In [1]:
```python
import random as rnd

import matplotlib.pyplot as plt

import numpy as np

import math

from scipy.stats import norm

from PIL import Image
from IPython.display import display
```

I worked with the wonderful __!~!!Michael Scutari!!! on this project

And also the great Peter Banyas __ ^^

Question 1

In [2]:
```python
img = Image.open('q1.png')

display(img)
```

Airline Overbooking

Posted on September 11, 2012 by Jonathan Mattingly | Comments Off

An airline knows that over the long run, 90% of passengers who reserve seats for a flight show up. On a particular flight with 300 seats, the airline sold 324 reservations.

1. Assuming that passengers show up independently of each other, what is the chance that the flight will be overbooked ?
2. Suppose that people tend to travel in groups. Would that increase of decrease the probability of overbooking ? Explain your answer.
3. Redo the the calculation in the first question assuming that passengers always travel in pairs. Are your answers to all three questions consistent ?

Q1, part 1

In [3]:
```python
img = Image.open('q1_part1.png')

display(img)
```

1) 90 % passengers. 300 seats 324 reservations.

$\therefore \mathbb{P}(\text{person appears}) = p = 0.9$

$\therefore \mathbb{P}(\text{More than 300 people appear}) = \mathbb{P}(\text{overbooked})$

let $X \sim \text{Binomial}(\underset{n}{\underset{\uparrow}{324}}, \underset{p}{\underset{\uparrow}{0.9}})$

$$\therefore \mathbb{P} = \mathbb{P}(X > 300) = \mathbb{P}(X \geq 301)$$

$$= \sum_{k=301}^{324} \binom{324}{k} \cdot (0.9)^k \cdot (0.1)^{324-k}$$

Normal approximation:

$\mu = np$

$\sigma = \sqrt{npq} = \sqrt{np(1-p)}$

$$\therefore \alpha = \frac{a - \frac{1}{2} - \mu}{\sigma}, \quad \beta = \frac{b + \frac{1}{2} - \mu}{\sigma}$$

$$\therefore \int_{\alpha}^{\beta} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} \, dz = \Phi(\beta) - \Phi(\alpha)$$

$$\approx 0.0497 \ (3sf)$$

$$\therefore \mathbb{P}(X > 300) \approx \underline{\underline{0.0497}}$$

In [4]:
```
#Q1 normal approx.

def normalApproxWithContinuityCorrection(n, p, a, b):

    mu = n * p
    sigma = (n * p * (1 - p)) ** 0.5

    #shifts
    a = a - 0.5
    b = b + 0.5

    return norm.cdf(b, loc=mu, scale=sigma) - norm.cdf(a, loc=mu, scal
```

In [5]:
```
res = normalApproxWithContinuityCorrection(n=324, p=0.9, a=301, b=324)
print(str(res))
```

```
0.049661136484298374
```

Q1, part 2

In [6]:
```
img = Image.open('q1_part2.png')
display(img)
```

2) Another binomial $Y \sim B(\frac{324}{g}, 0.9)$

g = group size (only valid for values of g that go perfectly into 0.9)

$P(Y > \frac{300}{g})$ let $N_{groups} = \frac{324}{g}$

$= P(Y_g > 300) = 1 - P(Y_g \leq 300)$

$$= \sum_{k=\frac{300}{g}+1}^{\frac{324}{g}} \binom{324/g}{k} \cdot (0.9)^k \cdot (0.1)^{\frac{324}{g}-k}$$

Binomial approximation:

$$\mu = \frac{324}{g} \times 0.9$$

$$\sigma = \sqrt{\frac{324}{g} \times 0.9 \times 0.1}$$

$$\alpha = \frac{a - \frac{1}{2} - \mu}{\sigma}$$

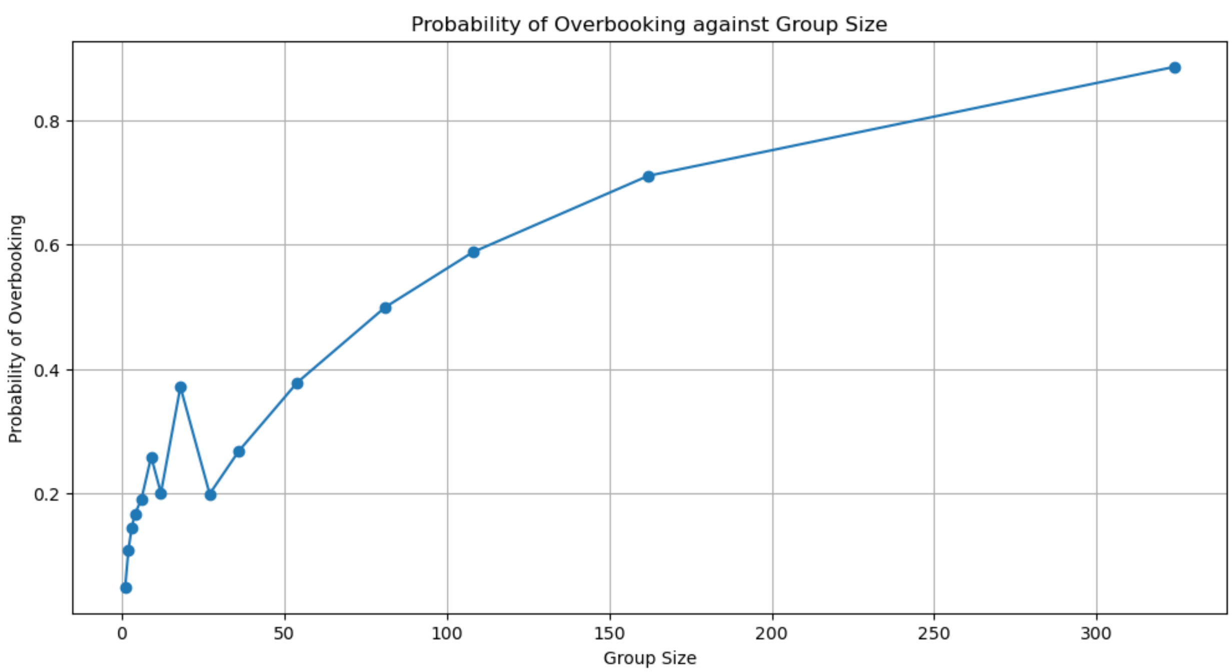
$$\int_{\beta}^{\alpha} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z} \, dz \qquad \beta = \frac{b + \frac{1}{2} - \mu}{\sigma}$$

$\therefore$ I expect the range of the bounds (α, β) to increase ($\because$ σ gets smaller as g increases).

$\therefore$ as g gets larger, I expect it to be more and more probable that the flight is overbooked.

In [7]:

```
### image of part two

#group size values that can divide 324 exactly
g_vals = [1, 2, 3, 4, 6, 9, 12, 18, 27, 36, 54, 81, 108, 162, 324]

#this array will contain the probabilities of being overbooked for eac
probs = []

for g in g_vals:

    n_in = 324//g
    p_in = 0.9

    a_in = (300//g) + 1
    b_in = 324//g

    probs.append(normalApproxWithContinuityCorrection(n=n_in, p=p_in,


# Plot the result
plt.figure(figsize=(12, 6))
plt.title("Probability of Overbooking against Group Size")
plt.plot(g_vals, probs, marker='o')
plt.ylabel("Probability of Overbooking")
plt.xlabel("Group Size")
plt.grid(True)
plt.show()

# Print the result as a rounded percentage with 2 decimal places
print("Group Size | Probability of Overbooking")
print("-------------------------------------")
for i in range(len(g_vals)):
    print(f"{g_vals[i]:<10} | {probs[i]*100:.2f}%")
```



Group Size	Probability of Overbooking
1	4.97%
2	10.92%
3	14.48%
4	16.71%
6	19.07%
9	25.92%
12	20.07%
18	37.15%
27	19.94%
36	26.85%
54	37.87%
81	49.94%
108	58.80%
162	71.08%
324	88.60%

Q1, part 3

In [8]:
```
img = Image.open('q1_part3.png')
display(img)
```

3) As can be seen in code above

for case group size, $s = 2$

the probability of being overbooked = 10.92%

∴ It agrees with my observation in part 2

Question 2

In [9]:
```
img = Image.open('q2.png')
display(img)
```

Leukemia Test

Posted on August 13, 2012 by Jonathan Mattingly | Leave a comment

A new drug for leukemia works 25% of the time in patients 55 and older, and 50% of the time
in patients younger than 55. A test group has 17 patients 55 and older and 12 patients younger than 55.

1. A uniformly random patient is chosen from the test group, and the drug is administered and it is a success. What is the probability the patient was 55 and older?
2. A subgroup of 4 patients are chosen and the drug is administered to each. What is the probability that the drug works in all of them?

In [10]:
```python
img = Image.open('q2_part1.png')
display(img)
```

1)

	12 younger than 55	17 older than 55
Drug works	1/2	1/4
Drug does not work	1/2	3/4

$$\mathbb{P}(\text{older } 55 \mid \text{success}) \Rightarrow \frac{\mathbb{P}(A)}{\mathbb{P}(B)} \mathbb{P}(B \mid A)$$

$$= \frac{\mathbb{P}(\text{older } 55)}{\mathbb{P}(\text{success})} \cdot \mathbb{P}(\text{success} \mid \text{older } 55)$$

$$\mathbb{P}(\text{success}) = \mathbb{P}(\text{success} \mid \text{older } 55)\, \mathbb{P}(\text{older } 55)$$
$$+ \mathbb{P}(\text{success} \mid \text{younger } 55)\, \mathbb{P}(\text{younger } 55)$$

$$= \frac{1}{4} \cdot \frac{17}{29} + \frac{1}{2} \cdot \frac{12}{29} = \frac{41}{116}$$

$$\therefore \mathbb{P}(\text{older } 55 \mid \text{success}) = \frac{17}{29} \cdot \frac{116}{41} \cdot \frac{1}{4} = \frac{17}{41} \quad (\approx 0.415)$$

In [11]:
```
img = Image.open('q2_part2.png')
display(img)
```

2) $\mathbb{P}$(drug works on all of them).

$= \mathbb{P}$(drug works on P_1) $\cdot \mathbb{P}$(drug works on P_2)
$\cdot \mathbb{P}$(drug works on P_3) $\cdot \mathbb{P}$(drug works on P_4)

Assuming we know nothing about which group each person is in, have to assume independent.

$\therefore = \mathbb{P}$(drug works on a person)^4

$= \left(\frac{41}{116}\right)^4 \approx 0.0156$

from previous q.

NOTE: Prof Mattingly mentioned that we have not learnt appropriate tools to do this $\therefore$ this may be incorrect.

Question 3

In [12]:
```
img = Image.open('q3.png')
display(img)
```

Exercise 5.1: Using the Normal Approximation to Binomial

Write code to solve each of the follwing problems concering rolls of a fair die. Organize your results in a nice table.

1. What is the exact probablity of rolling a 6-sided die an even number more than 70% of the time in 10 rolls, 100 rolls, 1000 rolls.
2. What is the exact probablity of rolling the number 1 more than 20% of the time in 10 rolls, 100 rolls, 1000 rolls of a 6-sided die.
3. What is the exact probablity of rolling the number 1 more than 9% of the time in 10 rolls, 100 rolls, 1000 rolls of a 20-sided die.
4. Approximate each of the above probablities using a normal approximation with and without the contiunity correction. (You should use the contiunity corection at the level of the number of rolls anot the percentages).

Discuss any trends you see. How does the approximation behave as n increases or as thje probablities of the events decrease? How does the region you are integrating the normal curve over change as n increase? Does this have any implications for the approximation by a normal?

In [13]:
```python
img = Image.open('q3_parts1-3.png')
display(img)
```

1) $X \sim B(n, p)$

$$\mathbb{P}(X = k) = \binom{n}{k} \cdot (p)^k \cdot (q)^{n-k}$$

$p = 0.5 \quad \because \quad \mathbb{P}(\text{even}) = \frac{3}{6} = \frac{1}{2}.$

$$\mathbb{P}(X > 0.7n) = \sum_{k=0.7n+1}^{n} \binom{n}{k} (p)^k (q)^{n-k}$$

2) $X \sim B(n, \frac{1}{6})$

$$\mathbb{P}(X > 0.2n) = \sum_{k=0.2n+1}^{n} \binom{n}{k} \left(\frac{1}{6}\right)^k \left(\frac{5}{6}\right)^{n-k}$$

3) $X \sim B(n, \frac{1}{20})$

$$\mathbb{P}(X > 0.09n) = \sum_{k=0.09n+1}^{n} \binom{n}{k} \left(\frac{1}{20}\right)^k \left(\frac{19}{20}\right)^{n-k}$$

Q3, part 1

In [14]:

```
def numEven(n):
    # n = num trials
    # p = probability of success on each trial

    running_prob = 0

    for i in range(math.ceil((0.7*n) + 1), n+1):
        running_prob += math.comb(n, i) * ( (1/2)**i ) * ( (1/2)**(n-i

    return running_prob
```

In [15]:

```
print("Probablity of rolling a 6-sided die an even number more than 70
print()
print("
print()
print("
print()
print("
```

```
Probablity of rolling a 6-sided die an even number more than 70% of t
he time,


in 10 rolls: 5.46875%


in 100 rolls: 0.0016080007647833168%


in 1000 rolls: 3.7668507235258085e-36%
```

Q3, part 2

In [16]:

```
def numOnes(n):
    # n = num trials

    running_prob = 0

    for k in range(math.ceil((0.2*n) + 1), n+1):

        running_prob += math.comb(n, k) * ( (1/6)**k ) * ( (5/6)**(n-k

    return running_prob
```

```
In [17]: print("What is the exact probablity of rolling the number 1 more than
         print()
         print("
         print()
         print("
         print()
         print("
```

```
What is the exact probablity of rolling the number 1 more than 20% of
the time,


in 10 rolls: 22.477320212874055%


in 100 rolls: 15.188784790416667%


in 1000 rolls: 0.2487549278695209%
```

Q3, part 3

```
In [18]: def numTwentySidedOnes(n):
             # n = num trials
             running_prob = 0

             for k in range(math.ceil((0.09*n) + 1), n+1):

                 running_prob += math.comb(n, k) * ( (1/20)**k ) * ( (19/20)**(

             return running_prob
```

In [19]:

```
print("What is the exact probablity of rolling the number 1 (20-sided
print()
print("
print()
print("
print()
print("
```

```
What is the exact probablity of rolling the number 1 (20-sided dice)
more than 9% of the time in,


in 10 rolls: 8.61383558993164%


in 100 rolls: 2.8188294163416%


in 1000 rolls: 5.0649899888673875e-06%
```

Q3, part 4

```python
In [20]: img = Image.open('q3_part4.png')
display(img)
```

4) i) $n = 10, 100, 1000$

$p = \frac{1}{2}$

$a = 0.7n + 1$

$b = n$

ii) $n = 10, 100, 1000$

$p = 1/6$

$a = 0.2n + 1$

$b = n$

iii) $n = 10, 100, 1000$

$p = \frac{1}{20}$

$a = 0.09n + 1$

$b = n$

In [21]:
```
def binomialApproximationWContinuity(n, p, a, b):

    mu = n*p
    sigma = math.sqrt(n*p*(1-p))

    alpha = (a - 0.5 - mu) / sigma

    beta = (b + 0.5 - mu) / sigma

    return norm.cdf(beta) - norm.cdf(alpha)


def binomialApproximationNoContinuity(n, p, a, b):

    mu = n*p
    sigma = math.sqrt(n*p*(1-p))

    alpha = (a - mu) / sigma

    beta = (b - mu) / sigma

    return norm.cdf(beta) - norm.cdf(alpha)
```

Q3, part 4-1

```
In [22]: rolls = [10,100,1000]

print("For 1. What is the exact probablity of rolling a 6-sided die ar
print()

for roll in rolls:
    print("Normal approximation for " + str(roll) + " rolls w/ continu
    print("

print()

for roll in rolls:
    print("Normal approximation for " + str(roll) + " rolls w/ no cont
    print("
```

```
For 1. What is the exact probablity of rolling a 6-sided die an even
number more than 70% of the time in 10 rolls, 100 rolls, 1000 rolls.

Normal approximation for 10 rolls w/ continuity correction:
                                                             5.66710
3988860456%
Normal approximation for 100 rolls w/ continuity correction:
                                                             0.00206
57506912491463%
Normal approximation for 1000 rolls w/ continuity correction:
                                                             0.0%

Normal approximation for 10 rolls w/ no continuity correction:
                                                             2.81070
84432797413%
Normal approximation for 100 rolls w/ no continuity correction:
                                                             0.00133
45749015902797%
Normal approximation for 1000 rolls w/ no continuity correction:
                                                             0.0%
```

Q3, part 4-2

In [23]:
```
rolls = [10,100,1000]

print("For 2. What is the exact probablity of rolling the number 1 mor
print()

for roll in rolls:
    print("Normal approximation for " + str(roll) + " rolls w/ continu
    print("

print()

for roll in rolls:
    print("Normal approximation for " + str(roll) + " rolls w/ no cont
    print("
```

```
For 2. What is the exact probablity of rolling the number 1 more than
20% of the time in 10 rolls, 100 rolls, 1000 rolls of a 6-sided die.

Normal approximation for 10 rolls w/ continuity correction:
                                                               23.9750
06109344356%
Normal approximation for 100 rolls w/ continuity correction:
                                                               15.1835
8910818507%
Normal approximation for 1000 rolls w/ continuity correction:
                                                               0.20468
25778043937%

Normal approximation for 10 rolls w/ no continuity correction:
                                                               12.8949
51764540096%
Normal approximation for 100 rolls w/ no continuity correction:
                                                               12.2464
38911801505%
Normal approximation for 1000 rolls w/ no continuity correction:
                                                               0.17882
691051520627%
```

Q3, part 4-3

In [24]:
```python
rolls = [10,100,1000]

print("For 3. What is the exact probablity of rolling the number 1 mor
print()

for roll in rolls:
    print("Normal approximation for " + str(roll) + " rolls w/ continu
    print("

print()

for roll in rolls:
    print("Normal approximation for " + str(roll) + " rolls w/ no cont
    print("
```

```
For 3. What is the exact probablity of rolling the number 1 more than
9% of the time in 10 rolls, 100 rolls, 1000 rolls of a 20-sided die.

Normal approximation for 10 rolls w/ continuity correction:
                                                               9.58005
5337629666%
Normal approximation for 100 rolls w/ continuity correction:
                                                               1.94737
27871012758%
Normal approximation for 1000 rolls w/ continuity correction:
                                                               2.09695
89398234234e-07%

Normal approximation for 10 rolls w/ no continuity correction:
                                                               2.11105
876235812%
Normal approximation for 100 rolls w/ no continuity correction:
                                                               1.08907
31395559738%
Normal approximation for 1000 rolls w/ no continuity correction:
                                                               1.34970
91222447466e-07%
```

Question 4

In [25]:

```
def monteCarlo(f, boxX, boxY, area, numberSamples=10000, showPlot=True
    samplesIn=[]
    samplesOut=[]

    for k in range(numberSamples):
        x = boxX()
        y = boxY()

        if f(x) > y:  # Check to see if point is below curve y=f(x)
            samplesIn.append([x,y]) # point below
        else:
            samplesOut.append([x,y]) # point above

    numIn=len(samplesIn)    # number of points below
    numOut=len(samplesOut)  # number of points above

    ratioIn = numIn/numberSamples # = P(R)
    totalArea = area()


    areaRegion = ratioIn * totalArea

    if (showPlot):
        print("Out of {:,} samples, {:,} are in the blue region under
        print("Hence fraction of the samples below the curve is {:,} \
        print()
        print("Hence our esitmate of the area under the curve is {:}."

        x_samplesIn=[ p[0] for  p in samplesIn]
        y_samplesIn=[ p[1] for  p in samplesIn]

        x_samplesOut=[ p[0] for  p in samplesOut]
        y_samplesOut=[ p[1] for  p in samplesOut]

        plt.scatter(x_samplesOut,y_samplesOut,color='black',s=3)
        plt.scatter(x_samplesIn,y_samplesIn,color='lightblue',s=3)
        plt.xlabel("x")
        plt.ylabel("y")
        plt.show()

    #monteCarlo estimate of region for num samples

    return areaRegion
```

Q4, part 1

My working for this question is below, and the calculations are below

In [26]:

```python
img = Image.open('q4_part1-1.png')
display(img)
img = Image.open('q4_part1-2.png')
display(img)
img = Image.open('q4_part1-3.png')
display(img)
```

1) $\hat{p}_n$ = prob after n independent samples of point from R falling in A.

p = true probability

$A = \{ (x,y) \in R : \cos(x)\sin(2x)+1 \leq y \}$

How big does n need to be to ensure $\mathbb{P}(|\hat{p}_n - p| > 0.05) < 0.01$

$$\mathbb{P}(|\hat{p}_n - p| > 0.05) = \mathbb{P}(-0.05 < \hat{p}_n - p < 0.05)$$

let $\delta = 0.05$

lower bound: $-\delta = \hat{p} - p = \frac{a}{n} - \frac{np}{n} = \frac{a-np}{n}$

$\Rightarrow a = np - n\delta$, $b = np + n\delta$

np = expected successes

$n\hat{p}_n$ = observed successes

a, b represent bounds $\Rightarrow$ on n.

such that $|\hat{p}_n - p| > 0.05$ is true w/ < 0.01 probability

$\delta = \frac{a-np}{n}$ transforms for use on standard normal.

$$\mathbb{P}(|\hat{p}_n - p| > 0.05) = \mathbb{P}(\hat{p}_n - p > 0.05 \text{ or } \hat{p}_n - p < -0.05) < 0.01$$

$$\mathbb{P}(\hat{p}_n - p > 0.05) + \mathbb{P}(\hat{p}_n - p < -0.05) < 0.01$$

$$\Rightarrow 2 \cdot \mathbb{P}(\hat{p}_n - p < -0.05) < 0.01$$

$$z = \frac{x-\mu}{\sigma} = \frac{n\hat{p}_n - np}{\sqrt{npq}} = \frac{n(\hat{p}_n - p)}{\sqrt{npq}}$$

$$= \frac{\sqrt{n}(\hat{p}_n - p)}{\sqrt{pq}} = \frac{\delta\sqrt{n}}{\sqrt{pq}}$$

$$= 2 \cdot \Phi\left(\frac{\delta\sqrt{n}}{\sqrt{pq}}\right) - 0.01 < 0$$

$$\Rightarrow \Phi\left(\frac{\delta\sqrt{n}}{\sqrt{pq}}\right) = 0.005 \qquad \delta = -0.05$$

$$\frac{\delta\sqrt{n}}{\sqrt{pq}} = \Phi^{-1}(0.005)$$

$$\Rightarrow \sqrt{n} = \frac{\sqrt{pq}}{\delta}\Phi^{-1}(0.005)$$

$$\Rightarrow n = \frac{pq}{\delta^2}\left(\Phi^{-1}(0.005)\right)^2$$

As we assume we do not know I, we cannot know true p

$\therefore$ Must take worst case $p = \frac{1}{2}$

From code : 663.47

$\Rightarrow n \geq 664$

Q4, part 2

In [27]:
```
#assume worst case scenario
p = 1/2
q = 1-p

delta = -0.05

confidence = 0.01 ## this is the confidence for which we are trying to

n = ((p*q) / ((delta)** 2)) * (norm.ppf(confidence/2)**2)

print(n)
```

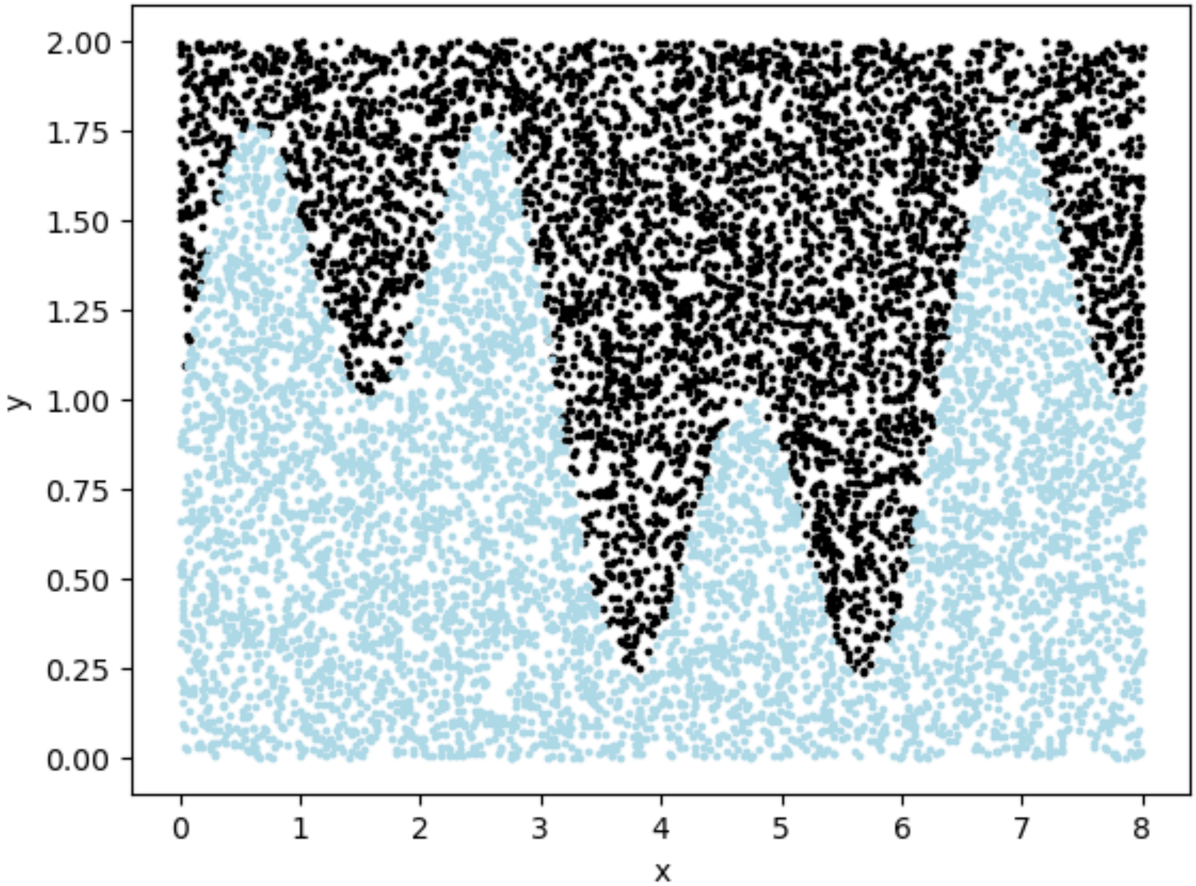
663.4896601021214

In [28]:
```
#function to integrate
def function(x):
    return math.cos( x ) * math.sin( 2*x) + 1

#random x plot value is between 0 and 8
def boxX():
    return (8)*rnd.random()

# random y plot value is between 0 and 2
def boxY():
    return (2)*rnd.random()

# box area = 2*8
def trueArea():
    return 2*8
```

In [29]:

```
monteCarlo(function, boxX, boxY, trueArea)
```

```
Out of 10,000 samples, 5,350 are in the blue region under
         the curve  and 4,650 are above.
Hence fraction of the samples below the curve is 0.535
         and the fraction above is 0.465.

Hence our esitmate of the area under the curve is 8.56.
```

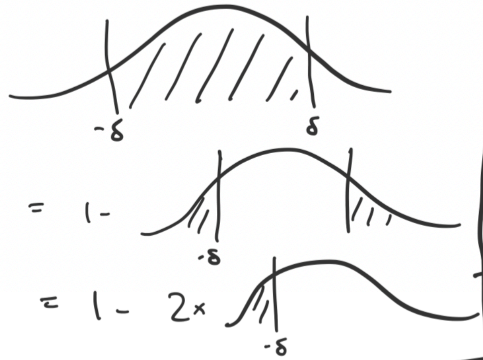


Out[29]: 8.56

In [30]:
```python
img = Image.open('q4_part2-1.png')
display(img)
img = Image.open('q4_part2-2.png')
display(img)
```

2) $P(|\hat{I} - I| \le \delta) > 0.9$

$P(-\delta \le \hat{I} - I \le \delta) > 0.9$



$1 - 2P(\hat{I} - I \le \delta) > 0.9$

$\Rightarrow 2P(\hat{I} - I \le \delta) < 0.1$

$\Rightarrow P(\hat{I} - I \le \delta) < 0.05$

$\Phi(z) < 0.05$

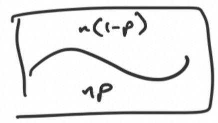
$z = \frac{x - \mu}{\sigma}$ $\quad x = n\hat{p}_n$

$\hat{I} = \left(\frac{n\hat{p}_n}{n}\right)$ boxSize.

$= \hat{p}_n$ boxSize $= 16\hat{p}_n$

$I = \frac{np}{n}$ box Size

$= p \cdot$ boxSize. $= 16p$.



$z = \frac{x - \mu}{\sigma}$

$\Phi(z) = 0.05 \Rightarrow z = \Phi^{-1}(0.05) = -1.6449$

$\therefore \frac{x - \mu}{\sigma} = \frac{n\hat{p}_n - np}{\sqrt{npq}} = \frac{\sqrt{n}\left(\frac{\hat{I} - I}{\text{boxSize}}\right)}{\sqrt{pq}}$

$\Rightarrow \hat{I} - I = \frac{-1.645 \cdot \sqrt{pq}\ \text{boxSize}}{\sqrt{n}} = -0.1861 > \delta$

$\therefore \delta > 0.1861$

In [31]:
```python
((-1.645) * (1/2) * 16)/math.sqrt(5000)
```

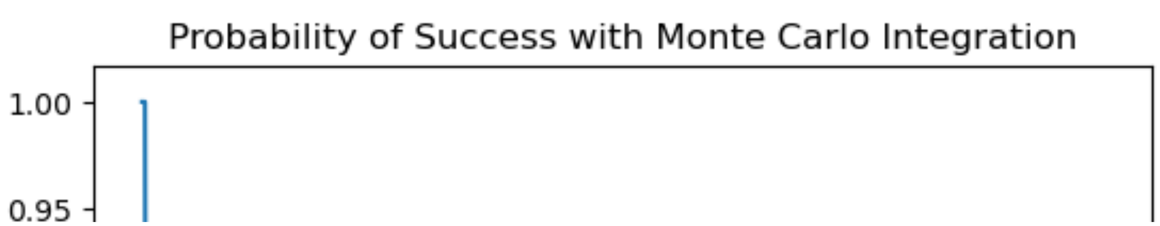
Out[31]: -0.1861105048082993

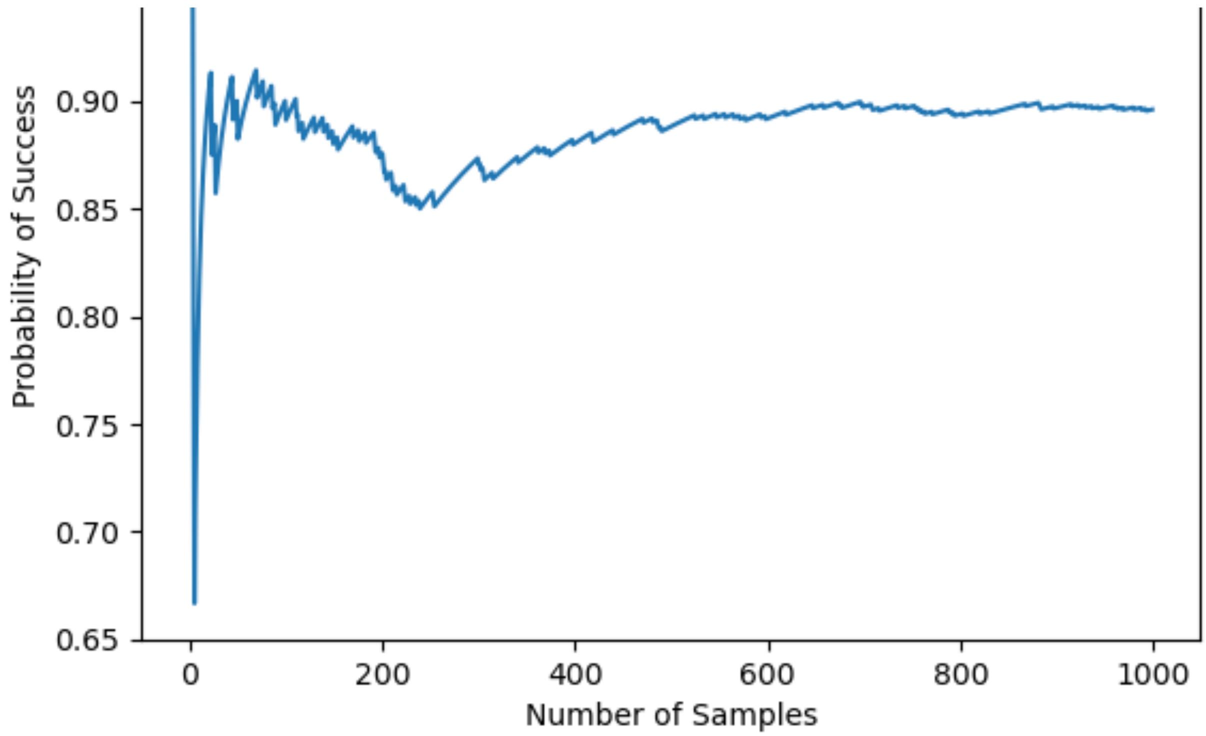
Q4, part 3

In [32]:
```python

```

```
numSamp = 5000
delta = 0.18611

history = []
average_history = []
error = []
running_probability = []
average_running_probability = []
num_success = 0

def function(x):
    return math.cos( x ) * math.sin( 2*x) + 1

#random x plot value is between 0 and 8
def boxX():
    return (8)*rnd.random()

# random y plot value is between 0 and 2
def boxY():
    return (2)*rnd.random()

# box area = 2*8
def trueArea():
    return 2*8

true_integral = 8.6687
print("True integral is ", true_integral)

for m in range(1, 1001):
    history.append(monteCarlo(function, boxX, boxY, trueArea, numberSa

    if abs(history[-1] - true_integral) <= delta:
        num_success += 1
    running_probability.append(num_success/m)
    average_running_probability.append(sum(running_probability)/len(ru

print(f"The probability that the Monte Carlo approximation was within 

plt.plot(running_probability)
plt.xlabel('Number of Samples')
plt.ylabel('Probability of Success')
plt.title('Probability of Success with Monte Carlo Integration')
plt.show()
```

```
True integral is  8.6687
The probability that the Monte Carlo approximation was within 0.18611
of the true integral after 1000 samples is 0.8872
```





Q4, part4

In [33]:
```
img = Image.open('q4_part4.png')
display(img)
```

4) $P(|\hat{I}-I| \le 0.25) > 0.9$

$$1-2+2\Phi(z) > 0.9$$

$$z = \frac{x-\mu}{\sigma} = \frac{n\hat{p}_n - np}{\sigma}$$

$$= \frac{\sqrt{n}\cdot 1/4}{\sqrt{pq}\ \text{boxSize}}$$

$$\therefore \sqrt{n} > \frac{\text{boxSize}\sqrt{pq}}{1/4}\ \Phi^{-1}(0.95)^2$$

$$\therefore n > 2771$$

In [34]:
```python
img = Image.open('q4_part5.png')
display(img)
```

5) $\int_0^8 \cos x \sin 2x + 1 \, dx \approx \underline{\underline{8.6687}}$

Very similar to the approximate answers I got

In [ ]:
```python

```