Stochastic Approximation in Mathematical Finance*

Jonas Papazoglou-Hennig (EPFL)

Lausanne, January 6, 2022

1 Introduction and setting

In this report we record findings based on the project conducted on Stochastic Approximation in Mathematical Finance (Project 6)[1], as part of the examination requirements for the course *Stochastic Simulations* in the autumn semester 2021. The main goal is to implement and investigate the performance of a stochastic root-finding method, the Robbins-Monro (RM) algorithm [RM85], for the problem:

$$\text{find } \theta^* \ : \ \mathbb{E}[f_{\theta^*}(X)] = 0, \tag{1}$$

where X is a $\mathbb{R}^d$-valued random variable and $f_\theta : \mathbb{R}^d \to \mathbb{R}$ is a family of functions parameterized in $\theta \in \mathbb{R}^d$. Problem (1) appears naturally in mathematical finance, more specifically in the calculation of implied volatilities, which will be used as an application domain for this project.

2 RM-algorithm

We briefly review the main components of the RM-algorithm from the project description. Consider problem (1) and let $J(\theta) = \mathbb{E}[f_\theta(X)]$. The RM-algorithm is an iterative scheme which updates θ in each step as

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

where $\hat{J}(\theta)$ is an unbiased estimator of $J(\theta)$ and α_n is the *learning rate*. A first choice for $\hat{J}(\theta)$ could be a simple Monte-Carlo estimator, that is

$$\hat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N} f_\theta(X^{(i)}), \tag{2}$$

where $X^{(i)}$ are sampled iid from the distribution of X. However, one goal of the project is to improve this crude Monte-Carlo approach by using importance sampling as a variance reduction technique. We will not go into proving convergence of the algorithm (for this, see as proposed [AG07]), but mention that it is required that $\sum_n \alpha_n = \infty$ and $\sum_n \alpha_n^2 < \infty$. Per the project description we will choose $\alpha_0 > 0$ and $\alpha_n = \alpha_0/n^\rho$ for some $\rho \in (0.5, 1]$.

3 Black-Scholes model, options and implied volatility

In this section we recall some theory from financial mathematics, which allows us to price options in the Black-Scholes model. We do not go into detailed derivations, except for those explicitly asked for in the project description, as well as some helpful calculations with regard to later implementation of the RM-algorithm. For more details on the mathematical finance fundamentals, consult any standard textbook, e.g. [Shr04].

We consider a Black-Scholes market with a single stock, whose price-process over time $t \in [0, T]$ is modeled as an Ito-diffusion:

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

where r represents the interest rate at which money can be invested without any risk (i.e. a bank/savings account), $\sigma > 0$ is the volatility and W_t is a standard 1-dimensional Brownian motion. Additionally, one must prescribe a starting value S_0 for the asset at time $t = 0$. By applying Ito's formula, (3) can be explicitly solved as a geometric Brownian motion:

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

*Supervised by Prof. Fabio Nobile and Sundar Ganesh, lecturer and assistant of the course *Stochastic Simulations*.

[1]The project was accompanied with a description, which is mentioned here and there throughout the report. When there is a reference made to lecture notes, the course lecture notes of Prof. Nobile are meant [Nob22].

hence for each $t > 0$, S_t/S_0 has a Lognormal(μ, τ^2)-distribution with $\mu = (r - \sigma^2/2)t$ and $\tau^2 = \sigma^2 t$. A linear transformation of the lognormal density with the given parameters yields that the density of S_t is hence given by:

$$\psi(s) = \frac{1}{s\sigma\sqrt{t}}\phi\left(\frac{\log(s/S_0) - (r - \sigma^2/2)t}{\sigma\sqrt{t}}\right),\ s > 0, \tag{5}$$

where ϕ is the standard normal density.

An *option* is a financial contract between two parties (holder and seller), which gives the holder the right to some payoff from the seller based on the evolution of the asset in $[0, T]$. So the payoff is a function h of the price evolution (S_t). If the payoff only depends on the value of the asset at maturity, i.e. $h(S_T)$, we speak of *European options.* On the other hand, if the payoff also depends on other points of the trajectory of (S_t), we say it is an *Asian option.* Both cases will be considered in what proceeds, however in the Asian option regime we restrict ourselves to *discrete (finite) monitoring*, i.e. the option payoff depends only on the asset value at finitely many time points in $[0, T]$. More precisely, we will only look at options, whose payoff is of the form $h = h(S_{T/m}, S_{2T/m}, ..., S_T)$ for some $m \in \mathbb{N}$. It is therefore of interest to also know the joint density of the random vector $(S_{T/m}, S_{2T/m}, ..., S_T)$.

Lemma 3.1. *The joint density of* $(S_{T/m}, S_{2T/m}, ..., S_T)$ *in the Black-Scholes model is given by*

$$p(s) = p(s_1, ..., s_m) := \prod_{i=1}^{m} \frac{1}{s_i\sigma\sqrt{\Delta t}}\phi\left(\frac{\log(s_i/s_{i-1}) - (r - \sigma^2/2)\Delta t}{\sigma\sqrt{\Delta t}}\right),\ s \in \mathbb{R}^m_{>0}, \tag{6}$$

where $s_0 = S_0$, $\Delta t = T/m$ *and* ϕ *is the standard normal density.*

Proof. We prove the claim by induction. The case $m = 1$ coincides with (5), hence there is nothing more to show. Now assume the claim holds for $m-1$. Denote $t_j = jT/m$ and let $A_1, ..., A_m \in \mathcal{B}(\mathbb{R})$. Due to independence of the Brownian motion increments, we have that $S_{t_m}|[S_{t_{m-1}} = s_{t_{m-1}}] \sim s_{t_{m-1}} \exp((r-\sigma^2/2)(t_m - t_{m-1}) + \sigma W_{t_m - t_{m-1}})$. Therefore, by (5):

$$\mathbb{P}[S_{t_m} \in A_m | S_{t_{m-1}} = s_{t_{m-1}}] = \int_{A_m} \frac{1}{s_{t_m}\sigma\sqrt{t_m - t_{m-1}}}\phi\left(\frac{\log(s_{t_m}/s_{t_{m-1}}) - (r - \sigma^2/2)(t_m - t_{m-1})}{\sigma\sqrt{t_m - t_{m-1}}}\right) ds_{t_m} \tag{7}$$

$$= \int_{A_m} \frac{1}{s_{t_m}\sigma\sqrt{\Delta t}}\phi\left(\frac{\log(s_{t_m}/s_{t_{m-1}}) - (r - \sigma^2/2)\Delta t}{\sigma\sqrt{\Delta t}}\right) ds_{t_m}. \tag{8}$$

It is well-known that geometric Brownian motion is a Markov process, hence we can calculate:

$$\begin{aligned}
\mathbb{P}[S_{t_1} \in A_1, ..., S_{t_m} \in A_m] &= \mathbb{E}[\prod_{j=1}^{m} \mathbb{1}_{A_j}(S_{t_j})] = \mathbb{E}_{S_{t_1},...,S_{t_{m-1}}}[\mathbb{E}[\prod_{j=1}^{m} \mathbb{1}_{A_j}(S_{t_j}) | S_{t_1}, ..., S_{t_{m-1}}]] \\
&= \mathbb{E}_{S_{t_1},...,S_{t_{m-1}}}[\prod_{j=1}^{m-1} \mathbb{1}_{A_j}(S_{t_j})\mathbb{E}[\mathbb{1}_{A_m}(S_{t_m}) | S_{t_1}, ..., S_{t_{m-1}}]] \\
&= \mathbb{E}_{S_{t_1},...,S_{t_{m-1}}}[\prod_{j=1}^{m-1} \mathbb{1}_{A_j}(S_{t_j})\mathbb{E}[\mathbb{1}_{A_m}(S_{t_m}) | S_{t_{m-1}}]] \\
&= \mathbb{E}_{S_{t_1},...,S_{t_{m-1}}}[\mathbb{1}_{A_1\times...\times A_{m-1}}(S_{t_1}, ..., S_{t_{m-1}})\mathbb{P}[S_{t_m} \in A_m | S_{t_{m-1}}]],
\end{aligned}$$

where we used the tower property in the second equality and the Markov property in the third line. By (8) this can be further written as:

$$= \mathbb{E}_{S_{t_1},...,S_{t_{m-1}}}\left[\mathbb{1}_{A_1\times...\times A_{m-1}}(S_{t_1}, ..., S_{t_{m-1}}) \int_{A_m} \frac{1}{s_{t_m}\sigma\sqrt{\Delta t}}\phi\left(\frac{\log(s_{t_m}/S_{t_{m-1}}) - (r - \sigma^2/2)\Delta t}{\sigma\sqrt{\Delta t}}\right) ds_{t_m}\right],$$

and by induction hypothesis applied to the joint density of the first $m - 1$ of the S_{t_j}:

$$\begin{aligned}
&= \int_{A_1\times...\times A_{m-1}} p(s_{t_1}, ..., s_{t_{m-1}}) \int_{A_m} \frac{1}{s_{t_m}\sigma\sqrt{\Delta t}}\phi\left(\frac{\log(s_{t_m}/s_{t_{m-1}}) - (r - \sigma^2/2)\Delta t}{\sigma\sqrt{\Delta t}}\right) ds_{t_m} d(s_{t_1}, ..., s_{t_{m-1}}) \\
&= \int_{A_1\times...\times A_m} p(s_{t_1}, ..., s_{t_m}) d(s_{t_1}, ..., s_{t_m}),
\end{aligned}$$

where we indeed recover the desired structure of the formula for the joint density of all m random variables in the last equality, hence the induction is complete. □

