TP3 Correction

February 14, 2024

1 Méthode de Monte Carlo

Tout d'abord, on illustre numériquement les deux résultats probabilistes sur lesquels reposent la méthode dite de Monte Carlo:

- la loi forte de grands nombres,
- le théorème central limit (TCL).

On considère ensuite un premier exemple d'estimateur de Monte Carlo et l'importance de l'intervalle de confiance (IC) dans lequel se trouve la valeur recherchée avec probabilité grande (0.95).

Enfin on applique la méthode de Monte Carlo à un exemple multidimensionnel où on illustre l'efficacité de 2 méthodes de réduction de variance:

- variables antithétiques,
- variable de contrôle.

[1]:
```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
from numpy.random import default_rng
rng = default_rng()
```

1.1 Illustration de la loi des grands nombres

Soit $(X_n)_{n\geq 1}$ une suite de variables aléatoires *i.i.d.* de carré intégrable. On définit les suites $(m_n)_{n\geq 1}$ et $(\sigma_n^2)_{n\geq 2}$ (non définie pour $n=1$) de la façon suivante

$$m_n = \frac{1}{n}\sum_{k=1}^n X_k \qquad \text{et} \qquad \sigma_n^2 = \frac{1}{n-1}\sum_{k=1}^n (X_k - m_n)^2 \quad \text{pour } n \geq 2$$

et on veut illustrer la Loi Forte des Grands Nombres et le Théorème Central Limite (étendu en utilisant le lemme de Slutsky pour remplacer $\sigma^2 = \mathrm{var}(X_1)$ par l'estimateur σ_n^2) c'est à dire les convergences

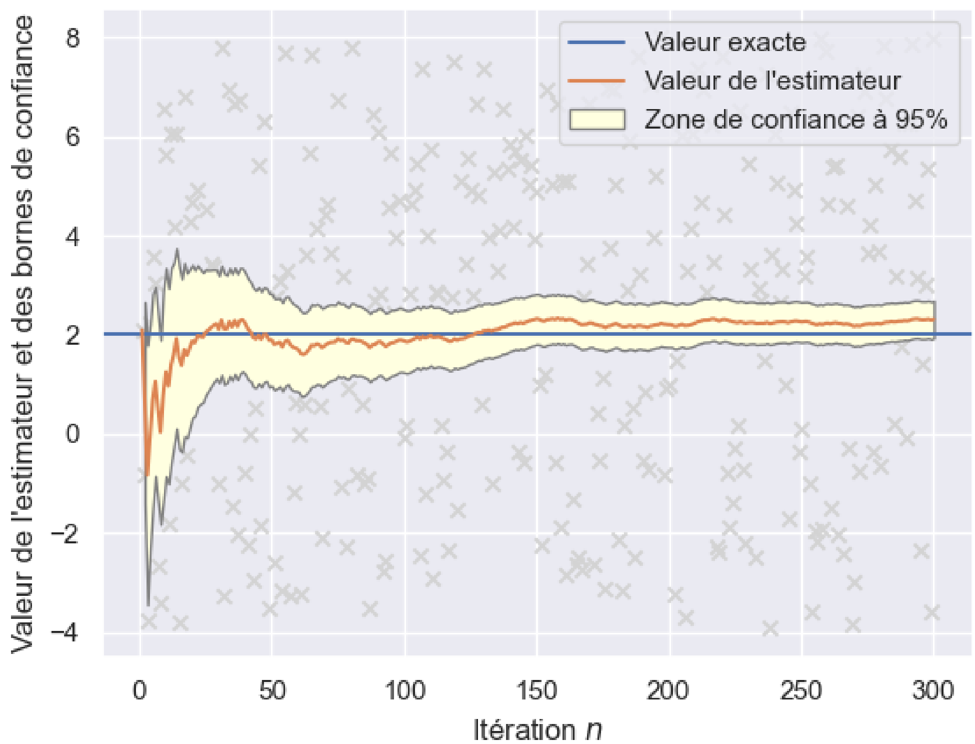
$$m_n \xrightarrow{p.s.} m \qquad \text{et} \qquad \sqrt{n}\Big(\frac{m_n - m}{\sigma_n}\Big) \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

Plus précisément on construit l'intervalle de confiance (asymptotique) à 95% à partir du TCL c'est à dire

$$\text{pour } n \text{ grand} \quad \mathbf{P}\Big(m \in \Big[m_n - \frac{1.96\sigma_n}{\sqrt{n}}, m_n + \frac{1.96\sigma_n}{\sqrt{n}}\Big]\Big) \simeq 0.95$$

1.1.1 Question: LFGN loi uniforme

Reproduire le tracé suivant où les points (les croix 'x') sont les réalisations X_n (en fonction de n) de loi uniforme sur $[-4, 8]$. La ligne bleue (couleur 'C0', première couleur de la palette utilisée) correspond à la moyenne m, la courbe orangée (couleur 'C1') correspond à la suite m_n et les lignes grises correspondent aux bornes de l'intervalle de confiance. La zone de confiance en jaune s'obtient par la méthode `fill_between` de `ax`.

