Documentation for Cost.py

1 Introduction

The file cost.py has 2 classes **Cost** and **DLWCost**.

- **Cost** is an abstract cost class for the EZ-Climate model.
- **DLWCost** is a class to evaluate the cost curve for the EZ-Climate model.

2 Model of Mitigation Cost Function

2.1 Traditional Mitigation Cost: GHG Tax Rate

The traditional method to mitigate emission is implementing GHG tax rate. In this model, we consider

- τ: the tax rate per ton of emission.
- g: resulting flow of emissions in gigatonnes of CO_2-equivalent emissions per year, GT CO_2e.
- x: fraction of emission reduced.

The paper gives the marginal abatement cost curve from McKensey's estimates that

$$x(\tau) = 0.0923\tau^{0.414} \tag{1}$$

whose inverse function gives the appropriate tax rate to achieve the mitigation level x:

$$\tau(x) = 314.32x^{2.413} \tag{2}$$

Essentially, this is the marginal cost with GHG tax rate.

Using the envelope theorem, we can find the total social cost corresponding to arbitrary fractional-mitigation x with a coefficient m:

$$K(x) = m \cdot g_0 \cdot x^{3.413} = 92.08 \cdot g_0 \cdot x^{3.413} \tag{3}$$

whose average should be

$$\kappa(x) = \left(\frac{m \cdot g_0}{c_0}\right) x^{3.413} = \left(\frac{92.08 \cdot g_0}{c_0}\right) x^{3.413} \tag{4}$$

where $g_0 = 52$ GT CO_2 represents the level of global annual emissions and $c_0 = \$31$ trillion/year is global consumption in 2015. The equation $\kappa(x)$ expresses the social cost of an arbitrary fractional-mitigation level as a percentage of consumption. We assume that, absent technological change, the function is time-invariant.

2.2 Backstop Technology

In addition to standard mitigation, modern technologies are available for pulling CO_2 directly out of the atmosphere, namely *backstop technologies*.

The backstop technology does not kick in until the mitigation level achieves x^*. We assume the backstop technology is available at a marginal cost of τ^* for the first ton of carbon that is removed from the atmosphere. The marginal cost increases as extraction increases, but it has an upper bound $\widetilde{\tau}$, since we assume that unlimited amounts of CO_2 can be removed at this cost.

Fitting the marginal cost curve to τ^* and $\widetilde{\tau}$ gives us

$$B(x) = \widetilde{\tau} - \left(\frac{k}{x}\right)^{\frac{1}{b}} \tag{5}$$

where

$$b = \frac{\widetilde{\tau} - \tau^*}{(\alpha - 1)\tau^*} \tag{6}$$

$$k = x^*(\widetilde{\tau} - \tau^*)^b \tag{7}$$

By equalizing the marginal costs for the benchmark mitigation level x^* under traditional tax rate $\tau(x^*)$ and backstop technology $B(x^*)$, we find

$$x^* = \left(\frac{\tau^*}{m \cdot \alpha}\right)^{\frac{1}{\alpha-1}} = \left(\frac{\tau^*}{92.08 \cdot 3.413}\right)^{\frac{1}{2.413}} \tag{8}$$

Hence, when we have mitigation level above x^*, the cost from the backstop technology is given by

$$\int_{x^*}^{x} B(s)ds = \int_{x^*}^{x} \widetilde{\tau} - \left(\frac{k}{s}\right)^{\frac{1}{b}} ds = \widetilde{\tau}x - \widetilde{\tau}x^* - \frac{bx(\frac{k}{x})^{\frac{1}{b}}}{b-1} + \frac{bx^*(\frac{k}{x^*})^{\frac{1}{b}}}{b-1} \tag{9}$$

2.3 Technological Change

The marginal cost curve is allowed to decrease at a rate determined by a set of technological change parameters:

- ϕ_0: a constant component.
- ϕ_1: a component linked to X_t, the average mitigation up to time t.

At time t, the total cost curve is given by:

$$\kappa(x, t) = \kappa(x)[1 - \phi_0 - \phi_1 X_t]^t$$

3 Inputs

The python file uses many parameters that can be derived from the cost model described above. Here, we use the **TreeModel** from tree.py and the other parameteres are all floats.

- **a**: α, curvature of the cost function. In our model, $a = 3.413$.
- **g**: m, coefficient of the total traditional cost function. In our model, its value is 92.08.
- **cons_at_0**: c_0, current global consumption. The default value is \$ 30460 billion based on US 2010 values.
- **emit_at_0**: g_0, current GHG emission level.
- **join_price**: τ^*, the lower bound for the marginal cost when the backstop technology kicks in. The example uses 2000.0.
- **max_price**: $\widetilde{\tau}$, the upper bound for the marginal cost from the backstop technology. The example uses 2500.0.
- **tech_const**: ϕ_0, the degree of exogenous technological improvement over time. For example, a value of 1.0 implies that the mitigation cost decreases by 1 percent per year. The example uses 1.5.
- **tech_scale**: ϕ_1, the sensitivity of technological change to previous mitigation level. The example uses 0.0.

4 Python: Cost

Define the class Cost.

```
class Cost(object):
        """Abstract Cost class for the EZ-Climate model."""
        __metaclass__ = ABCMeta

        @abstractmethod
        def cost(self):
                pass

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

4.1 Attributes

The class **DLWCost** has some attributes that stand for important parameters in our cost functions. We have seen some of them in the Section *Parameters*.

- **a**: α, curvature of the cost function. In our model, $a = 3.413$.
- **g**: m, coefficient of the total traditional cost function. In our model, its value is 92.08.
- **max_price**: $\widetilde{\tau}$, the upper bound for the marginal cost from the backstop technology.
- **cbs_level**: x^*, the fractional-mitigation level at which the backstop technology kicks in.
- **cbs_b**: $b = \frac{\widetilde{\tau}-\tau^*}{(\alpha-1)\tau^*}$
- **cbs_k**: $k = x^*(\widetilde{\tau} - \tau^*)^b$
- **cons_per_ton**: $\text{cons\_per\_ton} = \frac{\text{cons\_at\_0}}{\text{emit\_at\_0}} = \frac{c_0}{g_0}$ is the denominator of **cbs_level** that finally gives us $\kappa(x)$.
- **tech_const**: ϕ_0, the degree of exogenous technological improvement over time. For example, a value of 1.0 implies that the mitigation cost decreases by 1 percent per year.
- **tech_scale**: ϕ_1, the sensitivity of technological change to previous mitigation level.

Define the class DLWCost.

```
class DLWCost(Cost):
        """Class to evaluate the cost curve for the EZ-Climate model.

        Parameters
        ----------
        tree : `TreeModel` object
                tree structure used
        emit_at_0 : float
                initial GHG emission level
        g : float --> const of k = gx^a
                initial scale of the cost function
        a : float --> alpha
                curvature of the cost function
        join_price : float -->tau_*
                price at which the cost curve is extended
        max_price : float --> tau_tilda
                price at which carbon dioxide can be removed from atmosphere in unlimi
        tech_const : float  --> alpha_0
                determines the degree of exogenous technological improvement over time
                        of 1.0 implies 1 percent per year lower cost
        tech_scale : float  --> alpha_1
```

```
        determines the sensitivity of technological change to previous mitigat
cons_at_0 : float   --> c_bar
        initial consumption. Default £30460bn based on US 2010 values.

Attributes cbs: cost as a fraction of baseline consumption
----------
tree : `TreeModel` object
        tree structure used
g : float
        initial scale of the cost function
a : float
        curvature of the cost function
max_price : float
        price at which carbon dioxide can be removed from atmosphere in unlimi
tech_const : float
        determines the degree of exogenous technological improvement over time
                of 1.0 implies 1 percent per year lower cost
tech_scale : float
        determines the sensitivity of technological change to previous mitigat
cons_at_0 : float
        initial consumption. Default £30460 billion based on US 2010 values.
cbs_level : float
        constant
cbs_deriv : float
        constant
cbs_b : float
        constant
cbs_k : float
        constant
cons_per_ton : float
        constant

"""

def __init__(self, tree, emit_at_0, g, a, join_price, max_price,
                        tech_const, tech_scale, cons_at_0):
        self.tree = tree
        self.g = g
        self.a = a
        self.max_price = max_price
        self.tech_const = tech_const
        self.tech_scale = tech_scale
        self.cbs_level = (join_price / (g * a))**(1.0 / (a - 1.0)) #after which
        self.cbs_deriv = self.cbs_level / (join_price * (a - 1.0))
        self.cbs_b = self.cbs_deriv * (max_price - join_price) / self.cbs_level
```

```
        self.cbs_k = self.cbs_level * (max_price - join_price)**self.cbs_b
        self.cons_per_ton = cons_at_0 / emit_at_0
```

4.2 Methods

The DLWCost has two methods **cost** and **price** that give us the aggregate social cost and the marginal social cost from the mitigation.

cost: give the total social mitigation cost, given a period and corresponding fractional-mitigation and average mitigation level. The underlying equations are

$$\text{total social mitigation cost} = [1-\phi_0-\phi_1 X_t]^t \cdot \begin{cases} \frac{g_0}{c_0} \cdot m \cdot x^{\alpha}, & x \leq x^* \\ \frac{g_0}{c_0} \cdot m \cdot (x^*)^{\alpha} + \widetilde{\tau} x - \widetilde{\tau} x^* - \frac{bx(\frac{k}{x})^{\frac{1}{b}}}{b-1} + \frac{bx^*(\frac{k}{x^*})^{\frac{1}{b}}}{b-1}, & x > x^* \end{cases}$$

```
    def cost(self, period, mitigation, ave_mitigation):
        """Calculates the mitigation cost for the period. For details about the
        see DLW-paper.

        Parameters
        ----------
        period : int
            period in tree for which mitigation cost is calculated
        mitigation : ndarray
            current mitigation values for period
        ave_mitigation : ndarray
            average mitigation up to this period for all nodes in the peri

        Returns
        -------
        ndarray
            cost

        """
        years = self.tree.decision_times[period]
        tech_term = (1.0 - ((self.tech_const + self.tech_scale*ave_mitigation) /
        cbs = self.g * (mitigation**self.a) #cbs is a power function of mitigat
        bool_arr = (mitigation < self.cbs_level).astype(int) # check if backsto
        if np.all(bool_arr): # cost of traditional mitigation
            c = (cbs * tech_term) / self.cons_per_ton
        else: # cost with backstop technology
            base_cbs = self.g * self.cbs_level**self.a #cost of normal miti
            bool_arr2 = (mitigation > self.cbs_level).astype(int)
            extension = ((mitigation-self.cbs_level) * self.max_price
                        - self.cbs_b*mitigation * (self.cbs_k/mi
```

```
                                        + self.cbs_b*self.cbs_level * (self.cbs_
                #cost of implementing backstop technology
                c = (cbs * bool_arr + (base_cbs + extension)*bool_arr2) * tech_t
        return c
```

Determine the price, or marginal cost, given a period and corresponding fractional-mitigation and average mitigation level.

$$\text{price for each level of fractional-mitigation} = \begin{cases} \tau(x) = m \cdot \alpha x^{\alpha-1}[1 - \phi_0 - \phi_1 X_t]^t, & x \leq x^* \\ \tau(x) = \widetilde{\tau} - \left(\frac{k}{x^*}\right)^{\frac{1}{b}}[1 - \phi_0 - \phi_1 X_t]^t, & x > x^* \end{cases}$$

```
    def price(self, years, mitigation, ave_mitigation):
        """Inverse of the cost function. Gives emissions price for any given
        degree of mitigation, average_mitigation, and horizon.

        Parameters
        ----------
        years : int y
                years of technological change so far
        mitigation : float
                mitigation value in node
        ave_mitigation : float
                average mitigation up to this period

        Returns
        -------
        float
                the price.

        """
        tech_term = (1.0 - ((self.tech_const + self.tech_scale*ave_mitigation) /
        if mitigation < self.cbs_level:
                return self.g * self.a * (mitigation**(self.a-1.0)) * tech_term
        else:
                return (self.max_price - (self.cbs_k/mitigation)**(1.0/self.cbs_
```