As mentioned in the previous section, one of the goals of this project is to apply the RM-algorithm within this model and seek to improve performance using importance sampling (IS). In this context, we will select an IS-distribution that follows a geometric Brownian motion, but with a drift change. I.e. if $S_t = S_0 \exp((r - \sigma^2/2)t + \sigma W_t)$ is our asset price process, we may want to sample from $\tilde{S}_t = S_0 \exp((\tilde{r} - \sigma^2/2)t + \sigma W_t)$ instead, for some other $\tilde{r} \in \mathbb{R}$. During the course we saw that this approach can significantly reduce the variance of Monte-Carlo estimators, but to apply IS we must always correct the sample by the likelihood-ratio of the target density and the IS-density. In our setting, we obtain a convenient formulation:

Lemma 3.2. *Let p be the joint density of $(S_{T/m}, S_{2T/m}, ..., S_T)$ and $\tilde{p}$ the joint density of $(\tilde{S}_{T/m}, \tilde{S}_{2T/m}, ..., \tilde{S}_T)$, where in $\tilde{S}_t$ the drift is changed from r to $\tilde{r}$. Then, the likelihood-ratio of the densities is given as:*

$$\frac{p(s_1, ..., s_m)}{\tilde{p}(s_1, ..., s_m)} = \exp\left(-\frac{(\tilde{r} - r)}{2\sigma^2}(2\log(s_m/s_0) - m\Delta t(r + \tilde{r} - \sigma^2))\right)$$

Proof. Let $z_i = \log(s_i/s_{i-1}) - (r - \sigma^2/2)\Delta t$ and $\tilde{z}_i = \log(s_i/s_{i-1}) - (\tilde{r} - \sigma^2/2)\Delta t$. Plugging the formula for the joint densities in, we have

$$\frac{p(s_1, ..., s_m)}{\tilde{p}(s_1, ..., s_m)} = \prod_{i=1}^{m} \frac{\phi(z_i/(\sigma\sqrt{\Delta t}))}{\phi(\tilde{z}_i/(\sigma\sqrt{\Delta t}))} = \prod_{i=1}^{m} \frac{\exp(-z_i^2/(2\sigma^2\Delta t))}{\exp(-\tilde{z}_i^2/(2\sigma^2\Delta t))} = \exp\left(-\frac{1}{2\sigma^2\Delta t}\sum_{i=1}^{m}(z_i^2 - \tilde{z}_i^2)\right).$$

By the binomial theorem,

$$z_i^2 - \tilde{z}_i^2 = (z_i + \tilde{z}_i)(z_i - \tilde{z}_i) = ((\tilde{r} - r)\Delta t)(2\log(s_i/s_{i-1}) - (r + \tilde{r} - \sigma^2)\Delta t).$$

Plugging this in above yields,

$$\begin{aligned}\frac{p(s_1, ..., s_m)}{\tilde{p}(s_1, ..., s_m)} &= \exp\left(-\frac{(\tilde{r} - r)\Delta t}{2\sigma^2\Delta t}\sum_{i=1}^{m}(2\log(s_i/s_{i-1}) - (r + \tilde{r} - \sigma^2)\Delta t)\right) \\ &= \exp\left(-\frac{(\tilde{r} - r)}{2\sigma^2}(2\log(s_m/s_0) - m\Delta t(r + \tilde{r} - \sigma^2))\right),\end{aligned}$$

where we split the sum and used the telescoping structure of the log-ratios to collapse to the desired expression. □

At what price should one buy or sell an option? Financial mathematics answers this question using the *risk-neutral pricing principle*, which says that the fair price of an option is the discounted expected value of the payoff. If we write $S = (S_{T/m}, S_{2T/m}, ..., S_T)$ and the discounted payoff function as $f(S) = e^{-rT}h(S)$, then the option price is given as

$$I := \mathbb{E}[f(S)] = \int_{\mathbb{R}^m} f(s_1, ..., s_m)p(s_1, ..., s_m)\, ds. \tag{9}$$

Let us shed light on two concrete examples of options we study in this project:

Definition 3.3. *A European put option is an option with discounted payoff*

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

where $(x)_+$ is the positive part of x and K is the so-called strike.

Definition 3.4. *An m-step Asian discrete monitoring average (ADMA) put option is an option with discounted payoff*

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

where $(x)_+$ is the positive part of x, K is the strike and $\bar{S}_T = \frac{S_0}{m}\sum_{k=1}^{m} S_{kT/m}$ is the discrete monitoring average.

Notice that the latter option reduces to the former, when $m = 1$. In fact, in the case of the European put, there exists an explicit formulation of the price, known as the *Black-Scholes formula.*

Theorem 3.5. *The price of a European put option can be written as*

$$I = Ke^{-rT}\Phi(d) - S_0\Phi(d - \sigma\sqrt{T}),$$

where Φ is the CDF of the standard normal law and

$$d = \frac{\log(K/S_0) - (r - \sigma^2/2)T}{\sigma\sqrt{T}}.$$

As exposed above, we can see that, assuming a Black-Scholes market, option prices depend on the parameters (σ, r, S_0, T). While initial asset prices, maturities and interest rates are readily available in a real-world market, the volatility of a given asset is not revealed that immediately. In finance, it is therefore of great interest to learn something about the volatility parameter in view of observed market option prices I_m. If we consider the option price as a function of σ, i.e. $I = I(\sigma)$, this yields the problem

$$\text{find } \sigma^* \ : \ I(\sigma^*) - I_m = 0. \tag{10}$$

σ^* is called the *implied volatility* and the above can be recast into the setting of (1) by invoking (9):

$$\text{find } \sigma^* \ : \ \mathbb{E}[\hat{f}_{\sigma^*}(S)] = 0, \tag{11}$$

where $\hat{f}_\sigma(S) = f(S^{(\sigma)}) - I_m$ and $S^{(\sigma)}$ corresponds to the underlying asset price process with volatility σ.

We would like to further investigate the dependence of these option prices with respect to volatility in the Black-Scholes model. It turns out that in both cases of the European and ADMA put, the price $I(\sigma)$ is increasing[2] in σ. A natural question to ask, is how I evolves when σ is close to 0 or gets arbitrarily large.

Lemma 3.6. *For an m-step ADMA put option, we have the following limiting behavior for the price function $I(\sigma)$ with respect to the Black-Scholes volatility:*

$$\lim_{\sigma\to 0} I(\sigma) = e^{-rT}\left(K - \frac{S_0}{m}\sum_{k=1}^{m} e^{rkT/m}\right)_+, \ \lim_{\sigma\to\infty} I(\sigma) = Ke^{-rT}$$

Proof. Recall from (4) that for all $t > 0$, $S_t = S_0 \exp((r - \sigma^2/2)t + \sigma W_t)$, i.e. a geometric Brownian motion. It is well-known that:

$$\mathbb{E}[S_t] = S_0 e^{rt}, \ Var(S_t) = S_0^2 e^{2rt}(e^{\sigma^2 t} - 1).$$

