Matthew Lacaire
UID: 205288439
CEE/MAE M20
November 14, 2020

Homework 6

Problem 1:

a. Introduction

In problem 1, I am tasked with performing a type of numerical integration using Monte Carlo. I generate choice numbers of random points and tally up those whose x and y values are under the curve, resulting in integral estimations. Then, results are graphed.

b. Models and Methods

I use a 4x1 array numIter with entries 10e2, 10e3, 10e4, 10e5 as the total number of random points generated. Then, I ascribe an x, y point to each one of these values through a for loop:

```
for k = 1:1:length(numIter) %For loop that generates random
values along the domain. Range is (0 1)
    for j = 1:1:numIter(k)
        x(j) = 10*rand(); %X coordinates of random points
        y(j) = rand(); %Range is (0 1) because I know that the
function is never more than 1 for x >=0.
    End
```

My y values go from 0 to 1, not including endpoints. This is because I know that the function given never has y >= 1 in the given domain x = 0 to 10. The x values are anywhere from 0,10, not including endpoints either.

In these critical lines, I determine the values of the x, y arrays that satisfy the condition (that they are under or on the curve) and create a boolean array. Then, I sum the nonzero values, and divide them by the total number of iterations:

```
    integralhits = (y <= (ones(length(numIter(k))) ./ (x.^3 +
1))); %Generating ...
    ...bool array of points under curve.
    integral(k) = 10.*sum(integralhits) / (numIter(k));
```

Note that I also *multiply by ten*. I do so because the test area has an area of 10x1, meaning that the integral is a fraction of this area. Thus, due to this range of the x and y points, we also need to multiply the integral by this value to get correct results.

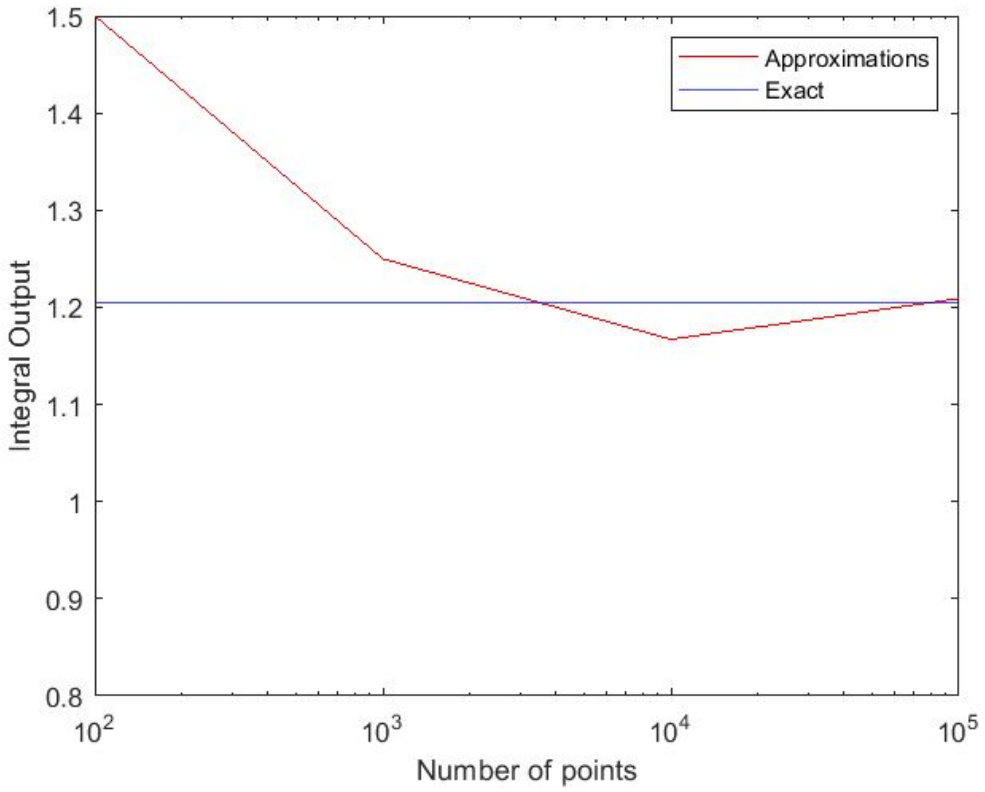
c. Calculations and Results

Most of the details about calculations have been exhausted in the last section; however, there is still something to discuss. In the second section of my code, I perform an exact integral using the following lines:

```
syms x  %Symbolic x variable
f = 1/(x^3 + 1); %Defining function
exactintegral = double(int(f, 0, 10));
```

X is declared as a "symbolic variable", which allows it to represent something else. Alternatively, it represents exact irrational numbers and fractions. It's used in the next line as an argument of f. Then, these symbolic arguments are taken in the int() function, which performs an integral. Note that the result is by default symbolic as well, or exact. It's not a decimal, but rather a combination of logarithms and trig arguments. I need to convert this to a double so that it can be displayed.

Regarding results, I plot my number of iterations with the integral approximations, and also create a horizontal line of the exact value. I label, and choose colors. Note that the xaxis is logarithmic. See below graph:



d. Discussion

From the above graph, we can see that a larger number of test points approximates the integral better. In many successive compilations, this is usually the case as well.

A few details are worth noting. One is that the rand() generates values that are not on "borders", but rather never reach true 0 or 1. This means that in my estimates, I never generated or considered points on borders. Also, I considered all points that have y *less than or equal to* the function to be under the curve. I could have only classified those purely under the curve as satisfactory alternatively. Regardless, this doesn't seem super important, and is rather a question of precision.

Problem 2

e. Introduction

In problem 2, I am tasked with implementing a custom probability distribution. Using a large number of samples (I take n=10,000), I generate a histogram of the x-values of the probability density function. Then, I graph the distribution alongside p(x) and compare.

f. Models and Methods

I begin by declaring numSamples to be 10,000. I then use a for loop to create a vector y of length numSamples. Each iteration, the function myRand() is called and assigned to y(k).

myRand is below, and I'll explain my thinking:

```
function x = myRand()

y = rand(); %Random num. b/t 0,1.

x = 2 - sqrt(4-4*y); %Equation for finding the p(x) value. See
report for derivation.

end
```

Essentially, I know that the double improper integral of the cumulative probability distribution function is 1, and that this can aid calculations. I first create a y between 0 and 1, and work backwards to find the upper bound of the integral P(x), which is x itself. I perform the following steps to get x with a given y generated:

$$p(x) = -0.5x + 1$$

Here, I notice that p(x) is zero for x outside of 0,2. Thus, all I need to do is integrate this equation for x b/t 0 and 2 to get the cumulative density function. This is what I do, and I integrate:

$$P(x) = -0.25x^2 + x + C$$

I take C to be 0, because at x=0, the integral should be infinitely thin and thus zero. I rearrange further:

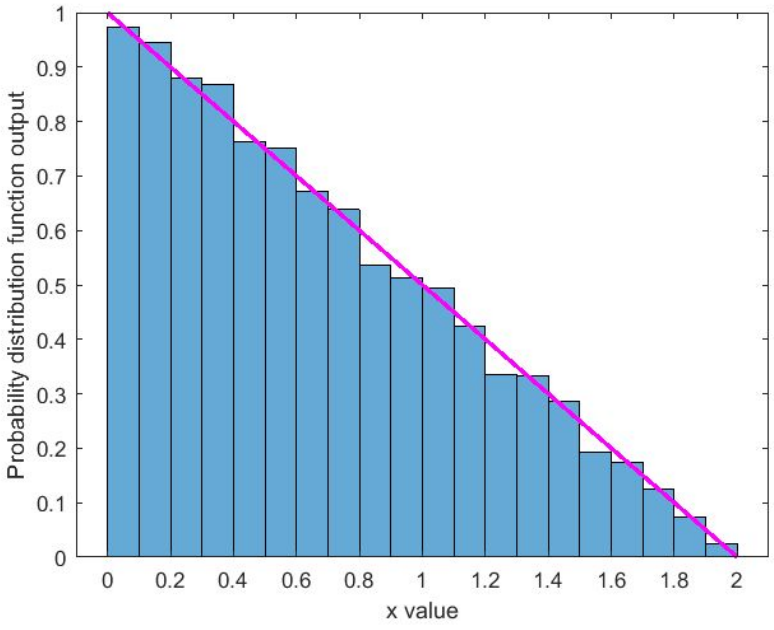
$$0 = -0.25x^2 + x - y$$

If I continue solving (I did so on paper), I use the quadratic formula and get:
$2 - sqrt(4 - 4y)$. This is the negative root of the quadratic, since the positive root makes no sense. The positive root would return numbers outside 0,2 for acceptable y.

This is thus the argument in the myRand function. I then begin graphing and comparing.

g. Calculations and Results

My probability distribution function, when graphed along with the x values of the given points in twenty bins, looks like this:



I also explicitly graphed the p(x) for 0 to 2 in thick magenta. Note that outside of the range of 0,2, both the probability distribution outputs (the values in bins) and the magenta line are flat, horizontal lines at y = 0. This is because p(t) returns 0 for anything outside of 0,2.

h. Discussion

This result makes sense; the nonzero p(x) graphed with the estimated p(x) based on 10,000 trials is almost exact; the histogram follows the shape of the explicit p(x). To summarize, by stating that the integral of the prob. distribution function over infinity is at max one, I determined given x values of the probability function by integrating p(x) where nonzero, then solving for x explicitly in terms of constants and y with the quadratic formula. Thus, I determined the general shape of the distribution and normalized it such that it predicts the function p(x) as necessary.

Problem 3

i. Introduction

In problem 3, I'm tasked with working with the Birthday paradox to simulate the number of two or more people sharing a birthday (or birthdays) among a select cohort of people . Note that other quantitative/numerical derivations are available in the discussion. For a given size of a cohort, I iterate two thousand times. Then, summing the total number of shared birthday instances and dividing this by the number of iterations, I get approximate estimates.

j. Models and Methods

I begin by establishing a numPeople, the cohort size for each iteration. Then, I establish a numIter, the number of times the cohort is randomized and birthday results are evaluated. I also instantiate a sharedbdays array of length numIter, so that for each cohort I can store the number of instances of people sharing birthdays.

Skipping the algorithm (which I will explain in detail in the next section) I calculate probability of these same-birthday-events and print out a line of the form below with fprintf:

```
With 23 people, there is a 50.900000000 percent chance two or
more people share birthdays.
```

k. Calculations and Results

My algorithm is rather complicated, but I believe that it outputs the correct results. I'll try to walk through it logically here. It's pasted below:

```
for k = 1:numIter
    for n = 1:numpeople
        b(n) = unidrnd(365); %Generating random numbers 1 to 365
for each cell of b array.
    end
    for i = 1:numpeople
        for j = i+1:numpeople %j starts at i+1, because if at
just i, we would count people having...
            ...birthdays with themselves, which is obviously
illogical.
            if b(i) == b(j)
                sharedbdays(k) = sharedbdays(k) + 1;%Searching
along array. If bday vals are the same, we count...
                ...a shared birthday for the given iteration
val.
            end
        end
    end
end
sharedBdays = 0; %Shared birthdays in 1x1 form; not a vector.
```

Basically, I begin by creating a for loop for each iteration. Within each iteration, I create a vector b, and assign "numpeople" amount of random birthdays inside. After this, I create a further double-nested for loop, which uses variables i and j. Basically, for each b(i), I check the points from (i+1) to the end of the array to see if there are any duplicate values (same birthday instances). If an array has two same birthdays, the algorithm outputs one same-birthday event. If there are three people having the same birthday, the algorithm outputs *three* events, because each person has a same-birthday-event with the other two, totaling three instances. The algorithm also checks for other duplicates in the arrays (other possible shared birthdays).

Each time an instance is catalogued, the value sharebdays(k) is increased by 1. Because the vector is a function of k, each iteration has its own sharebdays cell.

Then, I do the following:
sharedBdays = 0; %Shared birthdays in 1x1 form; not a vector.

```
for i = 1:numIter
```

```
    if sharedbdays(i) ~= 0 %Checking all of the cells where
people actually have shared birthdays.
        sharedBdays = sharedBdays + 1; %Finding total number of
shared birthdays.
    end
end
```

In this for loop, I basically "dump" all of the nonzero contents of the vector sharedbdays into the integer sharedBdays. This is the total number of instances over all iterations. I divide this by the number of iterations to get average instances per cohort (probability). I also multiply by 100 for further readability in display as well.

l. Discussion

My results seem logical, and when compared to analytical calculations, are fairly robust. Statistically, the probability of each person *not* having the birthday of any of the previous members of the cohort is:

(364/365)*(363/365)*(362/365)...

Finally, we subtract one from this number to get the probability that *at least two people* share a birthday in the group (any given birthday. This is why the numerator decreases with # of people). The result we get is 0.507, which tells us that at the critical cohort size 23, it is actually more likely than not that two people share the same birthday (which can be any day of the year). As shown above, this is very close to my numerical approximation. Additionally, when comparing with different analytical results derived and freely available online and altering the cohort size in my code, I confirm that my code is fairly accurate as well.