Question 1:. [5pts, HELP] Write a function called contains that takes a region and checks if it contains a given point. You may find it helpful to use this function and python's "list comprehensions" in order to answer questions in this assignment.

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt

def contains(point, region):
    if point >= region[0] and point <= region[1]:
        return True
    else:
        return False
```

Question 2: [10pts, HELP] What is the probability of getting x=1 for regions containing x=0? NOTE: this is not just one over the number of regions containing both 1 and 0 – that doesn't take into account the fact that each region assigns x a different probability of occurring.

In [14]:

```
def findProbability(regions, x):

    probability = 0
    possibleRegions = []
    numerator = 0
    denominator = 0

    #first count # of regions that contain 0 and store them in t
heir own array
    for r in regions:
        if contains(0, r):
            possibleRegions.append(r)

    #next count # of regions that contain x out of our regions t
hat contain 0
    consRegions = []
    for r in possibleRegions:
        if contains(x, r):
```

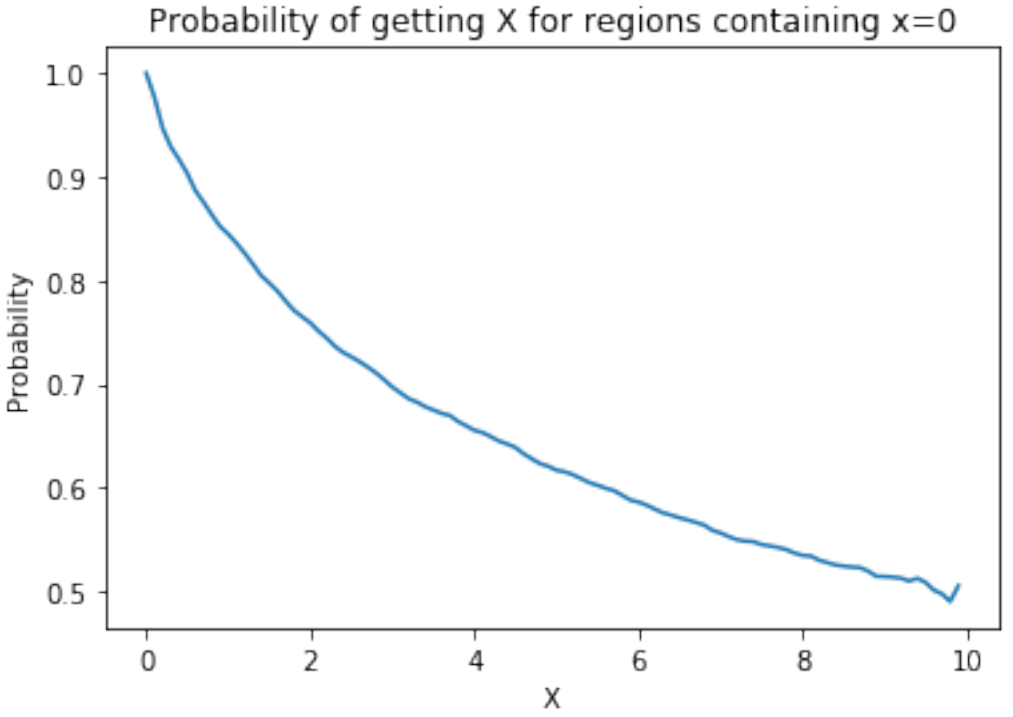
```
            consRegions.append(r)

    #add up the probabilities accordingly for numerator
    for r in consRegions:
        numerator += 1/len(consRegions) * (1/(r[1]-r[0]))

    #add up the probabilities accordingly for denominator
    for r in possibleRegions:
        denominator += 1/len(possibleRegions) * (1/(r[1]-r[0]))

    probability = numerator/denominator
    return probability


startpoints = np.random.uniform(-10, 10, size = 10000)
endpoints = np.random.uniform(-10, 10, size = 10000)

regions = []
for i in range(len(startpoints)):
    if startpoints[i] <= endpoints[i]:
        regions.append((startpoints[i], endpoints[i]))
    else:
        regions.append((endpoints[i], startpoints[i]))

probability = findProbability(regions, 1)
print(probability)
```

```
0.8513556014374062
```

Question 3: [20pts, SOLO] Plot the probability of getting x for x ranging from 0 to 10, for regions containing x=0. What does this function look like? Write a sentence explaining why intuitively.

In [24]:

