\documentclass[12pt]{article}

\usepackage[utf8]{inputenc}

\usepackage{latexsym}

\usepackage{float}

\usepackage{parskip}

\usepackage{amsfonts}

\usepackage{caption}

\usepackage{commath}

\usepackage{amssymb,amsmath}

\usepackage{graphicx}

\usepackage[backend=bibtex,style=numeric,sorting=none]{biblatex}

\usepackage[export]{adjustbox}

\usepackage{subcaption}

\usepackage[top=1in, bottom=1in,left=1in, right=1in]{geometry}

\usepackage{minted}

\usepackage{color}

\newcommand{\tc}{\textcolor{red}}

\newcommand{\bs}{\boldsymbol}

\newenvironment{alphafootnotes}

{\par\edef\savedfootnotenumber{\number\value{footnote}}

\renewcommand{\thefootnote}{\alph{footnote}}

\setcounter{footnote}{0}}

{\par\setcounter{footnote}{\savedfootnotenumber}}

\begin{document}

\begin{center}

\Large\bf{Documentation for forcing.py}

\end{center}

\normalsize

\pagenumbering{arabic}

\section{Introduction}

\textbf{Radiative Forcing} is the difference between sunlight absorbed by the Earth and energy radiated back to space. Positive radiative forcing means Earth receives more incoming energy from sunlight than it radiates to space. This net gain of energy will cause warming. Conversely, negative radiative forcing means that Earth loses more energy to space than it receives from the sun, which produces cooling. This is to say, positive forcing warms the system, while negative forcing cools it. Causes of radiative forcing include changes in insolation and the concentrations of radiatively active gases, commonly known as greenhouse gases and aerosols. This is why we include forcing as a part of our damage model.

According to Wikipedia, radiative forcing can be used to estimate a subsequent change in equilibrium surface temperature $\Delta T\_t$ arising from that forcing via the equation

\begin{equation}

\Delta T\_t = \lambda \Delta F

\end{equation}

where $\lambda$ is the climate sensitivity and $\Delta F$ is the radiative forcing.

In our case, we take radiative forcing into consideration to better model the damage function. Basically, at each decision point $t$, the representative agent chooses a mitigation level $x\_t$. The consequent GHG emission results in radiative forcing $CRF\_t$, which in turn determines the GHG level and average mitigation $\bar{x}\_t$ up to $t$. We assume that the 3 base scenarios have constant mitigation levels $\bar{x}$ and calculate their cumulative radiative forcings $CRF\_t$ for reference. Given any path with mitigation level $x\_t$'s, we calculate its cumulative radiative forcings $CRF\_t$ and average mitigation level $\bar{x}\_t$. From $\bar{x}\_t$ we would like to infer the consequent damage with the $D\_t(CRF\_t)$ function that we interpolate from the base scenario simulations in damage.py. To serve this end, forcing.py is essentially a class that computes the cumulative radiative forcing $CRF\_t$ and GHG level $GHG\_t$ for a node with mitigation path $x\_t$ up to period $t$ in the EZ-Climate model.

\section{Model of $CRF\_t$ as a Function of $x\_t$}

As a cumulative variable, $CRF\_t$ depends on the mitigation path up to point $t$ in time. The cumulative mitigation is given by:

\begin{equation}

X\_t = \frac{\sum\_{s=0}^t g\_s \cdot x\_s}{\sum\_{s=0}^t g\_s}

\end{equation}

where $g\_s$ is the flow of GHG emissions into the atmosphere in period $s$, for each period up to $t$, absent any mitigation. The cumulative GHG emissions that must be absorbed into the atmosphere or oceans is $G\_t(1-X\_t)$, where $G\_t=\sum\_{s=0}^t g\_s$ denotes the cumulative emissions under the BAU scenario.

The radiative forcing is assumed to be given by a log-function fitted to the 3 base case scenarios, named Representative Concentration Pathway (RCP) scenarios in the paper. The carbon absorption itself is similarly fit to the RCP scenarios, and is assumed to be proportional to the difference between the GHG level in the atmosphere and the cumulative carbon absorption up to that point in time, raise to a power.

According to the paper, radiative forcing in a ten-year interval is given by

\begin{equation}

5.351\cdot [log(\text{GHG})-log(278.063)]

\end{equation}

where GHG is the average level of atmospheric $CO\_2$. The carbon absorption in a ten-year interval is given by

\begin{equation}\label{absorption}

0.94835\cdot\abs{\text{GHG}-(285.6268+0.88414\cdot \sum absorption)}^{0.741547}

\end{equation}

where the sum is over absorption in previous periods, represented by \textbf{cum\\_sink} in the code.

