EXERCICIO PROGRAMA 3 - MAP2212 2024

Victor Rocha Cardoso Cruz	11223757
Larissa Aparecida Marques Pimenta Santos	12558620

1 Enunciado

Neste EP, você deve analisar experimentalmente a velocidade de convergência dos procedimentos de integração implementados no EP 2 ao substituir o gerador de números pseudo-aleatórios por um gerador quase-aleatório.

- Como um aprimoramento do EP 2, considere trocar o gerador de números pseudo-aleatórios por um gerador de números quase-aleatórios.
- Suas rotinas de integração por Monte Carlo funcionam melhor? Empiricamente, o quão mais rápidas são as novas rotinas?
- Explique cuidadosamente por que e como você fez suas análises empíricas e chegou à sua conclusão.

2 Desenvolvimento

A partir do código implementado no EP 2, para as diversas variantes do método de Monte Carlo, foi feita a subsitituição das funções de distribuição do scipy.stats (*uniform, beta, gamma e weibull min*), responsáveis pela geração de números pseudo-aleatórios, pelo módulo *Quasi-Monte Carlo* (qmc) da mesma biblioteca, capaz de gerar uma distribuição uniforme quase aleatória. Dentre os modelos de distribuições disponíveis, optou-se pela sequência de Halton, devida a sua baixa variância e elevada uniformidade.

Para simular as distribuições Beta, Gama e Weibull, através da geração quase aleatória, os valores passaram por uma transformação pela função P.P.F, que é a inversa da função P.P.F, característica de cada distribuição. Além disso, para Gama e Weibull, que geram valores maiores que um, foi feita a normalização para os limites de integração desejados, conforme já implementado no EP 2.

3 Conclusão

Os códigos do EP 2 e 3 foram submetidos a uma série de testes, promovendo os valores de variância para cada método, bem como a duração (em segundos) de toda a execução. Nota-se que ambos os geradores, pseudo e quasi, possuem uma variância baixa e são convergentes entre si, gerando uma quantidade de execuções n próximos. Portanto, os dois métodos são equivalentes, não sendo possível definir um como superior ao outro nesse quesito.

Entretanto, dentro da execução para dimensões maiores, como $n = 100000$, percebe-se uma clara vantagem dos números pseudo aleatórios, uma vez que a velocidade de execução é 3 vezes menor que o gerador quase aleatório. Em um dos testes, a implementação de números pseudo aleatórios levou $35,76$ s para conclusão, enquanto o quase aleatórios demorou $233,03$ s. Essa grande diferença de duração concentra-se no MMC Amostragem por Importância, especialmente para as distribuições Gamma e Weibull, elas chegam a levar 10x mais tempo para finalizar. Para os demais métodos de distribuição uniforme e Beta, há uma equivalência entre os dois, assim como se enxergou nas variâncias.

4 Código em Python

```
'''

C DIGO DO EP 3

'''
import numpy as np
from scipy.stats import uniform, beta, gamma, weibull_min, qmc
import time

# Defini  o dos par metros a e b
a = 0.564263989  # RG
b = 0.51197762809  # CPF

# Fun  o alvo f(x)
def f(x):
    fx = np.exp(-a * x) * np.cos(b * x)
    return fx

print("*** EP 3 INICIANDO ***")
print()
print("*** EXPERIMENTO PILOTO ***")
# EXPERIMENTO PILOTO #
# M todo Cru para defini  o da integral esperada
# Defini  o a largura da distribui  o
n = 1000000
# Amostragem usando o m todo de Halton
halton_samples_x = qmc.Halton(d=1).random(n)
halton_samples_y = f(halton_samples_x)
# Aproxima  o atrav s de Distribui  o Halton
AreaPilot = np.mean(halton_samples_y)
print(f" rea  Piloto {AreaPilot}")
print()