```
startpoints = np.random.uniform(-10, 10, size = 10000)
endpoints = np.random.uniform(-10, 10, size = 10000)

regions = []
for i in range(len(startpoints)):
    if startpoints[i] <= endpoints[i]:
        regions.append((startpoints[i], endpoints[i]))
    else:
        regions.append((endpoints[i], startpoints[i]))

probability = 0
probabilityplot = []
countplot = np.arange(0, 10, 0.1)
count = 0

#then sum probabilities accordingly
for x in countplot:
    probability = findProbability(regions, x)
    probabilityplot.append(probability)

plt.plot(countplot, probabilityplot)
plt.title("Probability of getting X for regions containing x=0")
plt.xlabel("X")
plt.ylabel("Probability")
#plt.yscale("log")
plt.show()
```

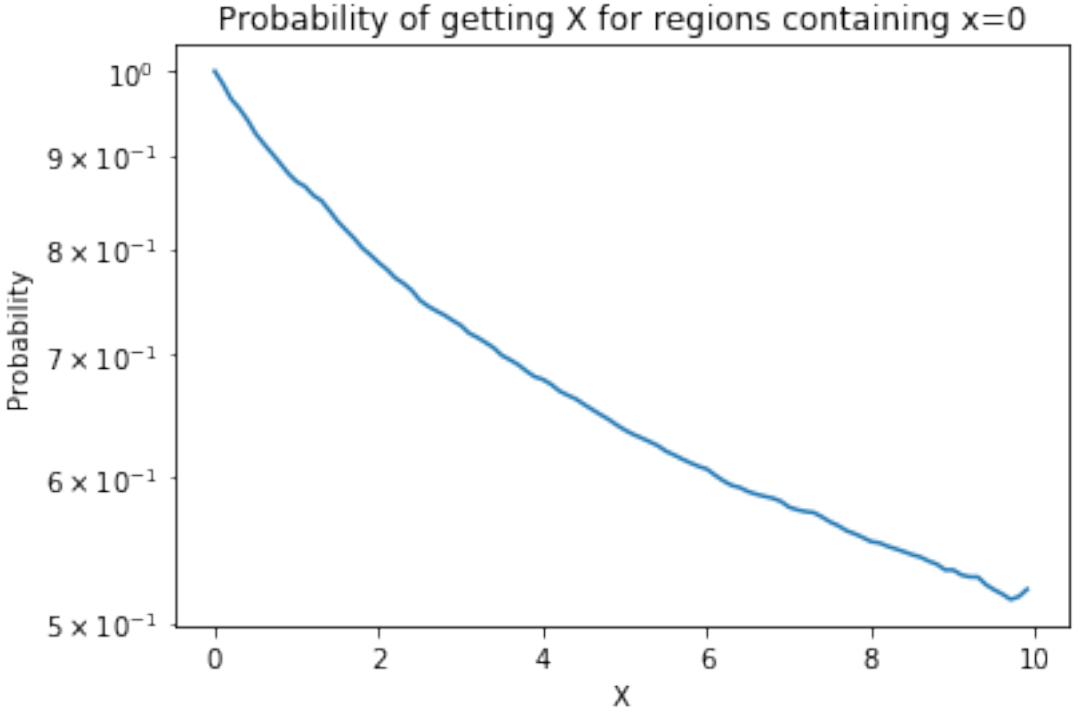


This function looks like an exponentially decreasing curve. Intuitively, this makes sense because as the regions become larger, two factors are at play. The number of regions that contain x=0 decreases as the size of the regions increases, and additionally, the probability of "getting" x = 0 decreases as the size of the region increases. Thus, the probability of getting x decreases at a decreasing rate.

Question 4. [5pts, HELP] One way to check if the curve has an exponential decrease is to plot a logarithmic y axis and look for a straight line. Why does this check if the curve is exponential?

In [22]:

