Sina Mahbobi

February 20, 2020

1 EE511 Project 4

1.0.1 Question 1

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy import random
```

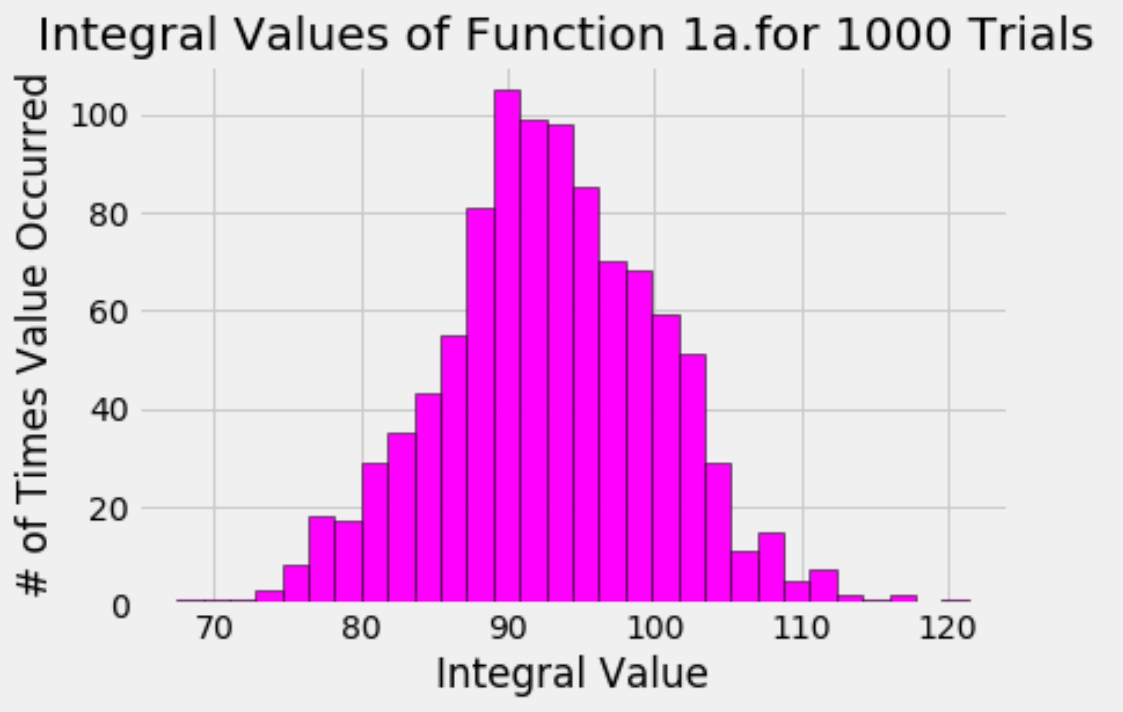
Approximate the following integrals using a Monte Carlo simulation. Compare your estimates with the exact values (if known):

a.

$$\int_{-2}^{2} e^{x+x^2} dx$$

```
In [3]: # 1000 different calculations for 1000 samples using Monte Carlo

        a = -2
        b = 2
        N = 1000
        b_minus_a = b - a
        integrals = []

        def func(x):
            #print(np.exp(x+x*x))
            return np.exp(x+x*x)

        for i in range(0,N):
            x_rand = np.zeros(1000)
            integral_sum = 0
            for j in range(0,N):
                x_rand[j] = random.uniform(a,b)
                integral_sum += func(x_rand[j])
            integral_sum = integral_sum * b_minus_a / N
            integrals.append(integral_sum)
        #bins = np.arange(0,120)-1
        plt.hist(integrals, bins = 30, edgecolor = 'black', facecolor = 'magenta' )
        #plt.xticks(np.arange(70,120))
        plt.xlabel("Integral Value")
```

```
plt.ylabel("# of Times Value Occurred")
plt.title("Integral Values of Function 1a.for 1000 Trials ")
plt.style.use('fivethirtyeight')
plt.show()
```

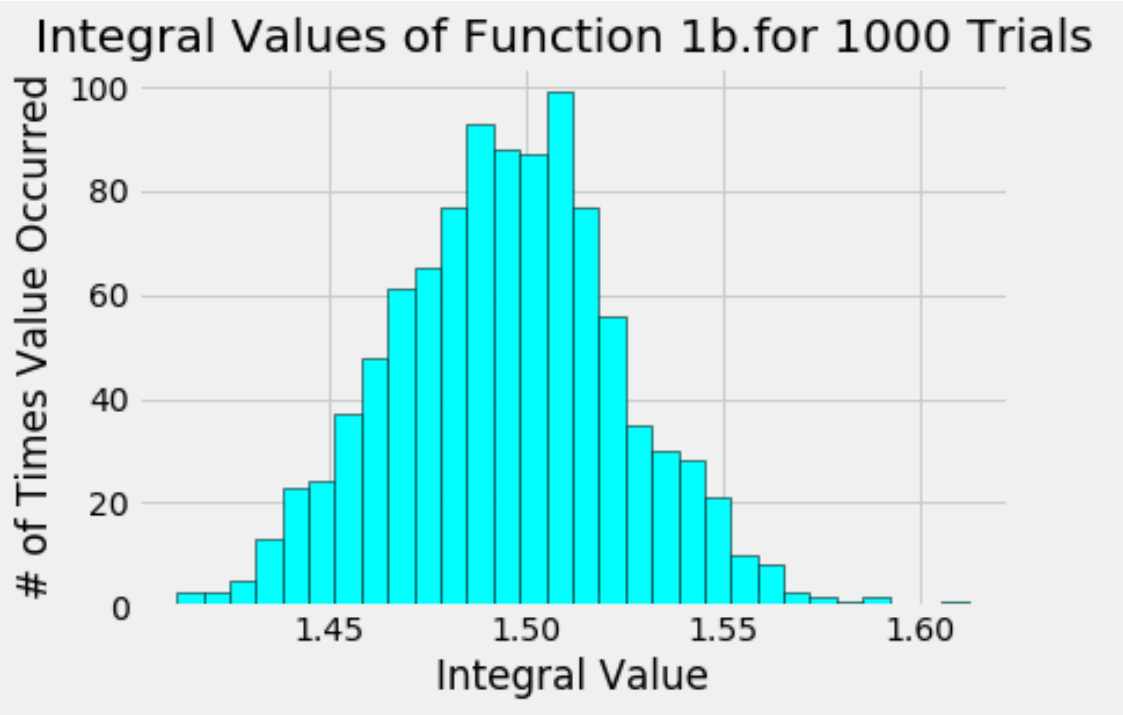


b.

$$\int_{-\infty}^{\infty} e^{-x^2} dx$$

