Steven Conflenti

ASTR 3750

16 November 2015

Cratering Saturation Equilibrium Simulation

I took an object-oriented approach in this project’s implementation. My program consists of two Python classes I wrote, Metorite and PlanetarySurface. Each Eeteorite object has fields in which its impact time, impact location, and destruction time are stored. The majority of the data processing is done within the second class, PlanetarySurface. Here, I wrote helper functions to determine things like where on the surface an impact took place, or whether or not any old craters were in the destruction range of a new impact. Each year has a 1/1000 chance of having an impact occur. If there is an impact in a given year, I create a new Meteorite object and store it sequentially in an array. After each impact happens I check the surface to see if saturation equilibrium has been achieved. Once the surface reaches equilibrium I am able to simply iterate through the array containing all of the Meteorite objects to collect and process the data. Finally, my program formats the data and creates a plot showing the total number of intact craters and the number of total impacts versus time until saturation equilibrium was reached.

Figure 1: Example Output 1

Figure 2: Example Output 2

Figure 3: Example Output 3

Figure 4: Source Code – Meteorite.py

class Meteorite(object):

def \_\_init\_\_(self, time, location):

self.created = time

self.location = location

self.destroyed = -1

def getCreated(self):

return self.created

def getLocation(self):

return self.location

def getDestroyed(self):

return self.destroyed

def setDestroyed(self, time):

self.destroyed = time

Figure 5: Source Code – PlanetarySurface.py

from \_\_future\_\_ import division

import sys

import math

import random

import matplotlib.pyplot as plt

from Meteorite import Meteorite

class PlanetarySurface(object):

def \_\_init\_\_(self):

self.impacts = []

#returns True if there is an impact in a given year

def checkIfImpact(self):

#generates a random integer from 0 - 999

randomInt = random.randrange(1000)

#1/1000 chance of having an impact

if(randomInt == 0):

return True

return False

def getImpactLocation(self, time):

#generates random x and y coordinates for each impact

#granularity down to 1 m

x = random.randrange(500000)

y = random.randrange(500000)

meteorite = Meteorite(time, (x,y))

self.impacts.append(meteorite)

#use pythagorean theorem to check if the center of the old impact is within the destruction range of the new impact

def checkInRange(self, oldImpact, newImpact):

#the destruction radius of each impact

#each meteor is 50km + 20% for ejecta blanket = 60m => r = 30km = 30000 m

destructionRadius = 30000

#c^2 used in pythag calculation

c2 = (destructionRadius \*\* 2)

#use distance formula to calculate a and b with impact coordinates

a = abs(oldImpact[0] - newImpact[0])

b = abs(oldImpact[1] - newImpact[1])

#calculate (a^2)(b^2)

a2b2 = (a \*\* 2) + (b \*\* 2)

#if the crater center is in destruction range

if(a2b2 < c2):

return True

else:

return False

def checkCraterDestruction(self):

numCraters = len(self.impacts)

if(numCraters <= 1):

#if # of impacts is 0 or 1, no obliterations

return

newImpact = self.impacts[numCraters - 1]

#iterates through all of the old craters to check if the new one obliterated any

for i in range(0, numCraters - 2):

oldImpact = self.impacts[i]

#destroyed time is initialized to -1 and only updated when destroyed

#if this old impact hasn't been destroyed yet

if(oldImpact.getDestroyed() < 0):

#and if the center is in the destruction range of the new impact

if(self.checkInRange(oldImpact.getLocation(), newImpact.getLocation())):

#update the old crater's destruction time

oldImpact.setDestroyed(newImpact.getCreated())

#gets the number of intact craters at a given time

def getNumCraters(self, time):

craterCount = 0

for meteorite in self.impacts:

#if the meteorite was created after the time we are counting up to, don't add to count

if meteorite.getCreated() > time:

break

#if the metorite hasn't been destroyed yet or it's destroyed after the time we are checking at

if meteorite.getDestroyed() < 0 or meteorite.getDestroyed() > time:

#increase the count

craterCount += 1

return craterCount

#gets the total number of impacts that have occured up to a given time

def getNumImpacts(self, time):

impactCount = 0

for meteorite in self.impacts:

#if the meteorite was created after the time we are counting up to, don't add to count

if meteorite.getCreated() > time:

break

impactCount += 1

return impactCount

def checkSaturation(self):

length = len(self.impacts) - 1

#the surface can't be saturated unless it has a sufficient amount of impacts

if(length < 25):

return False

#we want to check for saturation over a period of when the time is doubled

#not quite

time1 = self.impacts[length].getCreated()

time2 = time1 / 2

#count the number of craters at each time

craterCount1 = self.getNumCraters(time1)

craterCount2 = self.getNumCraters(time2)

#calculate the percent change in the number of craters

delta = (abs(float(craterCount2 - craterCount1)))/craterCount2

#if the percent change is more than 5%, the surface isn't saturated yet

if(delta > 0.05):

return False

else:

return True

if \_\_name\_\_ == "\_\_main\_\_":

print 'The simulation is starting...'

#create a PlanetarySurface object

surface = PlanetarySurface()

time = 0

while(1):

#if there's an impact this year

if(surface.checkIfImpact()):

#print('IMPACT year: {0}!'.format(time))

#keep track of x,y coordinates and time of impact

surface.getImpactLocation(time)

#update newly destroyed craters

surface.checkCraterDestruction()

#stop if saturated

if(surface.checkSaturation()):

break

else:

time += 1

else:

#print('No impact in year: {0}'.format(time))

time += 1

#saturation is defined as less than a 5% change in the number of craters when time is doubled

#this means that the time when the surface first became saturated is half of the simulation time

saturationTime = int(math.ceil(time / 2))

totalImpacts = 0

creationTimes = []

numNotDestroyed = []

numImpacts = []

for meteorite in surface.impacts:

#only want data from meteorites that impacted before/at saturation

if(meteorite.getCreated() <= saturationTime):

totalImpacts += 1

creationTimes.append(meteorite.getCreated())

numNotDestroyed.append(surface.getNumCraters(creationTimes[len(creationTimes) - 1]))

numImpacts.append(surface.getNumImpacts(creationTimes[len(creationTimes) - 1]))

#calculate crater densities

intactDensity = (numNotDestroyed[len(numNotDestroyed) - 1] / 500)

totalDensity = (totalImpacts / 500)

#set up output to print and put on graph

output1 = ('The time to saturation was {0} years.'.format(saturationTime))

output2 = ('At this time, {0} undestroyed craters exist.'.format(numNotDestroyed[len(numNotDestroyed) - 1]))

output3 = ('There were {0} total impacts.'.format(totalImpacts))

output4 = ('The density of intact craters was {0} impacts/km.'.format(intactDensity))

output5 = ('The density of all craters was {0} impacts/km.'.format(totalDensity))

print('Simulation complete.')

print(output1)

print(output2)

print(output3)

print(output4)

print(output5)

#plot data

totalLine = plt.plot(creationTimes, numImpacts, label = 'Total Number of Impacts', linewidth = 2)

intactLine = plt.plot(creationTimes, numNotDestroyed, label = 'Number of Intact Craters', linewidth = 2)

#create figure and axes titles

plt.title('Planetary Surface Cratering Simulation')

plt.xlabel('Time Elapsed (years)')

plt.ylabel('Quantity')

#create the legend

plt.legend(loc = 4)

#put output data on plot

plt.figtext(.138, .86, output1)

plt.figtext(.138, .83, output2)

plt.figtext(.138, .80, output3)

plt.figtext(.138, .77, output4)

plt.figtext(.138, .74, output5)

#display figure

plt.show()