```
plt.plot(countplot, probabilityplot)
plt.title("Probability of getting X for regions containing x=0")
plt.xlabel("X")
plt.ylabel("Probability")
plt.yscale("log")
plt.show()
```



This mathematical operation checks if a curve is exponential becuase a logarithmic function is the inverse function to an exponential function. Thus, the log plot scales the y-axis so that each increment increases by a power of ten. Therefore, the y-axis exponentially decreases as you move down. Plotting a decreasing exponential function on this kind of scale then results in a linear graph with a negative slope. The "rise" in the rise/run of this graph decreases exponentially, but appears to decrease linearly due to the logarthmic scaling of the y-axis.

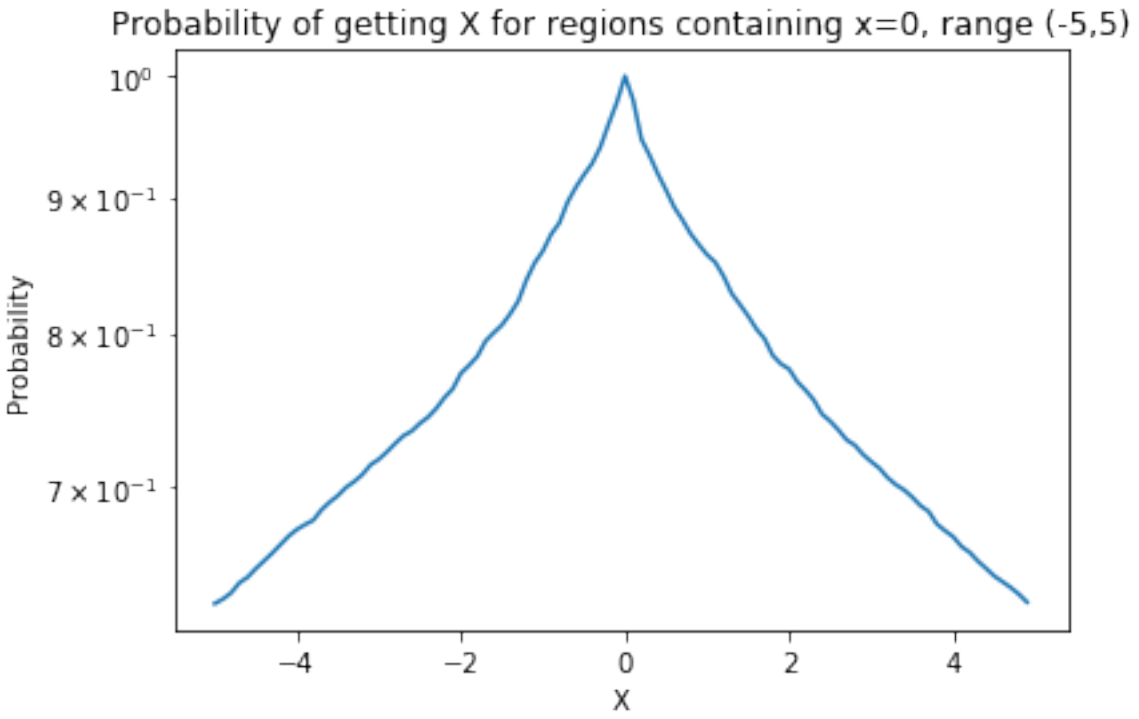
Question 5. [10pts, SOLO] Plot Q3 with a logarithmic y axis for x ranging from -5 to 5, and x ranging from -10 to 10. What do these two plots show? How do you interpret them? Explain in a few sentences.

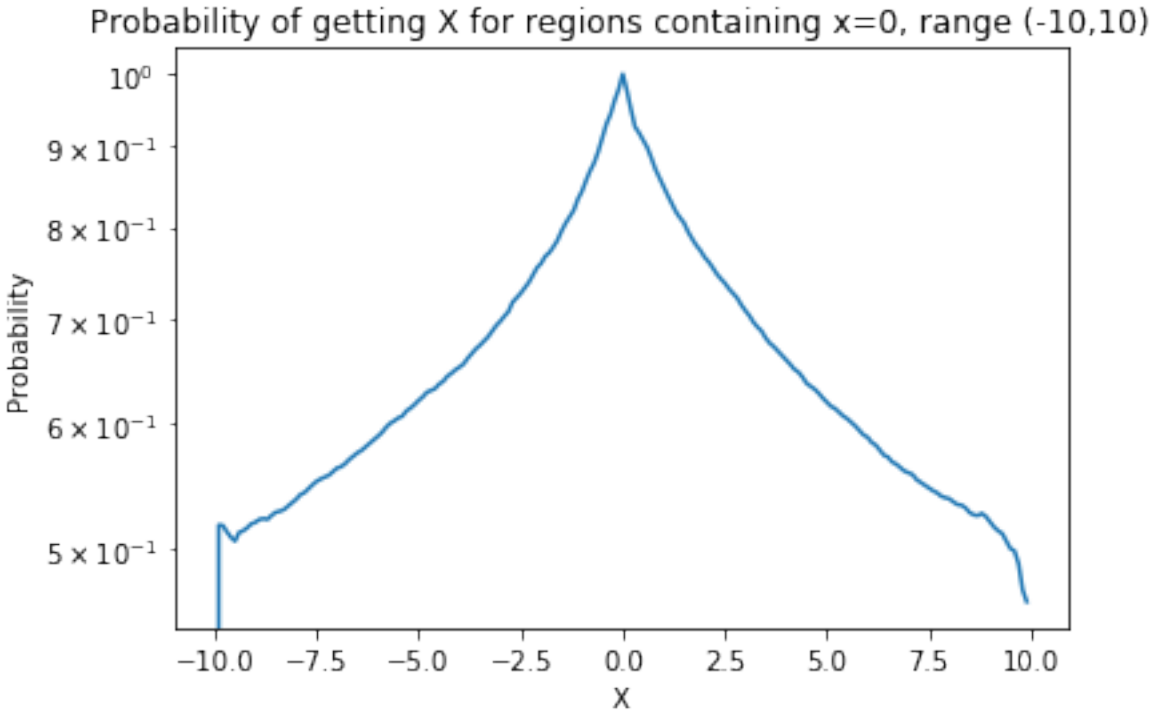
In [25]:

