



# Ultimate Cost of Carbon computation

## The model defined in function calls

Running this document (costs.ipynb) interactively will require python 3 with libraries matplotlib, math, numpy, and IPython. The pdf version of the file is a non-interactive printout of the original.

The model is divided into two components, the code for which are in the following module. One function (climateModel) simulates climate dynamics and the other (econModel) estimates costs.

The arguments to the physical model (function climateModel) are

Parameter Name	Description	Value note
CReleaseGton	Gigatons of anthropogenic carbon	the buffer chemistry is good for the range 1000 - 5000
CFeedbackFactor	Fraction by which the natural carbon cycle amplifies the human source	< 1.5
oceanAcidTime	Time scale for pH recovery of the ocean, years	models find 1-10kyr
thermostatTime	Time scale for atmospheric CO2 recovery	models find 200kyr, must be longer than 100 kyr
warmingTime	Time scale for temperature equilibration	governed by ocean overturning
iceMeltingTime	Time scale for collapsing ice sheets, years	models predict a few thousand years
dt2X	The equilibrium climate sensitivity, degrees C per doubling of CO2	IPCC range 2.5 - 4.5

The parameters other than the magnitude of the carbon release are grouped into an array for convenience in uncertainty analysis. The model spans one million years of time, in time steps which become larger through time as things change more slowly. The model returns the following lists (arrays) of values:

Results Name	Description
times	A list of time points in years
deltaTs	Value of each time step in years
CAtmFactors	Atmospheric CO2 concentrations relative to preanthropogenic
temperatures	Equilibrium radiative temperature anomalies
seaLevels	Sea Level in meters

The climate module imposes a two-stage drawdown on the atmospheric pCO<sub>2</sub> perturbation. The timing of the first stage of drawdown is determined by the pH recovery of the oceans (oceanAcidTime). The timing of the second (final) recovery is set by silicate weathering in the CO<sub>2</sub> thermostat (thermostatTime). The airborne fractions of the CO<sub>2</sub> at each stage were taken from model results from the tailMIP model intercomparison project, Archer et al., (2009), from model values using 1,000 and 5,000 Gton C releases, at time points of 1,000 and 10,000 years respectively. Airborne fractions for other release magnitudes in the model are interpolated between the 1,000 and 5,000 Gton C release values.

The model neglects time-dependence in the global temperature anomaly, and derives a list of temperatures using the equilibrium climate sensitivity and the ratio of atmospheric CO2 to natural.

Sea level changes relative to temperature are based on the correlation between sea level and global mean temperature in the paleo reconstructions, from Archer and Brovkin (). The time scale for ice sheet melting is a parameter in the argument list (iceMeltingTime); the time scale for ice sheet recovery is taken to be the thermostat time scale (thermostatTime).

The economic model takes results from the climate model and adds three new parameters:

Parameter Name	Description
climCostFactor	Economic penalty due to climate change, in percent GDP per degree C
SLCostFactor	Economic penalty due to sea level rise, in percent GDP for complete melting (70 m)
econRecoveryTime	Economic recovery time for both types of cost. This could be a soil time of 10kyr or a CO2 time of 200 kyr

The economic model returns the following results:

Result Name	Description
climCosts	A list of economic penalties due to climate change, in % GDP
seaLevelCosts	A list of economic penalties due to sea level rise, in % GDP
cumClimCost	Cumulative cost due to climate, in dollars / ton C
cumSeaLevelCost	Cumulative cost due to sea level, in dollars / ton C

Also included in the following code section is a wrapper for both models, with all parameters packaged in a

```

In [1]: import matplotlib.pyplot as plt
import math
import numpy as np
from IPython.display import Markdown, display

def climateModelWrapper( parmList ):
    CReleaseGton = parmList[0]
    CFeedbackFactor = parmList[1]
    oceanAcidTime = parmList[2]
    thermostatTime = parmList[3]
    warmingTime = parmList[4]
    iceMeltingTime = parmList[5]
    dT2x = parmList[6]
    times, deltaTs, CAtmFactors, temperatures, seaLevels = \
        climateModel( CReleaseGton, CFeedbackFactor, oceanAcidTime, therm
ostatTime, \
                        warmingTime, iceMeltingTime, dT2x )
    return times, deltaTs, CAtmFactors, temperatures, seaLevels

def climateModel( CReleaseGton, CFeedbackFactor, oceanAcidTime, thermost
atTime, warmingTime, iceMeltingTime, dT2x ):

    SLMax = 70.
    seaLevelMperC = 17
    CAtmNatGton = 500    # Gton C in a natural atmosphere, used for radia
tive forcing calculation
    CReleaseTotal = CReleaseGton * ( 1. + CFeedbackFactor )
    CAirborneOcnEquil = 0.1 + (CReleaseTotal - 1000)/4000 * 0.05
    CAirborneOcnNeut = 0.1 + (CReleaseTotal - 1000)/4000 * 0.15

    CAtmFactors = []
    temperatures = []
    seaLevels = []
    times = []
    deltaTs = []

    tInit = 100    # these parameters produce a series of time points t
hat plot nicely in logs to 1 million years
    tFactor = 1.08
    nTSteps = 120

    tEvolving = 0

    timeNow = tInit
    for iTime in range(0,nTSteps):
        dt = timeNow * ( tFactor - 1. )
        timeNow *= tFactor
        CShort = CAirborneOcnEquil * CReleaseGton / CAtmNatGton \
            * math.exp( -timeNow / oceanAcidTime )
        CLong = CAirborneOcnNeut * CReleaseGton / CAtmNatGton \
            * math.exp( -timeNow / thermostatTime )
        CAtmFactor = 1 + CShort + CLong
        tEquil = math.log( CAtmFactor ) / math.log(2) * dT2x
        if iTime == 0:
            tEvolving = tEquil * 0.8
        elif dt < warmingTime:

```

```

        tEvolving += ( tEquil - tEvolving ) * dt / warmingTime
    else:
        tEvolving = tEquil

    seaLevel = tEvolving * seaLevelMperC \
        * ( 1 - math.exp( - timeNow / iceMeltingTime ) ) \
        * ( math.exp( - timeNow / thermostatTime ) )
    if seaLevel > SLMax:
        seaLevel = SLMax

    times.append( timeNow )
    deltaTs.append( dt )
    CAtmFactors.append( CAtmFactor )
    temperatures.append( tEvolving )
    seaLevels.append( seaLevel )
    return times, deltaTs, CAtmFactors, temperatures, seaLevels

def econModel( times, deltaTs, temperatures, seaLevels, CReleaseGton, \
    climCostFactor, SLCostFactor, econRecoveryTime ):
    SLMax = 70
    climCosts = []
    seaLevelCosts = []
    cumClimCost = 0
    cumSeaLevelCost = 0
    for index, time in enumerate(times):
        climCost = temperatures[index] * climCostFactor \
            * math.exp( - time / econRecoveryTime )
        seaLevelCost = seaLevels[index] * SLCostFactor / SLMax \
            * math.exp( - time / econRecoveryTime )
        climCosts.append( climCost )
        seaLevelCosts.append( seaLevelCost )
        cumClimCost += climCost * deltaTs[index] / CReleaseGton * 1E3 #
        cumSeaLevelCost += seaLevelCost * deltaTs[index] / CReleaseGton
    * 1E3
    return climCosts, seaLevelCosts, cumClimCost, cumSeaLevelCost

```

## Simulations for 1000, 5000 Gton C releases, with economic recovery on geomorphic or carbon cycle time scales

The next module runs the model for an array of values of the amount of carbon released and the time scale for recovery of the economic damages. Values can be added or deleted to the arrays at the top of the module to expand or contract the number of runs that will be plotted in modules below this.

```

In [6]: # run for all combinations of the following parameter settings
CReleaseGtons      = [ 1000, 5000 ]

# used in Monte Carlo, below, defined here to find log means as defaults
parmNames = [ "CReleaseGton", \
               "CFeedbackFactor", "oceanAcidTime", "thermostatTime", "war
               mingTime", "iceMeltingTime", "dT2x", \
               "climCostFactor", "SLCostFactor", "econRecoveryTime", \
               "AllGeo", "All" ]
parmRanges = [ [1000,5000],      # 0, Gton C released \
                [0.1,0.5],      # 1, magnitude of possible carbon cycl
                e feedback \
                [2000.,8000.],   # 2, oceanAcidTime \
                [1.e5,4.e5],     # 3, thermostatTime, with factor of 2
                uncertainty on either side \
                [100.,1000.],    # 4, warmingTime, in reality a range f
                rom different parts of the ocean \
                [300.,3000.],    # 5, iceMeltingTime, from Heinrich eve
                nts (fast) and models (slow) \
                [1.5,4.5],       # 6, dT2x, range from IPCC
                [1,4],           # 7, climCostFactor
                [1,15],          # 8, SLCostFactor
                [10000,200000]]  # 9, econRecoveryTime

numParms = 10
numParmsGeo = 7

# geometric means to use in base scenario
climBaseParmList = []
parmLogRanges = []
for parmRange in parmRanges:
    parmLogRange = [ math.log(parmRange[0]), math.log(parmRange[1]) ]
    parmLogRanges.append( parmLogRange )
    climBaseParmList.append( math.exp( (parmLogRange[0]+parmLogRange[1])
    /2.))

#for i in range(1,6):
#    print(parmNames[i],climBaseParmList[i])

CFeedbackFactor = climBaseParmList[1]
oceanAcidTime = climBaseParmList[2]
thermostatTime = climBaseParmList[3]
warmingTime = climBaseParmList[4]
iceMeltingTime = climBaseParmList[5]
dT2x = climBaseParmList[6]

CAtmFactorLists = []
temperatureLists = []
seaLevelLists = []
for CReleaseGton in CReleaseGtons:
    times, deltaTs, CAtmFactors, temperatures, seaLevels = \
        climateModel( CReleaseGton, CFeedbackFactor, oceanAcidTime, ther
        mostatTime, warmingTime, \
                       iceMeltingTime, dT2x )
    CAtmFactorLists.append( CAtmFactors )
    temperatureLists.append( temperatures )
    seaLevelLists.append( seaLevels )

```

## Figures from the simulations

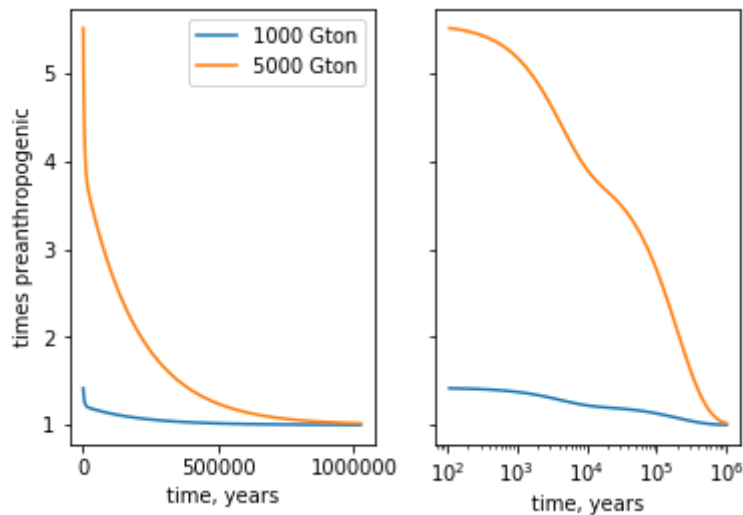
Time evolution of model parameters will be plotted on a linear time scale on the left, and a log scale on the right.

```
In [4]: display(Markdown("## Atmospheric CO2"))

fig,axarr = plt.subplots(nrows=1,ncols=2,sharex=False,sharey=True,squeeze=False)

for index, CReleaseGton in enumerate(CReleaseGtons):
    ax = axarr[0][0]
    myString = str(CReleaseGton) + " Gton"
    ax.plot(times,CAtmFactorLists[index],label=myString)
    ax.set_xlabel("time, years")
    ax.set_ylabel("times preanthropogenic")
    ax.legend()
    ax.set_xticks([0,5e5,1e6])
    ax = axarr[0][1]
    ax.semilogx(times,CAtmFactorLists[index])
    ax.set_xlabel("time, years")
plt.savefig("figure_01.pdf")
plt.show()
```

