Modélisation des files d'attente dans le secteur sanitaire.

Réalisé par:
Bourazza Hamza
N SCEI:20573

Plan:

- 1-Notations.
- 2-Théorie des files d’attente.
- 3-modélisation des files d’attente.
- 4-simulation.

Notations:

- <u>*Notation de kendall:*</u>

La nomenclature générale d'une

file d'attente est de la forme A/B/n/m/d

A type d'arrivées

B:type de service

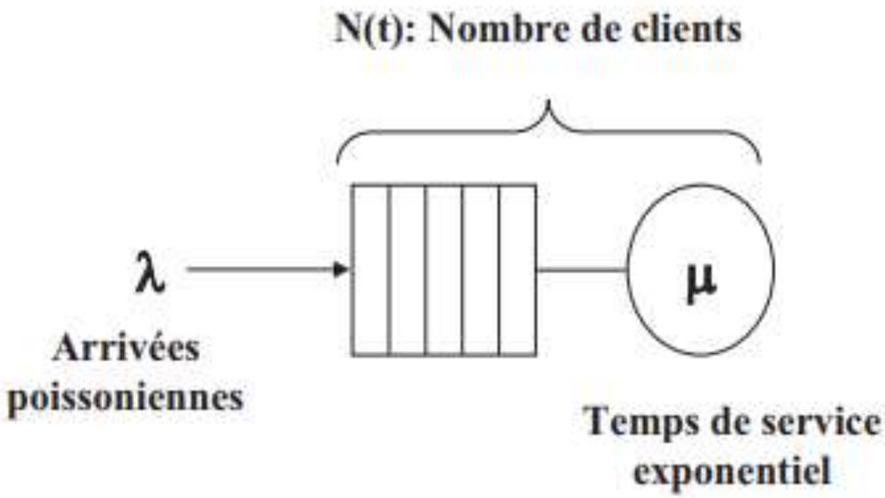
n:nombre de serveurs

m:la capacité du système

Dis:la discipline lifo ou fifo

Les types possibles pour A et B sont M (markovien), D (déterministe), G (général)...

File M/M1:


N(t): Nombre de clients
λ
μ
Arrivées
poissoniennes
Temps de service
exponentiel

A/B/c/K/m/Z:

A	loi de l'intervalle de temps entre 2 arrivées $A = M$: loi exponentielle $A = D$: inter arrivées constantes $A = G$: loi non précisée (cas général)
B	la loi du temps de service
C	nombre de serveurs
K	nombre maximum de clients dans la file
m	nombre de clients potentiels
Z	discipline de la file FIFO, LIFO, FIRO (R=Random), FIPO ? ? ?

Rq : lorsque K ou m sont infinis, on ne les note pas, idem pour $Z = FIFO$

c	nombre de serveurs
λ	nombre moyen de clients entrant par unité de temps
μ	nombre moyen de clients servis par unité de temps
$\frac{1}{\mu}$	temps moyen de service
$\varphi = \frac{\lambda}{\mu}$	intensité du trafic **ou** nombre moyen de clients entrant par u.t. multiplié par le temps moyen de service, **ou encore** temps d'occupation cumulé sur les c serveurs (*i.e.*, le nombre moyen de clients en service)
$\rho = \frac{\varphi}{c} = \frac{\lambda}{c\mu}$	probabilité qu'un serveur soit occupé **ou** taux d'utilisation du système **ou** nombre moyen d'arrivées pendant le temps moyen de service
τ	intervalle de temps moyen entre deux arrivées

N_t	variable aléatoire donnant le nombre moyen de clients présents dans le système (indépendant du temps) ($N_t = N$)
$p_k = P(N = k)$	$(p_k)_{k>0}$ la loi de N
$N_{q,t}$	nombre de clients dans la file ($N_{q,t} = N_q$)
$N_{s,t}$	nombre de clients en cours de traitement ($N_{s,t} = N_s$)
$L = \mathbb{E}(N)$	nombre moyen de clients dans le système
$L_q = \mathbb{E}(N_q)$	nombre moyen de clients dans la file
$L_s = \mathbb{E}(N_s)$	nombre moyen de clients en cours de traitement

q	variable aléatoire donnant le temps passé à attendre
s	variable aléatoire donnant le temps de service
w	variable aléatoire donnant le temps passé dans le système
$W = \mathbb{E}(w)$	temps moyen passé dans le système
$W_q = \mathbb{E}(q)$	temps moyen passé à attendre
$W_s = \mathbb{E}(s)$	temps moyen de service

Relations importantes.

- $w = q + s$
- $W = W_q + W_s$
- $N = N_q + N_s$
- $L = L_q + L_s$

- Q:queue.
- S:service.

Lois de Little (relation entre nombre de clients et le temps passé)

- $L = \lambda W$
- $Lq = \lambda Wq$
- $Ls = \lambda Ws$

Théorie Pour M/M/1

- pn *:probabilité qu'il y ait n clients dans le système:*

$$p_n = P(N = n) = \varphi^n(1 - \varphi)$$

Il vient alors:

$$L = \mathbb{E}(N) = \frac{\varphi}{1 - \varphi}$$

- *L:* nombre moyen de clients dans le système

Le temps passé dans le système est:

$$W = \frac{L}{\lambda} = \frac{1}{\mu(1-\varphi)} = \frac{\mathbb{E}(s)}{1-\varphi}$$

temps moyen de service :

$$W_s = \mathbb{E}(s) = \frac{1}{\mu}$$

temps moyen passé dans la file:

$$W_q = W - \mathbb{E}(s) = \frac{\varphi}{\mu(1-\varphi)}$$

nombre moyen de clients dans la file: loi de little

$$L_q = \lambda W_q = \frac{\varphi^2}{(1-\varphi)}$$

la probabilité que le serveur soit vide:

$$p_0 = P(N = 0) = 1 - \varphi$$

probabilité d'avoir moins de n clients dans le système:

$$p_{k<Nn} = P(k < n) = 1 - \varphi^n$$

Pour une file M/M/C.

