EXERCICIO PROGRAMA 2 - MAP2212 2024

Victor Rocha Cardoso Cruz

11223757

1 Enunciado

Neste EP, você deve implementar um programa para estimar π.

- Encontre, em seu a ambiente computacional, funções para gerar amostras de variáveis aleatórias com distribuições Uniforme, Beta, Gamma e Weibull.
- Implemente as quatro variantes de integração por Monte Carlo estudadas para integrar a função $f(x) = e^{-ax}cos(bx)$ em [0,1], onde $a = 0.RG$, $b = 0.CPF$, e RG e CPF correspondem aos dígitos de seus documentos de identificação.
- Escolha parâmetros para cada distribuição amostral via inspeção visual ou outro método que quiser. Escolha uma função polinomial para a variável de controle. Escolha n para obter um erro relativo $|\gamma - \widehat{\gamma}|/\gamma < 0.0005$ (sem conhecer γ!).

2 Introdução

Para esse exercício programa, foram utilizados as quatro variantes do Método de Monte Carlo (MMC) para estimar o valor da integral definida γ de $f(x)$, no intervalo [0,1]. Os métodos de MMC aplicados são o Simples (ou Cru), Acerto ou Erro (*Hit or Miss*), Amostragem de Importância (*Important Sampling*) e Variáveis de Controle. A partir dos procedimentos computacionais estabelecidos, os métodos foram submetidos a grandes larguras de distribuições, com o objetivo verificar o valor da integral estimada e precisar sua variância. Desse modo, foi possível definir o número de iterações esperado para se obter uma margem de erro de $0,05$ %.

3 Método de Monte Carlo Simples

Para a variante simples do MMC, utilizou-se a distribuição uniforme da biblioteca numpy (numpy.radom.uniform), que gera valores psudoaleatórios no intervalo [0,1]. Com base nas amostras, foram calculados, a partir da função f, os pontos correspondentes sob a curva. Sendo assim, o valor aproximado da integral nesse intervalo é a média entre áreas dos retângulos de altura $f(x_i)$ e base 1 (limitado pelo intervalo de interesse).

4 Método de Monte Carlo Acerto ou Erro

O pressuposto principal desse método é o sorteio aleatório de pontos (x, y) e a verificação se os mesmos residem abaixo da curva da função $f(x)$. Dessa forma, foi utilizado a Distribuição Uniforme (numpy.radom.uniform) para sortear as coordenadas e, caso o y_i seja menor ou igual à $f(x_i)$, o ponto é contabilizado. A estimativa da área é dada pela somatória desses pontos dividido pelo volume de pontos sorteados n.

5 Método de Monte Carlo Amostragem de Importância

Essa variante é comumente aplicada para avaliar a relevância das amostras de uma certa distribuição na obtenção da variável de interesse, nesse caso, a integral definida da função $f(x)$. A estimativa é dada pelo quociente entre os diversos $f(x_i)$ calculados e a Função Densidade de Probabilidade (F.D.P.) de cada distribuição. Para tal, foram utilizadas 4 distribuições diferentes (Uniforme, Beta, Gama e Weibull), tendo seus parâmetros ajustados visualmente, através do *plot* de seu gráfico, para melhor se adequarem à $f(x)$.

5.1 Distribuição Uniforme

A distribuição uniforme gera pontos pseudoaletórios no intervalo [0,1] a partir de sua F.D.P, definida conforme equação 1 abaixo, sendo a e b os limites do intervalo de interesse. Assim, os valores de x_i foram obtidos e $f(x_i)$ calculados a partir das amostras da distribuição. Os pesos são definidos como o quociente de $f(x_i)$ pela F.D.P da distribuição no mesmo ponto x_i, e a estimativa da área da integral é encontrada por meio da média dos pesos.

$$f(x; a, b) = \frac{1}{b - a} \tag{1}$$

5.2 Distribuição Beta

Para a distribuição Beta, utilizou-se o mesmo método que a Uniforme, uma vez que ambas as distribuições sorteam pontos automaticamente no intervalo [0,1]. Porém, a F.D.P de Beta possui uma definição diferente, conforme equação 2, tendo os parâmetros α e β controlando o seu formato. Por meio da análise visual, representando a F.D.P de Beta através do *plot* do Pyhton, os parâmetros foram fixados como $\alpha = 1$ e $\beta = 1$, pois com eles a distribuição possui maior verossimilhança com a curva da função $f(x)$.

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta) x^{\alpha-1} (1 - x)^{\beta-1}}{\Gamma(\alpha)\Gamma(\beta)} \tag{2}$$

5.3 Distribuição Gama

A distribuição Gamma, ao contrário das anteriores, está definida e retorna valores de zero a ∞ , tornando-se necessário normalizá-la. A Função Distribuição Acumulada (F.D.A) pode ser utilizada nesse momento, definindo a constante de probabilidade z (equação 3) como o inverso da diferença entre a função F.D.A calculada nos limites do intervalo. Dessa forma, pode-se utilizar a F.D.P normalizada através de z no cálculo via *Important Sampling*.

$$z = \frac{1}{f.d.a(1) - f.d.a(0)} \tag{3}$$

A distribuição possui um parâmetro a, que controla suas condições de contorno, e ele foi encontrado a partir da análise visual da F.D.P de Gama dentro do intervalo desejado, assim como foi feito anteriormente com a Beta.