## Atmospheric CO<sub>2</sub>

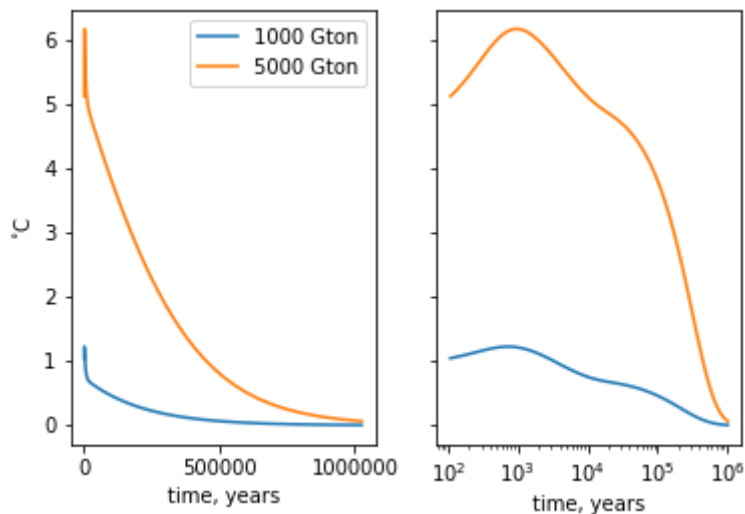


```
In [5]: display(Markdown("## Global Temperature Anomaly"))

fig,axarr = plt.subplots(nrows=1,ncols=2,sharex=False,sharey=True,squeeze=False)

for index, CReleaseGton in enumerate(CReleaseGtons):
    ax = axarr[0][0]
    myString = str(CReleaseGton) + " Gton"
    ax.plot(times,temperatureLists[index],label=myString)
    ax.set_xlabel("time, years")
    ax.set_ylabel("$^{\circ}\text{C}$")
    ax.legend()
    ax.set_xticks([0,5e5,1e6])
    ax = axarr[0][1]
    ax.semilogx(times,temperatureLists[index])
    ax.set_xlabel("time, years")
plt.savefig("figure_02.pdf")
plt.show()
```

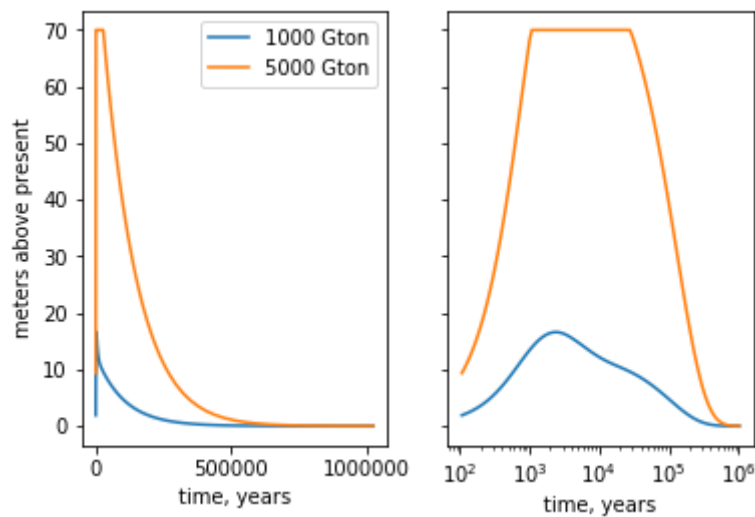
## Global Temperature Anomaly





```
In [6]: display(Markdown("## Sea Level"))
fig, axarr = plt.subplots(nrows=1, ncols=2, sharex=False, sharey=True, squeeze=False)
for index, CReleaseGton in enumerate(CReleaseGtons):
    ax = axarr[0][0]
    myString = str(CReleaseGton) + " Gton"
    ax.plot(times, seaLevelLists[index], label=myString)
    ax.set_xlabel("time, years")
    ax.set_ylabel("meters above present")
    ax.set_xticks([0, 5e5, 1e6])
    ax.legend()
    ax = axarr[0][1]
    ax.semilogx(times, seaLevelLists[index])
    ax.set_xlabel("time, years")
plt.savefig("figure_03.pdf")
plt.show()
```

## Sea Level



```

In [7]: display(Markdown("## Costs"))
fig,axarr = plt.subplots(nrows=3,ncols=2,sharex="col",sharey="row",squeeze=False,figsize=(10,10))

myString = "<table border 1px>"

myString += "<tr><td colspan=1>Direct Costs</td></tr>"

myString += "<tr><td>Carbon Release</td><td>Recovery Time Scale</td>"
myString += "<td>Climate Costs \$/tonC</td><td></td>"
myString += "<td>Total Costs \$/tonC</td></tr>"

CFeedbackFactor = climBaseParmList[1]
oceanAcidTime = climBaseParmList[2]
thermostatTime = climBaseParmList[3]
warmingTime = climBaseParmList[4]
iceMeltingTime = climBaseParmList[5]
dT2x = climBaseParmList[6]

econRecoveryTimes = [ 10000, 200000 ]

climCostFactor = 1 # direct climate costs
SLCostFactor = 0

for CReleaseGton in CReleaseGtons:

    for rIndex, econRecoveryTime in enumerate(econRecoveryTimes):

        times, deltaTs, CATmFactors, temperatures, seaLevels = \
            climateModel( CReleaseGton, CFeedbackFactor, oceanAcidTime,
thermostatTime, warmingTime, \
                iceMeltingTime, dT2x )
        climCosts, seaLevelCosts, cumClimCost, cumSeaLevelCost = \
            econModel( times, deltaTs, temperatures, seaLevels, \
                CReleaseGton, climCostFactor, SLCostFactor, econRecoveryTime )

        labelString = str(int(CReleaseGton/1000)) + "k Gton / " + str(int(econRecoveryTime/1000)) + "k years"
        ax = axarr[0][0]
        ax.plot(times,climCosts,label=labelString)
        ax.legend()
        ax.set_title("Direct Climate")
        ax.set_ylabel("Percent GDP")
        ax = axarr[0][1]
        ax.semilogx(times,climCosts)

        if(rIndex == 0):
            myString += "<tr><td>" + "{:,}".format(CReleaseGton) + " Gt.
</td><td>"
        else:
            myString += "<tr><td></td><td>"
myString += "{:,}".format(econRecoveryTime) + " yr.</td><td>\$"
myString += "{:,.1f}".format( cumClimCost/1000. )
myString += "k</td><td></td>"
myString += "<td>\$" + "{:,.1f}".format( cumClimCost/1000. )