By Chebyshev's inequality, for any $\varepsilon > 0$:

$$\mathbb{P}[|S_t - \mathbb{E}[S_t]| > \varepsilon] \leq \frac{Var(S_t)}{\varepsilon^2} = \frac{S_0^2 e^{2rt}(e^{\sigma^2 t} - 1)}{\varepsilon^2} \to 0, \ \sigma \to 0,$$

hence $S_t \to S_0 e^{rt}$ in probability as $\sigma \to 0$ for each $t > 0$. In particular, $S = (S_{T/m}, S_{2T/m}, ..., S_T) \to S_0(e^{rT/m}, e^{2rT/m}, ..., e^{rT}) = \tilde{S}$ in probability. Note that the ADMA put discounted payoff function f is continuous, therefore also $f(S) \to f(\tilde{S})$ in probability, and bounded by Ke^{-rT}. This allows us to invoke dominated convergence, to see

$$\lim_{\sigma\to 0} I(\sigma) = \lim_{\sigma\to 0} \mathbb{E}[f(S)] = \mathbb{E}[f(\tilde{S})],$$

where plugging in the expression for $\tilde{S}$ into f immediately yields the desired limit. For the other limit, note that for the σ-dependent portion of the exponent in (4), we have:

$$-\frac{\sigma^2 t}{2} + \sigma W_t \sim -\frac{\tilde{t}}{2} + W_{\tilde{t}},$$

where $\tilde{t} = \sigma^2 t$. Clearly, if $\sigma \to \infty$ then $\tilde{t} \to \infty$, but $W_{\tilde{t}}/\tilde{t} \to 0$ almost surely as $\tilde{t} \to \infty$, hence

$$-\frac{\tilde{t}}{2} + W_{\tilde{t}} = \tilde{t}\left(-\frac{1}{2} + \frac{W_{\tilde{t}}}{\tilde{t}}\right) \to -\infty,$$

almost surely. Thus $\exp(-\frac{\tilde{t}}{2} + W_{\tilde{t}}) \to 0$ almost surely, and therefore $\exp(-\frac{\sigma^2 t}{2} + \sigma W_t) \to 0$ in distribution. Convergence in distribution to a constant implies convergence in probability and therefore by continuity of all terms present in the geometric Brownian motion formula for S_t, we have for all $t > 0$ that $S_t = S_0 \exp((r - \sigma^2/2)t + \sigma W_t) = S_0 e^{rt} \exp(-\frac{\sigma^2 t}{2} + \sigma W_t) \to 0$ in probability. Now the proof can be completed as before, where this time $S = (S_{T/m}, S_{2T/m}, ..., S_T) \to (0, ..., 0)$ in probability, therefore $f(S) \to f(0, ..., 0)$ in probability. Finally, dominated convergence allows us to exchange limits and expectation to see that $\lim_{\sigma\to\infty} I(\sigma) = \mathbb{E}[f(0, ..., 0)] = f(0, ..., 0) = Ke^{-rT}$. □

We have thus shown that the price of an ADMA put must be contained in the interval

$$\left[e^{-rT}\left(K - \frac{S_0}{m}\sum_{k=1}^{m} e^{rkT/m}\right)_+, Ke^{-rT}\right], \tag{12}$$

and provided an observed market price I_m lies in this interval, the increasingness of $I(\sigma)$ implies that the problem in (10) is well-posed. This concludes our discussion of theory from mathematical finance in the context of this project.

[2]This was provided in the project description.

4 Numerical experiments

We now turn to exposing the results from numerical experiments from this project, in applying the RM-algorithm to the setting of calculating implied volatilities, as motivated in (11). In what follows, we address most of the tasks given in the project description[3]. We begin by briefly looking at the European put (i.e. points 1, 2), followed by an analysis for the ADMA put (i.e. points 5,6,7). Implementations were carried out in R and the code can be found in the Appendix, on GitHub and will be additionally submitted together with this report as a ZIP-archive. Note that a bouquet of auxiliary functions are defined in Appendix A for the efficient simulation of price processes in the Black-Scholes model, calculation of relevant option payoffs, the Black-Scholes formula and likelihood-ratios, which all later implementations rely on.

4.1 European put option

In this section, we consider a Black-Scholes market with interest rate $r = 0.05$, time horizon $T = 0.2$ and a European put option on an asset with initial value $S_0 = 100$ and strike $K = 120$. We further assume to observe a market price for this option to be quoted at $I_m = 22$, which is an admissible price with regard to (12), since we can consider a European put to be an ADMA put with $m = 1$ and so we see $22 \in [(Ke^{-rT} - S_0)_+, Ke^{-rT}] = [18.8, 118.8]$. In particular, the implied volatility problem is well-posed.

4.1.1 Implied volatility via root-finder

With the Black-Scholes formula (Theorem 3.5) we have an explicit formulation of the price of a European put as a function of volatility, leaving all other market parameters fixed. We can thus employ a root-finding algorithm to find an approximate solution σ^* such that $I_m = I(\sigma^*)$, or equivalently $I(\sigma^*) - I_m = 0$. Effective inversion of the Black-Scholes formula has been addressed in many publications. Here we use the Brent algorithm [Bre73] as proposed in [LOB19] to calculate the implied volatility. Utilizing the *brent()* implementation of the Brent algorithm in the pracma package, we obtain an implied volatility (see Appendix B):

$$\sigma^* = 0.5083729$$

4.1.2 Implied volatility via RM-algorithm

We now implement the RM-algorithm using a crude MC-estimator with sample sizes N for each RM-iteration, that is, in each iteration we approximate $I(\sigma)$ by $\frac{1}{N}\sum_{j=1}^{N} f(S_{(\sigma)}^{(i)})$, where the $S_{(\sigma)}^{(i)}$ are drawn iid from the price process with volatility σ and all other parameters fixed as previously, and f is the discounted payoff function of a European put option. The algorithm is run for $n = 1000$ iterations. It is required to set the initial learning rate α_0 and iterate σ_0. Moreover, we fix the learning rate to the initial value α_0 for 200 iterations. This showed to remarkably improve the initial convergence of the algorithm to the desired solution[4]. After 200 iterations, the proposed learning rate decay $\alpha_0/(n-200)^\rho$ kicks in, for the algorithm to converge. We follow the project description for the former and set $\alpha_0 = 2/(K + S_0)$, the latter we select to be $\sigma_0 = 0.2$. We run the algorithm for two different modulations $\rho \in \{0.8, 1\}$ of the learning rate and the following sample sizes: $N \in \{1, 3, 10, 31, 100, 316, 1000, 3162, 10000\}$. For each N, ρ the algorithm produces an estimate $\sigma_{N,\rho}$ of the implied volatility corresponding to $I_m = 22$. We run the algorithm for each pair $N_{sims} = 20$ times and compare the obtained RM-estimates to the robust σ^* calculated by inversion, by computing the (empirical) MSE for each N, ρ, that is:

$$MSE = \frac{1}{N_{sims}} \sum_{i=1}^{N_{sims}} (\sigma_{N,\rho}^{(i)} - \sigma^*)^2$$

The code for the RM-algorithm can be found in Appendix C and the implementation of the convergence analysis follows in Appendix D. Figure 1 plots the results from this convergence analysis. We observe that for both $\rho = 0.8$ and 1, the MSE appears to decay in the order of $O(1/N)$, which is equal to the order of decay in the variance of a crude MC-estimator with respect to sample size. This suggests that the variance of the MC-estimator plays an important role with regard to the accuracy of the RM-algorithm. Regarding the choice of ρ, we observe a very similar, albeit slightly better performance for $\rho = 0.8$, therefore it should be the preferred parameter choice in this context.

[3]Note that point 4 was already addressed in Lemmas 3.1, 3.6.

[4]This could be viewed in analogy to a burn-in period for an MCMC algorithm. More on this in 4.2.1.

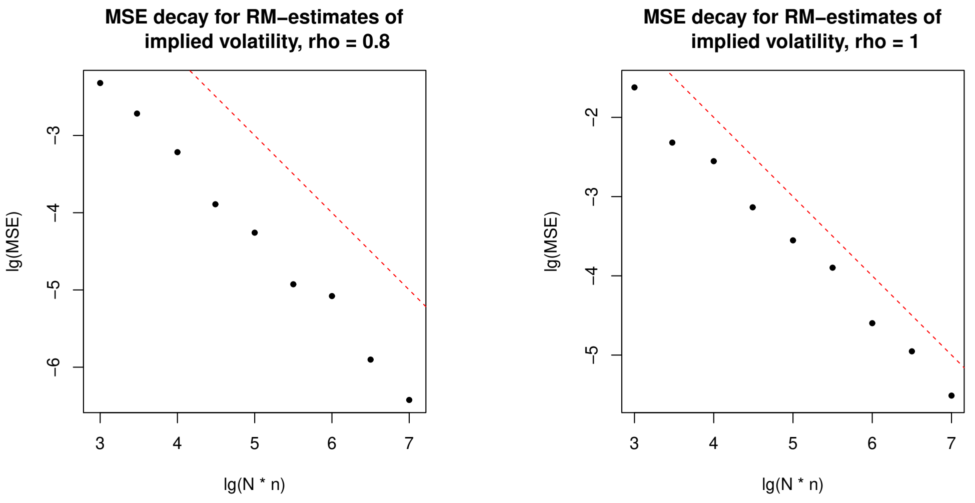

Figure 1: Plot of (empirical) MSEs with respect to the total number of samples produced during the runtime of the RM-algorithm (i.e. $N \cdot n$) on base-10 logarithmic axes. The dashed red line has slope -1 and hence indicates an order of decay $O(1/N)$ (since the number of iterations $n = 1000$ is constant). Note that this line is fixed in position across both plots, so we can observe slightly better performance in the $\rho = 0.8$ case.