- ***Pour que le système soit stable il faut que la capacité des serveurs soit supérieure au taux d'arrivée :***

$$\lambda < C\mu \text{ on pose } \rho = \frac{\lambda}{C\mu} = \frac{\varphi}{C} < 1$$

En régime permanent:

La probabilité qu'il y ait n clients dans le système est:

$$\begin{cases} p_n = \frac{\varphi^n}{n!} p_0 & \text{pour } n = 1, \ldots, C-1 \\ p_n = \frac{\varphi^n}{C!C^{n-C}} p_0 & \text{pour } n = C, C+1, \ldots \end{cases}$$

- La Probabilité de que le système soit vide :

$$p_0 = \frac{1}{\sum_{n=0}^{C-1} \frac{\varphi^n}{n!} + \frac{\varphi^C}{C!(1-\rho)}}$$

La probabilité de que tous les serveurs soient occupés :

$$p_c = \frac{\varphi^C}{C!} p_0$$

Nombre moyen de clients dans le système

$$L = \varphi + p_0 \frac{\varphi^C}{C!} \frac{\rho}{(1-\rho)^2}$$

Temps moyen passé dans le système : formule de Little L = λ W :

$$W = \frac{1}{\mu} + p_c \frac{1}{C\mu} \frac{1}{(1-\rho)^2}$$

- Temps moyen passé dans le système à attendre:

$$W_q = p_c \frac{1}{C\mu} \frac{1}{(1-\rho)^2}$$

Nombre moyen de clients en attente : Lq = λ Wq

$$L_q = p_c \frac{\rho}{(1-\rho)^2}$$

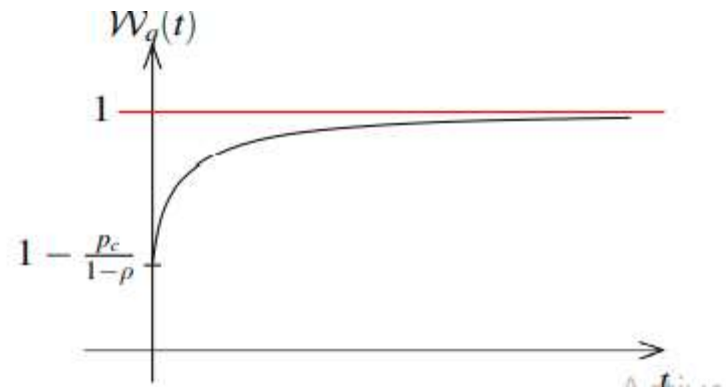
la probabilité qu'un client n'attende pas:

$$W_q(0) = P(q=0) = P(N \leqslant C-1) = 1 - \frac{p_c}{1-\rho}$$

- q est la variable aléatoire donnant le temps passé à attendre... :

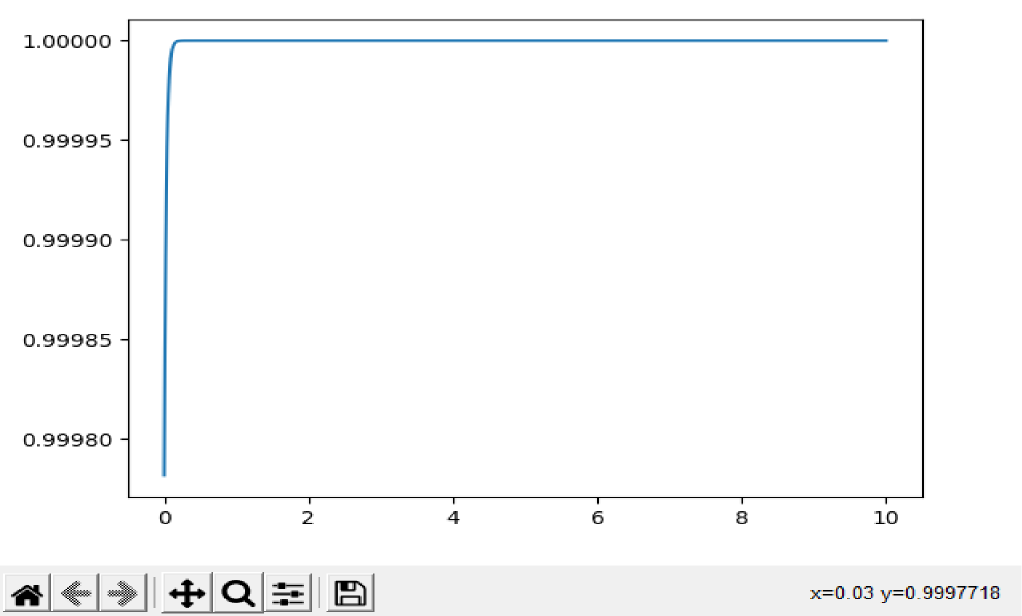
Wq(t) = P(q < t) est :

$$\begin{cases} \mathcal{W}_q(t) = P(q \leqslant t) \\ \mathcal{W}_q(t) = \mathbb{1}_{[0,t[}(t)\left(1 - \frac{p_c}{1-\rho} e^{-C\mu(1-\rho)t}\right) \end{cases}$$

