Stochastic Simulations

Autumn Semester 2024

Prof. Fabio Nobile *Assistant: Matteo Raviola*

Lab 05 – 10 October 2024

The Monte Carlo method and Variance Reduction Techniques

Exercise 1

A simulator would like to produce an unbiased estimate of $\mathbb{E}(XY)$, where the two independent random variables X and Y have bounded first moments and can be generated by a stochastic simulation. To this end, she simulates $R \in \mathbb{N}$ replications $X_1, \ldots, X_R$ of X and, independently of this, R replications $Y_1, \ldots, Y_R$ of Y. She thus has the following two natural estimators for $\mathbb{E}(XY)$ at her disposal:

$$\mathrm{Est}_1 := \left(\frac{1}{R}\sum_{r=1}^{R} X_r\right)\left(\frac{1}{R}\sum_{r=1}^{R} Y_r\right) \quad \text{and} \quad \mathrm{Est}_2 := \frac{1}{R}\sum_{r=1}^{R} X_r Y_r \, .$$

1. Verify that both estimators Est_1 and Est_2 are unbiased.
2. Show that $\mathrm{Var}(\mathrm{Est}_1) < \mathrm{Var}(\mathrm{Est}_2)$.
3. Use the delta method to show that $\sqrt{R}(\mathrm{Est}_1 - \mu_x\mu_y) \xrightarrow{d} N(0, \tau^2)$. Find τ^2 explicitly and derive a $1-\alpha$ asymptotic confidence interval.

Solution

1. Introducing the notation $\bar{X}_R = \frac{1}{R}\sum_{i=1}^{R} X_i$ and $\bar{Y}_R = \frac{1}{R}\sum_{r=1}^{R} Y_j$, it holds by the independence of the random variables X_i and Y_j, and the identical distribution of elements in the sequence $X_1, X_2, \ldots, X_R$ and ditto for elements in the sequence $Y_1, Y_2, \ldots, Y_R$, that for any $1 \le i, j \le R$,

   $$\mathbb{E}X_iY_j = \mathbb{E}X_i\mathbb{E}Y_j = \mathbb{E}X\mathbb{E}Y = \mathbb{E}XY.$$

   Consequently,

   $$\mathbb{E}\bar{X}_R\bar{Y}_R = \frac{1}{R^2}\sum_{i,j=1}^{R} \mathbb{E}X_iY_j = \frac{1}{R^2}\sum_{i,j=1}^{R} \mathbb{E}XY = \mathbb{E}XY,$$

   and

   $$\mathbb{E}\frac{1}{R}\sum_{r=1}^{R} X_rY_r = \frac{1}{R}\sum_{r=1}^{R} \mathbb{E}XY = \mathbb{E}XY.$$

2. Let us recall first that since the sequence $X_1, X_2, \dots$ is iid,

$$\mathrm{Var}(\bar{X}_R) = \mathrm{Cov}(\bar{X}_R, \bar{X}_R) = \frac{1}{R}\mathrm{Var}(X),$$

and

$$\mathbb{E}\bar{X}_R = \mathbb{E}X.$$

(And, similarly, $\mathrm{Var}(\bar{Y}_R) = \frac{1}{R}\mathrm{Var}(Y), \quad \mathbb{E}\bar{Y}_R = \mathbb{E}Y.$)

Using that $\bar{X}_R$ and $\bar{Y}_R$ are independent,

$$\begin{aligned}
\mathrm{Var}(\mathrm{Est}_1) &= \mathbb{E}\bar{X}_R^2\bar{Y}_R^2 - \left(\mathbb{E}\bar{X}_R\bar{Y}_R\right)^2 \\
&= \mathbb{E}\bar{X}_R^2\mathbb{E}\bar{Y}_R^2 - \left(\mathbb{E}\bar{X}_R\bar{Y}_R\right)^2 \\
&= \left(\mathrm{Var}(\bar{X}_R) + \left(\mathbb{E}\bar{X}_R\right)^2\right)\left(\mathrm{Var}(\bar{Y}_R) + \left(\mathbb{E}\bar{Y}_R\right)^2\right) - \left(\mathbb{E}\bar{X}_R\bar{Y}_R\right)^2 \\
&= \mathrm{Var}(\bar{X}_R)\,\mathrm{Var}(\bar{Y}_R) + \mathrm{Var}(\bar{X}_R)\left(\mathbb{E}\bar{Y}_R\right)^2 + \mathrm{Var}(\bar{Y}_R)\left(\mathbb{E}\bar{X}_R\right)^2 \\
&= \frac{1}{R^2}\mathrm{Var}(X)\,\mathrm{Var}(Y) + \frac{1}{R}\mathrm{Var}(X)\left(\mathbb{E}Y\right)^2 + \frac{1}{R}\mathrm{Var}(Y)\left(\mathbb{E}X\right)^2.
\end{aligned}$$

