****

Total Score:

Question

a)

b)

c)

d)

e)

f)

g)

h)

**ANL252**

**PYTHON FOR DATA ANALYTICS**

**Tutorial Group T09**

**TMA01**

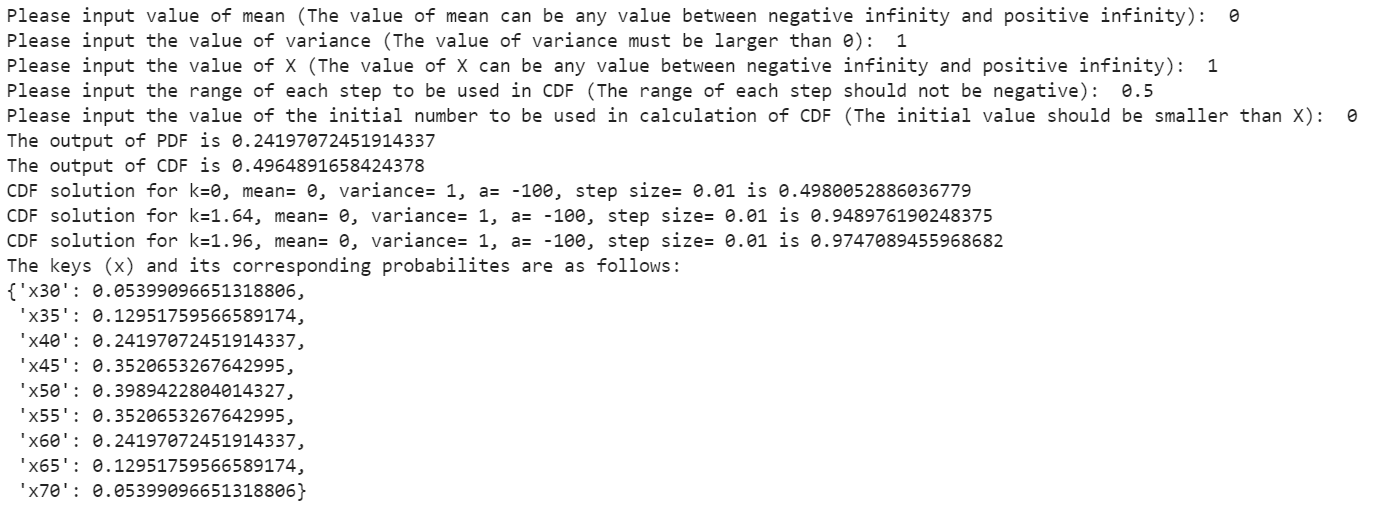
|  |  |
| --- | --- |
| **Name** | **PI Number** |
| **LU KAI WEN** | **J1911368** |

**Submission Date: 15 Aug 2021**

**Question part (g)**

Firstly, I created an empty list and converted into a numpy array. Secondly, I used the while condition to determine if x=k > initial value. If it is greater, I calculated the PDF (initial value), appended into the array and reinitialised the initial value as initial value + step size. The while condition will keep running until x=k < initial value. Thirdly, I sum all elements in the numpy array consisting of multiple PDF (initial value(s)). Finally, I printed the result and returned the result of CDF as output.

**Output of code:**



**.py file with code:**



import math

from sys import exit

import numpy as np

import pprint

def pdf(mean, variance, x): #created a probability density function

fx = (1/math.sqrt((2\*math.pi\*float(variance))))\*math.exp(-((float(x)-float(mean))\*\*2)/(2\*float(variance))) #followed the formula given

print(f'The output of PDF is {fx}')

return fx #the desired output of PDF

def pdf\_noprint(mean, variance, x): #created a PDF without the print() function

fx = (1/math.sqrt((2\*math.pi\*float(variance))))\*math.exp(-((float(x)-float(mean))\*\*2)/(2\*float(variance)))

return fx

def cdf(width, x, ini): #created a cumulative density function

emptylist = []

px = np.array(emptylist) #created an empty numpy array

while float(x) >= float(ini): #conditional to check if we have reached the final term in the CDF function

pdfsol = pdf\_noprint(mean, variance, ini) #calculated the output of each PDF

px = np.append(px, [pdfsol]) #appended the solution of each PDF into the empty numpy array created earlier

ini = float(ini) + float(width) #increased the initial value by the value of the step size

else:

output = float(width)\*np.sum(px) #sum entire numpy array and multiplied by the step size

print(f'The output of CDF is {output}')

return output #desired output of CDF

def cdf\_noprint(width, x, ini): #created a PDF without the print() function

emptylist = []

px = np.array(emptylist)

while float(x) >= float(ini):

pdfsol = pdf\_noprint(mean, variance, ini)

px = np.append(px, [pdfsol])

ini = float(ini) + float(width)

else:

output = float(width)\*np.sum(px)

return output

mean = input('Please input value of mean (The value of mean can be any value between negative infinity and positive infinity): ') #mean input

if len(mean) == 0: #if no values are provided, mean is set at 0

mean = 0

else:

try:

float(mean) #attempt to convert input into float. If ValueError happens, input is non-numeric.