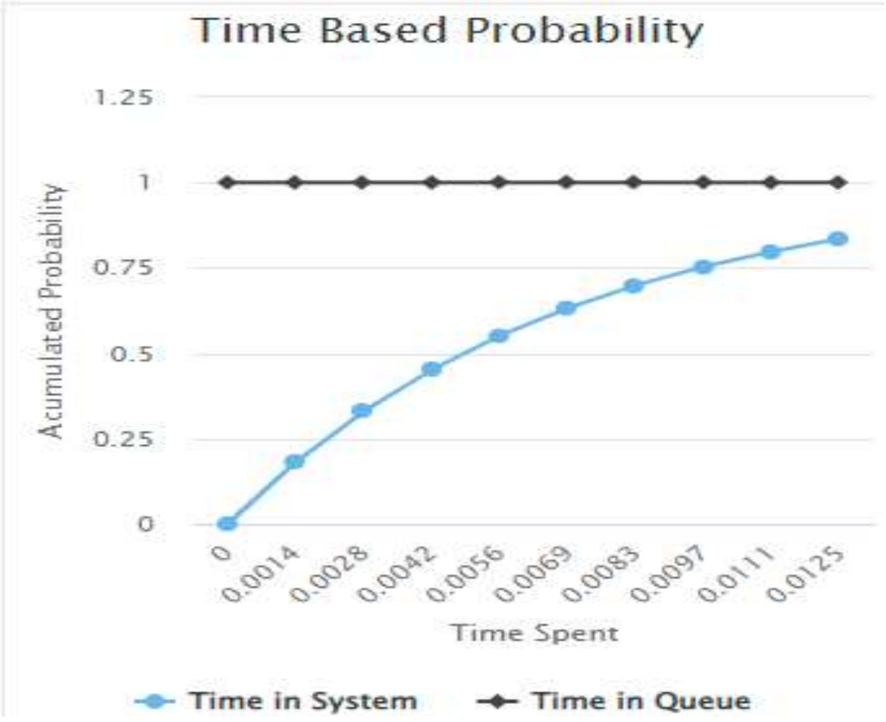


Résultat matplotlib **Exemple** c=λ=7 μ =6

```
>>> T=np.linspace(0,10,1000)

>>> Y=repartition(7,6,7,T)

>>> plt.plot(T,Y)
[<matplotlib.lines.Line2D object at 0x00000266AE0A4B50>]

>>> plt.show()
```



