MAP2212 - Laboratório de Computação e Simulação

Relatório - EP03

Vítor Garcia Comissoli - 11810411

1 Introdução

O objetivo desse EP é a refacção do EP02 só que utilizando geradores de números quasi-random ao invés de geradores de números pseudo-aleatórios nos quatro métodos de Monte Carlo (**Crude**, **Hit or Miss**, **Control Variate** e **Importance Sampling**).

Além disso, deve-se comparar a diferenças nos valores obtidos para **n** nos quatro métodos, para que a estimativa de γ obtida tenha um erro de, no máximo, 0.05% de γ, com 95% de confiança, ou seja, que fique dento do Intervalo de Confiança $[\gamma \cdot (1 - 0.0005), \gamma \cdot (1 + 0.0005)]$ com $\alpha = 0.95$.

Geradores quasi-random são aqueles que, ao contrário dos pseudo-aleatórios, seguem um certo padrão sistemático durante a geração dos pontos, ou seja, são geradores que geram números pouco discrepantes entre sí, o que costuma levar a um melhor desempenho quando utilizados em casos como o desse EP (estimações que tendem a um valor específico, nesse caso que tendem a γ).

Vale ressaltar também que, neste EP, deve-se tratar o valor da integral γ como desconhecido, então todos os testes devem ser realizadas com um estimador qualquer para γ ($\hat{\gamma}$).

2 Discussão Teórica

Como esse EP é um complemento do EP02, a maior parte da discussão teórica seria a mesma, por isso optei por não me repetir. Em suma, temos que os valores de **n** calculados para os quatro métodos no caso pseudo-aleatório são:

Para **Crude**, $n = 4502884$
Para **Hit or Miss**, $n = 14853316$
Para **Control Variate**, $n = 100000$
Por fim, para **Importance Sampling**, $n = 3541924$

Como o objetivo desse EP é a subistituição dos geradores númericos pseudo-aleatórios por geradores quasi-random, haverá uma discrepância nos valores de **n** necessários para que se atinja a precisão de $\gamma \cdot (1+0.0005)]$ com $\alpha = 0.95$. Sabendo disso, Devemos encontrar os novos valores de **n** para que possamos analisar a diferença entre cada um dos valores de **n** para cada um dos métodos.

Os novos valores de n foram encontrados de forma experimental, ou seja, os valores de n de EP02 foram sendo gradativamente reduzidos (cada método foi individualmente observado e testado para que se encontra-se o valor de n específico para o mesmo.) até que não se enquadrassem mais no intervalo solicitado, decidindo-se assim que o novo valor de n seja o menor valor de n que mostrou-se coerente a precisão solicitada em cada um dos métodos.

Os testes de intervalo foram realizados por meio de uma função main() que chama as funções de cada um dos métodos **n** vezes e retorna a razão das estimativas para γ que se encontram dentro do intevalo de confiança sobre o número total de estimativas realizadas, sendo que o intervalo utilizado pela mesma foi obtido através do uso de $\hat{\gamma} = 0.74391$ (proveniente da função control_variates() para um n = 1000000).

Disso, chegamos em quatro novos valores para n, um para cada método, sendo eles:

Para **Crude**, novo $n = 700000$
Para **Hit or Miss**, novo $n = 6000000$
Para **Control Variate**, novo $n = 750$
Por fim, para **Importance Sampling**, novo $n = 150000$

3 Discussão do Código

Os prints abaixo do código estão comentados linha por linha, explicitando todos os processos tomados.

```python
def f(x):
    """
    Esta funcao deve receber x e devolver f(x), como especifcado no enunciado
    Escreva o seu codigo nas proximas linhas
    """

    func = np.exp(-A*x)*np.cos(B*x) # Calcula o valor de f(x) para o x recebido na função e usando A e B como especificados no enunciado

    return func # Retorna o valor obtido


def crude(Seed = None):
    random.seed(Seed)
    """
    Esta funcao deve retornar a sua estimativa para o valor da integral de f(x)
    usando o metodo crude
    Escreva o seu codigo nas proximas linhas
    """

    n = 700000 # N obtido experimentalmente para que IC seja [+- gamma_chapeu * 0.005] com alpha 0.95

    x_array = (ch.Uniform(0, 1)).sample(n) # Array de coordenadas X uniformemente quasi-random entre 0 e 1

    valor = f(x_array) # Array com os valores de F(x) para o X obtido acima
    soma = sum(1 * valor) # Soma as áreas do retângulo de altura f(x) e largura 1
    gamma = soma/n # É a estimativa para o valor da integral de f(x)

    return gamma # Retorna a estimativa
```

