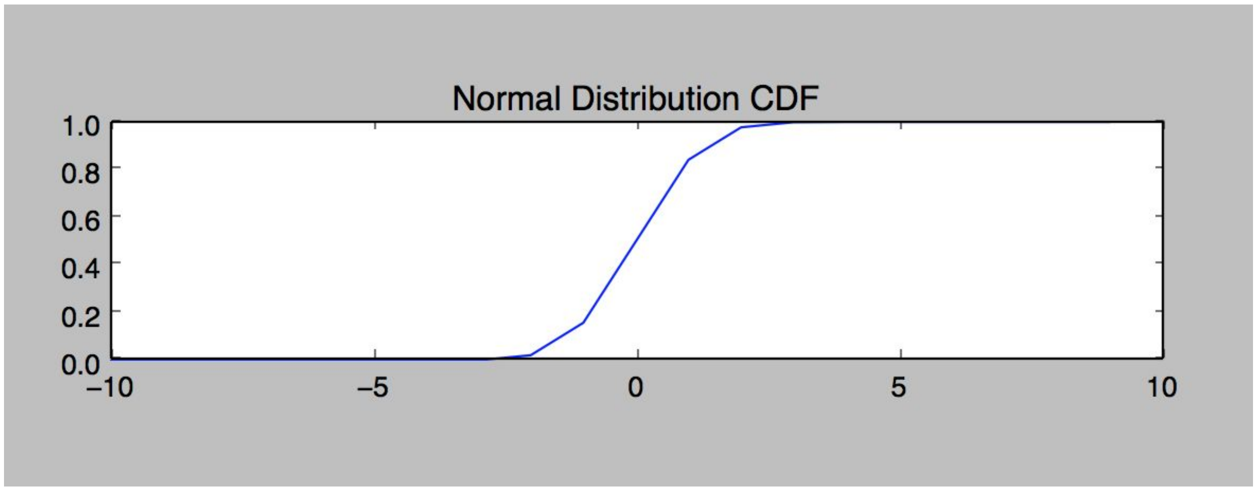
Keeyan Haghshenas
April 30th 2017
Probability and Random Processes
Homework Assignment

