PROJECT REPORT

Stochastic approximation in mathematical finance

Bollero Francesco

École Polytechnique Fédérale de Lausanne

January 19, 2022

1 Introduction

Stochastic approximation methods are a family of iterative methods typically used for root-finding problems or for optimization problems. To be more precise stochastic approximation algorithms deal with a function of the form $f(\theta) = \mathrm{E}_\xi[F(\theta, \xi)]$ which is the expected value of a function depending on a random variable. The goal is to recover properties of such a function f without evaluating it directly.

2 Robbins Monro algorithm

2.1 General introduction to the method

The Robbins Monro algorithm was one of the first algorithms introduced for stochastic approximation. It presents a method to find the root of a function that is represented as an expected value. Suppose to have a function $J(\theta)$, that we can not directly observe, and a constant α, such that $J(\theta) = \alpha$ has a unique solution in $\theta*$. Suppose also that we can obtain measurements of the random variable $f_\theta(X)$ where $J(\theta) = \mathrm{E}[f_\theta(X)]$. Then *Robbins Monro* algorithm would find the root in the following way:

$$\theta_{n+1} = \theta_n - \alpha_n(\hat{J}(\theta_n) - \alpha) \tag{1}$$

Here the series α_n is a sequence of positive step sizes. This algorithm converges to the solution $\theta*$ in L^2. It was then proved that it converges with probability one if the following conditions are satisfied:

- $f_\theta(x)$ is uniformly bounded
- $J(\theta)$ is non decreasing function
- $J'(\theta*)$ exists and is positive
- the sequence α_n respect the following conditions:

$$\sum_{n=0}^{\infty} \alpha_n = \infty, \sum_{n=0}^{\infty} \alpha_n^2 < \infty \tag{2}$$

2.2 Robbins Monro to extract volatility in option pricing

Before introducing the main problem of this project we introduce the stock price $\{S_t, t \in [0, T]\}$, which is the solution of

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{3}$$

It is possible to show that the solution to this stochastic differential equation is the following

$$S_t = S_0 exp((r - \sigma^2/2)t + \sigma W_t) \tag{4}$$

In finance the value S_0 represents the initial value of the stock, r the interest rate, σ the volatility and W_t is a standard one dimensional Wiener process. We now consider a path-dependent option $f(S)$, where $S = (S_{T/m}, S_{2T/m}, ..., S_T)$ is the vector of a discrete number of stock prices for $t \in [0, T]$. The option price is given by

$$I = \mathrm{E}[f(S)] \tag{5}$$

The aim of this project is to estimate the volatility σ such that $I(\sigma)$ is equal to a certain given value I_{market}, i.e., to determine $\sigma*$ such that $I_{market} - \mathrm{E}[f_{\sigma*}(S)] = 0$. To do so it is possible to apply the algorithm mentioned above where α will be equal to I_{market} and the function $J(\theta)$ will be $I(\sigma)$ in our problem.

3 Monte Carlo applied to Robbins Monro

Monte Carlo methods are subset of computational algorithms, that are very powerful for modeling complex situations where random variables are involved. Since in each iteration of the *Robbins Monro* algorithm it is necessary to compute the value of $I(\sigma)$ which is defined as the expected value of $f(S)$, Monte Carlo techniques are good alternatives to approximate its value.

3.1 Crude Monte Carlo

The first algorithm that we will apply in the project is the *crude Monte Carlo* method, which simply consists in estimating $I(\sigma)$ with the *sample mean estimator*:

$$\hat{I}(\sigma) = \frac{1}{N}\sum_{n=1}^{N} f(X_i) \tag{6}$$

where $X_1, X_2, ..., X_N$ are independent and identical distributed random variables generated from the stock price S in our situation. This estimator has many important properties:

- it is *unbiased*
- its variance is equal to $\frac{\sigma^2}{N}$, where σ is the standard deviation of the variables X_i
- it converges almost surely to the the real expected value
- by *Central Limit Theorem* it converges in distribution to a normal distribution and thanks to this property it is possible to find an asymptotic confidence interval for the estimated expected value

3.2 Importance sampling

The issue with a *crude Monte Carlo* method is that it can be computationally expensive. There are situations, indeed, where it is necessary to generate too many variables to obtain a precise approximation of the objective function. This could happens in the case of a function like the the European put option defined as $f(S_T) = e^{-rT}(K - S_T)_+$, where $(K - S_T)_+$ is the $max\{0, K - S_T\}$. In such a situation, indeed, it is possible that the probability for S_T of being bigger than K is near 1, which means that the function $f(S)$ will be equal to 0 in most cases. As a result it will be necessary to sample too many variables to have enough relevant values of f and a precise approximation of $\mathrm{E}[f(S)]$. An option to solve such a problem is to use *Importance Sampling.* Its idea is that if we want to estimate the $\mathrm{E}_h[f(S)]$, where $h(S)$ is the density function of S, we can sample variables from a density function $g(S)$ and compute the $\mathrm{E}_g[\frac{f(S)h(S)}{g(S)}]$. The expected values are equal in fact:

$$\mathrm{E}_h[f(S)] = \int_{R^d} f(S)h(S)ds = \int_{R^d} \frac{f(S)h(S)}{g(S)} g(S)ds = \mathrm{E}_g\left[\frac{f(S)h(S)}{g(S)}\right] \tag{7}$$

Consequently it is possible to sample from a distribution where f(S) is different from 0 with high probability and obtain a good approximation of the objective function without a very heavy computational cost. The unique condition for this method is that $g(S)$ should be 0 only if $f(S)h(S)$ is 0. The other thing to observe is that since the variance of this model is equal to $\mathrm{E}_g[\frac{f(S)^2h(S)}{g(S)}]$ it is important to choose the distribution $g(S)$ in such a way that the variance of the model is minor than the variance obtained from a *crude Monte Carlo* model.

4 European put option

Let's now consider the problem of finding the volatility σ such that $I(\sigma)$ is equal to a certain value I_{market} for a European put option which is described by the following function:

$$f(S) = f(S_T) = e^{-rT}(K - S_T)_+ \tag{8}$$

In this situation only σ is unknown. The maturity time T is fixed to 0.2, the initial asset price S_0 is equal to 100, the strike price K is equal to 120 and the interest rate is equal to 5%. In such a situation there is an explicit form of the function $I(\sigma)$ given by Black-Scholes formula:

$$I(\sigma) = e^{-rT}K\Phi[\omega*(\sigma)] - S_0\Phi[\omega*(\sigma) - \sigma\sqrt{T}] \tag{9}$$

Where $\Phi[\cdot]$ is the cumulative density function of a standard Gaussian r.v. and

$$\omega^*(\sigma) = \frac{\log[\frac{K}{S_0}] - (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \tag{10}$$

Having the explicit formula of $I(\sigma)$ it is simple to see that it is increasing and that $\lim_{\sigma\to 0} I(\sigma) = (e^{-rT}K - S_0)_+$ and $\lim_{\sigma\to\infty} I(\sigma) = e^{-rT}K$ so for every I_{market} between these two values there will be a unique σ such that $I(\sigma) = I_{market}$.

4.1 Simple root-finding technique

For the parameter fixed above we want to find σ^* such that $I(\sigma^*) = 22$. Because there is an explicit formula for $I(\sigma)$ I first computed the zero of the function $I(\sigma) - I_{market}$ with an already implemented root-finding function in python and I have obtained that our objective volatility is $\sigma^* = 0.50837293$. From now we have considered that one as the *real value* of the volatility in such a situation.

