Documentation for bau.py

1 Introduction

The file bau.py has two classes: **BusinessAsUsual** and **DLWBusinessAsUsual**. They stand for the business-as-usual scenario of emissions. Emission growth is assumed to slow down exogenously, so we here attempt to model emission growth in business-as-usual scenario that is absent of incentives. Here **BusinessAsUsual** is the super class of **DLWBusinessAsUsual**.

- **BusinessAsUsual**: an abstract business-as-usual class for the the emission growth in EZ-Climate model.
- **DLWBusinessAsUsual**: a class that model the business-as-usual scenario for the emission growth in DLW paper.

Since this file contains 2 classes, we discuss them separately.

2 BusinessAsUsual

2.1 Inputs

- **ghg_start**: a float that represents current GHGs level. In our example, this value is 400.0.
- **ghg_end**: a float that represents GHGs level in the last period. In our example, this value is 1000.0.

2.2 Python

Import necessary packages.

```
import numpy as np
from abc import ABCMeta, abstractmethod
```

```
class BusinessAsUsual(object):
    """Abstract BAU class for the EZ-Climate model.

    Parameters
    ----------
    ghg_start : float
        today's GHG-level
    ghg_end : float
```

```
        GHG-level in the last period

    Attributes
    ----------
    ghg_start : float
        today's GHG-level
    ghg_end : float
        GHG-level in the last period
    emission_by_decisions : ndarray
        emission levels at decision points
    emission_per_period : ndarray
        amounts of emission for each single period.
    emission_to_ghg : ndarray
        change in GHGs level attributed to each period
    emission_to_bau : float
        constant: the last period increase in GHGs level divided by its emission

    """
__metaclass__ = ABCMeta
    def __init__(self, ghg_start, ghg_end):
        self.ghg_start = ghg_start
        self.ghg_end = ghg_end
        self.emission_by_decisions = None
        self.emission_per_period = None
        self.emission_to_ghg = None
        self.emission_to_bau = None
        self.bau_path = None

   @abstractmethod
    def emission_by_time(self):
        pass
```

3 DLWBusinessAsUsual

3.1 Inputs

DLWBusinessAsUsual requires inputs about the green house gases (GHGs) and emissions.

- **ghg_start**: a float that represents current GHGs level. In our example, this value is 400.0.
- **ghg_end**: a float that represents GHGs level in the last period. In our example, this value is 1000.0.

- **emit_time**: a list or multi-dimensional, homogeneous array that represents time points, in years, from now when emissions occur. In our example, emit_time=[0, 30, 60], so emissions are assumed to happen today, 30 years from today, 60 years from today, respectively.
- **emit_level**: a list or multi-dimensional. homogeneous array that represents emission levels at **emit_time**. In our model, emit_level = [52, 70, 81.4]

3.2 Attributes

DLWBusinessAsUsual has some attributes that reveal the relationship between GHGs levels and emission levels, including some that we've seen in *Inputs*.

- **ghg_start**: a float that represents current GHGs level.
- **ghg_end**: a float that represents GHGs level in the last period.
- **emit_time**: a list or multi-dimensional, homogeneous array that represents time points, in years, from now when emissions occur.
- **emit_level**: a list or multi-dimensional. homogeneous array that represents emission levels at **emit_time**.
- **emission_by_decision**: a multi-dimensional, homogeneous array that represents the emission levels at different decision times.
- **emission_per_period**: a multi-dimensional, homogeneous array that represents the amount of emission during an arbitrary period/ between two decision points.
- **emission_to_ghg**: a multi-dimensional, homogeneous array that represents the amounts of change in GHGs level attributed to each period according to the weights of their periodic emission **emission_per_period**.
- **emission_to_bau**: a float that represents the change of GHGs level in the last period divided by its periodic emission.

