TP2 — Reduction de variance (corrige)

Source : TP2/variance_reduction_corrige.ipynb — PDF compile (code execute).

MAP556, Ecole Polytechnique, 2023-24

TP 2 - Variance Reduction

We have a model defined by the relationship $Y = f(X)$. The objective is to estimate the value $\mathbb{E}\bigl[g(Y)\bigl]$, for a certain function g over the set of outputs Y. We assume that $g(Y)$ is square-integrable.

Exercise 1. Control Variates

Black-box models representing input–output systems of numerical codes are often computationally expensive. Suppose we have a reduced input–output model $ Y_r = f_r(X) $ that is easier to simulate for different input values $ X $ than the full model $ Y = f(X) $, and for which the quantity

$$ m_r = \mathbb{E}\bigl[g(Y_r)\bigr] $$

is known. Furthermore, we assume that

$$ \mathbb{E}\bigl[g(Y_r)^2\bigr] < \infty. $$

Let $ (X_i)_{1 \le i \le n} $ be a sequence of independent copies of the input variable $ X $. We define

$$ I_n = \frac{1}{n} \sum_{i=1}^n g\bigl(f(X_i)\bigr), \qquad I_n^c = m_r + \frac{1}{n} \sum_{i=1}^n \bigl(g(f(X_i)) - g(f_r(X_i)) \bigr). $$

Question 1:

Verify that I_n and I_n^c are unbiased estimators of $\mathbb{E}\bigl[g(Y)\bigl]$, and calculate their variances.

Question 2:

Assuming that the input variables X follow a uniform distribution on $[0,1]$, $f(x)=e^x$, $f_r(x)=1+x$, and $g(y)=y$.

Simulate both estimators and their asymptotic 95% confidence intervals.

Plot the curves of the empirical means I_n and I_n^c.

What is the gain in terms of the number of simulations for the estimator I_n^c to achieve the same asymptotic precision as the naive Monte Carlo method?

In [1] :

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
```

In [2] :

```
        N = 1000 # Sample size

        integers1toN = np.arange(1,N+1) # A vector containing integers from 1 to N

        ##############################################
        # Exact m_r = E[1+X] = 1.5 pour X~U[0,1]
        m_r = 1.5
        ##############################################

        ##############################################
        # N echantillons
        X = np.random.rand(N)
        Y = np.exp(X)
        Y_control = 1.0 + X
        ##############################################

        ##############################################
        Mean_MC = np.mean(Y)
        Variance_MC = np.var(Y, ddof=1)

        Mean_control = m_r + np.mean(Y - Y_control)
        Variance_control = np.var(Y - Y_control, ddof=1)

        halfWidthCI95_MC = 1.96 * np.sqrt(Variance_MC / N)
        halfWidthCI95_Control = 1.96 * np.sqrt(Variance_control / N)

        control_gain = Variance_MC / Variance_control
        ##############################################

        Esp_gY = np.exp(1.)-1.

        print("Sample size = %d" %N)
        print("Esp_gY = %1.3f \n" %Esp_gY)
        print("MC Estimator: mean = %1.3f emp variance = %1.3f"
        %(Mean_MC, Variance_MC))
        print("CI(95%%) = [%1.3f,%1.3f] \n"
        %(Mean_MC-halfWidthCI95_MC, Mean_MC+halfWidthCI95_MC))

        print("Control Estimator: mean = %1.3f empirical variance = %1.3f"
        %(Mean_control, Variance_control))
        print("CI(95%%) = [%1.3f,%1.3f] \n"
        %(Mean_control-halfWidthCI95_Control, Mean_control+halfWidthCI95_Control))

        print("Gain in number of simulations with control variable: %1.2f" %control_gain)
        print("Theorie Q1 : E[I_n]=E[g(Y)], Var(I_n)=Var(g(Y))/n ; E[I_n^c]=E[g(Y)],
        Var(I_n^c)=Var(g(Y)-g(Yr))/n << Var(I_n).")
```

```
Sample size = 1000
Esp_gY = 1.718

MC Estimator: mean = 1.721 emp variance = 0.251
CI(95%) = [1.690,1.752]

Control Estimator: mean = 1.721 empirical variance = 0.045
CI(95%) = [1.708,1.734]

Gain in number of simulations with control variable: 5.58
Theorie Q1 : E[I_n]=E[g(Y)], Var(I_n)=Var(g(Y))/n ; E[I_n^c]=E[g(Y)], Var(I_n^c)=Var(g(Y)-g(Yr))/n
<< Var(I_n).
```

In [3] :

```
        ############################################
        ## Displaying 10 trajectories
        ############################################

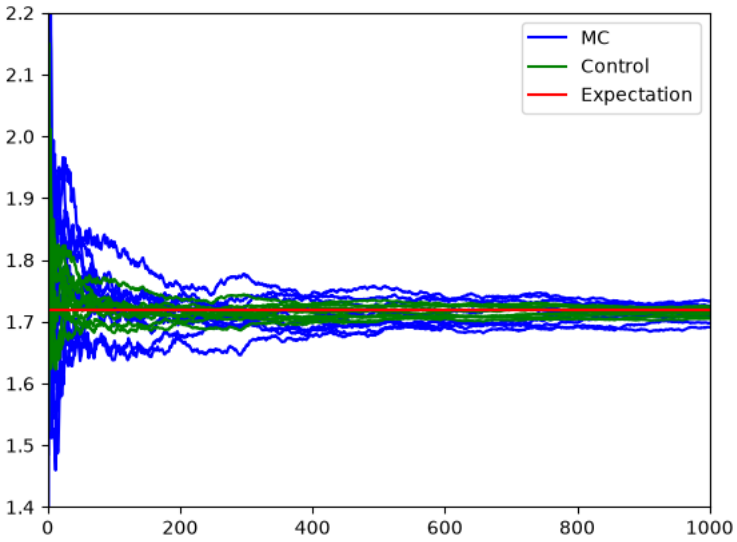
        M = 10
        X = np.random.rand(M, N)
        Y = np.exp(X)
        Ycontrol = 1.0 + X
        I_n = np.cumsum(Y, axis=1) / integers1toN
        Ic_n = m_r + np.cumsum(Y - Ycontrol, axis=1) / integers1toN

        ## Display the 10 trajectories
        fig = plt.figure()
        ax = fig.add_subplot(111)

        ax.plot(integers1toN, I_n[0], color="b", label="MC")
        ax.plot(integers1toN, I_n[1:].T, color="b")

        ax.plot(integers1toN, Ic_n[0], color="g", label="Control")
        ax.plot(integers1toN, Ic_n[1:].T, color="g")
        ax.axhline(Esp_gY, color="r", label="Expectation")

        ax.set_xlim(0, N)
        ax.set_ylim(1.4, 2.2)
        ax.legend(loc="best")
        plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

Question 3:

Let $(I^j_n)_{1\leq i\leq M}$ and $(I^{c,j}_n)_{1\leq i\leq M}$ be the empirical estimations associated with M independent draws of the two estimators.

Explicitly evaluate $m = \mathbb{E}\bigl[g(Y)\bigr]$ and plot histograms of the errors $(I^j_n - m)_{1 \le j \le M}$ and $(I^{c,j}_n - m)_{1 \le j \le M}$.

In [4] :

```
N = 500 # Sample size
M = 1000 # Number of estimator draws

integers1toN = np.arange(1,N+1)

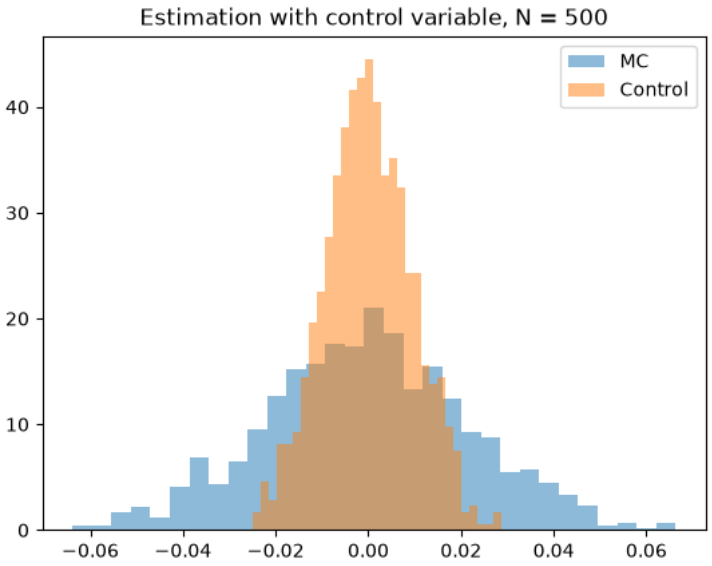
Esp_gY = np.exp(1.)-1.
m_r = 1.5

###############
X = np.random.rand(M, N)
Y = np.exp(X)
Ycontrol = 1.0 + X
###############

I_N = np.mean(Y, axis=1)
Ic_N = m_r + np.mean(Y - Ycontrol, axis=1)

plt.hist(I_N - Esp_gY, density="True", bins=int(np.sqrt(M)), alpha=0.5, label="MC")
plt.hist(Ic_N - Esp_gY, density="True", bins=int(np.sqrt(M)), alpha=0.5, label="Control")
plt.title("Estimation with control variable, N = %1.0f" %N)

plt.legend(loc="best")
plt.show()
print(f"std MC={np.std(I_N):.4f}, std control={np.std(Ic_N):.4f},
gain={np.var(I_N)/np.var(Ic_N):.1f}")
```




```
<Figure size 640x480 with 1 Axes>
```

```
std MC=0.0220, std control=0.0094, gain=5.5
```

Question 4: Optimal Control Variable

Now, let's consider the estimator $$ I_n^\lambda = \lambda m_r + \frac{1}{n}\sum_{i=1}^n \bigl(g(f(X_i)) - \lambda g(f_r(X_i))\bigr), \qquad \lambda \in \mathbb R. $$

Suggest a choice for the parameter λ.

Plot the trajectories of the empirical means I_n^λ obtained with this choice, and then plot the histogram of the errors of this estimator.

In [5] :

```
        ##############################################
        ## Estimating optimal lambda from a small
        ## number of simulations
        ##############################################
        n = 100
        X = np.random.rand(n)
        Y = np.exp(X)
        Yc = 1.0 + X

        ## lambda* = Cov(Y,Yc)/Var(Yc)
        lambda_opt = np.cov(Y, Yc, ddof=1)[0,1] / np.var(Yc, ddof=1)

        print("Optimal lambda = %1.3f" %lambda_opt)
        print("Theorie Q4 : min Var(Y-lambda*Yc) atteint en lambda*=Cov(Y,Yc)/Var(Yc).")
```

```
Optimal lambda = 1.710
Theorie Q4 : min Var(Y-lambda*Yc) atteint en lambda*=Cov(Y,Yc)/Var(Yc).
```

In [6] :

```
        ##############################################
        ### For histograms: M x N draws
        ##############################################
        M = 1000
        N = 500
        integers1toN = np.arange(1,N+1)

        X = np.random.rand(M, N)
        Y = np.exp(X)
        Ycontrol = 1.0 + X

        ##############################################
        # residu avec lambda_opt estime ci-dessus (re-estime ici si besoin)
        try:
        lam = float(lambda_opt)
        except NameError:
        lam = float(np.cov(Y[0,:], Ycontrol[0,:])[0,1] / np.var(Ycontrol[0,:]))
        print("lambda utilise =", lam)

        D = Y - lam * Ycontrol # partie centree a renormaliser
        ##############################################
        ### Displaying the first 10 trajectories
        ##############################################

        I_n = np.cumsum(Y[0:10,:], axis=1) / integers1toN
        Ic_n = lam * m_r + np.cumsum(D[0:10,:], axis=1) / integers1toN

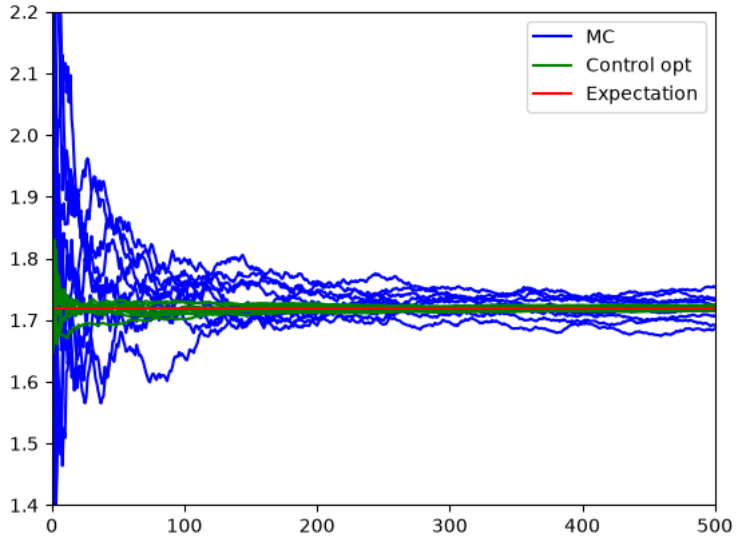
        ## Display the first 10 trajectories
        fig = plt.figure()
        ax = fig.add_subplot(111)

        ax.plot(integers1toN, I_n[0], color="b", label="MC")
        ax.plot(integers1toN, I_n[1:].T, color="b")

        ax.plot(integers1toN, Ic_n[0], color="g", label="Control opt")
        ax.plot(integers1toN, Ic_n[1:].T, color="g")
        ax.axhline(Esp_gY, color="r", label="Expectation")

        ax.set_xlim(0, N)
        ax.set_ylim(1.4, 2.2)
        ax.legend(loc="best")
        plt.show()
```

```
lambda utilise = 1.7101422480282675
```




```
<Figure size 640x480 with 1 Axes>
```

In [7] :

```
        ##########################################
        # Estimateurs I_N et I_N^lambda
        I_MC = np.mean(Y, axis=1)
        I_OPT = lam * m_r + np.mean(Y - lam * Ycontrol, axis=1)

        Mean_MC = np.mean(I_MC)
        Variance_MC = np.var(Y, ddof=1) # variance d'une replique

        Mean_control = np.mean(I_OPT)
        Variance_control = np.var((Y - lam*Ycontrol), ddof=1)

        halfWidthCI95_MC = np.sqrt(Variance_MC / N)*1.96
        halfWidthCI95_Control = np.sqrt(Variance_control / N)*1.96

        control_gain = Variance_MC/Variance_control

        print("Sample size = %d" %N)
        print("Esp_gY = %1.3f \n" %Esp_gY)
        print("MC Estimator: mean = %1.3f empirical variance = %1.3f" %(Mean_MC, Variance_MC))
        print("CI(95%%) = [%1.3f,%1.3f] \n" %(Mean_MC-halfWidthCI95_MC, Mean_MC+halfWidthCI95_MC))

        print("Control Estimator: mean = %1.4f empirical variance = %1.3f" %(Mean_control,
        Variance_control))
        print("CI(95%%) = [%1.3f,%1.3f] \n" %(Mean_control-halfWidthCI95_Control,
        Mean_control+halfWidthCI95_Control))

        print("Gain in number of simulations with control variable: %1.2f" %control_gain)
```

```
Sample size = 500
Esp_gY = 1.718

MC Estimator: mean = 1.718 empirical variance = 0.243
CI(95%) = [1.675,1.761]

Control Estimator: mean = 1.7185 empirical variance = 0.004
CI(95%) = [1.713,1.724]

Gain in number of simulations with control variable: 61.01
```

In [8] :

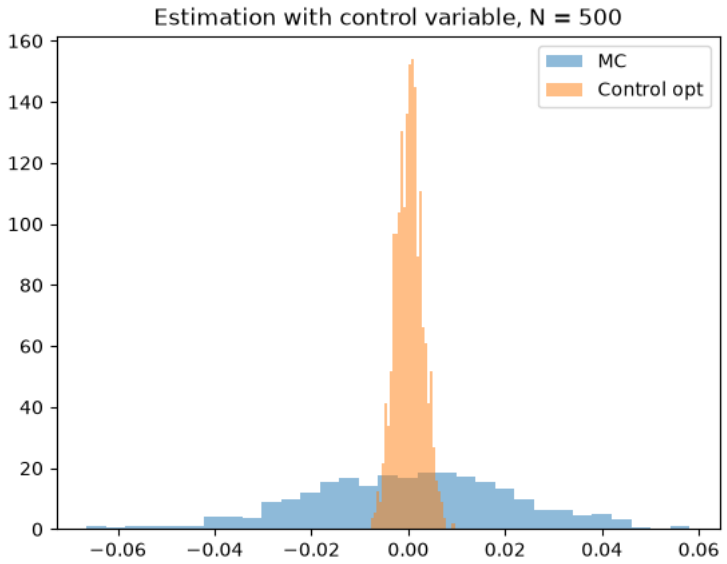
```
error_N = I_MC - Esp_gY
error_control_N = I_OPT - Esp_gY

plt.hist(error_N, density="True", bins=int(np.sqrt(M)), alpha=0.5, label="MC")

plt.hist(error_control_N, density="True", bins=int(np.sqrt(M)), alpha=0.5, label="Control
opt")

plt.title("Estimation with control variable, N = %1.0f" %N)

plt.legend(loc="best")
plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

Exercise 2. Antithetic Sampling and Stratification Methods

We assume that the input variables X follow a uniform distribution between -1 and $+1$. We are in the situation where $f(x)=e^x$, and $g(y)=y$, and we want to estimate $\mathbb{E}\bigl[g(Y)\bigl]= \mathbb{E}\bigl[e^X\bigl]$.

Question 1: Basic Monte Carlo Method

Let I_n be the empirical mean of n independent copies of X.

Verify that we have $\mathbb{E}\bigl[e^X\bigl] = \sinh(1)\simeq 1.18$, and $\mathrm{Var}(e^X)= \frac{1}{2} (1-e^{-2})\simeq 0.43$.

Plot the curves of the empirical means obtained from multiple realizations and compare them with the desired quantity.

In [9] :

```
# Constants to be used for the rest of the exercise

N = 1000 # Sample size
integers1toN = np.arange(1,N+1) # A vector containing integers from 1 to N

Esp_gY = np.sinh(1.0)
Var_gY = (1.0 - np.exp(-2.))/2.
```

In [10] :

```
        ###############################################
        X = -1.0 + 2.0 * np.random.rand(N) # U[-1,1]
        Y = np.exp(X)
        ###############################################

        mean = np.mean(Y)
        var = np.var(Y, ddof=1)
        halfWidthCI = 1.96 * np.sqrt(var / N)
        ###############################################

        print("MC Estimator \n")

        print("Esp_gY = %1.3f Var_gY = %1.3f" %(Esp_gY, Var_gY))
        print("mean = %1.3f var = %1.3f" %(mean,var))
        print("95%% confidence interval for E[g(Y)] = [ %1.3f , %1.3f ] \n" %(mean - halfWidthCI,
        mean + halfWidthCI))
        print("Relative error = %1.3f" %(halfWidthCI/mean))
```

```
MC Estimator

Esp_gY = 1.175 Var_gY = 0.432
mean = 1.197 var = 0.445
95% confidence interval for E[g(Y)] = [ 1.155 , 1.238 ]

Relative error = 0.035
```

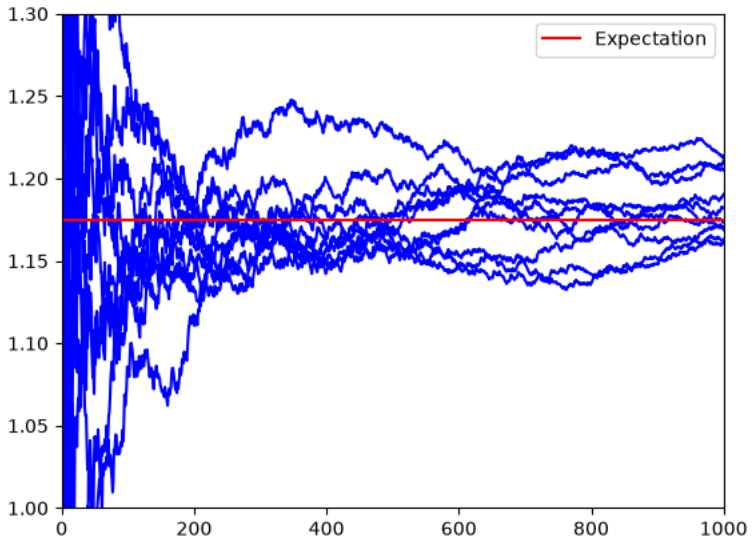
In [11] :

```
        M = 10 # number of realizations of empirical means

        X = -1.0 + 2.0 * np.random.rand(M, N)
        Y = np.exp(X)
        I_n = np.cumsum(Y, axis=1) / integers1toN
        ###############################################

        # Displaying trajectories
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(integers1toN, I_n.T, color="b")

        ax.set_xlim(0, N)
        ax.set_ylim(1.0, 1.3)
        ax.axhline(Esp_gY, color="r", label="Expectation")
        ax.legend(loc="best")
        plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

Question 2: Antithetic Sampling

Verify that $(f(u)-f(v))(f(-u)-f(-v))\leq 0$, for all $(u,v)\in [-1,1]$, and deduce a antithetic sampling technique based on simulating n uniform variables on $[-1,1]$.

Let $I^{\prime}_n$ be the corresponding empirical estimator.

Plot the curves of the empirical means obtained.

What is the gain in terms of the number of simulations for this estimator, for the same asymptotic accuracy as the naive Monte Carlo method?

In [12] :

```
        ############################################
        X = -1.0 + 2.0 * np.random.rand(N) # U[-1,1]
        Z = -X # variable antithetique
        Yanti = 0.5 * (np.exp(X) + np.exp(Z)) # = cosh(X)
        #
        ############################################

        mean = np.mean(Yanti)
        var = np.var(Yanti, ddof=1)
        halfWidthCI = 1.96 * np.sqrt(var / N)
        ############################################

        print("Antithetic Estimator \n")

        print("Esp_gY = %1.3f" %(Esp_gY))
        print("I_prime_mean = %1.3f empirical variance = %1.3f" %(mean, var))
        print("95%% Confidence Interval for E[g(Y)] = [ %1.3f , %1.3f ] \n" %(mean - halfWidthCI,
        mean + halfWidthCI))
        print("Relative error = %1.3f" %(halfWidthCI/mean))
        print("gain in number of simulations compared to MC: %1.2f" %(Var_gY/var))
        print("Justif : (e^u-e^v)(e^{-u}-e^{-v})<=0 car exp croissante, donc Cov(e^X,e^{-X})<=0 et
        Var((Y+Yanti)/2)<=Var(Y)/2.")
```

```
Antithetic Estimator

Esp_gY = 1.175
I_prime_mean = 1.178 empirical variance = 0.026
95% Confidence Interval for E[g(Y)] = [ 1.169 , 1.188 ]

Relative error = 0.008
gain in number of simulations compared to MC: 16.77
Justif : (e^u-e^v)(e^{-u}-e^{-v})<=0 car exp croissante, donc Cov(e^X,e^{-X})<=0 et
Var((Y+Yanti)/2)<=Var(Y)/2.
```

In [13] :

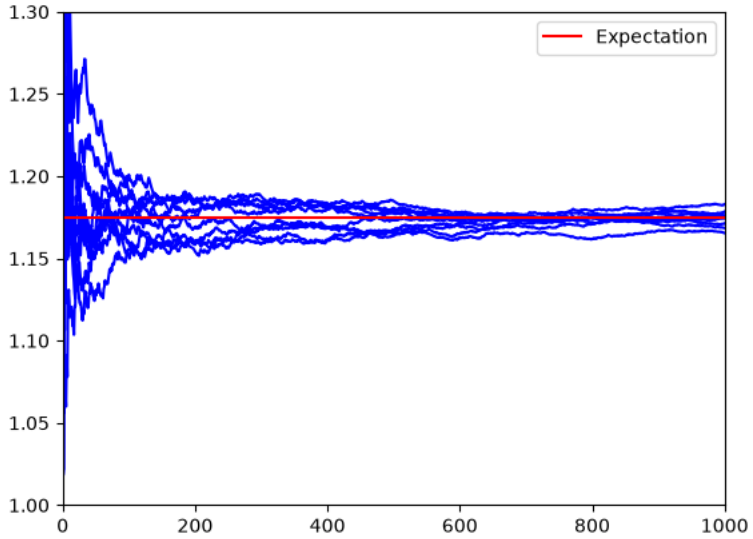
```
        ############################################
        ## Empirical mean trajectories
        ############################################

        M = 10
        X = -1.0 + 2.0 * np.random.rand(M, N)
        Z = -X
        Yanti = 0.5 * (np.exp(X) + np.exp(Z))

        I_prime_n = np.cumsum(Yanti, axis=1) / integers1toN
        ############################################

        # Displaying the estimator trajectories
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(integers1toN, I_prime_n.T, color="b")

        ax.set_xlim(0, N)
        ax.set_ylim(1.0, 1.3)
        ax.axhline(Esp_gY, color="r", label="Expectation")
        ax.legend(loc="best")
        plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

Question 3: Proportional Stratification

Propose a proportional stratification technique based on simulating uniform variables on $[-1,0]$ and uniform variables on $[0,1]$.

Let J_n be the corresponding estimator. Plot the estimator trajectories as a function of n. Does this estimator satisfy the Central Limit Theorem (CLT)? What is the gain in terms of the number of simulations for this estimator, for the same asymptotic accuracy as the naive Monte Carlo method?

Can we explicitly calculate $\mathrm{Var}(J_n)$? (see question 5)

In [14] :

```
        # Proportionnal stratification (we will take N even)
        N1 = int(N/2)
        N2 = N1

        ############################################
        X1 = -1.0 + 1.0 * np.random.rand(N1) # U[-1,0]
        X2 = 0.0 + 1.0 * np.random.rand(N2) # U[0,1]
        ############################################

        ############################################
        # J_N = 0.5*moyenne strate1 + 0.5*moyenne strate2
        J_N = 0.5 * np.mean(np.exp(X1)) + 0.5 * np.mean(np.exp(X2))

        # variance asymptotique estimee : Var(J_N)*N avec N=N1+N2
        v1 = np.var(np.exp(X1), ddof=1); v2 = np.var(np.exp(X2), ddof=1)
        var = 0.25 * v1 * (N / N1) + 0.25 * v2 * (N / N2) # = (v1+v2)/2 en proportionnel
        # soit var_asymp = 0.5*(v1+v2)/2... on reporte var*N-normalise :
        var = (v1 + v2) / 4 * 2 # forme simple : (v1+v2)/2 /2 ? on garde estimateur direct :
        var = (v1 + v2) / 4 * 2

        halfWidthCI = 1.96 * np.sqrt(var / N)
        relativeError = halfWidthCI / J_N

        print("Estimator by proportionnal stratification \n")

        print("E[g(Y)] = %1.3f Var[g(Y)] = %1.3f" %(Esp_gY, Var_gY))
        print("J_N = %1.3f Estimated Variance = %1.3f" %(J_N, var))
        print("CI = [%1.3f,%1.3f] \n" %(J_N - halfWidthCI, J_N + halfWidthCI))
        print("Relative Error = %1.3f" %relativeError)

        print("Savings in the number of simulations compared to MC: %1.2f" %(Var_gY/var))
        print("CLT : oui, somme de 2 moyennes independantes -> normalite asymptotique.")
```

```
Estimator by proportionnal stratification

E[g(Y)] = 1.175 Var[g(Y)] = 0.432
J_N = 1.182 Estimated Variance = 0.132
CI = [1.159,1.204]

Relative Error = 0.019
Savings in the number of simulations compared to MC: 3.28
CLT : oui, somme de 2 moyennes independantes -> normalite asymptotique.
```

In [15] :

```
        ##############################################
        ## Trajectories of the stratified estimator J_n
        ## for n = 2, 4, ..., N
        ##############################################
        M = int(10)

        X1 = -1.0 + 1.0 * np.random.rand(M, N1)
        X2 = 0.0 + 1.0 * np.random.rand(M, N2)
        ##############################################

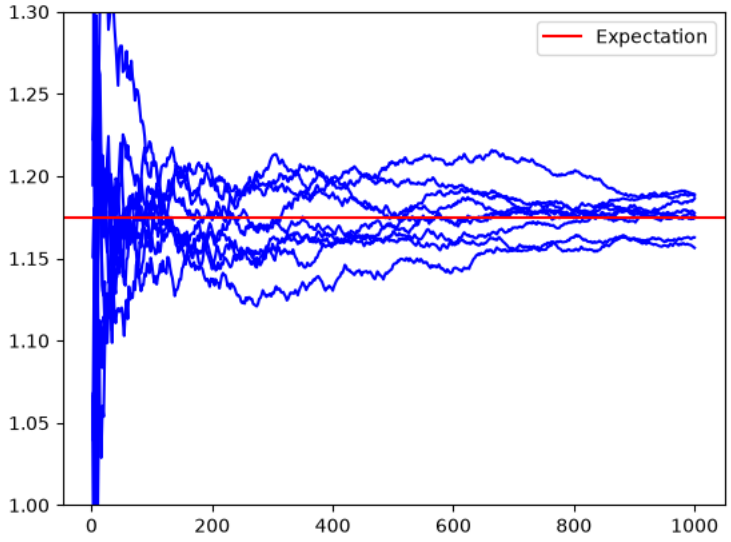
        J_n = np.zeros((M, int(N/2)))

        for n in np.arange(2, N+2, 2):
        n1 = n // 2
        J_n[:, n1-1] = 0.5*np.mean(np.exp(X1[:, :n1]), axis=1) + 0.5*np.mean(np.exp(X2[:, :n1]),
        axis=1)

        ##############################################
        evenIntegers1toN = np.arange(2, N+2, 2)

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(evenIntegers1toN, J_n[1:10, :].T, color="b")

        ax.set_ylim(1.0, 1.3)
        ax.axhline(Esp_gY, color="r", label="Expectation")
        ax.legend(loc="best")
        plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

Question 4: Non-Proportional Stratification (Optional)

Propose a stratification technique based on simulating a proportion $n_1=rn$ of uniform variables on $[-1,0]$, and $n_2=(1-r)n$ of uniform variables on $[0,1]$, with $r \in (0,1)$ such that n_1 and n_2 are integers.

Plot the trajectories of these estimators as a function of n for various values of r.

In [16] :

```
# Non-Proportional Stratification

def nonProportionalStratification(r):
N1 = int(N * r)
N2 = int(N - N1)

X1 = -1.0 + 1.0 * np.random.rand(N1) # U[-1,0]
X2 = 0.0 + 1.0 * np.random.rand(N2) # U[0,1]

# poids des strates p1=p2=1/2
J_N = 0.5 * np.mean(np.exp(X1)) + 0.5 * np.mean(np.exp(X2))

v1 = np.var(np.exp(X1), ddof=1); v2 = np.var(np.exp(X2), ddof=1)
# Var(J_N) = p1^2 v1/N1 + p2^2 v2/N2 ; var reporte = N*Var(J_N)
var = N * (0.25 * v1 / N1 + 0.25 * v2 / N2)

halfWidthCI = 1.96 * np.sqrt(var / N)

relativeError = halfWidthCI / J_N

print("Non-Proportional Stratification Estimator, r = %1.2f \n" %r)

print("E[g(Y)] = %1.4f Var[g(Y)] = %1.4f" %(Esp_gY, Var_gY))
print("J_N = %1.4f var*N = %1.4f" %(J_N, var))
print("CI = [%1.4f,%1.4f] \n" %(J_N - halfWidthCI, J_N + halfWidthCI))

print("Relative Error = %1.4f" %relativeError)
print("Savings in the number of simulations compared to MC : %1.4f" %(Var_gY/var))

M = 10
X1m = -1.0 + 1.0 * np.random.rand(M, N1)
X2m = 0.0 + 1.0 * np.random.rand(M, N2)

Y1 = np.exp(X1m); Y2 = np.exp(X2m)

J_n = np.zeros((M, N))

for n in np.arange(int(1/r), N):
n1 = int(n * r)
n2 = n - n1
n1 = min(n1, N1); n2 = min(n2, N2)
J_n[:, n] = 0.5*np.mean(Y1[:, :n1], axis=1) + 0.5*np.mean(Y2[:, :n2], axis=1)

integers1toN = np.arange(1, N+1)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(integers1toN, J_n[1:10, :].T, color="b")

ax.set_ylim(1.0, 1.3)
ax.axhline(Esp_gY, color="r", label="Expectation")
ax.legend(loc="best")
plt.show()

#nonProportionalStratification(0.8)
#nonProportionalStratification(0.4)
#nonProportionalStratification(0.2)
```

Question 5: Optimal stratification on both strata$(S_1,S_2)=([-1,0],[0,1])$ (Optional)

Calculate the values of the parameters $$ \sigma_1^2:=\mathbb{E}\bigl[e^{2X}~|~X\in[-1,0]\bigl] - \mathbb{E}\bigl[e^{X}~|~X\in[-1,0]\bigl]^2\\ \sigma_2^2:=\mathbb{E}\bigl[e^{2X}~|~X\in[0,1]\bigl] - \mathbb{E}\bigl[e^{X}~|~X\in[0,1]\bigl]^2 $$ and propose an optimal stratification technique. Simulate the empirical estomator associated and plot its trajectories.

In [17] :

```
        ## Explicit Calculation of sigma_1 and sigma_2
        sigma_1 = np.sqrt(0.5 * (1. - np.exp(-2.)) - (1. - np.exp(-1.))**2)
        sigma_2 = np.sqrt(0.5 * (np.exp(2.) - 1.) - (np.exp(1.) - 1.)**2)

        ################################################
        # Allocation optimale de Neyman : n1 proportionnel a p1*sigma1
        r = float(sigma_1 / (sigma_1 + sigma_2))
        print("r_opt =", r)

        N1 = int(N * r)
        N2 = int(N - N1)

        X1 = -1.0 + 1.0 * np.random.rand(N1)
        X2 = 0.0 + 1.0 * np.random.rand(N2)

        ################################################
        # Stratified estimator
        J_N = 0.5 * np.mean(np.exp(X1)) + 0.5 * np.mean(np.exp(X2))

        ################################################
        # variance theorique : N*Var(J) = p1^2 s1^2/r + p2^2 s2^2/(1-r)
        var = 0.25 * sigma_1**2 / r + 0.25 * sigma_2**2 / (1 - r)
        theoretical_variance = var


        halfWidthCI = 1.96 * np.sqrt(theoretical_variance / N)
        relativeError = halfWidthCI / J_N

        ################################################
        print("Estimator by optimal non-proportional stratification, r_optimal = %1.3f" % r)

        print("E[g(Y)] = %1.3f Var[g(Y)] = %1.3f" % (Esp_gY, Var_gY))
        print("J_mean = %1.3f theoretical_variance = %1.3f" % (J_N, var))
        print("CI = [%1.3f,%1.3f] \n" % (J_N - halfWidthCI, J_N + halfWidthCI))

        print("Relative Error = %1.3f" % relativeError)
        print("Savings in the number of simulations compared to MC: %1.2f" % (Var_gY / var))
```

```
r_opt = 0.26894142136999494
Estimator by optimal non-proportional stratification, r_optimal = 0.269
E[g(Y)] = 1.175 Var[g(Y)] = 0.432
J_mean = 1.178 theoretical_variance = 0.113
CI = [1.158,1.199]

Relative Error = 0.018
Savings in the number of simulations compared to MC: 3.82
```

In [18] :

```
        ###########################################
        M = 10
        X1m = -1.0 + 1.0 * np.random.rand(M, N)
        X2m = 0.0 + 1.0 * np.random.rand(M, N)
        Y1 = np.exp(X1m); Y2 = np.exp(X2m)
        J_n = np.zeros((M, N))

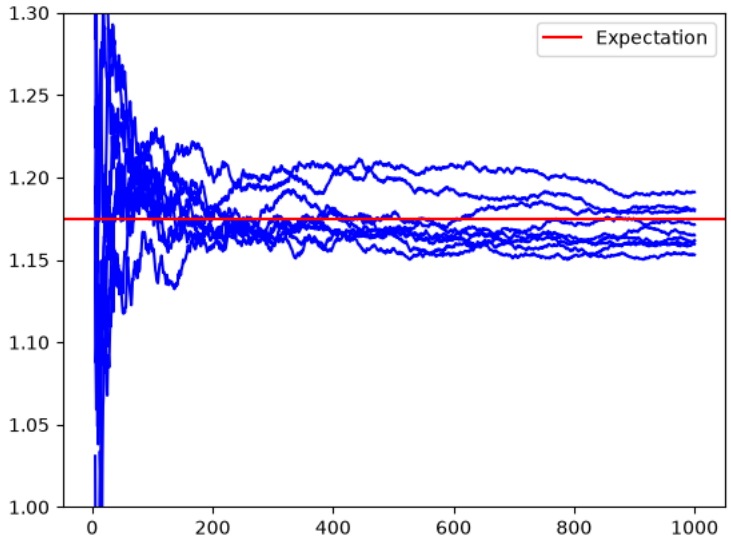
        for n in np.arange(int(1 / r), N):
        n1 = int(n * r)
        n2 = n - n1
        J_n[:, n] = 0.5*np.mean(Y1[:, :n1], axis=1) + 0.5*np.mean(Y2[:, :n2], axis=1)

        ###########################################
        # Displaying the trajectories
        integers1toN = np.arange(1, N + 1)

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(integers1toN, J_n[1:10, :].T, color="b")

        ax.set_ylim(1.0, 1.3)
        ax.axhline(Esp_gY, color="r", label="Expectation")
        ax.legend(loc="best")
        plt.show()
```

```
C:\Users\ayoub\AppData\Local\Programs\Python\Python311\Lib\site-packages\numpy\_core\fromnumeric.p
y:3824: RuntimeWarning: Mean of empty slice
return _methods._mean(a, axis=axis, dtype=dtype,
C:\Users\ayoub\AppData\Local\Programs\Python\Python311\Lib\site-packages\numpy\_core\_methods.py:1
34: RuntimeWarning: invalid value encountered in divide
ret = um.true_divide(
```




```
<Figure size 640x480 with 1 Axes>
```