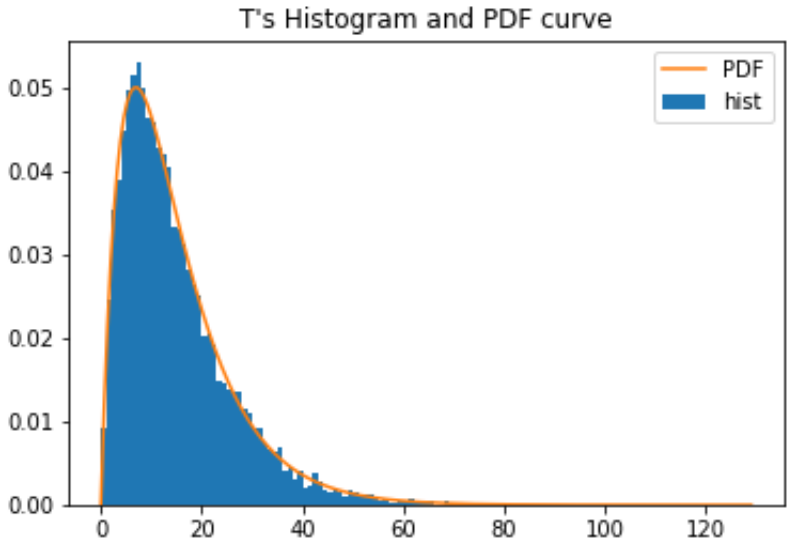
Mini Project 1 Report

Jiadao Zou
jxz172230

Q1

- **(a) Use the above density function to analytically compute the probability that the lifetime of the satellite exceeds 15 years.**
  we could directly calculate what we want, here I use computer to calculate the result for me:
  $P(T > 15) = \int_{15}^{+\infty} f_T(t)dt = \int_{15}^{+\infty}(0.2 * (e^{-0.1t} - e^{-0.2t}))dt = 0.396473251928996$
- **(b) Use the following steps to take a Monte Carlo approach to compute E(T) and P(T >15).**
  - **Simulate one draw of the block lifetimes XA and XB. Use these draws to simulate one draw of the satellite lifetime T.**
    From the definition of Exponential distribution:
    $\lambda e^{-\lambda x}$ has Mean of $\frac{1}{\lambda}$.
    we build our sample from it:
    $X_A = 25.484996076390996$
    $X_B = 4.338367154791759$
    $P(T) = \max(X_A, X_B) = 25.484996076390996$

    In the following, I would use Integer instead of float to make it easier to express the time.
  - **Repeat the previous step 10,000 times. This will give you 10,000 draws from the distribution of T. Try to avoid 'for' loop. Use 'replicate' function instead. Save these draws for reuse in later steps.**
  - **Make a histogram of the draws of T using 'hist' function. Superimpose the density function given above. Try using 'curve' function for drawing the density. Note what you see.**



sum(hist) = 1

- **Use the saved draws to estimate E (T). Compare your answer with the exact answer given above.**
  Expected Value: 14.4343
- **Use the saved draws to estimate the probability that the satellite lasts more than 15 years. Compare with the exact answer computed in part (a).**
  $P(T > 15) = 0.3594$
  Now comparing with the above answer 0.3964, due to here I use integer to computing and in the first question I used float type, in addition of the iteration times, the error is fair.
- **Repeat the above process of obtaining an estimate of E(T) and an estimate of the probability four more times. Note what you see.**

	Expected Value	P(T>15)
1st	14.5759	0.3691
2nd	14.5622	0.3687
3rd	14.4304	0.3556
4th	14.5563	0.3635

By repeating the whole progress 4 times, we could see 10000 iteration time is stable around 14.5%, that make me realized we could use Monte Carlo to avoid complex computing if we could know or preset a distribution for the event we are interested and by simple sampling and calculating with enough data, we could get a barely close result to the truth.

- **(c) Repeat part (vi) five times using 1,000 and 100,000 Monte Carlo replications**

instead of 10,000. Make a table of results. Comment on what you see and provide an explanation.