Figura 1: Código e comentários das funções f(x) e Crude

```python
def hit_or_miss(Seed = None):
    random.seed(Seed)
    """
    Esta funcao deve retornar a sua estimativa para o valor da integral de f(x)
    usando o metodo hit or miss
    Escreva o seu codigo nas proximas linhas
    """

    n = 6000000 # N obtido experimentalmente para que IC seja [+- gamma_chapeu * 0.005] com alpha 0.95

    x_array = (ch.Uniform(0, 1)).sample(n) # Array de coordenadas X uniformemente quasi-random entre 0 e 1
    y_array = (ch.Uniform(0, 1)).sample(n) # Array de coordenadas Y uniformemente quasi-random entre 0 e 1

    dentro = sum(y_array <= f(x_array)) # Testa e soma as coordenadas Y que são menores iguais aos valores de f(x), o que implica que Y está dentro da área de gamma

    gamma = dentro/n # É a estimativa para o valor da integral de f(x)

    return gamma # Retorna a estimativa
```

Figura 2: Código e comentários da função Hit or Miss

```python
def control_variate(Seed = None):
    random.seed(Seed)
    """
    Esta funcao deve retornar a sua estimativa para o valor da integral de f(x)
    usando o metodo control variate
    Escreva o seu codigo nas proximas linhas
    """

    n = 750 # N obtido experimentalmente para que IC seja [+- gamma_chapeu * 0.005] com alpha 0.95

    x_array = (ch.Uniform(0, 1)).sample(n) # Array de coordenadas X uniformemente quasi-random entre 0 e 1

    soma = sum(f(x_array) - h(x_array) + (3/4)) # Soma todos os resultados de f(x) + h(x) + integral definida em [0,1] de h(x) (= 3/4)

    gamma = soma/n # É a estimativa para o valor da integral de f(x)

    return gamma # Retorna a estimativa


def h(x): # Função que se assemelha muito ao comportamento de f(x) no intervalo [0,1]

    func = (- 0.5 * x) + 1 # Calcula o valor de g(x) para o x recebido na função

    return func # Retorna o valor obtido
```

Figura 3: Código e comentários da função Control Variate

```python
def importance_sampling(Seed = None):
    random.seed(Seed)
    """
    Esta funcao deve retornar a sua estimativa para o valor da integral de f(x)
    usando o metodo importance sampling
    Escreva o seu codigo nas proximas linhas
    """

    n = 150000 # N obtido experimentalmente para que IC seja [+- gamma_chapeu * 0.005] com alpha 0.95

    x_array = (ch.Beta(1, 1.2)).sample(n) # Array de coordenadas X quasi-random entre 0 e 1 com fdp Beta(1,1.2)

    y_array = beta.pdf(x_array,1,1.2) # Um Array de g(x) que é uma fdp de Beta(1,1.2)

    soma = sum(f(x_array)/y_array) # Soma o valor de todas as f(x) / g(x)

    gamma = soma/n # É a estimativa para o valor da integral de f(x)

    return gamma # Retorna a estimativa
```

Figura 4: Código e comentários da função Importance Sampling

Além disso, vale ressaltar que foi-se utilizada a biblioteca **chaospy** para a obtenção das coordenadas quasi-random nos quatro métodos.

4 Resultado

O teste dos resultados foi realizado pela criação de uma função main() que chama as funções dos quatro métodos **n** vezes e imprime a razão das estimativas para γ que se encontram dentro do intevalo de confiança (intervalo esse obtido através do uso de $\hat{\gamma} = 0.74391$, proveniente da função control_variates() para um n = 1000000) sobre o número total de estimativas realizadas, além de apresentar a média amostral dessas **n** saídas e o tempo total gasto.

Para o método Crude, cujo valor de **n** era 4502884 para geradores pseudo-aleatórios (EP02), obteve-se experimentalmente para geradores quasi-random $n = 700000$, o que demonstra a necessidade de um n aproximadamente 6.5 vezes menor para que se atinja uma precisão semelhante. Na main(), para $n = 1000$, essa razão foi de 0.964, com média $\mu = 0.74391$. O tempo gasto para um loop de $n = 1000$ foi de, aproximadamente, 110.77 segundos (1.85 min).

Já para o método Hit or Miss, cujo valor de **n** era 14853316 para geradores pseudo-aleatórios (EP02), obteve-se experimentalmente para geradores quasi-random $n = 6000000$, o que demonstra a necessidade de um n aproximadamente 2.5 vezes menor para que se atinja uma precisão semelhante. Na main(), para $n = 100$, essa razão foi de 0.97, com média $\mu = 0.74391$. O tempo gasto para um loop de $n = 100$ foi de, aproximadamente, 1645.25 segundos (27.42 min).

Para o método Control Variate, cujo valor de **n** era 100000 para geradores pseudo-aleatórios (EP02), obteve-se experimentalmente para geradores quasi-random $n = 750$, o que demonstra a necessidade de um n aproximadamente 133 vezes menor para que se atinja uma precisão semelhante. Na main(), para $n = 10000$, essa razão foi de 0.9564, com média $\mu = 0.74392$. O tempo gasto para um loop de $n = 10000$ foi de, aproximadamente, 11.80 segundos (0.20 min).

Por fim, para o método Important Sampling, cujo valor de **n** era 3541924 para geradores pseudo-aleatórios (EP02), obteve-se experimentalmente para geradores quasi-random $n = 150000$, o que demonstra a necessidade de um n aproximadamente 23.6 vezes menor para que se atinja uma precisão semelhante. Na main(), para $n = 1000$, essa razão foi de 0.972, com média $\mu = 0.74392$. O tempo gasto para um loop de $n = 1000$ foi de, aproximadamente, 467.78 segundos (7.80 min).

Todos os valores apresentados acima mostram-se coerentes aos parâmetros de acurácia $\alpha = 0.95$ e $\epsilon = \gamma \cdot 0.0005$, estipulados pelo enunciado do EP.

Com isso podemos concluir que o resultado obtido foi dentro do esperado.

5 Conclusão

Como já dito previamente, todos o métodos se mostraram coerentes aos parâmetros de acurácia de $\alpha = 0.95$ e $\epsilon = \gamma \cdot 0.0005$. Contudo, existem outros fatores que podem ser analisados.

Analisando quanto a performance computacional, não faz sentido comparar os métodos do EP03 com os mesmos no EP02, visto que para esse EP otimizei todos os quatro métodos por meio da vetorização e uso de arrays, enquanto o anterior se utiliza-va somente de "for _ in range(_)", o que faz com que todos os métodos desse EP se mostrem mais rápidos que suas contrapartidas no EP anterior. Faz sentido porém, comparar a performance de cada um dos quatro métodos desse EP entre sí. Temos que o método de Hit or Miss se mostrou muito lento e custoso quando comparado com os outros 3 métodos, seguido pelo método Importance Sampling, o método Crude e, por fim, o método Control Variate, que se mostrou o mais eficiente.

Por fim, quanto ao tamanho do **n** encontrado, pode-se perceber que todos os valores de n para os métodos do EP03 foram menores que seus respectivos n obtidos no EP02, o que mostra que o uso de geradores de números quasi-random mostra-se mais eficaz nesse tipo de aplicação que o uso de geradores pseudo-aleatórios comúns, já que os quasi-random convergiram mais rapidamente para o valor de γ. Quando comparamos o tamanho dos valores de n entre os modelos do próprio EP03, temos que o método Hit or Miss necessitou do maior valor de n, seguido pelo método Crude, pelo método Importance Sampling e, por fim, pelo método Control Variate, que foi o que utilizou o menor valor de n dentre os quatro.