Exercício Programa 2

Gabriel Tetsuo Haga

Maio 2021

1 Problema

A integral é uma ferramenta matemática muito utilizada em diversas áreas do conhecimento. Atualmente, é conhecido a primitiva de várias funções, ou seja, é possível determinar analiticamente a integral definida de várias funções. Contudo ainda há certa limitação, o que faz necessário recorrer há métodos estocástico, como os métodos de Monte Carlo.

Assim pede-se para estimar, considerando que não é conhecido o valor analítico, a seguinte integral:

$$\gamma_1 = \int_0^1 e^{-ax} \cos{(bx)} \mathrm{d}x \tag{1}$$

Sendo $a = 0.n_{RG}$, e n_{RG} é o número do RG do aluno, no caso é 54.114.733-x, o x será desconsiderado, logo $a = 0.54114733$. Já o $b = 0.n_{CPF}$, sendo o n_{CPF}, o número do CPF do aluno, no caso é 424.900.238-10, logo $b = 0.42490023810$

Pede-se para fazer a estimativa para os quatro métodos de Monte Carlo apresentados na aula A2 dessa disciplina. Sendo eles o método de Monte Carlo Cru, o método Hit or Miss, o método Importance Sampling e o método Control Variate. Na próxima seção apresenta-se de forma simplificada esses métodos. E explica-se a implementação desses métodos em códigos na linguagem ***Python***.

2 Métodos

Antes de explicar os métodos vamos definir que queremos estimar a seguinte integral genérica:

$$\gamma = \int_a^b f(x) \mathrm{d}x \tag{2}$$

No caso estudado, $a = 0$, $b = 1$ e $f = e^{-ax} \cos bx$.

2.1 Crude Monte Carlo Method (Método Cru)

Esse primeiro método consiste em pegar valores, sendo $i \in \{1, 2, ..., n\}$, que seguem uma distribuição uniforme $U_{[a,b]}$, e calcular:

$$\hat{\gamma_c} = \frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{3}$$

Esse $\hat{\gamma_c}$ será o estimador para γ definido na equação 2. Perceba que a ideia desse estimador é parecido com a ideia de uma soma de Riemann, contudo utiliza-se pontos de forma aleatória. Além disso a variância σ_c^2 encontrado para esse estimador será:

$$\sigma_c^2 = \frac{1}{n}\int_a^b (f(x) - \hat{\gamma_c})^2 \mathrm{d}x \tag{4}$$

Percebe-se que essa forma de calcular a variância não poderá ser utilizada, pois envolve calcular a integral de $f(x)$, e como é dito no enunciado do problema o valor analítico dela não é conhecido. E para tal, pode-se utilizar a variância amostral s_c^2 para estimar o a variância desse estimador, definido, como:

$$s_c^2 = \frac{1}{n-1}\sum_{i=1}^{n}(f(x_i) - \hat{\gamma_c})^2 \tag{5}$$

Contudo seja mais facil utilizar a fórmula simplificada obtida através de manipulações algébricas, mostrada abaixo:

$$s_c^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n} f(x_i)^2 - n\hat{\gamma_c}^2\right) \tag{6}$$

E utiliza-se a ideia do intervalo de confiança explicado no *Exercício Programa 1* para estimar o erro, no caso como se está utilizando a variância amostral o estimador $\hat{\gamma_c}$ deve seguir uma distribuição *t-Student*, sendo o erro e_{h_0} dado por:

$$e_{c_0} = t_{n-1;\alpha/2}\sqrt{\frac{s_c^2}{n}} \tag{7}$$

Sendo que o $t_{n-1;\alpha/2}$ é o valor de t, que segue a distribuição *t-Student*, com $n-1$ graus de liberdade e com probabilidade de $\mathbb{P}(-t_{n-1;\alpha/2} \leq t \leq t_{n-1;\alpha/2}) = 1 - \alpha$.

Com essas ideias foi possível implementar em um código na linguagem ***Python*** o método de Monte Carlo Cru. Abaixo mostra-se esse código:

```
## Crude Monte Carlo Method

# Importa as funções utilizadas nesse código
from random import  seed, random
from math import cos, exp,sqrt
from time import time
from scipy.stats import t

```

```python
# Define os parâmetros de entrada
a = 0.541147330 # 0.RG
b = 0.42490023810 #0.CPF
seed(1) # Define uma seed para os resultados obtidos pelo aluno
#sejam os mesmos obtidos pelo monitor

# Define as funções
def Print_info(media, n, err, t): # Função para printar os resultados obtidos no método
    print("Media: ", media)
    print("O numero de iterações: ", n)
    print("O erro: ", err*100, "%")
    print("Tempo de simulação: ", t)

def f(x): # Função que se quer integrar
    return exp(-a*x)*cos(b*x)

# Define o método Crude para estimar o valor da integral
def Crude_MC():
    n = 0 # Inicializa a variável n que representa o número de iterações necessária
    #para obter um erro menor que 0.05%
    soma = 0 # Inicializa o somatório de f(x_i) de i = 1 até n
    soma_quad = 0 # Inicializa o somatório de f(x_i)^2 de i = 1 até n
    err = 1 # Inicializa a variável do erro relativo
    t1 = time() # Utilizado para calcular o tempo para rodar a função
    while err > 0.0005: # Só sairá do loop se o erro relativo for menor que 0.0005 ou 0.05%
        n +=1
        x1 = random() # Atribui o valor a variável aleatória x_i
        f1 = f(x1) # Calcula f(x_i)
        soma = soma + f1 # Adiciona termo ao somatório de f(x_i)
        soma_quad = soma_quad + f1**2 # Adiciona termo ao somatório de f(x_i)^2

        if n == 1: # Condição para que não dê erro na primeira roda pois há um divisão por n-1
        # e quando n = 1 haverá divisão por zero, ou seja, dará um erro
            err = 1
        else:
            var = (soma_quad - soma**2/n)/(n-1) # Calcula a variância do método
            err = -t.ppf(0.005,n-1)*sqrt(var/n) # Calcula o erro desse método para dado n
    t2 = time()
    media = soma/n # Calcula o estimador para a integral de f(x)
    return media, n, err, t2-t1 # Retorna a estimativa, o n, o erro e o tempo para rodar a rotina

media_c, n_c, err_c, t_c = Crude_MC()
print("Crude Monte Carlo Method:")
Print_info(media_c, n_c, err_c, t_c)
```

2.2 Hit or Miss Monte Carlo Method

Pode-se dizer que o método consiste em gerar de forma aleatória pontos dentro do seu espaço, no caso o espaço é o $\mathbb{R}^2$, pois trabalha-se com um função de variável única, logo deve-se gerar aleatoriamente pares (x, y). Também define-se uma função $T(x, y)$ que verifica se $y \leq f(x)$ e recebe o valor 1 caso a afirmação seja verdadeira e caso o contrário recebe ovalor 0. E assim gera-se pares (x_i, y_i), tal que $x_i, y_i \sim U_{[a;b]}$ e $i \in \{1, 2, 3, ..., n\}$, e com eles calcula-se o estimador $\hat{\gamma_h}$ para a γ (definida na eq. 2) que será dado por:

$$\hat{\gamma_h} = (b-a)^2 \frac{1}{n} \sum_{i=1}^{n} T(x_i, y_i) \tag{8}$$

Note que o termo $(b-a)^2$ é a área do quadrado que contém a área embaixo da curva $f(x)$, no intervalo $[a; b]$ (numericamente igual a γ), pois sem ele estaria sendo calculado a proporção entre a área embaixo da curva $f(x)$ e a área do quadrado de lado $b-a$. No caso estudado, a proporção e o estimador $\hat{\gamma_h}$ serão numericamente iguais, pois $a = 0$ e $b = 1$, logo a área do quadrado tem uma área unitária.

A variância σ_m^2 desse estimador é dado por:

$$\sigma_m^2 = \frac{1}{n} \frac{\gamma}{(b-a)^2} \left(1 - \frac{\gamma}{(b-a)^2}\right) \tag{9}$$

Contudo como anteriormente será utilizado a variância amostral s_h^2 que será dado por:

$$s_h^2 = \frac{\hat{\gamma_h}}{(b-a)^2} \left(1 - \frac{\hat{\gamma_h}}{(b-a)^2}\right) \tag{10}$$

E utiliza-se novamente a ideia de intervalo de confiança, contudo nesse caso a distribuição é normal (lembrando que issocsó vale por $n \gg 1$ e assim $n \frac{\hat{\gamma_h}}{(b-a)^2} \geq 5$ e $n\big(1 - \frac{\hat{\gamma_h}}{(b-a)^2}\big) \geq 5$), logo pode-se utilizar $z_{\alpha/2}$, sendo que a variável z segue um distribuição normal de probalidade e $\mathbb{P}(-z_{\alpha/2} \leq z \leq z_{\alpha/2}) = 1 - \alpha$. Assim o erro e_{h_0} será dado por:

$$e_{h_0} = z_{\alpha/2} \sqrt{\frac{s_h^2}{n}} \tag{11}$$

Com essas equações é possível implementar esse método num código na linguagem ***Python***, mostrado a seguir:

```python
## Hit or Miss Monte Carlo Method

# Importa as funções utilizadas nesse código
from math import exp, cos, sqrt
from random import random, uniform, seed
from time import time

```

```python
# Define os parâmetros de entrada
a = 0.541147330 # 0.RG
b = 0.42490023810 #0.CPF
seed(10) # Define uma seed para os resultados obtidos pelo aluno
#sejam os mesmos obtidos pelo monitor

# Define as funções
def Print_info(media, n, err, t): # Função para printar os resultados obtidos no método
    print("Media: ", media)
    print("O numero de iterações: ", n)
    print("O erro: ", err*100, "%")
    print("Tempo de simulação: ", t)

def f(x):# Função que se quer integrar
    return exp(-a*x)*cos(b*x)

def T(x,y): # Função que testa se y<=f(x)
    if y <= f(x):
        return 1
    else:
        return 0

# Define o método Hit or Miss para estimar o valor da integral
def Hit_Miss_MC():
    n = 0 # Inicializa a variável n que representa o número de iterações necessária
    #para obter um erro menor que 0.05%
    soma = 0 # Inicializa o somatório de T(x_i,y_i) de i = 1 até n
    err = 1 # Inicializa a variável do erro relativo
    t1 = time() # Utilizado para calcular o tempo para rodar a função
    z_alpha = 3.3 # Valor da variável z para alpha = 0.1%, esse valor muda para alpha diferente
    while err>0.0005: # Só sairá do loop se o erro relativo for menor que 0.0005 ou 0.05%
        n +=1
        # for i in range(n):
        x = random() # Atribui o valor a variável aleatória x_i
        y = random() # Atribui o valor a variável aleatória y_i
        soma = soma + T(x,y) # Adiciona termo ao somatório de T(x_i, y_i)
        if n == 1 or soma == 0 or soma == n: # Condição para que não saia do loop, pois
        # caso uma dessas condições for satisfeita o erro relativo vai ser igual a 0
            err = 1
        else:
            err = z_alpha*sqrt(soma/n*(1-soma/n)/n) # Calcula o erro relativo do método para dado n
            # n1 = round((3/(0.0005*(soma/n)))**2*soma/n*(1-soma/n))
    t2 = time()
    media = soma/n # Calcula o estimador para a integral de f(x)
    return media, n, err, t2-t1
media, n, err, t = Hit_Miss_MC()
print("Hit or Miss Carlo Method:")
```

```
Print_info(media, n, err, t)
```

