Laboratório de Computação e Simulação : MAP-2212

# EP 2: Métodos de Monte Carlo

Marcos Pereira - 11703832

## Atividade

Implementar as 4 variantes do Método de Monte Carlo descritas na Sec. G.2 para integrar, em $I = [0, 1]$, a função:

$$f(x) = \mathrm{e}^{-ax} \cos(bx)$$

onde $a = 0.RG$ e $b = 0.NUSP$ (RG e Numero USP do aluno). A resposta deve ser dada com erro relativo ($|g^* - g|/g) < 1\%$ onde $g^*$ é a estimativa numérica do valor da integral, e g é o valor real da integral (desconhecido).

## 1 Integração e Redução de Variância

### 1.1 Método de Monte Carlo Puro

Considere o problema de calcular a integral

$$I_S = \int_S f(x)dx \tag{1}$$

onde $f(x)$ é uma função suave por partes $\mathcal{C}^1$ em $S = [a, b]$, um intervalo real. Utilizamos o método 'crude MC' e aproximamos o valor da integral como

$$\int_S f(x)dx \approx \sum_{n=1}^{N} f(x_n)\,\Delta x_n$$

com $\Delta x_n = \frac{b-a}{N}$ obtemos:

$$\int_S f(x)dx \approx \frac{b-a}{N} \sum_{n=1}^{N} f(x_n) \implies \frac{1}{b-a} \int_S f(x)\,dx \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n) \tag{2}$$

sabendo que o valor médio de uma função em $S$ é dado por

$$\langle f \rangle = \frac{1}{b-a} \int_S f(x)\,dx = \frac{I_S}{\Delta x}$$

verificamos que uma aproximação para o valor médio de uma função num intervalo definido é

$$\langle f \rangle \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n)\ . \tag{3}$$

Agora suponha uma amostra aleatória uniforme com $N$ elementos $\mathcal{U}[a, b]$, o valor médio de $f$ em $\mathcal{U}$ é representado por $\bar{f}$ e é calculado como

$$\bar{f} = \frac{1}{N} \sum_{n=1} f(x_n)\ , \tag{4}$$

ou seja, podemos substituir o lado esquerdo da 3 e obter[1]

$$\langle f \rangle \approx \bar{f}\,,$$

ou seja,

$$\frac{I_S}{\Delta x} \approx \frac{1}{N} \sum_{x_n \in \mathcal{U}} f\left(x_n\right) . \tag{5}$$

Uma aproximação de $I_S$ pode ser obtida a partir da média da função em uma amostra aleatória uniforme, basta isolar $I_S$ em 5:

$$I_S \approx \frac{\Delta x}{N} \sum_{x_n \in \mathcal{U}} f\left(x_n\right) . \tag{6}$$

O Método apresentado acima é o Método de Monte Carlo "Cru"[2] [1] ou Puro e podemos realizá-lo para calcular o valor de $I_S$ a partir do valor médio da função no intervalo de integração.

A variância de $f\left(x\right)$ em $\mathcal{U}$ é calculada a partir do fato de que

$$\sigma^2 = \langle (f - \langle f \rangle)^2 \rangle \tag{7}$$

que resulta em

$$\begin{aligned} \sigma^2 &= \frac{1}{N} \sum_{x_n \in \mathcal{U}} \left(f\left(x_n\right) - \langle f \rangle\right)^2 = \frac{1}{N} \sum_{x_n \in \mathcal{U}} \left(f^2\left(x_n\right) - 2\langle f \rangle f\left(x_n\right) + \langle f \rangle^2\right) \\ &= \frac{1}{N} \sum_{x_n \in \mathcal{U}} f^2\left(x_n\right) - \underbrace{\frac{2\langle f \rangle}{N} \sum_{x_n \in \mathcal{U}} f\left(x_n\right)}_{=-2\langle f \rangle^2} + \underbrace{\frac{1}{N} \sum_{x_n \in \mathcal{U}} \langle f \rangle^2}_{=\langle f \rangle^2} \\ &= \langle f^2 \rangle - \langle f \rangle^2 \end{aligned} \tag{8}$$