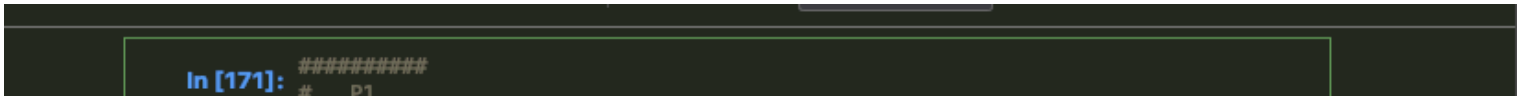
	Expected Values (All)	Expected Value (Average)	P(T>15)(All)	P(T>15) (Average)
1,000 iterations	14.728, 14.482, 14.663, 14.601, 13.98	14.4908	0.376, 0.357, 0.373, 0.369, 0.339	0.3628
10,000 iterations	14.5759, 14.5622, 14.4304, 14.5563, 14.4343	14.51182	0.3691, 0.3687, 0.3556, 0.3635, 0.3594	0.36326
100,000 iterations	14.51896, 14.471, 14.49154, 14.52974, 14.47991	14.49823	0.36454, 0.36038, 0.36322, 0.36487, 0.36251	0.363104

As we could see, even 1,000 iterations is enough to achieve the result of 100,000 iterations, which means the bottleneck is not the size of sample but the way how we generate data (by which random distribution) and the data type we use during the computation. Also, from the view of computer science, we could always find some way to optimize memory usage and increasing the speed by considering about the physical location (use sequentially stored data in cash at one time instead of umping over blocks to read the data in memory or in disk) of the data beyond the algorithms.

Q2

- Use a Monte Carlo approach estimate the value of π based on 10,000 replications.
  - idea:
    - The size of circle: πR^2
    - The size of square: $4R^2$
    - Replication:
      - Use Uniform Distribution to random choose a node inside the square.
        – Finally, the result would approach to $\frac{\pi}{4}$
  - My Pi: 3.1264
    My iteration: 10,000
    Error rate: 0.4836%

Code



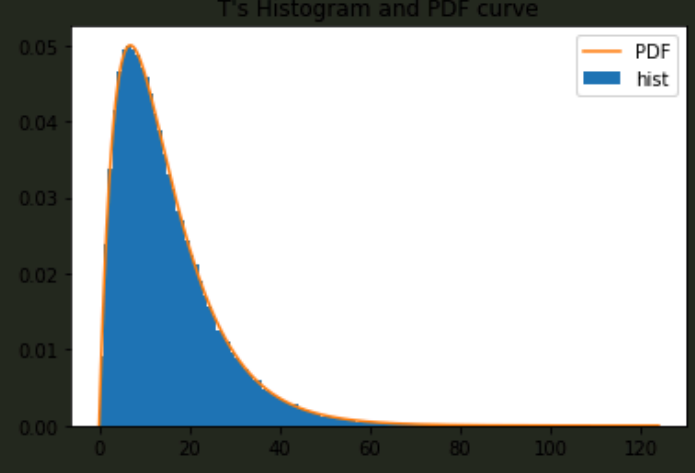
```
##########
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

from PIL import Image
import pylab
from scipy.optimize import curve_fit
```

In [172]:
```
# P1.1
t = Symbol('t')
f = 0.2*(exp(-0.1*t)-exp(-0.2*t))
print(integrate(f,(t,15,oo)))
```

```
0.396473251928996
```