```
In [20]: a = -1* np.pi/2
         b = np.pi/2
         N = 1000
         b_minus_a = b - a
         integrals = []
         #print(b_minus_a)
         #select a function that goes to infinity at finite bounds: tan(x) is a good choice

         def func(x):
             return np.exp(-1*(np.arctan(x)*np.arctan(x))) * np.cos(np.arctan(x)) * np.cos(np.ar



         for i in range(0,1000):
             x_rand = np.zeros(1000)
             integral_sum = 0
```

```
        for j in range(0,N):
            x_rand[j] = random.uniform(a,b)
            integral_sum += func(x_rand[j])
        integral_sum = integral_sum * b_minus_a / N
        integrals.append(integral_sum)
    plt.hist(integrals, bins = 30, edgecolor = 'black', facecolor = 'cyan' )
    #plt.xticks(np.arange(70,120))
    plt.xlabel("Integral Value")
    plt.ylabel("# of Times Value Occurred")
    plt.title("Integral Values of Function 1b.for 1000 Trials ")
    plt.style.use('fivethirtyeight')
    plt.show()
```

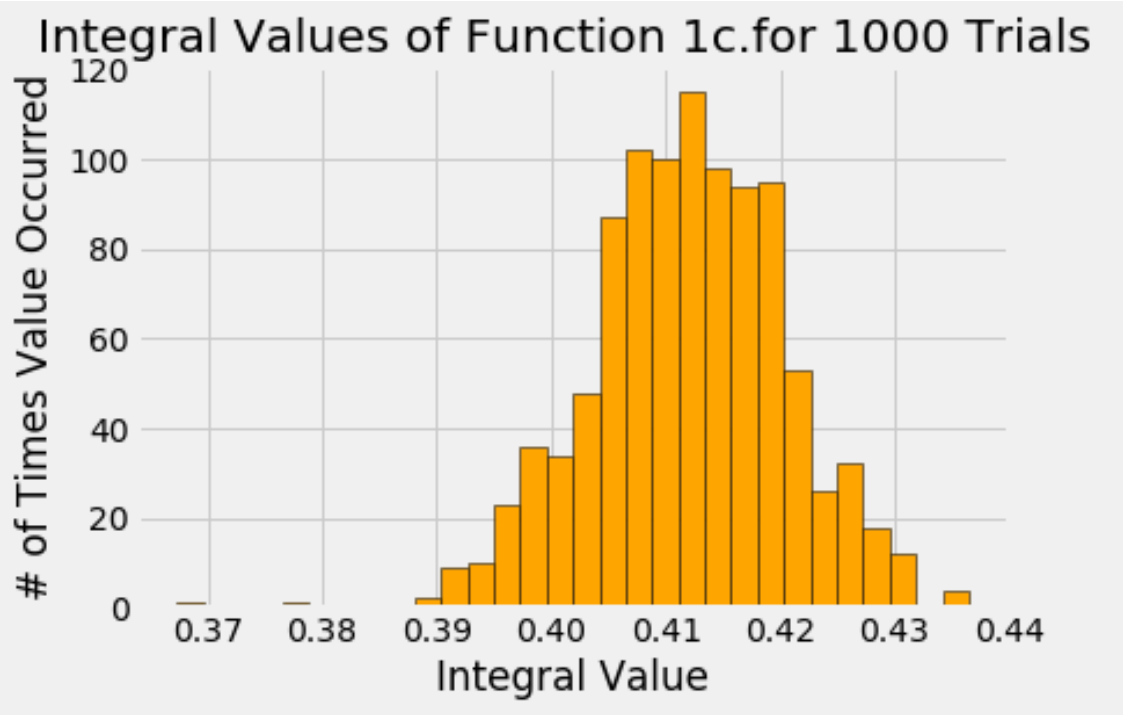


c.

$$\int_0^1 \int_0^1 e^{-(x+y)^2} dy dx$$

In [8]:
```
N = 1000
integrals = []

def func(x,y):
    return np.exp(-1*(x+y)*(x+y))