```

```

myString += "k</td></tr>"

myString += "<tr><td colspan=1>Costs Assuming Population Feedback</td></tr>"
myString += "<tr><td>Carbon Release</td><td>Recovery Time Scale</td>"
myString += "<td>Climate Costs \$/tonC</td><td>Sea Level Costs \$/tonC</td>"
myString += "<td>Total Costs \$/tonC</td></tr>"

climCostFactor = 4 # 3 + 1 for feedback + direct
SLCostFactor = 15
for CReleaseGton in CReleaseGtons:
    for rIndex, econRecoveryTime in enumerate(econRecoveryTimes):

        times, deltaTs, CAtmFactors, temperatures, seaLevels = \
            climateModel( CReleaseGton, CFeedbackFactor, oceanAcidTime,
thermostatTime, \
                        warmingTime, iceMeltingTime, dT2x )
        climCosts, seaLevelCosts, cumClimCost, cumSeaLevelCost = \
            econModel( times, deltaTs, temperatures, seaLevels, \
                        CReleaseGton, climCostFactor, SLCostFactor, econRecoveryTime )

        labelString = str(int(CReleaseGton/1000)) + "k Gton / " + str(int(econRecoveryTime/1000)) + "k years"

        ax = axarr[1][0]
        ax.plot(times,climCosts,label=labelString)
        ax.legend()
        ax.set_title("Climate with Population Feedback")
        ax.set_ylabel("Percent GDP")
        ax = axarr[1][1]
        ax.semilogx(times,climCosts)
        ax = axarr[2][0]
        ax.plot(times,seaLevelCosts,label=labelString)
        ax.set_ylabel("Percent GDP")
        ax.set_xlabel("time, years")
        ax.set_title("Sea Level")
        ax.set_xticks([0,5e5,1e6])
        ax.legend()
        ax = axarr[2][1]
        ax.semilogx(times,seaLevelCosts)
        ax.set_xlabel("time, years")

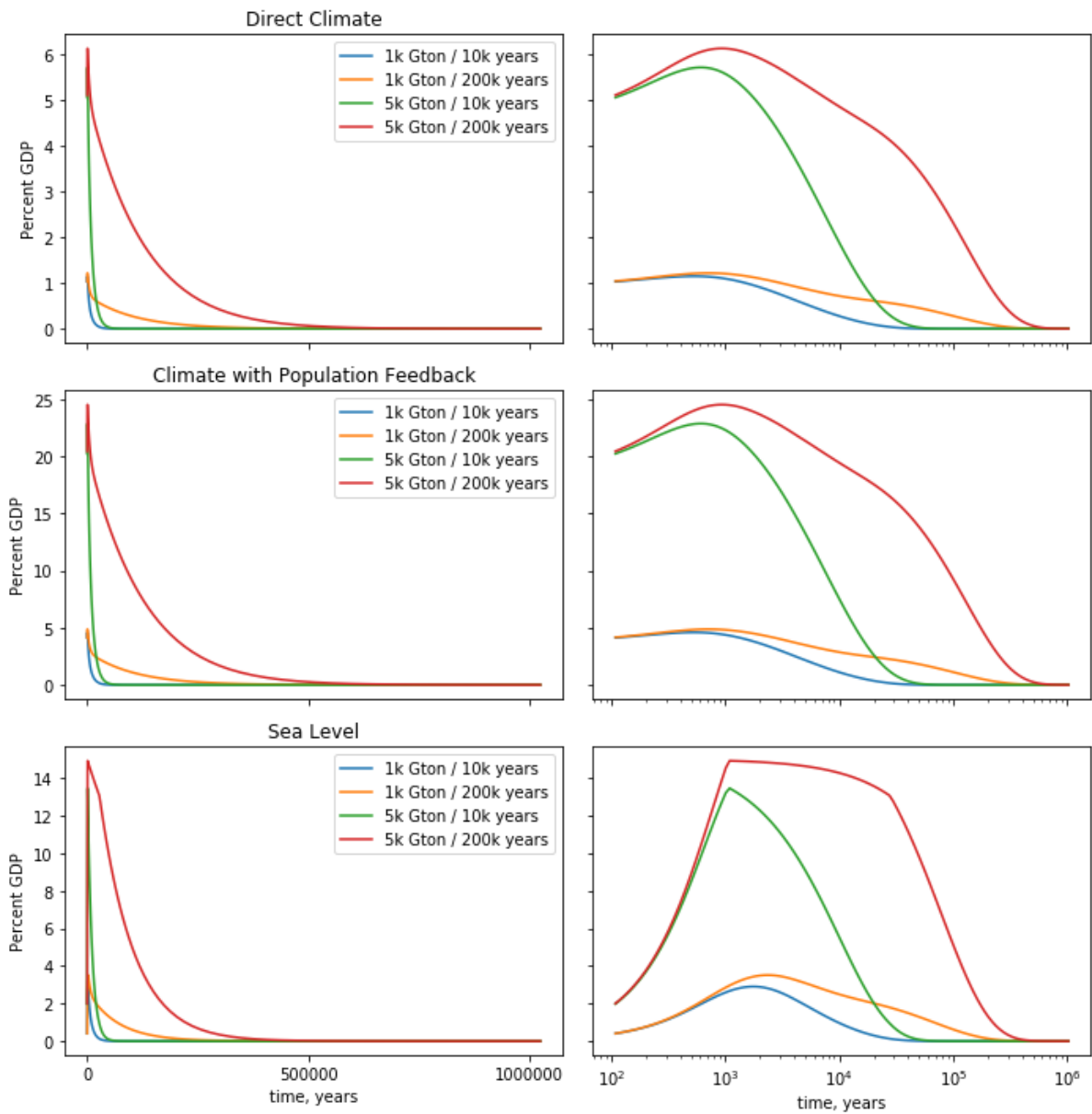
        if(rIndex == 0):
            myString += "<tr><td>" + "{:,}".format(CReleaseGton) + " Gt.
</td><td>"
        else:
            myString += "<tr><td></td><td>"
            myString += "{:,}".format(econRecoveryTime) + " yr.</td><td>\$"
            myString += "{:,.1f}".format( cumClimCost/1000. )
            myString += "k</td><td>\$" + "{:,.1f}".format( cumSeaLevelCost/1000. ) + "k</td>"
            myString += "<td>\$" + "{:,.1f}".format( ( cumClimCost + cumSeaLevelCost )/1000. )
            myString += "k</td></tr>"

```

```
plt.tight_layout()
plt.savefig("figure_06.pdf")
plt.show()

display(Markdown(myString))
```