4.2 Robbins Monro for European put option

Now it possible to find the zero for such a problem also with the *Robbins Monro* algorithm. We have fixed $\alpha_0 = \frac{2}{K+S_0}$ and we have taken two different possible $\{\alpha_n\}$ the first one equal to α_0/n and the second one equal to $\alpha_0/n^{0.8}$. It is important to notice that both the sequences respect the Equation 2. Consequently we are sure that the algorithm will converge to the solution. Inside the algorithm we used *crude Monte Carlo* to approximate the value $\hat{I}$ in every iteration. The implementation of the algorithm is the following:

Algorithm 1 : Robbins Monro algorithm with crude Monte Carlo

```
Initialize $\sigma_0$
Assign $N$ and $\rho$
for i=1,2,... do
    $\alpha_i = \frac{\alpha 0}{i^\rho}$
    Generate $S_T^{(1)}, S_T^{(2)}, \dots, S_T^{(N)}$ from the distribution with $\sigma_{i-1}$
    Approximate $\hat{I}(\sigma_{i-1}) = \frac{1}{N}\sum_{i=1}^{N} f(S_T^{(i)})$
    $\sigma_i = \sigma_{i-1} - \alpha_i(\hat{I}(\sigma_{i-1}) - I_{market})$
end for
```

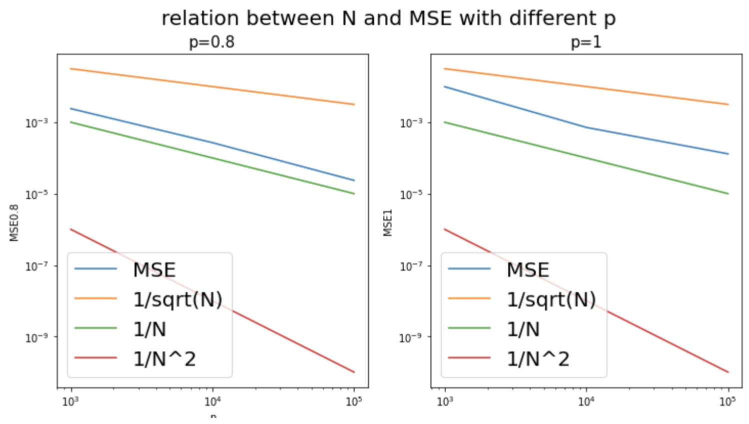


Figure 1: Relation between number of samples in Monte Carlo and MSE.

Once implemented the algorithm we tried to run it with different parameters to measure its performances. Firstly we fixed $N = 1$ and we tried to run the algorithm with both $\rho = 1$ and $\rho = 0.8$. We then tried to put N=10 and run again the algorithm with the two ρ. Figure 2 presents the *mean square error* of the models. As can be noticed from the output there is quite an important difference between the algorithms with different ρ, fact that suggests to choose its value wisely. The convergence rate for $\rho = 0.8$ is almost $\frac{1}{\sqrt{n}}$ while for $\rho = 1$ is near $\frac{1}{\sqrt[4]{n}}$. Additionally we can already notice, from Figure 2, that there is a big difference between the algorithms with $N = 1$ (the two at the top) and the algorithms with $N = 10$ (the two at the bottom). To understand better which is the relation between the number of samples used for *crude Monte Carlo* and the *MSE* I decided to fix the number of iteration of *Robbins Monro* algorithm to 1000 and to run the model with different values of N. The *MSE* is decreasing as $\frac{1}{N}$ as can be seen in Figure 1.

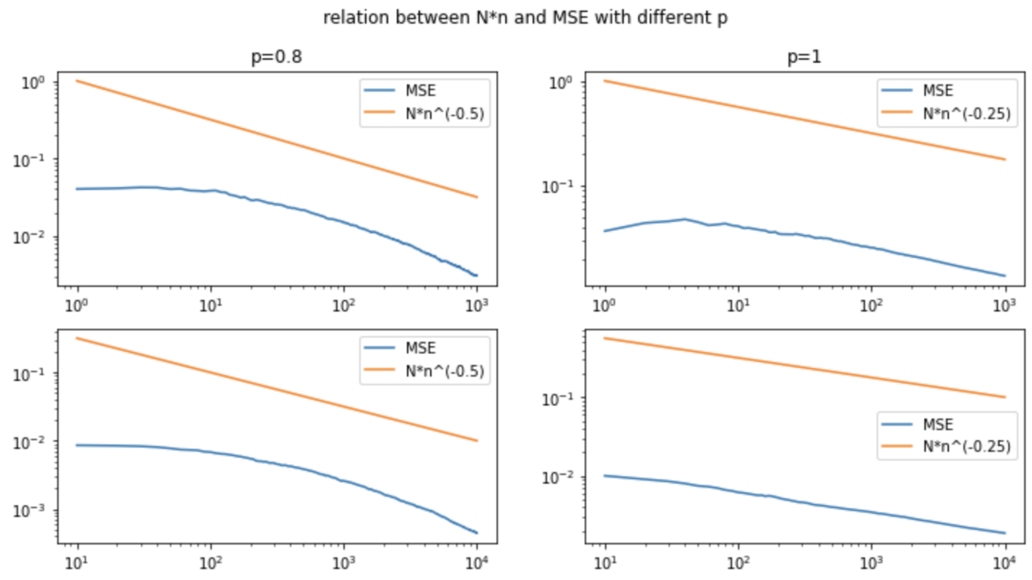


Figure 2: Convergence rate for the different ρ.

5 Asian put option

We now consider Asian put option for which:

$$f(S) = e^{-rT}(K - \bar{S}_T)_+ \tag{11}$$

where $\bar{S}_T = \frac{1}{m}\sum_{i=1}^{m} S_{iT/m}$. I kept the parameters as in the previous section. Then again $I(\sigma)$ is increasing. It is now necessary to find out which is the range of values where I_{market} is unique.

5.1 limits of I for Asian put option

First of all it is simple to observe that when σ goes to 0 each variable S_t converges to S_0e^{rt}, which means that they are not random variables. As a consequence since $I(0) = \mathrm{E}[f(S;\sigma = 0)]$ and S is not a random variable $I(0) = f(S;\sigma = 0)$, that is $I(0) = e^{-rT}(K - \frac{S_0}{m}\sum_{i=1}^{m} e^{riT/m})_+$. Let's now pass to the other limit. To prove it we first find the density function of the vector S. We will divide the proof into two steps:

- Find the distribution of the vector $W = (W_{\frac{T}{m}}, ..., W_T)$
- Consequently find the distribution of the vector S

What we will also use for this demonstration is the following theorem for the change of density function.

