# TP2 — Files d'attente M/M/1 (corrige)

Source : TP2/TP2_corrige.ipynb — PDF compile (code execute).

### MAP556, Ecole Polytechnique, 2023-24

## TP2 - Simulation de file d'attentes M/M/1 et M/M/c.

Dans la nomenclature de Kendall, les files d'attentes M/M/ correspondent à des temps d'arrivées successives indépendants distribués selon une loi exponentielle, et le temps de traitement d'un client dans une file d'attente suit un comportement similaire.

Le premier paramètre $\lambda$ est le nombre moyen d'arrivées par unité de temps. Le deuxième paramètre $\mu$ est le nombre moyen de clients traités par unité de temps.

Le but du TP est de calculer/simuler le nombre moyen de clients dans la file d'attente, et le temps moyen qu'un client reste dans le système avant d'être traité.

## Exercice 1. Simulation de variables aléatoires dans Python. Loi exponentielle.

### 1.1 Fonction de répartition.

### $\blacktriangleright$ Question [math+code] (a) : Expliquer comment à l'aide de la fonction de répartition $F$ de la loi exponentielle $\mathcal{E}(\lambda)$ et d'un échantillon $(U_i)_{1\leq i\leq N}$de $N$ variables i.i.d. de loi uniforme $\mathcal{U}(0,1)$ peut-on obtenir un échantillon $(E_i)_{1\leq i\leq N}$ de $N$ variables i.i.d. de loi exponentielle $\mathcal{E}(\lambda)$ de paramètre $\lambda$.

On pourra utiliser la fonction `numpy.random.rand` pour obtenir les tirages de la loi uniforme.

In [1] :

```
import numpy as np
import matplotlib.pyplot as plt
```

### Réponse [math] (a) :

Si $U\sim\mathcal{U}(0,1)$ et $F(x)=1-e^{-\lambda x}$ ($x\ge 0$) est la f.d.r. de $\mathcal{E}(\lambda)$, alors $E=F^{-1}(U)=-\log(1-U)/\lambda\sim\mathcal{E}(\lambda)$ (inversion). En pratique $1-U\stackrel{loi}{=}U$ donc $E=-\log(U)/\lambda$ convient aussi.

In [2] :

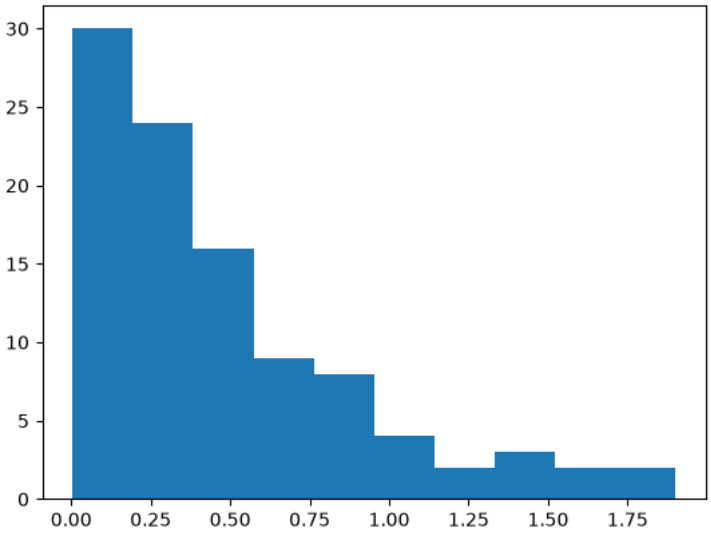
```
######################################################
# Réponse [code] (a) :
######################################################

N = 100 # Nombre de tirages
Uniform = np.random.rand(N) # Tirages independants de la loi uniforme sur [0,1]

######################################################
## Génération d'un échantillon de loi exponentielle (inversion)
param_lambda = 2
Expo = -np.log(1.0 - Uniform) / param_lambda # = -log(U)/lambda en loi

######################################################
## Affichage de l'échantillon
plt.hist(Expo, label="Histogramme de l'échantillon")

# On affiche la figure dans la console
plt.show()
```




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

## $\blacktriangleright$ Question [code] (a-bis) : Pour pouvoir être utilisé plus tard, on pourra encapsuler le code précédent dans deux fonctions avec comme paramètres le taux d'arrivée $\lambda$, le taux de traitement $\mu$, ainsi que la taille de l'échantillon.