Costs



Direct Costs	Carbon Release	Recovery Time	Scale	Climate Costs \$/tonC	Total Costs \$/tonC
1,000 Gt.	10,000 yr.	\$8.3k	\$8.3k	200,000 yr.	\$75.2k
5,000 Gt.	10,000 yr.	\$10.2k	\$10.2k	200,000 yr.	\$121.2k
1,000 Gt.	10,000 yr.	\$33.1k	\$25.8k	\$59.0k	\$301.0k
5,000 Gt.	10,000 yr.	\$40.8k	\$27.5k	\$68.3k	\$484.9k

```

In [8]: display(Markdown("## Sensitivity to the Economic Recovery Rate"))
fig,axarr = plt.subplots(nrows=1,ncols=2,sharex=True,sharey=True,squeeze=False)

CFeedbackFactor = climBaseParmList[1]
oceanAcidTime = climBaseParmList[2]
thermostatTime = climBaseParmList[3]
warmingTime = climBaseParmList[4]
iceMeltingTime = climBaseParmList[5]
dT2x = climBaseParmList[6]

econRecoveryRange = []
econRecoveryPlot = []
for i in range(0,10):
    econRecoveryRange.append( float((i+1)*10000) )
    econRecoveryPlot.append( float((i+1)*10 ) )

for CReleaseGton in CReleaseGtons:
    cumClimCostList = []
    cumSeaLevelCostList = []

    for rIndex, econRecoveryTime in enumerate(econRecoveryRange):
        # print(EconRecoveryTime)
        times, deltaTs, CAtmFactors, temperatures, seaLevels = \
            climateModel( CReleaseGton, CFeedbackFactor, oceanAcidTime,
thermostatTime, \
                        warmingTime, iceMeltingTime, dT2x )
        climCosts, seaLevelCosts, cumClimCost, cumSeaLevelCost = \
            econModel( times, deltaTs, temperatures, seaLevels, \
                        CReleaseGton, climCostFactor, SLCostFactor, econRecoveryTime )
        cumClimCostList.append( cumClimCost/1000. )
        cumSeaLevelCostList.append( cumSeaLevelCost/1000. )

    labelString = str(int(CReleaseGton/1000)) + "k Gton"

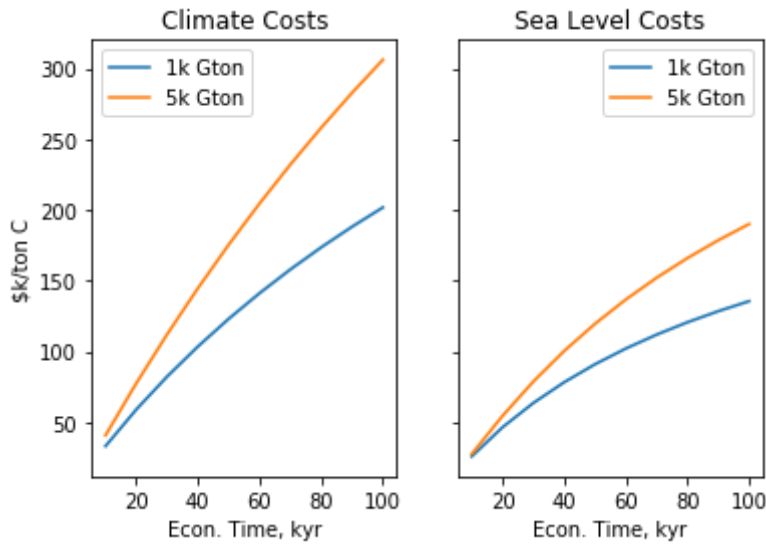
    ax = axarr[0][0]
    ax.plot(econRecoveryPlot,cumClimCostList,label=labelString)
    ax.legend()
    ax.set_title("Climate Costs")
    ax.set_ylabel("\$k/ton C")
    ax.set_xlabel("Econ. Time, kyr")

    ax = axarr[0][1]
    ax.plot(econRecoveryPlot,cumSeaLevelCostList,label=labelString)
    ax.legend()
    ax.set_title("Sea Level Costs")
    ax.set_xlabel("Econ. Time, kyr")

plt.savefig("figure_s02.pdf")
plt.show()

```

## Sensitivity to the Economic Recovery Rate



## Monte Carlo Uncertainty Propagation

In which uncertainty is applied to the input parameters, to derive the sensitivity of model output.

The code in the following module begins by running the base case. Then for as many iterations as specified in the variables numIters, the code applies variation to each model parameter in turn, drawing from a log-uniform distribution between the low and high values specified in list parmRanges, keeping the other model parameters at their base values. Finally the code varies all parameters simultaneously and independently. The results are tabulated and plotted.

The results show that of the geophysical parameters, the climate sensitivity has the strongest impact on the uncertainty in the cost values.

```

In [4]: import random

numIters = 10000

def modelWrapper( parmList ):

    CReleaseGton = parmList[0]
    CFeedbackFactor = parmList[1]
    oceanAcidTime = parmList[2]
    thermostatTime = parmList[3]
    warmingTime = parmList[4]
    iceMeltingTime = parmList[5]
    dT2x = parmList[6]
    climCostFactor = parmList[7]
    SLCostFactor = parmList[8]
    econRecoveryTime = parmList[9]

    times, deltaTs, CAtmFactors, temperatures, seaLevels = \
        climateModel( CReleaseGton, CFeedbackFactor, oceanAcidTime, ther
mostatTime, \
                        warmingTime, iceMeltingTime, dT2x )

    climCosts, seaLevelCosts, cumClimCost, cumSeaLevelCost = \
        econModel( times, deltaTs, temperatures, seaLevels, CReleaseGton
, \
                    climCostFactor, SLCostFactor, econRecoveryTime )

    return cumClimCost, cumSeaLevelCost

cumClimCost, cumSeaLevelCost = modelWrapper( climBaseParmList )
baseCosts = [cumClimCost, cumSeaLevelCost]

costValues = [] # structure cost[iCost][iParm][iter] where iCost is cli
mate or sea level
for costType in range(0,3):
    costValues.append([])
    for index in range(0,numParms+2): # 2 additional for the var of all
geophysics params, all
        costValues[costType].append([])

for iter in range(0,numIters):

    # first tweak each parameter individually, keeping others constant
    for iParm in range(0,numParms): # loop over parameters that need to
be tweaked
        parmList = climBaseParmList.copy()
        s = parmLogRanges[iParm][0] + random.random() * ( parmLogRanges[
iParm][1] - parmLogRanges[iParm][0] )
        parmList[iParm] = math.exp(s) # resetting only one
        cumClimCost, cumSeaLevelCost = modelWrapper( parmList )
        costValues[0][iParm].append( cumClimCost )
        costValues[1][iParm].append( cumSeaLevelCost )
        costValues[2][iParm].append( cumClimCost + cumSeaLevelCost )

    # tweak just the geophysical parameters
    parmList = climBaseParmList.copy()