In [173]:
```
# p1.2.2
def MC_T_draw(iter_time):
    t = np.array([np.maximum(np.random.exponential(10), np.random.exponential(1
    t = t.astype(int)
    # draw hist
    bin = np.arange(np.floor(t.min()),np.ceil(t.max()))
    bin_density, bin_boarders, _ = plt.hist(t, bins=bin, density=True, label='h
    # fit
    xdata = np.linspace(0,bin_boarders[-1],iter_time/50)
    def T_density(t):
        return 0.2*(np.exp(-0.1*t)-np.exp(-0.2*t))
    ydata = T_density(xdata)
    # popt, _ = curve_fit(T_density, xdata, ydata)
    plt.plot(xdata,ydata, label='PDF')
    # plt.plot(xdata, T_density(x_interval_for_fit, *popt), label='fit')
    plt.title("T's Histogram and PDF curve")
    plt.legend()
    # plt.axis('on')
    # frame = plt.gca()
    # # y axis visible
    # frame.axes.get_yaxis().set_visible(True)
    # # x axis visible
    # frame.axes.get_xaxis().set_visible(True)
    plt.savefig('/Users/jiadao/Google Drive/19Spring/6313STAT/Pj1/1-3-100000.pn
    plt.show()
    return t, bin_density, bin_boarders, iter_time
# run the function
t, density, boarders, iter_time = MC_T_draw(100000)
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9: DeprecationWarning: object
of type <class 'float'> cannot be safely interpreted as an integer.
  if __name__ == '__main__':
```



In [175]:
```
# calculate E(T)
np.mean(t)
```

Out[176]: 14.47991

In [176]:
```
# probability of T >= 15
tmp = 0
for tt in t:
    if tt > 15:
        tmp += 1
tmp/iter_time
```

Out[176]: 0.36251

In [4]:
```
###################
### P2
###################
import random
import math
def mypi(iter):
    n=iter
    # number of falling inside the circle
    total=0
    for i in range(n):
        x=random.random()
        y=random.random()
        if math.sqrt(x**2+y**2)<1.0:
            total+=1
    # get the pi
    ppi=4.0*total/n
    print(ppi)
    print('Error rate: ',abs(math.pi-ppi)/math.pi)

mypi(10000)
```

```
3.1264
Error rate:  0.004835971834996855
```

In [ ]:

```
##########
#   P1
##########
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

from PIL import Image
import pylab
```

```
from scipy.optimize import curve_fit

# P1.1
t = Symbol('t')
f = 0.2*(exp(-0.1*t)-exp(-0.2*t))
print(integrate(f,(t,15,oo)))

# p1.2.2
def MC_T_draw(iter_time):
    t = np.array([np.maximum(np.random.exponential(10), np.random.exponential(10
)) for x in range(iter_time)])
    t = t.astype(int)
    # draw hist
    bin = np.arange(np.floor(t.min()),np.ceil(t.max()))
    bin_density, bin_boarders, _ = plt.hist(t, bins=bin, density=True, label='hi
st')
    # fit
    xdata = np.linspace(0,bin_boarders[-1],iter_time/50)
    def T_density(t):
        return 0.2*(np.exp(-0.1*t)-np.exp(-0.2*t))
    ydata = T_density(xdata)
    # popt, _ = curve_fit(T_density, xdata, ydata)
    plt.plot(xdata,ydata, label='PDF')
    # plt.plot(xdata, T_density(x_interval_for_fit, *popt), label='fit')
    plt.title("T's Histogram and PDF curve")
    plt.legend()
    # plt.axis('on')
    # frame = plt.gca()
    # # y axis visible
    # frame.axes.get_yaxis().set_visible(True)
    # # x axis visible
    # frame.axes.get_xaxis().set_visible(True)
    plt.savefig('/Users/jiadao/Google Drive/19Spring/6313STAT/Pj1/1-3-100000.png
')
    plt.show()
    return t, bin_density, bin_boarders, iter_time
# run the function
t, density, boarders, iter_time = MC_T_draw(100000)

# calculate E(T)
np.mean(t)

# probability of T >= 15
tmp = 0
for tt in t:
    if tt > 15:
        tmp += 1
tmp/iter_time
```

```
##################
### P2
##################
import random
import math
def mypi(iter):
    n=iter
    # number of falling inside the circle
    total=0
    for i in range(n):
        x=random.random()
        y=random.random()
        if math.sqrt(x**2+y**2)<1.0:
            total+=1
    # get the pi
    ppi=4.0*total/n
    print(ppi)
    print('Error rate: ',abs(math.pi-ppi)/math.pi)

mypi(10000)
```