The paper does not explain enough about the way forcing.py calculates the variables, but we can formulate the process without understanding the parameters. In particular, forcing.py uses \textbf{bau.py} to get GHG emissions under the business-as-usual scenario. In case of confusion, we denote it here as $\bar{g}\_t$ and the GHG emission in the given path $g\_t$ for $t \in \{0,1,2,3,4,5,6\}$. We look into each period of the path in concern by unit called \textbf{subinterval} and compare the variables with those of the business-as-usual scenario. Then we calculate \textbf{forcing} and \textbf{absorbing}, which determine $CRF\_t$ and GHG level $GHG\_t$. Their beginning values are:

\begin{equation}

\begin{cases}

CRF\_0 & = 4.926\\

GHG\_0 & = 400\\

sink\_0 & = 35.396

\end{cases}

\end{equation}

Here is how we calculate the variables for each subinterval:

\begin{itemize}

\item For each period $t$:

\begin{equation}

\begin{cases}

\text{beginning emission }& g\_{t,0}=(1-x\_t)\times \bar{g}\_{t}\\

\text{ending emission}& g\_{t,t}=\begin{cases}

(1-x\_t)\times \bar{g}\_{t+1}, &\text{if }t<5\\

(1-x\_t)\times \bar{g}\_{t}, & \text{else}

\end{cases}

\end{cases}

\end{equation}

\item For each subinterval $i$:

\begin{equation}

\label{set}

\begin{cases}

\text{p\\_co2}\_i& =0.71\times\left[g\_{t,0}+i\times\frac{g\_{t,t}-g\_{t,0}}{\text{number of subintervals in period }t}\right]\\

\text{p\\_c}\_i&=\frac{\text{p\\_co2}\_i}{3.67}\\

\text{add\\_p\\_ppm}\_i&=\text{length of subinterval}\times \frac{\text{p\\_c}\_i}{2.13}\\

\text{lsc}\_i&=285.6268+\text{cum\\_sink}\_i \times0.88414\\

\text{absorption}\_i& = 0.5\times0.94835\times\lvert GHG\_i-\text{lsc}\_i\rvert^{0.741547}\\

\text{cum\\_sink}\_i & =\text{cum\\_sink}\_{i-1}+ \text{absorption}\_i\\

GHG\_i &=GHG\_{i-1}+\text{add\\_p\\_ppm}\_i-\text{absorption}\_i\\

CRF\_i & = CRF\_{i-1} + 0.13183\times\lvert GHG\_i-315.3785\rvert ^{0.607773}

\end{cases}

\end{equation}

\end{itemize}

\section{Inputs}

The methods embedded in the Forcing class require similar inputs that are explained below.

\begin{itemize}

\item \textbf{m}: an array of fractional mitigation levels that are denoted $x\_t$ in the paper.

\item \textbf{node}: an integer that represents the node in the \textbf{TreeModel} for which forcing-related values are to be calculated.

\item \textbf{tree}: the \textbf{TreeModel} object whose tree structure is to be used.

\item \textbf{bau}: the \textbf{BusinessAsUsual} object that gives the emission levels under the business-as-usual scenario.

\item \textbf{subinterval\\_len}: a float that represents the length of a subinterval. It is the size of intervals we model the probability of hitting a Tipping Point in the damage function. Within one period, the subintervals of this length are call \textbf{increments}. In our example, this value is 5. This has to do with the \textbf{Tipping Point} that we introduce in damage function.

\item \textbf{returning}: an optional string that selects the output. Valid returns include "forcing", "ghg", and "both".

\begin{itemize}

\item "forcing": returns the forcing only;

\item "ghg": returns the GHGs level only;

\item "both": returns both the GHGs level and forcing.

\end{itemize}

\end{itemize}

\section{Python: Forcing}

\begin{minted}{python}

from \_\_future\_\_ import division

import numpy as np

\end{minted}

\subsection{Attributes}

The attributes of the Forcing class are basically parameters of the theoretical model, so they are all of the float type. \tc{The meanings of those in red are still unknown.} Equation \ref{absorption} and \ref{set} may help as references.

\begin{itemize}

\item \tc{sink\\_start} = 35.596, the beginning value for cumulative absorption.

\item \tc{forcing\\_start} = 4.926

\item \tc{forcing\\_p1} = 0.13173

\item \tc{forcing\\_p2} = 0.607773

\item \tc{forcing\\_p3} = 315.3785

\item \textbf{absorption\\_p1} = 0.94835

\item \textbf{absorption\\_p2} = 0.741547

\item \textbf{lsc\\_p1} = 285.6268

\item \textbf{lsc\\_p2} = 0.88414

\end{itemize}

\begin{minted}{Python}

class Forcing(object):

"""Radiative forcing for the EZ-Climate model. Determines the excess energy created

by GHGs in the atmosphere.

Attributes

----------

sink\_start : float

sinking constant

forcing\_start : float

forcing start constant

forcing\_p1 : float

forcing constant

forcing\_p2 : float

forcing constant

forcing\_p3 : float

forcing constant

absorption\_p1 : float

absorption constant

absorption\_p2 : float

absorption constant

lsc\_p1 : float

class constant

lsc\_p2 : float

class constant

"""

\end{minted}

\begin{minted}{python}

# parameters that I have no idea about

sink\_start = 35.596

forcing\_start = 4.926

forcing\_p1 = 0.13173

forcing\_p2 = 0.607773

forcing\_p3 = 315.3785

absorption\_p1 = 0.94835

absorption\_p2 = 0.741547

lsc\_p1 = 285.6268

lsc\_p2 = 0.88414

\end{minted}

\subsection{Methods}

There are 3 methods in this file, while \textbf{forcing\\_at\\_node} and \textbf{ghg\\_level\\_at\\_node} simply executes \textbf{forcing\\_and\\_ghg\\_at\\_node} upon different requests. \textbf{forcing\\_and\\_ghg\\_at\\_node} implements the computation along a deterministic path and gives outputs specified by \textbf{forcing\\_at\\_node} and \textbf{ghg\\_level\\_at\\_node}.

\textbf{forcing\\_and\\_ghg\\_at\\_node}: calculates the radiative forcing based on GHGs evolution that leads up to damage calculation. The steps are given below.

\begin{minted}{python}

@classmethod

def forcing\_and\_ghg\_at\_node(cls, m, node, tree, bau, subinterval\_len, returning="forcing"):

"""Calculates the radiative forcing based on GHG evolution leading up to the

damage calculation in `node`.

Parameters

----------

m : ndarray

array of mitigations

node : int

node for which forcing is to be calculated

tree : `TreeModel` object

tree structure used

bau : `BusinessAsUsual` object

business-as-usual scenario of emissions

subinterval\_len : float

subinterval length

returning : string, optional

\* "forcing": implies only the forcing is returned

\* "ghg": implies only the GHG level is returned

\* "both": implies both the forcing and GHG level is returned

Returns

-------

tuple or float

if `returning` is

\* "forcing": only the forcing is returned

\* "ghg": only the GHG level is returned

\* "both": both the forcing and GHG level is returned

"""

\end{minted}

\begin{itemize}

\item Specify the case when the node is 0.

\begin{minted}{python}

#for the start state, return 0 for forcing and ghg\_start for the ghg\_level

#call bau to get the ghg level

if node == 0:

if returning == "forcing":

return 0.0

elif returning== "ghg":

return bau.ghg\_start

else:

return 0.0, bau.ghg\_start

\end{minted}

\item Based on the node given, find its period, path, and decision times with \textbf{TreeModel}.

\item Determine the number of increments within each period, of the length specified by \textbf{subinterval\\_len}. (This number is converted into integer later, for convenience of computation, so estimation errors do exist.)

\begin{minted}{Python}

# get the period and the path that the target node are in

period = tree.get\_period(node)

path = tree.get\_path(node, period)

# the decision time is the time when we make a mitigation, i.e. an array like [15,30,45,70]

period\_lengths = tree.decision\_times[1:period+1] - tree.decision\_times[:period]

#increments are the number counts of subintervals within a period

increments = period\_lengths/subinterval\_len

\end{minted}

\item Assign the starting values of forcing and GHGs level.

\begin{minted}{python}

#assign beginning values

cum\_sink = cls.sink\_start

cum\_forcing = cls.forcing\_start

ghg\_level = bau.ghg\_start

\end{minted}

\item For each period that a node has undergone, determine its beginning and ending GHG levels $g\_t$ under mitigation with the base case outputs from \textbf{bau.emission\\_by\\_decisions}. We assume that the emission level remains constant since the second to last period.

\begin{minted}{python}

for p in range(0, period):

#for each period, we calculate the start\_emission and end\_emission from bau.emission\_by\_decisions

#! problem: when will the act takes in to effect? either way, code here should be wrong.