Theorem 1. *Let (X, Y):$\Omega \to R^2$ be a continuous random vector and $(U, V) = h((X, Y)) = (h_1(X, Y), h_2(X, Y))$, with $h_1, h_2 : R^2 \to R^1$. If h is C^1 in the support of (X, Y), the determinant of the Jacobian of h is different from 0 and $\exists g = h^{-1}$. Then (U, V) is a random continuous vector with the following probability density function:*

$$f_{(U,V)}(u, v) = f_{(X,Y)}(g_1(u, v), g_2(u, v))\,|\det(J_g)(u, v)| \tag{12}$$

First step. In this proof we will use t_i instead of $\frac{iT}{m}$. To find the density of the vector composed by the steps of the standard Wiener process we first consider the following vector:

$$Y = (W_{t_1}, W_{t_2} - W_{t_1}, \dots, W_{t_m} - W_{t_{m-1}}) \tag{13}$$

One of the properties of a Wiener process is that its increments are independent: $W_{t_i} - W_{t_{i-1}}$ is independent from $W_{t_{i-1}} - W_{t_{i-2}}$ for every t. Additionally each $W_{t_i} - W_{t_{i-1}} \sim \mathcal{N}(0, t_i - t_{i-1})$. So because they are independent the density of the vector is simply the product of the single densities:

$$P(y_1, \dots, y_m) = \prod_{i=1}^{m} \frac{1}{\sqrt{t_i - t_{i-1}}} \phi\left(\frac{y_i}{\sqrt{t_i - t_{i-1}}}\right) \tag{14}$$

Φ is the *pdf* of a standard gaussian random variable. Let's now consider the vector $X = (W_{t_1}, \dots, W_{t_m})$, we know that $Y = A(X)$, where A is the function $(x_1, x_2 - x_1, \dots, x_m - x_{m-1})$, a linear transformation. Its *Jacobian* is the following matrix:

$$J = \begin{bmatrix} 1 & & & \\ -1 & \ddots & & \\ & \ddots & & \\ & & -1 & 1 \end{bmatrix}$$

Also $X = A^{-1}(Y)$ is a linear transformation of Y so X is a *gaussian* vector. The determinant of the matrix J is equal to 1, as a consequence also the determinant of J^{-1} is equal to 1. Now the function g of the theorem exists and is $A(X)$ and obviously all the others hypothesis are satisfied since A^{-1} is linear. By Theorem 1 we obtain that:

$$P_X(x_1, \dots, x_m) = P_Y(A(X)) \left|\det(J^{-1})(x_1, \dots, x_m)\right| = P_Y(A(X)) = \prod_{i=1}^{m} \frac{1}{\sqrt{t_i - t_{i-1}}} \phi\left(\frac{x_i - x_{i-1}}{\sqrt{t_i - t_{i-1}}}\right) \tag{15}$$

□

Second step. We now consider our random vector $S = (S_{t_1}, S_{t_2}, \dots, S_{t_m})$, we can note that S in function of X, their relation is described in Equation 4. It is obviously C^1, and its inverse exists. $W_{t_i} = g_i(S) = \frac{\log(S_{t_i}/S_0) - (r - \sigma^2/2)t_i}{\sigma}$ for every i. We will call $g(S) = (g_1(S), \dots, g_m(S))$ the vector of the single functions. So the *Jacobian* in this case is simply $Diag(\frac{1}{\sigma S_{t_1}}, \frac{1}{\sigma S_{t_2}}, \dots, \frac{1}{\sigma S_{t_m}})$, because every g_i depends only on the term S_i. Its determinant, which is the product of the terms in the diagonal, is not 0 almost surely. We can apply again Theorem 1:

$$P_S(s_1, \dots, s_m) = P_X(g_1(S), \dots, g_m(S)) \left|\det(J_g)(s_1, \dots, s_m)\right| \tag{16}$$

Now the determinant is only the product of the terms in the *Jacobian* and the numerator inside the parenthesis of each Φ is the following:

$$W_{t_i} - W_{t_{i-1}} = \log(\frac{S_{t_i}}{S_0}) - (r - \sigma^2/2)t_i - \left(\log(\frac{S_{t_{i-1}}}{S_0}) - (r - \sigma^2/2)t_{i-1}\right) = \log(\frac{S_{t_i}}{S_{t_{i-1}}}) - (r - \sigma^2/2)(t_i - t_{i-1}) \tag{17}$$

So finally we find the density of the vector S:

$$P_S(s_1, \dots, s_m) = \prod_{i=1}^{m} \frac{1}{s_i \sigma \sqrt{t_i - t_{i-1}}} \phi\left(\frac{\log(\frac{s_i}{s_{i-1}}) - (r - \sigma^2/2)(t_i - t_{i-1})}{\sigma\sqrt{t_i - t_{i-1}}}\right) \tag{18}$$

□

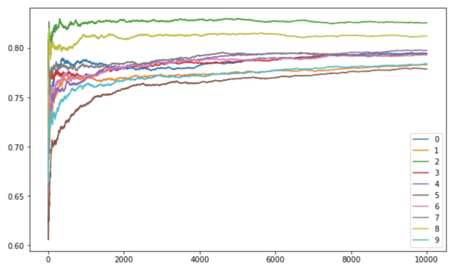
Consequently once obtained the Equation 18 it is clear that for $\sigma \to \infty$ the density of the vector S goes to 0 in all the points where $S \neq (0, \dots, 0)$ and that all the distribution is concentrate in that point. As a consequence $\lim_{\sigma \to \infty} I(\sigma) = e^{-rT}K$, because $\lim_{\sigma \to \infty} \bar{S} = 0$.

5.2 Apply RM to Asian put option problem

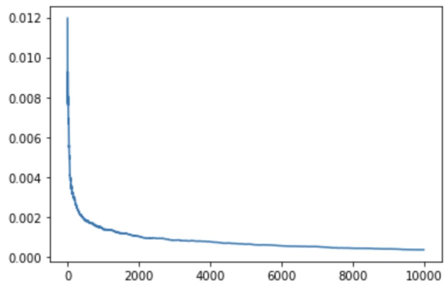
I have now chosen the previous value of I_{market} and also we have kept the same values for all the parameters, with the except of ρ which from now will be fixed to 0.8. We have first implemented a quite precise method to find the value of the "real" σ, since we don't have an explicit expression of I.

Real solution. The idea is to first run RM algorithm few times to find an approximate interval where the real sigma is. We then approximate the value of $I(\sigma)$ for many values inside that interval with *Monte Carlo* algorithm. I then define the points $I(\sigma) - I_{market}$ and finally I interpolate these points with cubic splines. Once obtained this equation it is possible to find σ^* with a classic root-finding algorithm. To check whether the solution was precise I then applied *Monte Carlo* with 10000000 samples to see whether $I(\sigma^*) - I_{market}$ was near 0 and the result was quite precise. Better for sure than the simple RM algorithm with few iterations. □

I then applied *Robbins Monro* to the problem with 10000 iterations. I ran 10 algorithms. The results are showed below.



(a) development of the different algorithms


(b) MSE of the model

As can be observed, when is near the real solution the algorithm starts to improve really slowly and in some cases it stops improving because it has reached the stability. It is therefore necessary to find out how to detect when it reaches the stability to avoid useless iterations. What could theoretically be done to stop it would be to put a tolerance and arrest the algorithm when $\hat{I} - I_{market} < tolerance$. Theoretically it would be the best solution, because it would means that $I(\sigma) - I_{market}$ is near 0, which is exactly what the algorithm aim to obtain. Unfortunately, this kind of stopping criterion is not useful in our algorithm, because we are approximating $\hat{I}$ with *Monte Carlo* and to have a quite precise approximation of it it should be necessary to use at least 100000 samples, but this would made the algorithm too slow, and also there would be no certainty of having reached the stability, because crude *Monte Carlo* could give a wrong approximation even with many samples.