for i in range(0,N):
    integral_sum = 0
```

```
        x_rand = np.zeros(1000)
        y_rand = np.zeros(1000)
        for j in range(0,N):
            x_rand[j] = np.random.rand()
            y_rand[j] = np.random.rand()
            integral_sum += func(x_rand[j], y_rand[j])
        integral_sum = integral_sum / N
        integrals.append(integral_sum)
    plt.hist(integrals, bins = 30, edgecolor = 'black', facecolor = 'orange' )
    plt.xlabel("Integral Value")
    plt.ylabel("# of Times Value Occurred")
    plt.title("Integral Values of Function 1c.for 1000 Trials ")
    plt.style.use('fivethirtyeight')
    plt.show()
```



Q1 Analysis

- Each of the three histograms above plots the integral values calculated for 1000 different Monte Carlo Trials.
- For the first integral, the correct answer is around 93. By looking at the histogram, it is evident that the value seen the most is near this expected value.
- For the second integral, the function y = tan(x) was substituted for x because this function approaches the value of infinity at finite bounds of pi/2 and -pi/2, allowing an easy substitution for finite integral bounds. The correct value for this integral is the square root of pi,

which is around 1.77. This Monte Carlo estimation using tangent gets close to the value, but is not as accurate as the other estimations in this question. The values we see most often for this estimation range form 1.45 to 1.55 for the 1000 trials.

- For the third integral, we now have two integration bounds: x and y. In this case, we generate two uniform RV's between 0 and 1 and calculate the Monte Carlo estimation. The correct value of this integral is around 0.43, and the value seen most often in the histogram is 0.41.

In [ ]:

1.0.2 Question 2

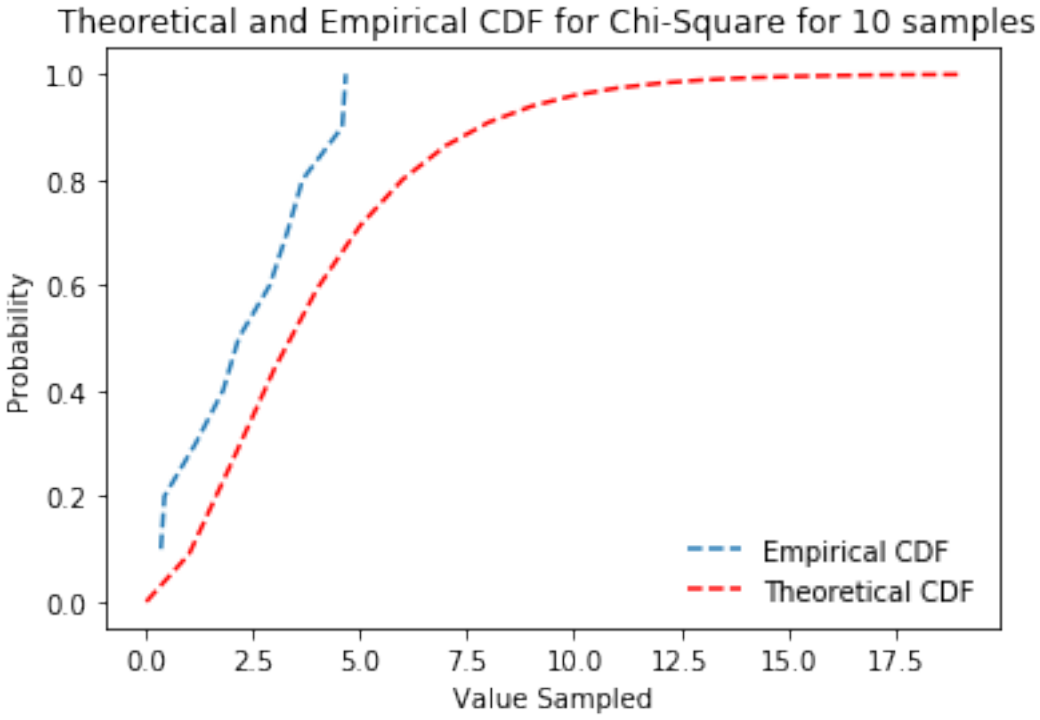
```
In [155]: import numpy as np
          import matplotlib.pyplot as plt
          import scipy.stats as stats
          from sklearn import preprocessing
```

Define the random variable X = Z21 + Z22 + Z23 + Z24 where Zk ~ N(0,1) Then X ~ *X*2(4). Generate 10 samples from X by first sampling Zi for i= 1, 2, 3, 4 and then computing X. Plot the empirical distribution F*10(x) for your samples and overlay the theoretical distribution F(x). Estimate a lower bound for* || *F*10(x) - F(x) || by computing the maximum difference at each of your samples: max || F*10(x) - F(x) || .Then find the 25th, 50th, and 90th percentiles using your empirical distribution and compare the value to the theoretical percentile values for *X*2(4). Repeat the above using 100 and 1000 samples from X

```
In [159]: samples = [10, 100, 1000]
          bins = [5, 15, 30]
          ranges = [20,30,40]

          for i in range (0,len(samples)):
              X = np.zeros(samples[i])
              diff = np.zeros(samples[i])
              for j in range(0,samples[i]):
                  z1 = np.random.normal()
                  z1 = z1 * z1

