CS 237 Fall 2019 Homework Seven Solution

Due date: PDF file due Friday October 25th @ 11:59PM in GradeScope with 12-hour grace period

Late deadline: If submitted up to 24 hours late, you will receive a 10% penalty (with same 6 hours grace period)

General Instructions

Please complete this notebook by filling in solutions where indicated. Be sure to "Run All" from the Cell menu before submitting.

There are two sections to the homework: problems 1 - 8 are analytical problems about last week's material, and the remaining problems are coding problems which will be discussed in lab next week.

In [1]:
```
# Useful imports and definitions for CS 237

import numpy as np                  # arrays and functions which operate on array
from numpy import linspace, arange, mean
import matplotlib.pyplot as plt    # normal plotting
#import seaborn as sns              # Fancy plotting
#import pandas as pd                # Data input and manipulation

from random import random, randint, uniform, choice, sample, shuffle, seed
from collections import Counter
from math import log,floor,ceil,exp

%matplotlib inline

# Calculating permutations and combinations efficiently

def P(N,K):
    res = 1
    for i in range(K):
        res *= N
        N = N - 1
    return res

def C(N,K):
    if(K < N/2):
        K = N-K
    X = [1]*(K+1)
    for row in range(1,N-K+1):
        X[row] *= 2
        for col in range(row+1,K+1):
            X[col] = X[col]+X[col-1]
    return X[K]

def round4(x):
    return round(x+0.00000000001,4)

def round4_list(L):
    return [ round4(x) for x in L]

# Useful code from HW 01



# This function takes a list of outcomes and a list of probabilities and
# draws a chart of the probability distribution.
# It allows labels for x axis with numbers or strings; for the latter, you
# still need to give the numeric labels, but can overwrite them with your string labels.

def draw_distribution(Rx, fx, title='Probability Distribution', my_xticks = [], f_size = (8,6
)):
    plt.figure(figsize=f_size)
    plt.bar(Rx,fx,width=1.0,edgecolor='black')
    plt.ylabel("Probability")
    plt.xlabel("Outcomes")
    if my_xticks != []:
        plt.xticks(Rx, my_xticks)
    elif (Rx[-1] - Rx[0] < 30):
        ticks = range(Rx[0],Rx[-1]+1)
        plt.xticks(ticks, ticks)
    plt.title(title)
    plt.show()

# Example of use

#  draw_distribution([1,2,3,4], [0.25,0.35,0.15,0.25])

#  p = 0.14159234368

#  draw_distribution( [0,1], [p,1.0-p],"Distribution for Unfair Coin", ['Heads','Tails'],(5,4))

# This draws a useful bar chart for the distribution of the
# list of integer in outcomes

def show_distribution(outcomes, title='Probability Distribution', my_xticks = [], f_size = (8,6
)):
    plt.figure(figsize=f_size)
    num_trials = len(outcomes)
    X = range( int(min(outcomes)), int(max(outcomes))+1 )
    freqs = Counter(outcomes)
    Y = [freqs[i]/num_trials for i in X]
    plt.bar(X,Y,width=1.0,edgecolor='black')
    if my_xticks != []:
        plt.xticks(X, my_xticks)
    elif (X[-1] - X[0] < 30):
        ticks = range(X[0],X[-1]+1)
        plt.xticks(ticks, ticks)
    plt.xlabel("Outcomes")
    plt.ylabel("Probability")
    plt.title(title)
    plt.show()

# Example of use

#show_distribution([1,4,3,5,4,6,2,4,3,5,4])


# Scipy statistical functions

from scipy.stats import norm, binom

# https://docs.scipy.org/doc/scipy/reference/stats.html

#### Normal Distribution    #####

# Note that in this library loc = mean and scale = standard deviation

# Examples assume random variable X (e.g., housing prices) normally distributed with  mu = 60,
 sigma = 10

# Probability Density Function
#  f(x) = P(X == x)

norm.pdf(x=50,loc=60, scale= 10)

# Cumulative Density Function
#  F(x) = P(X < x)

norm.cdf(x=50,loc=60,scale=10) # 0.4012936743170763

# Example:  Find P(60<X<80)
norm.cdf(x=80,loc=60,scale=40) - norm.cdf(x=60,loc=60,scale=40)

# Percentage Point Function: Inverse of the CDF:
# For which value of x does P( X < x ) = q  ?

# Example: What is the maximum cost of the 5% cheapest houses, i.e., for which x does P(X < x)
 = 0.05?
norm.ppf(q=0.05,loc=60,scale=40)

#g. generate a random variate
norm.rvs(loc=60, scale=40)

#h. generate random variates, returns list of length = size
norm.rvs(loc=60, scale=40, size=10)


##### Binomial Distribution  X ~ B(n,p)  ####

#  n = number of independent Bernoulli trials
#  p = probability of success for Bernoulli trial
#  k = outcome in range [0 .. n]
# Generate a random variate
binom.rvs(n=10, p=0.5)

# Generate a list of random variates
binom.rvs(n=10, p=0.5,size=100)

# Probability mass/density function.
binom.pmf(k=4, n=10, p=0.5)

# Cumulative distribution function
binom.cdf(k=4, n=10, p=0.5)


print()
```

Problem One (Poisson)

This problem is a collection of several different applications of the basic formulae for the Poisson Distribution. You **must** use the Poisson and show all work.

(a) Assume that the probability that a poker hand is a full house is 0.0014. What is the probability that in 500 random poker hands there are at least two full houses?

(b) On average, there are three misprints in every 10 pages of a particular book. If every chapter of the book contains 35 pages, what is the probability that Chapters 1 and 5 have exactly 10 misprints each?

(c) Suppose that on a summer evening, shooting stars are observed at a Poisson rate of one every 12 minutes. What is the probability that three shooting stars are observed in 30 minutes?

(d) Suppose that in Japan earthquakes occur at a Poisson rate of three per week. What is the probability that the next earthquake occurs after two weeks?

(e) Suppose that, for a telephone subscriber, the number of wrong numbers is Poisson, at a rate of + = 1 per week. A certain subscriber has not received any wrong numbers from Sunday through Friday. What is the probability that he receives no wrong numbers on Saturday either?

Hint: Remember that the Poisson Distribution does not have the memory-less property, but each arrival is *independent* of every other arrival, and in particular, two intervals of time or space which do not overlap have an independent number of arrivals; so, like coins, arrivals don't have memory...

Solution

(a)

$\lambda = (500)(0.0014) = 0.7$. The answer is $1 - \dfrac{e^{-0.7}(0.7)^0}{0!} - \dfrac{e^{-0.7}(0.7)^1}{1!} \approx 0.156$.

(b)

$\lambda = (3/10)35 = 10.5$. The probability of 10 misprints in a given chapter is $\dfrac{e^{-10.5}(10.5)^{10}}{10!} = 0.124$. Therefore, the desired probability is $(0.124)^2 = 0.0154$.

(c)

Let $N(t)$ be the number of shooting stars observed up to time t. Let one minute be the unit of time. Then $\{N(t) : t \geq 0\}$ is a Poisson process with $\lambda = 1/12$. We have that

$$P\big(N(30) = 3\big) = \frac{e^{-30/12}(30/12)^3}{3!} = 0.21.$$

(d)

$$P\big(N(2) = 0\big) = e^{-3(2)} = e^{-6} = 0.00248.$$

(e)

Let $N(t)$ be the number of wrong calls up to t. If one day is taken as the time unit, it is reasonable to assume that $\{N(t) : t \geq 0\}$ is a Poisson process with $\lambda = 1/7$. By the independent increment property and stationarity, the desired probability is

$$P\big(N(1) = 0\big) = e^{-(1/7)\cdot 1} = 0.87.$$

Problem Two (Poisson compared with Exponential)

Suppose the emergency room at Mass General opens at 6am and has a mean arrival rate throughout the day of 6.9 patients per hour (that is λ = 6.9). </p>

You must use the Poisson for (a) - (c) and Exponential for (d) and (e). Give the name of the distribution and show the parameters (for example, (a) is P(X=12) for X ~ Poi(6.9)), and show all work.

(a) What is the probability that exactly 12 patients arrive between 6am and 7am? (Use Poisson)

(b) What is the probability that no patient arrives before 7am? (Use Poisson)

(c) What is the probability that a patient arrives between 6:15 and 6:45? (Use Poisson)

(d) What is the probability that the first patient arrives between 6am and 7am? (Use Exponential)

(e) What is the probability that no patient arrives between 6:15 and 6:45? (Use Exponential)

Hint: Some of these are complements of each other...