```

```

    for iParm in range(0,numParmsGeo):
        s = parmLogRanges[iParm][0] + random.random() * ( parmLogRanges[
iParm][1] - parmLogRanges[iParm][0] )
        parmList[iParm] = math.exp(s)
        cumClimCost, cumSeaLevelCost = modelWrapper( parmList )
        costValues[0][numParms].append( cumClimCost )
        costValues[1][numParms].append( cumSeaLevelCost )
        costValues[2][numParms].append( cumClimCost + cumSeaLevelCost )

    # now tweak them all together
    parmList = climBaseParmList.copy()
    for iParm in range(0,numParms):
        s = parmLogRanges[iParm][0] + random.random() * ( parmLogRanges[
iParm][1] - parmLogRanges[iParm][0] )
        parmList[iParm] = math.exp(s)
        cumClimCost, cumSeaLevelCost = modelWrapper( parmList )
        costValues[0][numParms+1].append( cumClimCost )
        costValues[1][numParms+1].append( cumSeaLevelCost )
        costValues[2][numParms+1].append( cumClimCost + cumSeaLevelCost )

means = [[],[],[ ]]
stds = [[],[],[ ]]
relStd = [[],[],[ ]]
maxCost = 0.
for iParm in range(0,numParms+2):
    for iCost in range(0,3): # climate, sea level, total
        mean = np.mean( costValues[iCost][iParm] )
        median = np.median( costValues[iCost][iParm] )
        std = np.std( costValues[iCost][iParm], ddof=1 )
        means[iCost].append( median )
        stds[iCost].append( std )
        relStd[iCost].append( std / mean )
        max = np.amax( costValues[iCost][iParm] )
        if max > maxCost:
            maxCost = max

#csv output
import csv
with open('monte_carlo.csv', mode='w') as monte_file:
    monte_writer = csv.writer(monte_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

    monte_writer.writerow(['','Low','High','Base','Median Climate Cost
\\$k/ton','sigma','pct', \
                        'Median Sea Level Cost $k/ton','sigma','pct',
'Median Total Cost','sigma','pct'])
    monte_writer.writerow(['Base','','','baseCosts[0]','','',baseCosts[
1],','','baseCosts[0]+baseCosts[1],','',''])

# individual parameter variations
for iParm in range(0,numParms+2):
    rowList = []
    rowList.append(parmNames[iParm])
    if(iParm < numParms):
        rowList.append( parmRanges[iParm][0] )
        rowList.append( parmRanges[iParm][1] )
        rowList.append( climBaseParmList[iParm] )

```



```

else:
    rowList.append("")
    rowList.append("")
    rowList.append("")
for iCost in range(0,3):
    rowList.append( means[iCost][iParm] )
    rowList.append( stds[iCost][iParm] )
    rowList.append( relStds[iCost][iParm] )
monte_writer.writerow(rowList)

# table header
myString = "<table border 1px><tr><td>n=" + str(numIters)
myString += "</td><td colspan=3>Parameter Range</td><td colspan=2>Climate Cost, k$/tonC</td><td></td>"
myString += "<td colspan=2>Sea Level Cost, k$/tonC</td><td></td>"
myString += "<td colspan=2>Total Cost, k$/tonC</td><td></td></tr>"

# subheader
myString += "<tr><td></td><td>low</td><td>high</td><td>base</td>"
myString += "<td>median</td><td>sigma</td><td>%</td>"
myString += "<td>median</td><td>sigma</td><td>%</td>"
myString += "<td>median</td><td>sigma</td><td>%</td></tr>"

# base results
myString += "<tr><td>Base</td><td></td><td></td><td></td>"
myString += "<td>" + "{:,.1f}".format(baseCosts[0]/1000.) + "</td><td></td><td></td>"
myString += "<td>" + "{:,.1f}".format(baseCosts[1]/1000.) + "</td><td></td><td></td>"
myString += "<td>" + "{:,.1f}".format((baseCosts[0]+baseCosts[1])/1000.) + "</td><td></td><td></td></tr>"

# individual parameter variations
for iParm in range(0,numParms+2):
    myString += "<tr><td>" + parmNames[iParm] + "</td>"
    if(iParm < numParms):
        myString += "<td>" + "{:,.1f}".format(parmRanges[iParm][0]) + "</td>"
        myString += "<td>" + "{:,.1f}".format(parmRanges[iParm][1]) + "</td>"
        myString += "<td>" + "{:,.1f}".format(climBaseParmList[iParm]) + "</td>"
    else:
        myString += "<td></td><td></td><td></td>"
        for iCost in range(0,2):
            myString += "<td>" + "{:,.1f}".format(means[iCost][iParm]/1000.) + "</td>"
            myString += "<td>" + "{:,.1f}".format(stds[iCost][iParm]/1000.) + "</td>"
            myString += "<td>" + "{:,.0f}".format(relStds[iCost][iParm]*100.) + "</td>"
        myString += "<td>" + "{:,.1f}".format(means[2][iParm]/1000.) + "</td>"
        myString += "<td>" + "{:,.1f}".format(stds[2][iParm]/1000.) + "</td>"
        myString += "<td>" + "{:,.0f}".format(relStds[2][iParm]*100.) + "</td>"

```

```
d>"
    myString += "</tr>"

myString += "</table>"
display(Markdown(myString))
```

<table border 1px>n=10000Parameter RangeClimate Cost, k\$/tonCSea Level Cost,  
k\$/tonCTotal Cost,  
k\$/tonClowhighbasemediansigma%mediansigma%mediansigma%Base70.927.398.2CReleaseG