                  z2 = np.random.normal()
                  z2 = z2 * z2

                  z3 = np.random.normal()
                  z3 = z3 * z3

                  z4 = np.random.normal()
                  z4 = z4 * z4

                  x = z1 + z2 + z3 + z4

                  X[j] = x

              #for k in range(1,samples[i]+1):
```

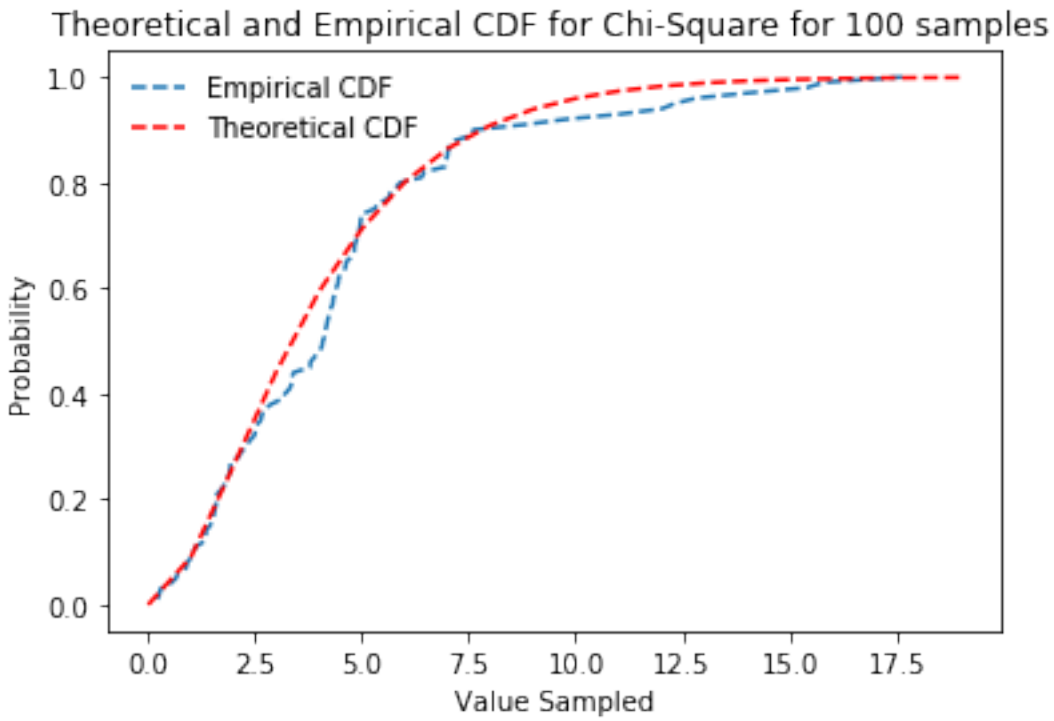
```
#diff[k-1] = abs((X[k-1]/X.max() - stats.chi2.cdf(X[k-1], df = 4))

#print(diff.tolist())
#x_max = X.max()
X.sort()
for k in range(0,samples[i]):
    samp = (k+1)/samples[i]
    diff[k] = abs(samp - stats.chi2.cdf(X[k], df = 4))
    #print((X[k]/x_max))
    #print(stats.chi2.cdf(X[k], df = 4))
    #print(X[k])
    #print(diff[X[k]])
#print(diff)

lower_bound = diff.max()

Y = np.arange(1,len(X)+1)/len(X)
plt.plot(X,Y, linestyle='dashed', label = 'Empirical CDF')

x = np.arange(0,20)
y = stats.chi2.cdf(x, df = 4)



plt.plot(x,y, ms=8, label='Theoretical CDF', color = 'red', linestyle = 'dashed')
plt.legend(loc='best', frameon=False)
plt.xlabel("Value Sampled")
plt.ylabel("Probability")
plt.title('Theoretical and Empirical CDF for Chi-Square '
'for {samples} samples '.format(samples=samples[i]))
plt.show()

#print(X)

quant_25_emp = np.quantile(X, .25)
quant_50_emp = np.quantile(X, .50)
quant_90_emp = np.quantile(X, .90)

quant_25_theo = stats.chi2.ppf(.25, df = 4)
quant_50_theo = stats.chi2.ppf(.50, df = 4)
quant_90_theo = stats.chi2.ppf(.90, df = 4)


print('The empirical 25th percentile for {samples} '
      'samples: '.format(samples=samples[i]), str(quant_25_emp))
print('The theoretical 25th percentile for {samples} '
      'samples: '.format(samples=samples[i]), str(quant_25_theo))
print('\n')
```

```
print('The empirical 50th percentile for {samples} '
      'samples: '.format(samples=samples[i]), str(quant_50_emp))
print('The theoretical 50th percentile for {samples} '
      'samples: '.format(samples=samples[i]), str(quant_50_theo))
print('\n')

print('The empirical 90th percentile for {samples} '
      'samples: '.format(samples=samples[i]), str(quant_90_emp))
print('The theoretical 90th percentile for {samples} '
      'samples: '.format(samples=samples[i]), str(quant_90_theo))
print('\n')

print('The lower bound for {samples} '
      'samples is: '.format(samples = samples[i]),str(lower_bound))
```

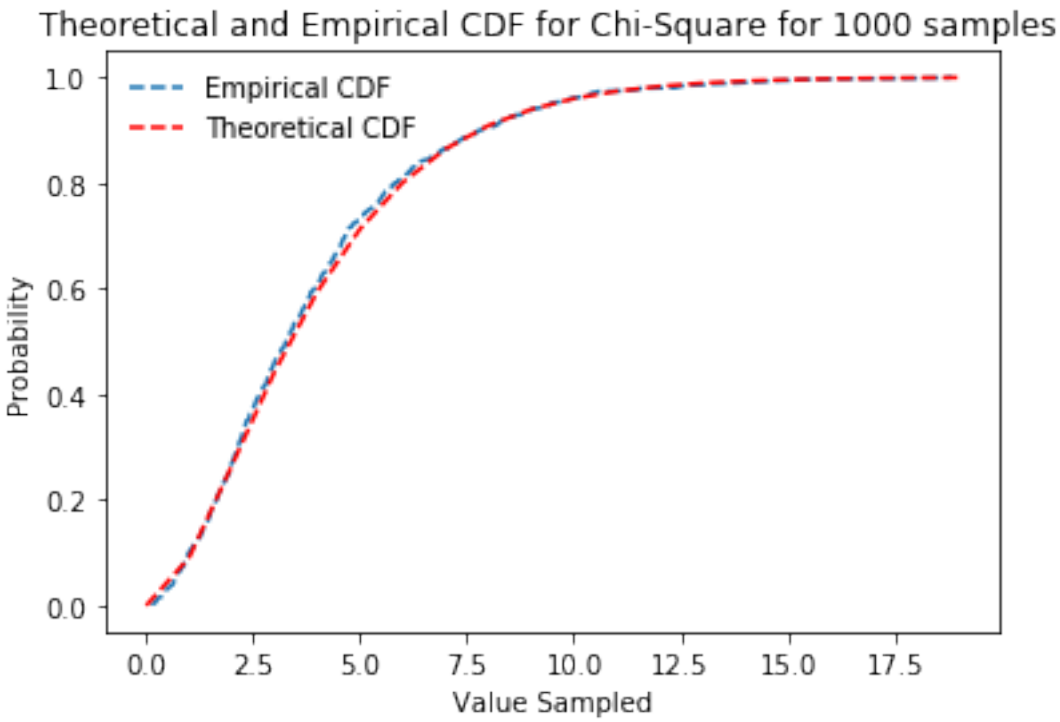