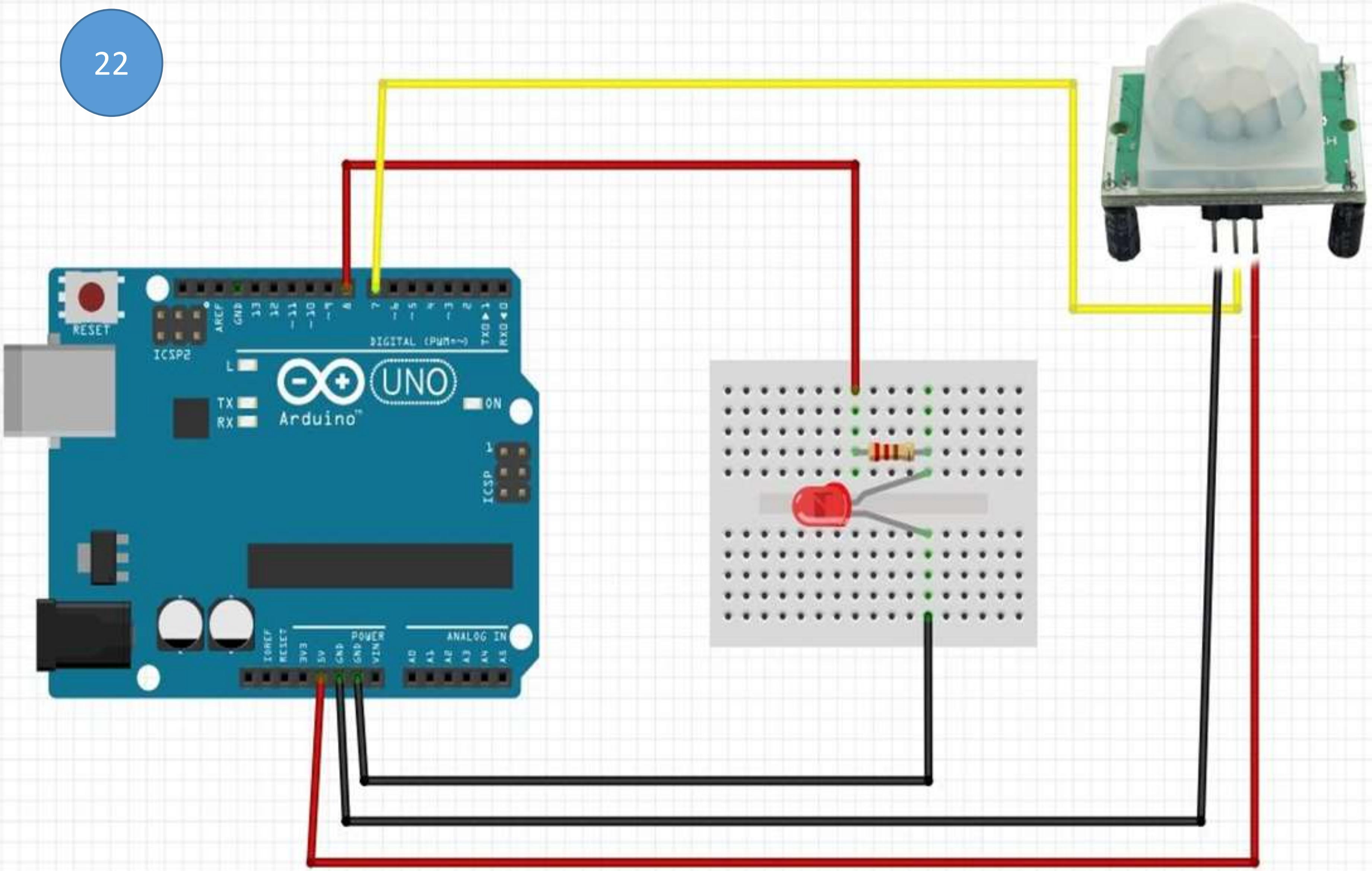
Résultat théorique

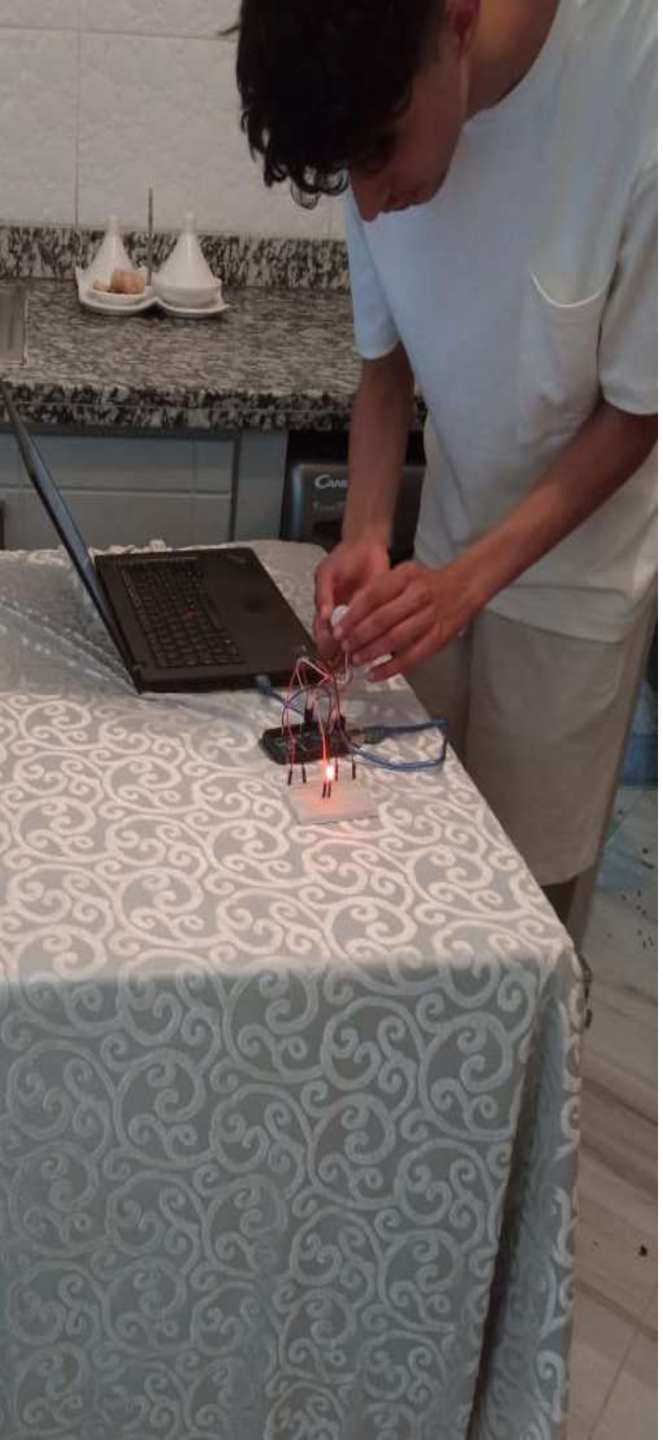


Modelisation d’une file d’attente type M/M/C

```
def modelisation_file_dattenteMM(l,u,c,n):#l:nombre moyen de clients par unite de temps u :nombre servis par unite de temps c nombre de serveurs m model
    return("Probabilité de que le système soit vide est ",proba_vide(l,u,c),"la proba que tous les serveurs soient vides est",proba_serveurs_vides(l,u,c)  /n ,"le systeme est",est_stable_systeme(l,u,c),"Nombre moyen de clients dans le système",moyen_clients_système(l,u,c),   "Temps moyen passé dans le système",temps_moy_s(l,u,c),"Temps moyen passé dans la file",attente_queue(l,u,c),"Temps moyen passé dans le systeme",att_syst(l,u,c),"Nombre moyen de clients en attente",moy_clients_att(l,u,c),"probabilité qu'il y ait",n," clients dans le système",proba_clients(l,u,c,n),"probabilité qu'un client n'attende pas",proba_non_attente(l,u,c))
```

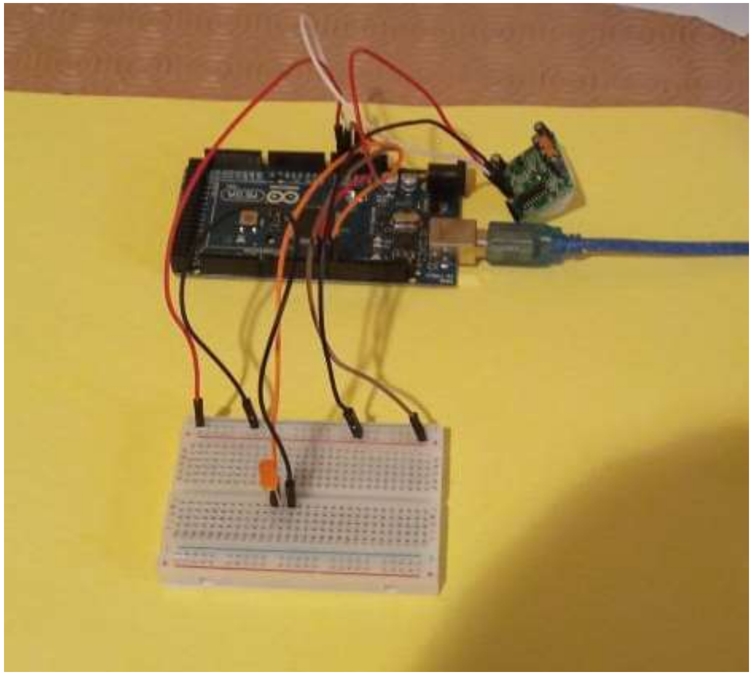
- Ainsi il est clair qu'on doit collecter les variables λ et μ , pour ce faire on utilise le montage a base arduino suivant .





Problèmes rencontrés:

faible précision du capteur.



Dans toute cette partie on travaille avec:

- Tn: l'instant d'arrivée du nième patient.

$(\tau_n = T_{n+1} - T_n)_{n \geq 0}$ les inter arrivées qui sont exponentielles (λ).

- $N(t) = \sum_{n \geq 1} T_n \qquad \forall T_n \in [0, t]$

représente le nombre de clients entrées .

Un :Le temps d'arrivée au guichet du nième client au serveur.

$$U_n - U_{n-1} \rightsquigarrow \exp(\lambda).$$

Rn le temps de service du n eme patient.

On suppose qu'il existe deux constantes a, D > 0 telles que :

$$R_n = aA_n + D.$$

Avec An:le nombre de services fournis au n eme patient.

- On travaille dans le cas d’une file de type M/M/K.
- Nous visons maintenant d’optimiser le nombre de serveurs afin que la file demeure raisonnable ainsi nous nous intéressons a pour un $p = P\{\max_{[0,t]} F > S\}$
- Instant t fixé et des valeur k de serveurs, nous choisirons alors le plus petit entier k rendant cette quantité inférieure à 0,05.
- le calcul de p revient à un calcul numérique d'une espérance à l'aide de la loi des grands nombres.

Proposition:

On définit:

$$\tau = \max\{n \geq 1,\ T_n \leq t\}$$

Et pour τ≥ 1:

$$\forall\, 1 \leq i \leq \tau\ ,\ m_i = \sum_{k=1}^{i} \{1_{T_i < U_k}\}.$$

Alors

$$\underset{[0,t]}{max}\ F = \underset{i=1,2..\tau}{max}\ (m_i)$$

Avec

$$\max(\varnothing) = 0.$$

- On suppose dans ce qui suit :
- L'unité de temps est la minute.

Le temps d’ouverture 10 heures par jour.

La constante a est trente secondes par service et la constante d à une minute et Trente secondes.

Le temps moyen d'inter-arrivée est d'une minute et vingt secondes.

Le nombre moyen de services offerts à un patient entrant est de 3.

Le seuil critique est 20 personnes.

- En utilisant ces données numériques, nous avons simulé, pour des valeurs de k allant de 1 à 4, une réalisation de la loi binomiale de paramètres 100 et p.
- Nous avons consigné les résultats obtenus ci-dessous en prenant soin de faire apparaitre la moyenne empirique associée:
- Pour k=1 : Réalisation de la somme : 100. Moyenne empirique : 1 .
- Pour k=2 : Réalisation de la somme : 100. Moyenne empirique : 1.
- Pour k=3 : Réalisation de la somme : 2. Moyenne empirique : 0.02 .
- Pour k=4 : Réalisation de la somme : 0. Moyenne empirique : 0

- D'apres l'inégalité de Bienyamé Tchebechyv

$$\mathbb{P}(|X-\mathbb{E}[X]|\geq\alpha)\leq\frac{\sigma^2}{\alpha^2}.$$

- on trouve en choisissant :

$$\varepsilon = \sqrt{\frac{-ln(0.05)}{200}};$$

Si k = 1, p ∈]0.87, 1],

Si k = 2, p ∈]0.87, 1]

Si k = 3, p ∈ [0, 0.15],

Si k = 4, p∈ [0, 0.13].

- *k = 1 et k = 2 sont éliminées*

  k = 3 apparait satisfaisante

Simulation de taille de la le d'attente à un instant donné a serveur unique

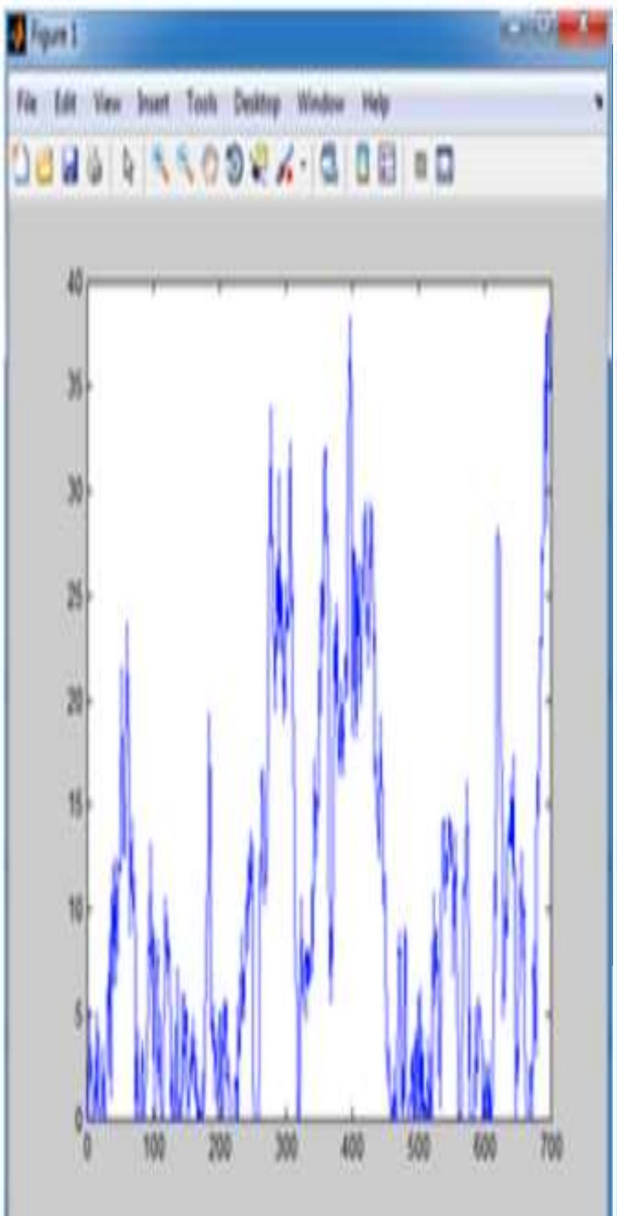
Le codage suivant permet de simuler une réalisation du vecteur (F(1), F(2), ..., F(p)) (la taille de la file)

```
>> lamda=0.75
p=100
a=3
c=1
d=1

T(1) = - log(rand)/lamda
U(1) = T(1)
n=1
while (T(n)<p)
T(n+1) = T(n) - log(rand)/lamda
U(n+1) = max(U(n)+ a*randn(c) + d,T(n+1))
n=n+1
end


for (j=1:p)
N(j) = T(j)
end
for (j=1:p)
S(j) = U(j)
end
for (j=1:p)
 F(j) = N(j)-S(j)
end
plot(-F)
```

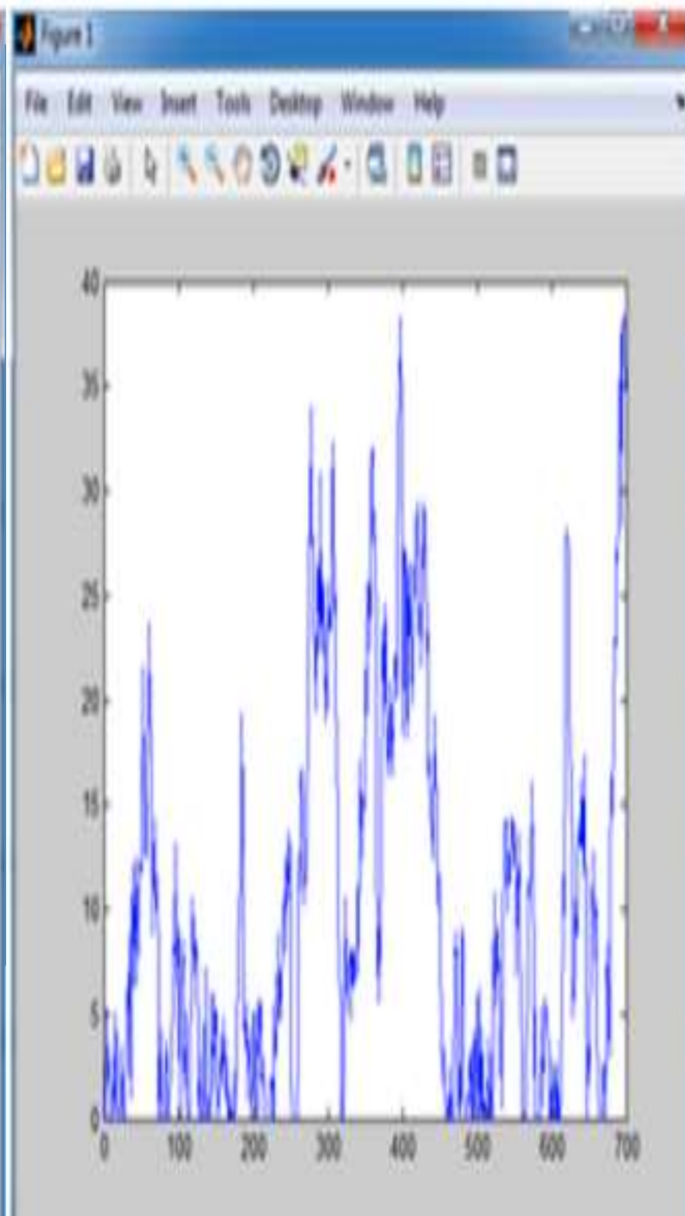
```
>> k=3;
a=2;
c=1;
d=1;
p=100;
lamda=0.7;
T(1) = - log(rand)/lamda;
U(1) = T(1);
n=1;
while(T(n)<p)& (n < k)
T(n+1) = T(n) - log(rand)/lamda;
U(n+1) = T(n+1);
n=n+1;
end

v=U+a*randn(c)+d;
v;
while (T(n)<p)
T(n+1) = T(n) - log(rand)/lamda;
U(n+1) = max(v(1),T(n+1));
v;
n=n+1;
end
for (j=1:p)
N(j) = T(j);
end
for (j=1:p)
S(j) = U(j);
end
for (j=1:p)
end
plot(-F)
```

- Ce qui donne les résultats suivants :

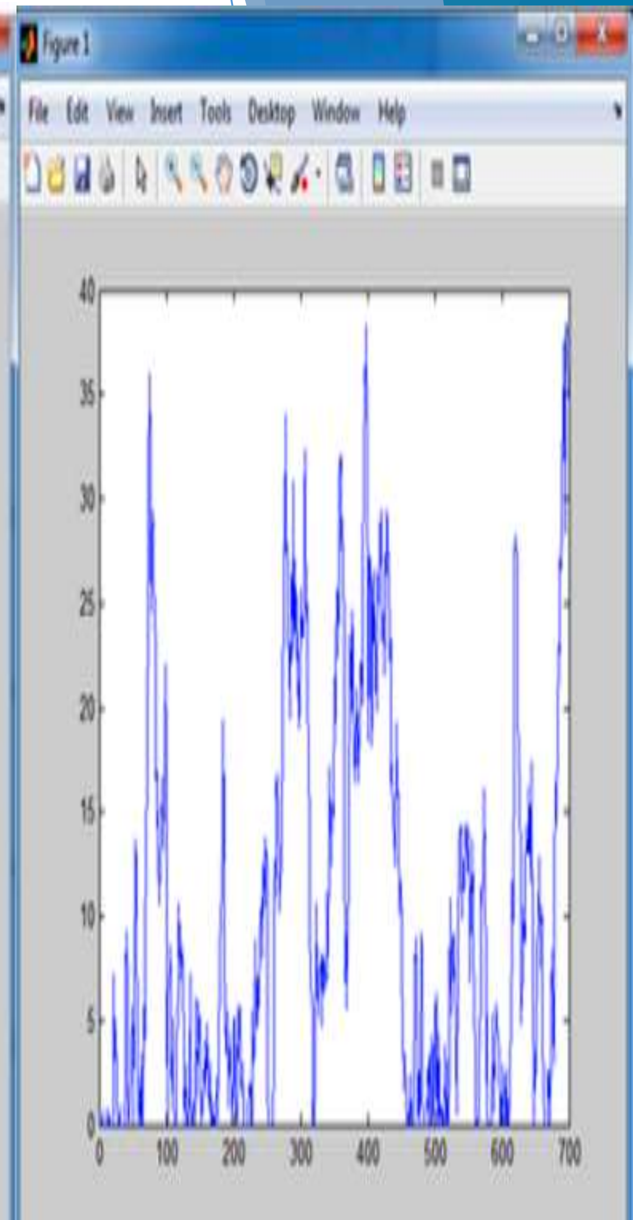
λ = 0.75:



Pour λ=0.25

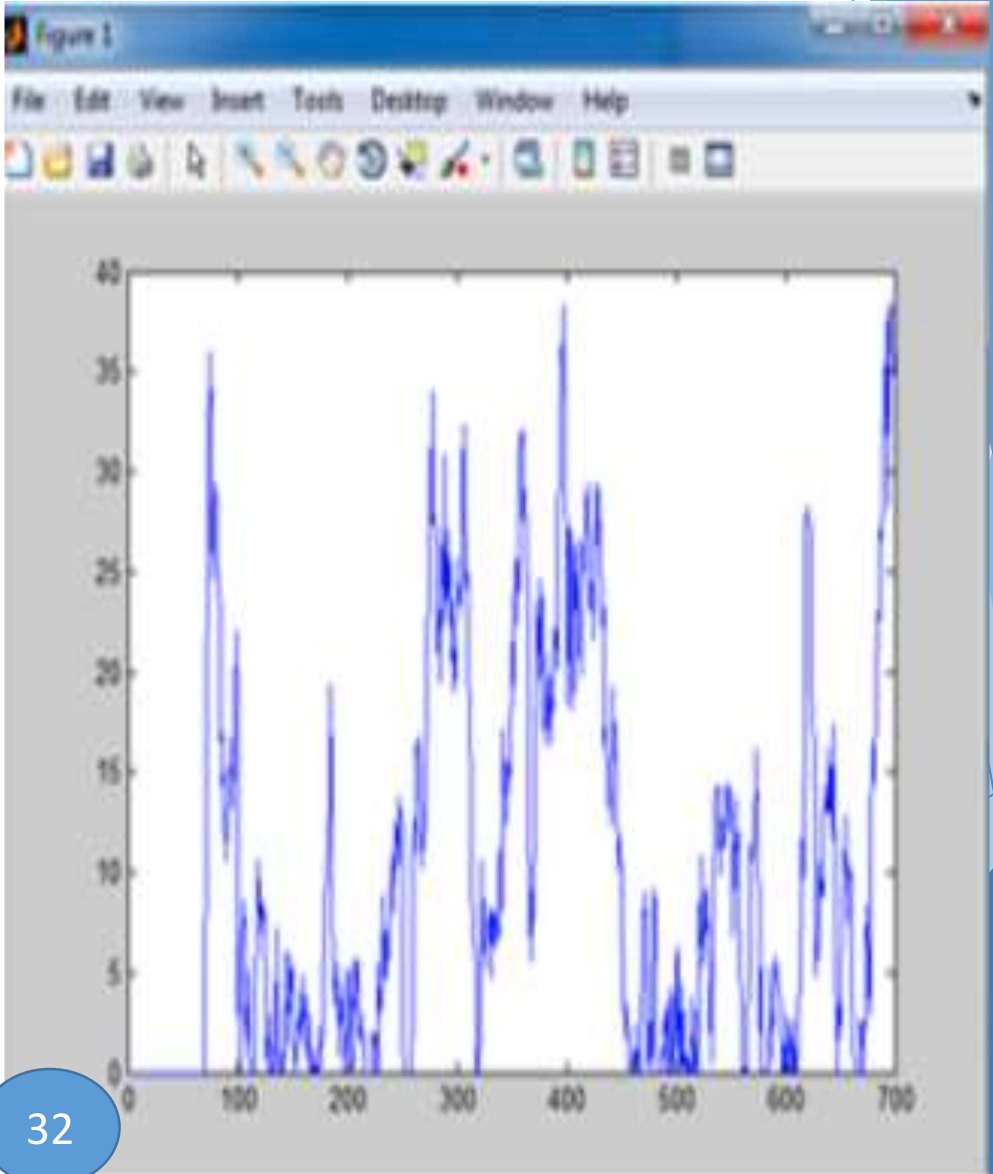


Pour λ=0.15



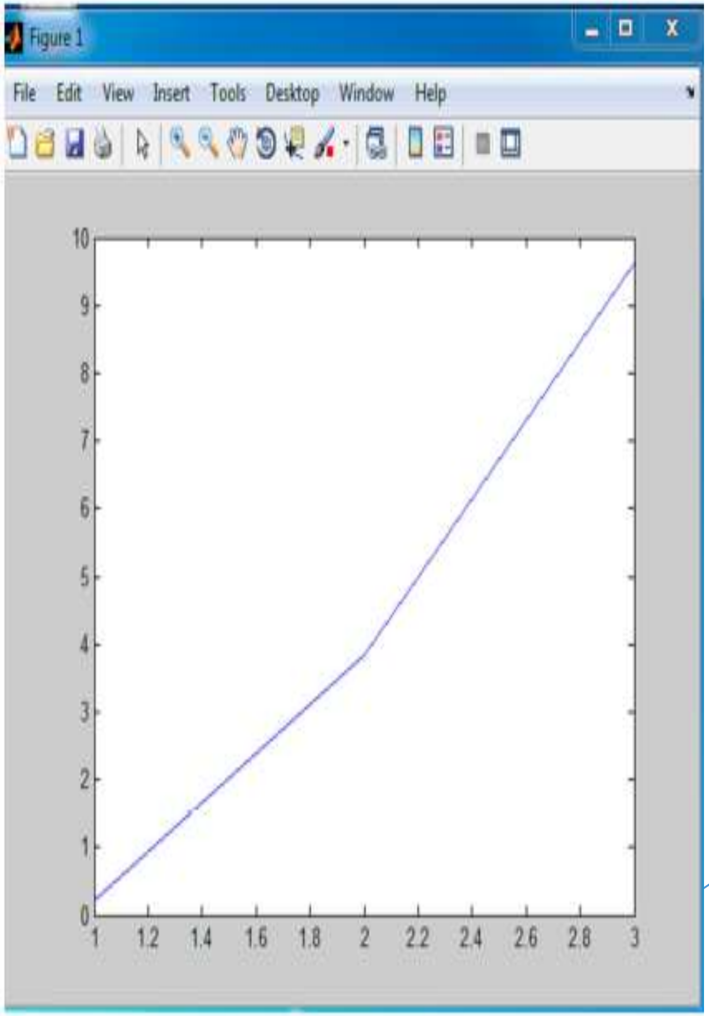
Cas d'un nombre fini de serveurs.

```
>> k=3; a=2; c=1; d=1;
p=100;
lamda=0.7;
T(1) = - log(rand)/lamda;
U(1) = T(1);
n=1;
while(T(n)<p)& (n < k)
T(n+1) = T(n) - log(rand)/lamda;
U(n+1) = T(n+1);
n=n+1;
end
v=U+a*randn(c)+d; v;
while (T(n)<p)
T(n+1) = T(n) - log(rand)/lamda;
U(n+1) = max(v(1),T(n+1)); v;
n=n+1;
end
for (j=1:p)
N(j) = T(j);
end
for (j=1:p)
S(j) = U(j);
end
plot(-F)
```



Algorithme d’optimisation.

```
>> k=3;
a=3;
c=1;
d=1;
p=0.88;
lamda=0.2;
T(1) = - log(rand)/lamda;
U(1) = T(1);
n=1;
while(T(n)<p)& (n < k)
T(n+1) = T(n) - log(rand)/lamda;
U(n+1) = T(n+1);
n=n+1;
end
v=U+a*randn(c)+d; v;
while (T(n)<p)
T(n+1) = T(n) - log(rand)/lamda;
U(n+1) = max(v(1),T(n+1)); v; n=n+1;
end
nombre=1; T; U; t=2;
for (i=1:t)
for (j=1:i)
if U(j) >T(i)
m(j)=U(j);
nombre=nombre+1;
end
end
end
plot(T)
```



Conclusion:

- Ainsi en se basant sur les formules fournies par la théorie des files d’attente , après avoir collecté les deux valeurs λ et μ ,nous avons pu implémenter en python des fonctions de modélisation et de faire une simulation en Matlab qui nous a permis de proposer la meilleure stratégie (k=3 serveurs:))

Merci de votre attention

Annexe 1: Code Arduino.

```
int ledPin = 8;
int mouvPin = 7;
int comptage = 0;

void setup() {
    Serial.begin(9600);

  pinMode(ledPin, OUTPUT);
  pinMode(mouvPin, INPUT);

  digitalWrite(ledPin, LOW);
}

void loop() {
  if(digitalRead(mouvPin) == HIGH)
  {
    digitalWrite(ledPin, HIGH);
     comptage++;

    Serial.println(comptage);
    delay(1000);
  }

  else
  {
    digitalWrite(ledPin, LOW);
  }
 }
```

Annexe2:Implémentation python

```
def est_stable_systeme(l,u,c):
    if l<c*u:
        return("stable")
    return("instable")
```

Figure1 :Fonction vérifiant si la file est stable.

```
def factoriel(i):
    if i==0:
        return(1)
    else:
        s=1
        for i in range(1,i+1):
            s*=i
        return(s)

def proba_vide(l,u,c):  #Probabilité de que le système soit vide
    phi=l/u
    rho=phi/c
    s=0
    for i in range(0,c):
        s+=((phi)**(i))/(factoriel(i))
        o=s+(phi**(c))/(factoriel(c)*(1-rho))
    return(1/o)
```

Figure2 :La probabilité qu’il y ait n clients dans le système.

```python
def moyen_clients_système(l,u,c) :#Nombre moyen de clients dans le système
    phi=l/u
    rho=phi/c
    return(phi+proba_vide(l,u,c)*((phi**(c))/(factoriel(c)))*(rho/(1-rho**2)))
def temps_moy_s(l,u,c):#Temps moyen passé dans le système : formule de Little L = λW

    return(moyen_clients_système(l,u,c)/l)

def attente_queue(l,u,c):#Temps moyen passé dans le système à attendre : W = Wq + Ws = Wq + E(S)
    phi=l/u
    rho=phi/c
    return((proba_serveurs_vides(l,u,c))*(1/(c*u*(1-rho)**2)))

def att_syst(l,u,c) : #W = Wq + Ws  ws=E(S)=1/u
    return((1/u)+attente_queue(l,u,c))

def moy_clients_att(l,u,c):#Nombre moyen de clients en attente : Lq = λWq
    return(l*attente_queue(l,u,c))

def proba_clients(l,u,c,n):#probabilité qu'il y ait n clients dans le système
    phi=l/u
    if n in range(1,c):

        return(((phi**(n)/factoriel(n))*proba_vide(l,u,c)))
    else:
        return((phi**n)/(factoriel(c)*c**(n-c))*proba_vide(l,u,c))
```

Wq(t) = P(q < t)

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


def factoriel(i):
    if i==0:
        return(1)
    else:
        s=1
        for i in range(1,i+1):
            s*=i
        return(s)

def proba_vide(l,u,c):  #Probabilité de que le système soit vide
    phi=l/u
    rho=phi/c
    s=0
    for i in range(0,c):
        s+=((phi)**(i))/(factoriel(i))
        o=s+(phi**(c))/(factoriel(c)*(1-rho))
    return(1/o)


def proba_serveurs_vides(l,u,c):
    phi=l/u
    return(((phi**(c))/factoriel(c))*proba_vide(l,u,c))


def repartition(l,u,c,t):#Wq(t) la fonction de répartition de la variable aléatoire q :
Wq(t) = P(q < t) la probabilité que le temps d'attente soit inférieur à t.
    phi=l/u
    rho=phi/c
    return(1-(proba_serveurs_vides(l,u,c)/(1-rho))*np.exp(-c*u*(1-rho)*t))
    T=np.linspace(0,100,1000)
    Y=repartition(l,u,c,T)
    plt.plot(T,Y)
    plt.s
```

```python
#q est la variable aléatoire donnant le temps passé à attendre... Soit Wq(t) la fonction de répartition de la variable aléatoire q : Wq(t) = P(q < t)
def proba_non_attente(l,u,c):#probabilité qu'un client n'attende pas
    phi=l/u
    rho=phi/c
    return(1-(proba_serveurs_vides(l,u,c)/(1-rho)))
def proba_attente(l,u,c):
    return(1-proba_non_attente(l,u,c))
```