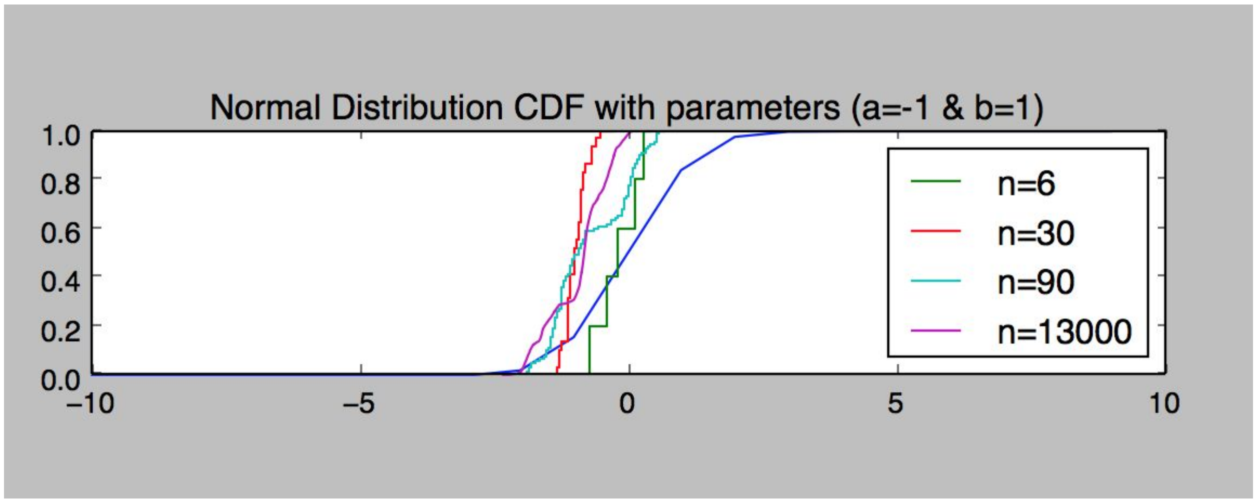
Homework 3: Central Limit Theorem

Part 1:
Normal Distribution of $\Phi(z)$

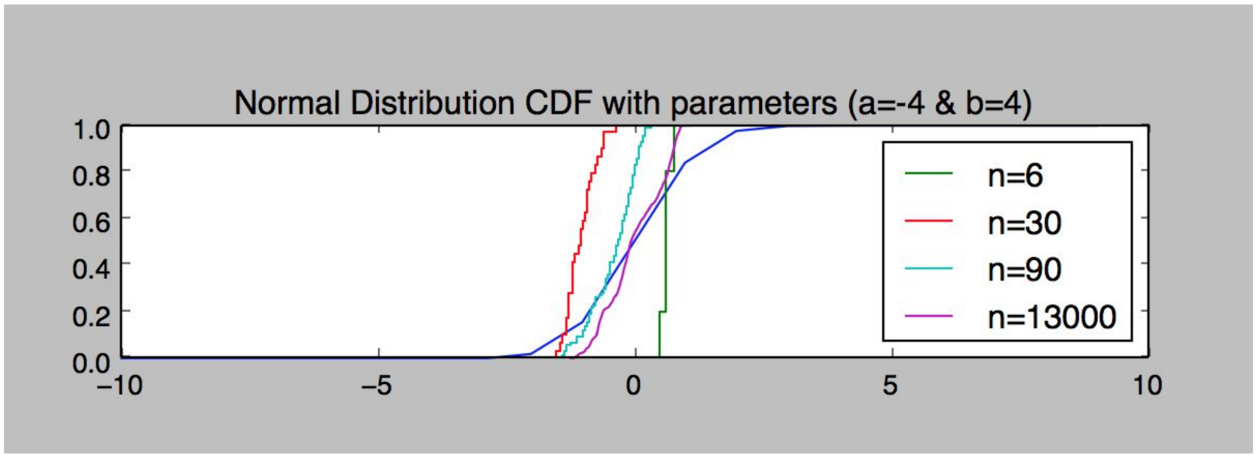


What is graphed here is the normal distribution of the CDF. The expected value is equal to zero and the variance is equal to one.

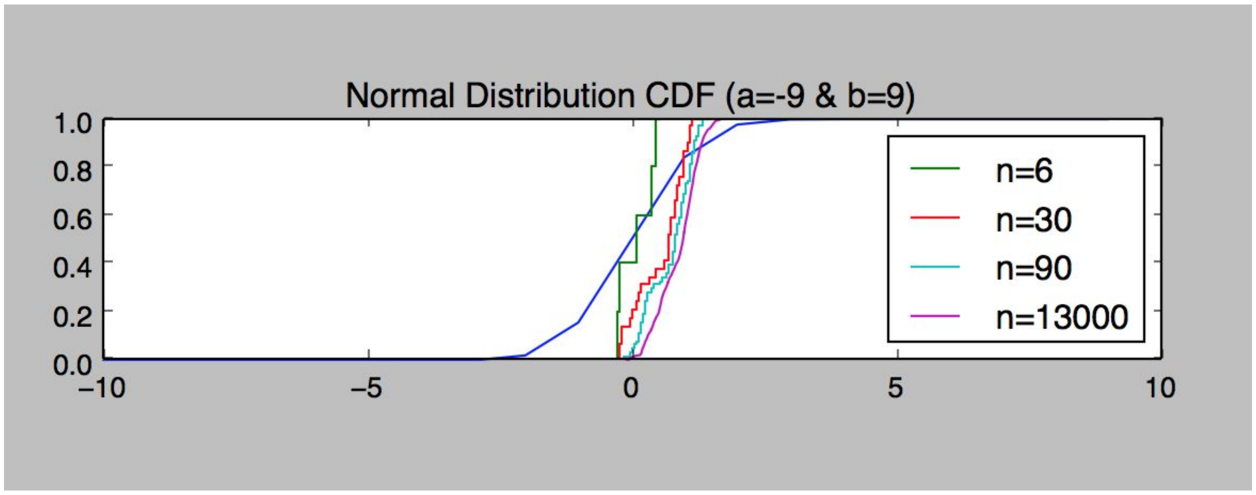
Part 2:



Comments
Here what is graphed is the line previously and I am using that as my trend line. Then using the parameters to determine the expected values and variance, I varied the n values drastically to show the phenomena of how when n increases the line better fits the normal distribution with no parameters.



Comments
Here what is graphed is the line from part 1 and I am using that as my trend line. Then using the parameters to determine the expected values and variance, I varied the n values drastically to show the phenomena of how when n increases the line better fits the normal distribution with no parameters.



Comments
Here what is graphed is the line from part 1 and I am using that as my trend line. Then using the parameters to determine the expected values and variance, I varied the n values drastically to show the phenomena of how when n increases the line better fits the normal distribution with no parameters.

Part 3:

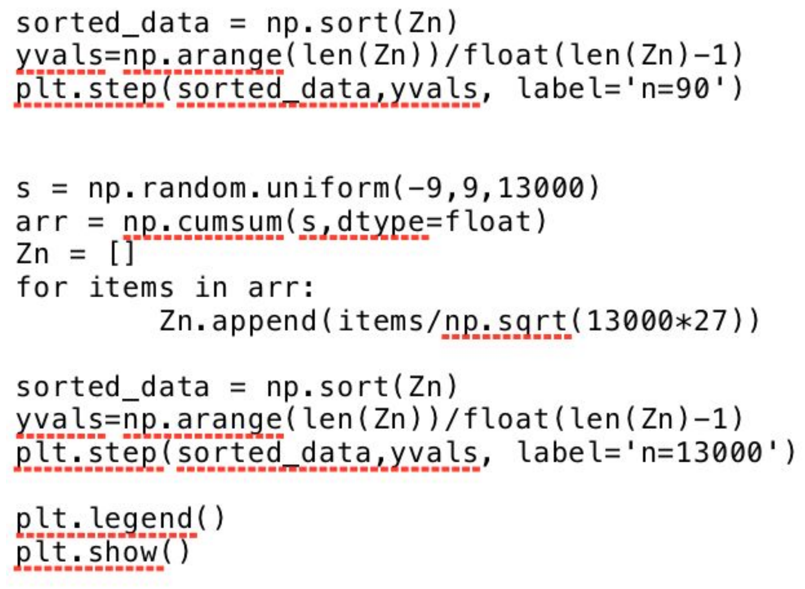
Through analysis of the graphs its is clear to see that as n grows larger, the line will better fit the normal distribution of the CDF. I purposely incremented the the graphs in such a manner where such phenomena would be obvious. The steepest of the lines is always when n is equal to 6 and as there is also a significant difference between n = 6 and n = 30 and 90. To further prove this point I made my last line something with a very large n to prove that as n increases the line better fits the trend line. The reason this occurs and the reason why as to why you need to increase n for the CDF to converge as b-a increases is because of variance. The variance function is given as ((b-a)^2)/12. So as the difference of b and a increases the variance increases as a squared function, also as the bounds change as well, the numerator of the fraction is constantly increasing, to ensure that the function fits the Normal Distribution of the CDF, n which is present in the denominator must also increase too.