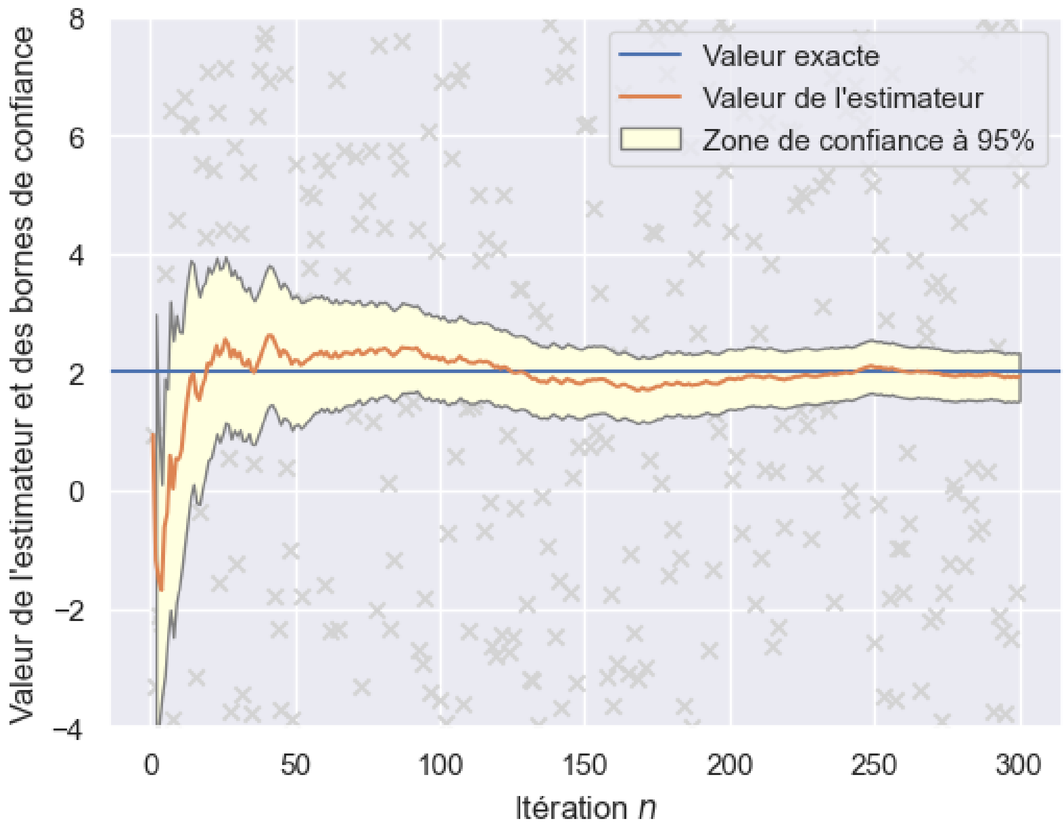


[2]:
```
N = 300
sample = rng.uniform(size = N, low = -4, high = 8)

n = np.arange(1, N+1)
mn = np.cumsum(sample) / n
sum_squares = np.cumsum(sample**2)
# attention: les vecteurs vn, ic, upper et lower sont définis pour n >= 2
vn = (sum_squares - n*mn**2)[1:] / (n[1:]-1)  # on développe le carré
ci_size = 1.96*np.sqrt(vn / n[1:])
upper = mn[1:] + ci_size
```

```
lower = mn[1:] - ci_size

fig, ax = plt.subplots()
ax.scatter(n, sample, marker="x", color='lightgrey')
ax.axhline(y=2, color='C0', label="Valeur exacte")
ax.plot(n, mn, color='C1', label="Valeur de l'estimateur")
ax.fill_between(n[1:], lower, upper, facecolor='lightyellow',
                edgecolor='grey', label="Zone de confiance à 95%")
ax.set(xlabel = "Itération $n$",
       ylabel = "Valeur de l'estimateur et des bornes de confiance")
ax.legend(loc='upper right')
ax.set_ylim(-4, 8)
#plt.savefig('img/tcl_unif.png')
plt.show()
```