Solution: (a) This is just P(X=12) for Poi(6.9), so

$$\frac{6.9^{12} \cdot e^{-6.9}}{12!} = 0.0245$$

(b) This is just P(X=0) for Poi(6.9), so

$$\frac{6.9^{0} \cdot e^{-6.9}}{0!} = 0.001$$

(c) This means the number of patients arriving in this half hour is non-zero, so $1 - P(X = 0)$ for $Poi(3.45)$, or

$$1.0 - \frac{3.45^{0} \cdot e^{-3.45}}{0!} = 0.9683.$$

(d) This is just a question about the probability of the interarrival time (which is given by the Poisson distribution) being less than one hour, so for $X \sim E(6.9)$ we have

$$P(X < x) = 1.0 - e^{-\lambda \cdot x} = 1.0 - e^{-6.9} = 0.999.$$

Note that this is in the complement of (b)!

(e) This is just a question about whether the next arrival (from the point of view of 6:15am) is more than a half hour away, so for $X \sim E(6.9)$, we have

$$P(X > 0.5) = e^{-3.45} = 0.0317.$$

Note that this is the complement of (c).

Problem Three (Exponential)

In the following assume that we are dealing with Poisson processes and can use the Exponential distribution.

(a) Suppose that every three months , on average, an earthquake occurs in California. What is the probability that the next earthquake occurs after three but before seven months?

(b) Suppose we model time to failure of TV tubes as an Exponential random variable and that tubes fail on average after 10 years. If Jim bought his TV set 10 years ago, what is the probability that its tube will last another 10 years?

(c) A guest arrives at a hotel, on average, every 10 minutes. Suppose that for the last 10 minutes no guest has arrived. What is the probability that the next one will arrive in less than 2 minutes?

(d) Considering the same situation as in (c), what is the probability that from the arrival of the tenth to the arrival of the eleventh guest takes no more than 2 minutes?

Hint: Make sure in you understand the difference between the rate parameter λ (= mean number of arrivals per unit time) and $\beta = 1/\lambda$ (= mean interarrival time).

Solution:

(a) Let X be the time (in months) until the next earthquake; then X is an exponential random variable with $\lambda = 1/3$ in units of earthquakes per month. To calculate $P(3 < X < 7)$, note that since F_X, the distribution function of X, is given by

$$F(t) \;=\; P(X \leq t) \;=\; 1 - e^{-t/3} \text{ for } t > 0,$$

we can write

$$P(3 < X < 7) \;=\; F(7) - F(3) \;=\; (1 - e^{-7/3}) - (1 - e^{-1}) \;=\; 0.2709.$$

(b) Let X be the lifetime of the tube. By the memoryless property, the assumption is that there is no deterioration with age of the tube. Hence

$$P(X > 20 \,|\, X > 10) \;=\; P(X > 10) \;=\; 1 - \left(1 - e^{(-1/10)10}\right) \;=\; e^{-1} \;=\; 0.3679.$$

(c) Suppose the next customer arrives in X minutes. By the memory-less property, we have

$$P(x \leq \frac{1}{30}) = 1.0 - e^{-5\cdot(1/30)} = 0.1535.$$

(d) By the memory-less property, the answer to (d) is the same as the answer to (c)!

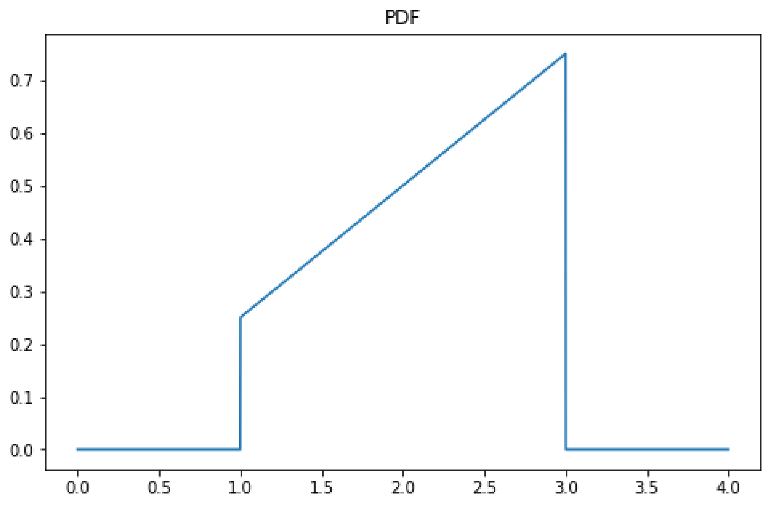
Problem Four

(Continuous Distributions) Let X be a continuous random variable with a frequency distribution (PMF) of the form

$$f(x) = \begin{cases} \frac{x}{4} & \text{if } 1 \leq x \leq 3 \\ 0 & \text{otherwise} \end{cases}$$

which can be graphed as follows:

```
In [2]: plt.figure(figsize=(8, 5))
        plt.title("PDF")
        plt.plot(np.arange(0,4,0.001),[x/4 if 1 <= x <= 3 else 0 for x in np.arange(0,4,0.001)])
        plt.show()
```



(a) Determine the formula for the CDF F_X using geometrical techniques (i.e., not using integrals, but considering what happens to the area to the left of a point a by considering the area of geometrical shapes)

(b) Determine the formula for the CDF $F_X(x)$ using an integral.

(c) Plot the CDF $F_X(x)$ (using the code above as a model).

(d) Find $P(X \geq 2)$

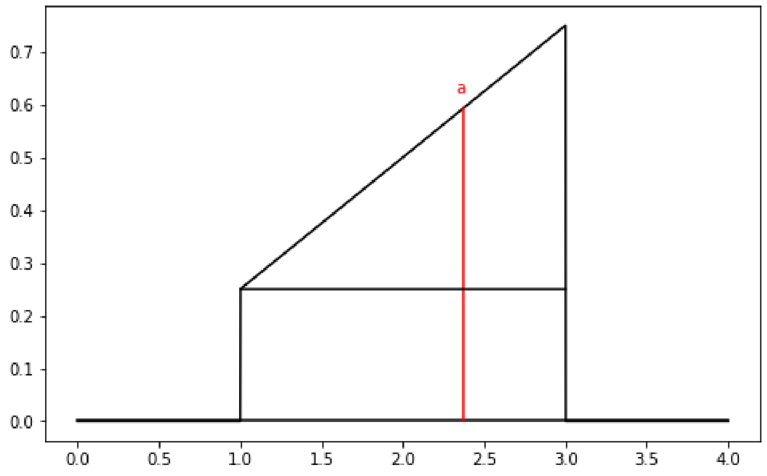
(e) Find $E(X)$

For (d) -- (e) you must use mathematical techniques and not just calculate it using iterative techniques in Python code. You may use Python to calculate results of mathematical formulae.

Solution:

(a) The figure is composed of a rectangle of height 0.25 and width 2.0, plus a triangle of height 0.5 and width 2.0. We must find the area to the left of a given point a, as shown here:

In [3]:
```
a = 2.37
plt.figure(figsize=(8, 5))
plt.plot([0,4],[0,0],color="k")
plt.plot(np.arange(0,4,0.001),[x/4 if 1 <= x <= 3 else 0 for x in np.arange(0,4,0.001)],color=
"k")
plt.plot([a,a],[0,a/4], color="r")
plt.text(a-0.04,a/4+0.03,"a",color="r")
plt.plot(np.arange(1,3,0.001),[0.25 if 1 <= x <= 3 else 0 for x in np.arange(1,3,0.001)],color
="k")
plt.show()
```



Doing these separately, at a point $1 \leq a \leq 3$, we have a rectangle of width $(a-1)$ and height 0.25, so its area is $\frac{a-1}{4}$; we also have a triangle of base $(a-1)$ and height $\frac{a-1}{4}$ (since the slope of the diagonal line is $1/4$), whose area is $\frac{(a-1)^2}{8}$. So we have

$$\frac{a-1}{4} + \frac{(a-1)^2}{8} = \frac{2(a-1)+(a-1)^2}{8} = \frac{a^2-1}{8}.$$

(b) The indefinite integral of $f(x)$ is:

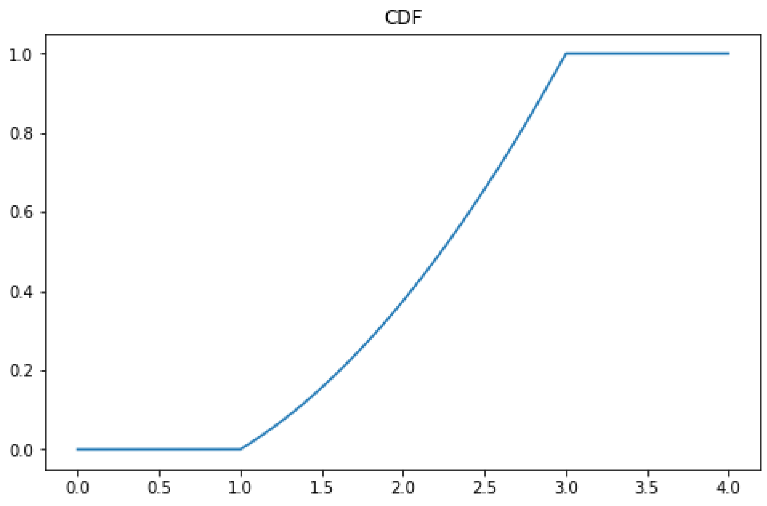
$$\int \frac{x}{4}\,dx \;=\; \frac{x^2}{8} + C$$

So then, breaking it into 3 pieces, we have:

$$F(x) = \begin{cases} 0 & \text{if } x \leq 1 \\ \frac{x^2-1}{8} & \text{if } 1 \leq x \leq 3 \\ 1 & \text{otherwise} \end{cases}$$

(c) The plot is as follows:

In [4]:
```python
plt.figure(figsize=(8, 5))
plt.title("CDF")
plt.plot(np.arange(0,4,0.001),[0 if x <= 1 else (x**2-1)/8 if 1 <= x <= 3 else 1 for x in np.a
range(0,4,0.001)])
plt.show()
```



(d)

$$P(X \geq 2) = 1 - P(X < 2) = 1 - F(2) = 1 - \frac{2^2-1}{8} = 1 - \frac{3}{8} = \frac{5}{8} = 0.625$$

(e)

$$E(X) \;=\; \int_1^3 \frac{x^2}{4}\,dx \;=\; \frac{x^3}{12}\Big|_1^3 = \frac{27}{12} - \frac{1}{12} = \frac{26}{12} = 2.1667$$

Scipy.stats

For the following, use the statistical functions given at the top of this notebook.

Also consider using the Distributions Notebook posted online to visualize these distributions (but use scipy.stats for the calculations).

You are not required to do so, but a nice touch is to print out your answer in a code block, i.e., if you were asked "What is the probability in the standard normal that a value occurs in the interval between -0.94 and 1.2 standard deviations from 0," you could answer as follows:

```
In [5]: mu = 0
        sigma = 1
        lo = -0.94
        hi = 1.2

        answer = norm.cdf(x=hi,loc=mu,scale=sigma) - norm.cdf(x=lo,loc=mu,scale=sigma)
        print("Solution: " + str(round4(answer)))
```

```
Solution: 0.7113
```

Problem Five (Normal Distribution)

Suppose that in a population of individuals, their height is normally distributed with a mean of 68 inches and a standard deviation of 1.45 inches. </p>

(a) What is the probability that a randomly-selected individual has a height less than 66 inches?

(b) What is the probability that a randomly-selected individual has a height more than 72 inches?

(c) What is the probability that a randomly-selected individual has a height between 66.5 and 71 inches?

```
In [6]: answer4a = norm.cdf(x=66,loc=68,scale=1.45)
        print("Solution (a) " + str(round4(answer4a)))
```

```
Solution (a) 0.0839
```

```
In [7]: answer4b = 1.0 - norm.cdf(x=72,loc=68,scale=1.45)
        print("Solution (b) " + str(round4(answer4b)))
```

```
Solution (b) 0.0029
```

```
In [8]: answer4c = norm.cdf(x=71,loc=68,scale=1.45) - norm.cdf(x=66.5,loc=68,scale=1.45)
        print("Solution (c) " + str(round4(answer4c)))
```

```
Solution (c) 0.8303
```

Problem Six (Standard Normal Distribution)

Using the same mean and standard deviation as in the previous problem (mean of 68 inches and a standard deviation of 1.45 inches), show how to answer the following questions by first converting to the standard normal distribution (with mean 0 and standard deviation 1), and then calculating the answer in terms of the standard normal (i.e., when you calculate the probabilities, you will be using N(0,1).) Show all work. For (c) you will not need to convert, but simply use the standard normal directly.

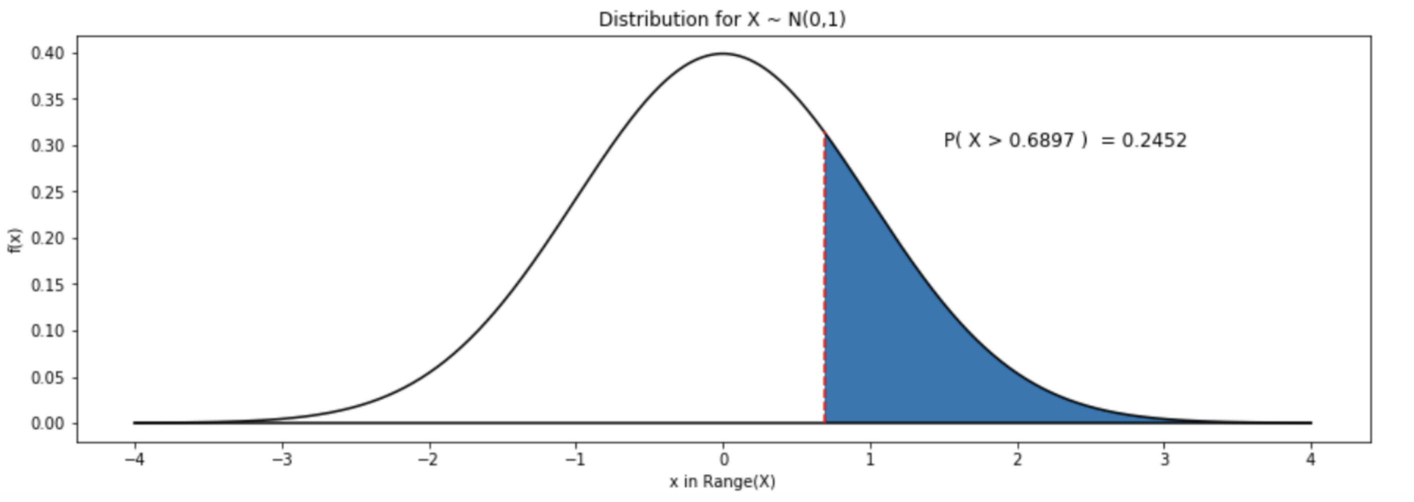
(a) What is the probability that a randomly-selected individual has a height more than 69 inches?

(b) What is the probability that a randomly-selected individual has a height between 63.2 and 70.7 inches?

(c) What is the probability that someone is more than 2.7 standard deviations taller than the average height?

Solution:

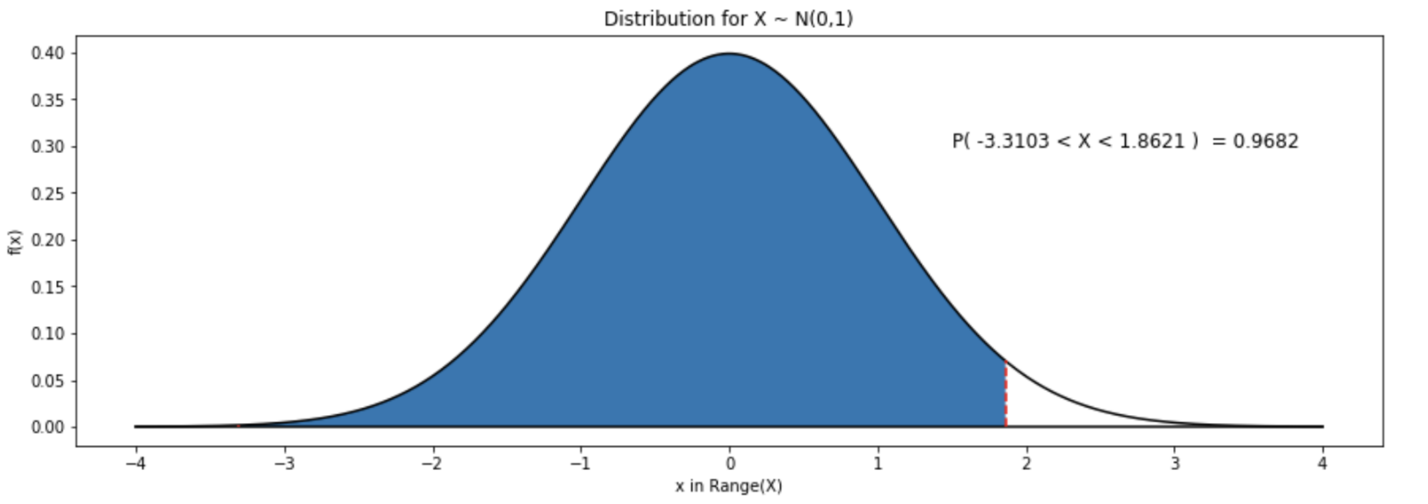
(a) z = (69-68)/1.45 = 0.6897



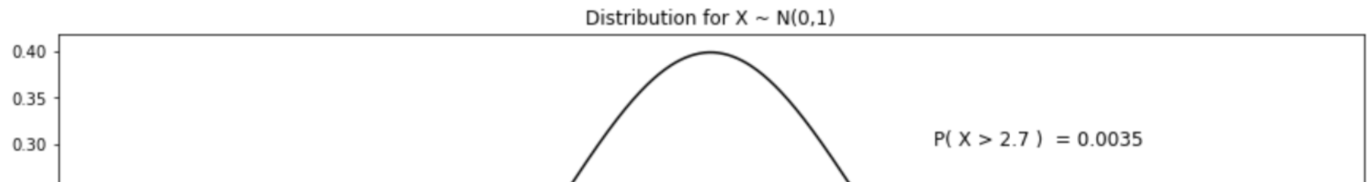
(b)

