Departamento de Informática - PUC-Rio

INF 1036 - Probabilidade Computacional

Nome: Felipe Holanda Bezerra

Matrícula: 1810238

Questão 1: (3.0 pontos)

Sejam $\{X_1, \ldots, X_n\}$ amostras de uma variável aleatória X. O terceiro momento central é dado por:

$$m_3 = \sum_{i=1}^{n} \frac{(X_i - \bar{X})^3}{(n-1)}$$

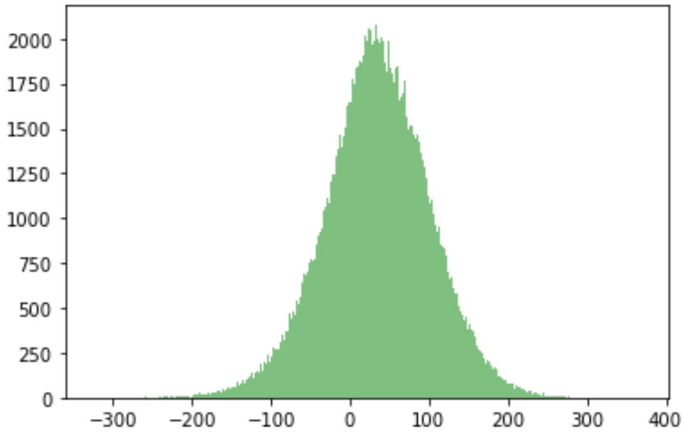
Faça um algoritmo utilizando a técnica de bootstrap para estimar o erro médio quadrático de m_3 Teste seu algoritmo utilizando $\{X_1, \ldots, X_n\}$ como: $\{5, 4, 9, 6, 21, 17, 11, 20, 7, 10, 21, 15, 13, 16, 8\}$

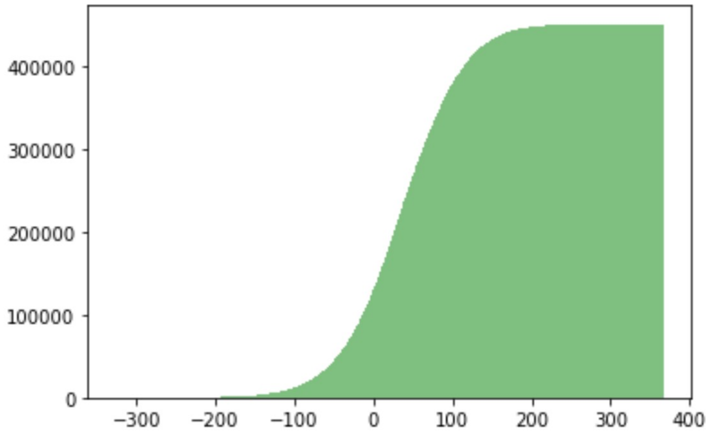
In [9]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

%matplotlib inline

# calculo do terceiro momento central
def g(X):
    m3 = 0.0
    n = len(X)
    for i in range(n):
        m3 += (((X[i] - np.mean(X)) ** 3)/ (n - 1))
    return m3

def BootstrapSample(X,n):
    Xstar = []
    for i in range(n):
        j = int(n*np.random.sample(1)[0])
        Xstar.append(X[j])
    return Xstar

#bootstrap básico
def Bootstrap(B,g,X):
    n = len(X)
    Tstar = []
    m3 = 0.0

    for i in range(B):
        Xstar = BootstrapSample(X,n)
        Tstar.append(g(Xstar))

    return Tstar,np.mean(Tstar),np.std(Tstar)