```

```
# Defini  o da margem de erro e quantil da normal
epsilon = 0.05 / 100 * AreaPilot
z_alfa = 1.64

total_time = 0

# ******************** M TODO DE MONTE CARLO CRU
    ******************** #
print("*** M TODO DE MONTE CARLO CRU ***")
# Distribui  o Uniforme
n = 100000
start_time = time.time()
halton_samples_x = qmc.Halton(d=1).random(n)
halton_samples_y = f(halton_samples_x)
# Aproxima  o atrav s de Distribui  o Uniforme
area_cru = np.mean(halton_samples_y)
variancia = np.sum((halton_samples_y - area_cru)**2)/n
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
end_time = time.time()
execution_time = end_time - start_time
total_time += execution_time
print(f" rea  estimada: {area_cru}")
print(f"Vari ncia do estimador: {variancia}")
print(f"Erro padr o do estimador: {erro_padrao:.10f}")
print(f"Largura da Distr.: {n}")
print()
n = round(((z_alfa * np.sqrt(variancia))/epsilon)**2)
print(f"Valor de n calculado: {n}")
print("Tempo de execu  o:", execution_time, "segundos \n")
print()

#
    ************************************************************************
     #
```

```
# ******************** M TODO DE MONTE CARLO 'HIT OR MISS' ******************** #
print("*** M TODO DE MONTE CARLO 'HIT OR MISS' ***")
# Aproxima  o atrav s de Distribui  o Uniforme
n = 100000
PontosDentro = 0
start_time = time.time()
hitmiss_samples_x = qmc.Halton(d=1).random(n)
hitmiss_samples_y = qmc.Halton(d=1).random(n)
for i in range(n):
    xi = hitmiss_samples_x[i]
    yi = hitmiss_samples_y[i]
    # Checa se o ponto sorteado est  abaixo da curva
    if yi <= f(xi):
        PontosDentro += 1
area_hitmiss = PontosDentro/n
# Variancia do estimador
variancia = area_hitmiss*(1 - area_hitmiss)/n
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
end_time = time.time()
execution_time = end_time - start_time
total_time += execution_time
print(f" rea  estimada: {area_hitmiss}")
print(f"Vari ncia do estimador: {variancia:.10f}")
print(f"Erro padr o do estimador: {erro_padrao:.10f}")
print(f"Largura da Distr.: {n}")
print()
n = round(((z_alfa * np.sqrt(variancia))/epsilon)**2)
print(f"Valor de n calculado: {n}")
print("Tempo de execu  o:", execution_time, "segundos")
print()

# ***************************************************************************
```

```
 #
# ******************** M TODO DE MONTE CARLO AMOSTRAGEM POR
  IMPORT NCIA ******************** #
print("*** M TODO DE MONTE CARLO AMOSTRAGEM POR IMPORT NCIA ***
  ")
# Distribui  o Uniforme
n = 100000
pesos = []
pdfx = []
start_time = time.time()
uniform_samples_x = qmc.Halton(d=1).random(n)
xi = uniform_samples_x
# Fun  o Densidade de probabilidade da Distribui  o Uniforme
pdfx = uniform.pdf(xi)
yi = f(xi)
# Calculo dos pesos, como o quociente entre os valores f(xi) da
  fun  o pela FDP da distribui  o uniforme
pesos = yi/pdfx
area_uniform = np.mean(np.array(pesos))
# Variancia do estimador
variancia = np.sum(np.array(pdfx)*(np.array(pesos) - area_uniform
  )**2)/n
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
end_time = time.time()
execution_time = end_time - start_time
total_time += execution_time
print(f"Uniforme:  rea  estimada: {area_uniform}")
print(f"Uniforme: Vari ncia do estimador: {variancia:.10f}")
print(f"Uniforme: Erro padr o do estimador: {erro_padrao:.10f}")
print(f"Uniforme: Quantidade de itera  es: {n}")
print()
n = round(((z_alfa * np.sqrt(variancia))/epsilon)**2)
