child}-1}{2}\right)$. For example, the parent node for node 11 is 5.

```
    def get_parent_node(self, child):
        """Returns the previous or parent node of the given child node.

        Parameters
        ----------
        child : int
```

```python
            the child node

        Returns
        -------
        int
            partent node

        Examples
        --------
        >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
        >>> t.get_parent_node(2)
        0
        >>> t.get_parent_node(4)
        1
        >>> t.get_parent_node(10)
        4

        """
        if child == 0:
            return 0
        if child > self.num_decision_nodes:
            return child - self.num_final_states
        if child % 2 == 0:
            return int((child - 2) / 2)
        else:
            return int((child - 1 ) / 2)
```

get_path: get the unique path through which we can arrive at a given node.

- Get the period of the node with **get_node** if not available.
- Start with the given node, determine the nodes of the path backwards by finding the parent node **get_parent_node** at each decision point.
- Reverse the order of the nodes to get the path in need.
- For example, the path to node 62 is [0, 2, 6, 14, 30, 62].

```python
    def get_path(self, node, period=None):
        """Returns the unique path taken to come to given node.

        Parameters
        ----------
        node : int
            the node

        Returns
        -------
```

```python
        ndarray
            path to get to `node`

        Examples
        --------
        >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
        >>> t.get_path(2)
        array([0, 2])
        >>> t.get_parent_node(4)
        array([0, 1, 4])
        >>> t.get_parent_node(62)
        array([ 0,  2,  6, 14, 30, 62])

        """
        if period is None:
            period = self.get_period(node)
        path = [node]
        for i in range(0, period):
            parent = self.get_parent_node(path[i])
            path.append(parent)
        path.reverse()
        return np.array(path)
```

get_probs_in_period: get the probabilities of each states in a given period.

- Get the nodes of the given period with **get_period**.
- Get the probabilities of the nodes in concern with **node_prob**.
- For example, assume that the probability of 'u' state is 0.5 at each decision point. Then the probabilities for nodes in period 2 should be [0.25, 0.25, 0.25, 0.25].

```python
    def get_probs_in_period(self, period):
        """Returns the probabilities to get from period 0 to nodes in period.

        Parameters
        ----------
        period : int
            the period

        Returns
        -------
        ndarray
            probabilities

        Examples
        --------
```

```
    >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
    >>> t.get_probs_in_period(2)
    array([ 0.25,  0.25,  0.25,  0.25])
    >>> t.get_probs_in_period(4)
    array([ 0.0625,  0.0625,  0.0625,  0.0625,  0.0625,  0.0625,  0.0625,
            0.0625,  0.0625,  0.0625,  0.0625,  0.0625,  0.0625,  0.0625,
            0.0625,  0.0625])

    """
    first, last = self.get_nodes_in_period(period)
    return self.node_prob[range(first, last+1)]
```

reachable_end_states: determine the range of states we have a possibility to reach in the last period, given a starting node.

- Get the state and period of the node, if not available, with **get_period** and **get_state**
- For a period that is beyond the decision time span, we already know the state we end up with, which is node − number of decision nodes. For example, the range of reachable states from node 32 is (1, 1).
- For a period within the decision time span, the range of possible final states should be $\text{int}\left(\frac{\text{number of final states}}{2^{\text{period}}}\right) \times (\text{state}, \text{state} + 1) + (0, -1)$. For example, the range of reachable states from node 10 is (12, 15).

```
def reachable_end_states(self, node, period=None, state=None):
    """Returns what future end states can be reached from given node.

    Parameters
    ----------
    node : int
        the node
    period : int, optional
        the period
    state : int, optional
        the state the node is in

    Returns
    -------
    tuple
        (worst end state, best end state)

    Examples
    --------
    >>> t = TreeModel([0, 15, 45, 85, 185, 285, 385])
    >>> t.reachable_end_states(0)
    (0, 31)
```

```
    >>> t.reachable_end_states(10)
    (12, 15)
    >>> t.reachable_end_states(32)
    (1, 1)

    """
    if period is None:
        period = self.get_period(node)
    if period >= self.num_periods:
        return (node - self.num_decision_nodes, node - self.num_decision_nodes)
    if state is None:
        state = self.get_state(node, period)

    k = int(self.num_final_states / 2**period)
    return (k*state, k*(state+1)-1)
```