# bootstrap estimando erro medio quadratico
def Bootstrap2(tol, B, g, X):
    n1 = len(X)
    Tstar = []

    for i in range(B):
        Xstar = BootstrapSample(X,n1)
        Tstar.append(g(Xstar))
    n = B
    mx = np.mean(Tstar)
    s2x = np.var(Tstar)
    while s2x > float(n) * (tol ** 2):
        Xstar = BootstrapSample(X, n1)
        nx = g(Xstar)
        Tstar.append(nx)
        nmx = mx + (nx - mx) / float(float(n) + 1)
        ns2x = (1. - 1. / float(n)) * s2x + (float(n) + 1.) * ((nmx - mx) ** 2)
        n += 1
        mx = nmx
        s2x = ns2x

    return Tstar, n


X = [5, 4, 9, 6, 21, 17, 11, 20, 7, 10, 21, 15, 13, 16, 8]

tol = 0.1

# Teste com o Bootstrap básico
# Ts,MTs,STs = Bootstrap(1000,g,X)

se, N = Bootstrap2(tol, 1000, g, X)

mse = np.mean(se)
```

```
print('Erro medio quadratico: ', mse, '\nvar: ' , np.var(se), '\nIterações: ', N)

plt.hist(se,1000, facecolor='green', alpha=0.5)
plt.show()
plt.hist(se,1000, cumulative=True,facecolor='green', alpha=0.5)
plt.show()
```

```
Erro medio quadrático:  34.792743795973564
Var:  4506.43941375221
Iterações:  450645
```





Questão 2: (7.0 pontos)

Considere um sistema de fila simples de um banco com clientes entrando de acordo com um Processo de Poisson homogêneo a uma taxa de 4.0 pessoas por hora. Um cliente só entrará no sistema se tiverem 3 ou menos clientes nele. O tempo de atendimento obedece uma distribuição exponential com taxa de 3.0. Nenhum cliente entra a partir de T = 8. Todos os tempos e taxas são dadas por hora.

- Faça um simulador para estimar a média da quantidade de tempo que um cliente fica no sistema. Usando o critério de parada como sendo uma tolerânica para o tamanho do intervalo de confiança, determine a média, o intervalo de confiança e o número de simulações utilizados para obter tal resultado. Considere a tolerância como sendo tol = 0.01 e o grau de confiança de 95%.
- Faça um simulador para estimar a média do tempo ocioso da caixa no sistema. Usando o critério de parada como sendo uma tolerância para o tamanho do intervalo de confiança, determine a média, o intervalo de confiança e o número de simulações utilizados para obter tal resultado. Considere a tolerência como sendo tol = 0.01 e o grau de confiança de 90%.

In [11]:

```python
# Questão 2
# Sistema de fila simples com entrada de acordo com Processo
# de Poisson Homogêneo e tempo de atendimento de acordo com
# uma função Exponencial
# Dados: Taxa para o Poisson é de 4.0
#        Taxa da Exponencial é de 3.0

import numpy as np
import math



def Exponential(nsamples, rateD):
    x = np.zeros(nsamples)
    u = np.random.sample(nsamples)

    for i in range(nsamples):
        x[i] = - math.log(u[i]) / rateD

    return x

def Exponential_2(nsamples,ratemax):
    x = np.zeros(nsamples)
    u = np.random.sample(nsamples)
    for i in range(nsamples):
        x[i] = - math.log(1.0 - u[i]) / ratemax
    return x


def PoissonHomogeneo(t, ratemax, T):
    N = 0
    tE = []
    while(1):
        Z = Exponential_2(1, ratemax)[0]
        if (t + Z) < (T * 60.0):
            t = t + Z
            N += 1
            tE.append(t)
        if len(tE) > 0:
            return N, tE[0]
    return N, 0.0

# ratemax é 4.0 (até pessoas por hora)
# rateD (3.0) é a taxa respeitada pelo tempo de atendimento
# T = 8

def SimpleQueue(ratemax, rateD, T, isTolFunction):
    t = 0.0
    TA = []
    TD = []
    ta = 0.0
    n = 0         # clientes na fila
    id_cliente = 0
    td = 1.0e+30 # tempo infinito
    clock = 0.0

    numero_clientes , ta = PoissonHomogeneo(t, ratemax, T)
    Atendimento = []
    while(1):
        if ((ta <= td) and (ta < T)): #cliente entra na fila
            t += ta
            TA.append(t)
            n += 1

            if isTolFunction == False:
                print(t * 60.0, ': cliente #', id_cliente, ' entrou na fila..')