da mesma forma

$$\sigma_c^2 = \overline{\left(f - \bar{f}\right)^2} = \overline{f^2} - \bar{f}^2 \tag{9}$$

implicando que o erro no cálculo pode ser calculado como

$$\sigma_c = \sqrt{\frac{\overline{f^2} - \bar{f}^2}{N}} =\equiv \sqrt{\frac{1}{N^2}\left[\sum_{x_n \in \mathcal{U}} f^2\left(x_n\right) - \frac{1}{N}\left(\sum_{x_n \in \mathcal{U}} f\left(x_n\right)\right)^2\right]} \tag{10}$$

já a variância esperada (calculada a partir da integral)[2] é dada por

$$\sigma_s^2 = \frac{1}{N} \int_S dx \left(f(x) - \frac{1}{N} \int_S f(x) dx\right) \tag{11}$$

a partir do erro então, podemos ajustar a precisão esperada da aproximação. Esse é o método de Monte Carlo menos preciso entre os demais, podemos finalmente estimar o valor da integral como

$$I_S \approx \bar{f}\Delta x \pm \sigma_c \tag{12}$$

podemos reduzir o erro cometido aumentando o número de elementos na distribuição uniforme, isso é, $N \to \infty$ $\sigma_c \to 0$. Mas também podemos reduzir significativamente a variância para obter uma acurácia maior no cálculo para isso usaremo a Amostragem por importância que é um método que visa realizar uma transformação de variáveis a fim de minimizar a variância.

[1] Importante notar que $\langle f \rangle$ é aproximado pelo valor médio de $f$ no intervalo particionado em partes iguais $\wp[a,b]$, pois esse resultado foi obtido a partir de uma Soma de Riemann, já $\bar{f}$ é o valor médio de $f$ na amostragem aleatória uniforme $\mathcal{U}[a,b]$

[2] "crude-MC

## 1.2 Amostragem por Importância

Esse método consiste em, a partir de uma função $g(x)$ de classe $\mathcal{C}^1(S)$, ou seja, suave por partes no intervalo $S = [a.b]$ tal que

$$g(x) > 0 \ \forall x \in S \qquad\qquad \int_S g(x)dx = 1\,,$$

e além disso também satisfaça a condição de aproximar muito de $f(x)$ no intervalo. Então geramos uma amostra aleatória uniforme a partir de $g(x)$, ou seja, se $\mathcal{U}[a,b]$ é uma distribuição uniforme do intervalo $S$, definimos

$$\mathcal{G} = g\left(\mathcal{U}[a,b]\right)$$

como uma distribuição auxiliar.

Nosso objetivo é encontrar o valor de $I_S$ como anteriormente, no entanto aqui realizaremos uma mudança de variáveis:

$$\int_S f(x)dx = \int_S \frac{f(x)}{g(x)}g(x)dx = \int_S \frac{f(x)}{g(x)}\partial_x G(x)dx = \int_{G(a)}^{G(b)} \frac{f\left(G^{-1}(G)\right)}{g\left(G^{-1}(G)\right)}dG \tag{13}$$

onde então assim como na eq. 2 verificamos que

$$\int_{G(a)}^{G(b)} \frac{f\left(G^{-1}(G)\right)}{g\left(G^{-1}(G)\right)}dG \approx \frac{\Delta G}{N}\sum_{n=1}^{N}\frac{f\left(G^{-1}(G)\right)}{g\left(G^{-1}(G)\right)} \implies \frac{1}{\Delta G}\int_{G(a)}^{G(b)} \frac{f\left(G^{-1}(G)\right)}{g\left(G^{-1}(G)\right)}dG \approx \frac{1}{N}\sum_{n=1}^{N}\frac{f\left(G^{-1}(G_n)\right)}{g\left(G^{-1}(G_n)\right)}\,,$$

note que $G^{-1}(x)$ é a inversa de $G$, além disso $G(x_n) = G_n$ com $x_n \in \wp[a,b]$ portanto: $G^{-1}\left(G(x)\right) = x$ e consequentemente $f(G^{-1}\left(G(x)\right)) = f(x)$, obtemos então uma aproximação para o valor médio de $\frac{f(x)}{g(x)}$ em $S$

$$\left\langle \frac{f}{g} \right\rangle \approx \frac{1}{N}\sum_{n=1}^{N}\frac{f(x_n)}{g(x_n)}\,, \tag{14}$$

essa condição equivale à condição observada na seção anterior

$$\left\langle \frac{f}{g} \right\rangle \approx \overline{\left(\frac{f}{g}\right)}$$

