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

Week of 2025-10-13 to 2025-10-17

Lab 6 of Thursday 16th October 2025

Exercice 1.

Consider the discrete time random walk $\{X_n \in \mathbb{Z} \colon X_0 = 0, n \in \mathbb{N}\}$ with transition probabilities:

$$\begin{aligned} p_{i,i+1} &\equiv \mathbb{P}(X_{n+1} = i+1 | X_n = i) = a, \\ p_{i,i-1} &\equiv \mathbb{P}(X_{n+1} = i-1 | X_n = i) = 1-a, \quad n \geqslant 0,\ i \in \mathbb{Z},\ a \in (0,1), \end{aligned}$$

and define the stopping time $\tau_N := \inf\{n \colon X_n = K\}$ for a given constant $K \in \mathbb{N}$. We aim at estimating $\mathbb{P}(\tau_K < T)$, for some given $T \in \mathbb{N}$.

1) Set $K = 4$, $a = 1/3$, $T = 10$. Compute a Monte Carlo estimate of $\mathbb{P}(\tau_K < T)$.

2) For the same values as in the previous point, estimate $\mathbb{P}(\tau_K < T)$ using the antithetic variate variance reduction technique and compare your results to those in point 1.

Solution

Recall that one way of simulating such random walk is to sample $U \sim \mathcal{U}(0,1)$ and setting $X_{n+1} = X_n + 1$ if $U < a$ and $X_{n+1} = X_n - 1$ otherwise. Since $\mathbb{E}[U] = 1/2$, the random variable $\tilde{U} = 2\mathbb{E}[U] - U = 1 - U$ can be used to generate the antithetic path. A possible implementation is attached. We implement both the crude Monte Carlo and the AV method with $N = 10^5$ samples. In this case, however, the variance reduction is quite poor, as shown in the table below.

Method	$\hat{\mu}$	95% CI
Monte Carlo	0.01288	$\pm$ 0.00070
AV	0.01268	$\pm$ 0.00069
$\mathrm{SE}_{\mathrm{MC}}/\mathrm{SE}_{\mathrm{AV}}$		1.01428

This is due to $Z = \mathbb{P}(\tau_K < T)$ and its antithetic variable $\tilde{Z}$ not having a strong negative correlation. This can be seen by examining the sample covariance matrix, which is given by

$$\mathrm{Cov}(Z, \tilde{Z}) = \begin{pmatrix} 0.01230 & -0.00016 \\ -0.00016 & 0.01273 \end{pmatrix}.$$

Python code:

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

np.random.seed(42)

N=100000 #Number of samples
N2=int(N/2) #Number of samples for AV
```

```
X0=0 #initial step
K=5 #stopping time
T=10 #maximum stopping time
alpha = 0.05
Z_mc=np.zeros(N)
Z_mc2=np.zeros(N2)
Z_av=np.zeros(N2)
C_alpha=st.norm.ppf(1-alpha/2)
a=1/3

# we define the random walker for the CMC and the AV
#-----------------------------------------------------
def rw(a,T,K,X0=0):

    finished_mc=0
    finished_av=0
    x_mc=X0
    x_av=X0
    for i in range(T):
        u=np.random.random(1)
        # moves crude MC
        if u<a:
            x_mc=x_mc+1
        else:
            x_mc=x_mc-1
        if x_mc>=K:
            finished_mc=1
         # moves AV with the same random realization
        if  1-u<a:
            x_av=x_av+1
        else:
            x_av=x_av-1
        if x_av>=K:
            finished_av=1


    return finished_mc,finished_av
#-----------------------------------------------------

for i in range(int(N)):
    Z_mc[i],_=rw(a,T,K,X0)

#computes statistics
mean_cmc=np.mean(Z_mc)
var_cmc=np.var(Z_mc)
ci_cmc=C_alpha*np.sqrt(var_cmc)/np.sqrt(N)

