Relatório do Exercício Programa 3: Métodos de Monte Carlo - Quasi-Random

Erik Davino Vincent - BMAC - Turma 54
NUSP: 10736584

April 25, 2019

IME - USP

Contents

1 Introdução

O seguinte texto tem por objetivo a análise de quatro métodos de Monte Carlo, utilizando geradores *Quasi-Random*, visando a comparação entre a eficiência deles e a de geradores *Pseudo-Aleatórios*, além do quanto são otimizados. Será discutido o tempo de computação de cada método, a comparação direta de seus resultados e a implementação para encontrar resultados bons, com erro $\leq 1\%$.

Os métodos serão aplicados para calcular a estimativa da integral da função, no intervalo $[0, 1]$, definida por:

$$f(x) = e^{-\alpha x} \cos \beta x$$

onde $\alpha = 0.10736584$, o meu número USP e $\beta = 0.50886257$, o meu número de RG.

1.1 Critério de Parada

A escolha do critério de parada foi simples: se o programa estimar o erro que eu quero e este for $\leq 1\%$, o programa interrompe o processo e devolve os resultados. Exatamente o mesmo critério utilizado para o EP 2.

2 Gerador escolhido

A primeira decisão a ser tomada foi o gerador *Quasi-Random* a ser utilizado. Encontrei uma boa biblioteca de geradores nos seguintes links: Quasi-Random-Sobol; Quasi-Random-Halton; Quasi-Random-VanDerCorput. (Cada um dos títulos anteriores é um link).

O gerador de sequencias de Sobol me pareceu interessante no primeiro momento, pois seu *plotting* pode ser muito semelhante ao de um gerador *pseudo-aleatório* uniforme normal, além de ser capaz de ter diferentes 'caras', definidas por uma *seed*. Porém, o gerador que obtive era limitado a gerar sequencias de no máximo $n = 40$ valores. Logo, na implementação fiz com que para $n > 40$ outra lista com $n - 40$ valores fosse gerada, de forma recursiva, e concatenada à lista original (vide algorítimo). Dessa forma, também gerei um efeito de aleatoriedade, pois fiz com que a *seed* fosse alterada para cada loop. Mas o problema que surgiu por essa implementação foi o tempo demasiadamente longo para a geração das listas, e consequentemente do calculo para resolver a integral e achar n suficientemente grande para um erro $\epsilon < 1\%$.

Para o método *Crud*, utilizando as sequencias de Sobol, obtive os seguintes resultados, para encontrar $\epsilon < 1\%$:

Tempo de computação: 96 segundos.
$\epsilon = 0.10499686985947774\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = 0.16626400498125143.
$n = 10000$ (lembrando que n cresce a uma taxa de 10 vezes).
$\hat{\gamma} = 0.7838860121075741$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

Como podemos ver o resultado é até mesmo pior do que vemos para uma distribuição *pseudo-aleatória* uniforme, em termos de tempo de computação (para encontrar $\epsilon < 1\%$):

Tempo de computação: 0.240 segundos.
$\epsilon = 0.4822873755308176\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = 0.16626400498125143.
$n = 46656$ (lembrando que n cresce a uma taxa de $2\sqrt{10}$ vezes).
$\hat{\gamma} = 0.7824057647774996$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

Por tal razão, fiz o mesmo teste para uma sequencia de *VanDerCorput*, e os resultados foram tão bons, que utilizei a sequencia para o resto dos métodos. Vejamos os resultados (para encontrar $\epsilon < 1\%$):

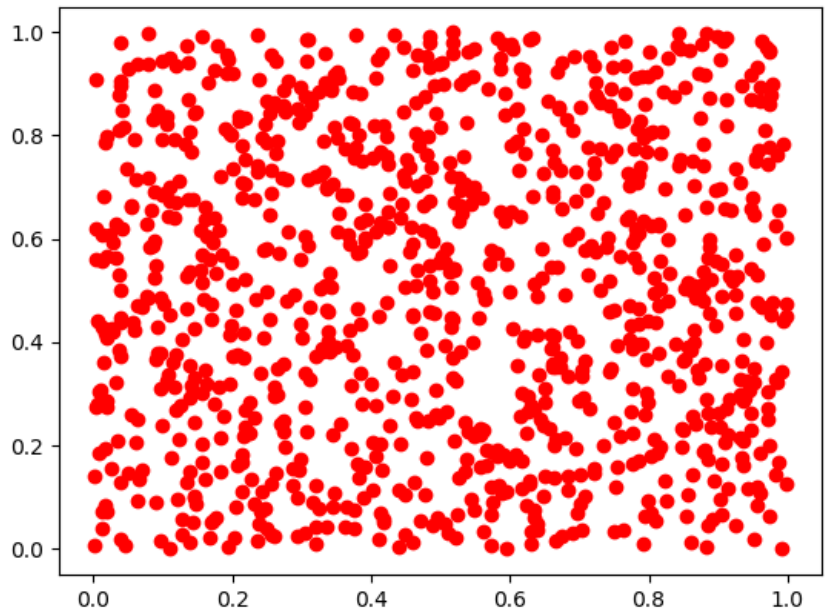
Tempo de computação: 0.371 segundos.
$\epsilon = 0.4814989332265667\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = 0.16313370287435794.
$n = 46656$ (lembrando que n cresce a uma taxa de $2\sqrt{10}$ vezes).
$\hat{\gamma} = 0.7824488038971255$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

Os resultados não são melhores do que da uniforme, para esse n, porém são muito semelhantes, e fixos para dados n, uma vez que a sequencia é sempre gerada da mesma maneira. Isso pode ser uma vantagem em relação ao gerador uniforme, uma vez que não possui aleatoriedade nos resultados. Para dado n, sempre terei valores garantidos.

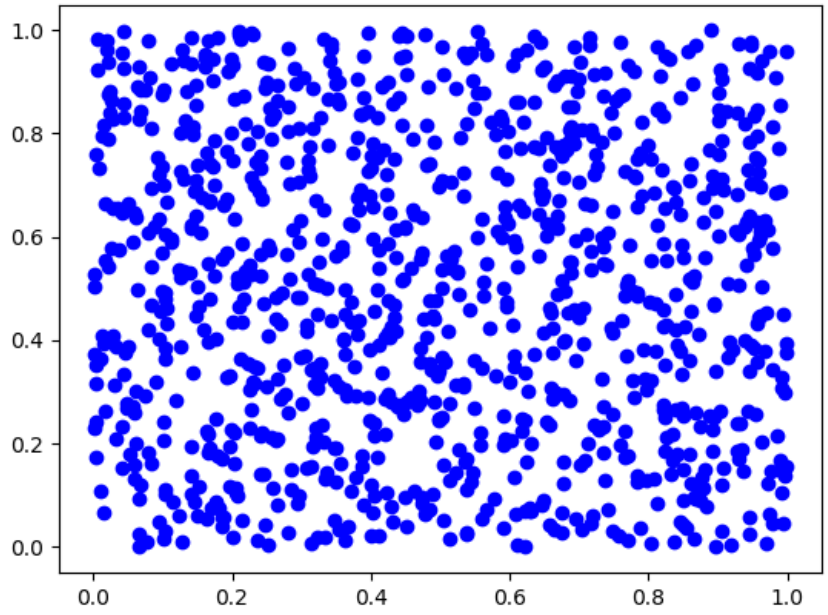
2.1 Gráficos das distribuições

Antes de prosseguirmos, vejamos abaixo as distribuições/sequencias mencionadas acima, em uma área 1x1:

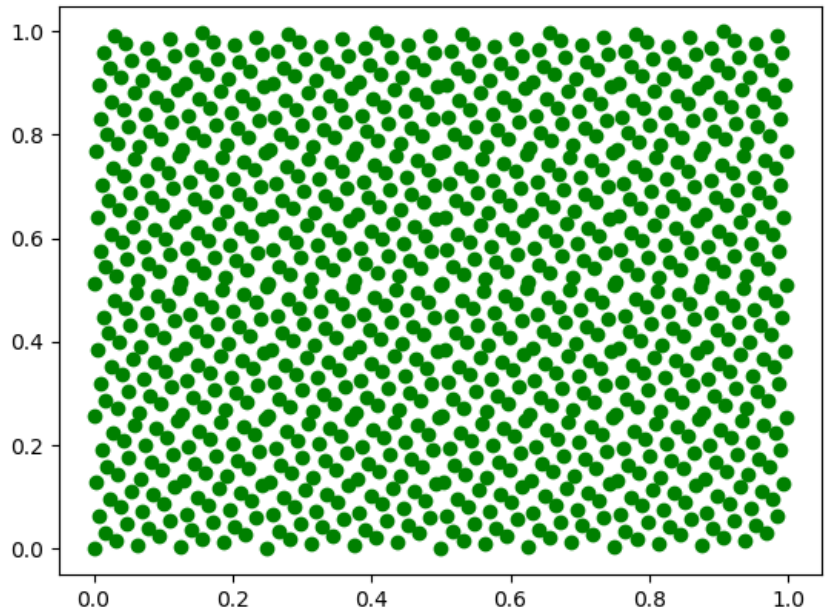
UNIFORME:



SOBOL:



VANDERCORPUT:

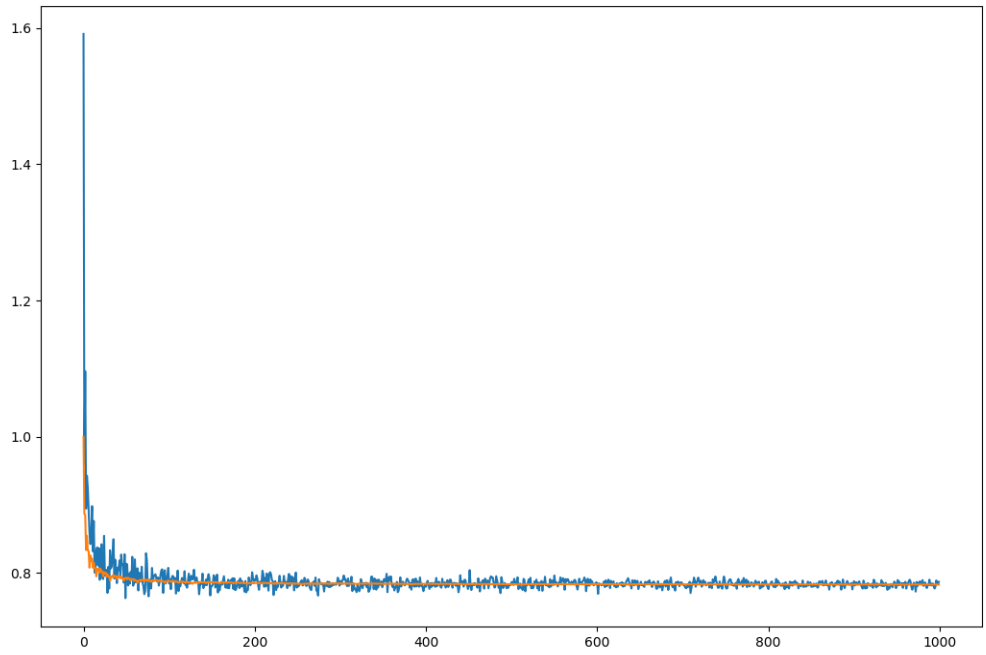


Como vemos, o gerador de Sobol e uniforme praticamente não apresentam diferenças notáveis. Além disso, vemos como o VanDerCorput possui distribuição muito bem comportada, o que, creio eu, permite um calculo muito mais preciso, especialmente para o caso do *Hit-Or-Miss*. Verificaremos isso mais a frente.