5.4 Distribuição Weibull

A distribuição Weibull apresenta uma situação parecida com a Gamma, sendo requisito a parametrização de suas amostras ao intervalo [0,1]. Assim, a F.D.A de Weibull foi aplicada à equação 3. A determinação dos parâmetros foi feita da mesma forma que as demais distribuições.

6 Método de Monte Carlo Variáveis de Controle

O método das Variáveis de Controle resume-se à utilização de uma função $\phi(x)$ com curva similiar a $f(x)$. Por meio dos valores da função $f(x)$ no intervalo, obtem-se as coordenadas (x, y) dos pontos e pode-se traçar uma reta que passe pelos mesmos (equação 5). Verifica-se, graficamente, a afinidade entre as funções $f(x)$ e $g(x)$, garantindo que o estimador aplicado terá muita proximidade com o valor esperado.

$$\phi(x) = \frac{(f(1) - f(0))}{(1 - 0)} x + 1 \tag{4}$$

$$\phi(x) = 0,5041516712993014x + 1 \tag{5}$$

7 Código em Python

```python

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

# 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 2 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
uniform_samples_x = uniform.rvs(size=n, loc=0, scale=1)
uniform_samples_y = f(uniform_samples_x)
# Aproxima  o atrav s de Distribui  o Uniforme
AreaPilot = np.mean(uniform_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

# ******************** M TODO DE MONTE CARLO CRU
    ******************** #
```

```
print("*** M TODO DE MONTE CARLO CRU ***")
# Distribui  o Uniforme
n = 100000
uniform_samples_x = uniform.rvs(size=n, loc=0, scale=1)
uniform_samples_y = f(uniform_samples_x)
# Aproxima  o atrav s de Distribui  o Uniforme
area_cru = np.mean(uniform_samples_y)
variancia = np.sum((uniform_samples_y - area_cru)**2)/n
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
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()

# ************************************************************************
    #
# ******************** 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
for i in range(n):
    xi = uniform.rvs()
    yi = uniform.rvs()
    # 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)
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()

# ************************************************************************ #
# ******************** 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 = []
for i in range(n):
    xi = uniform.rvs()
    # Fun  o Densidade de probabilidade da Distribui  o Uniforme
    pdf_value = uniform.pdf(xi)
    pdfx.append(pdf_value)
    yi = f(xi)
    # Calculo dos pesos, como o quociente entre os valores f(xi) da fun  o pela FDP da distribui  o uniforme
    peso = yi/pdf_value
    pesos.append(peso)
```

```
    i+=1
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)
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()

# Distribui  o Beta
# Calculo dos pesos, como o quociente entre os valores f(xi) da fun  o pela FDP da distribui  o Beta
pesos = []
pdfx = []
n = 100000
for i in range(n):
    xi = beta.rvs(size=1, a = 1, b = 1)
    # Obten  o da fun  o densidade de probabilidade, para a distribui  o Beta
    pdf_value = beta.pdf(xi, a = 1, b = 1)
    pdfx.append(pdf_value)
    yi = f(xi)
    peso = yi/pdf_value
    pesos.append(peso)
    i+=1
area_beta = np.mean(pesos)
# Variancia do estimador
```

```python
variancia = np.sum(np.array(pdfx)*(np.array(pesos) - area_beta)**2)/n
erro_padrao = np.sqrt(variancia)/np.sqrt(n)
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()

# 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 criada para obter apenas valores entre 0 e 1
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 = gamma.rvs(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
for i in range(n):
    xi = generate_n_gamma_samples(1)
```

```
    # 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)
    pdf_no_intervalo = pdf_value * const_normalizacao
    pdfx.append(pdf_no_intervalo)
    yi = f(xi)
    peso = yi/pdf_no_intervalo
    pesos.append(peso)
    i+=1
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)
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()

# 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 criada para obter apenas valores entre 0 e 1
def generate_n_weibull_samples(num_samples):
    samples_no_intervalo = []
    while len(samples_no_intervalo) < num_samples:
        # Gerar amostras aleat rias pela distribui  o Weibull
```

```python
        weibull_samples_x = weibull_min.rvs(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
for i in range(n):
    xi = generate_n_weibull_samples(1)
    # 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)
    pdf_no_intervalo = pdf_value * const_normalizacao
    pdfx.append(pdf_no_intervalo)
    yi = f(xi)
    peso = yi/pdf_no_intervalo
    pesos.append(peso)
    i+=1
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)
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()

```

```
# ************************************************************************ #
# ******************** 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
for i in range(n):
    xi = uniform.rvs()
    # Armezando os valores de f(xi) e g(xi) para o c lculo final da vari ncia
    fxn.append(f(xi))
    phixn.append(phi(xi))
    # Termo do somat rio para calculo de gama chapeu
    termo += f(xi) - phi(xi) + (phiPrim(limSuper)-phiPrim(limInfer))
```

```
    i+=1
area_control = termo/n
var_f = np.var(np.array(fxn))
var_phi = np.var(np.array(phixn))
correlacao = np.cov(fxn, phixn)[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)
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()
```