# We now compute the AV estimator
# We begin by doing a crude MC estimator
for i in range(int(N2)):
    Z_mc2[i],Z_av[i]=rw(a,T,K,X0)

Y_av=0.5*(Z_mc2+Z_av)
mean_av=np.mean(Y_av)
C=np.cov(Z_mc2,Z_av)
ci_av=C_alpha*np.sqrt( (np.sum(C))/(2*N)  )

#Print results
print('-----------------------------------------------------')
print('Reduction for a='+str(a))
print('mean MC '+str(mean_cmc)+' +- '+str(round(ci_cmc,5)))
print('mean AV '+str(mean_av)+' +- '+str(round(ci_av,5)))
print('Reduction '+str(round(ci_cmc/ci_av,5)))
```

```
print('-------------------------------------------------')
print('Sample covariance matrix')
print(C)
```

Exercice 2.

Suppose that we want to compute $p = \mathbb{P}(\boldsymbol{X} \in A)$, where $\boldsymbol{X}$ is a d-dimensional Gaussian random vector with mean $\boldsymbol{\mu} = \mathbf{0}$ and covariance matrix Σ. If the (Borel) set $A \subseteq \mathbb{R}^d$ contains the mean $\boldsymbol{\mu}$, then the event $\boldsymbol{X} \in A$ is typically not rare, and the use of importance sampling is generally unnecessary. If, on the other hand, $\boldsymbol{\mu} \notin A$ and p is small, then one may wish to consider the use of importance sampling. Let $\mathbb{P}^*$ denote the optimal (yet impractical) sampling measure with density g^*, that is

$$d\mathbb{P}^* = g^*\, d\boldsymbol{x} = \frac{1}{\mathbb{P}(\boldsymbol{X} \in A)} \mathbb{I}_A \phi_\Sigma \, d\boldsymbol{x} \,,$$

where ϕ_Σ is the density of the $\mathcal{N}(\mathbf{0}, \Sigma)$ distribution. Given the rapid decay of $\phi_\Sigma(\boldsymbol{x})$ as $\|\boldsymbol{x}\|_2 \to \infty$, most of the mass of $\mathbb{P}^*$ is typically located at the maximizer $\boldsymbol{x}^*$ of ϕ_Σ over A (which we assume to exist uniquely). This suggests using an importance sampling distribution $\tilde{\mathbb{P}}$ with density g that concentrates most of its mass near $\boldsymbol{x}^*$, that makes g easily computable, and from which realizations can efficiently be generated.

1) One such importance sampling distribution is the Gaussian distribution centered in $\boldsymbol{x}^*$ with covariance matrix Σ. Describe the importance sampling algorithm to estimate p.

2) Implement your algorithm for $d = 2$; take $A = \{\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2 \colon x_i \geqslant a,\ i = 1, 2\}$, and