```
The empirical 25th percentile for 10 samples:  1.3143259886088627
The theoretical 25th percentile for 10 samples:  1.922557526229554


The empirical 50th percentile for 10 samples:  2.524952633145161
The theoretical 50th percentile for 10 samples:  3.3566939800333224
```

```
The empirical 90th percentile for 10 samples:  4.590206794311635
The theoretical 90th percentile for 10 samples:  7.779440339734858


The lower bound for 10 samples is:  0.3236556079311699
```




```
The empirical 25th percentile for 100 samples:  1.9164360873410389
The theoretical 25th percentile for 100 samples:  1.922557526229554


The empirical 50th percentile for 100 samples:  4.112472930068304
The theoretical 50th percentile for 100 samples:  3.3566939800333224


The empirical 90th percentile for 100 samples:  7.729292244815722
The theoretical 90th percentile for 100 samples:  7.779440339734858


The lower bound for 100 samples is:  0.11527913978899829
```




```
The empirical 25th percentile for 1000 samples:  1.9351940424756182
The theoretical 25th percentile for 1000 samples:  1.922557526229554


The empirical 50th percentile for 1000 samples:  3.2716323505773337
The theoretical 50th percentile for 1000 samples:  3.3566939800333224


The empirical 90th percentile for 1000 samples:  7.842871799399977
The theoretical 90th percentile for 1000 samples:  7.779440339734858


The lower bound for 1000 samples is:  0.029698521483102258
```

Q2 Analysis

- The three plots above are for each sample value: 10, 100 and 1000.
- As the number of samples increases, the empirical CDF begins to resemble the theoretical CDF more and more.
- The 25th, 50th, and 90 percentiles for the empirical values also approach the theoretical values as the samples increase.

- As the number of samples increases, the lower bound also gets closer and closer to zero by the Glivenko-Cantelli theorem, which states that as the number of samples approaches infinity the difference between the empirical and theoretical CDF would equal 0.

In [ ]:

1.0.3 Question 3

A geyser is a hot spring characterized by an intermittent discharge of water and steam. Old Faithful is a famous cone geyser in Yellowstone National Park, Wyoming. It has a predictable geothermal discharge and since 2000 it has erupted every 44 to 125 minutes. Refer to the addendum data file that contains waiting times and the durations for 272 eruptions. Compute a 95% statistical confidence interval for the mean waiting time using data from only the first 15 eruptions. Compare this to a 95% bootstrap confidence interval using the same 15 data samples. Repeat these calculations using all the data samples. Comment on the relative width of the confidence intervals when using only 15 samples vs using all samples.

In [198]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats
import math
import random
```

In [199]:
```
f = open('faithful.dat.txt', "r")
lines = f.readlines()[25:]
```

In [200]:
```
df = pd.DataFrame(lines)
df.replace(r'\s+|\\n', ' ', regex=True, inplace=True)
df.columns = df.iloc[0]
#print(df.columns)
df = df.reindex(df.index.drop(0)).reset_index(drop=True)
df = df[' eruptions waiting '].str.split(' ', 1, expand=True)
df= df.drop(df.columns[0], axis=1)
df = df.rename(columns={1: "Waiting"})
df = df['Waiting'].str.split(' ', 1, expand=True)
df = df.rename(columns={0:"Eruptions", 1: "Waiting"})
df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)

# convert column "a" of a DataFrame
df["Waiting"] = pd.to_numeric(df["Waiting"])
```