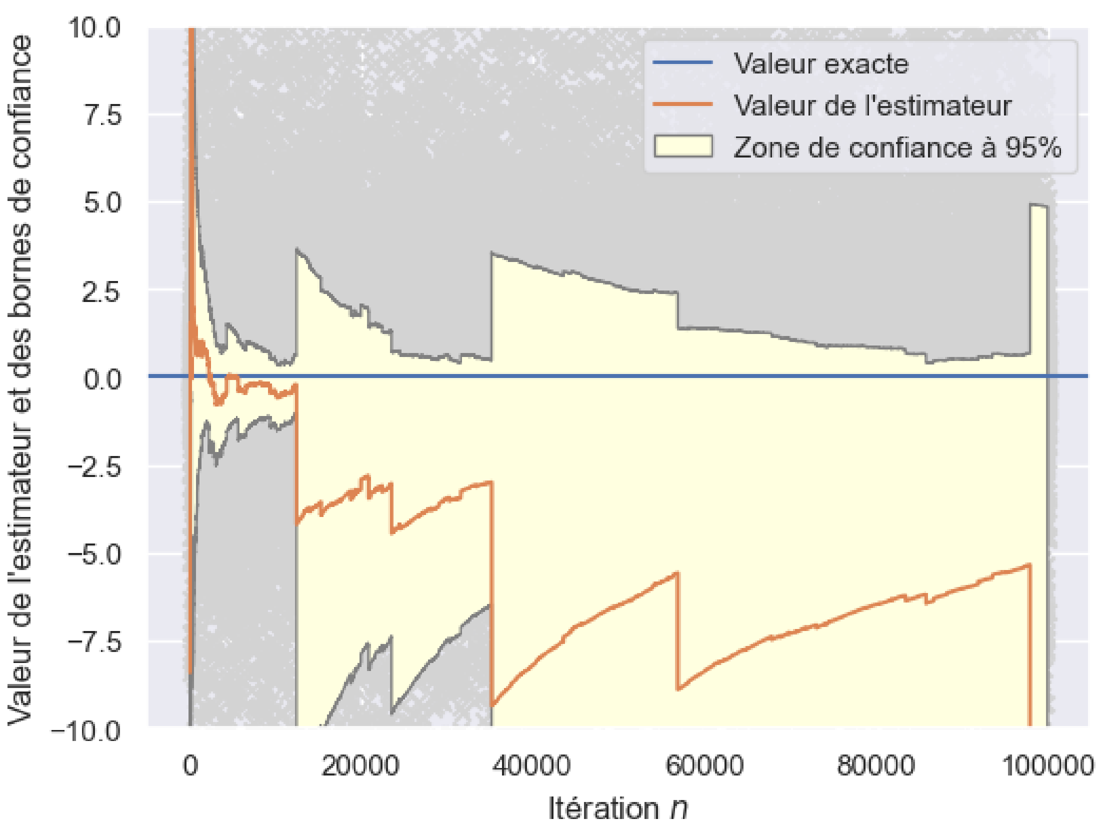
1.1.2 Question: LFGN loi de Cauchy

Reprendre rapidement l'exemple précédent en remplaçant la loi uniforme par la loi de Cauchy. On obtient des réalisations de la loi de Cauchy en utilisant la méthode `standard_cauchy` de l'objet `rng`. Répliquer plusieurs fois le tracé (avec l'axe des ordonnées restreint à $[-10, 10]$) pour différentes valeurs de $n = 100\,000$. Qu'en pensez-vous?

[3]:
```
N = 100000
sample = rng.standard_cauchy(size = N)

n = np.arange(1, N+1)
mn = np.cumsum(sample) / n
sum_squares = np.cumsum(sample**2)
# attention: les vecteurs vn, ic, upper et lower sont définis pour n >= 2
vn = (sum_squares - n*mn**2)[1:] / (n[1:]-1)
ci_size = 1.96*np.sqrt(vn / n[1:])
upper = mn[1:] + ci_size
lower = mn[1:] - ci_size

fig, ax = plt.subplots()
ax.scatter(n, sample, marker="x", color='lightgrey')
ax.axhline(y=0, color='C0', label="Valeur exacte")
ax.plot(n, mn, color='C1', label="Valeur de l'estimateur")
ax.fill_between(n[1:], lower, upper, facecolor='lightyellow',
                edgecolor='grey', label="Zone de confiance à 95%")
ax.set(xlabel = "Itération $n$",
       ylabel = "Valeur de l'estimateur et des bornes de confiance")
ax.legend(loc='upper right')
ax.set_ylim((-10,10))
plt.show()
```



1.2 Illustration du TCL

On veut illustrer la répartition de l'erreur renormalisée $\varepsilon_n = \sqrt{n}\Big(\frac{m_n - m}{\sigma_n}\Big)$ pour différentes valeurs de n. Lorsque n est grand cette erreur renormalisée est proche de la loi normale cenrée réduite, c'est ce qu'on veut vérifier numériquement. Pour illustrer cette répartition, il est nécessaire de répliquer un grand nombre de fois l'erreur c'est à dire de considérer un échantillon $(\varepsilon_n^{(j)})_{j=1,\dots,M}$ de taille M et de constuire l'histogramme de cet échantillon.

Attention: en pratique il n'est pas nécessaire de répliquer M fois l'estimateur m_n pour approcher m. L'estimateur de la variance v_n suffit pour donner la zone de confiance autour de m_n. C'est une information importante donnée par le TCL.

1.2.1 Question: TCL loi uniforme

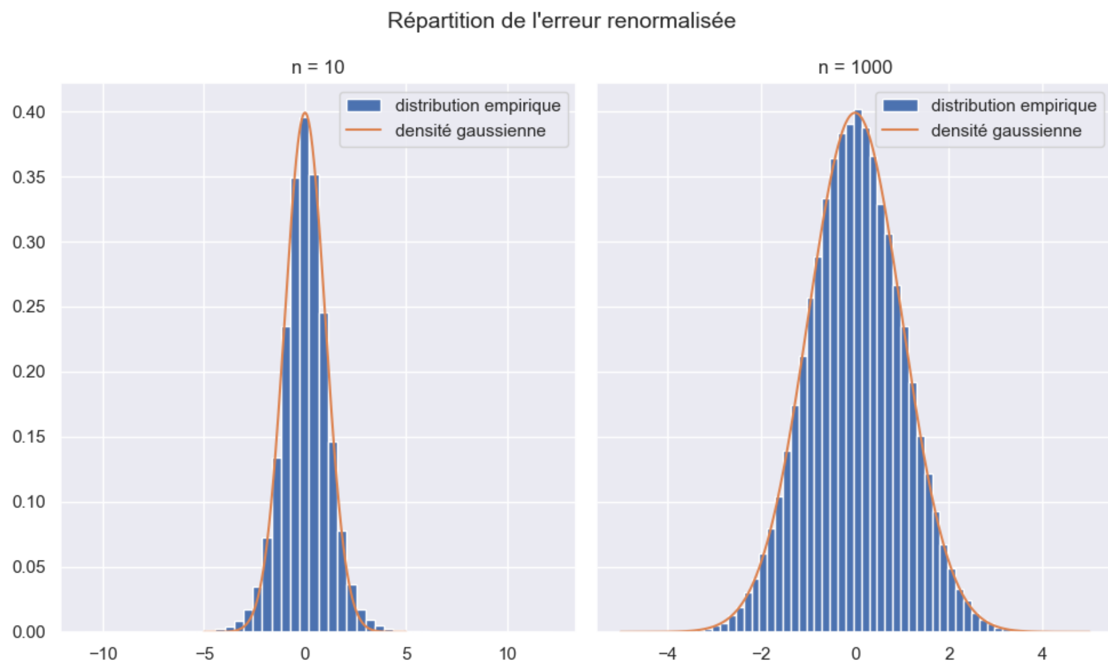
Dans le cas de la loi uniforme sur $[-4, 8]$ vérifier la répartition de l'erreur renormalisée ε_n pour $n = 10$ puis $n = 1\,000$ à partir d'un échantillon de taille $M = 100\,000$.

[4]:
```
def plot_error(ax, n, M = 100000):
    sample = rng.uniform(size = (M, n), low = -4, high = 8)

    means = np.mean(sample, axis = 1)
    sigms = np.std(sample, axis = 1, ddof=1)
    errs = np.sqrt(n) * (means - 2) / sigms

    ax.hist(errs, bins=50, density=True, label="distribution empirique")
    xx = np.linspace(-5, 5, 10000)
    ax.plot(xx, stats.norm.pdf(xx), label="densité gaussienne")
    ax.set(title = f"n = {n}")
    ax.legend()
    return ax

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey=True,
                               figsize=(10,6), layout='tight')
fig.suptitle("Répartition de l'erreur renormalisée", fontsize=14)
plot_error(ax1, n=10)
plot_error(ax2, n=1000)
plt.show()
# pour N = 10 la répartition de l'erreur ne semble pas vraiment gaussienne
# pour N = 1000 le comportement semble gaussien
```



[ ]:

```

```

1.3 Un premier exemple d'estimateur de Monte Carlo

On va mettre en oeuvre un estimateur de Monte Carlo pour calculer

$$I(\beta) = \mathbf{E}[\exp(\beta G)] \quad \text{où } G \sim \mathcal{N}(0,1) \text{ et } \beta \in \mathbf{R}.$$

La valeur exacte $I(\beta) = \exp(\beta^2/2)$ est connue mais cet exemple permet d'illustrer l'importance des bornes de l'intervalle de confiance (et donc de l'estimation de la variance) dans une méthode de Monte Carlo. La seule valeur moyenne $I_n = \frac{1}{n}\sum_{k=1}^{n} X_k$ n'est pas suffisante pour déterminer I.

1.3.1 Question: fonction `monte_carlo`

Ecrire une fonction `monte_carlo(sample, proba=0.95)` qui à partir d'un échantillon `sample` de réalisation indépendantes $(X_k)_{k=1,\dots,n}$ renvoie un tuple qui contient:

- la moyenne de l'estimateur Monte Carlo de $I = \mathbf{E}[X]$,
- l'estimateur de la variance asymptotique apparaissant dans le TCL,
- les bornes inférieures et supérieures de l'intervale de confiance de niveau de probabilité `proba`.

[5]:

```python
def monte_carlo(sample, proba = 0.95):
    """
    Computes the mean, variance, and a 95% confidence interval of a
    given sample data set using the Monte Carlo method.
    Parameters:
```

```
    -----------
    sample : array-like
        The data set to be analyzed
    proba : float, optional
        The probability that the true mean of the population is
        within the calculated interval. Default is 0.95
    Returns:
    --------
    tuple : float
        The mean, variance, lower bound of the 95% CI and upper bound of the␣
↪95% CI
    """
    mean = np.mean(sample)
    var = np.var(sample, ddof=1)
    alpha = 1 - proba
    quantile = stats.norm.ppf(1 - alpha/2)  # fonction quantile
    ci_size = quantile * np.sqrt(var / sample.size)
    return (mean, var, mean - ci_size, mean + ci_size)
```

1.3.2 Question: premier exemple

En utilisant la fonction `monte_carlo`, reproduire le tableau suivant où chaque ligne représente un résultat pour une valeur de $\beta \in \{0.2, 0.5, 1, 2, 3, 5\}$:

- la première colonne est la valeur moyenne I_n,
- la deuxième colonne l'estimateur de la variance,
- les colonnes 3 et 4 sont les bornes inférieures et supérieurs de l'IC à 95%,
- la colonne 5 contient la valeur exacte $\mathbf{E}[\exp(\beta G)] = \exp(\beta^2/2)$.

Ce tableau est obtenu pour $n = 1\,000\,000$. Comment interpréter ce tableau?

[6]:
```python
import pandas as pd
df = pd.read_pickle("data/first_df.pkl")
df
```

[6]:

	mean	var	low	high	exact
0.2	1.020551	4.245536e-02	1.020147	1.020955	1.020201
0.5	1.134027	3.650418e-01	1.132843	1.135212	1.133148
1.0	1.651060	4.676691e+00	1.646821	1.655298	1.648721
2.0	7.402685	2.379946e+03	7.307068	7.498301	7.389056
3.0	87.915075	8.558333e+06	82.181273	93.648877	90.017131
5.0	121963.825619	6.439313e+14	72228.169429	171699.481809	268337.286521

[7]:
```python
n = int(1e6)
sample = rng.standard_normal(size=n)

betas = [0.2, 0.5, 1, 2, 3, 5]
result = [ monte_carlo(np.exp(beta * sample)) for beta in betas ]
```

```python
# results est une liste de tuple on peut le convertir en DataFrame
# pour manipuler plus facilement ce résultat
import pandas as pd
res_df = pd.DataFrame(result,
                      columns=["mean", "var", "lower", "upper"],
                      index=betas)
res_df["exact"] = np.exp(0.5 * np.array(betas)**2)
res_df
#res_df.to_pickle("data/first_df.pkl")
```

[7]:

	mean	var	lower	upper	exact
0.2	1.019807	4.235379e-02	1.019404	1.020210	1.020201
0.5	1.131875	3.626613e-01	1.130694	1.133055	1.133148
1.0	1.643802	4.568327e+00	1.639612	1.647991	1.648721
2.0	7.270406	1.956263e+03	7.183717	7.357094	7.389056
3.0	81.991614	5.095386e+06	77.567395	86.415834	90.017131
5.0	85058.933457	2.794353e+14	52295.549482	117822.317432	268337.286521

[ ]:

1.4 Option panier: un exemple multidimensionnel

On considère $d \geq 2$ actifs financiers dont la loi à l'instant $T > 0$ est modélisée par une loi log-normale c'est à dire

$$\forall i \in \{1, \dots, d\}, \quad S_T^i = S_0^i \exp\Big((r - \frac{\sigma_i^2}{2})T + \sigma_i \sqrt{T} \tilde{G}_i\Big)$$

où le vecteur $(\tilde{G}_1, \dots, \tilde{G}_d)$ est gaussien centré de matrice de covariance Σ et les constantes $r > 0$, $\sigma_i > 0$ sont fixées. Il s'agit d'actifs financiers $(S_t^i)_{t \in [0,T]}$, $1 \leq i \leq d$, modélisés par un processus de Black-Scholes multidimensionnel. On introduit la matrice L triangulaire inférieure obtenue par la décomposition de Cholesky de la matrice $\Sigma = LL^\top$.

A l'aide de cette matrice L, on définit la fonction $\Phi : \mathbf{R}^d \to \mathbf{R}^d$ telle que

$$(S_T^1, \dots, S_T^d) = \Phi(G_1, \dots, G_d) \quad \text{ou encore} \quad S_T^i = \Phi_i(G_1, \dots, G_d)$$

où $(G_1, \dots, G_d) \sim \mathcal{N}(0, I_d)$ (l'égalité précédente est à considérer en loi).

On s'intéresse au prix d'une option européenne (aussi appelé produit dérivé européen) sur le panier de ces d actifs financiers, c'est à dire qu'on veut calculer

$$\mathbf{E}[X] \quad \text{avec} \quad X = \left(\frac{1}{d} \sum_{i=1}^d S_T^i - K\right)_+.$$

1.4.1 Question: initialisation

Définir les paramètres globaux $d = 10$, $T = 1$, $r = 0.01$, $S_0^i = 100$ (pour tous les actifs), $\sigma_i = i/(2d)$ (on dit que certains actifs sont plus volatiles que d'autres) et la matrice de corrélation Σ définie par $\Sigma_{i,i} = 1$ et $\Sigma_{i,j} = \rho \in [0, 1]$ pour $i \neq j$, avec $\rho = 0.2$.

Initialiser la matrice L en utilisant la fonction `np.linalg.cholesky`.

[8]:
```python
d = 10
T = 1
r = 0.01
S0 = np.full(d, 100)
sigma = np.arange(1,d+1)/(2*d)
mu = r - 0.5*sigma**2
rho = 0.2
correl = np.full((d,d), rho) + (1-rho)*np.eye(d) #ou np.diag(np.full(d, 1-rho))
mat_L = np.linalg.cholesky(correl)
```

1.4.2 Question: simulation d'un échantillon d'actifs

Définir la fonction python `phi` qui transforme le vecteur $(G_1, \dots, G_d)$ en un vecteur $(S_T^1, \dots, S_T^d)$ (tous les paramètres sont des variables globales pour simplifier l'écriture du code). L'appel suivant doit fonctionner

```python
G = rng.standard_normal(size=d)
phi(G)
```

Si on veut implémenter un estimateur Monte Carlo il faut travailler avec des échantillons *i.i.d.* $(S_T^{(j)})_{j=1,\dots,n}$ où $S_T^{(j)} = (S_T^{(j),1}, \dots, S_T^{(j),d}) \in \mathbf{R}^d$. Modifier votre fonction `phi` pour création un tel échantillon à partir de l'appel suivant:

```python
sample_G = rng.standard_normal(size=(d, n))
phi(sample_G)
```

(il faut utiliser la technique du broadcasting en `numpy`, c'est très important à connaitre en pratique).

[9]:
```python
# première version
def phi(G):
    ST = S0 * np.exp(mu * T + sigma * np.sqrt(T) * mat_L @ G)
    return ST

G = rng.standard_normal(size=d)
print(phi(G))

# deuxième version pour obtenir un échantillon de taille `n`
def phi(sample_G):
    sample_ST = S0[:,np.newaxis] * np.exp(mu[:,np.newaxis] * T
                + sigma[:,np.newaxis] * np.sqrt(T) * mat_L @ sample_G)
    return sample_ST

n = 1000
sample_G = rng.standard_normal(size=(d, n))
print(phi(sample_G))
```

```
[ 94.46561942  98.98848327 103.55798932 122.23035497 138.85370456
 143.39926291 155.81386751  76.0757536   73.13553358  36.7905925 ]
[[ 97.54213834  95.99813341 101.9211037  … 105.51049372 106.85176668
   94.07152923]
```

```
[ 90.2871203  104.71665292  87.6644466  … 105.0711644  110.39150863
 100.64374365]
[ 85.01463629  87.99250314  98.49716662 … 119.77020243 100.6531541
 104.87699285]
…
[ 47.73845232  54.08737554  78.25195845 …  60.80592444 129.42325901
 297.06158733]
[119.4591629   88.40274467  63.5111637  …  70.33656346  67.40667413
 179.67362932]
[228.6792739   55.86163036  86.98766951 …  81.23000645 114.18498345
 131.41225884]]
```

1.4.3 Question: estimateur Monte Carlo

Définir une fonction $\psi : \mathbf{R}^d \times \mathbf{R}_+ \to \mathbf{R}_+$ telle que

$$\psi(G_1, \dots, G_d, K) = \left(\frac{1}{d} \sum_{i=1}^{d} \Phi_i(G_1, \dots, G_d) - K \right)_+$$

dans une fonction **python** appelée **psi**. Cette fonction doit fonctionner avec un échantillon $(G_1^{(j)}, \dots, G_d^{(j)})_{j=1,\dots,n}$.
Ecrire et programmer l'estimateur de Monte Carlo pour estimer la quantité $\mathbf{E}[X] = \mathbf{E}[\psi(G_1, \dots, G_d, K)]$ où $(G_1, \dots, G_d) \sim \mathcal{N}(0, I_d)$.

Pour différentes valeur de $K \in \{80, 90, 100, 110, 120\}$ et $n = 100\,000$ vous devez obtenir le tableau suivant:

[10]:
```python
df = pd.read_pickle("data/basket_mc.pkl")
df
```

[10]:

	mean	var	lower	upper
80	21.394471	228.318772	21.300818	21.488123
90	12.860460	181.187947	12.777032	12.943889
100	6.655165	111.553749	6.589702	6.720627
110	2.998650	54.132985	2.953049	3.044252
120	1.204158	21.976278	1.175102	1.233213

[11]:
```python
def psi(sample_G, K):
    return np.maximum(phi(sample_G).mean(axis=0) - K, 0)

n = int(1e5)
sample_G = rng.standard_normal(size=(d, n))
Ks = [80, 90, 100, 110, 120]
result = [ monte_carlo(psi(sample_G, K)) for K in Ks ]

df_mc = pd.DataFrame(result,
                     columns=['mean', 'var', 'lower', 'upper'],
                     index=Ks)
df_mc
```

```
#df_mc.to_pickle('data/basket_mc.pkl')
```

[11]:

	mean	var	lower	upper
80	21.224518	225.464261	21.131453	21.317583
90	12.698037	178.572759	12.615213	12.780861
100	6.530757	109.361728	6.465941	6.595573
110	2.926126	52.730532	2.881119	2.971133
120	1.170381	21.265586	1.141800	1.198963

1.4.4 Question: variables antithétiques

Sur le même modèle que précédemment, implémenter la méthode de Monte Carlo avec réduction de variance par variables antithétiques c'est à dire basée sur la représentation:

$$\mathbf{E}[X] = \mathbf{E}\Big[\frac{1}{2}(\psi(G_1, \dots, G_d, K) + \psi(-G_1, \dots, -G_d, K))\Big]$$

Calculer le ratio de variance (variance de la méthode naïve divisée par variance par variables antithétiques) pour les différentes valeurs de K.
Que signifie ce ratio de variance?

[12]:

```
sample_G = rng.standard_normal(size=(d, n))
Ks = [80, 90, 100, 110, 120]
result = []
for K in Ks:
    sample = 0.5 * (psi(sample_G, K) + psi(-sample_G, K))
    result.append(monte_carlo(sample))

df_antith = pd.DataFrame(result,
                         columns=['mean', 'var', 'lower', 'upper'],
                         index=Ks)
df_antith
```

[12]:

	mean	var	lower	upper
80	21.300742	14.309432	21.277296	21.324187
90	12.771685	26.031820	12.740062	12.803308
100	6.584760	34.225363	6.548501	6.621020
110	2.965127	22.810522	2.935525	2.994728
120	1.202002	10.485939	1.181932	1.222072

[13]:

```
# le ratio des variances pour les différentes valeurs de K
df_mc["var"] / df_antith["var"]
```

[13]:

```
80     15.756339
90      6.859788
100     3.195342
110     2.311676
120     2.028010
Name: var, dtype: float64
```

1.5 Option panier: une variable de contrôle

Dans le cas de la dimension 1 ($d = 1$), le prix est donnée par une formule fermée, on appelle cette formule la formule de Black-Scholes. Pour une option Basket (en dimension $d \geq 2$) on approche le prix par Monte Carlo mais on peut utiliser des approximations pour construire un problème unidimensionnel proche du produit Basket. Ces approximations servent de variables de contrôles: **on ne rajoute pas une erreur, on retire de la variance**.

On rappelle que, en posant $\mu_i = r - \frac{1}{2}\sigma_i^2$, et $(\tilde{G}_1, \dots, \tilde{G}_d)$ un vecteur gaussien centré de matrice de covariance Σ,

$$X = \Big(\frac{1}{d} \sum_{i=1}^{d} S_0^i e^{\mu_i T + \sigma_i \sqrt{T} \tilde{G}_i} - K \Big)_+$$

et en introduisant $a_0^i = \frac{S_0^i}{\sum_{j=1}^d S_0^j}$ (t.q. $\sum a_0^i = 1$) et $\bar{S}_0 = \frac{1}{d}\sum_{i=1}^d S_0^i$ on a

$$X = \Big(\bar{S}_0 \sum_{i=1}^{d} a_0^i e^{\mu_i T + \sigma_i \sqrt{T} \tilde{G}_i} - K \Big)_+ .$$

La variable de contrôle proposée est obtenue en échangeant l'exponentielle et la moyenne pondérée par les poids $\big(a_0^i\big)_{i=1,\dots,d}$:

$$Y = \big(\bar{S}_0 e^Z - K\big)_+ \quad \text{avec} \quad Z = \sum_{i=1}^{d} a_0^i (\mu_i T + \sigma_i \sqrt{T} \tilde{G}_i)$$

La variable aléatoire Z suit une loi gaussienne $Z \sim \mathcal{N}(mT, s^2 T)$ avec

$$m = \sum_{i=1}^{d} a_0^i \mu_i \quad \text{et} \quad s^2 = \sum_{i=1}^{d} \Big(\sum_{j=1}^{d} a_0^i \sigma_i L_{ij} \Big)^2 .$$

Ainsi l'espérance de la variable de contrôle Y est connue par la formule de Black-Scholes, car elle correspond au prix d'un call de strike K d'un actif Black-Scholes de dimension 1, de valeur initiale $\bar{S}_0$, de taux $\rho = m + \frac{1}{2}s^2$ et de volatilité s (à un facteur d'actualisation près… attention à ça). On a donc

$$e^{-\rho T} \mathbf{E}[Y] = P_{\mathrm{BS}}(\bar{S}_0, \rho, s, T, K),$$

où

$$P_{\mathrm{BS}}(x, r, \sigma, T, K) = x F_{\mathcal{N}(0,1)}(d_1) - K e^{-rT} F_{\mathcal{N}(0,1)}(d_2),$$

avec $F_{\mathcal{N}(0,1)}$ est la fonction de répartition de la loi normale centrée réduite et la notation

$$d_1 = \frac{1}{\sigma \sqrt{T}} \Big(\log\big(\frac{x}{K}\big) + (r + \frac{\sigma^2}{2}) T \Big) \quad \text{et} \quad d_2 = d_1 - \sigma \sqrt{T}$$

1.5.1 Question: préliminaires pour la variable de contrôle

- Définir la fonction `price_call_BS` qui code la fonction $P_{\mathrm{BS}}(x, r, \sigma, T, K)$ définie ci-dessus.
- Initialiser les paramètres $\bar{S}_0$, $(a_0^i)_{i=1,\dots,d}$, m, s^2 et ρ.
- Calculer $\mathbf{E}[Y]$ par la formule fermée.
- Calculer $\mathbf{E}[Y]$ par un estimateur Monte Carlo à partir de réalisations de $(G_1^{(j)}, \dots, G_d^{(j)})$, $j \in \{1, \dots, n\}$.
- Vérifier que tout est cohérent.

[14]:
```
def price_call_BS(x, r, sigma, T, K):
    d1 = (np.log(x / K) + T * (r + 0.5*sigma**2)) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return x * stats.norm.cdf(d1) - K * np.exp(-r * T) * stats.norm.cdf(d2)
```

[15]:
```
barS0 = S0.mean()
a = S0 / S0.sum()
m = (a * (r - 0.5*sigma**2)).sum()
s2 = (((a * sigma).T @ mat_L)**2).sum()
rho = m + 0.5*s2
```

[16]:
```
# calcul par formule fermée
Y_mean = np.exp(rho*T) * price_call_BS(barS0, rho, np.sqrt(s2), T=T, K=K)
print("True value:", Y_mean)
```

```
True value: 0.6195106600831698
```

[17]:
```
# calcul par Monte Carlo
n = int(1e5)
sample_G = rng.standard_normal(size=(d, n))
Z = np.sum(a[:,None] * (m*T + sigma[:,None] * np.sqrt(T) * mat_L @ sample_G),
           axis = 0)
Y = np.maximum(S0.mean() * np.exp(Z) - K, 0)
monte_carlo(Y)
```

[17]:
```
(0.6192770360837057, 9.47947250481712, 0.6001943205355565, 0.6383597516318548)
```

1.5.2 Question: MC avec variable de contrôle

Implémenter l'estimateur de Monte Carlo avec variable de contrôle pour le calcul de $\mathbf{E}[X]$ c'est à dire

$$\mathbf{E}[X] = \mathbf{E}[\psi(G_1, \dots, G_d, K) - (Y - \mathbf{E}[Y])],$$

où Y est la variable de contrôle introduite précédemment et $\mathbf{E}[Y]$ est calculée par la formule fermée. Comparer les ratios de variance pour les différentes valeurs de K.

[18]:
```
Ks = [80, 90, 100, 110, 120]
result = []
for K in Ks:
    Z = np.sum(a[:,None] * (m*T + sigma[:,None]*np.sqrt(T)*mat_L@sample_G),
               axis = 0)
    Y = np.maximum(S0.mean() * np.exp(Z) - K, 0)
    Y_mean = np.exp(rho*T) * price_call_BS(barS0, rho, np.sqrt(s2), T=T, K=K)
    control_variate = Y - Y_mean
    sample = psi(sample_G, K) - control_variate
    result.append(monte_carlo(sample))

df_cv = pd.DataFrame(result,
                     columns=['mean', 'var', 'lower', 'upper'],
```

```
                            index=Ks)
df_cv
```

[18]:

```
          mean       var      lower      upper
80   21.308780  6.556240  21.292910  21.324650
90   12.779176  7.755879  12.761915  12.796437
100   6.608412  8.023739   6.590855   6.625968
110   2.978748  6.653731   2.962760   2.994735
120   1.207675  4.501630   1.194525   1.220825
```

[19]:

```
# le ratio des variances pour les différentes valeurs de K
df_mc["var"] / df_cv["var"]
```

[19]:

```
80     34.389261
90     23.024180
100    13.629771
110     7.924957
120     4.723975
Name: var, dtype: float64
```