3 Método *Crud*

Em 2, praticamente cobrimos todo o básico para o método *Crud*. Em termos de implementação, é idêntica a do EP 2, porém o gerador *quasi-random* foi utilizado. Resta então fazer uma analise mais regrada, em termos de medidas, para qual gerador resulta em convergência mais rápida para $\hat{\gamma}$. Para isso, analisemos os gráficos:

Valor de $\hat{\gamma}$ até $n = 1000$:
laranja: *quasi-random*; azul: uniforme.



Como podemos ver, o gerador *quasi-random* leva, por pouco, $\hat{\gamma}$ para o seu valor final mais rapidamente. Isso é, para um n menor, o *gaṁma quasi-random* está mais próximo do valor real da integral; ele converge mais rápido. Além disso ele é estável, diferente do aleatório uniforme, o que permite que mais vezes, o valor de $\hat{\gamma}$ esteja correto. De certa forma, a probabilidade do erro estar correto, por ser estimado, é maior. Isso significa que, pelo menos para esse método, o gerador *quasi-aleatório* aumenta a eficiência. Não precisamos nos preocupar tanto com tempo de computação, uma vez que a convergência é mais rápida, e o tamanho de n pode ser menor, para uma mesma precisão (ou até melhor).

4 Método *Hit-or-Miss*

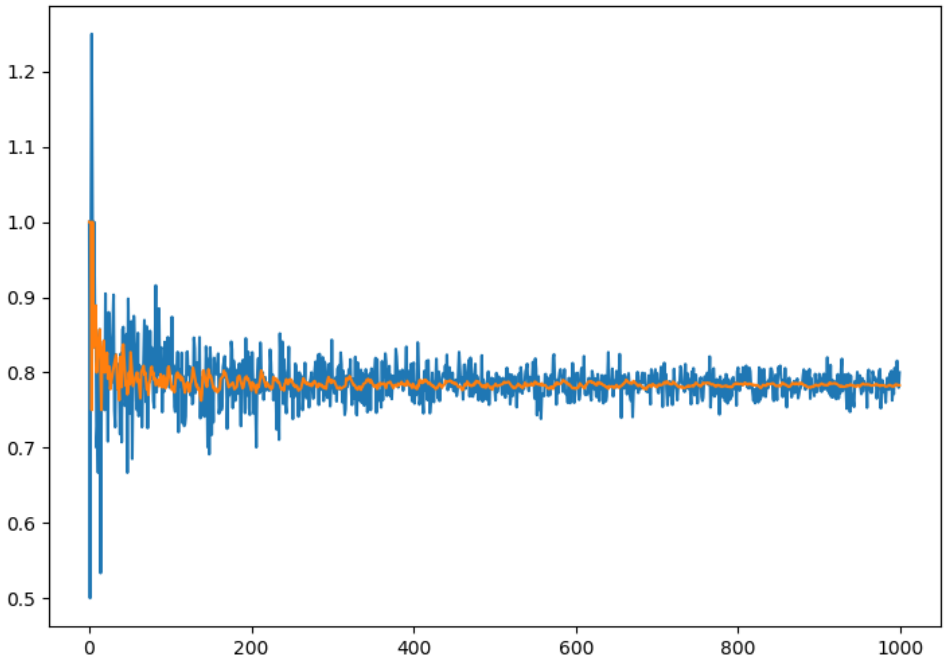
A implementação utilizada foi a mesma que a do EP 2, porém, com o gerador aleatório trocado por um *quasi-random* VanDerCorput. vejamos os resultados obtidos:
(Para encontrar erro $\epsilon < 1\%$)

Uniforme:
Tempo de computação: 0.132 segundos.
$\epsilon = 0.6818756060403957\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = 0.002648060605982119.
$n = 24300$. (lembrando que n cresce a uma taxa de $2\sqrt{10}$ vezes).
$\hat{\gamma} = 0.7821399176954732$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