#emission\_by\_decision: the emission level at a decision point

start\_emission = (1.0 - m[path[p]]) \* bau.emission\_by\_decisions[p] # this is a attr of func: emission\_by\_step, might needed recode this

if p < tree.num\_periods-1: #if not too late to implement mitigation

end\_emission = (1.0 - m[path[p]]) \* bau.emission\_by\_decisions[p+1] #if not the final states, the end emission is

else:

end\_emission = start\_emission #emission level remains constant since the second to last period

increment = int(increments[p])

\end{minted}

\item Within each period that a node has undergone, interpolate linearly the emission levels at the beginnings of all the increments in concern. Based on the emission levels, calculate the amounts of consequent ppm, absorption, and forcing.

\item Calculate the GHG level at the beginning of each increment by updating values for forcing and absorption based on the GHG emissions.

\begin{itemize}

\item \textbf{p\\_co2} \tc{precise meanings unknown}

\item \textbf{p\\_c}

\item \textbf{add\\_p\\_ppm}

\item \textbf{absorption}: $0.5\times 0.94835\abs{GHG-(285.6268+0.88414\cdot \sum absorption)}^{0.741547}$\\

This is the carbon absorption in a 5-year interval. Since our subinterval is of length 5, the absorption should be half of that in Equation \ref{absorption}.

\item \textbf{cum\\_forcing}: the cumulative forcing is given by $0.13173\times\abs{GHG-315.3785}^{0.607773}$

\end{itemize}

\begin{minted}{python}

# for each increment in a period, the forcing is affecting ghg\_level in the end

for i in range(0, increment):

#allocate the emission level change across time

p\_co2\_emission = start\_emission + i \* (end\_emission-start\_emission) / increment

p\_co2 = 0.71 \* p\_co2\_emission

p\_c = p\_co2 / 3.67

add\_p\_ppm = subinterval\_len \* p\_c / 2.13

lsc = cls.lsc\_p1 + cls.lsc\_p2 \* cum\_sink

absorption = 0.5 \* cls.absorption\_p1 \* np.sign(ghg\_level-lsc) \* np.abs(ghg\_level-lsc)\*\*cls.absorption\_p2

cum\_sink += absorption

cum\_forcing += cls.forcing\_p1\*np.sign(ghg\_level-cls.forcing\_p3)\*np.abs(ghg\_level-cls.forcing\_p3)\*\*cls.forcing\_p2

ghg\_level += add\_p\_ppm - absorption

\end{minted}

\item Return the outcome according to the selection.

\begin{minted}{python}

if returning == "forcing":

return cum\_forcing

elif returning == "ghg":

return ghg\_level

else:

return cum\_forcing, ghg\_level

\end{minted}

\end{itemize}

\textbf{forcing\\_at\\_node}: returns the cumulative radiative forcing $CRF\_t$ at a given node that leads up to the damage calculation.

\begin{minted}{python}

@classmethod

def forcing\_at\_node(cls, m, node, tree, bau, subinterval\_len):

"""Calculates the forcing based mitigation leading up to the

damage calculation in `node`.

Parameters

----------

m : ndarray

array of mitigations in each node.

node : int

the node for which the forcing is being calculated.

Returns

-------

float

forcing

"""

return cls.forcing\_and\_ghg\_at\_node(m, node, tree, bau, subinterval\_len, returning="forcing")

\end{minted}

\textbf{ghg\\_level\\_at\\_node}: returns the GHGs level at a given node that leads up to the damage calculation.

\begin{minted}{python}

@classmethod

def ghg\_level\_at\_node(cls, m, node, tree, bau, subinterval\_len):

"""Calculates the GHG level leading up to the damage calculation in `node`.

Parameters

----------

m : ndarray

array of mitigations in each node.

node : int

the node for which the GHG level is being calculated.

Returns

-------

float

GHG level at node

"""

return cls.forcing\_and\_ghg\_at\_node(m, node,tree, bau, subinterval\_len, returning="ghg")

\end{minted}

\begin{minted}{python}

\end{minted}

\begin{minted}{python}

\end{minted}

\begin{minted}{python}

\end{minted}

\end{document}