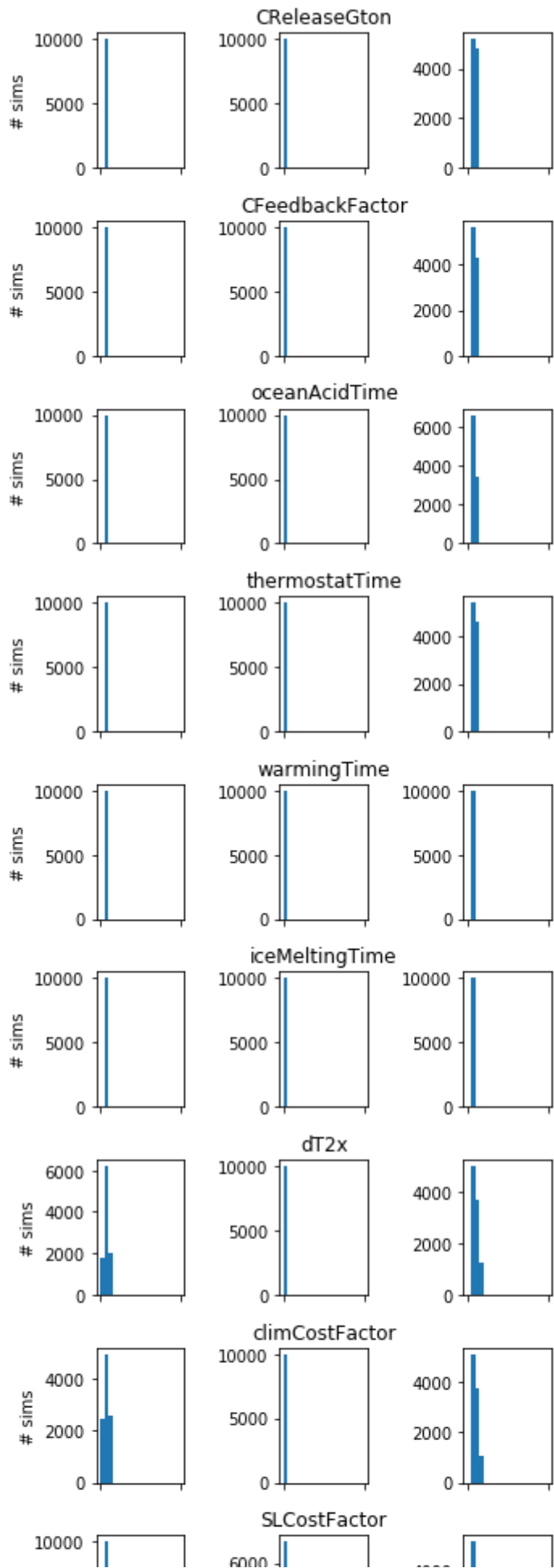
```

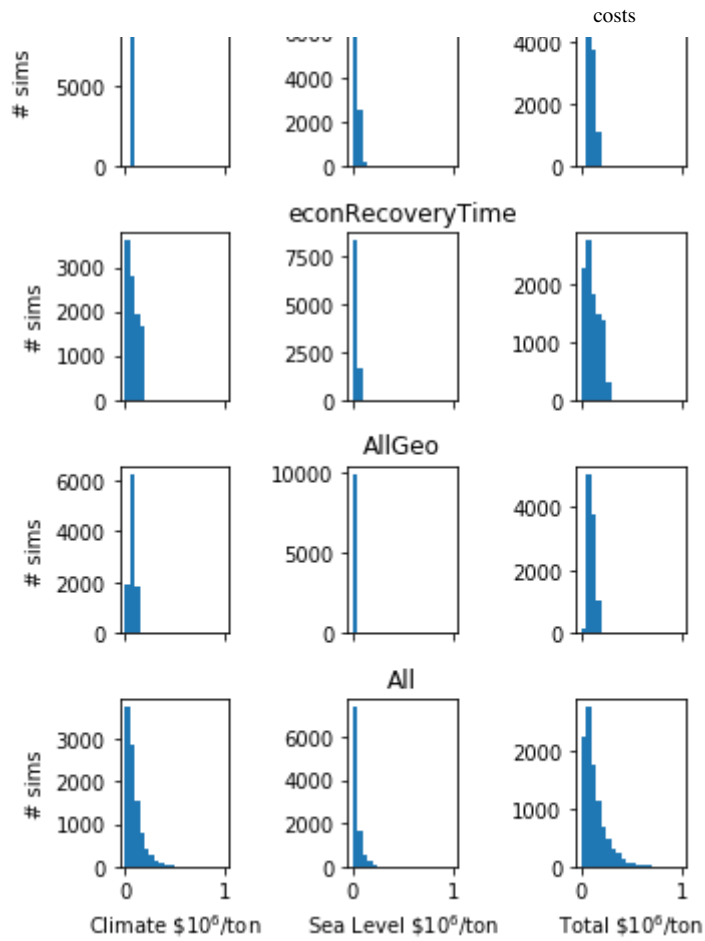
In [5]: fig,axarr = plt.subplots(nrows=numParms+2,ncols=3,sharex=True,sharey=False,
se,squeeze=False,figsize=(5,20))

for iParm in range(0,numParms+2):
    for iCost in range(0,2): # climate, sea level
        ax = axarr[iParm][iCost]
        # ax.set_xticks([0,1,2,3,4])
        costs = []
        for i in range(0,len(costValues[iCost][iParm])):
            costs.append(costValues[iCost][iParm][i]/1e6)
        ax.hist(costs,range=(0,maxCost/1e6),bins=20)
    ax = axarr[iParm][2] # total
    costs = []
    for i in range(0,len(costValues[iCost][iParm])):
        costs.append((costValues[0][iParm][i]+costValues[1][iParm][i])/1
e6)
    ax.hist(costs,range=(0,maxCost/1e6),bins=20)

for iParm in range(0,numParms+2):
    ax = axarr[iParm][0]
    ax.set_ylabel("# sims")
    ax = axarr[iParm][1]
    ax.set_title(parmNames[iParm])
ax = axarr[numParms+1][0]
ax.set_xlabel('Climate \ $10$^6$/ton')
ax = axarr[numParms+1][1]
ax.set_xlabel('Sea Level \ $10$^6$/ton')
ax = axarr[numParms+1][2]
ax.set_xlabel('Total \ $10$^6$/ton')
plt.tight_layout()
plt.savefig("figure_s01.pdf")
plt.show()