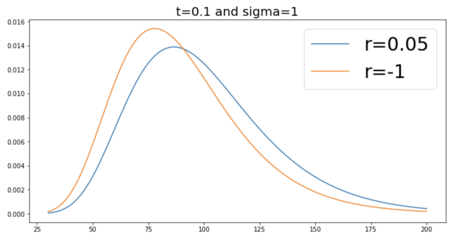
iterations	*stopping sigma*	*iterations*	*stopping sigma*
15000	0.7237227230370308	15000	0.7725099057126985
7089	0.8230631857593877	15000	0.7811562558940448
5022	0.8087492213397554	15000	0.7923257483794335
15000	0.7388571192180236	15000	0.779206432361851
5378	0.797310877884851	6884	0.8009541188896037

Table 1: table of results to compare the two algorithms.

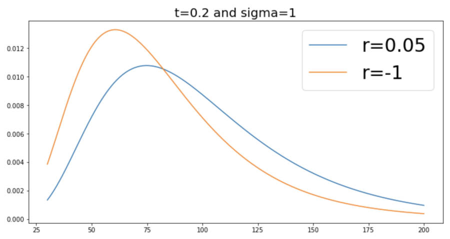
What I would propose is to arrest the algorithm when it starts to become too slow, because it would means that it is near the real solution. To detect this phenomena I suggest to fix a tolerance and stop the algorithm when $\sigma_i - \sigma_{i-num} < tolerance$. For instance I fixed *num*= 3000 and $tolerance = 10^{-6}$ and 15000 as maximum number of iterations. Table 1 present the outputs for my experiment: when the algorithm stops before the 15000 iterations the algorithm has always an error smaller than 0.02 (the real σ is 0.809). Obviously to have more precise results, it would be sufficient to reduce the *tolerance* or to increase the variable *num* or both of them. Obviously the conditions don't have to be too strict because the algorithm could stop only after too much time.

5.3 Importance sampling with fixed r-tilde

We have now fixed $K = 150$ and $I_{market} = 50$, in this situation crude *Monte Carlo* works better than before, because the probability for $\bar{S}$ to be bigger than 150 is lower. However it can be improved by an *Importance sampling* technique. The idea is to observe that $\mathrm{E}[S_t] = S_0 e^{rt}$, so the expected value depends on r. As a consequence to reduce the probability of each S_t of being bigger than K we are going to reduce its mean. It is sufficient to take a value $\tilde{r} < r$ and apply *Importance sampling* with that value. Doing so the algorithm will have more probability of taking values of $\bar{S}$ for whom $f(\bar{S}) \neq 0$. As mentioned in subsection 3.2, it is also important to choose a value of $\tilde{r}$ not too small because in that case the variance of the *Importance sampling* could be bigger than the crude *Monte Carlo* one, and there wouldn't be any improvement in the application of the new model. I chose $\tilde{r} = -1$ since from different plots of the new distribution against the old one it seems to fit perfectly the request for a good performance of *Importance sampling*. Figure 4b and Figure 4a present two examples of plots of the different log-normal distributions.



(a) two log-normal densities with t=0.1 and sigma=1



(b) two log-normal densities with t=0.2 and sigma=1

We then tested the performances in the approximation of $\hat{I}$ between the two models with different values of σ near the value $\sigma = 1$, because I know that in the following point the real σ will be near 1. To find the real solution I applied again the algorithm of subsection 5.2. Below I present the *Importance sampling* algorithm implemented to approximate $\hat{I}$.

Algorithm 2 : Importance sampling

```
Generate N iid replicas $S^{(1)}, \dots, S^{(N)} \sim p(\cdot; \tilde{r}, \sigma)$, p is the joint density function of S.
Compute for each $S^{(i)}$, $\bar{S}^i = \frac{1}{m} \sum_{i=1}^{m} S_{\frac{iT}{m}}$
compute $\hat{I} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\bar{S}^i) p(S^{(i)}; r, \sigma)}{p(S^{(i)}; \tilde{r}, \sigma)}$
```

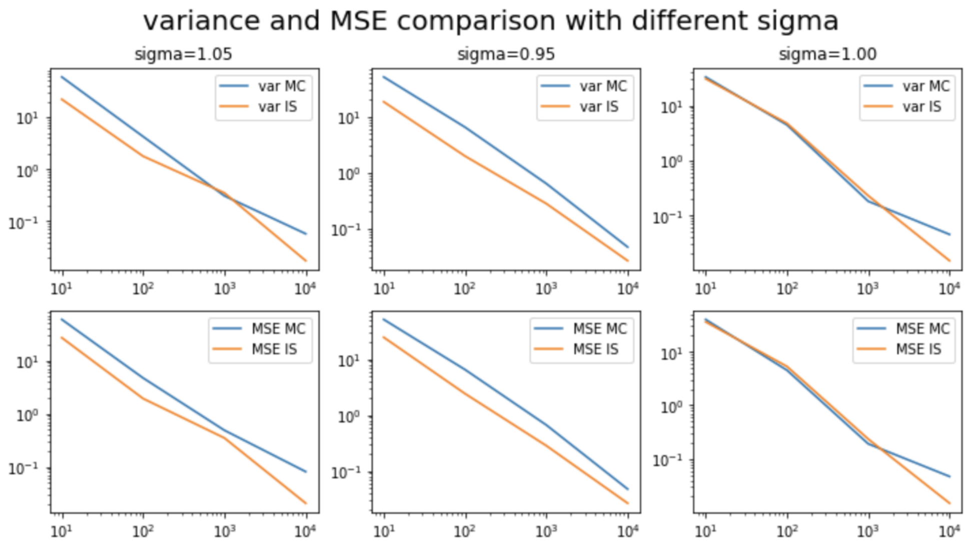


Figure 5: variance and MSE difference between the two models.

Figure 5 presents the performances of the two models. To compute the MSE, I applied crude *Monte Carlo* with 10000000 samples to find the value of $I(\sigma)$ to be considered as real value of I for a fixed σ. From the Figure 5 it appears clear that the *Importance Sampling* algorithm has better performances than the other model. It is also clear that the improvement is not really high, because as mentioned before we are trying to improve a model, which has not bad performances. What it is also interesting to notice is that the performances with different values of σ are different and that the chosen value of $\tilde{r}$ could not be the optimal value for every σ. This fact is suggesting that fixing a value of $\tilde{r}$ could not be the best option for the *Robbins Monro* algorithm, because inside every iteration the value of σ changes. I then implemented the RM algorithm with the *Importance Sampling* by simply replacing crude *Monte Carlo* with *Importance Sampling* to approximate $\hat{I}$. I then ran 100 times Robbins Monro both with *Importance Sampling* and crude *Monte Carlo* to find σ with $I_{market} = 50$. I've fixed the maximum number of iterations to 1000 and the number of samples to use for $\hat{I}$ approximation to 10. The following are the results of the two algorithms:

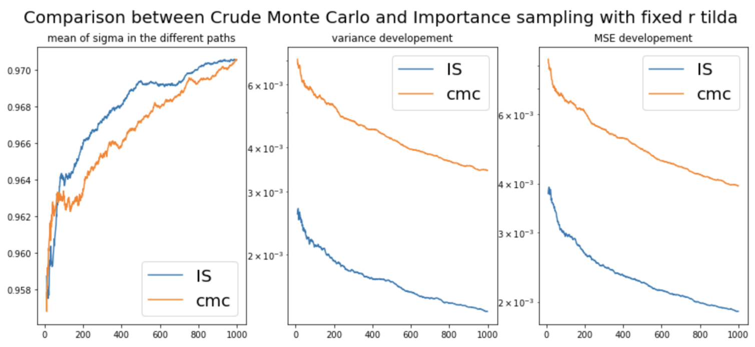


Figure 6: Comparison between crude Monte Carlo and Importance sampling. Mean of the sigmas on the left, variance in the middle and MSE on the right.