print(f"Valor de n calculado: {n}")
print("Tempo de execu  o:", execution_time, "segundos")
```

```
print()

# Distribui o Beta
# Implementa o do m todo de Halton para gerar amostras quase aleat rias
pesos = []
pdfx = []
n = 100000
start_time = time.time()
beta_samples_x = qmc.Halton(d=1).random(n)
xi = beta_samples_x
# Transforma o da vari vel quase-aleat ria uniforme na distribui o Beta
xi = beta.ppf(xi, a = 1, b = 1)
# Obten o da fun o densidade de probabilidade, para a distribui o Beta
pdfx = beta.pdf(xi, a = 1, b = 1)
yi = f(xi)
pesos = yi/pdfx
area_beta = np.mean(pesos)
# Variancia do estimador
variancia = np.sum(np.array(pdfx)*(np.array(pesos) - area_beta)**2)/n
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
end_time = time.time()
execution_time = end_time - start_time
total_time += execution_time
print(f"Beta:  rea estimada: {area_beta}")
print(f"Beta: Vari ncia do estimador: {variancia:.10f}")
print(f"Beta: Erro padr o do estimador: {erro_padrao:.10f}")
print(f"Beta: Quantidade de itera es: {n}")
print()
n = round(((z_alfa * np.sqrt(variancia))/epsilon)**2)
print(f"Valor de n calculado: {n}")
```

```
print("Tempo de execu  o:", execution_time, "segundos")
print()


# Distribui  o Gamma
# Normaliza  o da distribui  o Gamma a partir da fun  o cumulativa nos pontos de interesse
cdf_lower = gamma.cdf(0, a=1, scale=2)
cdf_upper = gamma.cdf(1, a=1, scale=2)
const_normalizacao = 1 / (cdf_upper - cdf_lower)
# Fun  o que gera valores quase aleat rios para a distribui  o Gama
def generate_n_gamma_samples(num_samples):
    samples_no_intervalo = []
    while len(samples_no_intervalo) < num_samples:
        # Gerar amostras aleat rias pela distribui  o Gama
        gamma_samples_x = qmc.Halton(d=1).random(1)
        gamma_samples_x = gamma.ppf(gamma_samples_x, a= 1, scale = 2)
        # Checa se a amostra est  dentro do intervalo
        if 0 <= gamma_samples_x <= 1:
            samples_no_intervalo.append(gamma_samples_x)
    return np.array(samples_no_intervalo)

pesos = []
pdfx = []
n = 100000
start_time = time.time()
gamma_samples_x = generate_n_gamma_samples(n)
xi = gamma_samples_x
# Obten  o da fun  o densidade de probabilidade, para a distribui  o Gamma, e normalizada   0 e 1
pdf_value = gamma.pdf(xi, a = 1, scale = 2)
pdfx = pdf_value * const_normalizacao
```

```
yi = f(xi)
pesos = yi/pdfx
area_gama = np.mean(pesos)
# Variancia do estimador
variancia = np.sum(np.array(pdfx)*(np.array(pesos) - area_gama)**2)/n
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
end_time = time.time()
execution_time = end_time - start_time
total_time += execution_time
print(f"Gama:  rea estimada: {area_gama}")
print(f"Gama: Vari ncia do estimador: {variancia}")
print(f"Gama: Erro padr o do estimador: {erro_padrao:.10f}")
print(f"Gama: Quantidade de itera  es: {n}")
print()
n = round(((z_alfa * np.sqrt(variancia))/epsilon)**2)
print(f"Valor de n calculado: {n}")
print("Tempo de execu  o:", execution_time, "segundos")
print()

# Distribui  o Weibull
# Normaliza  o da distribui  o Weibull a partir da fun  o cumulativa nos pontos de interesse
cdf_lower = weibull_min.cdf(0, c=1)
cdf_upper = weibull_min.cdf(1, c=1)
const_normalizacao = 1 / (cdf_upper - cdf_lower)
# Fun  o que gera valores quase aleat rios para a distribui  o Weibull
def generate_n_weibull_samples(num_samples):
    samples_no_intervalo = []
    while len(samples_no_intervalo) < num_samples:
        # Gerar amostras aleat rias pela distribui  o Gama
        weibull_samples_x = qmc.Halton(d=1).random(1)
        weibull_samples_x = weibull_min.ppf(weibull_samples_x, c
```

```
           =1)
        # Checa se a amostra est  dentro do intervalo
        if 0 <= weibull_samples_x <= 1:
            samples_no_intervalo.append(weibull_samples_x)
    return np.array(samples_no_intervalo)

pesos = []
pdfx = []
n = 100000
start_time = time.time()
weibull_samples_x = generate_n_weibull_samples(n)
xi = weibull_samples_x
# Obten  o da fun  o densidade de probabilidade, para a distribui  o Weibull, e normalizada   0 e 1
pdf_value = weibull_min.pdf(xi, c=1)
pdfx = pdf_value * const_normalizacao
yi = f(xi)
pesos = yi/pdfx
area_weibull = np.mean(pesos)
# Variancia do estimador
variancia = np.sum(np.array(pdfx)*(np.array(pesos) - area_weibull)**2)/n
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
end_time = time.time()
execution_time = end_time - start_time
total_time += execution_time
print(f"Weibull:  rea estimada: {area_weibull}")
print(f"Weibull: Vari ncia do estimador: {variancia:.10f}")
print(f"Weibull: Erro padr o do estimador: {erro_padrao:.10f}")
print(f"Weibull: Quantidade de itera  es: {n}")
print()
n = round(((z_alfa * np.sqrt(variancia))/epsilon)**2)
print(f"Valor de n calculado: {n}")
print("Tempo de execu  o:", execution_time, "segundos")
```

```
print()

# ************************************************************************ #
# ******************** M TODO DE MONTE CARLO VARI VEIS DE CONTROLE ******************** #
print("*** M TODO DE MONTE CARLO VARI VEIS DE CONTROLE ***")
# Defini  o dos limites de integra  o para a fun  o f e
limInfer, limSuper= 0, 1
# Fun  o polinominal   que aproxima-se   curva da fun  o f(x), encontrada a partir dos pontos (1, f(1)) e (0, f(0))
def phi(x):
    phix = ((f(1)-f(0))/(1 - 0))*x + 1
    #phix = 1 - 0.564263989*x + 0.02814*x**(2) + 0.04400*x**(3)-0.01377*x**(4)
    return phix
# Fun  o primitiva de
def phiPrim(x):
    phiPrim = x + ((f(1)-f(0))/(1 - 0))*x**(2)/2
    #phiPrim = x - 0.564263989*x**(2)/2 + 0.02814*x**(3)/3 + 0.04400*x**(4)/4 - 0.01377*x**(5)/5
    return phiPrim

termo = 0
fxn, phixn = [], []
n = 100000
start_time = time.time()
controlv_samples_x = qmc.Halton(d=1).random(n)
#xi = controlv_samples_x[i]
# Armezando os valores de f(xi) e g(xi) para o c lculo final da vari ncia
fxn = f(controlv_samples_x)
phixn = phi(controlv_samples_x)
```

```
# Termo do somat rio para calculo de gama chapeu
termo = np.sum(f(controlv_samples_x) - phi(controlv_samples_x) + (phiPrim(limSuper)-phiPrim(limInfer)))
area_control = termo/n
var_f = np.var(np.array(fxn))
var_phi = np.var(np.array(phixn))
correlacao = np.cov(fxn, phixn, rowvar=False)[0,1]
variancia = (1/n)*(var_f + var_phi - 2*correlacao*np.sqrt(var_f)*np.sqrt(var_phi))
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
end_time = time.time()
execution_time = end_time - start_time
total_time += execution_time
print(f" rea  estimada:: {area_control}")
print(f"Vari ncia do estimador: {variancia:.10f}")
print(f"Erro padr o do estimador: {erro_padrao:.10f}")
print(f"Quantidade de itera  es: {n}")
print()
n = round(((z_alfa * np.sqrt(variancia))/epsilon)**2)
print(f"Valor de n calculado: {n}")
print("Tempo de execu  o:", execution_time, "segundos")
print()
print("Tempo total de execu  o:", total_time, "segundos")
```