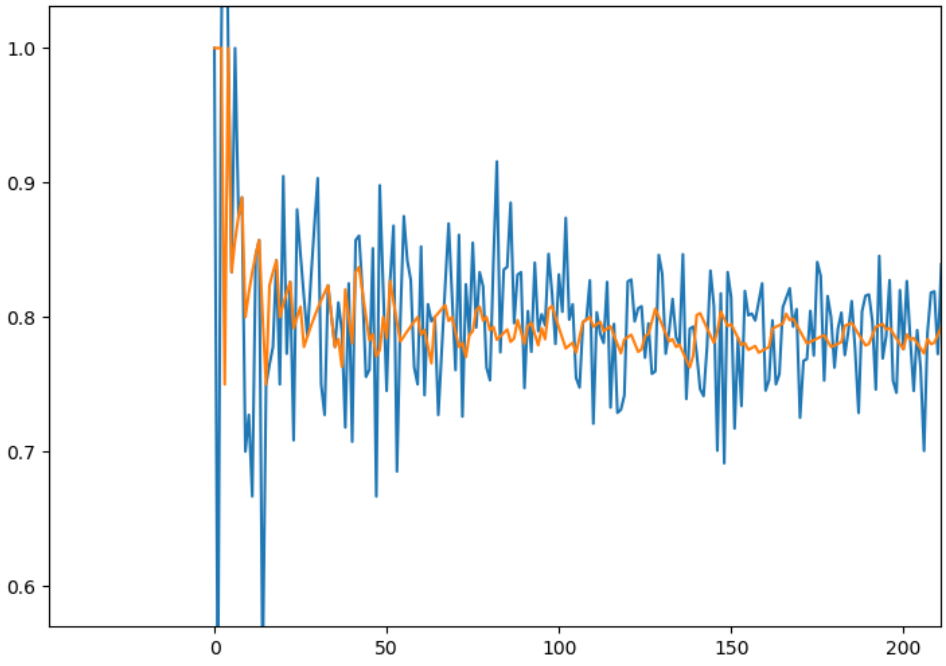
***VanDerCorput*:**
Tempo de computação: 0.209 segundos.
$\epsilon = 0.6816896770211912\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = 0.0026473385515386064.
$n = 24300$. (lembrando que n cresce a uma taxa de $2\sqrt{10}$ vezes).
$\hat{\gamma} = 0.7823045267489712$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

Vemos novamente resultados muito semelhantes entre as distribuições, com a opção *quasi-random* apresentando erro e variância amostral menores do que a opção uniforme aleatória, com única desvantagem o tempo de computação levemente maior. Porém, para n menor, vemos que a diferença de eficiência entre os métodos é ainda maior. Vejamos melhor com um gráfico:

Valor de $\hat{\gamma}$ até $n = 1000$:
laranja: *quasi-random*; azul: uniforme.



A primeira coisa que podemos notar é que ambos os gráficos oscilam muito, e de forma decrescente. Isso é esperado do método, pois ele possui menos informações sobre $f(x)$. Diferente do gráfico na 3, vemos que a convergência se trata menos de qual 'chega' mais rápido (decai mais rapidamente) para o valor real da integral, mas sim de qual oscila menos, conforme chega mais próximo do valor real. Como podemos ver, a linha que representa o método pelo gerador *quasi-random* oscila muito menos, o que nos permite, para cada n, ter mais certeza de que $\hat{\gamma}$ está fato mais próximos de γ. Vemos que essa diferença é ainda mais forte para n menor. Vejamos um *zoom* do gráfico acima, onde n é pequeno:



5 *Importance Sampling*

Esse foi o método que mais sofreu modificações, porém não tao grandes. Primeiramente, não há geradores *quasi-random* para todas as distribuições, logo, tive que criar a minha própria distribuição. De resto, a implementação é a mesma que a do EP 2.

O método em que me baseei para criação de geradores com distribuição de densidade de probabilidade aleatória para uma função $g(x)$ é feito da seguinte forma:

$$\text{Seja } g(x) \text{ função integrável, tal que } G(x) = \int g(x)dx, \text{ distribuição de densidade acumulada;} \quad (1)$$

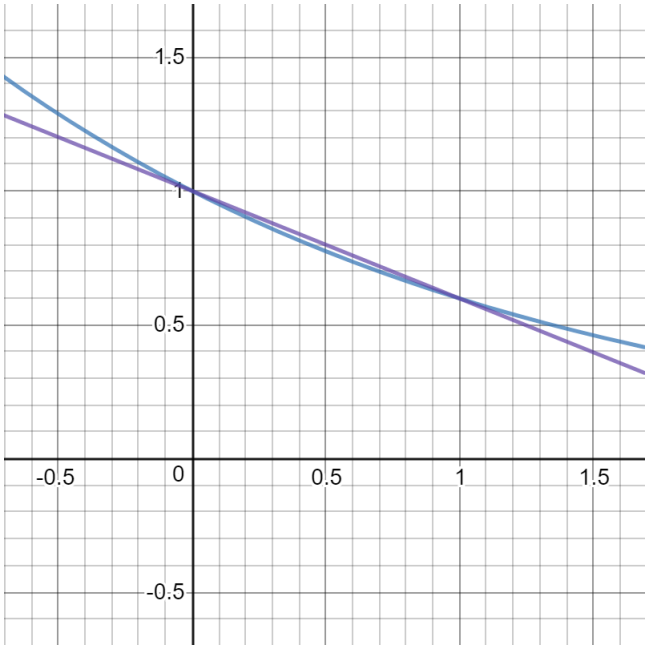
$$\text{Se } G^{-1}(x) \text{ existe, e } x_i \sim U[0,1], \text{ então:} \quad (2)$$

$$u_i = G^{-1}(x_i) \Rightarrow u_i \sim g(x) \quad (3)$$

Como não estamos utilizando uma distribuição uniforme, e a *qusi-random VanDerCorput* simula bem uma uniforme, basta substituir $x_i \sim U[0,1]$ por $x_i \sim VanDerCorput$, obtendo uma distribuição *quasi-random*, com 'cara' de $g(x)$.

5.1 Escolha da $g(x)$

Dessa vez, a escolha de $g(x)$ foi limitada a funções integráveis, e que as integrais fossem inversíveis. Não utilizei a distribuição beta, portanto. Ao invés disso, como a minha curva convenientemente se assemelha muito a uma reta, escolhi uma reta que passasse pelos pontos $(0, f(0))$ e $(1, f(1))$. A reta encontrada foi $g(x) = -0.40228x + 1$. Vejamos o gráfico:



Porém, para essa reta ser uma distribuição de probabilidade em $[0,1]$, vale notar que $G(x)|_0^1 = 1$. O resultado que obtive primeiramente foi $G(x) = x - 0.20114x^2$, onde $G(x)|_0^1 = 0.79886$. Para corrigir esse problema, somei ao valor obtido dessa integração o valor suficiente para resultar em 1, que era 0.20114, à $g(x)$. Logo, minha $g(x)$ final ficou $g(x) = -0.40228x + 1.20114$, com $G(x) = 1.20114x - 0.20114x^2$. Vejamos o gráfico na página seguinte:

5.1.1 Definição do gerador

Com $g(x)$ e $G(x)$ definidas, bastava então criar o gerador *quasi-random*. Para isso, primeiramente defini com a ajuda do *Wolframalpha*, $G^{-1}(x)$:

$$G^{-1}(x) = 2.98583 \pm 0.0000497166\sqrt{3.60684 \cdot 10^9 - 2.0114 \cdot 10^9 x}$$

A função criada (vide programa), consiste em transformar cada um dos x_i elementos de uma sequencia de *VanDerCorput* em $G^{-1}(x_i)$, e retornar uma nova lista *quasi-random*.

5.2 Resultados obtidos

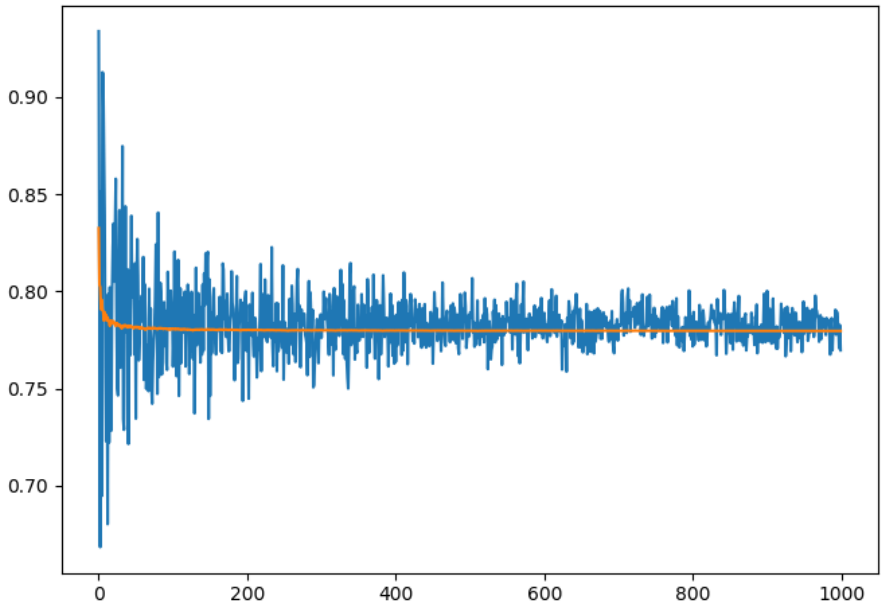
Beta
Tempo de computação: 2.51 segundos.
$\epsilon = 0.4681088501408622\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = 0.044354512807987204.
$n = 7776$. (lembrando que n cresce a uma taxa de $2\sqrt{10}$ vezes).
$\hat{\gamma} = 0.7853107345425413$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

***VanDerCorput - g(x)*:**
Tempo de computação: 0.00097 segundos.
$\epsilon = 0.59559405970736\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = 0.000664846013250364.
$n = 72$. (lembrando que n cresce a uma taxa de $2\sqrt{10}$ vezes).
$\hat{\gamma} = 0.783542216119383$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

Esse resultado me espantou. Não há sombra de duvidas para a maior do eficiência do *quasi-random* para este método, basta ver os resultados (a comparação ideal seria utilizar o gerador de distribuição $g(x)$ aleatório para encontrar os resultados do *importance sampling* aleatório). Não apenas em relação ao gerador

Beta e o método aleatório, mas também em relação a todos os outros métodos apresentados. Vejamos os gráficos de convergência para reforçar a conclusão sobre esse resultado:

Valor de $\hat{\gamma}$ até $n = 1000$:
laranja: *quasi-random*; azul: uniforme.