3.3 Python

```
class DLWBusinessAsUsual(BusinessAsUsual):
    """Business-as-usual scenario of emissions. Emissions growth is assumed to slow do
    exogenously - these assumptions represent an attempt to model emissions growth in
    business-as-usual scenario that is in the absence of incentives.

    Parameters
    ----------
    ghg_start : float
        today's GHG-level
```

```
    ghg_end : float
        GHG-level in the last period
    emit_time : ndarray or list
        time, in years, from now when emissions occur
    emit_level : ndarray or list
        emission levels in future times `emit_time`

    Attributes
    ----------
    ghg_start : float
        today's GHG-level
    ghg_end : float
        GHG-level in the last period
    emission_by_decisions : ndarray
        emission level a  specific decision time
    emission_per_period : ndarray
        the amount of emission for each period/between two decision times
    emission_to_ghg : ndarray
        change in GHGs level attributed to each period
    emission_to_bau : float
        constant: the last period increase in GHGs level divided by its emission
    emit_time : ndarray or list
        time, in years, from now when emissions occurs
    emit_level : ndarray or list
        emission levels in future times `emit_time`

    """

    #the default emit_time [0, 30, 60] should  work here, but with a tree model with d
    #[2015, 2030, ...]
    def __init__(self, ghg_start=400.0, ghg_end=1000.0, emit_time=[0, 30, 60], emit_leve
        super(DLWBusinessAsUsual, self).__init__(ghg_start, ghg_end)
        self.emit_time = emit_time
        self.emit_level = emit_level
```

3.3.1 Methods

emission_by_time:

- Take an integer input that represents a future time period in years. For example, an input 30 means 30 years from the starting point.
- Assume a linear relationship between time and emission. Then infer the emission level based on the time.

$$\text{In general, emission by time} = \begin{cases} \text{emit}[0] + texttime \cdot \frac{\text{emit}[1]-\text{emit}[0]}{\text{time}[1]-\text{time}[0]}, & \text{time} < \text{time}[1]. \\ \text{emit}[1] + (\text{time} - \text{time}[1]) \cdot \frac{\text{emit}[2]-\text{emit}[1]}{\text{time}[2]-\text{time}[1]}, & \text{time}[1] < \text{time} < \text{time}[2] \\ \text{emit}[2], & \text{time} \geq \text{time}[2] \end{cases}$$

- Return the emission.

```
def emission_by_time(self, time):
    """Returns the BAU emissions at any time

    Parameters
    ----------
    time : int
        future time period in years

    Returns
    -------
    float
        emission

    """
    if time < self.emit_time[1]:
        emissions = self.emit_level[0] + float(time) / (self.emit_time[1] - self.emi
                    * (self.emit_level[1] - self.emit_level[0])
    elif time < self.emit_time[2]:
        emissions = self.emit_level[1] + float(time - self.emit_time[1]) / (self.emi
                    - self.emit_time[1]) * (self.emit_level[2] - self.emit_level[1])
    else:
        emissions = self.emit_level[2]
    return emissions
```

bau_emission_setup:

- TreeModel in tree.py provides the tree structure in need.

```
def bau_emissions_setup(self, tree):
    """Create default business as usual emissions path. The emission rate in e
    assumed to be the average of the emissions at the beginning and at the end

    Parameters
    ----------
    tree : `TreeModel` object
        provides the tree structure used

    """
    num_periods = tree.num_periods
    self.emission_by_decisions = np.zeros(num_periods)
```

```
self.emission_per_period = np.zeros(num_periods)
self.bau_path = np.zeros(num_periods)
self.bau_path[0] = self.ghg_start
self.emission_by_decisions[0] = self.emission_by_time(tree.decision_times[0
period_len = tree.decision_times[1:] - tree.decision_times[:-1]
```

- Calculate the emission by each decision points and the emission for each single period. Take the emission rate to be the average of the emissions at the beginning and the end of the period.

```
for n in range(1, num_periods):
    self.emission_by_decisions[n] = self.emission_by_time(tree.decision_tim
    self.emission_per_period[n] = period_len[n] * (self.emission_by_decisio
    #the average is the average of the emission level at the beginning and
```

- Calculate the GHGs level in the business-as-usual scenario, based on the GHGs level over the emission in the last period.

```
#the total increase in ghg level of 600 (from 400 to 1000) in the bau path
self.emission_to_ghg = (self.ghg_end - self.ghg_start) * self.emission_per_
#emission_to_bau is essentially the last period ghg level divided by the e
self.emission_to_bau = self.emission_to_ghg[-1] / self.emission_per_period[
#find the bau_path based on the last period's emission_to_bau
for n in range(1, num_periods):
    self.bau_path[n] = self.bau_path[n-1] + self.emission_per_period[n]*sel
```