é muito importante notar que agora estamos tomando a média do integrando sobre a distribuição uniforme de pontos:

$$\overline{\left(\frac{f}{g}\right)} = \frac{1}{N}\sum_{x_n \in \mathcal{U}}\frac{f(x_n)}{g(x_n)}$$

note também a relação entre $x_n$, $g_n$ e $G_n$ através da forma explícita de $G$:

$$G(x) = \int_a^x g(x')dx'\,,$$

verifique que devido à condição que impomos a $g(x)$ no início limitamos $G(x)$ a um intervalo unitário. Além de tudo segue que a definição acima implica que $G_n = G(x_n)$. Nós podemos verificar pela a eq. 13 a integral é invariante sob mudanças de variáveis:

$$\int_0^1 \frac{f(G)}{g(G)}dG = \int_S f(x)dx$$

demonstrando que

$$\int_0^1 \frac{f(G)}{g(G)}dG = \frac{1}{\Delta x}\int_S f(x)dx \implies \left\langle \frac{f}{g} \right\rangle = \langle f \rangle$$

e portanto podemos aproximar $I_S$ como

$$I_S \approx \frac{\Delta x}{N} \sum_{x_n \in \mathcal{U}} \frac{f(x_n)}{g(x_n)}\,. \tag{15}$$

Agora vamos tomar algumas considerações primordiais. É importante antes ressaltar que $g(x)$ deve ser escolhida de forma com que a variância do integrando seja mínima, como imposto pela considerações iniciais, devemos escolher como $g(x)$ uma função que possua comportamento semelhante a $f(x)$ no intervalo, portanto

$$\frac{f(x)}{g(x)} \approx c \implies f(x) \approx cg(x)\,,$$

e verificando que

$$\sigma^2_{ami} = \overline{\left(\frac{f}{g}\right)^2} - \overline{\left(\frac{f}{g}\right)}^2 \tag{16}$$

e

$$\sigma_{ami} = \sqrt{\frac{\sigma^2_{ami}}{N}}\,.$$

### Forma Alternativa

A partir da variância calculada na seção anterior, tomamos sua forma contínua no intervalo $S$

$$\sigma^2_{ami} = \frac{1}{N} \int_0^1 \left(\frac{f(G)}{g(G)} - \left\langle \frac{f}{g} \right\rangle\right)^2 dG \tag{17}$$

portanto

## 1.3 Acertos e Erros

Para realizarmos esse método, suponha uma distribuição uniforme de números aleatórios $\mathcal{U}[a.b]$. Então definimos duas variáveis aleatórias $x'_n, y'_n \in \mathcal{U}[a,b]$ e escolhemos aleatoriamente um ponto $(x_n, y_n)$. Definimos então a função auxiliar $h$:

$$h(x,y) = \begin{cases} 1, & (x,y) = (x, f(x)) \\ 0, & (x,y) \neq (x, f(x)) \end{cases},$$

para ficar mais claro, defina uma amostra aleatória no $\mathbb{R}^2$ como $\mathcal{U}^2_S$ tal que $(x_n, y_n) \in \mathcal{U}^2_S$, a probabilidade de um ponto $(x_n, y_n)$ estar entre o gráfico da função $f(x)$ e o eixo $x$ é dada por

$$P = \frac{\iint_D h(x,y)dxdy}{\iint_D dxdy} = \frac{1}{\Delta A} \iint_D h(x,y)dxdy$$

e

$$P \approx \frac{N^*}{N}$$

portanto

$$\frac{1}{\Delta A} \iint_D h(x,y)dxdy = \frac{1}{(b-a)H} \int_S f(x)dx \approx \frac{N^*}{N} \tag{18}$$

aqui $N$ é o número de elementos da amostra uniforme, $N^*$ o número de pontos dentro da área que queremos calcular e $H$ é um valor real tal que $H \geq f(x), \ \ \forall x$, portanto

$$\int_S f(x)dx \approx \frac{(b-a)H}{N} \sum_{n=1}^{N} h(x_n, y_n) \tag{19}$$

cujo variância é calculada por

$$\sigma^2_{hm} = \frac{\mathbb{E}\left(h^2\right)}{N} = \frac{h - h^2}{N} \implies \sigma_{hm} = \sqrt{\frac{h-h^2}{N}}$$