2.3 Importance Sampling

No intuito de aumentar a velocidade de convergência, métodos como este e o próximo foram foram criados. Nesse caso define-se uma função distribuição de probabilidade $g(x)$, tal que essa função seja, aproximadamente, proporcional a função $f(x)$. Além disso os pontos x_i devem seguir essa densidade de probabilidade definida pela função $g(x)$, lembrando que $i \in \{1, 2, 3, ..., n\}$. Assim o estimador $\hat{\gamma_s}$ para γ é definido como:

$$\hat{\gamma_s} = \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_i)}{g(x_i)} \tag{12}$$

Isso se deve pela seguinte manipulação algébrica com a integral definida na equação 2:

$$\int_a^b f(x)\mathrm{d}x = \int_a^b \frac{f(x)}{g(x)}g(x)\mathrm{d}x \tag{13}$$

Perceba que isso é o valor esperado $\mathbb{E}(\frac{f(x)}{g(x)})$ da função $\dfrac{f(x)}{g(x)}$ considerando uma função distribuição de probabilidade $g(x)$, contudo isso só vale se essa distribuição de probabilidade só estiver definida no intervalo $[a; b]$. Dentre as funções recomendadas pelo professor (Gamma, Weibull e Beta distributions) a única que passa por essa restrição é a distribuição Beta, logo foi a única testada de fato.

Sobre a variância σ_s^2 desse método é dado pela seguinte fórmula:

$$\sigma_s^2 = \frac{1}{n}\int_a^b \left(\frac{f(x)}{g(x)} - \gamma\right)^2 \mathrm{d}x \tag{14}$$

Porém, como no método Cru, será utilizado a variância amostral s_s^2 para estimar a variância:

$$s_s^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{f(x_i)}{g(x_i)} - \hat{\gamma_s}\right)^2 \tag{15}$$

E assim como no método Cru, utiliza-se a forma siplificada para calcular a variância, pela maior facilidade na implementação, abaixo é mostrado essa forma:

$$s_s^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n}\left(\frac{f(x_i)}{g(x_i)}\right)^2 - n\hat{\gamma_s}^2\right) \tag{16}$$

Com isso pode-se estimar o erro desse método utilizando o intervalo de confiança, assim como no Crude Monte Carlo deve-se utiliza a distribuição *t-Student*. Define-se, para tal, $t_{n-1;\alpha/2}$, da mesma forma que definiu-se na subseção 2.1, Logo calcula-se o erro e_{s_0} do estimador $\hat{\gamma}_s$ com a seguinte equação:

$$e_{s_0} = t_{n-1;\alpha/2}\sqrt{\frac{s_s^2}{n}} \tag{17}$$

Antes de fazer a função era necessário descobrir os parâmetros da função distribuição de probabilidade Beta, pois essa função depende de dois parâmetros α e β. Para tal foi feita uma rotina em ***Python*** para determinar esses dois parâmetros que fazem a função Beta mais se aproximar da função $f(x)$, isso de forma aproximada:

```python
## Descobre os melhores parâmetros para a distribuução Beta

# Importa as funções utilizadas na rotina
from math import exp, cos
from scipy.stats import beta
from time import time
# Define a funções
a = 0.54114733
b = 0.11260680
def f(x): # Função que se quer integrar
    return exp(-a*x)*cos(b*x)

def g(x, alfa, betha): # Função distribuição de probalidade Beta
    return beta.pdf(x,alfa, betha, loc=0, scale=1)

def Calc_Err_Quad(tup): # Função que calcula o erro quadrático entre os pontos de beta e da função f(x)
    N = 2*10**3 # Número de pontos utilizados
    i = 10 # Tira os 10 primeiros pontos para não dar infinito
    alfa = tup[0] # Define o parâmetro alfa da função Beta através da tupla parâmetro desta função
    betha = tup[1] # Define o parâmetro beta da função Beta através da tupla parâmetro desta função
    Err_quad = 0 # Inicializa a variável do erro quadrático

    while i<N:
        Err_quad = Err_quad + (f(i/N)-g(i/N, alfa, betha))**2 # Calcula o erro quadrático
        i +=1
    Err_quad = Err_quad/N
    return Err_quad

def Procura_alpha_beta(list_tup): # Função procura quais são os melhores parâmetros alfa e beta
    l = len(list_tup) # Descobre quantas tuplas foram definidas
    i = 1 # Contador do loop
    trocou = 0
    escolha = list_tup[0] # Variável que conterá o melhor par alfa e beta
    Err_quad_escolha = Calc_Err_Quad(escolha) # Define o erro quadrático da primeira tupla
    while i < l-1:
        Err_quad_tup_i = Calc_Err_Quad(list_tup[i])
        if Err_quad_escolha>Err_quad_tup_i: # Condiciona que para trocar a tupla vigente
```

```
        # o erro quadrático dessa nova tupla deve ser menor que o da tupla
            escolha = list_tup[i] # Troca para a tupla com menor erro quadrático
            Err_quad_escolha = Err_quad_tup_i # Troca o erro quadrático para o da nova tupla
        i +=1
    return escolha

def Sort_tup(n , d, v_inicial): # Função que cria uma lista de tuplas para o alfa e o beta
# os parâmetros de entra dessa função são n o número de tuplas, d o incremento que os parâmetros
# recebem e v_inicial é a tupla que contém o valor inicial que os parâmetros alfa e beta terão
# inicialmente
    i = 0 # Contador do primeiro loop
    list_tup = [] # Inicializa a lista das tuplas
    while i<n:
        j = 0
        alfa = v_inicial[0]+i*d # Adiciona o incremento para o parâmetro alfa
        while j<n:
            betha = v_inicial[1]+j*d # Adiciona o incremento para o parâmetro beta
            list_tup.append((alfa,betha))
            j+=1
        i+=1
    return list_tup

list_tup = Sort_tup(12, 0.05, (0.5,0.5))

escolha = Procura_alpha_beta(list_tup)

print(escolha)
```

Os parâmetros encontrados foram $\alpha = 0,9$ e $\beta = 1,05$. Com essas equações e com os parâmetros α e β é possível implementar esse método em uma rotina na linguagem ***Python***, como é mostrado abaixo:

```
## Importance sampling

# Importa as funções utilizadas nesse código
from math import exp, cos, sqrt
from scipy.stats import beta
from random import betavariate, seed
from time import time
from scipy.stats import t

# Define os parâmetros de entrada
a = 0.541147330 # 0.RG
b = 0.42490023810 #0.CPF
alfa = 0.9 # Parâmetro alfa para a função Beta
betha = 1.05 # Parâmetro beta para a função Beta
seed(21) # Define uma seed para os resultados obtidos pelo aluno
#sejam os mesmos obtidos pelo monitor

# Define as funções
```

```python
def Print_info(media, n, err, t): # Função para printar os resultados obtidos no método
    print("Media: ", media)
    print("O numero de iterações: ", n)
    print("O erro: ", err*100, "%")
    print("Tempo de simulação: ", t)

def f(x): # Função que se quer integrar
    return exp(-a*x)*cos(b*x)

def g(x): # Função distribuição de probabilidade da variável aleatória
    return beta.pdf(x,alfa, betha, loc=0, scale=1)

# Define o método Importance Sampling para estimar o valor da integral
def Importance_Sampling():
    n = 0 # Inicializa a variável n que representa o número de iterações necessária
    #para obter um erro menor que 0.05%
    soma = 0 # Inicializa o somatório de f(x_i)/g(x_i) de i = 1 até n
    soma_quad = 0 # Inicializa o somatório de (f(x_i)/g(x_i))^2 de i = 1 até n
    # list_r = []
    err = 1 # Inicializa a variável do erro relativo
    t1 = time() # Utilizado para calcular o tempo para rodar a função
    while err > 0.0005: # Só sairá do loop se o erro relativo for menor que 0.0005 ou 0.05%
        n += 1
        x1 = betavariate(alfa,betha) # Atribui o valor a variável aleatória x_i
        f_1 = f(x1) # Calcula f(x_i)
        g_1 = g(x1) # Calcula g(x_i)
        # list_r.append(f_1/g_1)
        soma = soma + f_1/g_1 # Adiciona termo ao somatório de f(x_i)/g(x_i)
        soma_quad = soma_quad + (f_1/g_1)**2 # Adiciona termo ao somatório de (f(x_i)/g(x_i))^2
        if n == 1: # Condição para que não dê erro na primeira roda pois há um divisão por n-1
        # e quando n = 1 haverá divisão por zero, ou seja, dará um erro
            err = 1
        else:
            var = (soma_quad - soma**2/n)/(n-1) # Calcula a variância do método
            err = -t.ppf(0.005,n-1)*sqrt(var/n) # Calcula o erro desse método para dado n
    t2 = time()
    media = soma/n # Calcula o estimador para a integral de f(x)
    return media, n, err, t2-t1 # Retorna a estimativa, o n, o erro e o tempo para rodar a rotina

media, n, err, t = Importance_Sampling()
print("Importance Sampling Monte Carlo Method:")
```

```
Print_info(media, n, err, t)
```

2.4 Control Variate

Por fim mostra-se o método Control Variate ou Variável de Controle. Nesse método utiliza-se uma função $\varphi(x)$, que é próxima da função $f(x)$ e cuja a integral γ_1 é conhecida:

$$\gamma_1 = \int_a^b \varphi(x)\mathrm{d}x \tag{18}$$

Além disso pode-se fazer a seguinte manipulação algébrica da integral, definida na equação 2:

$$\gamma = \int_a^b f(x)\mathrm{d}x \Rightarrow \gamma = \int_a^b \big(f(x) - \varphi(x) + \varphi(x)\big) \Rightarrow \gamma = \int_a^b \big(f(x) - \varphi(x)\big) + \int_a^b \varphi(x)\mathrm{d}x \Rightarrow$$

$$\Rightarrow \gamma = \int_a^b \big(f(x) - \varphi(x)\big) + \gamma_1$$

Estabelecendo os pontos x_i, sendo $i \in \{1, 2, 3, ..., n\}$ e tal que $x_i \sim U_{[a;b]}$. Logo pode-se definir o estimador $\hat{\gamma_v}$ da seguinte forma:

$$\hat{\gamma_v} = \frac{1}{n}\sum_{i=1}^{n} \big(f(x_i) - \varphi(x+i)\big) + \gamma_1 \tag{19}$$

Assim como nas seções anteriores mostra-se a variância σ_v^2 que esse método possui:

$$\sigma_v^2 = \frac{1}{n}\Big(\sigma\big(f(x_i)\big)^2 + \sigma\big(\varphi(x_i)\big)^2 - 2\rho\big(f(x_i), \varphi(x_i)\big)\sigma\big(f(x_i)\big)\sigma\big(\varphi(_i)\big)\Big) \tag{20}$$

Sendo $\rho\big(f(x_i), \varphi(x_i)\big)$ a correlação entre as funções $f(x)$ e $\varphi(x)$. Uma substituição que facilitará na hora de implementar é utilizar ao invés a correlação a covariância $\sigma\big(f(x), g(x)\big)$ que é definida como:

$$\sigma(f(x), g(x)) = \rho\big(f(x_i), \varphi(x_i)\big)\sigma\big(f(x_i)\big)\sigma\big(\varphi(_i)\big)\Big) \tag{21}$$

Será utilizado a variância amostral de $f(x)$, denotada por s_f^2, e de $\varphi(x)$, denotada por s_φ^2 para estimar a variância do método e a covariância amostral entre $f(x)$ e $\varphi(x)$, denotada por $s_{f,\varphi}$, para estimar a covariância das funções, abaixo mostra-se as fórmulas delas:

$$s_f^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(x_i)\right)^2 \tag{22}$$

$$s_\varphi^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(\varphi(x_i) - \frac{1}{n}\sum_{i=1}^{n} \varphi(x_i)\right)^2 \tag{23}$$

$$s_{f,\varphi} = \frac{1}{n-1}\sum_{i=1}^{n}\left(\left(f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(x_i)\right)\left(\varphi(x_i) - \frac{1}{n}\sum_{i=1}^{n}\varphi(x_i)\right)\right) \tag{24}$$

Contudo será dado preferência para as seguintes expressões para calcular as variâncias e a covariâncias, pela maior facilidade na hora de implementação em uma rotina:

$$s_f^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n} f(x_i)^2 - \frac{1}{n}\left(\sum_{i=1}^{n} f(x_i)\right)^2\right) \tag{25}$$

$$s_\varphi^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n} \varphi(x_i)^2 - \frac{1}{n}\left(\sum_{i=1}^{n} \varphi(x_i)\right)^2\right) \tag{26}$$

$$s_{f,\varphi} = \frac{1}{n-1}\left(\sum_{i=1}^{n} f(x_i)\varphi(x_i) - \frac{1}{n}\left(\sum_{i=1}^{n} f(x)\right)\left(\sum_{i=1}^{n} \varphi(x)\right)\right) \tag{27}$$

Assim a variância amostral s_v^2 será dada por:

$$s_v^2 = s_f^2 + s_\varphi^2 - 2s_{f,\varphi} \tag{28}$$

Como o estimador $\hat{\gamma_v}$ baseia-se em uma média, ao utilizar a variância amostral esse estimador seguirá uma distribuição *t-Student*. Utilizando a ideia de intervalo de confiança pode-se dizer que o erro e_{v_0} desse estimador será:

$$e_{v_0} = t_{n-1;\alpha/2}\sqrt{\frac{s_v^2}{n}} \tag{29}$$

Sendo $t_{n-1;\alpha/2}$ definido da mesma forma que na subseções 2.1 e 2.3.

Para implementação desse método em uma rotina, pensou-se em duas funções para $\varphi(x)$, a primeira seria o polinômio de Taylor de grau 1 ou a função exponencial e^{-ax}, pois no intervalo $[0;1]$ a função $\cos(bx)$ varia muito pouco e seu valor inicial é 1. Assim implementou-se as duas funções e verificou-se qual convergia mais rápido, para tal se olhou para o valor de n. Percebeu-se assim que a função exponencial convergia mais rápida, logo foi ela a escolhida.