As can be notice from Figure 6 the new algorithm is converging faster to the real solution. The variance and the MSE of *Importance Sampling* are smaller than the one of crude *Monte Carlo*. The other strange thing is that although variance and MSE of *Importance sampling* are always smaller, they are decreasing almost with the same rate in the two algorithms. There are few possible explanations for such a phenomena:

- The first one is that $\tilde{r}$ could be optimal for some values of σ and not for others. More specifically it is possible that for some values of σ the variance of *Importance Sampling* with the fixed $\tilde{r}$ is bigger than the variance of crude *Monte Carlo*.
- The second one is that the probability of extracting significant values has not increased that much with *Importance Sampling*, because it was already high with crude *Monte Carlo*. So, because we are using only 10 samples to approximate $\hat{I}$ in each iteration it is possible that in some iterations the performance of the models is similar.

5.4 Importance sampling with the best possible r-tilde

The last thing that I tried to improve the algorithm was to find a function that gives the best possible $\tilde{r}$ for each value of σ. To do so I computed for some values of σ the best possible $\tilde{r}$ that is found by minimizing the variance of the importance sampling estimator. To approximate it I used the following formula:

$$\tilde{r} = \arg\max_{\nu} \frac{1}{N} \sum_{i=1}^{N} \frac{f^2(S^i; r, \sigma) p(S^i; r, \sigma)}{p(S^i, \nu, \sigma)} \tag{19}$$

Where S^i are N random variables, iid, extracted from a the distribution $p(\cdot; r, \sigma)$. Equation 19 is using crude *Monte Carlo* to approximate the variance. It was necessary to use a seed to avoid possible mistakes in the minimization. Because otherwise the value of S and consequently of $\bar{S}$ would have been different every time and the algorithm could have found the wrong optimal value of $\tilde{r}$ for a given σ. Once obtained $\tilde{r}$ for some values of $\sigma \in [0, 5]$, I applied splines cubic interpolation to obtain the function in the entire interval. I expect this approach to work better than the previous one, since in *Robbins Monro* algorithm we don't have a fixed value of σ.

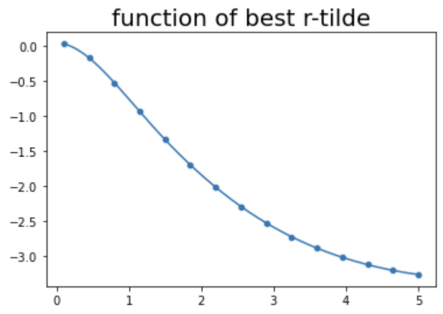


Figure 7: Interpolated function

What we can observe is that as expected the best values of $\tilde{r}$ are smaller that $r = 5\%$. Additionally it appears that the optimal value of $\tilde{r}$ is negative in this situation for almost all the values of σ. I actually applied the minimization for only few points of σ, this could bring to a higher percentage of error in the interpolation and consequently to a worse performance when running *Robbins Monro*. What I did was to run the algorithm with this $\tilde{r}$ instead of the constant one. So in Robbins Monro the implementation will be the same as before but before applying *Importance sampling* it is necessary to compute $\tilde{r} = interoplatedfunction(\sigma)$. Then the implementation is exactly the same as before. I first have run the two algorithms with *Importance sampling* 100 times, with 1000 iterations and 10 samples to approximate $\hat{I}$.

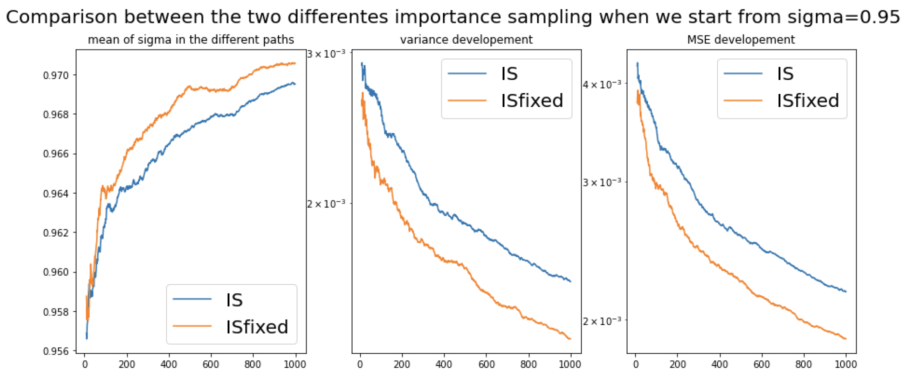


Figure 8: First comparison between model with r tilde fixed and the other importance sampling.

From this first result the new algorithm seems to have a worse performance than the one with $\tilde{r}$ fixed. But actually in such a situation it could happening that the one with fixed $\tilde{r}$ is faster due to the fact that the real solution we have computed is almost equal to 0.993 and the starting point for the algorithms was 0.95. As a result in the different iterations of our algorithm we expect the value of σ to be always in this interval. As can be noticed from Figure 7 the best value of $\tilde{r}$ for this interval is near -1. So the possible explanation for this performance is that the worse performance of the new algorithm could be explained by the error committed by the interpolation of the points.

Despite the difference in performance showed in Figure 8 it is not true in general that the model with fixed $\tilde{r}$ performs better, actually if we change the starting point this model stops to perform well as shown in Figure 9. In this case I ran ten times the two algorithms fixing as starting point $\sigma = 0.7$. The number of samples to approximate $\hat{I}$ was again $N = 10$. As can be seen in this situation the superiority of the new model is clear. This performance is given by the fact that $\tilde{r} = -1$ is a good value for *Importance sampling* only around $\sigma = 1$, once we move *Robbins Monro* too far from that value of σ the performance of the model starts decreasing. For this reason the last model implemented is the best one. Fixing a value of $\tilde{r}$ could give a good performance if and only if we fix as starting point a σ where that value of $\tilde{r}$ is optimal and the solution σ^* is in an interval where $\tilde{r}$ is still a good value for *Importance sampling* as in the cases of Figure 8 and Figure 6.

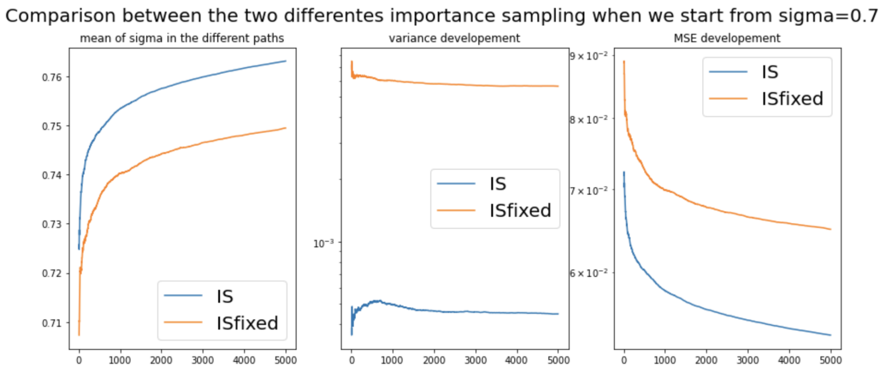


Figure 9: interpolated function.

In the end what I did was to have a comparison between the three models. I ran *Robbins Monro* with crude *Monte Carlo*, starting from $\sigma = 0.7$ and with the same condition as before. The results shown in Figure 10 are giving the proof of what I was saying before. As we can see, in fact, leaving from $\sigma = 0.7$ the performances of *Importance sampling* with fixed $\tilde{r}$ are even worse than the one of crude *Monte Carlo*. This fact indicates that the variance of *Importance sampling* with $\sigma = 0.7$ and $\tilde{r} = -1$ could even be bigger than the one of crude *Monte Carlo*, fact that could happening when using *Importance sampling* as already mentioned in subsection 3.2.

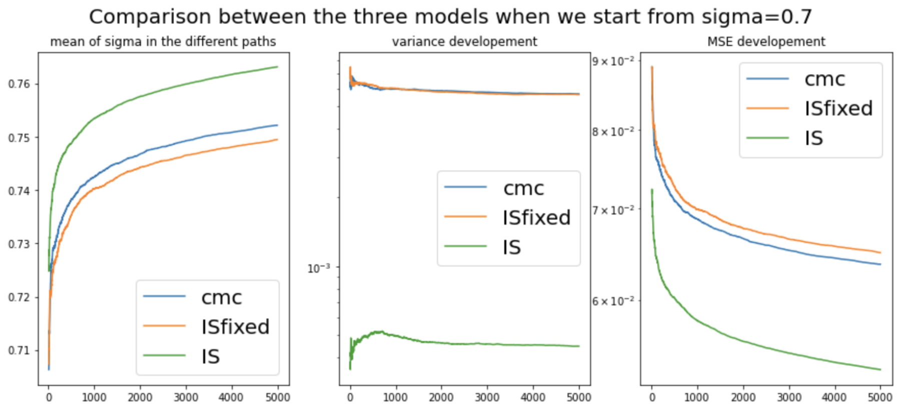
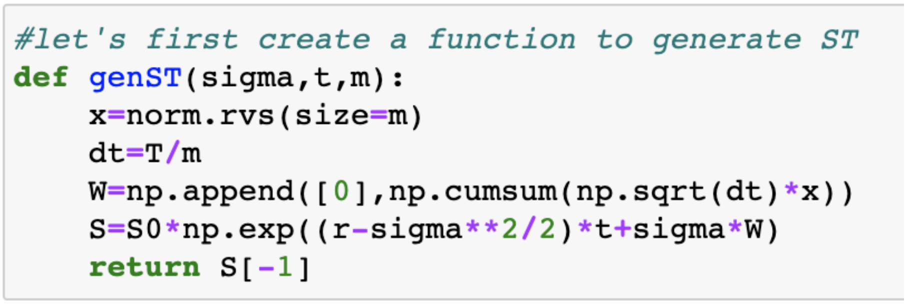


Figure 10: interpolated function.

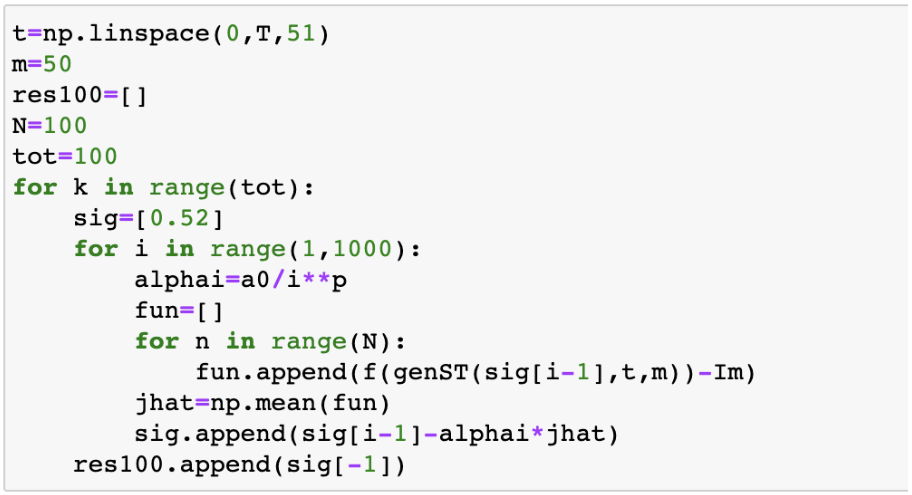
6 Code

6.1 European put option



```python
#let's first create a function to generate ST
def genST(sigma,t,m):
    x=norm.rvs(size=m)
    dt=T/m
    W=np.append([0],np.cumsum(np.sqrt(dt)*x))
    S=S0*np.exp((r-sigma**2/2)*t+sigma*W)
    return S[-1]
```

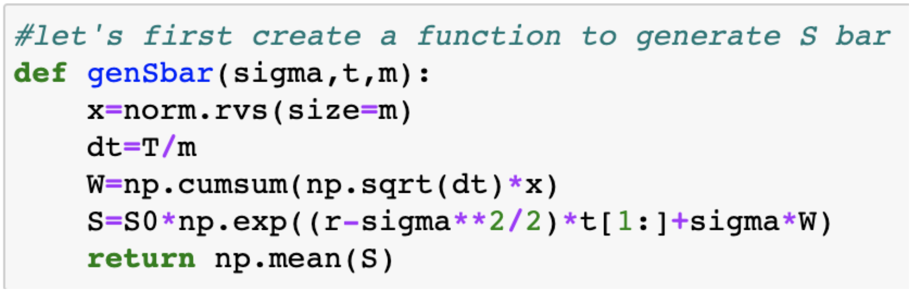
Figure 11: Function to generate European stock price.



```python
t=np.linspace(0,T,51)
m=50
res100=[]
N=100
tot=100
for k in range(tot):
    sig=[0.52]
    for i in range(1,1000):
        alphai=a0/i**p
        fun=[]
        for n in range(N):
            fun.append(f(genST(sig[i-1],t,m))-Im)
        jhat=np.mean(fun)
        sig.append(sig[i-1]-alphai*jhat)
    res100.append(sig[-1])
```