# 2 Implementação do Código Usando Python3

Primeiro importaremos as seguintes bibliotecas e funções

```
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import scipy.integrate as sint
```

em seguida defino uma função que realizará a uma integral imprópria a partir do método de integração de Simpson

```

#antiderivative
def integral(f,x,dx=1e-4):
  result = []
  for i in range(len(x)):
    xi= np.linspace(0,x[i])
    result.append(sint.simps(f(xi),xi,dx=dx))
  return result
```

desejamos realizar o primeiro método citado na primeira seção, dessa maneira, estamos interessados em integrar a função

$$f(x) = \mathrm{e}^{-ax}\cos(bx)$$

com $a = .3039110$ e $b = .11703832$ no intervalo $S = [0,1]$, portanto devemos definir $a$ e $b$ e a função que será integrada

```
#implementação do método


#definição das constantes
a, b = .3039110,.11703832


#definição da função a ser integrada
def f(x,a=.11703832,b=.3039110,n=1):
  F=0
  if n ==1:
    F=np.exp(-a*x)*np.cos(b*x)
  else:
    F=(np.exp(-a*x)*np.cos(b*x))**n
  return F
```

note que ao definir a função, implementei uma variável $n$ que será útil para realizar alguns cálculos, como a eq. 8, como descrito no código fonte, $n$ equivale à i-ésima potência da função $f^n(x)$.

## 2.1 Monte Carlo 'cru' (MC-'crude')

O método é implementado a partir uma função definida que realiza o método. Essa função deve seguir o seguinte algoritmo:

1. Recebe:
   (a) Função a ser integrada;
   (b) Intervalo $[a,b]$;
   (c) Número de interações;
2. Gera intervalo uniforme $\mathcal{U}[a,b]$;
3. Aplica o método;
4. Devolve:
   (a) Resultado, Média, Média Quadrática, Variância, Erro, Iterações.

A função definida abaixo foi definida para realizar o método:

```
  #Método cru

def MCint(f,a=0,b=1,N=10,seed=False):
  """
    f = função
    a,b = float, float - numeros da atividade
    N = tamanho da amostra
    seed = False ou numero inteiro: gerador de numero aleatório (PRNG).

    Devolve DataFrame com colunas {'Resultado','Média','Média Quadrática','Variância','Erro','Steps'}
  """
  I, U, mean, meanquad, var, err = 0, 0, 0, 0, 0, 0

  if seed == False:
    U = np.random.uniform(a,b,N)
```

```
  elif seed == False:
    print('Definir seed com um int ou False para experimentos aleatórios.')
  else:
      #experimento com semente aleatória escolhida.
    np.random.seed(seed)
    U =np.random.uniform(a,b,N)

    #calcular média, média quadrática, variância e erro
  mean, meanquad = sum(f(U))/N, sum(f(U,n=2)/N)
  var = meanquad-mean**2
  err = (var/N)**.5


    #Aplicar método
  I=((b-a)/N)*sum(f(U))

    #salvar resultados em um DataFrame
  return pd.DataFrame(np.array([['{} +/- {}'.format(I,err), mean, meanquad, var, err, int(N)]]),index=['
    Valores'],columns=['Resultado','Média','Média Quadrática','Variância','Erro','Passos'])
```