            if n == 1:
```

```
                Atendimento.append(id_cliente)
                td = t + Exponential(1, rateD)[0]
                if isTolFunction == False:
                    print(t * 60.0, ': início do atendimento de #', Atendimento[0],'


            if numero_clientes <= 3:  #  apenas se 3 clientes estiverem na fila é que
                numero_clientes, ta = PoissonHomogeneo(t, ratemax, T)
                id_cliente += 1
            else:
                continue

        elif ((td < ta) and (td < T)): # atendimento do cliente é concluído antes de
            t = td
            TD.append(t + td)
            n -= 1
            m = Atendimento.pop(0)

            if isTolFunction == False:
                print(t * 60.0, ': concluído o atendimento de #', m,' no caixa')

            if n == 0:
                td = 1.0e+30
            else:
                td = t + Exponential(1, rateD)[0]

                Atendimento.append(id_cliente - 1)
                if isTolFunction == False:
                    print(t * 60.0, ': início do atendimento de #', Atendimento[0],'


        elif ((min(ta, td) > T) and (n > 0)): # horas extras de atendimento
            while (n > 0):
                t = td
                TD.append(t + td)
                n -= 1
                m = Atendimento.pop(0)
                if isTolFunction == False:
                    print(t * 60.0, ': concluído o atendimento de #', m,' no caixa')

                if (n > 0):
                    td = t + Exponential(1, rateD)[0]
                    Atendimento.append(id_cliente - 1)
                    if isTolFunction == False:
                        print(t * 60.0, ': início do atendimento de #', Atendimento[0



        else:
            media = []

            for i in range(len(TD)):
                media.append((TD[i] - TA[i]) * 60.0)

            return (max(t-T,0.0)),TA, TD, np.mean(media)

ratemax = 4.0 # taxa de quantas pessoas entram na agencia
rateD = 3.0 # quantas pessoas atendidas / hora (3.0)
T = 8.0     # hora que o banco fecha T = 8.0
t , TA, TD, media = SimpleQueue(ratemax, rateD, T, False)

# print(t * 60.0)
# print(TA, len(TA))
# print(TD, len(TD))

print('Tempo a mais em minutos que o banco atendeu =', t * 60.0)
print('TA = ', [n * 60.0 for n in TA])
print('TD = ', [m * 60.0 for m in TD])
```

```
5.3831789895193145 : cliente # 0  entrou na fila..
```

```
5.3831789895193145 : início do atendimento de # 0  no caixa
10.810080537156326 : cliente # 1  entrou na fila..
5.561321159699078 : concluído o atendimento de # 0  no caixa
5.561321159699078 : início do atendimento de # 1  no caixa
46.245381793057625 : cliente # 2  entrou na fila..
46.03320799716541 : concluído o atendimento de # 1  no caixa
46.03320799716541 : início do atendimento de # 2  no caixa
97.50226513745619 : cliente # 3  entrou na fila..
82.87375754604693 : concluído o atendimento de # 2  no caixa
82.87375754604693 : início do atendimento de # 3  no caixa
210.591010866719 : cliente # 4  entrou na fila..
137.561531175475 : concluído o atendimento de # 3  no caixa
137.561531175475 : início do atendimento de # 4  no caixa
150.90696603118974 : concluído o atendimento de # 4  no caixa
436.57488147947987 : cliente # 5  entrou na fila..
436.57488147947987 : início do atendimento de # 5  no caixa
885.9211053015453 : cliente # 6  entrou na fila..
461.7782193251205 : concluído o atendimento de # 5  no caixa
461.7782193251205 : início do atendimento de # 6  no caixa
488.95617508965677 : concluído o atendimento de # 6  no caixa
Tempo a mais em minutos que o banco atendeu = 8.956175089656746
TA =  [5.3831789895193145, 10.810080537156326, 46.245381793057625, 97.50226513745619,
210.591010866719, 436.57488147947987, 885.9211053015453]
TD =  [11.122642319398157, 92.06641599433082, 165.74751509209386, 275.12306235095, 30
1.8139320623795, 923.556438650241, 977.9123501793135]
```

In [15]:

```
# Questão 2
# Item 1: Média de tempo de que o cliente fica no sistema

import numpy as np
import math
import scipy.stats


def SimpleQueueTol_1(tol, alpha, ratemax, rateD, T):
#     X = np.zeros(100)
    Tempos = []

    for i in range(10):
        t, TA, TD, media = SimpleQueue(ratemax, rateD, T, True)
        Tempos.append(media)

    n = 10
    m = np.mean(Tempos)
    s2 = np.var(Tempos)
    zalphaby2 = scipy.stats.norm.ppf(1.0 - alpha / 2.0)

    n = float(n)


    while (2.0*zalphaby2*math.sqrt(s2 /float(n)) >= tol):
        nx, TA, TD, media = SimpleQueue(ratemax, rateD, T, True)
        Tempos.append(media)
        m = np.mean(Tempos)
        nm = m + (nx - m) / (n + 1)
        ns2 = (1.0 - 1.0 / n) * s2 + (n + 1.0) * (nm - m) ** 2
        n += 1
        m = nm
        s2 = ns2

    return m, s2, n
####################################################################################
#
# executar com uma tolerancia de 0.01 é bastante demorado na minha máquina,
#         teste feito com tol = 7.0 apenas para fins de demonstrar o algoritmo
#
####################################################################################
tolerancia = 7.0
alpha = 0.05     # 95 % de confiança
ratemax = 4.0
rateD = 3.0
T = 8.0
media, s2, n = SimpleQueueTol_1(tolerancia, alpha, ratemax, rateD, T)
z = scipy.stats.norm.ppf(1.0 - alpha / 2.0)

i_1 = (media - math.sqrt(s2 / n) * z)
i_2 = (media + math.sqrt(s2 / n) * z)


print('Média de tempo dos clientes no sistema = ', media)

print('Média está no intervalo [', i_1 ,',', i_2, ']')

print('Número de simulações = ', n)
```

```
Média de tempo dos clientes no sistema =  197.37479597715702
Média está no intervalo [ 193.87481057167054 , 200.8747813826435 ]
Número de simulações =  11985.0
```

In [21]:

```python
# Questão 2
# Item 2: Média de tempo ocioso da caixa

import numpy as np
import math
import scipy.stats


def SimpleQueueTol_2(tol, alpha, ratemax, rateD, T):
    X = np.zeros(100)
    for i in range(100):
        t, TA, TD, media = SimpleQueue(ratemax, rateD, T, True)
        X[i] = t

    n = 100
    m = np.mean(X)
    s2 = np.var(X)
    zalphaby2 = scipy.stats.norm.ppf(1.0 - alpha / 2.0)

#     while(2.0 * (s2 / n) * zalphaby2 * zalphaby2 > (tol ** 2) ):
    while (2.0*zalphaby2*math.sqrt(s2 /float(n))) >= tol:
        nx, TA, TD, media = SimpleQueue(ratemax, rateD, T, True)
        nm = m + (nx - m) / (n + 1)
        ns2 = (1.0 - 1.0 / n) * s2 + (n + 1.0) * (nm - m) ** 2
        n += 1
        m = nm
        s2 = ns2
    return m, s2, n


tolerancia = 0.01
alpha = 0.1         # 90 % de confiança
ratemax = 4.0
rateD = 3.0
T = 8.0
m, s2, n = SimpleQueueTol_2(tolerancia, alpha, ratemax, rateD, T)
z = scipy.stats.norm.ppf(1.0 - alpha / 2.0)

i_1 = (m - math.sqrt(s2 / n) * z)
i_2 = (m + math.sqrt(s2 / n) * z)


print('Média de tempo ocioso dos caixas = ', m)

print('Média está no intervalo [', i_1 ,',', i_2, ']')

print('Número de simulações = ', n)
```

```
Média de tempo ocioso dos caixas =  1.8251388407791154
Média está no intervalo [ 1.8201388416827684 , 1.8301388398754623 ]
Número de simulações =  510042
```

In [ ]: