Stochastic Simulation
Professor: Prof. Fabio Nobile
Assistant: Eliott Van Dieren

Week of 2025-10-27 to 2025-10-31

Lab 7 of Thursday 30th October 2025

Exercice 1.

Suppose we are given a control variate Y with known mean $\mathbb{E}(Y)$ and consider the usual modified random variable

$$\tilde{Z}_\alpha = Z + \alpha(Y - \mathbb{E}(Y)) \,,$$

from which we aim at estimating $\mu = \mathbb{E}(Z)$. In fact, here we consider the following *one-shot algorithm* for estimating μ:

- Generate N i.i.d. replicas $(Z^{(i)}, Y^{(i)})$, $i = 1, \dots, N$.
- Estimate α_{opt} by $\hat{\alpha}_{\text{opt}} := -\hat{\sigma}^2_{Z,Y}/\hat{\sigma}^2_Y$, using the usual unbiased mean, variance, and covariance estimators based on the sample $\big(Z^{(i)}, Y^{(i)}\big)_{i=1,\dots,N}$.
- Compute the control variate estimator of μ as

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N}\Big(Z^{(i)} + \hat{\alpha}_{\text{opt}}\big(Y^{(i)} - \mathbb{E}(Y)\big)\Big) \,.$$

1) Show that the estimator $\hat{\mu}$ is asymptotically normally distributed, in the sense that

$$\sqrt{N}\frac{\hat{\mu} - \mu}{\sigma_{\text{opt}}} \underset{N\to\infty}{\Rightarrow} \mathcal{N}(0,1) \,, \quad \text{where} \quad \sigma_{\text{opt}} = \sqrt{\text{Var}(\tilde{Z}_{\alpha_{\text{opt}}})} \,.$$

Furthermore, explain why the asymptotic normality also holds when σ_{opt} is replaced by the usual empirical standard deviation based on a sample of realizations of $\tilde{Z}_{\alpha_{\text{opt}}}$.

Hint. Consider re-writing the estimator as the summation of the control variate estimator computed with the exact α_{opt} and a correction term involving $\hat{\alpha}_{\text{opt}} - \alpha_{\text{opt}}$ as follows:

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N}\Big(Z^{(i)} + \alpha_{\text{opt}}\big(Y^{(i)} - \mathbb{E}(Y)\big)\Big) + (\hat{\alpha}_{\text{opt}} - \alpha_{\text{opt}})\Big(\frac{1}{N}\sum_{i=1}^{N} Y^{(i)} - \mathbb{E}(Y)\Big).$$

Then, recall Slutsky's theorem, which states that if ξ_n converges in distribution to ξ and η_n converges in probability to a constant c, then $f(\xi_n, \eta_n) \underset{n\to\infty}{\Rightarrow} f(\xi, c)$ for any continuous function $f\colon \mathbb{R}\times\mathbb{R}\to\mathbb{R}$. Here, the symbol $\Rightarrow$ denotes convergence in distribution.

2) Implement the one-shot algorithm described above. Apply it to the examples considered in Lab 06, Exercise 1. That is, approximate the probability $p = \mathbb{P}(\boldsymbol{X} \in A)$ for the sets $A = \big\{\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2 \colon x_i \geqslant a,\ i = 1, 2\big\}$ with $a = 1, 3, 10$. Here, $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{0}, \Sigma)$ with $\Sigma = \left(\begin{smallmatrix} 4 & -1 \\ -1 & 4 \end{smallmatrix}\right)$.

a) First, explain why $Y = \mathbb{I}_{\{X_1+X_2 \geqslant 2a\}}$ for $\boldsymbol{X} = (X_1, X_2)$ could be a decent control variate for this problem.

b) Then perform simulations and investigate the variance reduction effect for the control variate Y. Moreover, use the result proved in point 1 to compute asymptotic 95% confidence intervals.

c) Can you think of other appropriate control variates?

Solution

1) Because $\hat{\alpha}_{\text{opt}} \equiv \hat{\alpha}_{\text{opt}}(N)$ depends on the samples $(Z^{(i)}, Y^{(i)})_{i=1,\dots,N}$, the control variate estimator $\hat{\mu}$ is not unbiased (notice that it is unbiased for any α deterministic though!), but only asymptotically unbiased. Consequently, the CLT (i.e. the asymptotic normality) for $\hat{\mu}$ is not immediate, although not very difficult either. In fact, first note that $\hat{\mu}$ can be written as

$$\begin{aligned}\hat{\mu} &= \frac{1}{N}\sum_{i=1}^{N}\Big(Z^{(i)} + \alpha_{\text{opt}}(Y^{(i)} - \mathbb{E}(Y))\Big) + (\hat{\alpha}_{\text{opt}} - \alpha_{\text{opt}})\Big(\frac{1}{N}\sum_{i=1}^{N} Y^{(i)} - \mathbb{E}(Y)\Big) \\ &= E_N(\tilde{Z}_{\alpha_{\text{opt}}}) + (\hat{\alpha}_{\text{opt}} - \alpha_{\text{opt}})\Big(E_N(Y) - \mathbb{E}(Y)\Big)\,,\end{aligned}$$

where $E_N(\xi) := \frac{1}{N}\sum_{i=1}^{N}\xi^{(i)}$ denotes the sample average of ξ. Since $\mathbb{E}(\tilde{Z}_\alpha) = \mu$ for any deterministic α, we find that

$$\begin{aligned}\sqrt{N}\,\frac{\hat{\mu}-\mu}{\sigma_{\text{opt}}} &= \sqrt{N}\,\frac{E_N(\tilde{Z}_{\alpha_{\text{opt}}}) - \mathbb{E}(\tilde{Z}_{\alpha_{\text{opt}}})}{\sqrt{\text{Var}(\tilde{Z}_{\alpha_{\text{opt}}})}} + \sqrt{N}\,\frac{E_N(Y) - \mathbb{E}(Y)}{\sqrt{\text{Var}(Y)}}\,\sqrt{\frac{\text{Var}(Y)}{\text{Var}(\tilde{Z}_{\alpha_{\text{opt}}})}}(\hat{\alpha}_{\text{opt}} - \alpha_{\text{opt}}) \\ &= \xi_N + C\,\zeta_N\eta_N\,,\end{aligned}$$

for $C = \sqrt{\text{Var}(Y)/\,\text{Var}(\tilde{Z}_{\alpha_{\text{opt}}})}$ and

$$\xi_N := \sqrt{N}\,\frac{E_N(\tilde{Z}_{\alpha_{\text{opt}}}) - \mathbb{E}(\tilde{Z}_{\alpha_{\text{opt}}})}{\sqrt{\text{Var}(\tilde{Z}_{\alpha_{\text{opt}}})}}\,, \quad \eta_N := \sqrt{N}\,\frac{E_N(Y) - \mathbb{E}(Y)}{\sqrt{\text{Var}(Y)}}\,, \quad \zeta_N := \hat{\alpha}_{\text{opt}}(N) - \alpha_{\text{opt}}\,.$$

The individual limits for these three random variables are

$$\xi_N \Rightarrow \mathcal{N}(0,1)\,, \quad \eta_N \Rightarrow \mathcal{N}(0,1)\,, \quad \zeta_N \to 0 \text{ in prob.}$$

It follows from Slutsky's theorem that $\zeta_N\eta_N \Rightarrow 0$. Since the product converges to a constant, it follows that $\zeta_N\eta_N \to 0$ in probability. The CLT for $\hat{\mu}$ then follows in view of using Slutsky's theorem again.

2) Implementation is identical to a Monte Carlo method for the random variable $\tilde{Z}_{\alpha_{\text{opt}}}$, once $\hat{\alpha}_{\text{opt}}$ has been estimated.

a) The control variate $Y = \mathbb{I}_{\{X_1+X_2\geqslant 2a\}}$ is motivated by the observation that $\min(X_1, X_2) \leqslant \frac{X_1+X_2}{2}$ and the fact that the distribution of the random variable $\xi = X_1 + X_2$ is known so that $\mathbb{E}(Y)$ is also given. In fact, $\xi = B\boldsymbol{X}$ for $B = (1,1)$, so that $\xi \sim \mathcal{N}(0, B\Sigma B^T) = \mathcal{N}(0,6)$. It follows that $\mathbb{E}(Y) = 1 - \Phi(2a/\sqrt{6})$, where Φ is the CDF of the $\mathcal{N}(0,1)$ distribution.

b) We observe a variance reduction for values of a that are not too big. For a rather big, the control variate method is not as effective as importance sampling for this problem. This is due to the fact that also $\mathbb{E}(Y) = 1 - \Phi(2a/\sqrt{6})$ becomes very small for increasing values of a. Running the attached code for different values of a, gives the following results

```
--------------------------------------------------
a= 1
Crude Monte Carlo mean 0.064955 +- 0.000483
control Variate mean   0.064891 +- 0.000414
SEmc/SEcv              1.1666666666666667
--------------------------------------------------
a= 2
Crude Monte Carlo mean 0.012451 +- 0.000217
control Variate mean   0.012388 +- 0.00019
SEmc/SEcv              1.1421052631578945
--------------------------------------------------
a= 3
Crude Monte Carlo mean 0.001429 +- 7.4e-05
control Variate mean   0.001405 +- 6.6e-05
SEmc/SEcv              1.121212121212121
--------------------------------------------------
a= 4
Crude Monte Carlo mean 8.4e-05 +- 1.8e-05
control Variate mean   8.2e-05 +- 1.7e-05
SEmc/SEcv              1.0588235294117647
--------------------------------------------------
a= 5
Crude Monte Carlo mean 2e-06 +- 3e-06
control Variate mean   3e-06 +- 3e-06
SEmc/SEcv              1.0
--------------------------------------------------
a= 6
Crude Monte Carlo mean 0.0 +- 0.0
control Variate mean   nan +- nan
SEmc/SEcv              nan
--------------------------------------------------
a= 7
Crude Monte Carlo mean 0.0 +- 0.0
control Variate mean   nan +- nan
SEmc/SEcv              nan
```

c) Another simple control variate could, for example, be $Y_j = \mathbb{I}_{\{X_j \geqslant a\}}$. Of course, this one does not perform as well as Y.

Python code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Created on Thu Nov  7 19:38:47 2019

@author: jmadriga
"""
#imports libraries
import numpy as np
from scipy.stats import norm

np.random.seed(42) #sets seed for reproducibility

#Matrix
sigma=np.array([[4,-1],[-1,4]])
A=np.linalg.cholesky(sigma)
#number of samples
N=int(1E6)
C_alpha=norm.ppf(1-0.05/2)

# Defines function and the cv function
psi = lambda x,a: 1.0*np.prod(x>a,1)
psi_cv= lambda x,a: 1.0*(np.sum(x,1)>2*a)

#samples iid from N
X=A@np.random.standard_normal((2,N))

#test values for a

avals=np.arange(1,8)

for i in range(len(avals)):
    #samples iid from N
    X=(A@np.random.standard_normal((2,N))).T #the T is a transpose, so
    #that it is a matrix of Nx2 samples
    a=avals[i]
    #creates Z and its cv Y
    Z=psi(X,a)
    Y=psi_cv(X,a)
    #obtaines the sample covariance matrix
    C=np.cov(Z,Y)
    EY=1-norm.cdf(2*a/np.sqrt(6)) #exact mean of Y
    alpha_hat=-C[0,1]/C[1,1]
    rho2=np.corrcoef(Z,Y)[0,1]  #computes the correlation coeff
    mu_cmc=round(np.mean(Z),6)
    Z_tilde=Z+alpha_hat*(Y-EY) #Writes doum \tilde Z , which is the correlated Var
    mu_cv=round(np.mean(Z_tilde),6) #computes the mean

    #Computes the standar errors of Z and \tilde Z.
    SE_mc=round(C_alpha*(np.var(Z)/N)**0.5,6)
    SE_cv=round(C_alpha*(np.var(Z_tilde)/N)**0.5,6)

    print('-----------------------------------------------------')
    print('a= '+str(a))
    print('Crude Monte Carlo mean '+str(mu_cmc)+ ' +- '+str(SE_mc))
    print('control Variate mean   '+str(mu_cv)+  ' +- '+str(SE_cv))
    print('SEmc/SEcv              '+str(SE_mc/SE_cv))
```

Exercice 2.

Suppose we wish to construct a Brownian motion path $\{B_t \colon B_0 = 0,\, 0 \leqslant t \leqslant T\}$ that finishes at $T = 1$ in S distinct strata. To stratify standard Brownian motion on its endpoint, one first generates the S values at $T = 1$ and then samples the Brownian paths on the interval $[0, T]$ conditional upon these stratified terminal values.

1) Implement an algorithm that generates stratified standard Brownian motion using S equiprobable strata. Specifically, for each stratum Ω_j, $j = 1, \dots, S$, your algorithm should produce N_j stratified Brownian samples paths $B_{t_m}^{(i,j)}$, $i = 1, \dots, N_j$, evaluated in the discrete times $t_m = m/M$ with $m = 1, \dots, M \in \mathbb{N}$. Test your implementation for $S = 12$, $M = 1000$, and $N_j = 2$ by plotting the stratified samples paths. *Hint: Brownian bridge sampling.*

2) Consider the geometric Brownian motion process X_t that solves

$$dX = rX\,dt + \sigma X\,dW\,, \quad X(0) = X_0\,,$$

and which is given by $X_t = X_0 e^{Y_t}$, where $Y_t = (r - \sigma^2/2)t + \sigma W_t$ with W being a standard Wiener process. For $M \in \mathbb{N}$, let

$$\Psi(X_{t_0}, \dots, X_{t_M}) = \max_{0 \leqslant m \leqslant M} X_{t_m} - \min_{0 \leqslant m \leqslant M} X_{t_m}\,,$$

where $t_m = m/M$ as before. We want to estimate $\mu = \mathbb{E}[\Psi(X_{t_0}, \dots, X_{t_M})]$ for $X_0 = 6$, $r = 0.05$, $\sigma = 0.3$, and $M = 100$. Use your procedure developed in point 1 to estimate μ using stratified sampling with $S = 10$ strata. Moreover, compute the total number of samples N such that the asymptotic 99% confidence interval is smaller than $2\,tol$ for $tol = 10^{-2}, 10^{-3}$. Investigate both proportional and optimal sampling allocation in each strata.

Remark: The function Ψ is related to the value of a look-back option whose payoff is equivalent to buying at the minimum and selling at the maximum price on the time interval $[0, T = 1]$. As given here, Ψ omits the (constant) discount factor e^{-rT} that compensates for waiting until time T to collect the payoff.

3) Repeat the previous point, but now consider only $N_j = 2$ samples per stratum Ω_j, $j = 1, \dots, S$, and investigate the estimator's variance decay as a function of S.

Solution

We will use the Brownian bridge sampler to sample Brownian motion stratified on its terminal value. Let's denote the Brownian bridge sampler by `brownian_bridge(t,a,b,Z)`, which has the vector of Brownian increments `Z` as input.

1) The code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
def brownian_bridge(t,a,b,Z):
        n=np.size(t)-2
        X=np.zeros(n+2,)
        X[0]=a
        X[-1]=b
        tfinal=t[-1]
        #main loop
        for k in range(1,n+1):
                mu = X[k-1] + (b-X[k-1])*(t[k]-t[k-1])/(tfinal-t[k-1]);
                sig2 = (tfinal-t[k])*(t[k]-t[k-1])/(tfinal-t[k-1]);
                X[k] = mu +np.sqrt(sig2)*Z[k-1];
        return X
```

can then be used to produce realizations of the stratified Brownian motion, as shown in Figure 1.

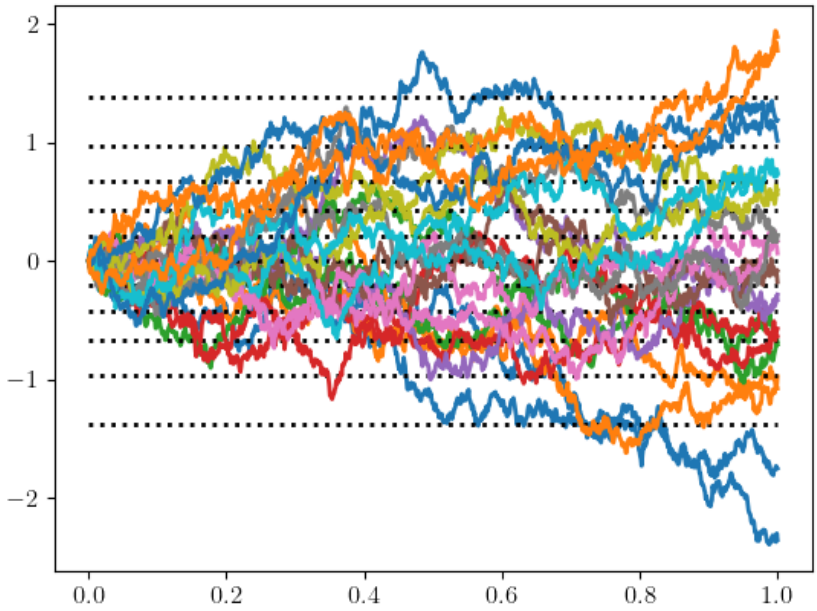

Figure 1: Two realizations per stratum of the stratified Brownian motion

2) Since the geometric Brownian motion is given as an explicit function of Brownian motion, realizations of stratified geometric Brownian motion can be easily generated using the procedure of point 1. In fact, let $B_{t_m}^{(i,j)}$ denote the i-th realization, $i = 1, \dots, N_j$, of the Brownian motion stratified to the stratum Ω_j, $j = 1, \dots, S$, evaluated in the discrete times $t_m = m/M$. Then the discrete time realizations of the stratified geometric Brownian motion are simply obtained by

$$X_{t_m}^{(i,j)} = X_0 \exp\left[\left(r - \frac{\sigma^2}{2}\right) t_m + \sigma B_{t_m}^{(i,j)}\right].$$

Compared to Monte Carlo, stratification yields a significant variance reduction. For example, for $tol = 10^{-2}$, the stratification estimator with optimal sample allocation requires $N_{\text{str}} \approx 30000$ for an 99% confidence interval, while the Monte Carlo method needs $N_{\text{mc}} \approx 75000$. The mean and the variance can be seen in Table 1.

Method	Mean	Variance
Monte Carlo	2.75	1.13
Statification (proportional)	2.79	0.50
Stratification (optimal)	2.79	0.44

Table 1: Summary of results for stratification with optimal allocation.

3) If we keep $N_j = 2$ fixed and increase S, effectively, we are increasing the sample size $N = 2S$. Figure 2 shows the variance of the estimator as a function of S. We see that it dereases as

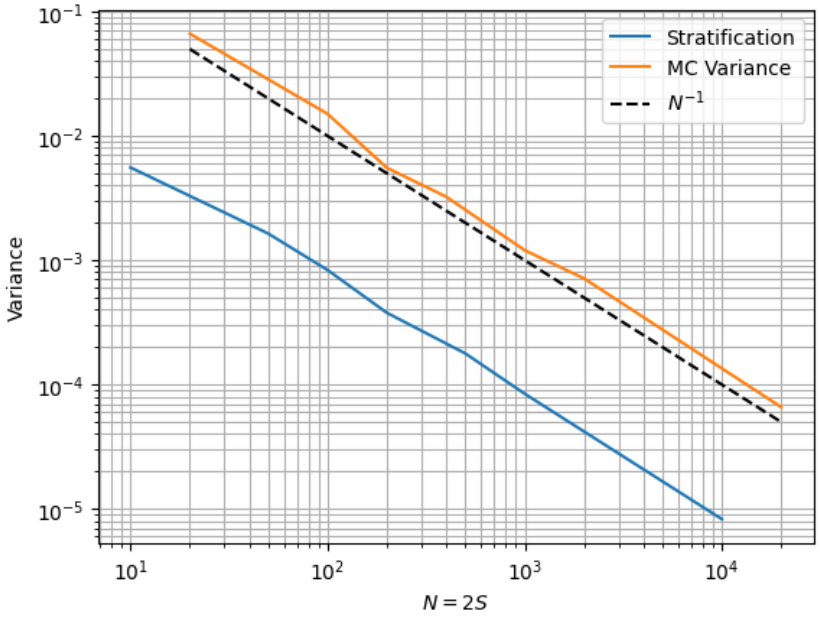

Figure 2: Decay of variance as a function of samples N

$S^{-1} \sim N^{-1}$, as expected for a Monte Carlo estimator. Implementing the code (below) gives the results shown in Figure 2.

Python code for points 1:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from matplotlib import rc
from numpy import matlib
########################################################################
rc('font',**{'family':'serif','serif':['Computer Modern Roman'],
     'size' : '12'})
rc('text', usetex=True)
rc('lines', linewidth=2)
plt.rcParams['axes.facecolor']='w'
import matplotlib
latex_preamble = r'\usepackage{amsmath} \usepackage{amssymb}'
matplotlib.rcParams.update({
    'text.usetex': True,
    'text.latex.preamble': latex_preamble
})

########################################################################
def brownian_bridge(t,a,b,Z):
    n=np.size(t)-2
    X=np.zeros(n+2)
    X[0]=a
    X[-1]=b
    tfinal=t[-1]
    #main loop
```

```
    for k in range(n):
        mu = X[k] + (b-X[k])*(t[k+1]-t[k])/(tfinal-t[k])
        sig2 = (tfinal-t[k+1])*(t[k+1]-t[k])/(tfinal-t[k])
        X[k+1] = mu +np.sqrt(sig2)*Z[k]
    return X

S = 12 # number of strata
N = [2]*S # list of no of samples per stratum
M = int(1e3) # number of points for path construction
T = 1 # terminal time at which we stratify
t = np.linspace(0,T,M)
print('j\tU_lo\tU_up\tX_lo\tX_up')

for j in range(S):
    print('%d'% (j+1),'\t%.2f' % (j/S),'\t%.2f' % ((j+1)/S),'\t%.2f' % norm.ppf(j/S)
     ↪  ,'\t%.2f' % norm.ppf((j+1)/S))
    plt.plot([0,1],[norm.ppf(j/S,0,np.sqrt(T)), norm.ppf(j/S,0,np.sqrt(T))],'k:')
    Nj = N[j]
    for i in range(Nj):
        U=(j+np.random.random(1))/S # sample uniform random number from Strata j
        x_s=norm.ppf(U,0,np.sqrt(T)) # invert it to get the stratified terminal value
        B=brownian_bridge(t,0,x_s,np.random.standard_normal(M))

        plt.plot(t,B,'C'+str(j))

plt.show()
```

Python code for point 2:

```
import numpy as np
import scipy.stats as st

np.random.seed(42)

def brownian_bridge(t,a,b,Z):
    n=np.size(t)-2
    X=np.zeros(n+2)
    X[0]=a
    X[-1]=b
    tfinal=t[-1]
    #main loop
    for k in range(n):
            mu = X[k] + (b-X[k])*(t[k+1]-t[k])/(tfinal-t[k])
            sig2 = (tfinal-t[k])*(t[k+1]-t[k])/(tfinal-t[k])
            X[k+1] = mu + np.sqrt(sig2)*Z[k]
    return X

def genPsi(xi,t,r,sigma,S0):
    dt=np.diff(t)
    W=np.concatenate([[0],np.cumsum(np.sqrt(dt)*xi)])
    S=S0*np.exp((r-0.5*sigma**2)*t+sigma*W)
    return np.max(S)-np.min(S)

#gives some initial values
X0 = 6
r = 0.05
sigma = 0.3
S = 10
M = int(1e2)
T = 1
t = np.linspace(0,T,M)

##-----------------------------------------------------------------------
```

```
#
#   Part 2.1 Monte Carlo
#
#-----------------------------------------------------------------------
# pilot run
p = np.ones(S)/S #probabilties of each strata
si2 = np.zeros(S)

# pilot run for variance
Npilot = int(1e2)
Z = np.zeros(Npilot)
for i in range(Npilot):
    Z[i] = genPsi(st.norm.rvs(size=(M-1)),t,r,sigma,X0)
si2Z = np.var(Z)
print('Monte Carlo Var(Z): ',si2Z)

alpha = 1-0.99
tol = 1e-2
cval = st.norm.ppf(1-alpha/2,0,1)
NN = int(np.ceil( (cval*np.sqrt(si2Z)/tol)**2 ))
print(1,'tol = ',tol)
print(1,'\tN_MC: ',NN)
Z = np.zeros(NN)
for i in range(NN):
    Z[i] = genPsi(st.norm.rvs(size=(M-1)),t,r,sigma,X0)
print('\tMC estimator: ',  np.mean(Z))

##-----------------------------------------------------------------------
#
#   Part 2.2  pilot run for optimal and proportional stratification
#
#-----------------------------------------------------------------------
print('\nPilot run for computing variances' )
print('-----------------------------------------------------------------')
print('j  U_0 U_f    X_0             X_f  ')
# test run for the optimal allocation
muj = []
for j in range(S):
    X = np.zeros((Npilot,M))
    Nj = Npilot
    Zj = np.zeros(Nj)

    for i in range(Nj):
        U=(j+st.uniform.rvs())/S
        x_s=st.norm.ppf(U,0,np.sqrt(T))
        X[i,:]=brownian_bridge(t,0,x_s,st.norm.rvs(size=(M-1)))
        X[i,:]= X0*np.exp( (r-sigma**2/2)*t + sigma*X[i,:] )
        Zj[i] = np.max(X[i,:]) - np.min(X[i,:])
    si2[j] = np.var(Zj)
    print((j+1),j/S,(j+1)/S,
            st.norm.ppf(j/S,0,np.sqrt(T)),st.norm.ppf((j+1)/S,0,np.sqrt(T)))
    muj.append(np.mean(Zj))
print('Screening mean = ', np.dot(muj,p))
sumSiP = np.dot(p,np.sqrt(si2))
sumSi2P = np.dot(p,si2)
print('Var Opt = ', sumSiP**2)
print('Var Prop = ', sumSi2P)

##-----------------------------------------------------------------------
#
#   Part 2.2.1  optimal stratification
#
#-------------------------------------------------------------
```

```
print('\nOptimal stratification:' )
# Let's go
alpha = 1-0.99
tol = 1e-2
cval = st.norm.ppf(1-alpha/2,0,1)
NN = int(np.ceil( (cval*sumSiP/tol)**2 ))
print('tol = ',tol)
print('\tN_Str:',NN)
mu = np.zeros(S)
for j in range(S):
    X = np.zeros((NN,M))
    Nj = int(NN*p[j]*np.sqrt(si2[j])/sumSiP)
    Zj = np.zeros(Nj)
    for i in range(Nj):
        U=(j+st.uniform.rvs())/S
        x_s=st.norm.ppf(U,0,np.sqrt(T))
        X[i,:]=brownian_bridge(t,0,x_s,st.norm.rvs(size=(M-1)))
        X[i,:]= XO*np.exp( (r-sigma**2/2)*t + sigma*X[i,:] )
        Zj[i] = np.max(X[i,:]) - np.min(X[i,:])
    mu[j] = np.mean(Zj)
    print((j+1),Nj)


mu_str = np.dot(p,mu)
print('\tStrat. estimate: ',  mu_str)

##------------------------------------------------------------------------
#
#   Part 2.2.2  stratification  proportional
#
#-------------------------------------------------------------

print('\nProportional stratification:' )
# Let's go
alpha = 1-0.99
tol = 1e-2
cval = st.norm.ppf(1-alpha/2,0,1)
NN = int(np.ceil( (cval/tol)**2*sumSi2P ))
print('tol = ',tol)
print('\tN_Str:',NN)
mu = np.zeros(S)
for j in range(S):
    X = np.zeros((NN,M))
    Nj = int(NN*p[j])
    Zj = np.zeros(Nj)
    for i in range(Nj):
        U=(j+st.uniform.rvs())/S
        x_s=st.norm.ppf(U,0,np.sqrt(T))
        X[i,:]=brownian_bridge(t,0,x_s,st.norm.rvs(size=(M-1)))
        X[i,:]= XO*np.exp( (r-sigma**2/2)*t + sigma*X[i,:] )
        Zj[i] = np.max(X[i,:]) - np.min(X[i,:])
    mu[j] = np.mean(Zj)
    print((j+1),Nj)

mu_str = np.dot(p,mu)
print('\tStrat. estimate: ',  mu_str)
```

Python code for point 3:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt

def genPsi(xi,t,r,sigma,S0):
```

```
    dt=np.diff(t)
    W=np.concatenate([[0],np.cumsum(np.sqrt(dt)*xi)])
    S=S0*np.exp((r-0.5*sigma**2)*t+sigma*W)
    return np.max(S)-np.min(S)

T = 1.
M = 100
Nj = 2

sig = 0.3
r = 0.05
X0 = 6

ii=0
S_all = [10,50,100, 200, 500, 1000, 10000]
VS=np.zeros(len(S_all))
for S in S_all:
    pis = np.linspace(0, 1., S+1)[1:]
    #print pis
    strata = st.lognorm.ppf(pis,s=sig*np.sqrt(T), scale = np.exp(r-sig**2/2.))
    #print strata
    Y1 = r - sig**2/2. + sig * np.sqrt(T) * np.random.normal(size = (1000000,))
    X1 = np.exp(Y1)
    Xj = np.zeros((S, Nj))
    for i in range(S):
            if i > 0:
            y = list(np.where((X1 > strata[i-1])*(X1 < strata[i]) == 1)[0])
            else:
                y = list(np.where((X1 < strata[i]) == 1)[0])
            Xj[i,:] = X1[y][:Nj]
            Yj = np.log(Xj)
    Wj = (Yj - (r - sig**2/2.)) / sig
    Ypaths = np.zeros((M + 1, Wj.shape[0], Wj.shape[1]))
    dt = 1./(M)
    t = np.linspace(0, 1., M+1)
    for i in range(S):
            for j in range(Nj):
            dw = np.sqrt(dt) * np.hstack([0,st.norm.rvs(size = (M,))])
            W = np.cumsum(dw)
            Ypaths[:,i,j] = (r - sig**2/2.)*t + sig * (W - t*(W[-1] - Wj[i,j]))
    Zj = X0*(np.exp(np.amax(Ypaths, axis=0)) - np.exp(np.amin(Ypaths, axis=0)))
    VS[ii]=np.var(Zj,axis = 1).sum() * (pis[1]-pis[0])**2 / 2.
    #print np.mean(Zj, axis = 1).sum() * (pis[1]-pis[0])
    print('NUMBER OF STRATA = ' + str(S))
    print('VARIANCE = ' + str(VS[ii]))
    ii+=1

plt.loglog(S_all,VS,label='Stratification')
plt.xlabel('Strata')
plt.ylabel('Variance')
plt.grid(True,which='both')

##---------------------------------------------------------------------
#
#    Monte Carlo
#
#---------------------------------------------------------------------
T = 1.
M = 100
Nj = 2
sig = 0.3
r = 0.05
X0 = 6
```

```
t = np.linspace(0, 1., M+1)

ii=0
S_all = [2*10,2*50,2*100, 2*200, 2*500, 2*1000, 2*10000]
VMC=np.zeros(len(S_all))
for NN in S_all:
    # pilot run for variance
    Z=np.zeros(NN)
    for i in range(1,NN):
        Z[i-1] = genPsi(np.random.standard_normal(M),t,r,sig,X0);
    VMC[ii]= np.var(Z);
    ii+=1

plt.loglog(S_all,VMC/S_all,label='MC Variance')
plt.xlabel(r'$N=2S$')
plt.ylabel('Variance')
plt.grid(True,which='both')
plt.loglog(S_all,np.asarray(S_all)**-1.0,'k',linestyle='dashed',label=r'$N^{-1}$')
plt.legend()
plt.show()
```

Exercice 3.

Consider the problem of estimating $\mu = \mathbb{E}(Z)$ for $Z = \psi(X)$ and $X \sim U(0,1)$.

1) Show that the *randomized midpoint quadrature* estimator

$$\hat{\mu}_S := \frac{1}{S}\sum_{j=1}^{S} \psi\left(\frac{j-1+U_j}{S}\right),$$

with $U_1, \dots, U_S$ i.i.d. $U(0,1)$, corresponds to a stratified sampling estimator of μ. *Hint: consider uniform strata.*

Optional: explain why $\hat{\mu}_S$ is called *randomized midpoint quadrature* estimator.

2) Suppose that $\psi \in C^1([0,1])$ Show that the estimator $\hat{\mu}_S$, which is a Monte Carlo type estimator, converges with *super-canonical rate* (i.e. faster than $S^{-1/2}$). Specifically, show that

$$\sqrt{\mathbb{E}[(\mu - \hat{\mu}_S)^2]} \leqslant cS^{-3/2},$$

for an appropriate positive constant $c < \infty$ independent of S. Determine also the constant c.

Solution

1) The estimator $\hat{\mu}_S$ is a stratified sampling estimator with

$$\Omega_j = \left(\frac{j-1}{S}, \frac{j}{S}\right), \quad \text{and} \quad N_j = 1,$$

for $j = 1, \dots, S$.

The name is motivated by the close similarity to the midpoint quadrature. That is, consider approximating $\int_0^1 \psi(x)\,dx$ by the midpoint rule using the partition Ω_j:

$$\int_0^1 \psi(x)\,dx = \sum_{j=1}^{S} \int_{\frac{j-1}{S}}^{\frac{j}{S}} \psi(x)\,dx \approx \sum_{j=1}^{S} \psi\left(\frac{j-1}{S} + \frac{1}{2S}\right).$$

Since $\mathbb{E}(U_j) = 1/2$, $\widehat{\mu}_S$ can be viewed as a randomized midpoint rule.

2) We compute the mean squared error of the stratified sampling estimator as

$$\mathbb{E}((\mu - \widehat{\mu}_S)^2) = \mathrm{Var}(\widehat{\mu}_S) = S^{-2} \sum_{j=1}^{S} \mathrm{Var}\left[\psi\left(\frac{j-1}{S} + \frac{U_j}{S}\right)\right].$$

To quantify the variance term, we linearize the function ψ around the point $(j-1)/S$, which yields

$$\mathrm{Var}\left[\psi\left(\frac{j-1}{S} + \frac{U_j}{S}\right)\right] = \mathrm{Var}\left[\psi\left(\frac{j-1}{S}\right) + \psi'(\xi_j)\frac{U_j}{S}\right] = S^{-2}\,\mathrm{Var}[\psi'(\xi_j)U_j],$$

for some ξ_j between $(j-1)/S$ and $(j-1+U_j)/S$. From the hypothesis, it follows that $\mathrm{Var}[\psi'(\xi_j)U_j] \leqslant C^2/3$ for all $j = 1, \ldots, S$, where $C = \sup_{x\in[0,1]}|\psi'(x)|$. Consequently, we find that

$$\mathbb{E}((\mu - \widehat{\mu}_S)^2) \leqslant \frac{C^2}{3} S^{-3},$$

as required.

Exercice 4.

1) Consider the random variable $Z = 4\,\mathbb{I}_{\{U_1^2+U_2^2\leqslant 1\}}$ with $U_1, U_2 \overset{\text{i.i.d.}}{\sim} \mathcal{U}(0,1)$, so that $\mathbb{E}(Z) = \pi$. Consider the control variates $\tilde{Z}_{\alpha,i} = Z - \alpha(Y_i - \mathbb{E}(Y_i))$ where the controls Y_i are given by:

$$Y_1 := \mathbb{I}_{\{U_1+U_2\leqslant 1\}}, \quad Y_2 := \mathbb{I}_{\{U_1+U_2\geqslant\sqrt{2}\}}, \quad \text{and} \quad Y_3 := (U_1+U_2-1)\mathbb{I}_{\{1<U_1+U_2\leqslant\sqrt{2}\}}.$$

Estimate their respective expected variance reduction $\mathbb{V}\mathrm{ar}(\tilde{Z}_{\alpha,i})/\mathbb{V}\mathrm{ar}(Z)$ using $N = 10^6$ simulations.

2) Consider again the random variable $Z = 4\,\mathbb{I}_{\{U_1^2+U_2^2\leqslant 1\}}$ as in point 1. We now wish to use multiple control variates and compare their variance reduction to the single control variate case. Consider the control variate $\tilde{Z}_{\boldsymbol{\alpha}} = Z - \boldsymbol{\alpha}\cdot(\boldsymbol{Y} - \mathbb{E}(\boldsymbol{Y}))$ where $\boldsymbol{\alpha} \in \mathbb{R}^d$ and $\boldsymbol{Y}$ is a d-dimensional control vector. Perform simulations and report the expected variance reduction $\mathbb{V}\mathrm{ar}(\tilde{Z}_{\boldsymbol{\alpha}})/\mathbb{V}\mathrm{ar}(Z)$ for each of the following control vectors:

$$\boldsymbol{Y}^1 := (Y_1, Y_2)^T, \quad \boldsymbol{Y}^2 := (Y_1, Y_3)^T, \quad \boldsymbol{Y}^3 := (Y_2, Y_3)^T, \quad \text{and} \quad \boldsymbol{Y}^4 := (Y_1, Y_2, Y_3)^T.$$

Here, the random variables Y_i, $i = 1, 2, 3$ are as described in point 1.

3) Implement a one-shot control variate algorithm for the control vector with the best variance reduction.

Solution

A possible implementation of this problem is shown below. The predicted variance reductions are compared to the variance of the control variates $\tilde{Z}_\alpha$ computed using the optimal choice of control parameter α for each combination of controls. The console output is shown below. As can be seen from the console output, the best performing case is when all three control variates are used.

```
Predicted reduction Y1 =  0.726474340958511
Actual reduction Y1 =  0.7264743409585116
Predicted reduction Y2 =  0.24086269725677067
Actual reduction Y2 =  0.24086269725677123
Predicted reduction Y3 =  0.9996980809405455
Actual reduction Y3 =  0.9996980809405457
Predicted reduction (Y1, Y2) =  0.22083566690614376
Actual reduction (Y1, Y2) =  0.2208354460704774
Predicted reduction (Y2, Y3) =  0.18052602040829968
Actual reduction (Y1, Y2) =  0.18052583988227974
Predicted reduction (Y1, Y3) =  0.6185755705672468
Actual reduction (Y1, Y2) =  0.6185749519916757
Predicted reduction (Y1, Y2, Y3) =  0.17423061155140251
Actual reduction (Y1, Y2) =  0.1742304373207916
```

Python code:

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

np.random.seed(42)

##############################################
#       Single Control Variate Case          #
##############################################

N = 10**6
sqrt2 = np.sqrt(2) # precomputing factors to make computations faster

print('computing RVs...')
U = st.uniform.rvs(size=(N,2))
Z = [4*int( (u[0]**2+u[1]**2)<=1 ) for u in U]
Y1 = [int( (u[0]+u[1])<=1 ) for u in U]
Y2 = [int( (u[0]+u[1])>=sqrt2 ) for u in U]
Y3 = [(u[0]+u[1]-1)*int( (u[0]+u[1])>1 and (u[0]+u[1])<sqrt2) for u in U]
Y = [Y1,Y2,Y3]

print('computing means..')
meanZ = np.mean(Z)
meanY1 = np.mean(Y1)
meanY2 = np.mean(Y2)
meanY3 = np.mean(Y3)
meanY = [meanY1,meanY2,meanY3]

print('computing variances')
varZ = np.var(Z,ddof=1)
varY1 = np.var(Y1,ddof=1)
varY2 = np.var(Y2,ddof=1)
varY3 = np.var(Y3,ddof=1)
```

```
print('computing covariances')
CovZY1 = np.sum([(z-meanZ)*(y-meanY1) for z,y in zip(Z,Y1)])/(N-1)
CovZY2 = np.sum([(z-meanZ)*(y-meanY2) for z,y in zip(Z,Y2)])/(N-1)
CovZY3 = np.sum([(z-meanZ)*(y-meanY3) for z,y in zip(Z,Y3)])/(N-1)

print('computing optimal alpha')
alpha1 = CovZY1/varY1
alpha2 = CovZY2/varY2
alpha3 = CovZY3/varY3

print('computing control variate samples')
Zcv1 = [z-alpha1*(y-meanY1) for z,y in zip(Z,Y1)]
Zcv2 = [z-alpha2*(y-meanY2) for z,y in zip(Z,Y2)]
Zcv3 = [z-alpha3*(y-meanY3) for z,y in zip(Z,Y3)]

print('Predicted reduction Y1 = ',(1 - CovZY1**2/varZ/varY1 ))
print('Actual reduction Y1 = ', np.var(Zcv1,ddof=1)/varZ)
print('Predicted reduction Y2 = ',(1 - CovZY2**2/varZ/varY2 ))
print('Actual reduction Y2 = ', np.var(Zcv2,ddof=1)/varZ)
print('Predicted reduction Y3 = ',(1 - CovZY3**2/varZ/varY3 ))
print('Actual reduction Y3 = ', np.var(Zcv3,ddof=1)/varZ)

##########################################
#       Multiple Control Variate Case      #
##########################################
# Compute covariance data
CovYY = np.zeros((3,3))
for i in range(3):
    for j in range(i,3):
        meanYi = meanY[i]
        meanYj = meanY[j]
        CovYY[i,j] = np.sum([(yi-meanYi)*(yj-meanYj) for yi,yj in zip(Y[i],Y[j])])/(N-1)

for i in range(1,3):
    for j in range(i):
        CovYY[i,j] = CovYY[j,i]
CovZY = [CovZY1, CovZY2, CovZY3]

##### First Vector Control Variate
CovYY_sub = CovYY[:2,:2]
CovYY_sub_inv = np.linalg.inv(CovYY_sub)
CovZY_sub = CovZY[:2]
print('Predicted reduction (Y1, Y2) = ', 1-np.dot( CovZY_sub, np.dot(CovYY_sub_inv,
 ↪  CovZY_sub))/varZ )
alpha = np.dot( CovYY_sub_inv, CovZY_sub )
Zcv = [z-alpha[0]*(y1-meanY1)-alpha[1]*(y2-meanY2) for z,y1,y2 in zip(Z,Y1,Y2)]
print('Actual reduction (Y1, Y2) = ', np.var(Zcv)/varZ)

##### Second Vector Control Variate
CovYY_sub = CovYY[1:3,1:3]
CovYY_sub_inv = np.linalg.inv(CovYY_sub)
CovZY_sub = CovZY[1:3]
print('Predicted reduction (Y2, Y3) = ', 1-np.dot( CovZY_sub, np.dot(CovYY_sub_inv,
 ↪  CovZY_sub))/varZ )
alpha = np.dot( CovYY_sub_inv, CovZY_sub )
Zcv = [z-alpha[0]*(y2-meanY2)-alpha[1]*(y3-meanY3) for z,y2,y3 in zip(Z,Y2,Y3)]
print('Actual reduction (Y1, Y2) = ', np.var(Zcv)/varZ)

##### Third Vector Control Variate
CovYY_sub = CovYY[::2,::2]
CovYY_sub_inv = np.linalg.inv(CovYY_sub)
CovZY_sub = CovZY[::2]
```

```
print('Predicted reduction (Y1, Y3) = ', 1-np.dot( CovZY_sub, np.dot(CovYY_sub_inv,
 ↪  CovZY_sub))/varZ )
alpha = np.dot( CovYY_sub_inv, CovZY_sub )
Zcv = [z-alpha[0]*(y1-meanY1)-alpha[1]*(y3-meanY3) for z,y1,y3 in zip(Z,Y1,Y3)]
print('Actual reduction (Y1, Y2) = ', np.var(Zcv)/varZ)

##### Fourth Vector Control Variate
CovYY_inv = np.linalg.inv(CovYY)
print('Predicted reduction (Y1, Y2, Y3) = ', 1-np.dot( CovZY, np.dot(CovYY_inv, CovZY))/varZ )
alpha = np.dot( CovYY_inv, CovZY )
Zcv = [z-alpha[0]*(y1-meanY1)-alpha[1]*(y2-meanY2)-alpha[2]*(y3-meanY3) for z,y1,y2,y3 in
 ↪  zip(Z,Y1,Y2,Y3)]
print('Actual reduction (Y1, Y2) = ', np.var(Zcv)/varZ)
```

(Optional) Exercice 5.

Let Z be a random variable of which we would like to estimate the mean value and let Y be a suitable control variate. It the mean of Y is known, we can build a Control Variate Monte Carlo estimator as

$$\hat{\mu}_{CV} = \frac{1}{N}\sum_i (Z^{(i)} - \alpha(Y^{(i)} - E[Y])), \text{ with } (Z^{(i)}, Y^{(i)}) \sim \text{i.i.d } (Z, Y) \tag{5.1}$$

Consider now that case in which $\mathbb{E}[Y]$ is not known and we need to estimate it via sampling.

1) A first idea is to estimate $\mathbb{E}[Y]$ by the sample average estimator $\hat{\mu}_Y = \frac{1}{N}\sum_j Y^{(j)}$ using the same sample as in Eq. (5.1). Show that the resulting estimator is unbiased but its variance is not smaller than (actually equal to) the one of a crude Monte Carlo estimator on Z.

2) A second idea is to estimate $\mathbb{E}[Y]$ with an independent Monte Carlo estimator using a sample size N_Y. Let us denote by C_Z the cost of generating $Z^{(i)}$ and by C_Y the cost of generating $Y^{(i)}$, which we assume smaller than C_Z, and rename the sample size N used in Eq. (5.1) as N_Z. For a given total budget $C = N_Z(C_Z + C_Y) + N_Y C_Y$ for this control variate estimator, determine the optimal choice of N_Z and N_Y and the minimal variance achievable by the above strategy.

3) Compare then the variance obtained with that of a crude Monte Carlo estimator that uses a sample size N the exhausts the same total budget $C = NC_Z$.

Solution

Contributions welcome!