```
startpoints = np.random.uniform(-10, 10, size = 10000)
```

```
endpoints = np.random.uniform(-10, 10, size = 10000)

regions = []
for i in range(len(startpoints)):
    if startpoints[i] <= endpoints[i]:
        regions.append((startpoints[i], endpoints[i]))
    else:
        regions.append((endpoints[i], startpoints[i]))

probability = 0
probabilityplot = []
countplot = np.arange(-5, 5, 0.1)

#then sum probabilities accordingly
for x in countplot:
    probability = findProbability(regions, x)
    probabilityplot.append(probability)

plt.plot(countplot, probabilityplot)
plt.title("Probability of getting X for regions containing x=0,
range (-5,5)")
plt.xlabel("X")
plt.ylabel("Probability")
plt.yscale("log")
plt.show()


startpoints = np.random.uniform(-10, 10, size = 10000)
endpoints = np.random.uniform(-10, 10, size = 10000)

regions = []
for i in range(len(startpoints)):
    if startpoints[i] <= endpoints[i]:
        regions.append((startpoints[i], endpoints[i]))
    else:
        regions.append((endpoints[i], startpoints[i]))

probability = 0
probabilityplot = []
countplot = np.arange(-10, 10, 0.1)

#then sum probabilities accordingly
for x in countplot:
    probability = findProbability(regions, x)
    probabilityplot.append(probability)
```