$$\Sigma = \begin{pmatrix} 4 & -1 \\ -1 & 4 \end{pmatrix} .$$

Then carry out the following points for $a = 1, 3, 10$:

a) First, try to provide simulation estimates of $p = \mathbb{P}(\boldsymbol{X} \in A)$ and the associated 95% confidence interval using the (naive) crude Monte Carlo method.

b) Next, find the point $\boldsymbol{x}^*$ that maximizes the $\mathcal{N}(\mathbf{0}, \Sigma)$ density over A and repeat point (a), with the crude Monte Carlo method replaced by importance sampling, where the importance distribution is $\mathcal{N}(\boldsymbol{x}^*, \Sigma)$.

c) In point (b), experiment with importance distributions of the form $\mathcal{N}(\boldsymbol{x}^*, \delta\Sigma)$ for different $\delta > 0$.

Solution

1) Using $\boldsymbol{x}^* \in \operatorname{argmax}_{\boldsymbol{x} \in A} \phi_\Sigma(\boldsymbol{x})$ (possibly making the maximizer unique) one can use the importance algorithm as described in the lecture notes. As a consequence of using a Gaussian importance distribution g with mean $\boldsymbol{x}^*$ and the same covariance as ϕ_Σ, the weight $w := \phi_\Sigma / g$ can be written as

$$w(\boldsymbol{x}) = \exp\left(-\boldsymbol{x}^{*T} \Sigma^{-1} \boldsymbol{x} + \frac{1}{2} \boldsymbol{x}^{*T} \Sigma^{-1} \boldsymbol{x}^* \right) ,$$

which is easily implemented.

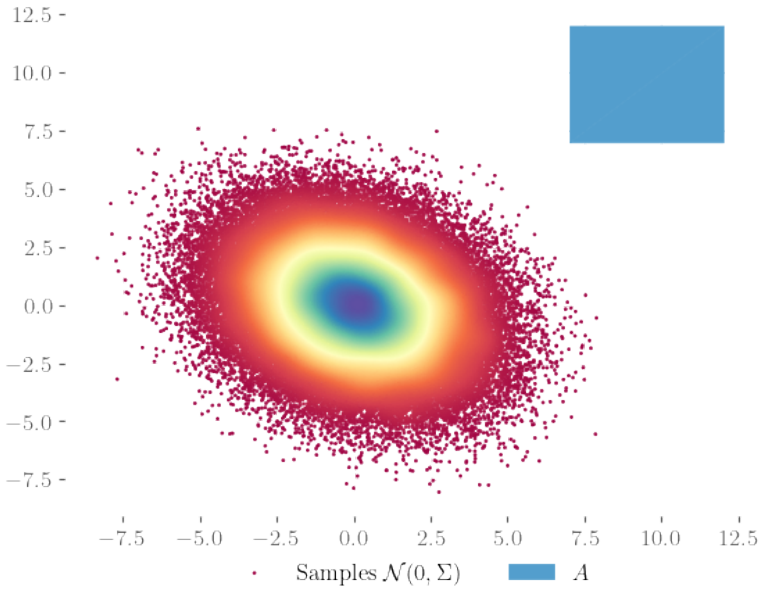

Figure 1: A scatter plot of $n = 10^5$ samples from $\mathcal{N}(0, \Sigma)$ and the region A in blue. As we can see, for this value of N, no samples land on the blue region.

2) The highly accurate approximations of the true values for p are given in the `Python` implementation below. Notice that the implementation of the confidence intervals below uses the quantiles of the student-t distribution instead of the ones of the normal distribution. For large N (often $N > 30$ is suggested) the approximation error of using the quantiles of a normal distribution is of course negligible.

   In part (a), we find that the crude MC method fails to provide estimates of p for $a \geqslant 5$ already when using $N = 100000$. Part (b) with $\boldsymbol{x}^* = (a, a)^T$ significantly improves the situation, as the following simulation output shows, where also the relative root mean squared error is shown:

```
a =  1 , N=  100000
MC confidence interval (alpha= 0.05  N=  100000 ) P   0.06395  +-  0.0015164327173989743
relative RMSE(MC) =  0.011897

IS confidence interval (alpha= 0.05  N=  100000 ) P   0.06477062961844884  +-  0.0008743838532980147
relative RMSE(IS) =  0.00686

-----------------------------------------------------------------------------------------------
a =  3 , N=  100000
MC confidence interval (alpha= 0.05  N=  100000 ) P   0.00145  +-  0.00023584260150583537
relative RMSE(MC) =  0.087186

IS confidence interval (alpha= 0.05  N=  100000 ) P   0.0013893145007358903  +-  2.8290446629351645e-05
relative RMSE(IS) =  0.010458

-----------------------------------------------------------------------------------------------
a =  5 , N=  100000
MC confidence interval (alpha= 0.05  N=  100000 ) P   1e-05  +-  1.9599779078088056e-05
relative RMSE(MC) =  3.356603

IS confidence interval (alpha= 0.05  N=  100000 ) P   3.0061017597175445e-06  +-  8.740749396565515e-08
relative RMSE(IS) =  0.014969

-----------------------------------------------------------------------------------------------
```

```
a =  8 , N=  100000
MC confidence interval (alpha= 0.05  N=  100000 ) P   0.0  +-  0.0
relative RMSE(MC) =  0.0

IS confidence interval (alpha= 0.05  N=  100000 ) P   2.90967611499275e-12  +-  1.2759187183016237e-13
relative RMSE(IS) =  0.022402

-----------------------------------------------------------------------------------------------
a =  10 , N=  100000
MC confidence interval (alpha= 0.05  N=  100000 ) P   0.0  +-  0.0
relative RMSE(MC) =  0.0

IS confidence interval (alpha= 0.05  N=  100000 ) P   1.1589894956354156e-17  +-  6.17092749430613e-19
relative RMSE(IS) =  0.025852

-----------------------------------------------------------------------------------------------
```

In part (c) we observe that $\delta = 1$ is actually optimal. We also observe that the variance explodes as δ gets too small, as can be seen if Figure 2 for $a = 3$.

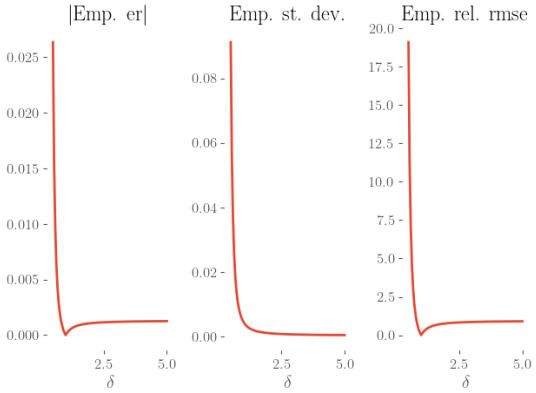

Figure 2: Plots for Exercise 1.2.(c) for $a = 3$.

Python code:

```
import numpy as np
from numpy import matlib
from scipy.stats import t as ts
import matplotlib.pyplot as plt
from matplotlib import rc
from aux_plot import density_scatter #auxiliary library to plot scatterdensitites
#
#   imports graphical paramters
#
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'
import matplotlib
latex_preamble = r'\usepackage{amsmath} \usepackage{amssymb}'
matplotlib.rcParams.update({
    'text.usetex': True,
    'text.latex.preamble': latex_preamble
```

```
})

np.random.seed(42)

digits=6 #how many digits of accuracy to display
mu = np.array([0,0])
Sigma = np.array([[4,-1],[-1,4]]);
A = np.linalg.cholesky(Sigma);
alpha = 0.05; # 0.95 = 1-alpha
N = int(1E5);
cval = ts.ppf(1-alpha/2,N-1);
alist = np.array([1,3,5,8,10])
y_normal=A@np.random.standard_normal((2,N))
fig,ax= plt.subplots(nrows = 1, ncols = 1, figsize = (6,4))
aa=7
ab=12

ax=density_scatter(y_normal[0,:],y_normal[1,:],sort=True,
                   bins=20,cmap='Spectral')

xy=np.linspace(aa,ab)
ax.fill_between(xy, xy, np.max(xy), color='#539ecd')
ax.fill_between(xy, xy, aa,color='#539ecd')
plt.legend([r'Samples $\mathcal{N}(0,\Sigma)$',r'$A$'],fancybox=True,
            framealpha=0.0,loc='upper center', bbox_to_anchor=(0.5, -0.05),
            shadow=False, ncol=2)
plt.show()

#these values can acutally be computed.
truep = np.array([0.06503208932, 0.001380140995, 0.000002979188004,\
                  2.905959243*10**(-12), 1.217860285*10**(-17)])
na  = np.size(alist);

def psi(x,a):
    return ((x[1,:]>a)*(x[0,:]>a))

def w(xstar,Sigma,x):
    return np.exp(-xstar.T@(np.linalg.solve(Sigma,x)) +
                  0.5*xstar.T@np.linalg.solve(Sigma,xstar))
#for 2c
def wd(xstar,Sigma,x,d):
    return np.exp(-xstar.T@(np.linalg.solve(Sigma,x)) +
                  0.5*xstar.T@np.linalg.solve(Sigma,xstar)/d)

for i in range(na):
    a = alist[i]
    print('a = ',a,', N= ',N);
    # set-up importance distribution and weight function
    xstar = np.array([a,a])

    # perform sampling
    xi = np.random.standard_normal((2,int(N)))
    X = matlib.repmat(mu,2,int(N/2)) + A@xi;
    Y = matlib.repmat(xstar,2,int(N/2)) + A@xi;
    # compute mean and standard dev.
    Z = psi(X,a);
    mean_mc = np.mean(Z);
    std_mc  = np.std(Z);
    print('MC confidence interval (alpha=',alpha,' N= ',N,') P  ',
          mean_mc, ' +- ',cval*std_mc/np.sqrt(N));
    print('relative RMSE(MC) = ',round(std_mc/np.sqrt(N)/truep[i],digits));
    print("                                                    ")
    ZZ = psi(Y,a)*w(xstar,Sigma,Y);
```

```
    mean_is = np.mean(ZZ);
    std_is = np.std(ZZ);
    print('IS confidence interval (alpha=',alpha,' N= ',N,') P  ', mean_is,
          ' +- ',cval*std_is/np.sqrt(N));
    print('relative RMSE(IS) = ',round(std_is/np.sqrt(N)/truep[i],digits));
    print("                                                    ")
    print("------------------------------------------------------------\
          ----------------------------------")

############ Plays with \delta
deltalist = np.linspace(0.5,5,100)
ndelta = np.size(deltalist);
for i in range(na):

    a = alist[i]
    # set-up importance distribution and weight function
    xstar = np.array([a,a]);
    #perform sampling
    xi = np.random.standard_normal([2,int(N)])
    errorlist = np.zeros(ndelta)
    stdlist = np.zeros(ndelta)
    for j in range(ndelta):
        delta = deltalist[j];
        Y = np.matlib.repmat(xstar,2,int(N/2)) + A@xi;
        ZZ = psi(Y,a)*wd(xstar,Sigma,Y,delta);
        mean_is = np.mean(ZZ);
        std_is = np.std(ZZ);
        errorlist[j] = mean_is - truep[i];
        stdlist[j] = std_is;

############# Plots
    plt.clf()
    print('a= ',a)
    plt.subplot(131)

    plt.plot(deltalist,abs(errorlist));
    plt.title(r'$\left|\text{Emp. er}\right|$');
    plt.xlabel('$\delta$')

    plt.subplot(132)

    plt.plot(deltalist,stdlist);
    plt.title('Emp. st. dev.');
    plt.xlabel('$\delta$')

    plt.subplot(133)

    plt.plot(deltalist,np.sqrt( (errorlist)**2 + stdlist**2/N )/truep[i]);
    plt.title('Emp. rel. rmse');
    plt.xlabel('$\delta$')
    plt.tight_layout()
    plt.show()
```

Exercice 3.

Suppose that we wish to compute $\mu = \mathbb{E}\big[\Psi_\tau(X_0, \dots, X_\tau)\mathbb{I}_{\{\tau<\infty\}}\big]$, where τ is a stopping time adapted to the discrete time, discrete state Markov chain $\{X_n \in \mathbb{Z}^d \colon n \in \mathbb{N}_0\}$ with initial probability distribution p_0 (i.e. $X_0 \sim p_0$) and with Markov transition probabilities $p_{i,j} \in [0,1]$ such that

$$p_{i,j} = \mathbb{P}(X_{n+1} = j | X_n = i) \ , \forall \, i, j \in \mathbb{Z}^d \ .$$

Instead of relying on the evolution of this Markov chain, it is natural to try to use importance measures that preserve the Markov property (so as to guarantee that the paths can be simulated efficiently under the importance measure). That is, one replaces the transition probabilities $p_{i,j}$ by other Markov transition probabilities $q_{i,j} \in [0,1]$, which dominate $p_{i,j}$ (i.e. $q_{i,j} = 0 \Rightarrow p_{i,j} = 0$); analogously for the initial distribution. Moreover, we require that the stopping time τ is almost surely finite for the Markov process with transition probabilities $q_{i,j}$, which appears natural from a practical point of view.

1) Prove that $\mu = \mathbb{E}\big[\Psi_\tau(X_0, \dots, X_\tau)\mathbb{I}_{\tau<\infty}\big]$ can be written as:

$$\mu = \mathbb{E}_q\big[\Psi_\tau(X_0, \dots, X_\tau)w(X_0, \dots, X_\tau)\big] \ , \quad \text{with} \quad w(X_0, \dots, X_m) = \frac{p_0(X_0)}{q_0(X_0)} \prod_{j=1}^{m} \frac{p_{X_{j-1},X_j}}{q_{X_{j-1},X_j}} \ .$$

2) Implement the importance sampling algorithm for discrete time Markov processes to the random walk $\{X_n \in \mathbb{Z} \colon X_0 = 0, n \in \mathbb{N}\}$ with transition probabilities:

$$p_{i,i+1} \equiv \mathbb{P}(X_{n+1} = i+1 | X_n = i) = \frac{1}{2} = \mathbb{P}(X_{n+1} = i-1 | X_n = i) \equiv p_{i,i-1} \ , \quad n \geqslant 0, \, i \in \mathbb{Z} \ .$$

Consider the stopping time $\tau_N := \inf\{n \colon X_n = N\}$ for a given constant $N = 4$ and apply the algorithm to estimate $\mathbb{P}(\tau_N < T)$ with $T = 10$. For the the importance sampling, consider the random walk with transition probabilities

$$q_{i,i+1} \equiv \mathbb{P}(X_{n+1} = i+1 | X_n = i) = \alpha > 1 - \alpha = \mathbb{P}(X_{n+1} = i-1 | X_n = i) \equiv q_{i,i-1} \ .$$

Experiment with different values for α.

Solution

1) The claim follows from the Markov property and properties of conditional expectations Indeed, let $\mathbb{E}_p$ denote the expectation with respect to the transition probabilities $p_{i,j}$ and $\mathbb{E}_{p,X_{0:k}}$ the expectation with respect to the density corresponding to the first k steps of the process. Then

$$\begin{aligned}
\mu &= \mathbb{E}_p\big[\Psi_\tau(X_0, \dots, X_\tau)\mathbb{I}_{\tau<\infty}\big] = \sum_{m=0}^{\infty} \mathbb{E}_p\big[\Psi_m(X_0, \dots, X_m)\mathbb{I}_{\tau=m}\big] \\
&= \sum_{m=0}^{\infty} \mathbb{E}_{p,X_{0:m-1}}\Big[\mathbb{E}_p\big[\Psi_m(X_0, \dots, X_m)\mathbb{I}_{\tau=m}\big|X_0, \dots, X_{m-1}\big]\Big] \\
&= \sum_{m=0}^{\infty} \mathbb{E}_{p,X_{0:m-1}}\Big[\mathbb{E}_q\big[\Psi_m(X_0, \dots, X_m)\mathbb{I}_{\tau=m}\frac{p_{X_{m-1},X_m}}{q_{X_{m-1},X_m}}\big|X_0, \dots, X_{m-1}\big]\Big] \\
&= \cdots = \sum_{m=0}^{\infty} \mathbb{E}_q\big[\Psi_m(X_0, \dots, X_m)\mathbb{I}_{\tau=m}\prod_{j=1}^{m}\frac{p_{X_{j-1},X_j}}{q_{X_{j-1},X_j}}\frac{p(X_0)}{q(X_0)}\big] \\
&= \mathbb{E}_q\big[\Psi_\tau(X_0, \dots, X_\tau)\prod_{j=1}^{\tau}\frac{p_{X_{j-1},X_j}}{q_{X_{j-1},X_j}}\frac{p(X_0)}{q(X_0)}\big] \ ,
\end{aligned}$$

where the last equality is due to the assumption that τ is almost surely finite under the importance sampling distribution (denoted by q).

2) For this example the weight function simplifies to

$$w(X_0, \dots, X_m) = \frac{1}{(2\alpha)^{n_{\text{up}}}} \frac{1}{(2(1-\alpha))^{n_{\text{down}}}} , \quad m = n_{\text{up}} + n_{\text{down}} ,$$

where $n_{\text{up}} = \sum_{j=1}^{m} \mathbb{I}(X_j - X_{j-1} = 1)$ is the number of increments in the process.

When applying the importance sampling method, we observe a variance reduction (i.e. efficiency improvement) compared to the crude MC method, which results in narrower confidence intervals. When repeating the experiment for different values of the transition probability α, Figure 3 indicates an optimal value of around 0.82.

```
1 a=   0.33
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.186  +/-  0.0241
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.2074  +/-  0.0678
-----------------------------------------------------------------
1 a=   0.37133333333333335
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.193  +/-  0.0245
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1874  +/-  0.0492
-----------------------------------------------------------------
1 a=   0.4126666666666667
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.183  +/-  0.024
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1863  +/-  0.0381
-----------------------------------------------------------------
1 a=   0.45399999999999996
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.192  +/-  0.0244
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1929  +/-  0.0309
-----------------------------------------------------------------
1 a=   0.4953333333333333
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.168  +/-  0.0232
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1869  +/-  0.0247
-----------------------------------------------------------------
1 a=   0.5366666666666666
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.176  +/-  0.0236
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1621  +/-  0.0193
-----------------------------------------------------------------
1 a=   0.578
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.18  +/-  0.0238
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1729  +/-  0.0163
-----------------------------------------------------------------
1 a=   0.6193333333333333
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.183  +/-  0.024
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1739  +/-  0.0136
-----------------------------------------------------------------
1 a=   0.6606666666666666
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.187  +/-  0.0242
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1809  +/-  0.0113
-----------------------------------------------------------------
1 a=   0.702
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.18  +/-  0.0238
```

```
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1856  +/-  0.0094
------------------------------------------------------------------
1 a=   0.7433333333333333
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.181  +/-  0.0239
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1836  +/-  0.008
------------------------------------------------------------------
1 a=   0.7846666666666666
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.188  +/-  0.0242
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1817  +/-  0.0074
------------------------------------------------------------------
1 a=   0.8259999999999998
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.174  +/-  0.0235
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1752  +/-  0.0069
------------------------------------------------------------------
1 a=   0.8673333333333333
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.181  +/-  0.0239
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1821  +/-  0.0087
------------------------------------------------------------------
1 a=   0.9086666666666665
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.187  +/-  0.0242
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1797  +/-  0.0117
------------------------------------------------------------------
1 a=   0.95
MC confidence interval (aa= 0.05 , NM= 1000 ):  0.152  +/-  0.0223
IS confidence interval (aa= 0.05 , NM= 1000 ):  0.1979  +/-  0.0233
------------------------------------------------------------------
```

Python code:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
from matplotlib import rc
#
#   imports graphical paramters
#
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'
import matplotlib
latex_preamble = r'\usepackage{amsmath} \usepackage{amssymb}'
matplotlib.rcParams.update({
    'text.usetex': True,
    'text.latex.preamble': latex_preamble
})

np.random.seed(42)

# functions
def genWalk(prob,X0,Nstop,T):
    X=X0;
```

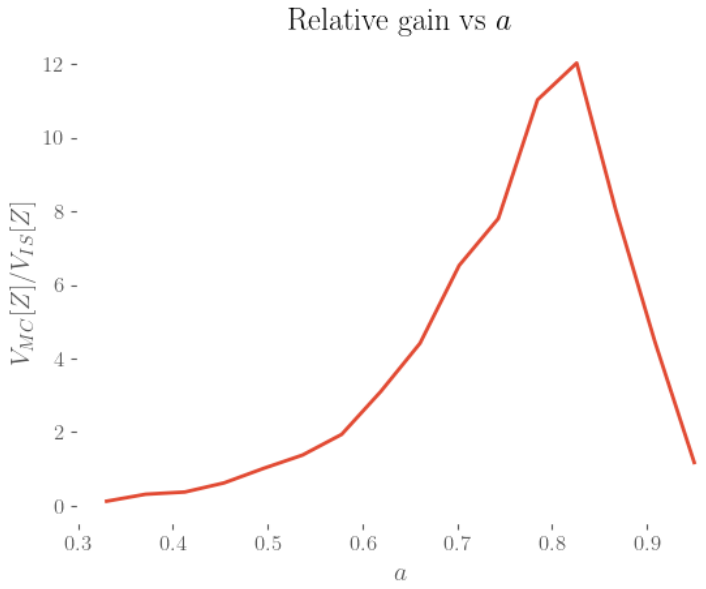

Figure 3: Variance ratio between the crude Monte Carlo and the importance sampling estimators as a function of α.

```
    n=0;
    while((X[n]<Nstop)*(n<=T)):
        X=np.append(X, X[n]-1+2*np.random.binomial(1,prob))
        n=n+1
    t=np.arange(0,n+1)
    return X,t

def weightfun(prob,Y):
    dY=np.diff(Y)
    # rvec is a vector where only one of the two terms in
    # the sum below appears in each component of the vector
    rvec=1/(2*prob)*(dY==1) + 1/(2*(1-prob))*(dY==-1)
    rval=np.prod(rvec)
    return rval

# problem set-up
alist = np.linspace(0.33,0.95,16)
na = np.size(alist);
Nstop = 4;
T = 10;
NMC = int(1e3);
X0 = np.zeros(1);
# for confidence intervals
aa = 1-0.95;
cval = st.norm.ppf(1-aa/2);
# let's get started
varis = np.zeros(na);
varmc = np.zeros(na);

for j in range(na):
    alpha = alist[j];
    print(1,'a=  ',alpha);
```

```
    Y = np.zeros(int(NMC),);
    tauNaive = np.zeros(int(NMC),);
    for i in range(NMC):
        # CMC estimator
        Xnaive,tnaive, = genWalk(0.5,X0,Nstop,T);
        tauNaive[i] = tnaive[-1]<T;
        # IS estimator
        X,t = genWalk(alpha,X0,Nstop,T);
        Y[i] = (t[-1]<T)*weightfun(alpha,X);

    varmc[j]=np.var(tauNaive)
    varis[j] = np.var(Y);
    print('MC confidence interval (aa=',0.05,', NM=',NMC,'): ',
          round(np.mean(tauNaive),4),' +/- ',round(cval*np.std(tauNaive)/np.sqrt(NMC),4));
    print('IS confidence interval (aa=',0.05,', NM=',NMC,'): ',
          round(np.mean(Y),4),' +/- ',round(cval*np.std(Y)/np.sqrt(NMC),4));
    print("-----------------------------------------------------------------")
np.var(tauNaive)

plt.plot(alist,varmc/varis);
plt.xlabel(r'$a$')
plt.ylabel(r'$V_{MC}[Z]/V_{IS}[Z]$')
plt.title(r'Relative gain vs $a$')
plt.show()
```