```





## Marginal Cost Calculation

```

In [10]: CReleaseStep = 100
CReleaseSeries = list(range(1000,5000,CReleaseStep))

CFeedbackFactor = climBaseParmList[1]
oceanAcidTime = climBaseParmList[2]
thermostatTime = climBaseParmList[3]
warmingTime = climBaseParmList[4]
iceMeltingTime = climBaseParmList[5]
dT2x = climBaseParmList[6]

climCostFactor = 4      # economy percent hit per degree C
SLCostFactor = 15      # percent hit for full sea level rise
econRecoveryTime = 200000

marginalCosts = []
for i in range(0,3):
    marginalCosts.append([])

for index, CReleaseGton in enumerate(CReleaseSeries):

    times, deltaTs, CAtmFactors, temperatures, seaLevels = \
        climateModel( CReleaseGton, CFeedbackFactor, oceanAcidTime, thermostatTime, warmingTime, \
            iceMeltingTime, dT2x )
    climCosts, seaLevelCosts, cumClimCost, cumSeaLevelCost = \
        econModel( times, deltaTs, temperatures, seaLevels, \
            CReleaseGton, climCostFactor, SLCostFactor, econRecoveryTime
        )

    costs = [ cumClimCost, cumSeaLevelCost, cumClimCost + cumSeaLevelCost ] # dollars / ton

    for i in range(0,3):
        costs[i] *= CReleaseGton * 1e9      # back to dollars

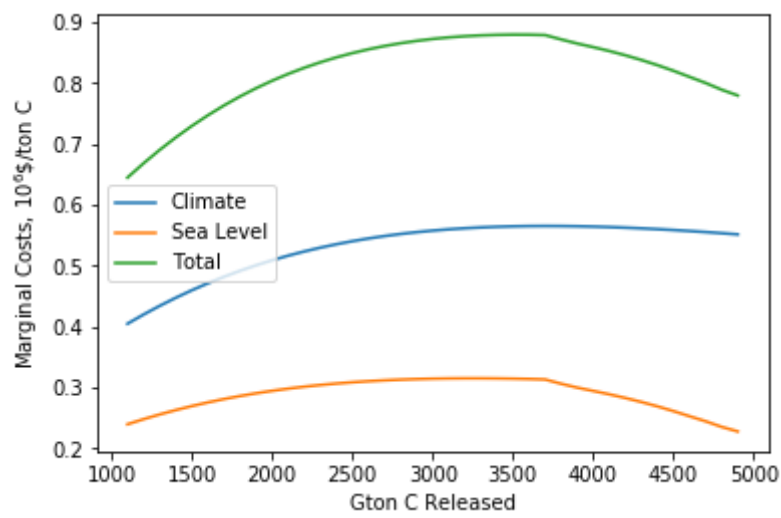
    if index > 0:
        for index, cost in enumerate(costs):
            marginalCost = ( cost - oldCosts[index] ) / ( CReleaseStep *
1.e15 ) # million dollars per ton
            marginalCosts[index].append( marginalCost )

    oldCosts = costs.copy()

CReleaseSeries.remove(1000)
labels = [ "Climate", "Sea Level", "Total" ]
for i in range(0,3):

    plt.plot(CReleaseSeries,marginalCosts[i],label=labels[i])
plt.legend()
plt.xlabel("Gton C Released")
plt.ylabel("Marginal Costs, 10$^6$/ton C")
plt.savefig("figure_07.pdf")
plt.show()

```

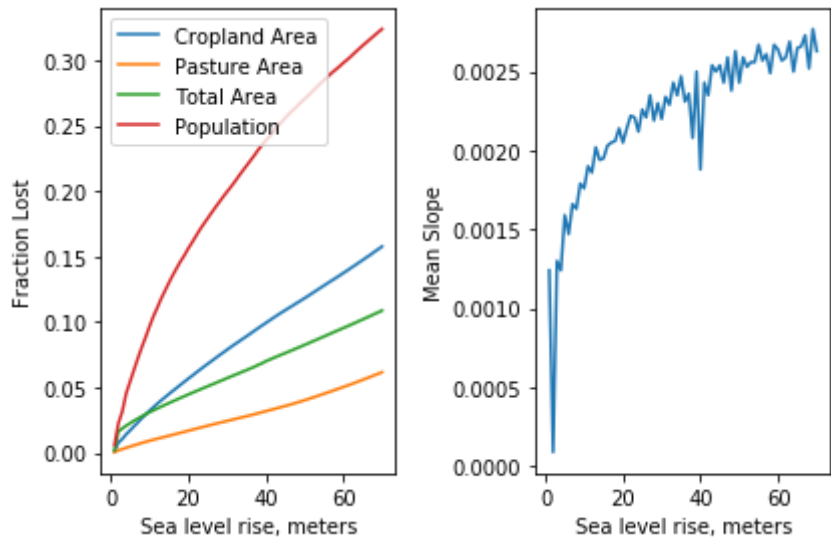


```

In [11]: dataSet = "assets_by_sealevel.csv"
altitudes, slopes = [],[]
totLists, cumTotLists = [ [], [], [], [] ],      [ [], [], [], [] ]
cumTots = [ 0, 0, 0, 0 ]
with open(dataSet,"r") as ins:
    for lineIndex, line in enumerate(ins):
        values = line.split(",")
        if lineIndex == 0:
            headers = values[1:6]
        elif lineIndex < 71:
            values = line.split(",")
            altitudes.append( float(values[0]) )
            slopes.append( float(values[5]) )
            for i in range(0,4):
                value = float(values[i+1])
                totLists[i].append( value )
                cumTots[i] += value
                cumTotLists[i].append( cumTots[i] )
        else:
            values = line.split(",")
            for i in range(0,4):
                value = float(values[i+1])
                cumTots[i] += value
    for iLine in range(0,70):
        for iVal in range(0,4):
            cumTotLists[iVal][iLine] /= cumTots[iVal]
fig,axarr = plt.subplots(nrows=1,ncols=2,sharex=True,sharey=False,squeeze=False)
ax = axarr[0][0]
for i in range(0,4):
    header = headers[i].replace(" (km2)","")
    ax.plot(altitudes,cumTotLists[i],label=header)
ax.set_xlabel("Sea level rise, meters")
ax.set_ylabel("Fraction Lost")
ax.legend()
ax = axarr[0][1]
ax.plot(altitudes,slopes)
ax.set_xlabel("Sea level rise, meters")
ax.set_ylabel("Mean Slope")
plt.tight_layout()
plt.savefig("figure_04.pdf")
plt.show()

```





```

In [12]: dataSet = "assets_by_latitude.csv"
latitudes = []
stuffByLats = [ [], [], [], [], [] ]
cumStuffs = [ 0, 0, 0, 0, 0 ]
with open(dataSet, "r") as ins:
    for lineIndex, line in enumerate(ins):
        values = line.split(",")
        if lineIndex == 0:
            headers = values[1:5]
        else:
            values = line.split(",")
            for i in range(0,5):
                value = float(values[i])
                stuffByLats[i].append( value )
                cumStuffs[i] += value
for iStuff in range(0,4):
    for iLat in range(0,9):
        stuffByLats[iStuff+1][iLat] /= cumStuffs[iStuff+1]
    header = headers[iStuff].replace(" (km2)", "")
    plt.plot(stuffByLats[0], stuffByLats[iStuff+1], label=header)
plt.legend()
plt.xlabel("Latitude")
plt.ylabel("Fraction per 10° bin")
plt.savefig("figure_05.pdf")
plt.show()

```