```
plt.plot(countplot, probabilityplot)
plt.title("Probability of getting X for regions containing x=0,
range (-10,10)")
plt.xlabel("X")
plt.ylabel("Probability")
plt.yscale("log")
plt.show()
```





These two plots show that the probability for geting X for regions containing x=0 is symmetrical at the axis of x=0. This means that when looking at regions that contain x=0, we can conclude that the probability of "getting" a value X decreases at a decreasing rate depending on the DISTANCE away from 0, not depending on the value. That is, the probability of getting -6 is equally smaller than 1 to the probability of getting +6, and these values are exponentially larger than the probabilities of getting - or + 7. In addition, the nearly linear graphs seen in these plots furthers the point that the probability decreases with distance away from x=0 at a decreasing rate.

Question 6. [10pts, SOLO] In previous questions, we've been assuming that people implement the law perfectly and we have been trying to approximate their behavior using 10,000 regions. However, people themselves have limited resources. What if people themselves only used a few consequential regions inorder to compute generalizations? Re-plot Question 2 using only 10, 100, and 1000 consequential regions. What patterns do you see?

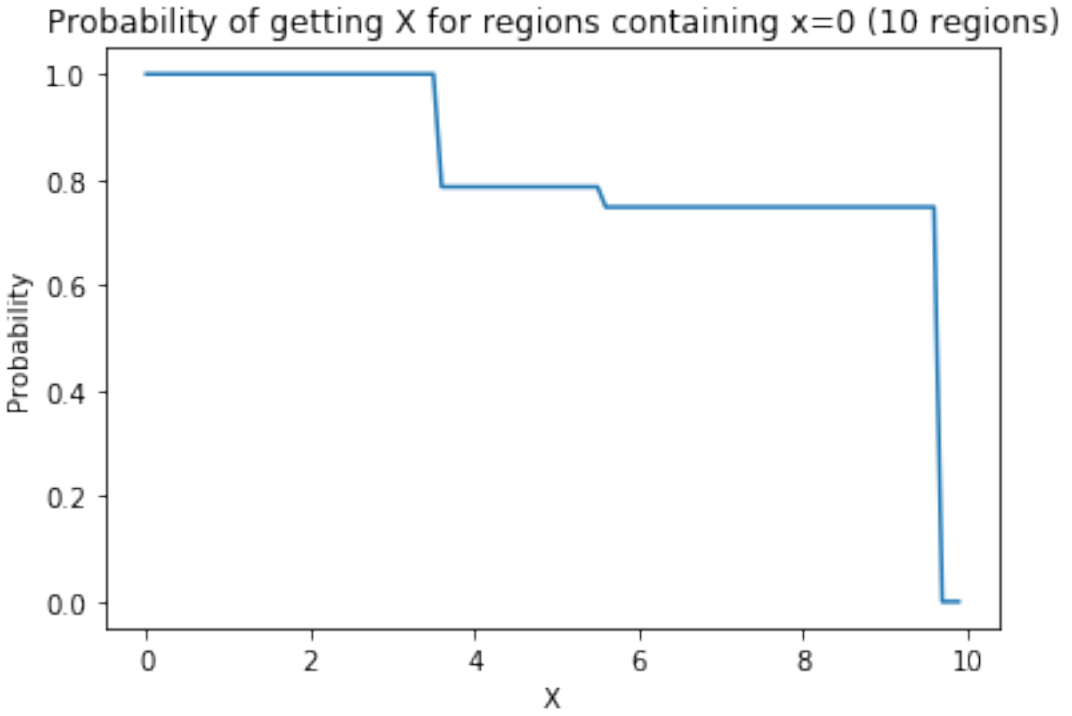
In [38]:

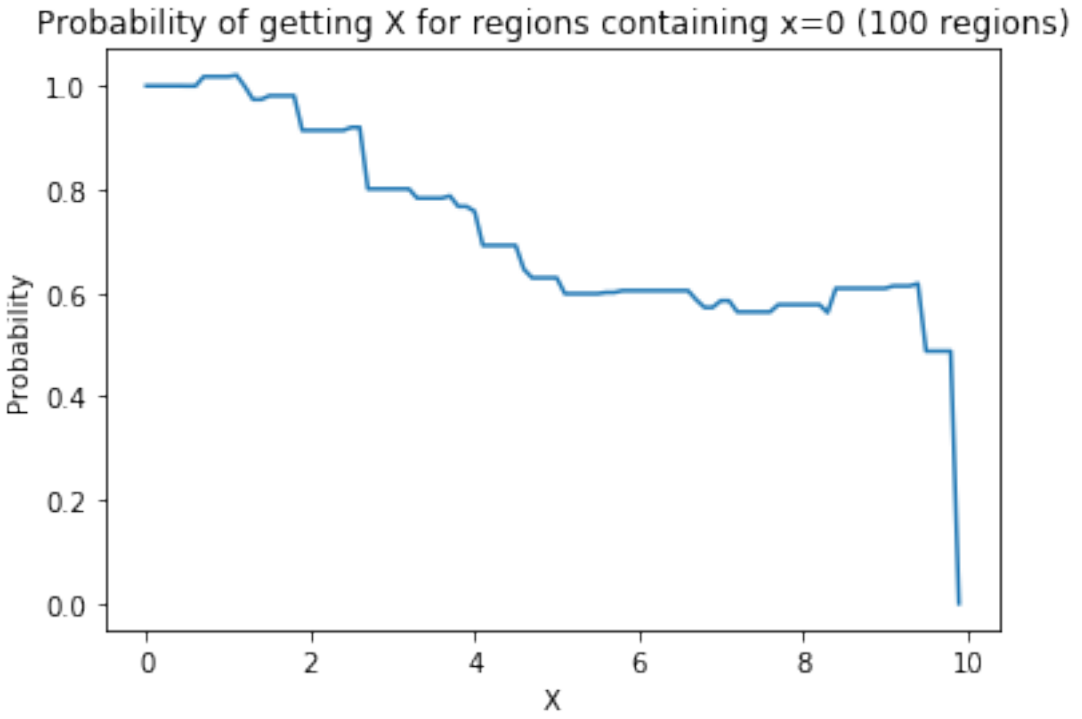
```
startpoints = np.random.uniform(-10, 10, size = 10000)
endpoints = np.random.uniform(-10, 10, size = 10000)