Com isso foi feito um código em ***Python*** implementando esse método. Abaixo mostra-se o código:

```python
## Control Variate

# Importa as funções utilizadas nesse código
from math import exp, cos, sqrt
from random import random, seed
from time import time
from scipy.stats import t


# Define os parâmetros de entrada
a = 0.541147330 # 0.RG
b = 0.42490023810 #0.CPF
```

```
Integral_exp = 1/a*(1-exp(-a)) # Valor da integral de 0 a 1 de exp(-ax)
Integral_1 = 1-a/2  # Valor da integral de 0 a 1 de 1-ax, Polinômio de Taylor de grau 1
seed(81) # Define uma seed para os resultados obtidos pelo aluno
#sejam os mesmos obtidos pelo monitor
# Define as funções

def Print_info(media, n, err, t): # Função para printar os resultados obtidos no método
    print("Media: ", media)
    print("O numero de iterações: ", n)
    print("O erro: ", err*100, "%")
    print("Tempo de simulação: ", t)

def f(x): # Função que se quer integrar
    return exp(-a*x)*cos(b*x)

def phi1(x): # Função utilizada como variável de controle
    return exp(-a*x)

def phi2(x): # Função utilizada como variável de controle
    return 1-a*x

# Define o método Control Variate para estimar o valor da integral
def Control_Variate(f,phi, Integral_known):
    n = 0 # Inicializa a variável n que representa o número de iterações necessária
    #para obter um erro menor que 0.05%
    soma_f = 0 # Inicializa o somatório de f(x_i) de i = 1 até n
    soma_phi = 0 # Inicializa o somatório de g(x_i) de i = 1 até n
    soma_quad_f = 0 # Inicializa o somatório de f(x_i)^2 de i = 1 até n
    soma_quad_phi = 0 # Inicializa o somatório de g(x_i)^2 de i = 1 até n
    soma_fphi = 0 # Inicializa o somatório de f(x_i)*g(x_i) de i = 1 até n
    err = 1 # Inicializa a variável do erro relativo
    t1 = time() # Utilizado para calcular o tempo para rodar a função
    while err > 0.0005: # Só sairá do loop se o erro relativo for menor que 0.0005 ou 0.05%
        n += 1
        x = random() # Atribui o valor a variável aleatória x_i
        f1 = f(x) # Calcula f(x_i)
        phi1 = phi(x) # Calcula phi(x_i)
        soma_f = soma_f + f1 # Adiciona termo ao somatório de f(x_i)
        soma_phi = soma_phi + phi1 # Adiciona termo ao somatório de phi(x_i)
        soma_quad_f = soma_quad_f + f1**2 # Adiciona termo ao somatório de f(x_i)^2
        soma_quad_phi = soma_quad_phi + phi1**2 # Adiciona termo ao somatório de phi(x_i)^2
        soma_fphi = soma_fphi + f1*phi1 # Adiciona termo ao somatório de f(x_i)*phi(x_i)
        if n == 1:# Condição para que não dê erro na primeira roda
```

```
pois há um divisão por n-1
        # e quando n = 1 haverá divisão por zero, ou seja, dará um erro
            err = 1
        else:
            Cov = (soma_fphi - soma_f*soma_phi/n)/(n-1) # Calcula a covariância das funções
            # f(x) e phi(x)
            var_f = (soma_quad_f - soma_f**2/n)/(n-1) # Calcula a variância de f(x)
            var_phi = (soma_quad_phi - soma_phi**2/n)/(n-1) # Calcula a variância de phi(x)
            var = var_f + var_phi -2*Cov # Calcula a variância desse método
            err = -t.ppf(0.005,n-1)*sqrt(var/n) # Calcula o erro desse método para dado n
    t2 = time()
    media = (soma_f-soma_phi)/n + Integral_known # Calcula o estimador para o valor da integral de f(x)
    return media, n, err, t2-t1 # Retorna a estimativa, o n, o erro e o tempo para rodar a rotina

media1, n1, err1, t1 = Control_Variate(f,phi1, Integral_exp) # Utiliza phi(x) = exp(-a*x)
media2, n2, err2, t2 = Control_Variate(f,phi2, Integral_1) # Utiliza phi(x) = 1 - a*x
print("Control Variate Monte Carlo Method:")
print("phi(x) = exp(-a*x) :")
Print_info(media1, n1, err1, t1)
print("\n")
print("phi(x) = 1-a*x :")
Print_info(media2, n2, err2, t2)
```

3 Comparação dos métodos e seus resultados

Após a realização dos códigos para cada método, obteve-se os seguintes resultados:

Método	**estimativa**	**n**	**tempo (s)**
Crude	0.75227718634077001	490995	83.795900344848604
Hit or Miss	0.7525603307521267	8111452	7.8101112842559797
Importance Sampling	0.75215026632369897	77547	27.1090695858001
Control Variate	0.75214987506770903	6.818	1.1938495635986299

Primeiramente, é preciso notar que, assim como era esperado, todos os métodos produziram estimativas dentro do intervalo de $[0.95\gamma; 1.05\gamma]$, que é igual a $[0.751916; 0.752668]$. Ao analisar os n's percebe-se que o método Hit or Miss tem um número muito maior que o dos outros métodos, e o Control Variate tem um n muito menor que o dos outros. Isso se deve ao fato do método do Control Variate utilizar em geral funções $\varphi(x)$ com um correlação alta com a função $f(x)$, diminuindo a variância do método, assim diminuido o n, já o Hit or Miss precisa de um número muito grande de pontos para ser representativo na proporção de áreas, por isso o n é grande.

Agora pelo tempo o método Hit or Miss tem um tempo bem menor que os métodos do Crude e do Importance Sampling, ao fazer um teste percebeu-se que a utilização da função que calcula o $t_{n-1;\alpha/2}$ é o responsável por esse tempo muito longo nessas rotinas. Percebeu-se que esse $t_{n-1;\alpha}$ converge para $t_{n-1;\alpha} = 2,576$ para os dois métodos, logo utilizou-se esse número, na função final com todos os métodos juntos, ao invés de ficar calculando o $t_{n-1;\alpha}$ a cada iteração. Com isso os novos resultados foram:

Método	**estimativa**	**n**	**tempo (s)**
Crude	0.7522765939773961	491045	0.8692910671234131
Hit or Miss	0.7525603307521267	8111452	7.8101112842559797
Importance Sampling	0.7521502663236996	77547	13.470654487609863
Control Variate	0.75214987506770903	6.818	1.1938495635986299

Percebe-se na redução significativa dos tempos nos métodos do Crude e do Importance Sampling. Contudo outro problema verficado no Importance Sampling para o tempo continuar grande é a utilização da função Beta tanto em $g(x)$ como na hora de criar as variáveis aleatórias x_i, e esse problema não é possível solucionar. Assim percebe-se que mesmo o Importance Sampling tendo n menor que o Hit or Miss ou o Crude, seu tempo é significativamente maior, logo nesse caso não é o melhor método para ser implementado computacionalmente.