And for the second estimator we use that since the sequence $X_1Y_1, X_2Y_2, \dots$ is i.i.d, we may argue as above to derive that

$$\mathrm{Var}(\mathrm{Est}_2) = \frac{1}{R}\mathrm{Var}(XY) = \frac{1}{R}\left(\mathrm{Var}(X)\,\mathrm{Var}(Y) + \mathrm{Var}(X)\left(\mathbb{E}Y\right)^2 + \mathrm{Var}(Y)\left(\mathbb{E}X\right)^2\right).$$

So whenever $\mathrm{Var}(X)\,\mathrm{Var}(Y) > 0$,

$$\mathrm{Var}(\mathrm{Est}_2) - \mathrm{Var}(\mathrm{Est}_1) = \mathrm{Var}(X)\,\mathrm{Var}(Y)\frac{R-1}{R^2} > 0.$$

3. We have that,

$$\sqrt{R}\left(\begin{bmatrix} \bar{X}_R \\ \bar{Y}_R \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}\right) \to N(0, \Sigma),$$

where $\Sigma = \begin{bmatrix} \mathrm{var}(X) & 0 \\ 0 & \mathrm{var}(Y) \end{bmatrix}$. Then for $h(x, y) = xy$, and after applying the delta method, we get that

$$\sqrt{R}\left(h(\bar{X}, \bar{Y}) - \mu_x\mu_y\right) \to N(0, \tau^2)$$

with $\tau^2 = [\mu_y, \mu_x]\Sigma\begin{bmatrix} \mu_y \\ \mu_x \end{bmatrix} = \mathrm{Var}(X)\mathbb{E}[Y]^2 + \mathrm{Var}(Y)\mathbb{E}[X]^2$. The confidence interval then follows easily.

Exercise 2

Algorithm 1 proposes a sequential Monte Carlo method to compute the expectation $\mathbb{E}[X]$ of a random variable X, where the sample size is doubled at each iteration until the estimated $1 - \alpha$ confidence interval based on a central limit theorem approximation is smaller than a prescribed tolerance ϵ. The algorithm then outputs the final sample size $N(\epsilon, \alpha)$, as well as the estimated value $\bar{X}_N$.

Algorithm 1: Sample Variance Based SMC

Input: N_0, distribution λ, accuracy $\epsilon > 0$, confidence $1 - \alpha > 0$.

Output: $\overline{X}_{\epsilon,\alpha}$ (i.e, approximation of $\mathbb{E}[X]$ with $X \sim \lambda$), N.

Set $k = 0$, generate N_k i.i.d. replica $\{X_i\}_{i=1}^{N_k}$ of $X \sim \lambda$ and

$$\bar{X}_{N_k} = \frac{1}{N_k} \sum_{i=1}^{N_k} X_i, \tag{1}$$

$$\overline{\sigma}^2_{N_k} := \frac{1}{N_k - 1} \sum_{i=1}^{N_k} (X_i - \overline{X}_{N_k})^2. \tag{2}$$

while $\bar{\sigma}_{N_k} C_{1-\alpha/2}/\sqrt{N_k} > \epsilon$ **do**

Set $k = k + 1$ and $N_k = 2N_{k-1}$.

Generate a new batch of N_k i.i.d. replicas $\{X_i\}_{i=1}^{N_k}$ of $X \sim \lambda$.

Compute the sample variance $\overline{\sigma}^2_{N_k}$ by (2).

end while

Set $N = N_k$, generate i.i.d. samples $\{X_i\}_{i=1}^{N}$ of λ and compute the output sample mean $\overline{X}_{\epsilon,\alpha}$.

Algorithm 1 can be particularly sensitive to the choice of initial sample size N_0, and as such, we would like to assess the robustness of such an algorithm in estimating $\mathbb{E}[X]$ for different distributions of X. For some values of N_0 ranging between 10 and 50, consider $\alpha = 10^{-1.5}$ and $\epsilon = 1/10$, and the following random variables:

1. $X \sim \text{Pareto}(x_m = 1, \gamma = 3.1)$ (i.e. with PDF $p(y) = \mathbb{1}_{y > x_m} x_m^\gamma \gamma y^{-(\gamma+1)}$), $\mathbb{E}[X] = \frac{\gamma x_m}{\gamma - 1}$.

2. $X \sim \text{Lognormal}(\mu = 0, \sigma = 1)$, $\mathbb{E}[X] = \exp\left(\mu + \frac{\sigma^2}{2}\right)$.

3. $X \sim U([-1, 1])$, $\mathbb{E}[X] = 0$.

Repeat the simulation $K = 20\alpha^{-1}$ times and record the sample sizes $\{N^{(i)}\}_{i=1}^K$ as well as the computed sample means $\{\bar{X}^{(i)}_{\epsilon,\alpha}\}_{i=1}^K$ returned by the algorithm for each run $i = 1, ..., K$. Estimate the probability of failure $\overline{p}$ of the algorithm :

$$\overline{p}_K(N_0, \epsilon, \alpha) = \frac{1}{K} \sum_{i=1}^K \mathbb{1}_{|\bar{X}^{(i)}_{\epsilon,\alpha} - \mathbb{E}[X]| > \epsilon}.$$

Then check whether $\overline{p}_K(N_0, \epsilon, \alpha) \leq \alpha$ holds. Repeat your experiment for different values of ϵ and α. Discuss your results. **Hint:** You may generate $\text{Pareto}(x_m, \alpha)$ r.v. by inversion.

Then, compare Algorithm 1 with the sequential Monte Carlo method in Algorithm 2, where one realization is added at a time.

Solution

A possible `Python` implementation for this part of the exercise is given below. Notice that since $\mathbb{E}[\bar{p}_N] \sim \alpha$, $\text{Var}[\bar{p}_N] \sim \alpha(1-\alpha)/N \sim \alpha/N$, the choice $N = 20\alpha^{-1}$ gives a 95% confidence

Algorithm 2: One-at-a-time Sample Variance Based SMC

Input: N_0, distribution λ, accuracy $\epsilon > 0$, confidence $1 - \alpha > 0$.

Output: $\overline{X}_{\epsilon,\alpha}$ (i.e, approximation of $\mathbb{E}[X]$ with $X \sim \lambda$), N.

Set $k = 0$, generate N_k i.i.d. samples $\{X_i\}_{i=1}^{N_k}$ of λ and compute the sample variance

$$\overline{\sigma}^2_{N_k} := \frac{1}{N_k - 1} \sum_{i=1}^{N_k} (X_i - \overline{X}_{N_k})^2. \tag{3}$$

while $\bar{\sigma}_{N_k} C_{1-\alpha/2}/\sqrt{N_k} > \epsilon$ **do**

Set $k = k + 1$ and $N_k = N_{k-1} + 1$.

Generate a new i.i.d. sample $X^{(N_k+1)}$ of λ.

Compute

$$\bar{\mu}_{N_k+1} = \frac{N_k}{N_k + 1}\bar{\mu} + \frac{1}{N_k + 1} X^{(N_k+1)} \tag{4}$$

$$\bar{\sigma}^2_{N_k+1} = \frac{N_k - 1}{N_k}\sigma^2_{N_k} + \frac{1}{N_k + 1}(X^{(N_k+1)} - \bar{\mu}_{N_k})^2 \tag{5}$$

end while

Set $N = N_k$, generate i.i.d. samples $\{X_i\}_{i=1}^{N}$ of λ and compute the output sample mean $\overline{X}_{\epsilon,\alpha}$.