regions = []
for i in range(len(startpoints)):
    if startpoints[i] <= endpoints[i]:
        regions.append((startpoints[i], endpoints[i]))
    else:
        regions.append((endpoints[i], startpoints[i]))

#create 3 (size 10, 100 and 1000) consequential region sets
regions10 = []
regions100 = []
regions1000 = []

for i in range(10):
    regions10.append(regions[i])

for i in range(100):
    regions100.append(regions[i+10])

for i in range(1000):
    regions1000.append(regions[i+110])
```

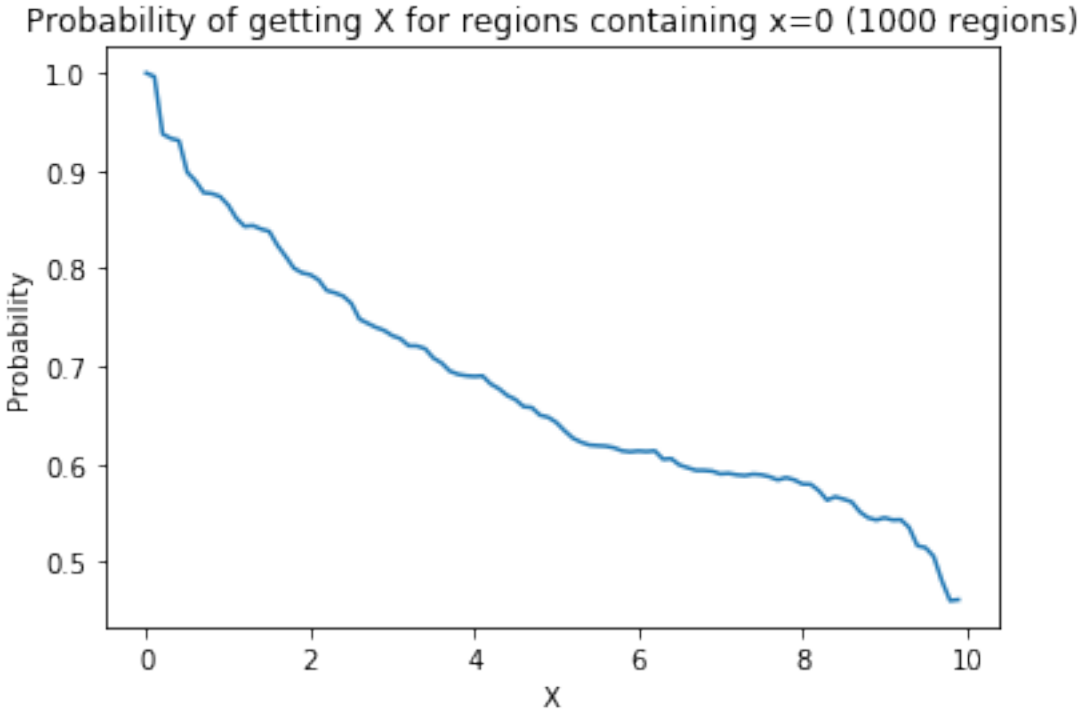
```
print(len(regions10))
print(len(regions100))
print(len(regions1000))