z_{lo} = (63.5-68)/1.45 = -3.1034 and z_{hi} = (70.7-68)/1.45 = 1.862

```
x = 63.2
y = 70.7
display_normal_range(0, 1,lo=(x-68)/1.45, hi=(y-68)/1.45)
```



(c) z = 2.7



Problem Seven (Normal Distribution)

Suppose the grades on the 237 midterm are normally distributed with mean 83 and variance of 23. Answer the following questions. By the "kth percentile" we mean the x value for which $P(X \leq x) = k/100$ (meaning k is in percentages, and $k/100$ is in terms of probabilities.)

Browse the functions from `scipy.stats.norm` given to see which one will solve each problem.

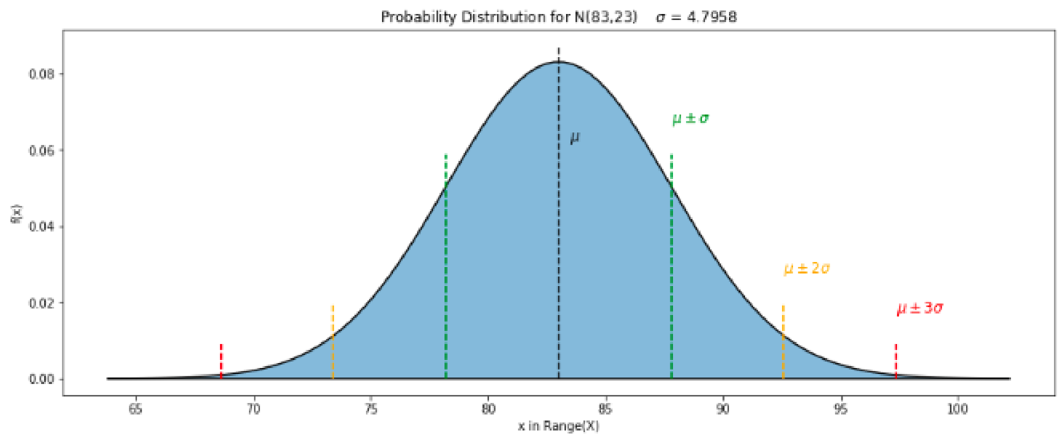
(a) If an A is 93 or above, what percentage of the class will get an A?

(b) What is the exam score which represents the 90th percentile? (Assume exam scores are real numbers.)

(c) If a score between 70 and 80 results in a C, and there are 120 people in the class, approximately how many people will get a C? (Round to the nearest integer.)

Solution:

Here is the distribution, using the Distribution notebook:



In [9]:
```
mu = 83
sigma = 33 ** 0.5
A = 93
# Find P(X>50)
answer6a = norm.sf(x=A,loc=mu,scale=sigma)
print("(a) " + str(round4(answer6a*100)) + "%")
```

```
(a) 4.0861%
```

In [10]:
```
top = 0.1
# Find P(X>50)
answer6b = norm.isf(q=top,loc=mu,scale=sigma)
print("(b) " + str(round4(answer6b)) + "%")
```

```
(b) 90.362%
```

In [11]:
```
n = 120
hi = 80
lo = 70
p = norm.cdf(x=hi,loc=mu,scale=sigma) - norm.cdf(x=lo,loc=mu,scale=sigma)
answer6c = p * n
print("(c) " + str(int(answer6c)))
```

```
(c) 34
```

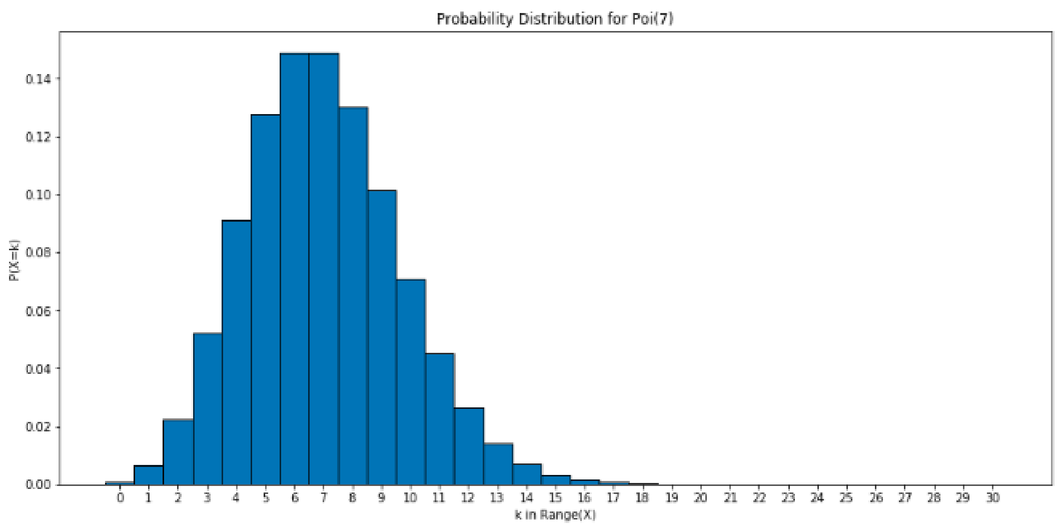
Problem Eight (Combining Poisson with other Distributions)

Suppose that in a certain region of California, earthquakes occur at the average rate of 7 per year. </p>

> (a) What is the probability that in exactly three of the next eight years, no earthquakes occur?
>
> (b) What is the expected number of years to wait until we have a year with exactly 7 earthquakes?
>
> (c) In the next century, how many years would you **expect** to see with more than 10 earthquakes?

Hint: When you see the word "expect" you should expect to use the expected value!

Here is the distribution, from the Distributions notebook:



In [12]:
```
def fact(x):
    if x < 2:
        return 1
    else:
        return x * fact(x-1)


def poi_pdf(lam,x):
    return exp(-lam)*lam**x / fact(x)
```

Solution:

This is obviously X ~ Poi(7).

(a) First we calculate

$$P(X = 0) = \frac{7^0 \cdot e^7}{0!} = 0.0009.$$

In [13]:
```
print(poi_pdf(7,0))
```

```
0.0009118819655545162
```

Then we have the number of earthquakes each year as $Y \sim B(8, 0.0009)$, so

$$P(Y = 3) = \binom{8}{3} \cdot 0.0009^3 \cdot 0.9991^5 = 4 \times 10^{-8}.$$

In [14]:
```
print(binom.pmf(3,8,poi_pdf(7,0)))
```

```
4.226908795802447e-08
```

(b) The probability that a year has exactly 7 earthquakes is $p = P(X = 7)$ for $X \sim Poi(7)$.

In [15]:
```
print(poi_pdf(7,7))
print(1/poi_pdf(7,7))
```

```
0.1490027796743379
6.711284193392975
```

Then the number of years to wait until such a year is $Y = Geo(p)$, and the expected number of years is 1/p = 6.7113 years.

(c) The probability that a year has more than 10 earthquakes is $p = P(X > 10)$ for $X \sim Poi(7)$.

In [16]:
```
answer8c = 1.0 - sum ([ poi_pdf(7,k) for k in range(11) ])
round4(answer8c)
```

Out[16]:
```
0.0985
```

Then the number of years in the next century with more than 10 earthquakes is $B(100, 0.0985...)$ and is the expected number is 100*0.985... = 9.852 years.

Problem Nine (Combining Normal with Other Distributions)

Suppose that in the Men's Olympic Ski Team, the chest size measurements are normally distributed with a mean of 39.8 inches and a standard deviation of 2.05 inches.

