Algoritmos de Monte Carlo e Cadeias de Markov - CPS767 2018/1

Prof. Daniel R. Figueiredo

Gabriel Matos Cardoso Leite

Terceira Lista de Exercícios

May 3, 2018

Questão 1: Calculando e

Vimos em aula um algoritmo de Monte Carlo para calcular o valor de π utilizando a relação entre áreas. Inspirado nesta mesma ideia, construa um algoritmo de Monte Carlo para calcular o valor da constante de Euler, e.

1. Descreva a variável aleatória cujo valor esperado está relacionado com a constante e. Obtenha o valor esperado da sua variável aleatória.

2. Calcule a variância dessa variável aleatória.

3. Implemente o método de Monte Carlo para gerar amostras da sua variável aleatória, calculando a média amostral M_n e utilizando-a para estimar e. Sua função deve usar o algoritmo de Mersenne Twister para geração de números pseudo-aleatórios uniformes entre 0 e 1 (usar este algoritmo em todos os problemas).

4. Seja $\hat{e}_n$ o valor do estimador após n amostras. Trace um gráfico do erro relativo do estimador, ou seja $|\hat{e}_n - e|/e$ em função de n, para $n = 1, \ldots, 10^6$ (utilize escala log - log). Trace no mesmo gráfico o erro padrão *(standard error)* do estimador. O que você pode concluir?

Solução:

1. A variável aleatória de interesse é N definida por

$$N = minn : S_n > 1$$

onde S_n é $S_n = \sum_{i=1}^{n} U_i$ e $U_1, U_2, \ldots, U_n$ são VAs uniformes em $(0, 1)$. O valor esperado da variável N é $E[N] = e$. Começamos a prova pela definição do valor esperado de N

$$E[N] = \sum_{n=2}^{\infty} n \cdot P[N = n]$$

N é maior ou igual a 2 pois cada uniforme é um número positivo menor que 1. Com isso podemos dizer que

$$P[S_n > 1] = P[S_{n-1} > 1] + P[N = n]$$

e manipulando obtemos,

$$P[N = n] = P[S_n > 1] - P[S_{n-1} > 1]$$

.

Para encontrar $P[S_n > 1]$ utilizamos o conceito de convolução. Seja f_n a função de densidade da variável aleatória S_n. Temos que

$$f_2(x) = \int_{-\infty}^{\infty} f_{U_1}(t) f_{U_2}(x-t)dt$$

Como $f_{U_1}(t) = 1$ no intervalo $0 \leq t \leq 1$ e 0 fora do intervalo,

$$f_2(x) = \int_0^1 f_{U_2}(x-t)dt$$

Além disso, o integrando vale 0 fora do intervalo $x - 1 \leq t \leq x$ e vale 1 no intervalo, portanto podemos reescrever a integral da seguinte forma,

$$f_2(x) = \int_0^x dt = x$$

Fazendo o mesmo para $f_3(x)$ teremos,

$$f_3(x) = \int_{-\infty}^{\infty} f_2(t) f_{U_3}(x-t)dt = \int_0^x t dt = \frac{x^2}{2}$$

Para $f_n(x)$, temos que

$$f_n(x) = \frac{x^{n-1}}{(n-1)!}$$

Portanto,

$$P[S_n > 1] = 1 - P[S_n \leq 1] = 1 - \int_0^1 f_n(t)dt = 1 - \frac{1}{n!}$$

Agora podemos obter $P[N = n]$,

$$P[N = n] = \left(1 - \frac{1}{n!}\right) - \left(1 - \frac{1}{(n-1)!}\right) = \frac{n-1}{n!}$$

Juntando tudo,

$$E[N] = \sum_{n=2}^{\infty} n \cdot \frac{n-1}{n!} = \sum_{n=2}^{\infty} \frac{1}{(n-2)!} = \sum_{n=0}^{\infty} \frac{1}{n!} = e$$

2. A variância de N é dada por

$$Var[N] = E[(N-e)^2]$$

Expandindo temos,

$$\begin{aligned}\sum_{n=2}^{\infty}(n-e)^2 \cdot \frac{n-1}{n!} &= \sum_{n=2}^{\infty}(n^2-2ne+e^2)\cdot\frac{1}{n(n-2)!}\\ &= \sum_{n=2}^{\infty}\frac{n}{(n-2)!} - \sum_{n=2}^{\infty}\frac{2e}{(n-2)!} + \sum_{n=2}^{\infty}\frac{e^2}{n(n-2)!}\end{aligned}$$

Resolvendo o primeiro somatório:

$$\begin{aligned}\sum_{n=2}^{\infty}\frac{n}{(n-2)!} &= \sum_{n=0}^{\infty}\frac{n+2}{n!}\\ &= 0 + \sum_{n=1}^{\infty}\frac{1}{(n-1)!} + 2\sum_{n=0}^{\infty}\frac{1}{(n)!}\\ &= e + 2e\\ &= 3e\end{aligned}$$

Resolvendo o segundo somatório:

$$\sum_{n=2}^{\infty}\frac{2e}{(n-2)!} = 2e\sum_{n=0}^{\infty}\frac{1}{n!} = 2e^2$$

Resolvendo o terceiro somatório:

$$\sum_{n=2}^{\infty}\frac{e^2}{n(n-2)!} = e^2\sum_{n=2}^{\infty}\frac{1}{n(n-2)!} = e^2\cdot 1 = e^2$$

Com isso temos que

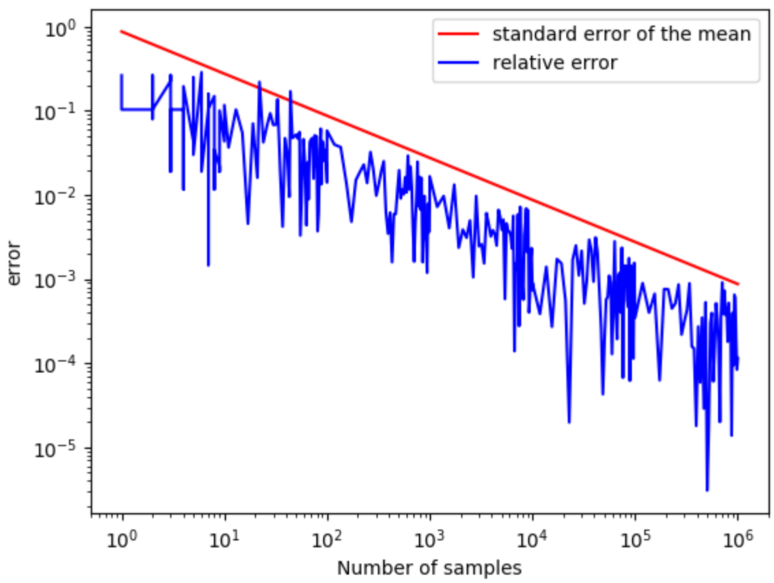
$$Var[N] = 3e - e^2$$

3.

```python
def estimateEuler(n_samples):
    sn = 0
    n = 0
    sample_mean = 0
    for s in range(n_samples):
        while sn <= 1:
            sn += np.random.random()
            n += 1
        sample_mean += n
```

```
        n = 0
        sn = 0
    return sample_mean/n_samples
```

4. Conforme mostra a figura abaixo, o erro relativo cai conforme o número de amostras aumenta assim como o erro padrão, porém pode-se observar que a variância é grande $(3e - e^2)$ devido aos picos na curva do erro relativo.



Questão 2: Transformada Inversa

Utilize o método da transformada inversa para gerar amostras de uma v.a. X com as seguintes densidades:

1. Distribuição exponencial com parâmetro $\lambda > 0$, cuja função densidade é dada por $f_X(x) = \lambda e^{-\lambda x}$, para $x \geq 0$.

2. Distribuição de Pareto com parâmetros $x_0 > 0$ e $\alpha > 0$, cuja função densidade é dada por $f_X(x) = \frac{\alpha x_0^\alpha}{x^{\alpha+1}}$, para $x \geq x_0$.

Solução:

1.

$$
\begin{aligned}
F_X(i) = P[X \leq i] &= \int_0^i \lambda e^{-\lambda x} dx \\
&= \lambda \int_0^i e^{-\lambda x} dx \\
&= e^{-\lambda x}\Big|_0^i \\
&= 1 - e^{-\lambda i}
\end{aligned}
$$

$$
\begin{aligned}
1 - e^{-\lambda i} &= u \\
e^{-\lambda i} &= 1 - u \\
\log(e^{-\lambda i}) &= \log(1-u) \\
-\lambda i &= \log(1-u) \\
i &= \frac{\log(1-u)}{\lambda}
\end{aligned}
$$

Então

$$
F_x^{-1}(u) = \frac{\log(1-u)}{\lambda}
$$

com $u \sim Unif(0,1)$

2.

$$
\begin{aligned}
F_X(i) = P[X \leq i] &= \int_{x_0}^i \frac{\alpha x_0^\alpha}{x^{\alpha+1}} dx \\
&= \alpha x_0^\alpha \int_{x_0}^i x^{-(\alpha+1)} dx \\
&= x_0^\alpha \cdot -x^\alpha\Big|_{x_0}^i \\
&= x_0^\alpha \cdot x_0^{-\alpha} - x_0^\alpha \cdot x^{-\alpha} \\
&= 1 - \left(\frac{x_0}{x}\right)^\alpha
\end{aligned}
$$

$$
\begin{aligned}
u &= 1 - \left(\frac{x_0}{x}\right)^\alpha \\
\left(\frac{x_0}{x}\right)^\alpha &= 1 - u \\
\frac{x_0}{x} &= (1-u)^{\frac{1}{\alpha}} \\
i &= \frac{x_0}{(1-u)^{\frac{1}{\alpha}}}
\end{aligned}
$$

Então

$$F_x^{-1}(u) = \frac{x_0}{(1-u)^{\frac{1}{\alpha}}}$$

com $u \sim Unif(0,1)$

Questão 3: Contando domínio na Web
Quantos domínios Web existem dentro da UFRJ? Mais precisamente, quantos domínios existem dentro do padrão de nomes *http://www.[a-z](k).ufrj.br*, onde *[a-z](k)* é qualquer sequência de caracteres de comprimento k ou menor? Construa um algoritmo de Monte Carlo para calcular este número.

1. Descreva a variável aleatória cujo valor esperado está relacionado com a medida de interesse. Obtenha o valor esperado da sua variável aleatória.
2. Calcule a variância dessa variável aleatória.
3. Implemente o método de Monte Carlo para gerar amostras reais da sua variável aleatória. Ou seja, você deve consultar o domínio gerado para determinar se o mesmo existe (utilize uma biblioteca para isto).
4. Assuma que $k = 4$. Seja $\hat{w}_n$ o valor do estimador do número de domínios após n amostras. Trace um gráfico de $\hat{w}_n$ em função de n para $n = 1, \ldots, 10^4$. O que você pode dizer sobre a convergência de $\hat{w}_n$?

Solução:

1. A_{t_k} = número de domínios web na UFRJ com até k caracteres
   W = conjunto de todas as sequências de caracteres de comprimento k ou menor
   N_k = número de sequências de caracteres de comprimento k ou menor

   $$g(w) = \begin{cases} 1, & \text{se } w \text{ é domínio} \\ 0, & \text{caso contrário} \end{cases}$$

   Com isso temos

   $$A_{t_k} = \sum_{w=1}^{N_k} g(w)$$

   Agora seja S uma VA uniforme em $(1, N_k)$,

   $$E[g(S)] = \sum_{s=1}^{N_k} P[S=s] \cdot g(s) = \frac{1}{N_k} \sum_{s=1}^{N_k} g(s) = \frac{A_{t_k}}{N_k}$$

Podemos estimar o total de domínios a partir de

$$A_{t_k} = E[g(S)]N_k$$

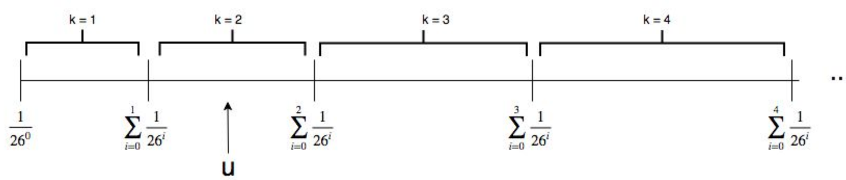
portanto, o nosso estimador é

$$M_n = \frac{1}{n}\sum_{s=1}^{n} g(s)$$

2. A variância do estimador é

$$Var[M_n] = \frac{1}{n}\sum_{s=1}^{n} Var[g(s)] = \frac{1}{N_k}\cdot\left(1 - \frac{1}{N_k}\right) = \frac{N_k - 1}{N_k^2}$$

3. O algoritmo foi feito de forma que primeiro gera-se uma uniforme entre 1 e $N_k + 1$ para saber quantos caracteres o domínio a ser gerado terá, pois o intervalo é dividido de acordo com a figura abaixo:



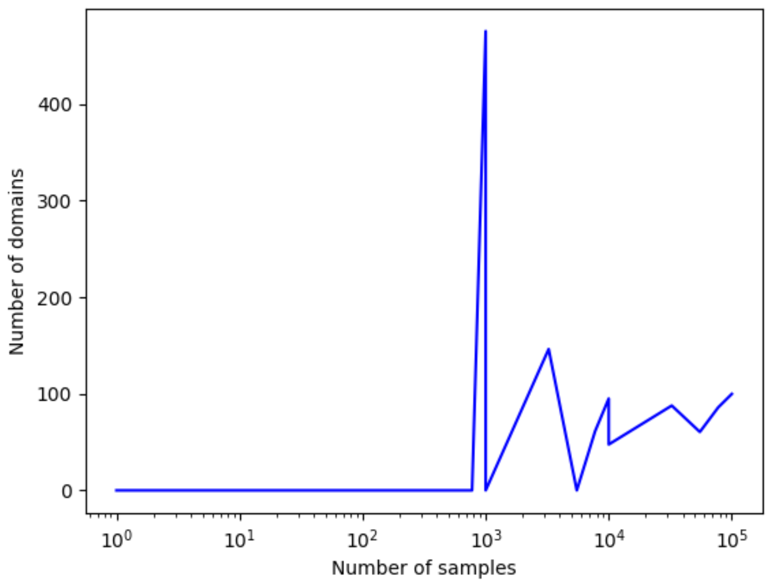
Uma vez definido o tamanho do domínio a ser gerado, o domínio é gerado com uma uniforme entre 1 e 26 para cada caractere da string do domínio. O algoritmo a seguir mostra como foi a implementação:

```
def estimateDomains(k, n_samples, N_k):
    Mn = 0
    for _ in tqdm(range(n_samples), desc='generating samples'):
        u = np.random.randint(1,N_k+1)
        t = 26
        d = 0
        for i in range(k):
            if u < t+1:
                d = i+1
                break
            else:
                t += 26**(i+2)
        domain = ''
        for i in range(d):
            domain += chr(np.random.randint(1,27)+96)

        url = 'www.' + domain + '.ufrj.br'
        try:
            request = requests.get(url, headers={'Connection':
                'close'})
```

```python
            res = request.status_code
            if res == 200:
                Mn += 1
        except:
            pass
    Mn = Mn/n_samples
    return Mn*N_k
```