In [3] :

```
        ##################################################
        # Réponse [code] (a-bis) :
        ##################################################

        def simule_temps_inter_arrivees(taux_arrivee, nombre_total_clients):
        return -np.log(1.0 - np.random.rand(nombre_total_clients)) / taux_arrivee

        def simule_temps_traitement(taux_traitement, nombre_total_clients):
        return -np.log(1.0 - np.random.rand(nombre_total_clients)) / taux_traitement
```

## $\blacktriangleright$ Question [code] (b) : Tracer une fonction de répartition empirique de l'échantillon obtenu pour vérifier l'adéquation de votre simulation avec la fonction de répartition théorique.

On rappelle que la fonction de répartition empirique pour un échantillon $(X_i)_{1\leq i\leq N}$ est défini par la formule suivante : $$ F_N := x \mapsto \frac{1}{N} \sum_{i=1}^N 1_{X_i \leq x} $$ pour tout $x\in\mathbb{R}$.

On notera que cet objet est une variable aléatoire contrairement à la fonction de répartition qui est un objet déterministe.

## Réponse [code] (b) :

On trie l'échantillon et on trace $F_N(x)=\frac{1}{N}\sum 1_{E_i\le x}$ en escalier, à comparer à $F(x)=1-e^{-\lambda x}$. L'écart sup se resserre quand $N$ grandit (Glivenko-Cantelli).

In [4] :

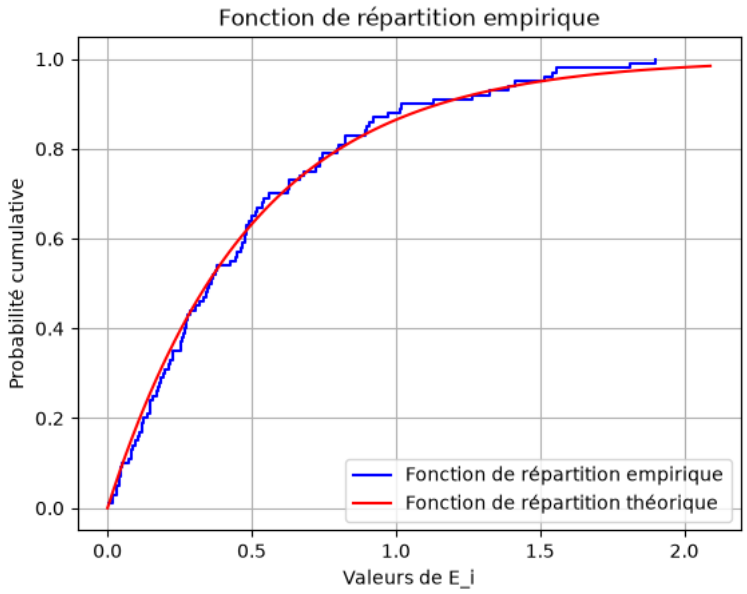
```
        ###################################################
        # Réponse [code] (b) :
        ###################################################

        ###################################################
        ## Calcul de la fonction de répartition empirique
        Expo_sorted = np.sort(Expo)
        F_emp = np.arange(1, N + 1) / N

        ###################################################
        ## Affichage des fonctions de répartition
        x = np.linspace(0, Expo_sorted.max() * 1.1, N)
        Fonc_Rep_Th = 1 - np.exp(-param_lambda * x)

        plt.step(Expo_sorted, F_emp, where='post', color="b", label="Fonction de répartition
        empirique")
        plt.plot(x, Fonc_Rep_Th, color="r", label="Fonction de répartition théorique")

        plt.xlabel('Valeurs de E_i')
        plt.ylabel('Probabilité cumulative')
        plt.title('Fonction de répartition empirique')
        plt.legend(loc="best")
        plt.grid(True)
        plt.show()
```




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

### $\blacktriangleright$ Question [math] (b - facultative - culture-générale) : Peut-on caractériser la convergence de $F_N$ vers $F$ ? Et surtout selon quelle notion de convergence ?

### Réponse [math] (b - facultative) :

Théorème de Glivenko-Cantelli : $\|F_N-F\|_\infty\to 0$ p.s. DKW : $P(\|F_N-F\|_\infty>\varepsilon)\le 2e^{-2N\varepsilon^2}$. Donc convergence uniforme presque sûre (et en probabilité) vers $F$.