(a) What the probability that of 20 randomly selected members of the team, at least 5 have a chest size of at least 41.7 inches?

(b) Supposing we choose men on the team repeatedly and with replacement, how many men would you expect to choose before finding a member with a chest measurement of less than 37 inches?

Hint: Let X is a normally distributed random variable according to the parameters given in the first sentence. Then consider Y and Z be appropriately distributed random variables for (a) and (b) respectively.

In [17]:
```
s = 0
for k in range(5,21):
    s +=  C(20,k)*(0.4611**k)*((1-0.4611)**(20-k))
print(s)
```

```
0.985200040833229
```

Solution

(a) Using the normal distribution, X ~ N(39.8, 2.05^2), and we have P(X > 41.7) = 0.1770. Then, using the binomial, we have Y ~ B(20, 0.177) and so P(Y >= 5) = C(20,5)*(0.177) 5* (1.0 - 0.177) 15 = 0.1276.

(b) For X, we have P(X < 37) = 0.086, and with Z ~ Geo(0.086), we have E(Z) = 1 / 0.086 = 11.6279

In [18]:
```
# P(X <= x)

def cdf_binomial(x,N,p):
    sum = 0
    for k in range(x+1):
        sum += C(N,k)*(p**k)*((1-p)**(N-k))
    return sum
```

In [35]:
```
mu = 39.8
sigma = 2.05
N = 20
p = norm.sf(x=41.7,loc=mu,scale=sigma)
answer8a = 1 - cdf_binomial(5,N,p)
print("(a) " + str(round4(answer8a)))
```

```
(a) 0.1276
```

In [20]:
```
mu = 39.8
sigma = 2.05
N = 20
p = norm.cdf(x=37,loc=mu,scale=sigma)

answer8b = 1/p
print("(b) " + str(round4(answer8b)))
```

```
(b) 11.6289
```

Problem Ten (Normal Approximation to the Binomial)

This problem concerns the normal approximation to the binomial. The "continuity correction" (sometimes called Yates's Continuity Correction) is a technique I will show in the Monday 10/21 lecture for improving the accurate of the normal approximation.

In this problem we will measure the accuracy of the approximations by using *absolute percentage error*, which is defined to be:

$$\frac{|\text{ measured value} - \text{actual value }|}{\text{actual value}} \times 100.$$

</p>

(a) Suppose of all the kids that show up on Halloween night, 58% are dressed in Deadpool costumes. If 60 kids show up, what is the probability that between 33 and 38 (inclusive) kids will be dressed in Deadpool costumes? (Use the binomial.)

(b) Repeat the previous question, but using the normal approximation to the binomial, without using the continuity correction, and express the accuracy of your approximation using the absolute percentage error.

(c) Repeat the previous question, but now using the continuity correction, again showing the accuracy using the absolute percentage error.

Solution: (all computed using appropriate functions in Python)

> (a) X ~ B(60,0.58) and P(33 ≤ X ≤ 38) = 0.5609.
>
> (b) Y ~ N(60*0.58, 60*0.58*(1-0.58)) and P(33 ≤ Y ≤ 38) = 0.4798.
> APE = abs(0.5609 - 0.4798)/0.5609 = 14.4589 %
>
> (c) Y ~ N(60\0.58, 60*0.58*(1-0.58)) and P(32.5 ≤ Y ≤ 38.5) = 0.5597.
> APE = abs(0.5609 - 0.5597)/0.5609 = 0.2139 %
>
> </blockquote>

In [21]:
```
p = 0.58
N= 60
hi = 38
lo = 33
limit = lo-1
answer7a = cdf_binomial(hi,N,p) - cdf_binomial(limit,N,p)
print("(a) " + str(round4(answer7a)))
```

```
(a) 0.5609
```

In [22]:
```
mu = N*p
sigma = (N*p*(1-p))**0.5

answer7b = norm.cdf(x=hi,loc=mu,scale=sigma)  - norm.cdf(x=lo,loc=mu,scale=sigma)
print("(b) " + str(round4(answer7b)))
```

```
(b) 0.4798
```

In [23]:
```
mu = N*p
sigma = (N*p*(1-p))**0.5
answer7b = norm.cdf(x=hi+0.5,loc=mu,scale=sigma)  - norm.cdf(x=lo-0.5,loc=mu,scale=sigma)
print("(c) " + str(round4(answer7b)))
```

```
(c) 0.5597
```

Lab Problems

Generating Variates from a Continuous Distribution

Samples from a given distribution are often called "random variates" or just "variates" for short; to generate *size* random variates from a normal distribution with mean *loc* and standard deviation *scale* we can use the `scipy.stats` function

```
    X = norm.rvs(loc=0,scale=1,size=num_trials)
```

(Note that in this case, the normal is defined in terms of the standard devation, and *not* the variance.)

Run the next cell several times to get a sense for how this function works

In [24]:
```
print("From N(0,1):")
X = norm.rvs()       # default is a standard normal with mean 0 and standard deviation 1
print(X)
print()
print("From N(10,2^2):")
X = norm.rvs(10,2)   # defined by mean and standard deviation (NOT the variance)
print(X)
print()
print("Ten  variates from N(66,3^2):")
X = norm.rvs(66,3,size=10)
print(X)
```

```
From N(0,1):
0.2998758973163816

From N(10,2^2):
8.843140786298884

Ten  variates from N(66,3^2):
[61.80156428 63.0634975  66.48022007 67.28660647 66.33211648 63.0501843
 65.5789761  65.79696359 66.75127323 60.6442569 ]
```

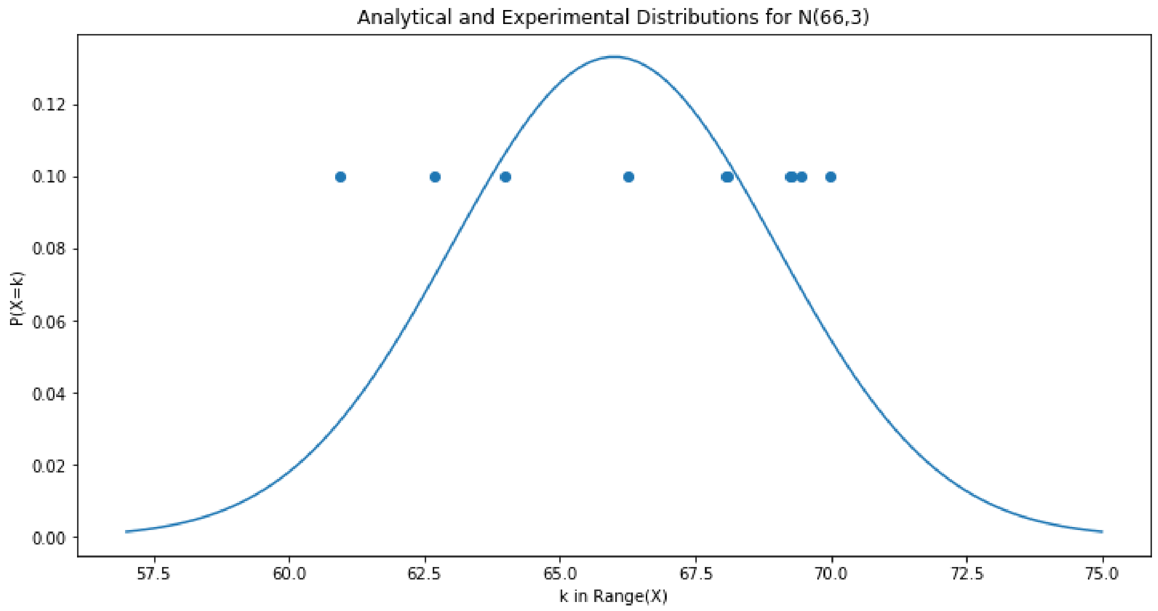
Graphing Continuous Variates: A Problem

So generating normal variates is easy! What we are going to concern ourselves with in this first problem is now to graph a collection of such normal numbers.

Here is the problem: since each value occurs (with high probability) only once, we can't just create a histogram and convert it into a frequency distribution.

Here is what happens if we do this, and graph it as a scatter plot against the theoretical (continuous) distribution:

In [25]:
```
def display_normal_samples(mu,sigma,num_trials):
    fig, ax = plt.subplots(1,1,figsize=(12,6))
    plt.title('Analytical and Experimental Distributions for N('+str(mu)+','+str(sigma)+')')
    plt.ylabel("P(X=k)")
    plt.xlabel("k in Range(X)")
    # use normal(...) to generate random samples
    X = sorted(norm.rvs(mu,sigma,num_trials))
    # Now convert frequency counts into probabilities
    D = Counter( X )
    P = [D[k]/num_trials for k in X]
    plt.scatter(X,P)
    # Now generate the theoretical normal with the same mean and
    X2 = np.linspace(mu-sigma*3,mu+sigma*3,100)
    Y = [norm.pdf(x,mu,sigma) for x in X2]
    plt.plot(X2,Y)
    plt.show()

# try setting the number of trials - the number of samples generated -- to 100 and 1000.
num_trials = 10
display_normal_samples(66,3,num_trials)
```



Graphing Continuous Variates with Bins

You see the problem: since each floating point number (an approximation of a real number) is generated with high probability at most once, we can't see the accumulation of samples that would indicate the probability.
Essentially we are trying to create a frequency distribution from values whose theoretical probability is 0. What to do?

Well, you know that probabilities can only be calculated in continuous distributions using *intervals*, so we will create equally-sized intervals to collect together our samples from the continuous distribution. Then we must "slot" each variate into its appropriate bin and calculate the probabilities.

Thus, we transform a continuous distribution into a discrete one for the purposes of visualizing it.

The pyplot function hist(...) does the slotting, if we give it a list of outcomes and the bin boundaries, so that really all we have to do is define the bins.

Graphing Normal Variates with Bins Calculated from Standard Deviation

For the normal distribution it makes sense to define the bin boundaries in terms of standard deviations from the mean, since we will be dealing with an unknown range of data; this bins can be made as wide or as narrow as we want, but will represent an interval defined in terms of the standard deviation sigma of the distribution.

We will graph the distribution in a range of at least 4 standard deviations of the mean, ignoring the rare occurance of a variate outside this range.

In [26]:
```
# Define the boundaries of bins with the specified width around the mean,
# to plus/minus at least 4 * sigma

# bin_width is in units of sigma, so bin_width = 0.1 means sigma/10

def makeBins(mu,sigma,bin_width):
    numBins = ceil(4/bin_width)
    bins = [mu+sigma*bin_width*x for x in range(-numBins,numBins+1)]
    return bins

# Change the parameters several times to see the effect of this

print(makeBins(0,1,.5))
```

```
[-4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]
```

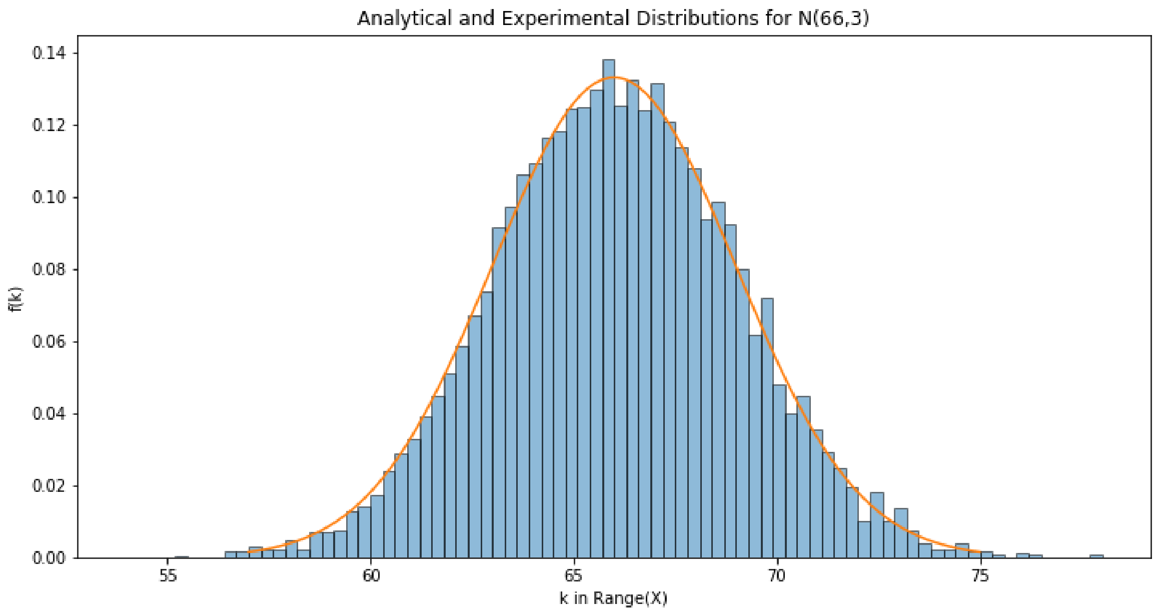
Problem Eleven: Generating and Graphing Normal Variates

How does the number of trials affect the fit of the data to the normal distribution?

Now let's do our previous experiment but trying various values for bin_width...

```
In [27]: def display_normal_samples_binned(mu,sigma,num_trials,bin_widths):
             fig, ax = plt.subplots(1,1,figsize=(12,6))
             plt.title('Analytical and Experimental Distributions for N('+str(mu)+','+str(sigma)+')')
             plt.ylabel("f(k)")
             plt.xlabel("k in Range(X)")
             # use norm.rvs(...) to generate random samples
             X = norm.rvs(mu,sigma,num_trials)
             plt.hist(X,bins=makeBins(mu,sigma,bin_widths),normed=True,edgecolor='k',alpha=0.5) # bins a
         re of width 1/10**decimals
             #plt.hist(X,bins=makeBins(mu,sigma,bin_widths),density=True,edgecolor='k',alpha=0.5) # use
          if get warning
             # Now generate the theoretical normal with the same mean and
             X2 = np.linspace(mu-sigma*3,mu+sigma*3,100)
             Y = [norm.pdf(x,mu,sigma) for x in X2]
             plt.plot(X2,Y)
             plt.show()

         #try each of these and observe the effects

         N = 10000     # try 100, 1000, and 1,000,000

         display_normal_samples_binned(66,3,N,0.1)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: MatplotlibDeprecationWarnin
g:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density'
instead.
```



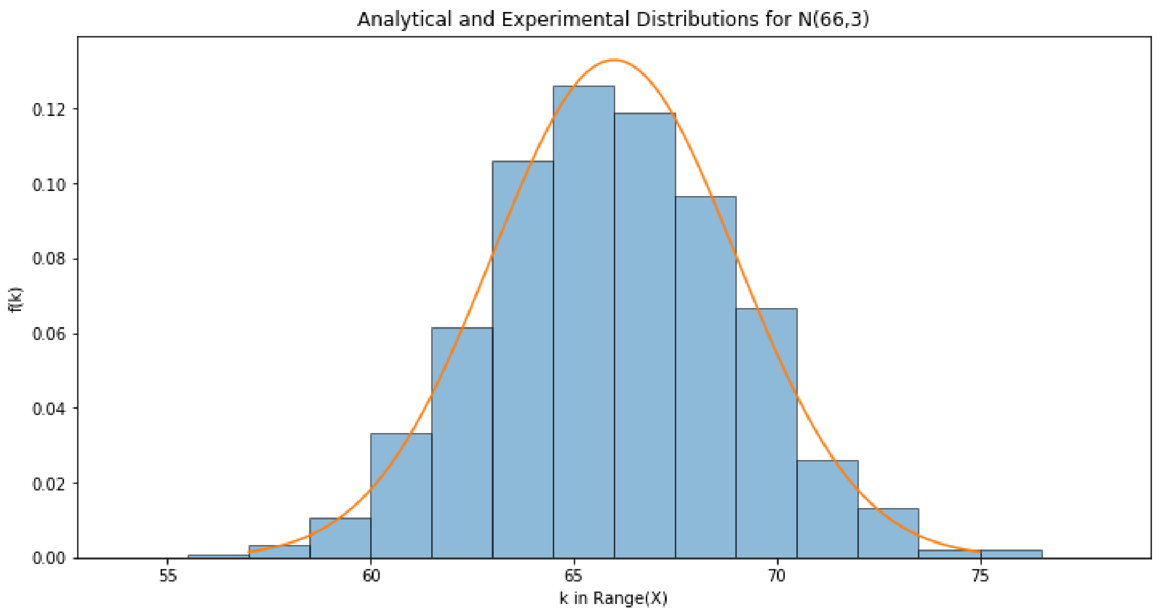
Clearly the data seems to fit the normal better when the number of trials increases...