4.2 ADMA put option

In this section we repeat the same application of the RM-algorithm as before, but this time to a 50-step ADMA put option. All market parameters remain the same for now and we still are concerned with the implied volatility problem for $I_m = 22 \in [19.3, 118.8]$, which again lies in the admissible price interval due to (12).

4.2.1 A stopping criterion for the RM-algorithm

Before, we let the RM-algorithm run for a fixed number of iterations and we gave 200 iterations without adjusting the learning rate. This choice is somewhat arbitrary, hence we propose the following more sophisticated stopping mechanism. Running the RM-algorithm without adjusting the learning rate *at all*, we observe a characteristic

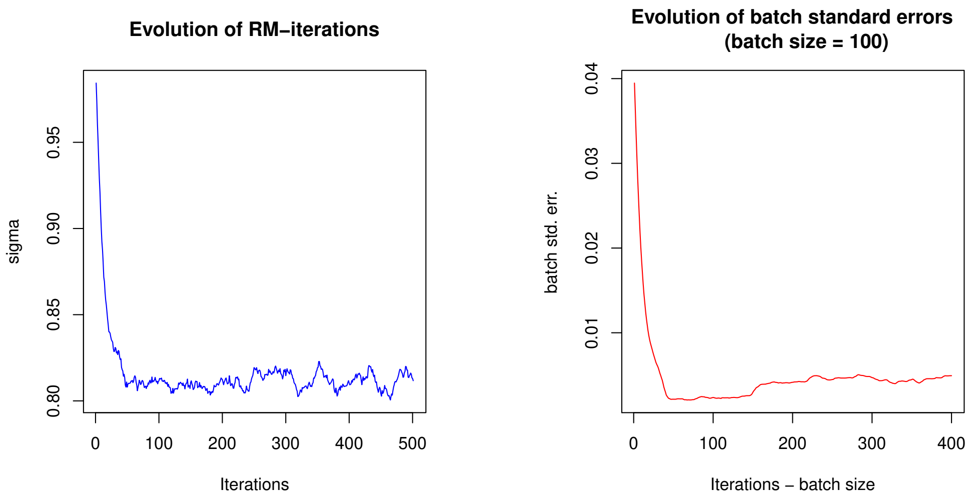

Figure 2: Iterates (left) and batch standard errors (right, batch size $b = 100$) produced in an RM-algorithm with constant learning rate $\alpha_0 = 2/(K + S_0)$, $\sigma_0 = 1$ and $N = 10000$ samples for the (crude) MC-estimator at each iteration, all other parameters as before. Observe the initial movement phase and the random oscillatory phase, in conjunction with the evolution of the BSDs.

behavior of the iterates in two phases. First, there is a rapid movement from the initial guess σ_0 to what we expect to be a level near the solution. Then the iterates start to enter a random oscillation (see Figure 2, left). It would be therefore convenient to allow the algorithm time until the general level of the solution is found, and then begin to adjust the learning rate to get convergence. To identify a suitable time where we expect the iterates to be in the neighborhood of the desired solution, we monitor the (empirical) batch standard errors (BSD) of the b last iterates[5]. We observe that the BSDs decay rapidly during the first phase, but as the algorithm enters the oscillatory phase, they begin to rise again, before they stabilize (see Figure 2, right). We select this as an indicator, that the iterates are in a general neighborhood of the solution and start adjusting the learning rate beyond that point. Since the BSDs stabilize, we can infer that any movement of the iterates is dictated predominantly by the learning rate alone, hence we select a tolerance *tol*, such that when the learning rate drops below it, the RM-algorithm terminates.

The implementation of the RM-algorithm for the given setting can be found in Appendix E. In addition to the algorithm itself, a Monte-Carlo pricing function has been defined to validate the accuracy of the procedure, as we do not have a confident value to compare to, as opposed to before. Using a batch size $b = 100$ and $tol = 10^{-4}$, we obtain $\sigma^* = 0.8107535$ and computing $I(\sigma^*)$ with the MC-pricer yields a value of 22.00101 (95%-CI: $[21.89878, 22.10323]$), which appears acceptable for the target of $I_m = 22$.

4.2.2 Importance sampling for the MC-estimator

As seen in the previous section, the performance of the RM-algorithm appears to depend significantly on the variance of the price estimation in each iteration. Our aim is to reduce the variance by introducing importance sampling (IS). Our IS-distribution will be samples from the price process, but with a modified drift $\tilde{r}$. Let $\tilde{S} = (\tilde{S}_{T/m}, \tilde{S}_{2T/m}, ..., \tilde{S}_T)$ be sampled from the process $(\tilde{S}_t)$, where $\tilde{S}_t = S_0 \exp((\tilde{r} - \sigma^2/2)t + \sigma W_t)$ and f the discounted payoff function of our ADMA put option. The IS estimate of the option price $I = \mathbb{E}[f(S)]$ is given by[6]:

$$\frac{1}{N}\sum_{i=1}^{N} f(\tilde{S}^{(i)})q(\tilde{S}^{(i)}), \tag{13}$$

where $q = p/\tilde{p}$ is the likelihood-ratio from Lemma 3.2 and the $\tilde{S}^{(i)}$ are iid. copies of $\tilde{S}$. How should $\tilde{r}$ be chosen, to reduce the variance, or equivalently, the standard error of this estimator?

First some heuristics: The ADMA put option becomes worthless, if the discrete monitoring average of the stock becomes too large, i.e. becomes larger than the strike K. If we sample from the default distribution of the stock price, which has a positive drift r, on average this will encourage the asset to gain value and this may lead to a significant portion of the probability mass of the discrete monitoring average to fall beyond the strike price. Hence, we could be sampling many times the value 0, which would make our estimator less efficient[7]. This would suggest that choosing the IS-drift $\tilde{r}$ smaller than r, perhaps even negative, might be useful.

Let us confirm this intuition by running a numerical study. We fix the following market parameters $S_0 = 100, r = 0.05, \sigma = 0.8, T = 0.2$ in the Black-Scholes model and consider an ADMA put option with strike $K = 150$. Next, we write an MC-pricer for the ADMA put, which uses importance sampling for an adjustable IS-drift $\tilde{r}$. We simply implement (13) using the convenient formula for the likelihood-ratio derived in Lemma 3.2. The code implementing this can be found in Appendix F. We now let the pricer run for different values of $\tilde{r} \in [-2, 1]$, and observe that indeed the standard error of the price estimate decreases, as we reduce the IS-drift away from 0.05. In fact, we observe an optimal drift $\tilde{r} \approx -0.5$ (see Figure 3, left), for which the MC-price standard error is nearly halved, making this pricer much more efficient.

4.2.3 Optimal importance sampling drift for different volatilities

In each iteration of the RM-algorithm our iterate σ is updated to a new value, which in turn changes the dynamics of the Black-Scholes asset we need to simulate to estimate the ADMA put price. Just now, we saw that using importance sampling in the MC-pricer can be very useful, hence it is of interest to know the optimal IS-drift $\tilde{r}$ for different values of σ. To that end, we implement the strategy proposed in the project description (point 7), that is, for a range of $\sigma \in [0, 5]$, we try to find the optimal drift $\tilde{r}$ by solving[8]:

$$\tilde{r}(\sigma) = \arg\min_{\eta} \frac{1}{\bar{N}}\sum_{i=1}^{\bar{N}} f^2(S^{(i)})q(S^{(i)}), \tag{14}$$

where the $S^{(i)}$ are this time iid. copies of $S = (S_{T/m}, S_{2T/m}, ..., S_T)$ from the original price process with volatility σ and all other market parameters fixed as before, f is the discounted ADMA put payoff function and q is again the likelihood-ratio for a modified drift η.

[5]i.e. if σ_k is the current iterate and $k > b$, then the BSD is $\sqrt{\sum_{i=k-b}^{k}(\sigma_i - \bar{\sigma}_i)^2/(b-1)}$

[6]Essentially this is Algorithm 6.2 from the lecture notes, which also prescribes how CIs and standard errors can be computed.

[7]Similar to Example 6.4 in the lecture notes. In Appendix F, Figure 6 shows how a portion of the probability mass can fall beyond strikes of 120 or even 150.

[8]This is indeed coincides with the procedure discussed on page 58 of the lecture notes.