4. Como mostra a figura a seguir, o estimador ainda não convergiu, mas com mais amostras a curva vai oscilar até convergir pro número de domínios com $k = 4$



Questão 4: Gerando amostras Normais
Seja Z uma variável aleatória com distribuição Normal com média 0 e variância 1. Em particular, a função densidade de Z é dada por $f_Z(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, com $-\infty < x < \infty$. Repare que Z assume valores positivos e negativos, mas com caudas que possuem a mesma probabilidade. Ou seja, $P[Z \geq z] = P[Z \leq -z]$, para todo $z \geq 0$. Construa um gerador de números aleatórios para Z. Dica: Utilize o método de amostragem por rejeição e a distribuição exponencial!

Solução:

Temos que $f_Z(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. Podemos usar a distribuição exponencial como envelope para a distribuição normal. Então $g_Z(x) = \lambda e^{-\lambda x}$.

Como $c \geq \frac{f_Z(x)}{g_Z(x)}$, precisamos encontra o máximo dessa razão, ou seja, $c = max\left\{\frac{f_Z(x)}{g_Z(x)}\right\}$.

Derivando a expressão em relação a x obtemos $\frac{1}{\lambda\sqrt{2\pi}} \cdot (-x + \lambda) e^{-x^2/2 + \lambda x}$. Para obter o máximo dessa expressão, igualamos ela a 0.

$$\frac{1}{\lambda\sqrt{2\pi}} \cdot (-x+\lambda)e^{-x^2/2+\lambda x} = 0$$
$$xe^{-x^2/2+\lambda x} = \lambda e^{-x^2/2+\lambda x}$$
$$x = \lambda$$

Para fins de simplificação, foi usado $\lambda = 1$. Portanto $c = \frac{f_Z(1)}{g_Z(1)}$. O algoritmo consiste em gerar uma amostra x de uma distribuição exponencial com $\lambda = 1$ utilizando o método da transformada inversa obtido na questão 2, gerar uma uniforme u entre $(0, c * g_Z(x))$. Se $u < f_Z(x)$, a amostra gerada é aceita como amostra da distribuição normal. Para saber se ela está do lado esquerdo ou direito da cauda, é "jogada uma moeda" para decidir se a amostra será positiva ou negativa.

A implementação do algoritmo é mostrada a seguir:

```
def generateExpRV(scale):
    u = np.random.uniform(0,1)
    return -np.log(1-u)/scale

def expDistribution(x, scale=1):
    return scale*np.exp(-scale*x)

def standardNormal(x):
    return (1/(np.sqrt(2*np.pi)))*np.exp(-((x**2)/2))

def generateNormalSamples(scale, n_samples=1):
    samples = []
    c = standardNormal(scale)/expDistribution(scale,scale)
    for n in range(n_samples):
        while True:
            x = generateExpRV(scale)
            u = np.random.uniform(0,c*expDistribution(x))
            if u < standardNormal(x):
                if np.random.random() > 0.5:
                    samples.append(x)
                    break
                else:
                    samples.append(-x)
                    break
    return samples
```

Questão 5: Estimando somas

Considere o problema visto em aula, de aplicar o método de Monte Carlo para estimar o valor de $G_N = \sum_{i=1}^{N} i \log(i)$. Use sua intuição para encontrar um função de probabilidade proponente, $h(i)$, que tenha variância inferior ao melhor estimador visto em aula.

1. Assuma que $N = 1000$. Calcule numericamente o segundo momento do seu estimador.

2. Implemente o método de Monte Carlo para estimar o valor de G_N . Trace um gráfico do erro relativo do estimador, em função de $n = 1, \ldots, 10^6$ (calcule o valor exato da soma para determinar o erro relativo).

Solução:

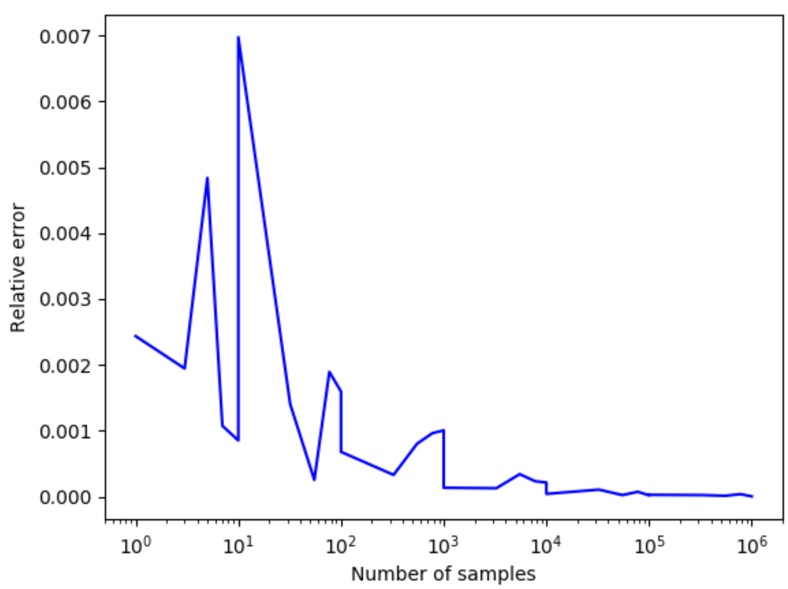
1. A função $h(i)$ escolhida é $h(i) = \frac{i^{1.2}}{K}$ onde $K = 1^{1.2} + 2^{1.2} + \cdots + N^{1.2}$

   Dessa forma o segundo momento é igual a

$$K \sum_{i=1}^{N} \frac{i^2 \log^2(i)}{i^{1.2}} = K \sum_{i=1}^{N} i^{0.8} \log^2(i)$$

   O termo $i^{0.8}$ vai diminuir o somatório sem que o K seja muito grande. O valor numérico obtido para o segundo momento foi 1.02896×10^{13} menor do que a melhor opção obtida em aula que era 1.03×10^{13}

2. A figura a seguir mostra como o erro relativo se aproxima de 0 conforme o número de amostras aumenta.



Questão 6: Gerando subconjuntos

Considere S um espaço amostral dado por todos os subconjunto de tamanho k dentre n objetos. Assuma que cada elemento deste espaço amostral tenha a mesma probabilidade, dada por $\frac{1}{|S|}$. Descreva um algoritmo eficiente para gerar amostras deste espaço. Dica: pense em permutação!

Solução:

Um algoritmo eficiente para esse problema é dado a seguir:

```
procedure GeraSubconjunto(n, k)
    vetor permuta[1, ..., n]
    for i ← 1, n do
        permuta[i] ← i
    end for
    vetor subconjunto[1, ..., k]
    for i ← 0, n − 1 do
        j ← Unif(0, n − i)
        tmp ← permuta[j]
        permuta[j] ← permuta[n − 1]
        permuta[n − 1] ← tmp
        subconjunto[i] ← tmp
        if i == k − 1 then
            break
        end if
    end for
    return subconjunto
end procedure
```

A complexidade de tempo do algoritmo depende do tamanho do subcojunto desejado e é igual a O(k).

Questão 7: Estimando probabilidade de caudas
Considere o problema de calcular a probabilidade de cauda de uma determinada distribuição, $P[X \geq x]$.

1. Descreva um algoritmo de Monte Carlo simples para estimar $P[X \geq x]$ para um determinado $x > 0$ fixo.

2. Calcule a variância deste estimador.

3. Para x grande, $P[X \geq x]$ pode ser bem pequena. Descreva uma abordagem utilizando importance sampling para reduzir a variância do estimador acima.