## 1.2 Processus de Poisson et temps d'arrivées.

### $\blacktriangleright$ Question [code] (c) : Les variables exponentielles génèrent les temps entre deux arrivées successives $T_i$ et $T_{i+1}$ dans la file d'attente : si on note $T_0=0$, alors $T_{i+1}-T_{i} = E_i$, où les $(E_i)$ sont i.i.d. exponentielles de paramètres $\lambda >0$.

## Construire l'échantillon $(T_i)_{1\leq i\leq N}$ à l'aide de l'échantillon $(E_i)_{1\leq i\leq N}$ précédemment construit.

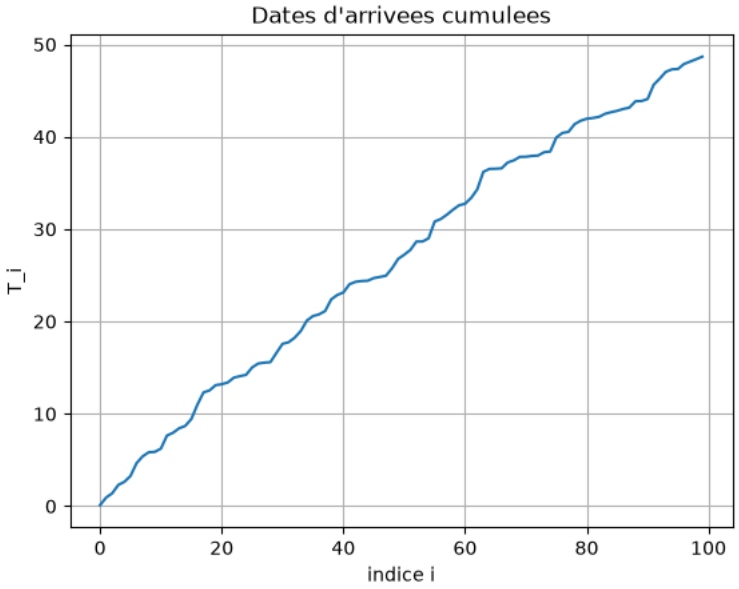
## Réponse [code] (c) :

$T_i=\sum_{j=1}^{i}E_j$ (cumsum). $T_i$ est la date d'arrivée du client $i$ (somme des inter-arrivées exponentielles = loi Erlang/Gamma).

In [5] :

```
##################################################
# Réponse [code] (c) :
##################################################

T_arrivees = np.cumsum(Expo) # T_i = somme des E_j, avec T_0 = 0
plt.plot(T_arrivees)
plt.xlabel("indice i"); plt.ylabel("T_i"); plt.title("Dates d'arrivees cumulees")
plt.grid(True); plt.show()
```




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

## $\blacktriangleright$ Question [math+code] (d) : Pouvez-vous donner un ordre de grandeur du comportement de $T_N$ lorsque $N$ tend vers $+\infty$ ? Simulez un échantillon de taille $M$ de variables aléatoires $T_N$ avec $N$ et $M$ grands pour vérifier votre conjecture. On pourra s'appuyer sur les résultats fournis par la loi forte des grand nombres et le théorème central limite.

## Réponse [math] (d) :

LFGN : $T_N/N\to E[E_1]=1/\lambda$ p.s. donc $T_N\sim N/\lambda\to+\infty$ linéairement. TCL : $\sqrt{N}(T_N/N-1/\lambda)\Rightarrow\mathcal{N}(0,1/\lambda^2)$, soit $(T_N-N/\lambda)/\sqrt{N/\lambda^2}\Rightarrow\mathcal{N}(0,1)$.