Affect of the bin width

But now let's think about the issue of precision, i.e., the width of the bins. Again, try each of the following and see what happens. You can see that too-wide bins don't give much information, but too-narrow bins don't show how the data fits the normal distribution. There is a relationship between the number of data points and the width of the bins.

```
In [28]: bin_width = 0.5       # try changing this to 0.5, 0.2, 0.1, 0.05, and 0.01

         display_normal_samples_binned(66,3,1000,bin_width)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: MatplotlibDeprecationWarnin
g:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density'
instead.
```

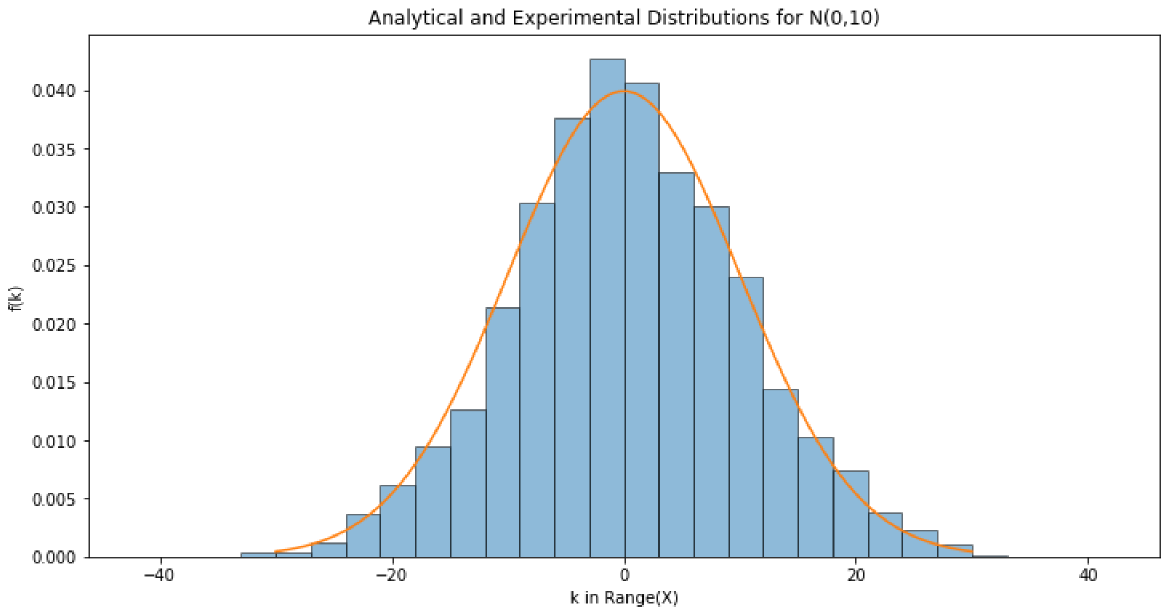


Problem 11 (a)

Clearly the "fit" with the normal curve depends on the width of the bins! For the following three examples, find a value for the indicated parameter which gives a good correspondence between the normal curve and the data.

```
In [29]: # Problem 1(a)
         bin_width = 0.3    # experiment with this value 0.01, 0.05, 0.1, 0.15, etc. -- find the
                            # largest number which still gives a good fit

         display_normal_samples_binned(0,10,3000,bin_width)     # don't change this line

         # Solution:  Around 0.1 - 0.5 is probably right
```

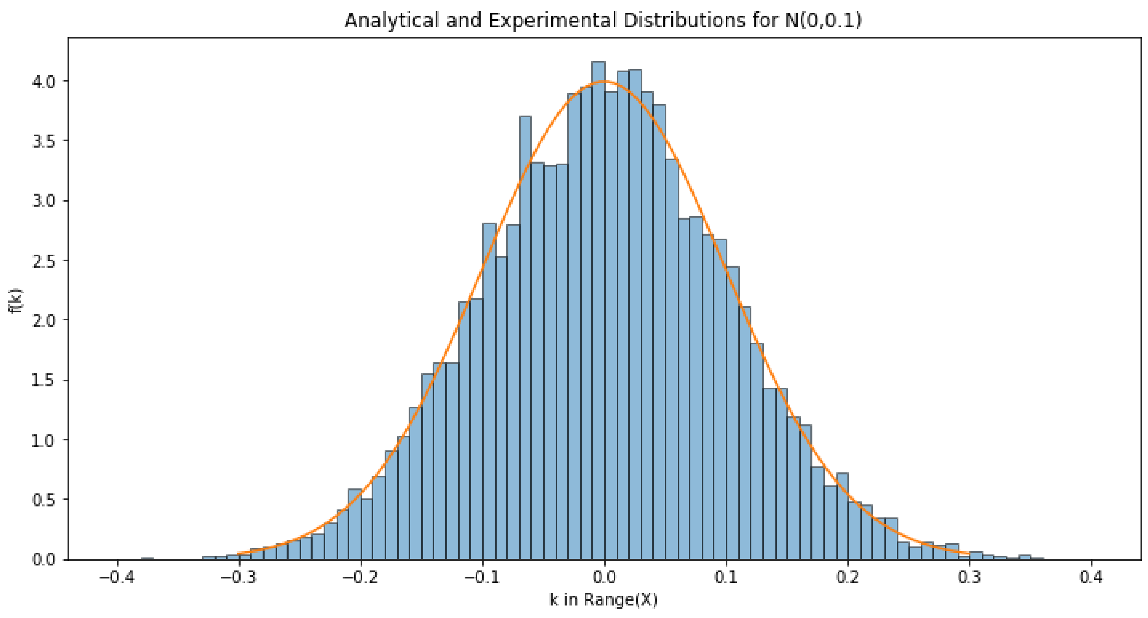
```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: MatplotlibDeprecationWarnin
g:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density'
instead.
```



Problem 11 (b)

```
In [30]: # Problem 1(b)

         bin_width = 0.1     # experiment with this value -- find the largest number which still gives a
         good fit

         display_normal_samples_binned(0,0.1,10000,bin_width)     # don't change this line

         # Solution:  0.1 - 0.05 is probably about right
```

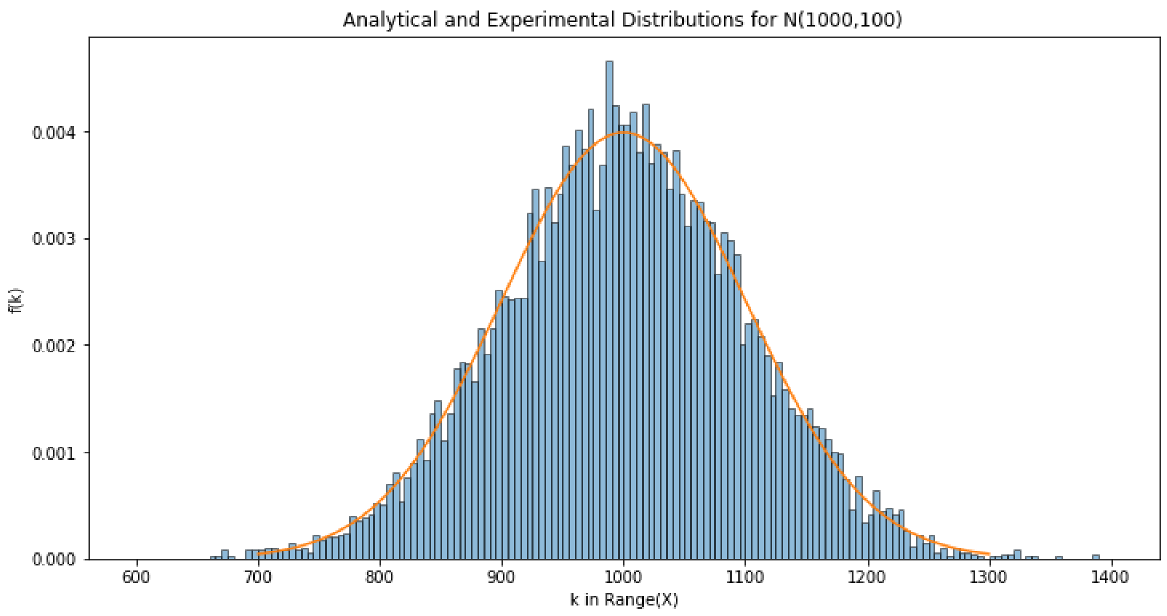
```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: MatplotlibDeprecationWarnin
g:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density'
instead.
```



Problem 11 (c)

```
In [31]: # Problem 11(c)

         bin_width = 0.05    # experiment with this value -- find the largest number which still gives a
         good fit
         display_normal_samples_binned(1000,100,10000,bin_width)     # don't change this line

         # Solution:  0.1 - 0.05 is probably right
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: MatplotlibDeprecationWarnin
g:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density'
instead.
```



Problem 11 (d)

Problem 11 (d)

What do you think is a good value for bin_width in general, assuming that num_trials is sufficient to give a reasonable approximation of the normal distribution

Solution: Probably about 0.1

Problem Twelve: Generating Poisson and Exponential Variates

Following up on HW 5, Problem 12, we now explore how to generate random variates from the Exponential and Poisson Distributions. We will display the results using the solutions to the previous problem.

Part (A)

The following formula should look familiar! If U is a random variable uniformly distributed in the range [0..1), then

$$Y = \frac{-\ln(U)}{\lambda}$$

is a real number which is distributed according to the Exponential Distribution with rate parameter λ, i.e.,

$$Y \sim Exp(\lambda).$$

Note: $\ln$ is log to the base e (just `log(...)` in Python).

For this problem, you must create a function `exp_rvs(...)` which generates exponential variates, using this formula (it is very similar to your solution to the last problem in HW 5), and demonstrate it by printing out 10 values.

```
In [32]: ## Solution for (A)

         def exp_rvs(lam):
             return ( - log( random() ) / lam  )

         # test it

         lam = 4

         num_trials = 10

         seed(0)

         outcomes12a = [ exp_rvs(lam) for k in range(num_trials)]
         outcomes12a               # first number should be 0.04227577129370466
```

```
Out[32]: [0.04227577129370466,
          0.06928301239535861,
          0.21653514653972164,
          0.3378121740979418,
          0.16771205450973864,
          0.22600771216986748,
          0.060900798344498956,
          0.2982477266126604,
          0.18527102628047376,
          0.1347282520024191]
```

Part (B)

Now you must complete the following code template to display the results of generating 10^5 exponential variates from Exp(1), and finding a way to bin the results so that you get an accurate idea of how well they fit the theoretical distribution.

The binning here is a little simpler than in the case of the normal distribution, we will simply give the values we want to display (since the range is infinite) and the number of bins to show. This might involve a little tweaking to get a good diagram....

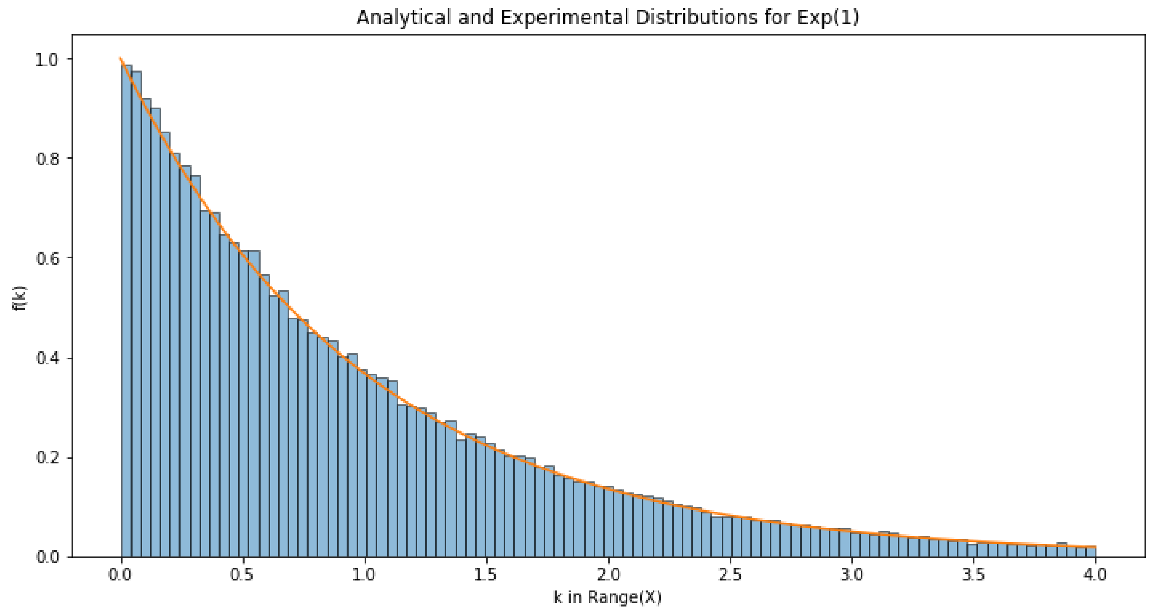
In order to prepare your experimental results with the theory, you will have to write the function for the exponential pdf (easy, but humor me!).

Optional: Can you think of a way of limiting the range based on λ?

One Solution: Bound it so that the probability of an outcome being over the limit is about 1/num_trials, that is, solve this for x: e^(-lam*limit) = 1/num_trials, or, more simply: limit = num_trials/lam

In [33]:
```
## Solution for (B)

def fact(n):
    if n < 2:
        return 1.0
    else:
        return n * fact(n-1)

def exp_pdf(lam,x):
    return lam * exp(-lam*x)                     # Your answer here

def display_exponential_samples_binned(lam,num_trials,limit,num_bins):
    fig, ax = plt.subplots(1,1,figsize=(12,6))
    plt.title('Analytical and Experimental Distributions for Exp('+str(lam)+')')
    plt.ylabel("f(k)")
    plt.xlabel("k in Range(X)")
    # use exp_rvs(...) to generate random samples
    X = [exp_rvs(lam) for k in range(num_trials)]
    # filter out those above the limit
    X1 = [ x for x in X if x < limit ]
    # make the bins
    bs = linspace(0,limit,num_bins)
    plt.hist(X,bins=bs,normed=True,edgecolor='k',alpha=0.5)
    #plt.hist(X,bins=bs,density=True,edgecolor='k',alpha=0.5)          # use this line if get w
arning about normed
    # Now generate the theoretical distribution with the same rate
    X2 = linspace(0,limit,100)
    Y = [ exp_pdf(lam,x) for x in X2]
    plt.plot(X2,Y)
    plt.show()

# test it

lam = 1

num_trials = 10**5

# You will only display values in range [0 .. limit]

limit = 4         # Your answer here

# The range will be put into this many bins

num_bins = 100     # Your answer here

seed(0)

display_exponential_samples_binned(lam,num_trials,limit,num_bins)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23: MatplotlibDeprecationWarnin
g:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density'
instead.
```



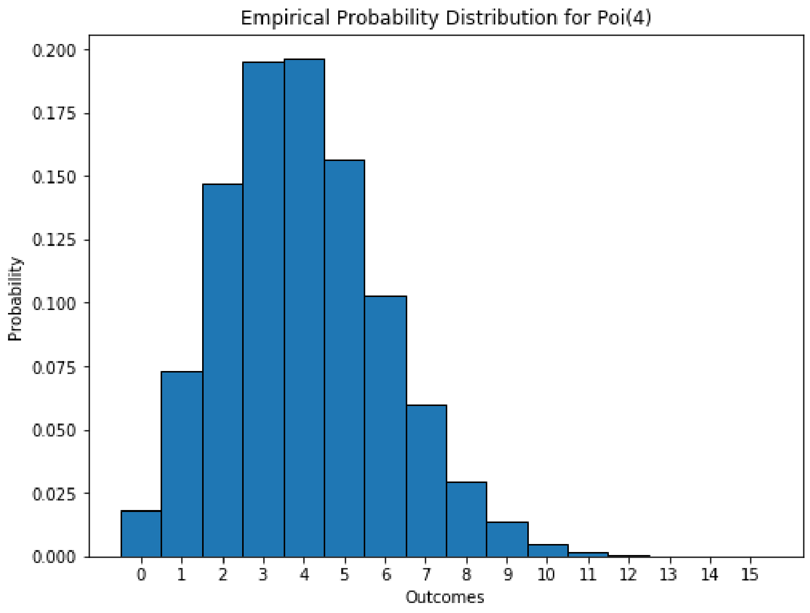
Part (C)

Now we will generate Poisson random variates by simulation, counting how many exponentially-distributed arrivals occur in a unit time (using our result from (a)).

For this problem, first recall that λ is the number of arrivals per unit time (meaning in 1.0 time units); so to simulate one "poke" at the Poisson RV `poi_rvs(lam)` you should "poke" the exponential RV you wrote above, summing the values until you exceed 1.0, then you can count how many arrivals in fact arrived between times 0 and 1.0.

Fill in the function stub and verify by running the code block to see the results (you can compare them with the theoretical distribution on the Distributions notebook).

```
In [34]: ## Solution for (C)

         def poi_rvs(lam):
             count = 0
             total = 0.0
             while (True):
                 count += 1
                 total += exp_rvs(lam)
                 if total > 1.0:
                     return count-1
             return -1

         # test it

         lam = 4

         num_trials = 10**5

         seed(0)

         Outcomes12c = [ poi_rvs(lam) for k in range(num_trials)]

         show_distribution( Outcomes12c , "Empirical Probability Distribution for Poi(" + str(lam) + ")"
         )

         print("mean = ", mean(Outcomes12c))
```



```
mean =  3.996
```