Seu input é: a função a ser integrada (função definida), os limites de integração(float{a,b}), a precisão do processo (float) e semente aleatória (int). Caso não seja definida uma semente, o programa criará uma distribuição uniforme sob o intervalo definido, A semente é aleatória é do tipo int. será criado uma amostra cujos elementos pseudos aleatórios serão calculados a partir dessa semente aleatória (entre as linhas 14 até 21) caso não haja. Em seguida calculamos a média, média quadrática, variância e o erro, usando as fórmulas da primeira seção e na linha 28 o programa aplica o método de Monte Carlo com precisão igual ao erro com $N$ passos, então é retornada uma tabela (DataFrame): podemos acessar os

| $I_S \pm \sigma_c$ | $\bar{f}$ | $\overline{f^2}$ | $\sigma_c^2$ | $\sigma_c$ | $N$ |
|---|---|---|---|---|---|

resultados específicos utilizando os métodos do pandas:

```
    MCint(f,0,1,10)['Resultado']        #importa apenas o valor calculado como um dataframe novo
    MCint(f,0,1,10)['mean'].values     #array com a média
    float(MCint(f,0,1,10)['Resultado'].values)     #converte como um float
```

Agora para calcular a integral com uma certa precisão, como indicado no exercício criaremos outra função, a definitiva MCcrude:

```
def MCcrude(f,a=0,b=1,Ni=10,precisao=.005,seed=False,log=True):
  """
    f = função def f(x);
    a,b = float, float - numeros da atividade;
    Ni = Tamanho inicial da amostra (int);
    seed = False ou numero inteiro: semente aleatória;
    log = True or False: printa a relação entre passos x erro;

    Devolve DataFrame com colunas {'Resultado','Média','Média Quadrática','Variância','Erro','Steps'}.
  """

#(f,a=0,b=1,Ni=10,precisao=.005,seed=False):
#variáveis iniciais

  N, U = Ni, MCint(f,N=Ni,seed=seed)
  erro = float(U['Erro'].values[0])

  #definir laço: enquanto erro > precisao somar +1 no N e imprimir os passos e o erro associado
  if log==True:
    while erro > precisao:
      erro=float(MCint(f,a,b,N,seed)['Erro'].values[0])
      N=N+1
      print('passos: {},\n erro: {}'.format(N,erro))
  elif log==False:
    while erro > precisao:
      erro=float(MCint(f,a,b,N,seed)['Erro'].values[0])
      N=N+1
  else:
    print('log=True or False')

  #Usar função MCint() para criar um dataframe que será o output
  F = MCint(f,a,b,N,seed)
  return pd.concat([F,pd.DataFrame(np.array([precisao]),columns=['Acurácia'],index=['Valores'])],axis=1)
```

a função recebe a função a ser integrada, intervalo de integração, tentativas iniciais, precisão e a "semente de aleatoriedade".

Primeiro declaramos as variáveis iniciais, N inicial, erro inicial. A partir da linha 10 definimos um laço de repetição. Se o erro for menor que a precisão o programa não alterará a variável $N$, caso contrário, somará $+1$ em $N$ até que o erro seja menor que a tolerância.

Em seguida o programa criará F uma variável que armazena MCint com o $N$ encontrado da interação anterior. Aqui o output será dado pela concatenação de $F$ com uma coluna com a precisão desejada: desejamos calcular a integral no

| $I_S \pm \sigma_c$ | $\bar{f}$ | $\overline{f^2}$ | $\sigma_c^2$ | $\sigma_c$ | $N$ | $\sigma_e$ |
|---|---|---|---|---|---|---|

intervalo com erro relativo menor que .01, portanto entramos com

```
In [731]: MCcrude(f,precisao=.01,seed=43526,log=False)
Out[731]:
                                        Resultado                Média    Média Quadrática
    Variância                   Erro Passos  Acurácia
Valores  0.9406394174548054 +/- 0.009211748661259917  0.9406394174548053  0.8864996399376799
    0.0016971262679644772  0.009211748661259917     20      0.01

In [732]:
```

## 2.2 Importância da Amostragem

Para realização do método deveremos definir a função $g(x)$ e consequentemente $G(x)$ e $G^{-1}(x)$. Seja:

$$g(x) = A\,\mathrm{e}^{-\lambda x}$$

implicando a condição de normalização obtemos

$$\int_S A\,\mathrm{e}^{-\lambda x}\,dx = 1 \implies g(x) = \frac{\lambda\,\mathrm{e}^{-\lambda x}}{\mathrm{e}^{-\lambda a} - \mathrm{e}^{-\lambda b}}\,. \tag{20}$$