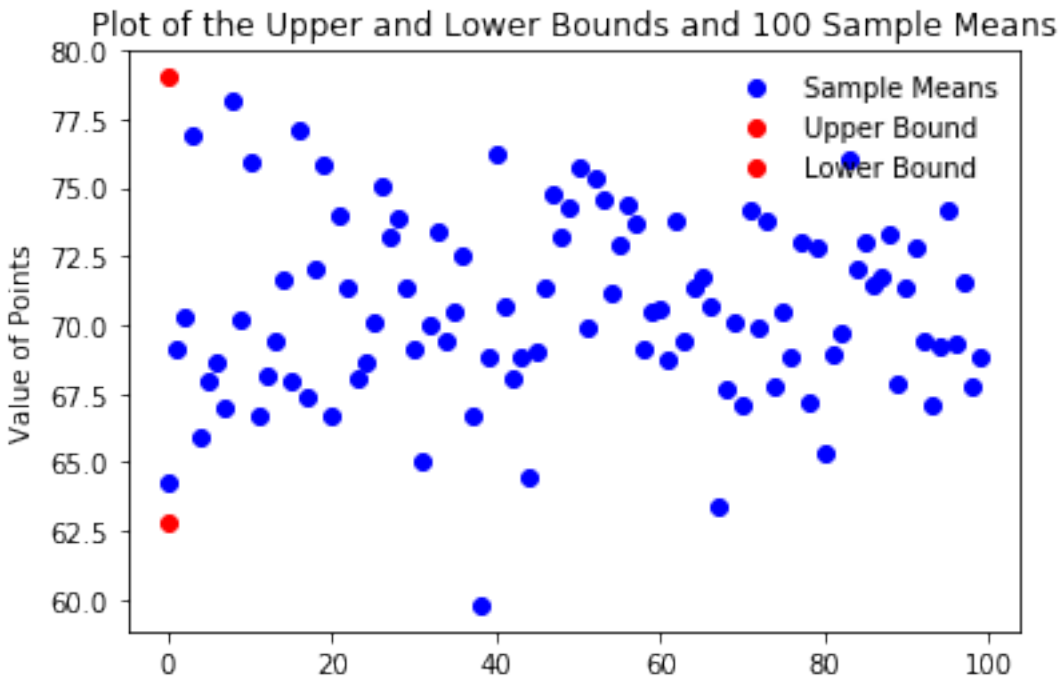
In [201]:
```
samples_15 = df[0:15]
```

In [270]:
```
waiting_samples_15 = samples_15.iloc[:,1].values

waiting_mean_15 = np.mean(waiting_samples_15)
waiting_std_dev_15 = np.std(waiting_samples_15)
```

```
        #print(waiting_std_dev_15)
        #print(waiting_mean_15)

        confidence_upper = waiting_mean_15 + stats.t.ppf(1-0.025, df = 14) * waiting_std_dev_1
        confidence_lower = waiting_mean_15 - stats.t.ppf(1-0.025, df = 14) * waiting_std_dev_1
        confidence_interval = confidence_upper - confidence_lower

        print('The upper bound for'
              ' 15 samples is ', str(confidence_upper))
        print('The lower bound'
              ' for 15 samples is ', str(confidence_lower))
        print("The interval width is ", str(confidence_interval))

        sample_means = []
        s = []
        for i in range(0,100):
            s = random.choices(waiting_samples_15,k = 15)
            s_m = sum(s)/len(s)
            #print(s_m)
            sample_means.append(s_m)


        plt.plot(sample_means, "bo", label = 'Sample Means')
        plt.plot(confidence_upper, "ro", label = 'Upper Bound')
        plt.plot(confidence_lower, "ro", label = 'Lower Bound')
        plt.legend(loc='upper right', frameon=False)
        plt.ylabel('Value of Points')
        plt.title('Plot of the Upper and Lower Bounds and 100 Sample Means')
```

```
The upper bound for 15 samples is  79.02553683853083
The lower bound for 15 samples is  62.841129828135855
The interval width is  16.18440701039497
```

```
Out[270]: Text(0.5, 1.0, 'Plot of the Upper and Lower Bounds and 100 Sample Means')
```



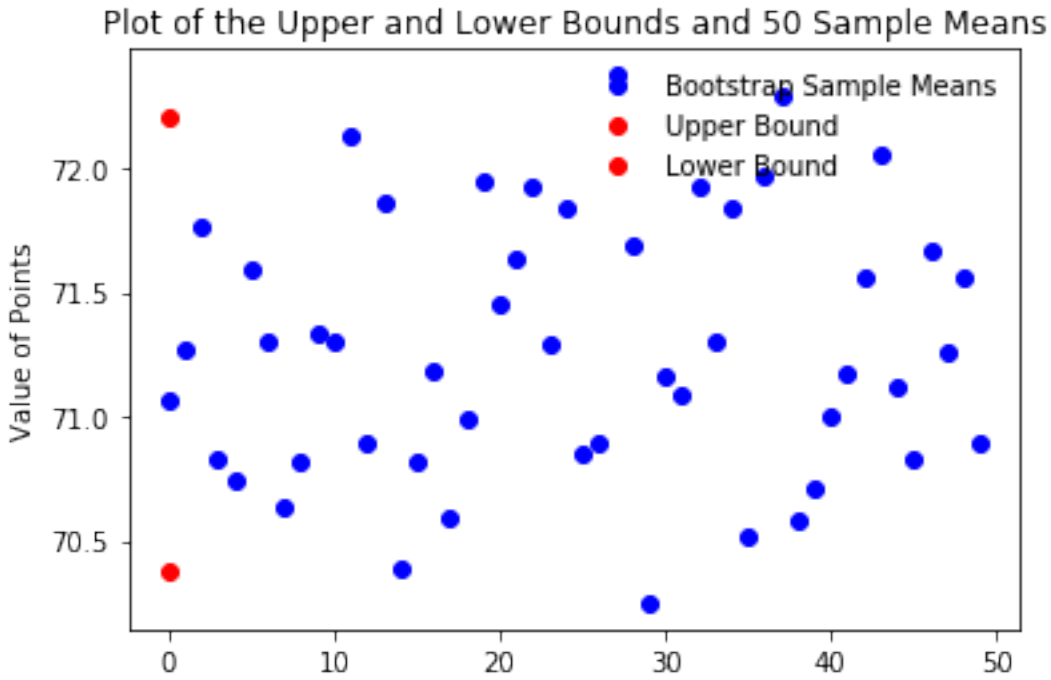
- As you can see in the graph above, of the 100 sample means by sampling from the 15 data points with replacement, only a few of them fall outside the upper and lower bounds. This is empirical evidence of the correctness of the 95% confidence interval.

