Documentation for Cost.py

1 Introduction

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

- **Cost** is an abstract cost class that defines abstract methods for cost and price.
- **DLWCost** is a class to evaluate the cost curve for the EZ-Climate model, based on the functions we give in the following section.

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, i.e. 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. We are interested in $\kappa(x)$, the cost to the society when a GHG tax rate of τ is imposed. We can calculate $\kappa(x)$ using the envelope theorem. Intuitively, GHG emissions are an input to the production process that generates consumption goods. At any tax rate τ, assuming agents choose the level of GHG emissions $g(\tau)$ so as to maximize consumption given τ, then the marginal cost of increasing the tax rate must be the quantity of emissions at that tax rate, that is:

$$\frac{dc(\tau)}{d\tau} = -g(\tau) \tag{3}$$

Thus, to calculate the consumption associated with a GHG tax rate of τ we integrate this expression and get

$$c(\tau) = \bar{c} - \int_0^\tau g(s)ds \tag{4}$$

where $\bar{c}$ is the endowed level of consumption (assuming zero damages). However, this equation is correct only if the GHG tax is purely dissipative - that is, if the government were to collect the tax and then waste 100% of the proceeds. In our analysis, we instead assume that the tax is non-dissipative, meaning that the proceeds of the tax $(g(\tau) \cdot \tau)$ would be refunded lump-sum, making the decrease in consumption just equal to the distortionary effect of the tax (in dollars) which is:

$$K(\tau) = \int_0^\tau g(s)ds - g(\tau) \cdot \tau \tag{5}$$

Writing $g(\tau) = g_0(1 - x(\tau))$, where g_0 is the baseline level of GHG emissions, we can rewrite $K(\tau)$ as:

$$K(\tau) = g_0 \int_0^\tau (1 - x(s))ds - \tau g_0(1 - x(\tau)) \tag{6}$$

$$= g_0 \left[\tau - \int_0^\tau x(s)ds\right] - \tau g_0 + \tau g_0 x(\tau) \tag{7}$$

$$= g_0 \left[\tau x(\tau) - \int_0^\tau x(s)ds\right] \tag{8}$$

Then by substituting and simplifying we can get the total cost K as a function of the tax rate τ:

$$K(\tau) = g_0 \left[0.09230 \cdot \tau^{1.414} - 0.06526 \cdot \tau^{1.414}\right] = g_0 \cdot 0.02704 \cdot \tau^{1.414} \tag{9}$$

Substituting gives K as a function of fractional-mitigation x: the code takes m as an input and set it as 92.08, and I do not know exactly why it is not taken as a fixed coefficient

$$K(x) = m \cdot g_0 \cdot x^\alpha = 92.08 \cdot g_0 \cdot x^{3.413} \tag{10}$$

where total cost $K(x)$ is expressed in dollars. Finally, we divide by 2015 aggregate consumption c_0 to determine the average cost as a fraction of baseline consumption:

$$\kappa(x) = \left(\frac{m \cdot g_0}{c_0}\right) x^\alpha = \left(\frac{92.08 \cdot g_0}{c_0}\right) x^{3.413} \tag{11}$$

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 total cost of an arbitrary fractional-mitigation level as a percentage of consumption, and we hold that fixed in all periods except for the impact of technological changes. 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*. Some examples are carbon dioxide removal (CDR) or direct carbon removal (DCR) (National Research Council, 2015). Here we need to consider the cost of CO_2 removal.

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 $\widetilde{\tau}$, so unlimited amounts of CO_2 can be removed as the marginal cost approaches $\widetilde{\tau} \geq \tau^*$. The paper assumes a price of \$350 per ton for τ^* and a price of \$400 per ton for $\widetilde{\tau}$. However, τ^* and $\widetilde{\tau}$ equal to 2000 and 2500 respectively in the example. The underlying cost curve for emissions mitigation imply that the backstop technology kicks in at mitigation levels above 104%, but we try to determine this threshold, namely x^*, in the code. We use x^* here because the x_0 in the original paper is a little bit confusing.

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

$$B(x) = \widetilde{\tau} - \left(\frac{k}{x}\right)^{\frac{1}{b}} \tag{12}$$

$$B(x^*) = \widetilde{\tau} - \left(\frac{k}{x^*}\right)^{\frac{1}{b}} = \tau^* \tag{13}$$

since $\widetilde{\tau}$ is the upper bound. If we impose a smooth-pasting condition, in which the derivative of the marginal cost curve is continuous at x^*, we can get:

$$b = \frac{\widetilde{\tau} - \tau^*}{(\alpha - 1)\tau^*} \tag{14}$$

$$k = x^*(\widetilde{\tau} - \tau^*)^b \tag{15}$$

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

$$x^* = \left(\frac{\tau^*}{m \cdot \alpha}\right)^{\frac{1}{\alpha - 1}} = \left(\frac{\tau^*}{92.08 \cdot 3.413}\right)^{\frac{1}{2.413}} \tag{16}$$

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

$$\int_{x^*}^{x} B(s)ds = \int_{x^*}^{x} \widetilde{\tau} - \left(\frac{k}{s}\right)^{\frac{1}{b}} ds = \widetilde{\tau}x - \widetilde{\tau}x^* - \frac{bx(\frac{k}{x})^{\frac{1}{b}}}{b-1} + \frac{bx^*(\frac{k}{x^*})^{\frac{1}{b}}}{b-1} \tag{17}$$

2.3 Technological Change

Since the first period, the marginal cost curve is allowed to decrease at a rate determined by a set of technological change parameters, ϕ_0 and $\phi_1 X_t$:

- ϕ_0: a constant component.
- ϕ_1: a component linked to mitigation efforts. It has to do with the average mitigation up to time t:

  $X_t = \frac{\sum_{s=0}^{t} g_s \cdot x_s}{\sum_{s=0}^{t} g_s}$, where g_s is the flow of GHG emissions into the atmosphere in period s

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

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

This functional form allows for easy calibration. For example, if $\phi_0 = 0.005$ and $\phi_1 = 0.01$, then with average mitigation of 50%, marginal costs decrease as a percentage of consumption at a rate of 1% per year.

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 all the other parameters are floats.

- **a**: α, curvature of the cost function. In our model, $a = 3.413$, as in equation 10.
- **g**: m, coefficient of the total traditional cost function. In our model, its value is 92.08, see equation 10.
- **cons_at_0**: c_0, current global consumption. The default value is $ 30460 billion based on US 2010 values. cannot find the source of this number
- **emit_at_0**: g_0, current GHG emission level, derived from bau.py.
- **join_price**: τ^*, the lower bound for the marginal cost regarding the backstop technology when it kicks in. The example uses 2000.0.
- **max_price**: $\widetilde{\tau}$, the upper bound for the marginal cost regarding the backstop technology. The example uses 2500.0.
- **tech_const**: ϕ_0, the degree of exogenous technological improvement over time in percentage. 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, in percentage, of technological change to previous mitigation efforts. The example uses 0.0.

4 Python: Cost

Define the abstract 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 **Inputs**.

- **a**: α, curvature of the cost function. In our model, $a = 3.413$. (see equation 10)
- **g**: m, coefficient of the total traditional cost function. In our model, its value is 92.08. (see equation 10)
- **max_price**: $\widetilde{\tau}$, the upper bound for the marginal cost regarding the backstop technology when it kicks in. The example uses 2000.0.
- **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**: 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 $\kappa(x)$. See equation 16.
- **tech_const**: ϕ_0, the degree of exogenous technological improvement over time in percentage. 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, in percentage, of technological change to previous mitigation efforts. The example uses 0.0.

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**, corresponding to the abstract methods in class **Cost**. They give us the aggregate social cost and the marginal social cost concerning the given mitigation levels.

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

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

```
    def cost(self, period, mitigation, ave_mitigation):
        """Calculates the mitigation cost for the period. For details about th
        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
```

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

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

```
    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_
```