Sabe-se, pelo TFC que $G(x)$ pode ser definida como

$$G(x) = \int_a^x g(x')dx'$$

portanto

$$G(x) = \frac{\mathrm{e}^{-\lambda a} - \mathrm{e}^{-\lambda x}}{\mathrm{e}^{-\lambda a} - \mathrm{e}^{-\lambda b}} \implies G(S) = [0,1] \tag{21}$$

e consequentemente calculamos sua inversa

$$G^{-1}(x) = -\frac{1}{\lambda}\ln\left[x(\mathrm{e}^{-\lambda b} - \mathrm{e}^{-\lambda a}) + \mathrm{e}^{-\lambda a}\right] \implies G\left([0,1]\right) = [a,b] \tag{22}$$

```
  #definir quem é g, Ginv:
def g_and_Ginv(x,lambdaa=.05):
  #definir constante de normalização, g, e array vazio

  A = 0
  g = np.exp(-lambdaa*x)
  final = []
  G = []
  integral_of_g, Ginv = [], []

  #constante de normalização
  A = 1/sint.simps(g,x)

  #multiplicar cada elemento de g por uma constante de normalização
  for i in range(len(g)):
    final.append(g[i]*A)

  #final = final.append(final[49])
  #G = integral(np.array(final),x)


  #definir a inversa de G que estará definida em [0,1]
```

```
  Ginv = -(1/lambdaa)*np.log(-lambdaa*x/A+np.exp(-lambdaa*a))
  z =  -(1/lambdaa)*np.log(-lambdaa*x/A+np.exp(-lambdaa*a))
  Ginverso = []

  #for i in range(len(x)):
  # Ginverso.append(Ginv(x[i]))
  #y = np.array(Ginverso)

  return [final,Ginv]
```

Em seguida definimos uma função para realizar o método. Ela recebe a função que será integrada, uma função que devolva $g$ e $G^{-1}$ e alguns argumentos optativos:

```
def MCImpS(f,Ginv,N=10,lambdai=.05,tests=100,seed=False):

  #Intervalo uniforme [0,1]
  if seed == False:
    r = np.random.uniform(0,1,N)
  elif seed==True:
    r = np.random.uniform(0,1,N)
  else:
    np.seed(seed)
    r = np.random.uniform(0,1,N)

  variancia = []
  lambdas = [i*.05 for i in range(1,tests+1)]


  for lamb in lambdas:

    #calcular f(Ginv)/g(Ginv)

    GinI = Ginv(r,lamb)[1]
    F = f(GinI)
    g = g_and_Ginv(r,lamb)[0]

    f_over_g = []
    f_over_g2 = []


    for i in range(len(r)):
      f_over_g.append(F[i]/g[i])

    for i in range(len(r)):
      f_over_g2.append((F[i]/g[i])**2)

    #definir variância, média e média quadrática
    var = 0
    mean = 0
    meanquad = 0

    #calcular média
    mean = 1/N*sum(f_over_g)

    #calcular média quadrática

    meanquad = 1/N*sum(f_over_g2)

    #calcular variância
    var = meanquad-mean**2
    variancia.append(var)


  U = [variancia, lambdas]

  #Escolher menor lambda
  df=pd.DataFrame(np.transpose(U),columns='variância lambdas'.split())
  df = df[df['variância']==df['variância'].min()]

  minlambda = df['lambdas'].values[0]

  #Aplicar o método

  F = f(Ginv(r,minlambda)[1])
  G = Ginv(r,minlambda)[0]
```

```
63
64  f_over_gfinal = [F[i]/G[i] for i in range(len(Ginv(r,minlambda)[0]))]
65
66  I=1/N*sum(f_over_gfinal)
67
68  return I
```

## Referências


[1] Sukanta Deb. “Variational Monte Carlo Technique”. Em: Resonance 19.8 (2014), pp. 713–739.

[2] Julio Michael Stern. “Cognitive Constructivism and the Epistemic Significance of Sharp Statistical Hypotheses”. Em: Tutorial book for MaxEnt (2008), pp. 6–11.