interval $\alpha \pm \frac{\sqrt{\alpha}}{\sqrt{N}} C_{1-\alpha/2} \sim \alpha \left(1 \pm \frac{C_{1-\alpha/2}}{\sqrt{20}}\right)$. The results of the numerical tests are plotted in Figure 1 for Algorithm 1 and in Figure 2 for Algorithm 2. For the estimate of $\overline{p}_N$ we have included errorbars of magnitude $\sigma_N := 2\sqrt{\overline{p}_N(1 - \overline{p}_N)/N}$ for all three distributions. Random variables from the Pareto distribution are sampled using inversion sampling with $F^{-1}(x) = (1-x)^{-1/3.1}$. For Algorithm 1, we observe that only the heavy-tailed Pareto$(x_m = 1, \alpha = 3.1)$ distribution fails to approximate its mean with the sought confidence as $p_N - 2\sigma_N > \delta$ for all N_0-values (and the failure is more pronounced for lower N_0-values), while for the other two distributions, one always has $p_N + 2\sigma_N < \delta$. When using Algorithm 2, the Pareto distribution once again fails to approximate the mean for all N_0 but the failure rate of the algorithm is higher for a given N_0 than Algorithm 1. We also observe that the algorithm fails to approximate the mean for all N_0 for the other distributions as well.

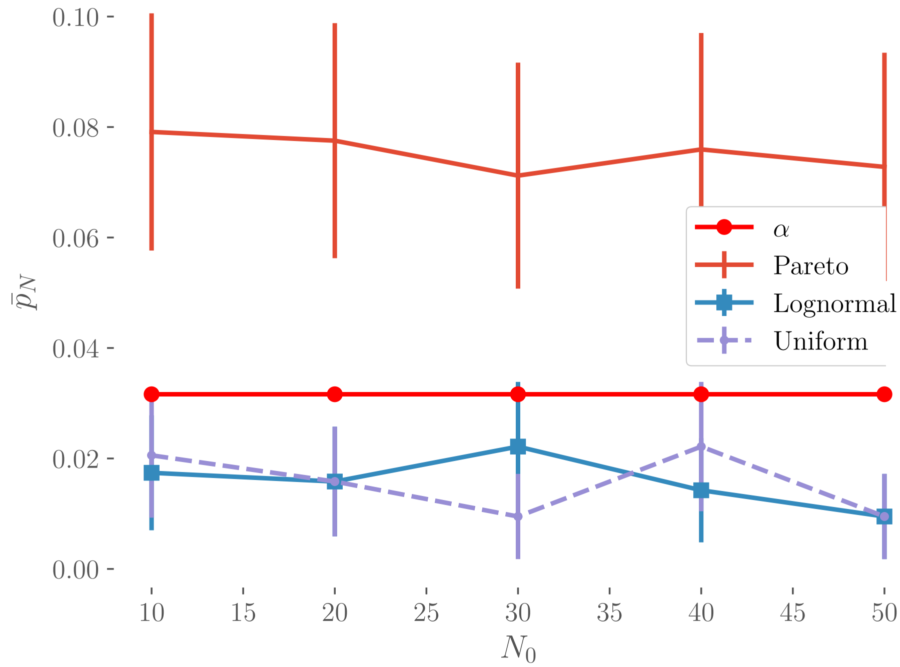

Figure 1: Numerical estimates of the probability of failure for (the SMC) Algorithm 1 when varying the initial number of samples N_0 along with error bars

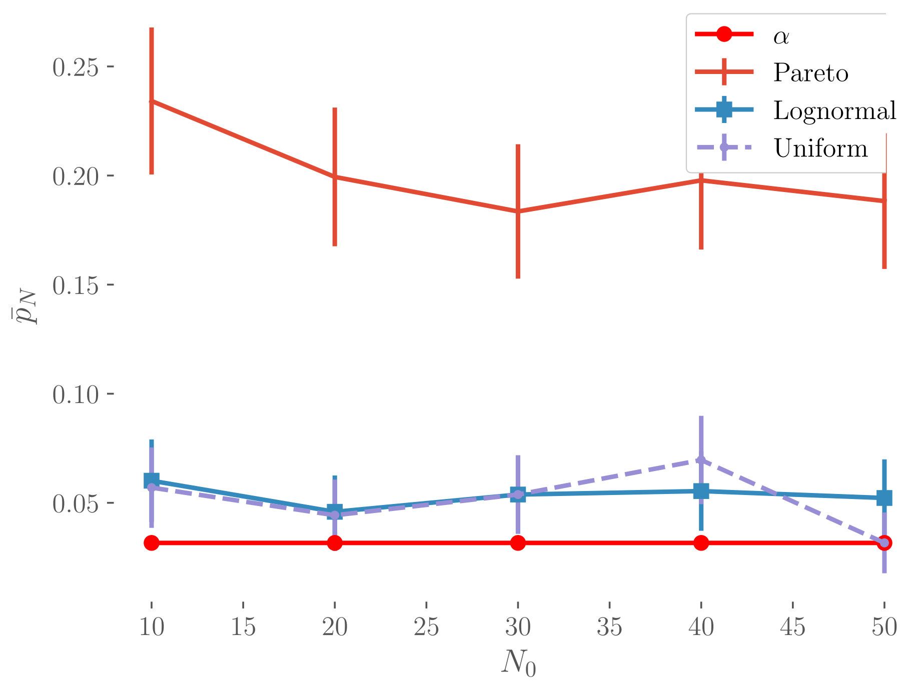

Figure 2: Numerical estimates of the probability of failure for (the SMC) Algorithm 2 when varying the initial number of samples N_0 along with error bars

Python code:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
from matplotlib import rc
#graphical parameters
################################################################################
plt.style.use('ggplot')
rc('font',**{'family':'serif','serif':['Computer Modern Roman'],
     'size' : '12'})
rc('text', usetex=True)
rc('lines', linewidth=2)
plt.rcParams['axes.facecolor']='w'
################################################################################

def Algo1(N0, lamda, alpha, eps):
    M = N0
    X = lamda(M)
    sig = np.std(X)

    while 2*(1.-st.norm.cdf(np.sqrt(M)*eps / sig)) > alpha:

            M = 2 * M
            X = lamda(M)
            sig = np.std(X)

    barX = np.mean(lamda(M))

    return barX, M

def Algo2(N0, lamda, alpha, eps):
    M = N0
    X = lamda(M)
    barX = np.mean(X)
    sig = np.std(X)

    while 2*(1.-st.norm.cdf(np.sqrt(M)*eps / sig)) > alpha:
        Xnew = lamda(1)
        barX = M*barX/(M+1) + 1*Xnew/(M+1)
        sig = np.sqrt(  (M-1)*sig**2/M + (Xnew-barX)**2/(M+1)  )
        M = M +1

    barX = np.mean(lamda(M))
    return barX, M

# Variable introduction
alpha = 10 ** (-1.5)
epsilon = 1./10
N0 = np.arange(10, 51, 10)

pNPareto = np.zeros(len(N0))
randPareto = lambda n: (1 - st.uniform.rvs(size = (n,))) ** (-1/ 3.1)
meanPareto = 3.1/2.1

pNLognormal = np.zeros(len(N0))
randLognorm = lambda n: np.exp(st.norm.rvs(size = (n,) ))
meanLognormal = np.exp(1./2)
```

```
pNUniform = np.zeros(len(N0))
randUni = lambda n: (2 * st.uniform.rvs(size = (n,)) - 1.)
meanUniform = 0

# Number of times to run each simulation for a given N0
N = int(20/alpha)

# Algorithm 1
for i in range(len(N0)):
    for j in range(N):
        print('Running simulation ',j,'/',N, 'for N0 = ', N0[i], ' for Algorithm 1')
        barX,M = Algo1(N0[i], randPareto, alpha, epsilon)
        pNPareto[i] = pNPareto[i] + float(np.abs(barX - meanPareto) > epsilon) / N

        barX,M = Algo1(N0[i], randLognorm, alpha, epsilon)
        pNLognormal[i] = pNLognormal[i] + float(np.abs(barX - meanLognormal) > epsilon) / N

        barX,M = Algo1(N0[i], randUni, alpha, epsilon)
        pNUniform[i] = pNUniform[i] + float(np.abs(barX - meanUniform) > epsilon) / N

plt.figure()
eBars = 2 * np.sqrt(pNPareto * (1 - pNPareto) / N)
plt.errorbar(N0, pNPareto, yerr = eBars, label = 'Pareto')

eBars = 2 * np.sqrt(pNLognormal * (1 - pNLognormal) / N)
plt.errorbar(N0, pNLognormal, yerr = eBars, fmt = '-s', label = 'Lognormal')

eBars = 2 * np.sqrt(pNUniform * (1 - pNUniform) / N)
plt.errorbar(N0, pNUniform, eBars, fmt = '--.', label = 'Uniform')

plt.plot(N0, alpha * np.ones(len(N0)), 'r-o', label = r'$\alpha$')
plt.xlabel(r'$N_0$')
plt.ylabel(r'$\bar{p}_N$')
plt.legend()
plt.savefig('../figures/Ex3.eps',format='eps', bbox_inches='tight')

# Algorithm 2
for i in range(len(N0)):
    for j in range(N):
        print('Running simulation ',j,'/',N, 'for N0 = ', N0[i], ' for Algorithm 2')
        barX,M = Algo2(N0[i], randPareto, alpha, epsilon)
        pNPareto[i] = pNPareto[i] + float(np.abs(barX - meanPareto) > epsilon) / N

        barX,M = Algo2(N0[i], randLognorm, alpha, epsilon)
        pNLognormal[i] = pNLognormal[i] + float(np.abs(barX - meanLognormal) > epsilon) / N

        barX,M = Algo2(N0[i], randUni, alpha, epsilon)
        pNUniform[i] = pNUniform[i] + float(np.abs(barX - meanUniform) > epsilon) / N

plt.figure()
eBars = 2 * np.sqrt(pNPareto * (1 - pNPareto) / N)
plt.errorbar(N0, pNPareto, yerr = eBars, label = 'Pareto')

eBars = 2 * np.sqrt(pNLognormal * (1 - pNLognormal) / N)
plt.errorbar(N0, pNLognormal, yerr = eBars, fmt = '-s', label = 'Lognormal')
```

```
eBars = 2 * np.sqrt(pNUniform * (1 - pNUniform) / N)
plt.errorbar(N0, pNUniform, eBars, fmt = '--.', label = 'Uniform')

plt.plot(N0, alpha * np.ones(len(N0)), 'r-o', label = r'$\alpha$')
plt.xlabel(r'$N_0$')
plt.ylabel(r'$\bar{p}_N$')
plt.legend()
plt.savefig('../figures/Ex4.eps',format='eps', bbox_inches='tight')
plt.show()
```

Exercise 3

Consider the problem of pricing a Barrier option with maturity $T > 0$ based on the stock price S, which is given as the solution to the stochastic differential equation

$$dS = rS\,dt + \sigma S\,dW\ , \quad S(0) = S_0\ ,$$

where W denotes a standard one-dimensional Wiener process. One can show that $S_t = S_0 e^{X_t}$, where $X_t = (r - \sigma^2/2)t + \sigma W_t$ with W being a standard Wiener process. It follows that S_t has a log-normal distribution for any $t > 0$. For $m \in \mathbb{N}$, let $t_i = i\Delta t$ with $\Delta t = T/m$ denote the discrete observation times of the stock price S (e.g. daily at market closure). The payoff of a call option subject to a lower barrier is then given by

$$\Psi(S_{t_0}, S_{t_1}, \ldots, S_T) = (S_T - K)_+ \mathbb{I}_{\left\{B \le \min_{i=0,\ldots,m}(S_{t_i})\right\}}\ ,$$

where $B < S_0$ denotes the Barrier and $K \le S_0$ the strike price. Here, $z_+ = (|z| + z)/2$ denotes the positive part of z. Estimate the expected payoff $\mathbb{E}\big(\Psi(S_{t_0}, S_{t_1}, \ldots, S_T)\big)$ with antithetic variables, using the process parameters $m = 1000$, $r = 0.5$, $\sigma = 0.3$, $T = 2$, $S_0 = 5$, and $K = 10$. Specifically, investigate the variance reduction effect for different barrier values B.

Solution

Instead of using Ψ as given in the exercise, we write is as a function of m increments of the m independent normally distributed random variables:

$$\Psi(S_{t_0}, \ldots, S_{t_m}) = \tilde{\Psi}(X_1, X_2, \ldots, X_m)\ .$$

This is of course possible, since S_0 is deterministic and because the trajectory of the process S observed at discrete times $t_1, \ldots, t_m$ is a function of independent increments $W_{t_i} - W_{t_{i-1}}$ of the Wiener process, each following a $\mathcal{N}(0, t_i - t_{i-1})$ distribution. Then we can apply the basic algorithm for antithetic variables as described in the lecture notes, viz.

$$\hat{\mu}_{\mathrm{AV}} = \frac{1}{N/2}\sum_{i=1}^{N/2} \frac{\Psi(\boldsymbol{X}^{(i)}) + \Psi(-\boldsymbol{X}^{(i)})}{2}\ ,$$

where each $\boldsymbol{X}^{(i)} \sim \mathcal{N}(\mathbf{0}, I)$. An exemplary `Python` implementation of the function $\tilde{\Psi}$ is the following:

```python
import numpy as np
#-----------------------------------------
#Defines the gen psi fucntion
def genPsi(xi,t,r,sigma,K,B,S0):
    dt=np.diff(t)
    W=np.append([0],np.cumsum(np.sqrt(dt)*xi))
    S=S0*np.exp((r-sigma**2/2)*t+sigma*W);
    rval = (abs(S[-1]-K)+ (S[-1]-K))/2*(B<=min(S));
    return rval
#-----------------------------------------
```

Using this implementation with $B = 4$, $N = 100000$, and 95% confidence interval, one obtains for example:

```
B =   4
MC confidence interval  (alpha= 0.05  N=  100000)  mean=  4.176  +-  0.033
N*variance MC estimator:  28.89512978810773
AV confidence interval  (alpha=0.05  N=    50000)  mean=  4.164  +-  0.022
N*variance AV estimator:  12.937820919730397
```

Python code

```
# Exercise 2
import numpy as np
import scipy.stats as st

m = 1000
r = 0.5
sigma = 0.3
T = 2
S0 = 5
K = 2*S0
Blist = np.arange(0,S0)
nB = np.size(Blist)
dt = T/m
t=np.arange(0,m)*dt
N = np.int(1E5)
M = np.int(N/2)
#Defines for the confidence intervals
alpha=0.05

cval = st.norm.ppf(1-alpha/2);

#Defines the gen psi fucntion
def genPsi(xi,t,r,sigma,K,B,S0):
    dt=np.diff(t)
    W=np.append([0],np.cumsum(np.sqrt(dt)*xi))
    S=S0*np.exp((r-sigma**2/2)*t+sigma*W);
    rval = (abs(S[-1]-K)+ (S[-1]-K))/2*(B<=min(S));
    return rval

#Begins the for loop
for j in range(nB):
    B=Blist[j]
    print('B =  ',B)
    psi = np.zeros(N)
    for i in range(N):
        xi = np.random.standard_normal(m-1)
        psi[i] = genPsi(xi,t,r,sigma,K,B,S0)
    print("MC confidence interval (alpha= ",alpha, ' N= ',N,') mean= ',
          np.mean(psi), " +- ", cval*np.std(psi)/np.sqrt(N))
    NvarMC = np.var(psi)
    print("N*variance MC estimator: " ,NvarMC)
    psiAV = np.zeros(M)
    for i in range(M):
        xi = np.random.standard_normal(m-1);
        psiAV[i] = (genPsi(xi,t,r,sigma,K,B,S0) + genPsi(-xi,t,r,sigma,K,B,S0))/2
```

```
print("AV confidence interval (alpha= ",alpha, ' N/2= ',M,') mean= ',
      np.mean(psiAV), " +- ", cval*np.std(psiAV)/np.sqrt(M))
NvarAV = 2*np.var(psiAV)
print('N*variance AV estimator: ',NvarAV)
```