In [6] :

```
        ######################################################
        # Réponse [code] (d) :
        # Loi forte des grands nombres.
        ######################################################

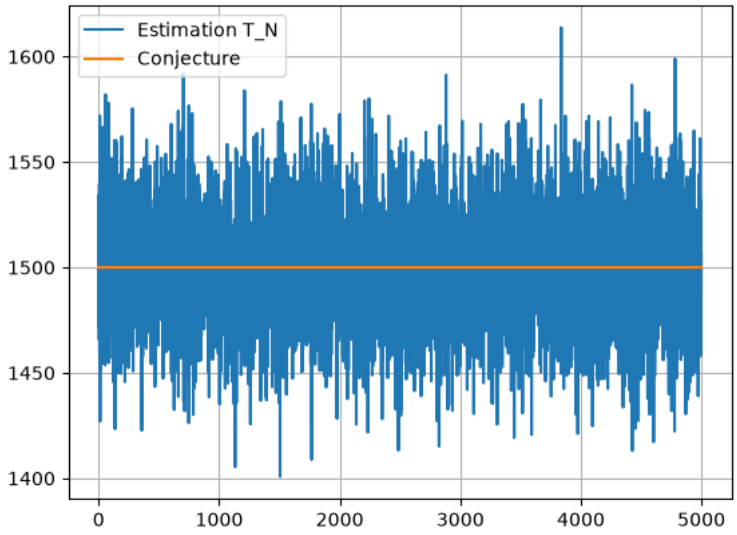
        N = 3000 # Nombre N de variables aléatoires E_i pour calculer T_N.
        M = 5000 # Taille de l'échantillon pour l'estimation Monte-Carlo.

        param_lambda = 2
        Expo = np.random.exponential(1/param_lambda, (M, N))

        esp = 1/param_lambda
        var = (1/param_lambda)**2

        # Echantillon de M valeurs de T_N = somme des N exponentielles
        T_N = np.sum(Expo, axis=1)
        # Moyenne renormalisee T_N/N -> 1/lambda
        moyT_N = T_N / N
        # Erreur normalisee -> N(0,1) par le TCL
        erreurNormalisee_N = (T_N - N * esp) / np.sqrt(N * var)

        ######################################################
        # Affichage des valeurs aléatoires et de votre conjecture
        Conjecture = esp*N
        plt.plot(T_N, label="Estimation T_N")
        plt.plot([1,M],[Conjecture,Conjecture],label="Conjecture")
        plt.legend(loc="best")
        plt.grid(True); plt.show()
        print(f"T_N/N moyen = {moyT_N.mean():.4f}, attendu 1/lambda = {esp:.4f}")
```




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

```
T_N/N moyen = 0.5000, attendu 1/lambda = 0.5000
```

In [7] :

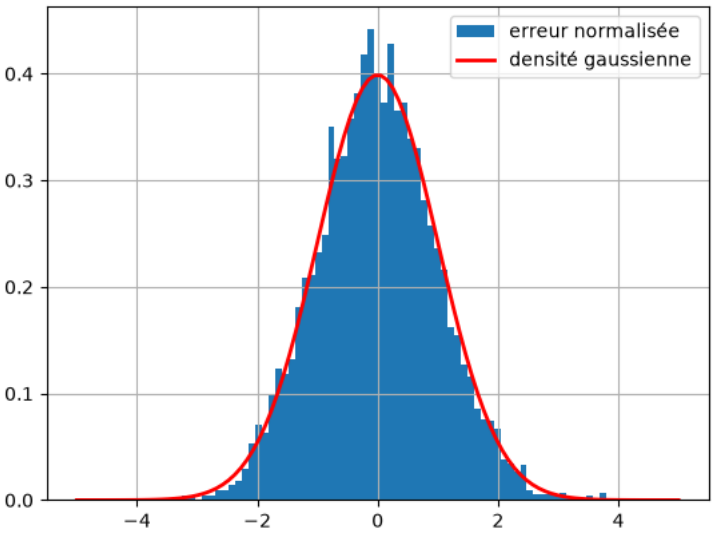
```
        ######################################################
        # Réponse [code] (d) :
        # Théorème Central Limite
        ######################################################

        N = 3000
        M = 5000
        param_lambda = 2
        Expo = np.random.exponential(1/param_lambda, (M, N))
        esp = 1/param_lambda
        var = (1/param_lambda)**2

        T_N = np.sum(Expo, axis=1)
        moyT_N = T_N / N
        erreurNormalisee_N = (T_N - N * esp) / np.sqrt(N * var)

        ######################################################
        # Affichage
        plt.hist(erreurNormalisee_N, density="True", bins=int(np.sqrt(M)), label="erreur
        normalisée")
        x = np.linspace(-5,5,100)
        densiteGaussienne = np.exp(-x*x/2.) / np.sqrt(2. * np.pi)

        plt.plot(x,densiteGaussienne, color="red", label="densité gaussienne", linewidth=2.0)
        plt.legend(loc="best")
        plt.grid(True); plt.show()
```




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

# Exercice 2. Simulation d'une file d'attente.

## 2.1 Un seul serveur de traitement. File d'attente M/M/1

On fixe le taux d'arrivée $\lambda=2$ et le taux de traitement $\mu=3$.

On rappelle la dynamique d'une file d'attente M/M/1 :

• Il n'y a qu'un seul serveur uniquement capable de traiter une personne à la fois.

• Quand un nouveau client arrive :

-- Si le serveur est libre, alors le serveur traite le client immédiatement. Dès que le client a été traité, il quitte la zone et libère le serveur. -- Si au contraire, le serveur est occupé par un client, le client s'ajoute à la file d'attente en attendant que le serveur se libère.

• On précise que la file d'attente est une FIFO (first in, first out).

## $\blacktriangleright$ Question [code] (a) : Compléter le code suivant qui enregistre les temps d'arrivées et les temps de traitement de 10 clients.

In [8] :

```
######################################################
# Réponse [code] (a) :
######################################################

######################################################
# Paramètres de la simulation.
param_lambda = 2.0 # Taux d'arrivée.
param_mu = 3.0 # Taux de traitement.
nombre_total_clients = 10 # Nombre total de clients à simuler.

######################################################
# Génération des temps d'arrivées et de traitement.
temps_inter_arrivees = simule_temps_inter_arrivees(param_lambda, nombre_total_clients)
temps_traitement = simule_temps_traitement(param_mu, nombre_total_clients)
temps_arrivees = np.cumsum(temps_inter_arrivees)

######################################################
# Initialise les temps de sortie de file et les temps d'attente.
temps_sortie = np.zeros(nombre_total_clients)
temps_attente = np.zeros(nombre_total_clients)
longueur_file = np.zeros(nombre_total_clients)

######################################################
# Simule la file d'attente M/M/1 (Lindley).
for i in range(nombre_total_clients):
if i == 0:
temps_sortie[i] = temps_arrivees[i] + temps_traitement[i]
temps_attente[i] = 0.0
else:
temps_entree_serveur = max(temps_arrivees[i], temps_sortie[i-1])
temps_sortie[i] = temps_entree_serveur + temps_traitement[i]
temps_attente[i] = temps_entree_serveur - temps_arrivees[i]

# Calcul du nombre de clients dans la file d'attente à cet instant
longueur_file[i] = max(0, np.searchsorted(temps_arrivees, temps_sortie[i])-i-1)

######################################################
# Affiche les résultats de la simulation.
for i in range(nombre_total_clients):
print(f"\nLe client {i+1} est arrivé à {temps_arrivees[i]:.3f}s, "
f"est traité par le serveur en {temps_traitement[i]:.3f}s, "
f"quitte le serveur à {temps_sortie[i]:.3f}, "
f"en ayant attendu {temps_attente[i]:.3f}s dans la file d'attente "
f"qui possède {longueur_file[i]:.3f} clients en attente quand il sort.")

print(f"\n Le temps moyen d'attente de ces {nombre_total_clients} clients est de
{np.mean(temps_attente):.3f}s.")
print(f"\n Le nombre moyen de clients en attente est de {np.mean(longueur_file):.3f}.")
```

```
Le client 1 est arrivé à 0.184s, est traité par le serveur en 0.433s, quitte le serveur à 0.617, en
ayant attendu 0.000s dans la file d'attente qui possède 1.000 clients en attente quand il sort.

Le client 2 est arrivé à 0.527s, est traité par le serveur en 0.112s, quitte le serveur à 0.730, en
ayant attendu 0.090s dans la file d'attente qui possède 1.000 clients en attente quand il sort.

Le client 3 est arrivé à 0.661s, est traité par le serveur en 0.234s, quitte le serveur à 0.964, en
ayant attendu 0.068s dans la file d'attente qui possède 0.000 clients en attente quand il sort.

Le client 4 est arrivé à 1.697s, est traité par le serveur en 0.065s, quitte le serveur à 1.762, en
ayant attendu 0.000s dans la file d'attente qui possède 0.000 clients en attente quand il sort.

Le client 5 est arrivé à 2.124s, est traité par le serveur en 0.318s, quitte le serveur à 2.442, en
ayant attendu 0.000s dans la file d'attente qui possède 1.000 clients en attente quand il sort.

Le client 6 est arrivé à 2.251s, est traité par le serveur en 0.374s, quitte le serveur à 2.815, en
ayant attendu 0.191s dans la file d'attente qui possède 1.000 clients en attente quand il sort.

Le client 7 est arrivé à 2.546s, est traité par le serveur en 0.299s, quitte le serveur à 3.114, en
ayant attendu 0.269s dans la file d'attente qui possède 1.000 clients en attente quand il sort.

Le client 8 est arrivé à 3.038s, est traité par le serveur en 0.487s, quitte le serveur à 3.602, en
ayant attendu 0.076s dans la file d'attente qui possède 0.000 clients en attente quand il sort.

Le client 9 est arrivé à 3.621s, est traité par le serveur en 0.524s, quitte le serveur à 4.145, en
ayant attendu 0.000s dans la file d'attente qui possède 1.000 clients en attente quand il sort.

Le client 10 est arrivé à 3.929s, est traité par le serveur en 0.416s, quitte le serveur à 4.561,
en ayant attendu 0.216s dans la file d'attente qui possède 0.000 clients en attente quand il sort.

Le temps moyen d'attente de ces 10 clients est de 0.091s.

Le nombre moyen de clients en attente est de 0.600.
```

### $\blacktriangleright$ Question [code] (b) : Après avoir encapsulé votre algorithme dans une fonction, estimer le temps moyen d'attente, et le nombre moyen de clients en attente, lorsque le nombre de clients simulés tend vers l'infini à l'aide de votre algorithme précédent.

Remarque : Avoir un nombre de clients très grand permet de se placer dans le régime stationnaire. On pourra alors estimer le temps moyen d'attente pour un client arrivant dans une file d'attente stationnaire. A priori, les temps d'attente des premiers clients ne sont effectivement pas significatifs.

In [9] :

```
        ####################################################
        # Réponse [code] (b) :
        ####################################################

        ####################################################
        # Fonction de simulation de la file d'attente M/M/1
        def simuler_mm1(taux_arrivee, taux_traitement, nombre_total_clients):
        temps_inter_arrivees = simule_temps_inter_arrivees(taux_arrivee, nombre_total_clients)
        temps_traitement = simule_temps_traitement(taux_traitement, nombre_total_clients)
        temps_arrivees = np.cumsum(temps_inter_arrivees)

        temps_sortie = np.zeros(nombre_total_clients)
        temps_attente = np.zeros(nombre_total_clients)
        longueur_file = np.zeros(nombre_total_clients)

        for i in range(nombre_total_clients):
        if i == 0:
        temps_sortie[i] = temps_arrivees[i] + temps_traitement[i]
        temps_attente[i] = 0.0
        else:
        temps_entree_serveur = max(temps_arrivees[i], temps_sortie[i-1])
        temps_sortie[i] = temps_entree_serveur + temps_traitement[i]
        temps_attente[i] = temps_entree_serveur - temps_arrivees[i]
        longueur_file[i] = max(0, np.searchsorted(temps_arrivees, temps_sortie[i])-i-1)

        return temps_attente, longueur_file
```

In [10] :

```
        nombre_total_clients = 10**6
        temps_attente, longueur_file = simuler_mm1(param_lambda,param_mu,nombre_total_clients)
        print(f"\n Le temps moyen d'attente de ces {nombre_total_clients} clients est de
        {np.mean(temps_attente):.3f}s.")
        print(f"\n Le nombre moyen de clients en attente est de {np.mean(longueur_file):.3f}.")
```

```
Le temps moyen d'attente de ces 1000000 clients est de 0.667s.

Le nombre moyen de clients en attente est de 1.998.
```

## $\blacktrianglerightright$ Question [code] (c) : Estimer le temps moyen d'attente par une procédure Monte-Carlo en réalisant $M$ tirages de votre algorithme précédent (pas forcément avec un grand nombre de clients). Vous pouvez tracer les différentes valeurs obtenues, ainsi qu'un histogramme pour visualiser la loi du temps d'attente.

In [11] :

```
        ####################################################
        # Réponse [code] (c) :
        ####################################################

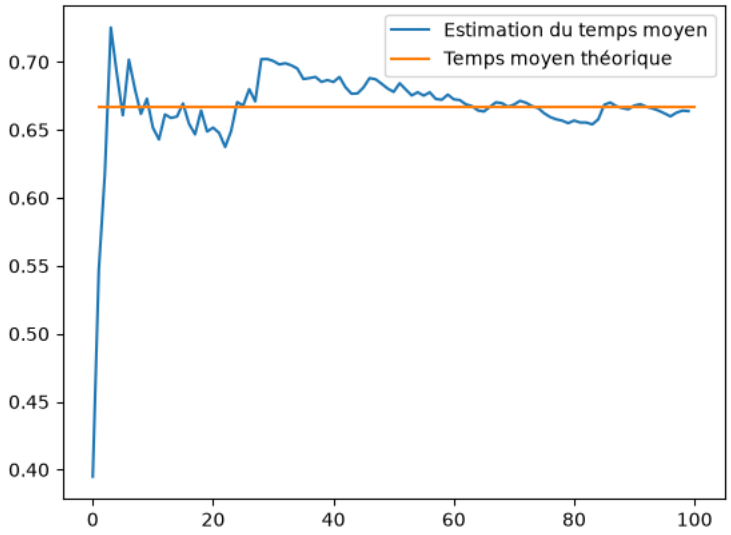
        nombre_total_clients = 10**3
        nombre_tirages_Monte_Carlo = 10**2 # Nombre de répétitions de votre algorithme.
        temps_moyen = np.zeros(nombre_tirages_Monte_Carlo)

        ####################################################
        # Procédure Monte-Carlo pour l'estimation
        for m in range(nombre_tirages_Monte_Carlo):
        ta, _ = simuler_mm1(param_lambda, param_mu, nombre_total_clients)
        # on ignore le transitoire (burn-in) : moyenne sur la 2e moitie
        temps_moyen[m] = np.mean(ta[nombre_total_clients//2:])
```

In [12] :

```
        def cummean(x):
        return np.cumsum(x) / np.arange(1,len(x)+1)
        ######################################################
        # Affichage des tirages Monte-Carlo
        temps_moyen_theorique = param_lambda/param_mu
        plt.plot(cummean(temps_moyen), label="Estimation du temps moyen")
        plt.plot([1,nombre_tirages_Monte_Carlo],[temps_moyen_theorique,temps_moyen_theorique],labe
        l="Temps moyen théorique")
        plt.legend(loc="best")
```

```
<matplotlib.legend.Legend at 0x27f26e01ed0>
```

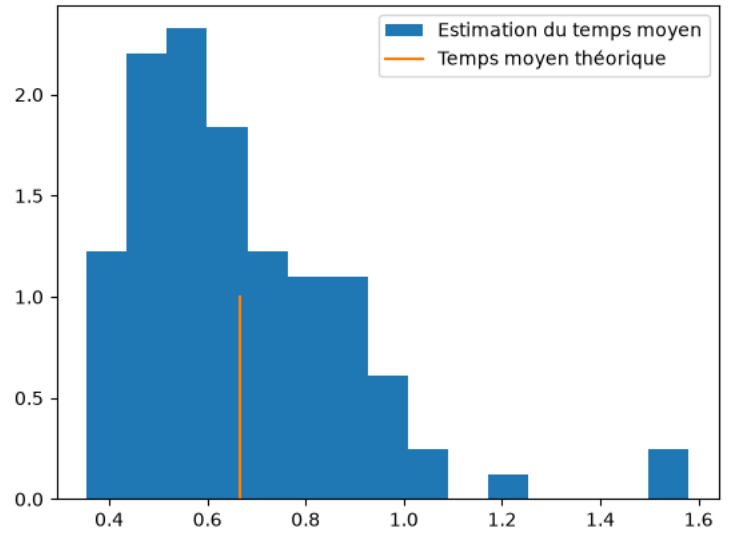



```
<Figure size 640x480 with 1 Axes>
```

In [13] :

```
        ######################################################
        # Visualisation de la loi
        plt.hist(temps_moyen, density="True", bins=int(15), label="Estimation du temps moyen")
        plt.plot([temps_moyen_theorique,temps_moyen_theorique],[0,1],label="Temps moyen
        théorique")
        plt.legend(loc="best")
```

```
<matplotlib.legend.Legend at 0x27f26e51ed0>
```




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

### $\blacktriangleright$ Question [code] (d) : Estimer le nombre moyen de clients dans la file d'attente par une procédure Monte-Carlo en réalisant $M$ tirages de votre algorithme précédent (pas forcément avec un grand nombre de clients). Vous pouvez tracer les différentes valeurs obtenues, ainsi qu'un histogramme pour visualiser la loi du nombre de clients en attente.

In [14] :

```
        ##################################################
        # Réponse [code] (d) :
        ##################################################

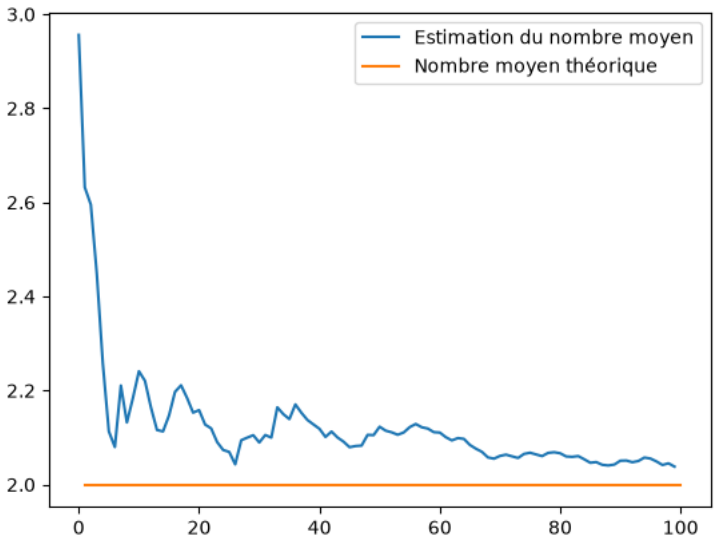
        nombre_total_clients = 10**3
        nombre_tirages_Monte_Carlo = 10**2 # Nombre de répétitions de votre algorithme.
        nombre_moyen = np.zeros(nombre_tirages_Monte_Carlo)

        ##################################################
        # Procédure Monte-Carlo pour l'estimation
        for m in range(nombre_tirages_Monte_Carlo):
        _, lq = simuler_mm1(param_lambda, param_mu, nombre_total_clients)
        nombre_moyen[m] = np.mean(lq[nombre_total_clients//2:])
```

In [15] :

```
        ##################################################
        # Affichage des tirages Monte-Carlo
        nombre_moyen_theorique = param_lambda/(param_mu-param_lambda)
        plt.plot(cummean(nombre_moyen), label="Estimation du nombre moyen")
        plt.plot([1,nombre_tirages_Monte_Carlo],[nombre_moyen_theorique,nombre_moyen_theorique],la
        bel="Nombre moyen théorique")
        plt.legend(loc="best")
```

```
<matplotlib.legend.Legend at 0x27f26dc9ed0>
```

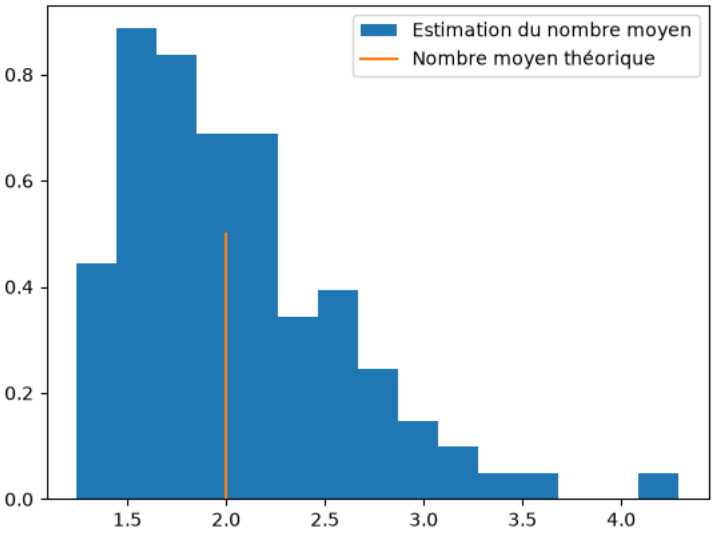



```
<Figure size 640x480 with 1 Axes>
```

In [16] :

```
        ##################################################
        # Visualisation de la loi
        plt.hist(nombre_moyen, density="True", bins=int(15), label="Estimation du nombre moyen")
        plt.plot([nombre_moyen_theorique,nombre_moyen_theorique],[0,0.5],label="Nombre moyen
        théorique")
        plt.legend(loc="best")
```

```
<matplotlib.legend.Legend at 0x27f26dd4b90>
```




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