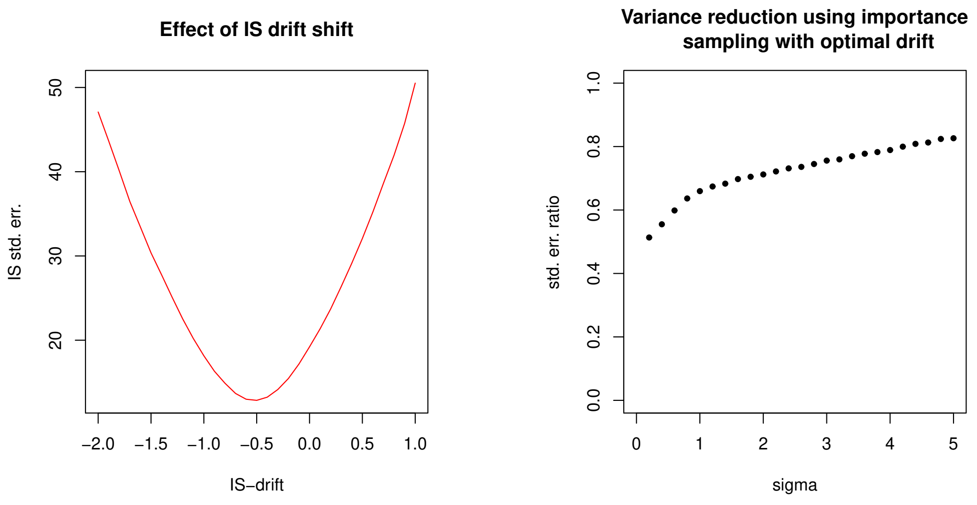

Figure 3: Variance reduction effect due to importance sampling. Left: Standard error of MC-pricer with different IS-drifts $\tilde{r}$ for ADMA put option ($K = 150$) with market parameters $S_0 = 100, r = 0.05, \sigma = 0.8, T = 0.2$. Right: Ratio of standard errors from MC-pricer with optimal IS-drift with respect to crude MC-pricing for different volatilities σ, other parameters as before. In each case the MC-pricers worked with $N = 10^5$ samples.

We implement this procedure for $\sigma \in \{0.1, 0.2, ..., 5\}$ and $\bar{N} = 10000$, then use a degree-5 spline interpolation to approximate the optimal drifts for the volatilities on the continuous range (for this we use the *bs()* function from the R-package splines). The code can be found in Appendix K, and Figure 4 shows the resulting optimal IS-drifts. One can observe that the optimal $\tilde{r}$ decreases, as volatility increases. This makes sense, since with higher

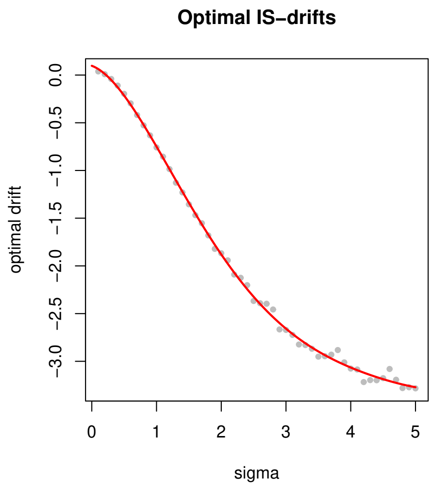

Figure 4: Optimal IS-drifts $\tilde{r}$ for different values of σ. The grey dots are the calculated minima from optimizing (14) for the corresponding σ, the red curve is the degree-5 spline interpolation.

volatility it is reasonable to assume that the distribution of the discrete monitoring average of a Black-Scholes asset will be more spread out over the domain, hence more values "leak" into the super-strike price range. This in turn creates more samples of the value 0 when appliying the ADMA-put payoff, reducing MC-efficiency. In addition, for our $\sigma^* = 0.8107535$, we recover an optimal $\tilde{r} \approx -0.53$ using this strategy, with coincides well with what we saw in Figure 3 (left). We can now also quantify the variance reduction effect for different values of σ (see Figure 3, right). It appears that the reduction works best for volatilities closer to 0, where we achieve almost a halving of the MC-pricer standard error. However, even near $\sigma = 5$ the standard error of a MC-pricer with optimal IS-drift is still just 80% of that from a crude MC-pricer.

4.2.4 Applying MC-pricers with importance sampling to RM-algorithm

At this point we can equip the RM-algorithm with 3 different MC-pricers to estimate the expectation in each iteration: a crude MC-pricer, a MC-pricer with constant IS-drift and a MC-pricer with adaptive optimal IS-drift according to the current iterate σ. We conclude our numerical experiments by comparing the performance of these strategies. Our market parameters are set as $S_0 = 100, r = 0.05, T = 0.2$ and we consider an ADMA put option with strike $K = 150$. Further, we assume to observe a valid market price[9] $I_m = 49.3 \in [48.996, 148.5]$ with respect to (12). The algorithms are run $N_{sims} = 10$ times with the same hyper-parameters, that is: $\rho = 0.8$, BSD-size $b = 100$, initial value $\sigma_0 = 1$, initial learning rate $\alpha_0 = 2/(K + S_0)$, MC-sample size $N = 10^5$. For the constant IS-drift strategy, we select $\tilde{r} = -0.5$[10]. Code for the adjusted RM-algorithms can be found in Appendix H, Appendix I and their analysis in Appendix J. The results of are displayed in Figure 5.

We observe that the means of the RM-estimates are relatively stable at $\sigma^* \approx 0.787$[11]. Further, it is visible that the IS-strategies drastically reduce the variance of the RM-estimates for the implied volatility. This is also confirmed by inspecting the standard errors: compared to the crude MC-pricing approach the standard error is nearly halved for the constant IS-drift approach and more than halved for the optimal IS-drift approach, where the optimal IS-drift algorithm appears to outperform the other two in that regard. We can thus expect it to be the most efficient in estimating the implied volatility.

With respect to runtime, we notice an increase when comparing the crude MC to both IS strategies. We can hypothesize that this is due to the extra computational load required to compute the likelihood-ratios and adjust the MC-sample at each iteration. It is worth noting however, that the average runtime of the RM-algorithm with optimal IS-drift strategy is lower than that which employs the constant IS-drift strategy. It would appear that adapting the IS-drift causes the iterates to move faster from the initial guess into a neighborhood of the implied volatility, where our stopping criterion takes effect.

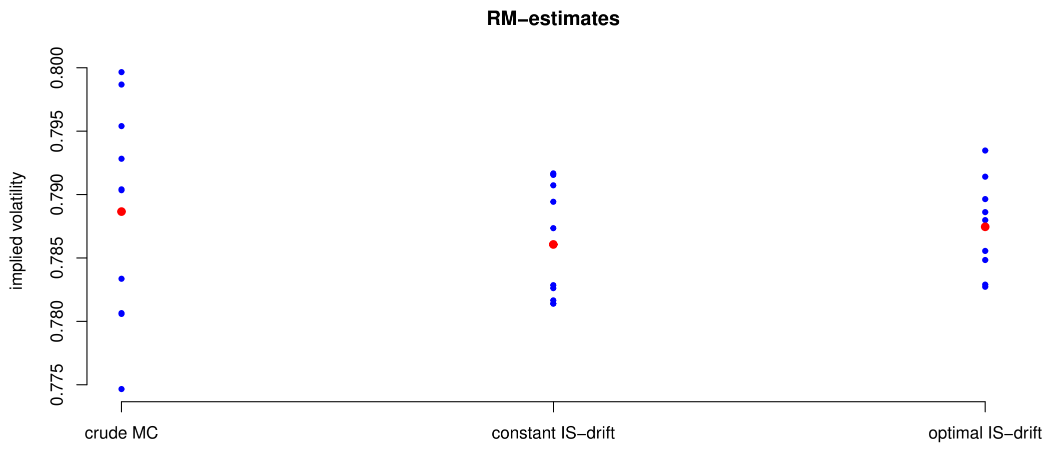

Figure 5: RM-estimates of the implied volatility for the three algorithms, having run them $N_{sims} = 10$ times each. Each of the blue dots represents the estimate of one run of the corresponding RM-algorithm. The red dots are the means of all runs for each algorithm. We also report the values of the means: 0.7887 (crude), 0.7861 (constant), 0.7875 (optimal), the empirical standard errors: 0.0085 (crude), 0.0045 (constant), 0.0035 (optimal), and the average runtimes: 53.49 sec (crude), 71.23 sec (constant), 69.60 sec (optimal).