Code:

```
from scipy.stats import norm
from scipy.stats import uniform
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.special import erf, erfc


# Normal Distribution Parameter Definition
mu = 0
sigma = 1
rng = range(-10,10)

# To create the normal distribution using the parameters defined previously, in this case Standard deviation and Mean
dist = norm(mu, sigma)

# This is the code to plot the PDF and CDF of the normal distribution created previously
plt.subplot(312)
plt.plot(rng, dist.cdf(rng))
plt.title('Normal Distribution CDF')

plt.show()

# Normal Distribution Parameter Definition
mu = 0
sigma = 1
rng = range(-10,10)

# To create the normal distribution using the parameters defined previously, in this case Standard deviation and Mean
dist = norm(mu, sigma)

# This is the code to plot the PDF and CDF of the normal distribution created previously
plt.subplot(312)
plt.plot(rng, dist.cdf(rng))
plt.title('Normal Distribution CDF with parameters (a=-1 & b=1)')

s = np.random.uniform(-1,1,6)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(6/3))
```

```
sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals,label='n=6')

s = np.random.uniform(-1,1,30)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(30/3))

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals,label='n=30')

s = np.random.uniform(-1,1,90)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(90/3))

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals,label='n=90')

s = np.random.uniform(-1,1,13000)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(13000/3))

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals, label='n=13000')

plt.legend()
plt.show()

# Normal Distribution Parameter Definition
mu = 0
sigma = 1
rng = range(-10,10)

# To create the normal distribution using the parameters defined previously, in this case Standard deviation and Mean
dist = norm(mu, sigma)
```

```
# This is the code to plot the PDF and CDF of the normal distribution created previously
plt.subplot(312)
plt.plot(rng, dist.cdf(rng))
plt.title('Normal Distribution CDF with parameters (a=-4 & b=4)')

s = np.random.uniform(-4,4,6)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append((items/np.sqrt(6))*.433012)

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals,label='n=6')

s = np.random.uniform(-4,4,30)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append((items/np.sqrt(30))*.433012)

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals,label='n=30')

s = np.random.uniform(-4,4,90)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append((items/np.sqrt(90))*.433012)

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals,label='n=90')


s = np.random.uniform(-4,4,13000)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(13000*5.333))

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals, label='n=13000')

plt.legend()
plt.show()

# Normal Distribution Parameter Definition
mu = 0
sigma = 1
rng = range(-10,10)

# To create the normal distribution using the parameters defined previously, in this case Standard deviation and Mean

dist = norm(mu, sigma)

# This is the code to plot the PDF and CDF of the normal distribution created previously
plt.subplot(312)
plt.plot(rng, dist.cdf(rng))
plt.title('Normal Distribution CDF (a=-9 & b=9)')

s = np.random.uniform(-9,9,6)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(6*27))

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals,label='n=6')

s = np.random.uniform(-9,9,30)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(30*27))

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals,label='n=30')

s = np.random.uniform(-9,9,90)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(90*27))

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals, label='n=90')
```



```
sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals, label='n=90')


s = np.random.uniform(-9,9,13000)
arr = np.cumsum(s,dtype=float)
Zn = []
for items in arr:
        Zn.append(items/np.sqrt(13000*27))

sorted_data = np.sort(Zn)
yvals=np.arange(len(Zn))/float(len(Zn)-1)
plt.step(sorted_data,yvals, label='n=13000')

plt.legend()
plt.show()
```

Code Explanation:
What is Initially written in the code is what is going to be manipulated to fit the different specifications that the homework requires. Essentially because the program does the same thing with each pairing, the values of n and values of a and b are the only things that change. So what is written after the initial CDF plot of a = -1 and b =1 is what is manipulate to fit the further parameters of the assignment. Moreover, it is like creating a template function that can be manipulated for different input and output values. The template starts with initially graphing the Normal distribution of the CDF with no parameters, which is the trend line to which we are trying to fit by increasing n. To generate the next graph, the random values for the bounds -1 and 1 have to be generated. Using the command “np.random.uniform(-1,1,6)” , using the bounds and number of steps for input to the command. The values that this command generates are then stored into an array where each index of the array is summed with the previous and this Sn is then used to create Zn which is then stored in another array. The Zn array is sorted and that is the CDF function. Then the rest of the code is used to plot the CDF function from the sorted Zn array using a step function. Once this all occurs and everything has been plotted the code then graphs all the graphs yielding the results.