#replot using regions10
probability = 0
probabilityplot = []
countplot = np.arange(0, 10, 0.1)
count = 0

#then sum probabilities accordingly
for x in countplot:
    probability = findProbability(regions10, x)
    probabilityplot.append(probability)

plt.plot(countplot, probabilityplot)
plt.title("Probability of getting X for regions containing x=0 (
10 regions)")
plt.xlabel("X")
plt.ylabel("Probability")
plt.show()

#replot using regions100
probability = 0
probabilityplot = []
countplot = np.arange(0, 10, 0.1)
count = 0

#then sum probabilities accordingly
for x in countplot:
    probability = findProbability(regions100, x)
    probabilityplot.append(probability)

plt.plot(countplot, probabilityplot)
plt.title("Probability of getting X for regions containing x=0 (
100 regions)")
plt.xlabel("X")
plt.ylabel("Probability")
plt.show()

#replot using regions1000
probability = 0
probabilityplot = []
```

```
countplot = np.arange(0, 10, 0.1)
count = 0

#then sum probabilities accordingly
for x in countplot:
    probability = findProbability(regions1000, x)
    probabilityplot.append(probability)

plt.plot(countplot, probabilityplot)
plt.title("Probability of getting X for regions containing x=0 (
1000 regions)")
plt.xlabel("X")
plt.ylabel("Probability")
plt.show()
```

```
10
100
1000
```







In each plot, I see a decreasing function, however not at a steady rate. The more consequential regions we sample from, the more exponential the curve begins to look. In addition, in the sample size of 10 and 100 region graphs, the probability of getting an x value of 10 is 0. However, in the 1000 region graph, the probability of getting x=10 is around ~0.5, which is more indicative of what my Question 3 graph showed.

Question 7. [10pts, HELP] Describe a way you could test how many consequential regions people actually made use of in this kind of generalization. Could you tell the difference between 10 and 10,000? Could you tell the difference between 10,000 and 20,000, why or why not?

One way you could test how many consequential regions people actually made use of in this kind of generalizaion would be to run the function several times, each time with an increasingly large number of consequential regions. Ideally, you could increment the number of regions by a small amount, like 10 or 20 regions per time running the function. Then, you could plot each outcome using the logarthmic y-scale like we did in question 4. You could then run a best fit line through the graph. Whichever graph matches the best fit line most accurately FIRST, would probably be close to the amount of consequential regions one would actually need to run this kind of generalization accurately.

In this way, you would see a large difference between 10 and 10000 consequential regions (as question 6 shows). However, the difference between 10,000 and 20,000 would probably be less noticeable. At one point the amount of consequential regions one would need to make an accurate best fit line would plateau somewhere in that interval, so the difference would be close to negligable.