4. Calcule a variância deste novo estimador

5. Assumindo que X possui distribuição Normal padrão, implemente os dois algoritmos acima e calcule o valor do estimador para $x = 5$. Trace um gráfico do valor estimado por cada algoritmo em função de $n = 1, \ldots, 10^6$.

Solução:

1. Considerando que eu sei a distribuição e consigo gerar amostras dela, no caso contínuo, $P[X \geq x] = \int_x^\infty h(s)f(s)ds$, onde $h(s)$ é a seguinte função:

$$h(s) = \begin{cases} 1, & \text{se } s \geq x \\ 0, & \text{caso contrário} \end{cases}$$

   O algoritmo consiste em gerar amostras de X e obter a fração de amostras maiores que x. Ou seja,

$$M_n = \frac{1}{n}\sum_{i=1}^{n} h(X_i)$$

2. A variância do estimador M_n é dada por:

$$Var[M_n] = \frac{1}{n^2}\sum_{i=1}^{n} Var[h(X_i)] = \frac{P[X \geq x]P[X < x]}{n}$$

3. Uma abordagem seria utilizar uma nova VA Y com distribuição Normal g com $\sigma = 1$ e $\mu = x$. O novo estimador será

$$M_n = \frac{1}{n}\sum_{i=1}^{n} \frac{h(Y_i)f(Y_i)}{g(Y_i)}$$

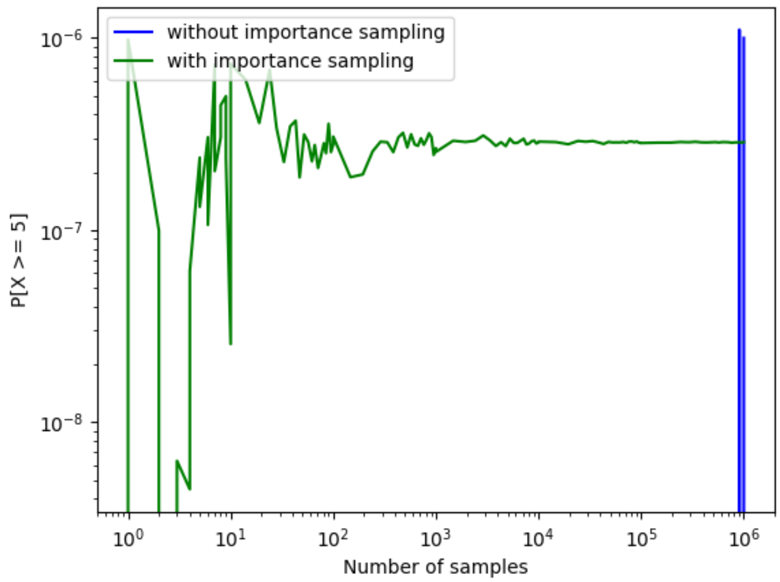
4.
$$Var[M_n] = \frac{1}{n^2}\sum_{i=1}^{n} Var\left[\frac{h(Y_i)f(Y_i)}{g(Y_i)}\right]$$

$$Var\left[\frac{h(Y)f(Y)}{g(Y)}\right] = \int_x^\infty \frac{h(s)f(s)}{g(s)}ds - E\left[\frac{h(Y)f(Y)}{g(Y)}\right]$$

   O novo segundo momento é dado por

$$\frac{\sqrt{2\pi}}{2\pi}\int_x^\infty e^{-s^2-2sx+x^2}ds$$

5. A figura a seguir mostra que o algoritmo utilizando *importance sampling* converge para o valor de $P[X \geq 5]$ enquanto que o algoritmo que não utiliza *importance sampling* praticamente não consegue gerar amostras na região $[5, \infty)$.



without importance sampling
with importance sampling
P[X >= 5]
10^{-6}
10^{-7}
10^{-8}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
Number of samples