5 Conclusion

In this project we applied the RM-algorithm to the stochastic approximation problem of implied volatility in the Black-Scholes model. Upon reviewing some of the relevant theory from mathematical finance, we implemented and investigated performance of the algorithm for European and path-dependent Asian options. Based on experiments in that setting, we introduced an effective stopping procedure for the algorithm by monitoring batch standard errors[12]. Finally, we investigated the effect of variance reduction to the performance of the RM-algorithm and were able to increase efficiency using both non-adaptive and adaptive methods of importance sampling, the latter appearing to yield the best results.

[9] Note that the previous price $I_m = 22$ is not valid for $K = 150$. The choice of $I_m = 49.3$ is chosen such that the market volatility is ≈ 0.8 as in 4.2.1 for consistency, where we selected this value based on MC-pricing with $\sigma = 0.8$ and all other market parameters as before. This also gives us an idea about what IS-drift should be used for the constant IS-strategy (see also next footnote).

[10] Since by design of this experiment we expect the general area of the implied volatility to be ≈ 0.8, we selected the optimal drift from before ($\tilde{r} = -0.5$). In practice, when one has only access to market prices, a pilot run may be useful to determine in what neighborhood the implied volatility lies in, and select $\tilde{r}$ accordingly.

[11] Calculating the ADMA put price with respect to this volatility with a MC-pricer using optimal drift importance sampling and $N = 10^6$ samples, we get $I(\sigma^*) = 49.30112$, (95%-CI: $[49.2764, 49.32583]$), which again appears acceptable for $I_m = 49.3$.

[12] Going beyond this work, it may be an interesting avenue of further inquiry, e.g. it is conceivable that the corresponding hyper-parameter b should be tuned with respect to the initialization of the algorithm (something we did not further discuss here).

References

[AG07] S. Asmussen and P.W. Glynn. Stochastic simulation: algorithms and analysis. *Springer Science Business Media*, 57, 2007.

[Bre73] R. P. Brent. *"Chapter 4: An Algorithm with Guaranteed Convergence for Finding a Zero of a Function" in Algorithms for Minimization without Derivatives.* Prentice-Hall, NJ, 1973.

[LOB19] S. Liu, W. Oosterlee, and S. Bohte. Pricing options and computing implied volatilities using neural networks. *Risks*, 7:16, 2019.

[Nob22] F. Nobile. *Lecture notes: Stochastic Simulation.* 2021-2022.

[RM85] H. Robbins and S. Monro. A stochastic approximation method, in Herbert Robbins selected papers. *Springer, New York, NY*, pages 102–109, 1985.

[Shr04] S. Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models.* Springer Finance, 2004.

Appendix

A BS_functions.r

```r
#Functions with respect to the Black-Scholes model,
#with relevant parameters as defaults

#price process simulation (at maturity)
#we use the GBM-formula and the standard sampling routines for the normal dist.
S <- function(N, S_0 = 100, r = 0.05, sigma = 0.1, T = 0.2){
  return(S_0 * exp((r - sigma^2/2)*T + sigma * sqrt(T) * rnorm(N)))
}

#discounted European put option payoff
f <- function(S = 100, K = 120, r = 0.05, T = 0.2){
  return(exp(-r*T) * pmax(K-S,0))
}

#price process simulation (path) with discrete (uniform) monitoring:
#the function computes a matrix with N rows, each row has m+1 entries
#corresponding to a path of the price process. Essentially, this is Alg. 3.4
#from the lecture notes. For efficiency, the normal increments are generated
#simultaneously and all calculations have been vectorized.
S_path <- function(N, m = 50, S_0 = 100, r = 0.05, sigma = 0.1,
                   T = 0.2){
  dt <- T / m
  t <- seq(0,T,dt)
  X <- matrix(rnorm((m+1)*N, 0, sqrt(dt)), nrow = N)
  X[,1] <- numeric(length = N)
  X <- t(apply(X,1,cumsum))
  S <- S_0 * exp(t( (r-sigma^2/2) * t + t(sigma * X) ))
  return(S)
}

#discounted put option payoff (Asian), can handle multiple paths as matrix
#generated from previous function
g <- function(S_path, K = 120, r = 0.05, T = 0.2){
  S <- S_path[,-1]
  if(is.null(dim(S))) return(exp(-r*T) * pmax(K - mean(S),0))
  return(as.vector(exp(-r*T) * pmax(K - rowMeans(S),0)))
}

#auxiliary function w for BS-formula wrt. volatility
w <- function(sigma = 0.1, S = 100, K = 120, r = 0.05, T = 0.2){
  return((log(K/S) - (r-sigma^2/2) * T) / sigma / sqrt(T))
}

#pricing formula (Black-Scholes formula)
Put <- function(sigma = 0.1, S = 100, K = 120, r = 0.05, T = 0.2){
  return(exp(-r*T) * K * pnorm(w(sigma, S, K, r, T)) -
           S * pnorm(w(sigma, S, K, r, T) - sigma * sqrt(T)))
}

#pricing formula wrt. sigma as objective for root-finder
I <- function(sigma) Put(sigma)

#drift-shift likelihood ratio (single path vector), computes likelihood ratio
#as derived in Lemma 3.2 to a single vector of length m
lratio <- function(x, S_0 = 100, r = 0.05, r_IS = 0.1, sigma = 0.1, T = 0.2){
  m <- length(x)
  dt <- T/m
  x_end <- x[length(x)]
  out <- exp(-(r_IS-r) * (2 * log(x_end/S_0) - m * dt * (r + r_IS - sigma^2))
             / (2 * sigma^2))
  return(out)
}

#drift-shift likelihood ratio (vectorized), employs the previous function to
#compute likelihood-ratios for multiple vectors of length m simultaneously
lratio_vectorized <- function(X, S_0 = 100, r = 0.05, r_IS = 0.1, sigma = 0.1,
                              T = 0.2){
  if(is.null(dim(X))) return(as.numeric(lapply(X, lratio, S_0 = S_0, r = r,
                                           r_IS = r_IS, sigma = sigma, T = T)))
  return(apply(X, 1, lratio, S_0 = S_0, r = r, r_IS = r_IS, sigma = sigma, T = T))
}
```

B BS_IV_rootfind.r

```
#We use the Brent root finding algorithm to calculate the implied volatility
#in the given setting:
source("BS_functions.r") #load auxiliary functions
library(pracma) #package implementing the Brent root-finding algorithm
objective <- function(sigma) I(sigma) - 22 #define objective to find root
sigma_IV <- brent(objective, 0.1, 2)$root #run Brent w/ search area [0.1,2]
#sigma_IV = 0.5083729
```

C RM_IV_Put.r

```
#We implement the RM-algorithm to find the IV with regard to the put option price
source("BS_functions.r") #load auxiliary functions

#note that we provide default input data according to the hint for alpha_0
RM_IV <- function(n = 1000, N = 10000, I = 22, sigma_0 = 0.2, alpha_0 = 2/(120+100),
                  rho = 1, lrate_delay = 200){
  sigma <- sigma_0
  sigmas <- sigma_0
  #RM-update:
  sigma_new <- sigma - alpha_0 * (mean(f(S(N, sigma = sigma))) - I) #crude MC-est.
  iter <- 0
  #iterate...
  while(iter < n){
    sigma <- sigma_new
    alpha <- alpha_0
    #learning rate is kept constant until lrate_delay iterations
    #to give the algorithm time to evolve to the area
    #of the solution, then it is modulated as proposed in the project description
    if(iter > lrate_delay) alpha <- alpha_0 / (iter+1-lrate_delay)^rho
    sigma_new <- sigma - alpha * (mean(f(S(N, sigma = sigma))) - I) #crude MC-est.
    sigmas <- c(sigmas, sigma_new)
    iter <- iter + 1
  }
  return(list(sigma = sigma_new, sigmas = sigmas))

  sigma_IV_RM <- RM_IV()
  # test: sigma_IV_RM$sigma = 0.5076115
}
```

D RM_IV_Put_Analysis.r

```
source("BS_IV_rootfind.r") #compute reference IV via root-finder (Brent)
source("RM_IV_Put.r") #load RM-algorithm
#We run the analysis of the RM-algorithm for calculating the BS implied vol.
#select n = 1000 iterations, N = 1, 3, 10, 31, 100, 316, 1000, 3162, 10000
#samples per iteration
#rho = 0.8 or 1
n <- 1000 #number of iterations of RM-algorithm
Ns <- floor(10^seq(0,4,0.5)) #sample sizes for MC-estimator to be tested
rho <- 1 #can be adjusted to 0.8 or 1 as desired
Nsims <- 20 #number of simulations to estimate MSE

MSE <- numeric(length = length(Ns))

for(i in 1:length(Ns)){
  N <- Ns[i]
  sims <- numeric(length = Nsims)
  for(j in 1:Nsims){
    sims[j] <- RM_IV(n,N, rho = rho)$sigma
  }
  MSE[i] <- mean((sims - sigma_IV)^2)
}

#plot simulation results
par(pty="s")
plot(log10(n*Ns), log10(MSE), pch = 20, main = "MSE decay for RM-estimates of
     implied volatility, rho = 1", xlab = "lg(N * n)", ylab = "lg(MSE)",
     asp = 1) #plot MSEs on log-scales
abline(a = 2, b = -1, col = "red", lty = "dashed") #O(1/N) for comparison
```