O melhor método é o Control Variate, pois o n é muito menor que os outros métodos e mesmo utilizando a função que calcula $t_{n-1;\alpha}$ em cada iteração o seu tempo é muito pequeno. Ele apresenta uma boa eficiência e uma boa estimativa.

Abaixo mostra-se a rotina com todos os métodos:

```python
# Importa as funções utilizadas nesse código
from random import  seed, random, betavariate
from math import cos, exp,sqrt
from time import time
from scipy.stats import t, beta

# Define os parâmetros de entrada
a = 0.541147330 # 0.RG
b = 0.42490023810 #0.CPF
alfa = 0.9 # Parâmetro alfa para a função Beta
betha = 1.05 # Parâmetro beta para a função Beta
Integral_exp = 1/a*(1-exp(-a)) # Valor da integral de 0 a 1 de exp
    (-ax)
Integral_1 = 1-a/2  # Valor da integral de 0 a 1 de 1-ax, Polinômio
     de Taylor de grau 1

# Define as funções
def Print_info(media, n, err, t): # Função para printar os
    resultados obtidos no método
```

```python
    print("Media: ", media)
    print("O numero de iterações: ", n)
    print("O erro: ", err*100, "%")
    print("Tempo de simulação: ", t,"\n")

def f(x): # Função que se quer integrar
    return exp(-a*x)*cos(b*x)

def T(x,y): # Função que testa se y<=f(x)
    if y <= f(x):
        return 1
    else:
        return 0

def g(x): # Função distribuição de probabilidade da variável aleatória
    return beta.pdf(x,alfa, betha, loc=0, scale=1)

def phi1(x): # Função utilizada como variável de controle
    return exp(-a*x)

def phi2(x): # Função utilizada como variável de controle
    return 1-a*x

###### Crude Monte Carlo Method
seed(1) # Define uma seed para os resultados obtidos pelo aluno
#sejam os mesmos obtidos pelo monitor

# Define o método Crude para estimar o valor da integral
def Crude_MC():
    n = 0 # Inicializa a variável n que representa o número de iterações necessária
    #para obter um erro menor que 0.05%
    soma = 0 # Inicializa o somatório de f(x_i) de i = 1 até n
    soma_quad = 0 # Inicializa o somatório de f(x_i)^2 de i = 1 até n
    err = 1 # Inicializa a variável do erro relativo
    t_alpha = 2.576 # Define o valor do t_Student
    t1 = time() # Utilizado para calcular o tempo para rodar a função
    while err > 0.0005: # Só sairá do loop se o erro relativo for menor que 0.0005 ou 0.05%
        n +=1
        x1 = random() # Atribui o valor a variável aleatória x_i
        f1 = f(x1) # Calcula f(x_i)
        soma = soma + f1 # Adiciona termo ao somatório de f(x_i)
        soma_quad = soma_quad + f1**2 # Adiciona termo ao somatório de f(x_i)^2

        if n == 1: # Condição para que não dê erro na primeira roda pois há um divisão por n-1
        # e quando n = 1 haverá divisão por zero, ou seja, dará um erro
            err = 1
        else:
            var = (soma_quad - soma**2/n)/(n-1) # Calcula a variância do método
```

```
            err = t_alpha*sqrt(var/n) # Calcula o erro desse método para dado n
    t2 = time()
    media = soma/n # Calcula o estimador para a integral de f(x)
    return media, n, err, t2-t1 # Retorna a estimativa, o n, o erro e o tempo para rodar a rotina

media_c, n_c, err_c, t_c = Crude_MC()
print("Crude Monte Carlo Method:")
Print_info(media_c, n_c, err_c, t_c)


###### Hit or Miss Monte Carlo Method
seed(10) # Define uma seed para os resultados obtidos pelo aluno
#sejam os mesmos obtidos pelo monitor

# Define o método Hit or Miss para estimar o valor da integral
def Hit_Miss_MC():
    n = 0 # Inicializa a variável n que representa o número de iterações necessária
    #para obter um erro menor que 0.05%
    soma = 0 # Inicializa o somatório de T(x_i,y_i) de i = 1 até n
    err = 1 # Inicializa a variável do erro relativo
    t1 = time() # Utilizado para calcular o tempo para rodar a função
    z_alpha = 3.3 # Valor da variável z para alpha = 0.1%, esse valor muda para alpha diferente
    while err>0.0005: # Só sairá do loop se o erro relativo for menor que 0.0005 ou 0.05%
        n +=1
        # for i in range(n):
        x = random() # Atribui o valor a variável aleatória x_i
        y = random() # Atribui o valor a variável aleatória y_i
        soma = soma + T(x,y) # Adiciona termo ao somatório de T(x_i, y_i)
        if n == 1 or soma == 0 or soma == n: # Condição para que não saia do loop, pois
        # caso uma dessas condições for satisfeita o erro relativo vai ser igual a 0
            err = 1
        else:
            err = z_alpha*sqrt(soma/n*(1-soma/n)/n) # Calcula o erro relativo do método para dado n
            # n1 = round((3/(0.0005*(soma/n)))**2*soma/n*(1-soma/n))
    t2 = time()
    media = soma/n # Calcula o estimador para a integral de f(x)
    return media, n, err, t2-t1
media_h, n_h, err_h, t_h = Hit_Miss_MC()
print("Hit or Miss Carlo Method:")
Print_info(media_h, n_h, err_h, t_h)


###### Importance Sampling Monte Carlo Method
seed(21) # Define uma seed para os resultados obtidos pelo aluno
#sejam os mesmos obtidos pelo monitor

```

```python
# Define o método Importance Sampling para estimar o valor da integral
def Importance_Sampling():
    n = 0 # Inicializa a variável n que representa o número de iterações necessária
    #para obter um erro menor que 0.05%
    soma = 0 # Inicializa o somatório de f(x_i)/g(x_i) de i = 1 até n
    soma_quad = 0 # Inicializa o somatório de (f(x_i)/g(x_i))^2 de i = 1 até n
    t_alpha = 2.576 # Define o valor do t_Student
    err = 1 # Inicializa a variável do erro relativo
    t1 = time() # Utilizado para calcular o tempo para rodar a função
    while err > 0.0005: # Só sairá do loop se o erro relativo for menor que 0.0005 ou 0.05%
        n += 1
        x1 = betavariate(alfa,betha) # Atribui o valor a variável aleatória x_i
        f_1 = f(x1) # Calcula f(x_i)
        g_1 = g(x1) # Calcula g(x_i)

        # list_r.append(f_1/g_1)
        soma = soma + f_1/g_1 # Adiciona termo ao somatório de f(x_i)/g(x_i)
        soma_quad = soma_quad + (f_1/g_1)**2 # Adiciona termo ao somatório de (f(x_i)/g(x_i))^2
        if n == 1: # Condição para que não dê erro na primeira roda pois há um divisão por n-1
        # e quando n = 1 haverá divisão por zero, ou seja, dará um erro
            err = 1
        else:
            var = (soma_quad - soma**2/n)/(n-1) # Calcula a variância do método
            err = -t.ppf(0.005,n-1)*sqrt(var/n) # Calcula o erro desse método para dado n
    t2 = time()
    media = soma/n # Calcula o estimador para a integral de f(x)
    return media, n, err, t2-t1 # Retorna a estimativa, o n, o erro e o tempo para rodar a rotina

media_s, n_s, err_s, t_s = Importance_Sampling()
print("Importance Sampling Monte Carlo Method:")
Print_info(media_s, n_s, err_s, t_s)


###### Control Variate Monte Carlo Method
seed(81) # Define uma seed para os resultados obtidos pelo aluno
#sejam os mesmos obtidos pelo monitor

# Define o método Control Variate para estimar o valor da integral
def Control_Variate(f,phi, Integral_known):
    n = 0 # Inicializa a variável n que representa o número de iterações necessária
    #para obter um erro menor que 0.05%
    soma_f = 0 # Inicializa o somatório de f(x_i) de i = 1 até n
```

```
    soma_phi = 0 # Inicializa o somatório de g(x_i) de i = 1 até n
    soma_quad_f = 0 # Inicializa o somatório de f(x_i)^2 de i = 1 até n
    soma_quad_phi = 0 # Inicializa o somatório de g(x_i)^2 de i = 1 até n
    soma_fphi = 0 # Inicializa o somatório de f(x_i)*g(x_i) de i = 1 até n
    err = 1 # Inicializa a variável do erro relativo
    t1 = time() # Utilizado para calcular o tempo para rodar a função
    while err > 0.0005: # Só sairá do loop se o erro relativo for menor que 0.0005 ou 0.05%
        n += 1
        x = random() # Atribui o valor a variável aleatória x_i
        f1 = f(x) # Calcula f(x_i)
        phi1 = phi(x) # Calcula phi(x_i)
        soma_f = soma_f + f1 # Adiciona termo ao somatório de f(x_i)
        soma_phi = soma_phi + phi1 # Adiciona termo ao somatório de phi(x_i)
        soma_quad_f = soma_quad_f + f1**2 # Adiciona termo ao somatório de f(x_i)^2
        soma_quad_phi = soma_quad_phi + phi1**2 # Adiciona termo ao somatório de phi(x_i)^2
        soma_fphi = soma_fphi + f1*phi1 # Adiciona termo ao somatório de f(x_i)*phi(x_i)
        if n == 1:# Condição para que não dê erro na primeira roda pois há um divisão por n-1
        # e quando n = 1 haverá divisão por zero, ou seja, dará um erro
            err = 1
        else:
            Cov = (soma_fphi - soma_f*soma_phi/n)/(n-1) # Calcula a covariância das funções
            # f(x) e phi(x)
            var_f = (soma_quad_f - soma_f**2/n)/(n-1) # Calcula a variância de f(x)
            var_phi = (soma_quad_phi - soma_phi**2/n)/(n-1) # Calcula a variância de phi(x)
            var = var_f + var_phi -2*Cov # Calcula a variância desse método
            err = -t.ppf(0.005,n-1)*sqrt(var/n) # Calcula o erro desse método para dado n
    t2 = time()
    media = (soma_f-soma_phi)/n + Integral_known # Calcula o estimador para o valor da integral de f(x)
    return media, n, err, t2-t1 # Retorna a estimativa, o n, o erro e o tempo para rodar a rotina

media1, n1, err1, t1 = Control_Variate(f,phi1, Integral_exp) # Utiliza phi(x) = exp(-a*x)
media2, n2, err2, t2 = Control_Variate(f,phi2, Integral_1) # Utiliza phi(x) = 1 - a*x
print("Control Variate Monte Carlo Method:")
print("phi(x) = exp(-a*x) :")
Print_info(media1, n1, err1, t1)
print("phi(x) = 1-a*x :")
```

```
Print_info(media2, n2, err2, t2)
```