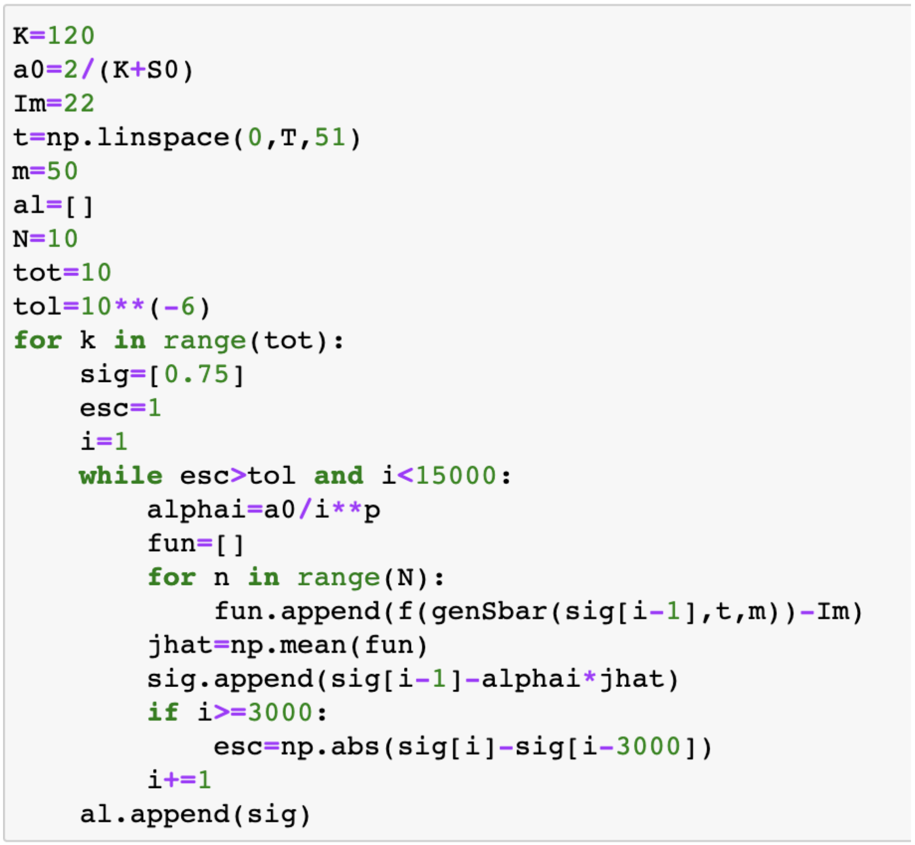
Figure 12: Robbins Monro with crude Monte Carlo.

6.2 Asian put option first part



```python
#let's first create a function to generate S bar
def genSbar(sigma,t,m):
    x=norm.rvs(size=m)
    dt=T/m
    W=np.cumsum(np.sqrt(dt)*x)
    S=S0*np.exp((r-sigma**2/2)*t[1:]+sigma*W)
    return np.mean(S)
```

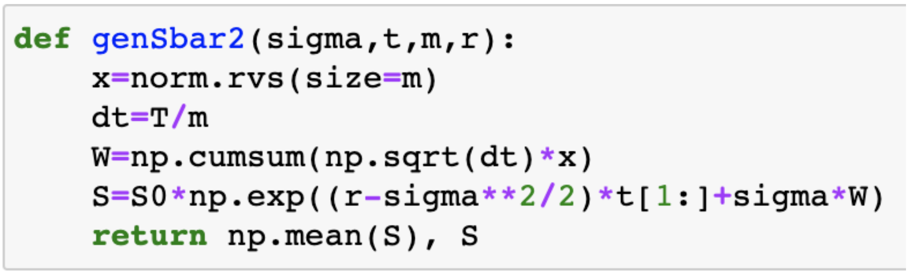
Figure 13: generate stock price for Asian put option.



```
K=120
a0=2/(K+S0)
Im=22
t=np.linspace(0,T,51)
m=50
al=[]
N=10
tot=10
tol=10**(-6)
for k in range(tot):
    sig=[0.75]
    esc=1
    i=1
    while esc>tol and i<15000:
        alphai=a0/i**p
        fun=[]
        for n in range(N):
            fun.append(f(genSbar(sig[i-1],t,m))-Im)
        jhat=np.mean(fun)
        sig.append(sig[i-1]-alphai*jhat)
        if i>=3000:
            esc=np.abs(sig[i]-sig[i-3000])
        i+=1
    al.append(sig)
```

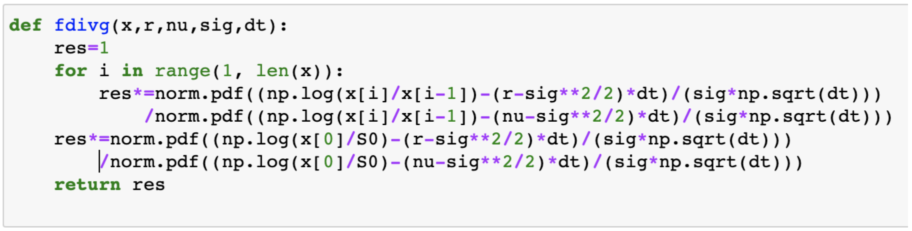
Figure 14: Robbins Monro with stopping condition.

6.3 Asian put option with IS



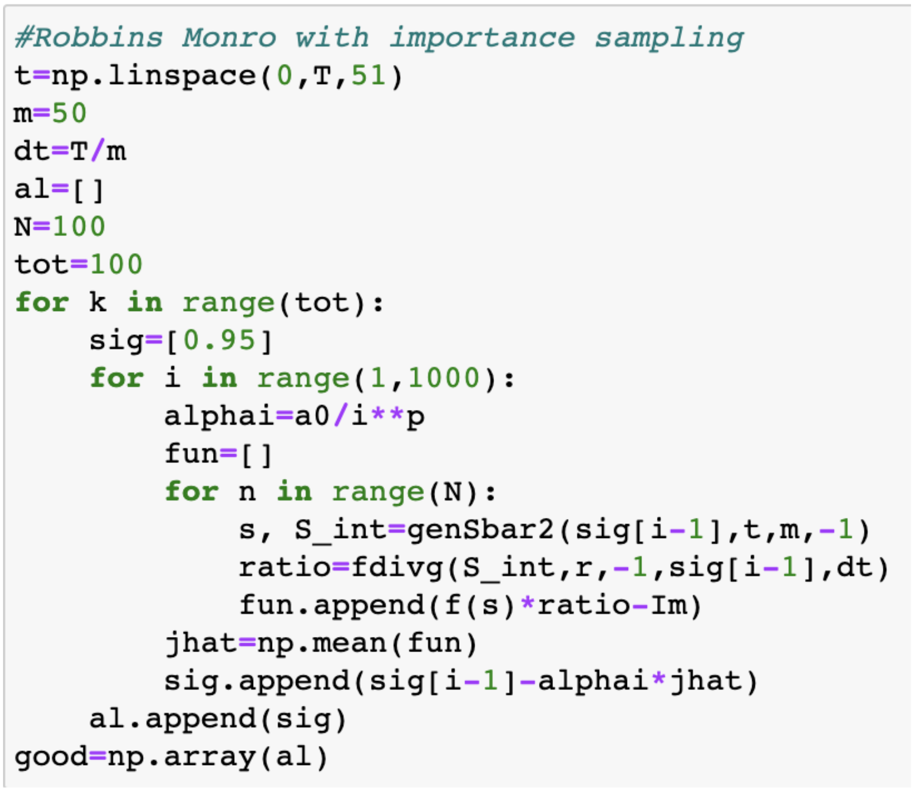
```
def genSbar2(sigma,t,m,r):
    x=norm.rvs(size=m)
    dt=T/m
    W=np.cumsum(np.sqrt(dt)*x)
    S=S0*np.exp((r-sigma**2/2)*t[1:]+sigma*W)
    return np.mean(S), S
```

Figure 15: Generate stock price for IS.