E BS_IV_Put_Asian.r

```
#We implement the RM-algorithm to find the IV with regard to the Asian
#put option price, having introduced a stopping procedure for the algorithm
source("BS_functions.r") #get auxiliary functions

#n = maximum iterations
#sd_monitor = flag to determine, if the stopping criterion should be used,
#otherwise the algorithm runs without learning rate!
#all other parameters according to notation in report
RM_IV_Asian <- function(n = 500, N = 10000, I = 22, sigma_0 = 1, alpha_0 = 2/(120+100),
                        rho = 0.8, K = 120, batch_sd = 100, sd_monitor = FALSE, tol = 10^-4){
  sigma <- sigma_0
  #RM-update (w/ crude MC-estimator):
  sigma_new <- sigma - alpha_0 * (mean(g(S_path(N, sigma = sigma), K = K)) - I)
  sigmas <- sigma_new
  batch_sds <- c()
  iter <- 0
  learn_flag <- FALSE
  k <- 1
  #iterate...
  while(iter < n){
    sigma <- sigma_new
    alpha <- alpha_0
    #adjust learning rate only when batch-sd starts to rise again
    if(learn_flag){
      alpha <- alpha_0 / k^rho
      k <- k+1
      if(alpha < tol) break
    }
    sigma_new <- sigma - alpha * (mean(g(S_path(N, sigma = sigma), K = K)) - I)
    sigmas <- c(sigmas, sigma_new)
    iter <- iter + 1
    #batch-sd monitoring routine: when batch-sd starts to go rise again,
    #let the learning rate kick in...
    if(iter > batch_sd){
      batch_sds <- c(batch_sds, sd(sigmas[(iter - batch_sd):iter]))
    }
    if(length(batch_sds) > 1 & sd_monitor){
      if(batch_sds[length(batch_sds)] > batch_sds[length(batch_sds)-1]){
        learn_flag <- TRUE
      }
    }
    print(iter) #print to see that everything works (comment out if desired)
  }
  return(list(sigma = sigma_new, sigmas = sigmas, batch_sds = batch_sds))
}

#we implement a MC-pricer to validate accuracy
Put_Asian_pricer <- function(N = 10^5, S_0 = 100, r = 0.05, sigma = 0.8,
                              K = 120, T = 0.2){
  price <- mean(g(S_path(N, S_0 = S_0, r = r, sigma = sigma, T = T), K = K,
                 r = r, T = T))
  se <- sd(g(S_path(N, S_0 = S_0, r = r, sigma = sigma, T = T), K = K, r = r,
             T = T))
  price_CI_lower <- price - qnorm(0.975) * se / sqrt(N)
  price_CI_upper <- price + qnorm(0.975) * se / sqrt(N)
  return(list(price = price, CI_lower = price_CI_lower,
              CI_upper = price_CI_upper, se = se))
}

#plot results
sigma_IV_RM_Asian <- RM_IV_Asian(sd_monitor = TRUE)
plot(sigma_IV_RM_Asian$sigmas, type = "l", main = "Evolution of RM-iterations",
     ylab = "sigma", xlab = "Iterations", col = "blue")
plot(sigma_IV_RM_Asian$batch_sds, type = "l",
     main = "Evolution of batch standard errors
     (batch size = 100)",
     xlab = "Iterations - batch size",
     ylab = "batch std. err.", col = "red")

#give out results
sigma_IV_RM_Asian$sigma
#check accuracy with high-iteration MC-pricer
Put_Asian_pricer(sigma = sigma_IV_RM_Asian$sigma, K = 120)

# sigma_IV_RM_Asian$sigma = 0.8107535

#output of pricer with computed sigma
#$price
#[1] 22.00101
#$CI_lower
#[1] 21.89878
#$CI_upper
#[1] 22.10323
#$se
#[1] 16.4937
```

F IS_pricing.r

```r
#functions/experiments regarding importance sampling for the MC-pricer
source("BS_functions.r") #auxiliary functions
source("optimal_drift.r") #to calculate optimal drifts for different sigmas

#crude MC-pricer for the Asian put
Put_Asian_pricer <- function(N = 10^5, S_0 = 100, r = 0.05, sigma = 0.8,
                             K = 120, T = 0.2){
  price <- mean(g(S_path(N, S_0 = S_0, r = r, sigma = sigma, T = T), K = K,
                r = r, T = T))
  se <- sd(g(S_path(N, S_0 = S_0, r = r, sigma = sigma, T = T), K = K,
          r = r, T = T))
  price_CI_lower <- price - qnorm(0.975) * se / sqrt(N)
  price_CI_upper <- price + qnorm(0.975) * se / sqrt(N)
  return(list(price = price, CI_lower = price_CI_lower,
              CI_upper = price_CI_upper, se = se))
}

#MC-pricer with importance sampling for Asian put
Put_Asian_pricer_IS <- function(N = 10^5, S_0 = 100, r = 0.05, r_IS = 0.1,
                                sigma = 0.8, K = 120, T = 0.2){
  ISample <- S_path(N, S_0 = S_0, r = r_IS, sigma = sigma, T = T)
  frac <- g(ISample, K = K, r = r, T = T) * lratio_vectorized(ISample[,-1],
                             S_0 = S_0, r = r, r_IS = r_IS, sigma = sigma, T = T)
  price <- mean(frac)
  se <- sd(frac)
  price_CI_lower <- price - qnorm(0.975) * se / sqrt(N)
  price_CI_upper <- price + qnorm(0.975) * se / sqrt(N)
  return(list(price = price, CI_lower = price_CI_lower,
              CI_upper = price_CI_upper, se = se))
}

#testing the pricers (observe variance/std. err. reduction)
test1 <- Put_Asian_pricer(K = 150, sigma = 0.8)
test2 <- Put_Asian_pricer_IS(K = 150, sigma = 0.8, r_IS = -0.5)

#search for optimal IS-drift for the given parameters
rs <- seq(-2,1,0.1)
ses <- numeric(length = length(rs))
prices <- numeric(length = length(rs))
for(i in 1:length(rs)){
  Pricer <- Put_Asian_pricer_IS(K = 150, sigma = 0.8, r_IS = rs[i])
  ses[i] <- Pricer$se
  prices[i] <- Pricer$price
}
plot(rs,ses, type = "l", main = "Effect␣of␣IS␣drift␣shift", xlab = "IS-drift",
     ylab = "IS␣std.␣err.", col = "red")

#variance reduction in terms of sigma for optimal IS-drifts
sigmas <- seq(0.2,5,0.2)
sd_ratios <- numeric(length = length(sigmas))
for(i in 1:length(sigmas)){
  sd_ratios[i] <- Put_Asian_pricer_IS(r_IS = optimal_r(sigmas[i]),
                                      K = 150, sigma = sigmas[i])$se /
    Put_Asian_pricer(K = 150, sigma = sigmas[i])$se
  print(i)
}
plot(sigmas, sd_ratios, xlim = c(0,5), ylim = c(0,1), pch = 20,
     xlab = "sigma", ylab = "std.␣err.␣ratio",
     main = "Variance␣reduction␣using␣importance
     sampling with optimal drift")
```

G optimal_drift.r

```
#finding optimal drift
library(splines) #package to do spline interpolation
source("BS_functions.r") #auxiliary functions

#objective (expression (18) in project description)
#to be minimized with respect to drift r_IS (log-transformed)
objective <- function(r_IS, S = S_sim, sigma = 0.1, r = 0.05,
                      K = 150, T = 0.2){
  objective <- numeric(length = length(r_IS))
  for(i in 1:length(r_IS)){
    objective[i] <- log(mean(g(S, K, r, T)^2 *
                              lratio_vectorized(S[,-1], S[1,1], r,
                                                r_IS = r_IS[i], sigma, T)))
  }
  return(objective)
}

#simulate N paths and optimize r_IS for given range of volatilities (sigmas)
#same paths used for optimization over each sigma
N <- 10000
sigmas <- seq(0.1,5,0.1)
optimal_rs <- numeric(length = length(sigmas))

for(i in 1:length(sigmas)){
  S_sim <- S_path(N, S_0 = 100, r = 0.05, sigma = sigmas[i], T = 0.2)
  optimal_rs[i] <- optimize(objective, interval = c(-6,1),
                            sigma = sigmas[i])$minimum
}

#plot optimal r_IS wrt. sigmas
plot(sigmas, optimal_rs, pch = 20, main = "Optimal IS-drifts",
     xlab = "sigma", ylab = "optimal drift", col = "gray")

#spline interpolation (degree 5) of optimal drifts based on sigma
data <- data.frame(sigmas, optimal_rs)
spline_fit <- lm(optimal_rs ~ bs(sigmas, degree = 5), data = data)

optimal_r <- function(sigma){
  return(as.numeric(predict(spline_fit, newdata = list(sigmas = sigma))))
}

#plot spline interpolation
x <- seq(0,5,0.01)
lines(x, optimal_r(x), type = "l", col = "red", lwd = 2)
```