6 *Control Variance*

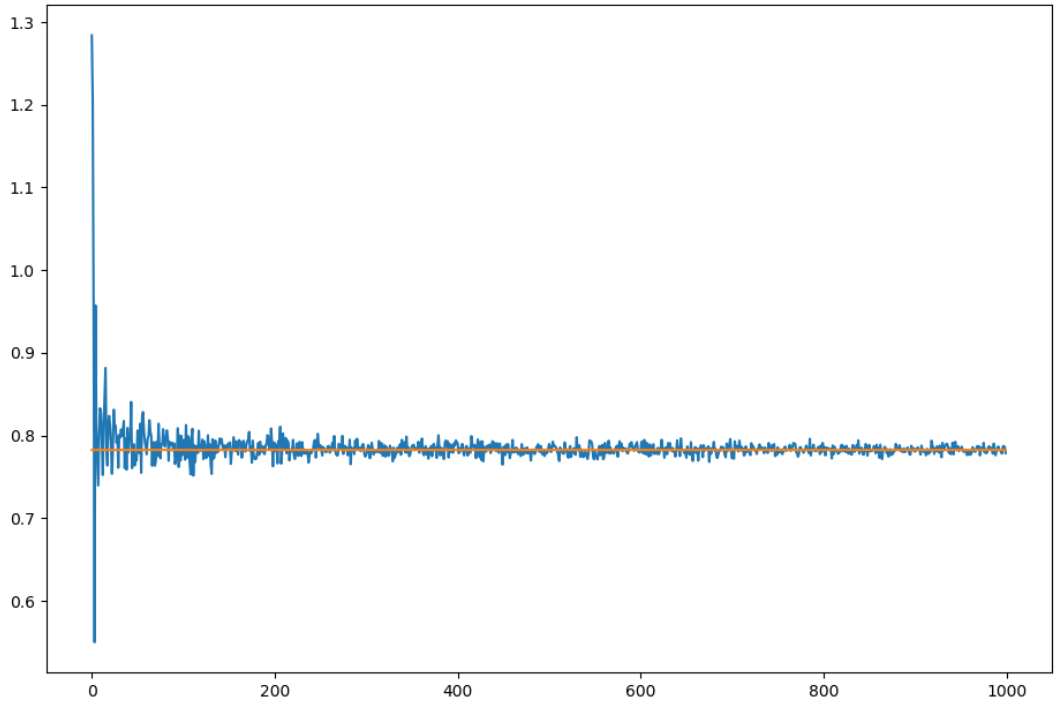
O ultimo método, como os demais, também seguiu a mesma implementação do EP 2, apenas com o gerador uniforme, trocado pelo *quasi-random*. Vejamos os resultados obtidos:

Uniforme
Tempo de computação: 2.257 segundos.
$\epsilon = 0.473428652971247\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = 0.04536837112649993.
$n = 7776$. (lembrando que n cresce a uma taxa de $2\sqrt{10}$ vezes).
$\hat{\gamma} = 0.7852965541990419$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

***VanDerCorput - g(x)*:**
Tempo de computação: 0.005983114242553711 segundos.
$\epsilon = 1.2057739535976042 \cdot 10^{-05}\%$ (lembrando que ϵ é a estimação do erro, assumindo distribuição normal).
Variância amostral = $4.385380947269368 \cdot 10^{-15}$.
$n = 2$. (lembrando que n cresce a uma taxa de $2\sqrt{10}$ vezes).
$\hat{\gamma} = 0.7824294083396494$ (lembrando que $\hat{\gamma}$ é a estimação da integral de $f(x)$).

Vejamos primeiramente o gráfico:

Valor de $\hat{\gamma}$ até $n = 1000$:
laranja: *quasi-random*; azul: uniforme.



O resultado pode parecer surpreendente, porém, pela forma como funciona esse método, o que acontece é que não importa o n, se usarmos o *quasi-random*. Na verdade o fator que afeta o resultado dessa estimativa é somente a precisão de $\int_0^1 \phi(x)$ (lembrando que $\phi(x)$ é a função de controle que se assemelha a $f(x)$). Minha suposição é de que $\phi(x)$ se assemelha tanto a $f(x)$ que a variância e covariância de $f(x)$ e $\phi(x)$ aproximam $\hat{\gamma}$ de $\int \phi(x)dx$ para um n muito pequeno, isso é: 1. Assim, todo o cálculo depende então da precisão de $\int \phi(x)dx$.

Isso me leva a supor que, caso não seja possível encontrar uma função alternativa $\phi(x)$ tão próxima de $f(x)$, esse resultado não seria tão bom.

7 Consideração final

Pelos resultados obtidos, é seguro afirmar que, ao menos para a função em questão os geradores *quasi-random* produzem resultados com precisão muito melhor, de forma geral. É difícil tirar conclusões do que aconteceria caso usássemos outras funções. Mas minha melhor suposição é de que, o método *Hit-or-Miss* e o *Importance Sampling* seriam os que produziriam sempre os melhores resultados, se comparado ao uso dos geradores aleatórios convencionais.