except ValueError:

print('Error: The input value of mean must be numeric!')

exit() #closes the entire code

variance = input('Please input the value of variance (The value of variance must be larger than 0): ')

if len(variance) == 0: #if no values are provided, variance is set at 1

variance = 1

else:

try:

float(variance)

except ValueError:

print('Error: The intput value of variance must be numeric!')

exit()

if float(variance) <= 0: #check if the variance is greater than 0

print('Error: The input value of variance must be greater than 0!')

exit()

x = input('Please input the value of X (The value of X can be any value between negative infinity and positive infinity): ') #X input

if len(x) == 0: #check if the value of x is empty i.e. the user merely pressed ENTER without providing any values

print('The value of X cannot be empty!')

exit()

else:

try:

float(x)

except ValueError:

print('Error: The input of X must be numeric!')

exit()

width = input('Please input the range of each step to be used in CDF (The range of each step should not be negative): ')

if len(width) == 0: #if no step size is provided, default step size is 0.01

width = 0.01

else:

try:

float(width)

except ValueError:

print('Error: The input value of range of each step is not numeric!')

exit()

if float(width) < 0: #if step size is lesser than 0, displays an error

print('Error: The input value of range of each step should not be negative!')

exit()

ini = input('Please input the value of the initial number to be used in calculation of CDF (The initial value should be smaller than X): ') #initial value input

if len(ini) == 0: #initial value must not be empty

print('The initial value cannot be empty!')

exit()

else:

try:

float(ini)

except ValueError:

print('The initial value must be numeric!')

exit()

if float(ini) > float(x): #initial value must be smaller than X - error is displayed if larger than X

print('The initial value must be smaller than X!')

exit()

pdf(mean, variance, x) #calls the PDF

cdf(width, x, ini) #calls the CDF

pdf1 = pdf\_noprint(0,1,0)

cdf1 = cdf\_noprint(0.01, 0, -100)

print(f'CDF solution for k=0, mean= 0, variance= 1, a= -100, step size= 0.01 is {cdf1}')

pdf2 = pdf\_noprint(0, 1, 1.64)

cdf2 = cdf\_noprint(0.01, 1.64, -100)

print(f'CDF solution for k=1.64, mean= 0, variance= 1, a= -100, step size= 0.01 is {cdf2}')

pdf3 = pdf\_noprint(0, 1, 1.96)

cdf3 = cdf\_noprint(0.01, 1.96, -100)

print(f'CDF solution for k=1.96, mean= 0, variance= 1, a= -100, step size= 0.01 is {cdf3}')

#Explanation for calculation of P(X<=k) in (f)

#First, I created an empty list and converted into a numpy array.

#Secondly, I used the while condition to determine if x=k > initial value

#If greater, I calculated the PDF(initial value), appended into the array and reinitialised the initial value as initial value + step size

#The while condition will keep running until x=k < initial value.

#Thirdly, I sum all elements in the numpy array consisting of multiple PDF(initial value(s))

#Finally, I printed the result and returned the result of CDF as output.

dict = {} #create empty dictionary

key = 'x'+str(0) #keys are designed to start from x0, x1, x2 ... xN; where N is the position of the last value

i = -5 #initialise the starting value to be at -5

j= 0

while i < 5:

key = 'x' + str(0+j)

dict[key] = str((round(i,2)))

j = j + 1 #add 1 to xN for each time the loop is completed

i = i + 0.1 #add 0.1 to obtain the desired normal distribution of x as described in the question

newdict = {} #creates a new filtered dictionary

for key,value in dict.items(): #filters out dict.items() such that the newdict = {-2, -1.5, -1 ... 2}

lim = -2

step = 0.5

while lim <= 2:

if float(value) == lim:

newdict[key] = value

lim = lim + step

finaldict = {} #creates a final dictionary to pair the key xN to its corresponding PDF output

for key,value in newdict.items(): #uses the PDF to calculate output

calc = pdf\_noprint(0, 1, value)

value = calc

finaldict[key] = value #reassigns key xN to corresponding PDF output

print('The keys (x) and its corresponding probabilites are as follows:') #print as requested

pprint.pprint(finaldict)