In [274]:
```
# 1000 sample bootstrap of original 15 samples

bootstrap_samples = random.choices(waiting_samples_15,k = 1000)
bootstrap_mean = np.mean(bootstrap_samples)
bootstrap_std_dev = np.std(bootstrap_samples)
#print(bootstrap_samples)

confidence_upper = bootstrap_mean + stats.t.ppf(1-0.025, df = 999) * bootstrap_std_dev
confidence_lower = bootstrap_mean - stats.t.ppf(1-0.025, df = 999) * bootstrap_std_dev
confidence_interval = confidence_upper - confidence_lower

print('The upper bound for'
      ' 1000 bootstrap samples from 15 samples is ', str(confidence_upper))
print('The lower bound'
      ' for 1000 bootstrap samples from 15 samples is ', str(confidence_lower))
print("The interval width is ", str(confidence_interval))


sample_means = []
s = []
for i in range(0,50):
```

```
        s = random.choices(bootstrap_samples,k = 1000)
        s_m = sum(s)/len(s)
        #print(s_m)
        sample_means.append(s_m)


    plt.plot(sample_means, "bo", label = 'Bootstrap Sample Means')
    plt.plot(confidence_upper, "ro", label = 'Upper Bound')
    plt.plot(confidence_lower, "ro", label = 'Lower Bound')
    plt.legend(loc='upper right', frameon=False)
    plt.ylabel('Value of Points')
    plt.title('Plot of the Upper and Lower Bounds and 50 Sample Means')
```

```
The upper bound for 1000 bootstrap samples from 15 samples is  72.20116941494832
The lower bound for 1000 bootstrap samples from 15 samples is  70.38283058505168
The interval width is  1.8183388298966463
```

```
Out[274]: Text(0.5, 1.0, 'Plot of the Upper and Lower Bounds and 50 Sample Means')
```



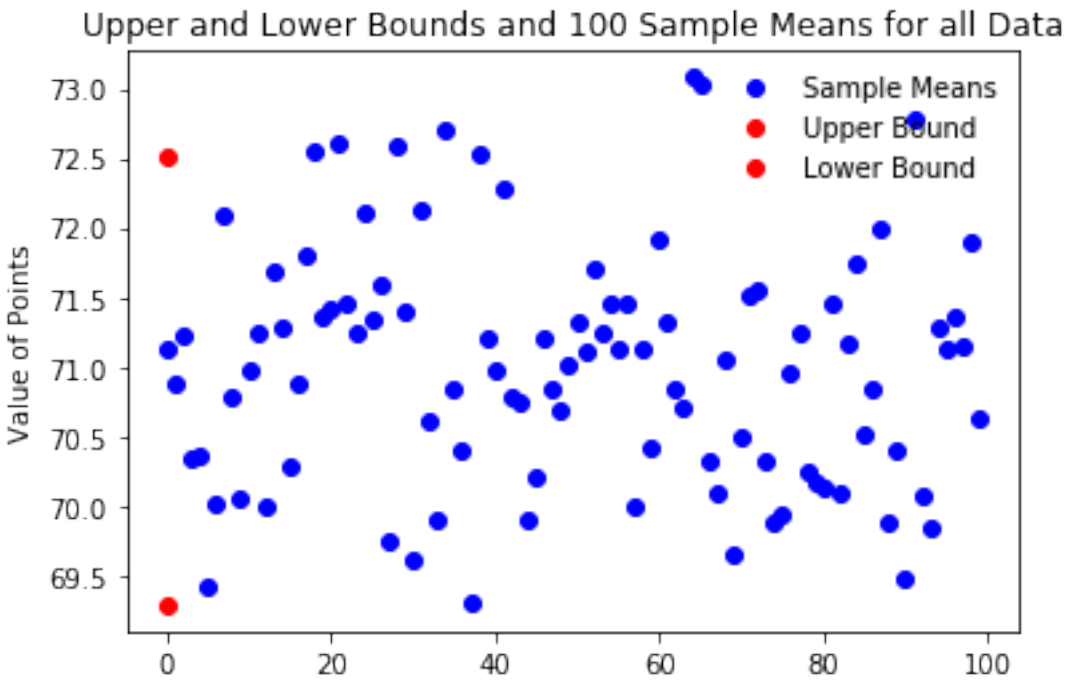
- Looking at the bootstrap confidence interval for 1000 samples, it is significantly lower than the confidence interval using only the original 15 samples.
- The confidence interval width using bootstrapping is less than 2, while the interval width using only the fifteen samples is around 16. This bootstrapping method allows for much greater certainty in a much smaller range, resulting in more accurate measurements.

- All but a few of the sample means recalculated from the bootstrapping method fall outside of the 95% confidence bound, which is empirical evidence for this bound being calculated correctly.