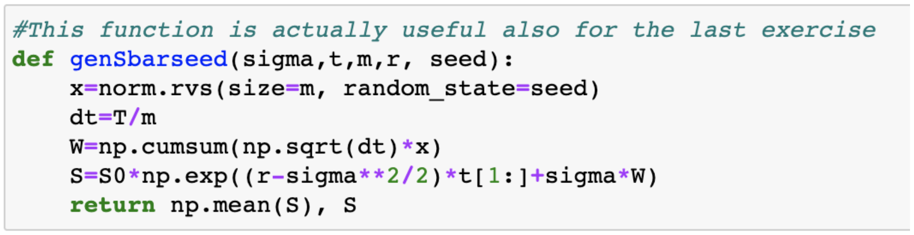
```
def fdivg(x,r,nu,sig,dt):
    res=1
    for i in range(1, len(x)):
        res*=norm.pdf((np.log(x[i]/x[i-1])-(r-sig**2/2)*dt)/(sig*np.sqrt(dt)))
            /norm.pdf((np.log(x[i]/x[i-1])-(nu-sig**2/2)*dt)/(sig*np.sqrt(dt)))
    res*=norm.pdf((np.log(x[0]/S0)-(r-sig**2/2)*dt)/(sig*np.sqrt(dt)))
        /norm.pdf((np.log(x[0]/S0)-(nu-sig**2/2)*dt)/(sig*np.sqrt(dt)))
    return res
```

Figure 16: Ratio between density functions for IS.



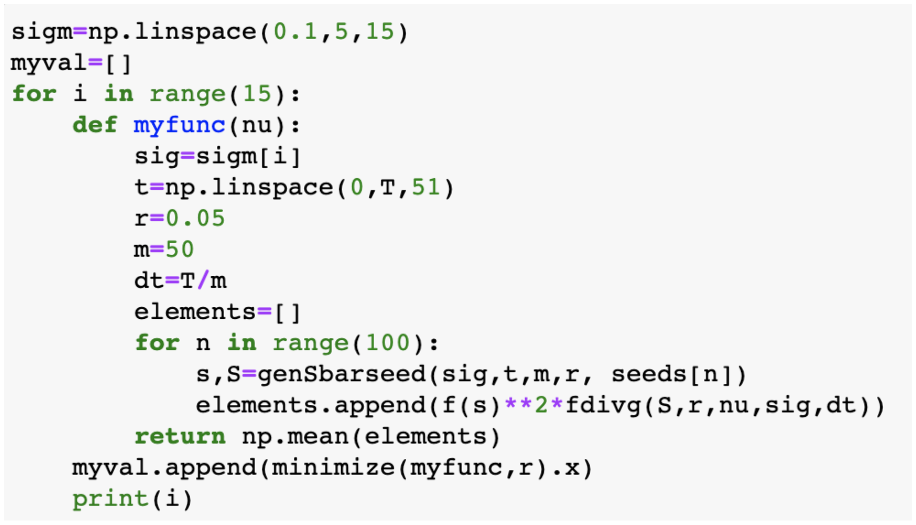
```
#Robbins Monro with importance sampling
t=np.linspace(0,T,51)
m=50
dt=T/m
al=[]
N=100
tot=100
for k in range(tot):
    sig=[0.95]
    for i in range(1,1000):
        alphai=a0/i**p
        fun=[]
        for n in range(N):
            s, S_int=genSbar2(sig[i-1],t,m,-1)
            ratio=fdivg(S_int,r,-1,sig[i-1],dt)
            fun.append(f(s)*ratio-Im)
        jhat=np.mean(fun)
        sig.append(sig[i-1]-alphai*jhat)
    al.append(sig)
good=np.array(al)
```

Figure 17: Robbins Monro with the fixed r tilde.



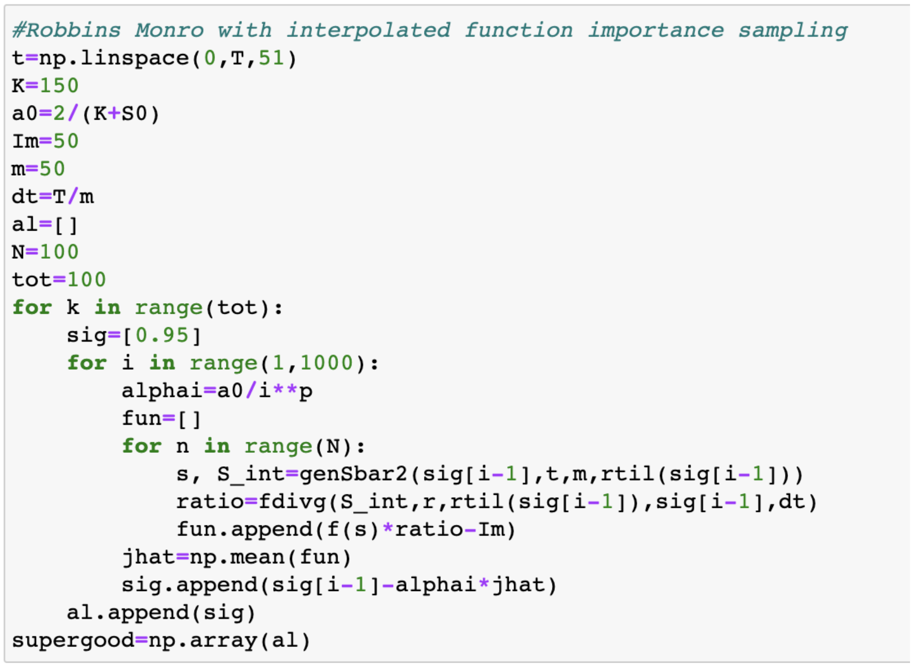
```
#This function is actually useful also for the last exercise
def genSbarseed(sigma,t,m,r, seed):
    x=norm.rvs(size=m, random_state=seed)
    dt=T/m
    W=np.cumsum(np.sqrt(dt)*x)
    S=S0*np.exp((r-sigma**2/2)*t[1:]+sigma*W)
    return np.mean(S), S
```

Figure 18: St generation for the minimization of the function in point 7.



```python
sigm=np.linspace(0.1,5,15)
myval=[]
for i in range(15):
    def myfunc(nu):
        sig=sigm[i]
        t=np.linspace(0,T,51)
        r=0.05
        m=50
        dt=T/m
        elements=[]
        for n in range(100):
            s,S=genSbarseed(sig,t,m,r, seeds[n])
            elements.append(f(s)**2*fdivg(S,r,nu,sig,dt))
        return np.mean(elements)
    myval.append(minimize(myfunc,r).x)
    print(i)
```

Figure 19: Minimization of the function to find best r tilde



```python
#Robbins Monro with interpolated function importance sampling
t=np.linspace(0,T,51)
K=150
a0=2/(K+S0)
Im=50
m=50
dt=T/m
al=[]
N=100
tot=100
for k in range(tot):
    sig=[0.95]
    for i in range(1,1000):
        alphai=a0/i**p
        fun=[]
        for n in range(N):
            s, S_int=genSbar2(sig[i-1],t,m,rtil(sig[i-1]))
            ratio=fdivg(S_int,r,rtil(sig[i-1]),sig[i-1],dt)
            fun.append(f(s)*ratio-Im)
        jhat=np.mean(fun)
        sig.append(sig[i-1]-alphai*jhat)
    al.append(sig)
supergood=np.array(al)
```

Figure 20: Robbins Monro for the last point.