H RM_IV_Put_Asian_IS.r

```
#We implement the RM-algorithm to find the IV with regard to the Asian
#put option price, using importance sampling
source("BS_functions.r")

#MC pricer with IS using modified drift
Put_Asian_pricer_IS <- function(N = 10^5, S_0 = 100, r = 0.05, r_IS = -0.5,
                                sigma = 0.8, K = 150, T = 0.2){
```

```
  ISample <- S_path(N, S_0 = S_0, r = r_IS, sigma = sigma, T = T)
  frac <- g(ISample, K = K, r = r, T = T) *
    lratio_vectorized(ISample[,-1], S_0 = S_0, r = r, r_IS = r_IS,
                      sigma = sigma, T = T)
  price <- mean(frac)
  se <- sd(frac)
  price_CI_lower <- price - qnorm(0.975) * se / sqrt(N)
  price_CI_upper <- price + qnorm(0.975) * se / sqrt(N)
  return(list(price = price, CI_lower = price_CI_lower,
              CI_upper = price_CI_upper, se = se))
}

RM_IV_Asian_IS <- function(n = 500, N = 10000, I = 49.3, sigma_0 = 1,
                           alpha_0 = 2/(150+100), rho = 0.8, K = 150,
                           batch_sd = 100, sd_monitor = FALSE, tol = 10^-4){
  sigma <- sigma_0
  #this time use MC-pricer with IS
  sigma_new <- sigma - alpha_0 *
    (Put_Asian_pricer_IS(N, K = K, sigma = sigma)$price - I)
  sigmas <- sigma_new
  batch_sds <- c()
  iter <- 0
  learn_flag <- FALSE
  k <- 1
  while(iter < n){
    sigma <- sigma_new
    alpha <- alpha_0
    #adjust learning rate only when batch-sd starts to rise again
    if(learn_flag){
      alpha <- alpha_0 / k^rho
      k <- k+1
      if(alpha < tol) break
    }
    sigma_new <- sigma - alpha *
      (Put_Asian_pricer_IS(N, K = K, sigma = sigma)$price - I)
    sigmas <- c(sigmas, sigma_new)
    iter <- iter + 1
    if(iter > batch_sd){
      batch_sds <- c(batch_sds, sd(sigmas[(iter - batch_sd):iter]))
    }
    if(length(batch_sds) > 1 & sd_monitor){
      if(batch_sds[length(batch_sds)] > batch_sds[length(batch_sds)-1]){
        learn_flag <- TRUE
      }
    }
    #print(iter)
  }
  return(list(sigma = sigma_new, sigmas = sigmas, batch_sds = batch_sds))
}
```

I RM_IV_Asian_IS_optimal_drift.r

```
#We implement the RM-algorithm to find the IV with regard to the Asian
#put option price, using importance sampling with optimal drift
source("BS_functions.r") #auxiliary functions
source("optimal_drift.r") #gives access to the optimal drift function wrt. sigma

RM_IV_Asian_IS_OD <- function(n = 500, N = 10000, I = 49.3, sigma_0 = 1,
                              alpha_0 = 2/(150+100), rho = 0.8, K = 150,
                              batch_sd = 100, sd_monitor = FALSE, tol = 10^-4){
```

```
  sigma <- sigma_0
  #this time use MC pricer with optimal IS-drift
  drifts <- optimal_r(sigma)
  sigma_new <- sigma - alpha_0 * (Put_Asian_pricer_IS(N, K = K, sigma = sigma,
                                              r_IS = drifts)$price - I)
  sigmas <- sigma_new
  batch_sds <- c()
  iter <- 0
  learn_flag <- FALSE
  k <- 1
  while(iter < n){
    sigma <- sigma_new
    r_IS <- optimal_r(sigma)
    alpha <- alpha_0
    #adjust learning rate only when batch-sd starts to rise again
    if(learn_flag){
      alpha <- alpha_0 / k^rho
      k <- k+1
      if(alpha < tol) break
    }
    sigma_new <- sigma - alpha * (Put_Asian_pricer_IS(N, K = K, sigma = sigma,
                                              r_IS = r_IS)$price - I)
    sigmas <- c(sigmas, sigma_new)
    drifts <- c(drifts, r_IS)
    iter <- iter + 1
    if(iter > batch_sd){
      batch_sds <- c(batch_sds, sd(sigmas[(iter - batch_sd):iter]))
    }
    if(length(batch_sds) > 1 & sd_monitor){
      if(batch_sds[length(batch_sds)] > batch_sds[length(batch_sds)-1]){
        learn_flag <- TRUE
      }
    }
    print(iter)
  }
  return(list(sigma = sigma_new, sigmas = sigmas,
              batch_sds = batch_sds, drifts = drifts))
}
```

J RM_Analysis.r

```
#Performance analysis of RM for BS-IV of Asian put option
library(tictoc) #package for time measurement
source("RM_IV_Put_Asian.r") #load RM-alg. with crude MC-pricer
source("RM_IV_Put_Asian_IS.r") #load RM-alg. with constant IS-drift MC-pricer
source("RM_IV_Put_Asian_IS_optimal_drift.r") #...with optimal IS-drift MC-pricer
#IMPORTANT: comment out diagnostics in the respective scripts before running
#this analysis!

Nsims <- 10
sigmas_crude <- numeric(length = length(Nsims))
times_crude <- numeric(length = length(Nsims))
sigmas_IS <- numeric(length = length(Nsims))
times_IS <- numeric(length = length(Nsims))
sigmas_opt_IS <- numeric(length = length(Nsims))
times_opt_IS <- numeric(length = length(Nsims))

#run RM-algorithms Nsims times and record estimates + runtimes
print("Starting performance analysis...")
for(i in 1:Nsims){
  tic()
  sigmas_crude[i] <- RM_IV_Asian(sd_monitor = TRUE)$sigma
  t <- toc()
  times_crude[i] <- t$toc - t$tic

  tic()
  sigmas_IS[i] <- RM_IV_Asian_IS(sd_monitor = TRUE)$sigma
  t <- toc()
  times_IS[i] <- t$toc - t$tic

  tic()
  sigmas_opt_IS[i] <- RM_IV_Asian_IS_OD(sd_monitor = TRUE)$sigma
  t <- toc()
  times_opt_IS[i] <- t$toc - t$tic
  print(paste("Iteration", i, "done.", sep = " "))
}

#average runtimes for each algorithm
mean(times_crude)
mean(times_IS)
mean(times_opt_IS)

#standard errors of the estimates
sd(sigmas_crude)
sd(sigmas_IS)
sd(sigmas_opt_IS)

#plot RM-estimates for all 3 algorithms
sds <- rbind(sigmas_crude,sigmas_IS,sigmas_opt_IS)
matplot(sds, col = "blue", pch = 20,
        axes = FALSE, ylab = "implied volatility", main = "RM-estimates")
axis(2)
axis(side=1, at=1:nrow(sds),
     labels=c("crude MC", "constant IS-drift", "optimal IS-drift"))
points(rowMeans(sds), pch = 19, col = "red")
```

K Density of discrete monitoring average

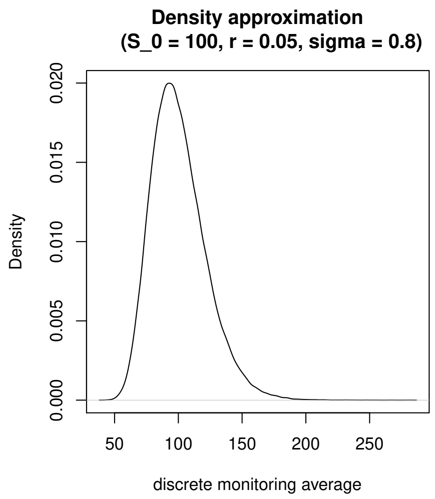

Figure 6: Density approximation for discrete monitoring average of a Black-Scholes asset using in-built R-function *density()*. Parameters as in title of plot.