In [277]:
```
all_samples = df.iloc[:,1].values

all_mean = np.mean(all_samples)
all_std_dev = np.std(all_samples)

confidence_upper = all_mean + stats.norm.ppf(1-0.025) * all_std_dev / math.sqrt(272)
confidence_lower = all_mean - stats.norm.ppf(1-0.025) * all_std_dev / math.sqrt(272)
confidence_interval = confidence_upper - confidence_lower

print('The upper bound for'
      ' all samples is ', str(confidence_upper))
print('The lower bound '
      ' for all samples is ', str(confidence_lower))
print("The interval width is ", str(confidence_interval))

sample_means = []
s = []
for i in range(0,100):
    s = random.choices(all_samples,k = 272)
    s_m = sum(s)/len(s)
    #print(s_m)
    sample_means.append(s_m)


plt.plot(sample_means, "bo", label = 'Sample Means')
plt.plot(confidence_upper, "ro", label = 'Upper Bound')
plt.plot(confidence_lower, "ro", label = 'Lower Bound')
plt.legend(loc='upper right', frameon=False)
plt.ylabel('Value of Points')
plt.title('Upper and Lower Bounds and 100 Sample Means for'
          ' all Data')
```

```
The upper bound for all samples is  72.5097165699662
The lower bound  for all samples is  69.28440107709261
The interval width is  3.225315492873591
```

Out[277]: Text(0.5, 1.0, 'Upper and Lower Bounds and 100 Sample Means for all Data')



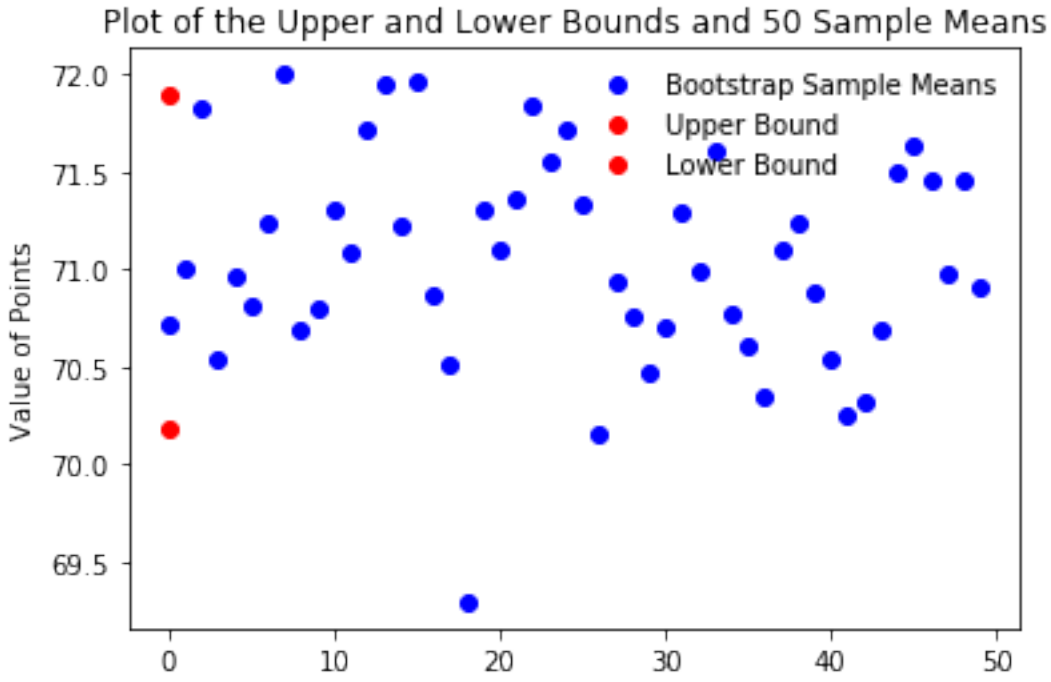
- Looking at the graph above, which plots the statistical 95% confidence interval using all the data, the interval width is much smaller than when only 15 samples of the data were used.
- Of the 100 resampled means from the total data, only a few fall outside the range of the confidence interval (the red points), which is empirical evidence for a properly calculated 95% confidence interval.

In [276]:
```
# 1000 sample bootstrap of all data

bootstrap_samples = random.choices(all_samples,k = 1000)
bootstrap_mean = np.mean(bootstrap_samples)
bootstrap_std_dev = np.std(bootstrap_samples)
#print(bootstrap_samples)

confidence_upper = bootstrap_mean + stats.norm.ppf(1-0.025) * bootstrap_std_dev / math
confidence_lower = bootstrap_mean - stats.norm.ppf(1-0.025) * bootstrap_std_dev / math
confidence_interval = confidence_upper - confidence_lower

print('The upper bound for'
      ' bootstrap samples is ', str(confidence_upper))
print('The lower bound'
      ' for bootstrap samples is ', str(confidence_lower))
print("The interval width is ", str(confidence_interval))


sample_means = []
```

```
s = []
for i in range(0,50):
    s = random.choices(bootstrap_samples,k = 1000)
    s_m = sum(s)/len(s)
    #print(s_m)
    sample_means.append(s_m)


plt.plot(sample_means, "bo", label = 'Bootstrap Sample Means')
plt.plot(confidence_upper, "ro", label = 'Upper Bound')
plt.plot(confidence_lower, "ro", label = 'Lower Bound')
plt.legend(loc='upper right', frameon=False)
plt.ylabel('Value of Points')
plt.title('Plot of the Upper and Lower Bounds and 50 Sample Means')
```

```
The upper bound for bootstrap samples is  71.89140283719402
The lower bound for bootstrap samples is  70.17859716280597
The interval width is  1.7128056743880506
```

```
Out[276]: Text(0.5, 1.0, 'Plot of the Upper and Lower Bounds and 50 Sample Means')
```



- Using the bootstrapping method for all the data points results in the lowest interval width of all the different graphs.
- Here the interval width is around 1.7. This similar but still less than bootstrapping with only 15 samples, as opposed to 1000 samples from all the data in this case.

- From all this analysis, we can conclude that the more samples one has, the smaller the range of the 95% confidence interval. Bootstrapping with a high number of samples also results in a shrunken interval.

In [ ]: