Projet 1 - Modélisation du slicing dans les réseaux 5G

Alexis Mellier

April 25, 2025

1 Import

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

2 Préliminaires

2.1) En reprenant la relation de récurrence de la formule d'Erlang-B :

$$\frac{1}{\mathrm{Erl}_B[\rho, 0]} = 1$$

$$\frac{1}{\mathrm{Erl}_B[\rho, S]} = 1 + \frac{S}{\rho\,\mathrm{Erl}_B[\rho, S-1]}.$$

La file M/M/S/S est définie comme :
- Arrivées de paramètre λ
- Service de paramètre μ
- S serveurs, pas de buffer
- $\rho = \frac{\lambda}{\mu}$
- Modèle à perte, car pas de file d'attente, si les S serveurs sont occupés, le client est perdu.

Je pose N, le nombre de clients à l'état stationnaire (ne varie pas), alors comme vu dans le cours, le nombre moyen de clients est donné par :

$$\mathbb{E}[N] = \sum_{k=1}^{S} k \cdot \pi_s(k)$$

Je sais que $\pi_s(i)$ est la proba stationnaire d'avoir i clients dans le système, donc :

$$\pi_s(i) = \frac{\frac{\rho^i}{i!}}{\sum_{j=0}^{S} \frac{\rho^j}{j!}} = \frac{\rho^i}{i!}\pi_s(0)$$

Donc on trouve cette formule finale :

$$\boxed{\mathbb{E}[N] = \sum_{n=1}^{S} n \cdot \frac{\frac{\rho^n}{n!}}{\sum_{j=0}^{S} \frac{\rho^j}{j!}}}$$

[3]:
```
#Fonction qui calcul #moyen de clients dans un système M/M/S/S à l'état stationnaire
def mean_number_waiting_customers(arrival_rate,service_rate,nb_of_servers):
    rho = arrival_rate / service_rate #(Fomule de lambda\mu)
    value = 1.0
    pi_0 = value
    numerateur = 0.0

    #Fais la reccurance sur tout les thermes et traite le numerateur et deniminateur separement pour ne pas faire de factorielle
    for n in range(1, nb_of_servers + 1):
        value *= rho / n
        pi_0 += value
        numerateur += n * value
    #Renvoie la moyenne E[N]
    return numerateur / pi_0


#print(mean_number_waiting_customers(4,2,10))
#EQ1 = mean_number_waiting_customers(4,2,10)

#lambda2 = (10 - EQ1)*1
#print(lambda2)
```

2.2) Via les résultats vus en cours sur le modèle $M/M/\infty$, on sait que lorsque $S \to +\infty$:

$$\pi_\infty(i) = \frac{\frac{\rho^i}{i!}}{\sum_{j=0}^{\infty} \frac{\rho^j}{j!}} = e^{-\rho}\frac{\rho^i}{i!}$$

Et cela correspond exactement à une loi de Poisson de paramètre ρ.

Par définition d'une loi de Poisson, le nombre moyen de clients dans le système quand $S \to +\infty$ est :

$$\mathbb{E}[N] = \sum_{i=0}^{\infty} i \cdot \pi_\infty(i) = \rho$$

Cela montre que, dans le cas d'une infinité de serveurs, le système ne sature jamais et le nombre moyen de clients est simplement donné par le rapport entre le taux d'arrivée et le taux de service.

Pour ce qui est de $Erl_B[\rho, S] \ll 1$, cela veut dire peu de pertes, donc pas d'attente : c'est comme si on avait des serveurs infinis du point de vue du client.

3 Modélisation

Dans cette partie, nous avons 2 types de clients :

Type 1 (priorité élevée) :
- Arrive selon $PP(\lambda_1)$
- Durée de service $Exp(\mu_1)$
- Non bufferisés
- Capacité de préemption

Type 2 :
- Arrive selon $PP(\lambda_2)$
- Durée de service $Exp(\mu_2)$
- Bufferisés

Je note (# = nombres) :
- Q_1 = #clients de type 1 dans le système, respectivement Q_2.
- S_1 = #serveurs occupés par des clients de type 1, respectivement S_2.
- B = #clients de type 2 dans le buffer.

3.1) Si j'écris mathématiquement les définitions :

- $S_1 + S_2 < S \Leftrightarrow$ Il reste des serveurs libres, donc un client de type 1/2 peut être servi immédiatement.
- $S_1 + S_2 = S \Leftrightarrow$ Tous les serveurs sont occupés. Grâce à la préemption, un client de type 2 est déplacé dans le buffer pour libérer un serveur pour un client de type 1.
- $S_1 = S \Leftrightarrow$ Tous les serveurs sont occupés par des clients de type 1. Un nouveau client de type 1 est alors perdu.
- $Q_1 = S_1$, car les clients de type 1 ne sont jamais bufferisés.
- $Q_2 = S_2 + B$, car les clients de type 2 peuvent être en service et/ou en file d'attente.

Donc :

Si $q_1 + s_2 < S \Rightarrow$ il reste au moins un serveur libre.

Mais si $B > 0 \Rightarrow$ il y a des clients de type 2 dans le buffer, et s'il reste un serveur libre, le client dans le buffer serait automatiquement mis dans un serveur.

Donc la situation $B > 0$ n'est valable qu'avec un système saturé : $q_1 + s_2 = S$.
Donc j'en conclus que :

$$\boxed{q_1 + s_2 < S \implies B = 0}$$

3.2) Le processus Q_1 est un processus de Markov car, comme j'ai montré dans la question précédente, Q_1 dépend uniquement de son état actuel, de son taux d'arrivée λ_1, et de son taux de départ μ_1. Contrairement à Q_2, qui n'est pas un processus de Markov car il dépend de la disponibilité des serveurs, donc de Q_1.

Q_1 est donc un processus de Markov, de caractéristiques M/M/S/S

3.3) En reprenant les définitions du dessus, les transitions du processus (Q_1, Q_2) sont :

- $(q_1, q_2) \to (q_1 + 1, q_2) = \lambda_1$
- $(q_1, q_2) \to (q_1, q_2 + 1) = \lambda_2$
- $(q_1, q_2) \to (q_1 - 1, q_2) = \min(q_1, S) \cdot \mu_1$
  (car on peut servir au plus S clients de type 1, grâce à la préemption)
- $(q_1, q_2) \to (q_1, q_2 - 1) = \min(q_2, S - q_1) \cdot \mu_2$
  (car les clients de type 1 sont prioritaires, donc on peut servir au plus $S - q_1$ clients de type 2 s'il y a q_1 clients de type 1 dans le système)

3.4)

[4]:
```python
#Focntion qui calcule la loi de Erlang-B theorique (sans la factorielle)
def erlang_b(rho, S):
    value = 1.0
    denominator = value
    for n in range(1, S + 1):
        value *= rho / n
        denominator += value
    return value / denominator
```

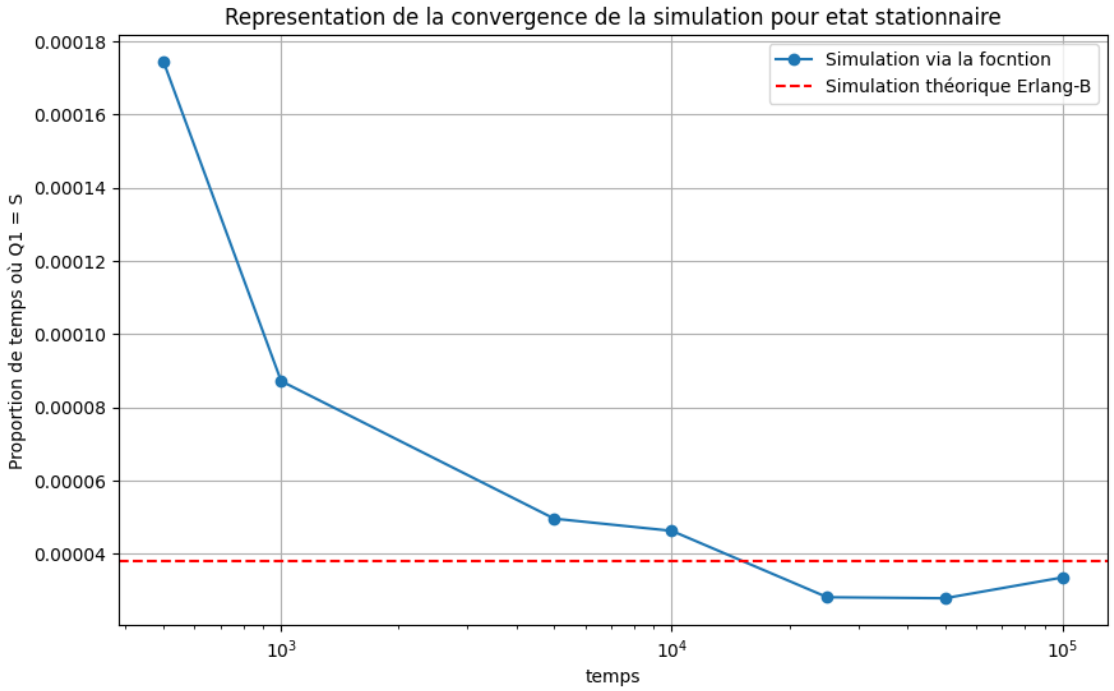
[5]:
```python
#Fonction qui fais la simulation du systeme avec 2 type de clients T_1 et T_2
def simulation(temps_simulation, S, mu1, mu2, lambda1, lambda2):
    np.random.seed(0)
    temps = 0
    Q1 = 0
    Q2 = 0
    historique_temps = []
    Q1_list = []
    Q2_list = []

    duree_pleine_charge_Q1 = 0
    arrivées_type2 = []
    attente_totale_t2 = 0
    nb_servis_t2 = 0

    while temps < temps_simulation:
        liste_evenements = []

        # Arrivée de type 1 ssi pas deja plein
```

```
        if Q1 < S:
            liste_evenements.append(('arriver_T1', lambda1)) #Formule de␣
↪transition (q1,q2) -> (q1+1,q2)
        else:
            liste_evenements.append(('arrive_T1_perdu', 0))  #Client type 1␣
↪perdu (Q1=S)

        liste_evenements.append(('depart_T1', min(Q1, S) * mu1)) #Formule de␣
↪transition (q1,q2) -> (q1-1,q2)
        liste_evenements.append(('arriver_T2', lambda2)) #Formule de transition␣
↪(q1,q2) -> (q1,q2+1)
        liste_evenements.append(('depart_T2', min(Q2, S - Q1) * mu2)) #Formule␣
↪de transition (q1,q2) -> (q1,q2-1)
        #print(evenements)

        evenements = []
        taux = []

        #Separe pour avoir nom et valeur en 2 variables
        for nom, val in liste_evenements:
            evenements.append(nom)
            taux.append(val)

        taux_total = sum(taux) #Pour voir le parametre de mon exp pour le␣
↪prochain evenement
        if taux_total == 0:
            break

        # Temps jusqu'au prochain événement
        dt = np.random.exponential(1 / taux_total)
        temps += dt

        # Si Q1 occupe tous les serveurs, mesure la durée de saturation
        if Q1 == S:
            duree_pleine_charge_Q1 += dt

        prochain_event = np.random.choice(evenements, p=np.array(taux) /␣
↪taux_total) #Formule de probabilité de transition

        # Mise a jour du système selon l'événement
        if prochain_event == 'arriver_T1' and Q1 < S:
            Q1 += 1
        elif prochain_event == 'depart_T1' and Q1 > 0:
            Q1 -= 1
        elif prochain_event == 'arriver_T2':
            Q2 += 1
```

```
            arrivées_type2.append(temps)
        elif prochain_event == 'depart_T2' and Q2 > 0 and min(Q2, S - Q1) > 0:
            Q2 -= 1
            nb_servis_t2 += 1
            if arrivées_type2:
                arrival_time = arrivées_type2.pop(0)
                attente_totale_t2 += temps - arrival_time

        historique_temps.append(temps)
        Q2_list.append(Q2)
        Q1_list.append(Q1)

    # Calcul final du temps d'attente moyen pour les clients type 2
    if nb_servis_t2 > 0 :
        moy_attente_t2 = attente_totale_t2 / nb_servis_t2
    else :
        moy_attente_t2 = 0

    return (duree_pleine_charge_Q1 / temps_simulation), historique_temps,␣
↪Q2_list, Q1_list, moy_attente_t2
```

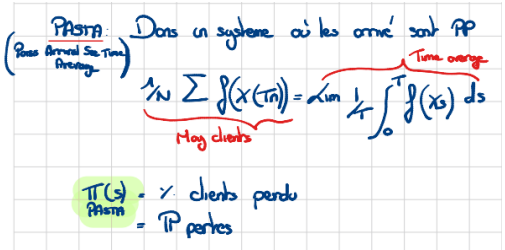
[28]:
```
liste_durees = [500, 1000,5000, 10000,25000,50000, 100000] # Paramètres de la␣
↪simulation pour voir la convergence

# Settings de l'annoncé pour le systeme
S = 10
mu1 = 2
mu2 = 1
lambda1 = 4
lambda2 = 3
rho1 = lambda1 / mu1

# Simulation des 2 cas (theorique et empirique) pour comparaison
theorique = erlang_b(rho1, S)
simulation_converg = [simulation(duree, S, mu1, mu2, lambda1, lambda2)[0] for␣
↪duree in liste_durees]

#Partie plot de la convergence
plt.figure(figsize=(10, 6))
plt.plot(liste_durees, simulation_converg, marker='o', label='Simulation via la␣
↪focntion')
plt.axhline(y=theorique, color='r', linestyle='--', label='Simulation théorique␣
↪Erlang-B')
plt.xscale('log')
plt.xlabel('temps')
plt.ylabel('Proportion de temps où Q1 = S')
```

```
plt.title('Representation de la convergence de la simulation pour etat␣
 ↪stationnaire')
plt.legend()
plt.grid(True)
plt.show()
```



On voit très bien la convergence du système vers la valeur théorique quand t augmente.

3.4 - Suite) Comme vu en cours :



$$\frac{1}{T}\int_0^T \mathbf{1}_{\{S\}}(x_1(s))\,ds$$

Représente la proportion du temps durant laquelle tous les serveurs sont occupés par des clients de type 1.
D'après la propriété PASTA, les arrivées de clients de type 1 voient le système dans une configuration

stationnaire du processus $x_1(t)$.
De plus, le processus $x_1(t)$ (ou $Q_1(t)$) est irréductible et récurrent.
Donc, comme précisé dans votre livre (2.5.2) et en cours, ce processus est ergodique, et d'après le théorème, on a la convergence :

$$\frac{1}{T}\int_0^T \mathbf{1}_{\{S\}}(x_1(s))\,ds \xrightarrow[T\to\infty]{} \pi_S(s)$$

Pour $\pi_S(s)$:

$$\pi_S(s) = \frac{\rho_1^S/S!}{\sum_{j=0}^{S}\rho_1^j/j!} \quad \text{avec} \quad \rho_1 = \frac{\lambda_1}{\mu_1}$$

Probabilité qu'un client de type 1 arrivant dans le système trouve tous les serveurs occupés, donc perdu.

4 Stationnarité

4.1) L'inégalité correspond à : $\rho_2 + \mathbb{E}[Q_1] < S$, pour une charge ρ_1, car $\mathbb{E}[Q_1] = \mathbb{E}[S_1] = \rho_1$ représente le nombre moyen de clients de type 1 dans le système.
En utilisant la fonction `mean_number_waiting_customers()`, on peut calculer $\mathbb{E}[Q_1]$.

Si je fixe $\lambda_1 = 4$ et $\mu_2 = 1$ (comme défini dans l'exercice 3.4), je vais essayer de déterminer la valeur de λ_2 telle que la condition $\rho_2 + \mathbb{E}[Q_1] < S$ soit à la limite de saturation. Donc,

On a :

$$\lambda_2 = (S - \mathbb{E}[Q_1]) \cdot \mu_2 = 8$$

Cela veut dire que si je définis mon système avec les paramètres suivants, le système ne devrait pas admettre de régime stationnaire car je ne respecte pas l'inégalité, et il va se saturer quand t augmente :

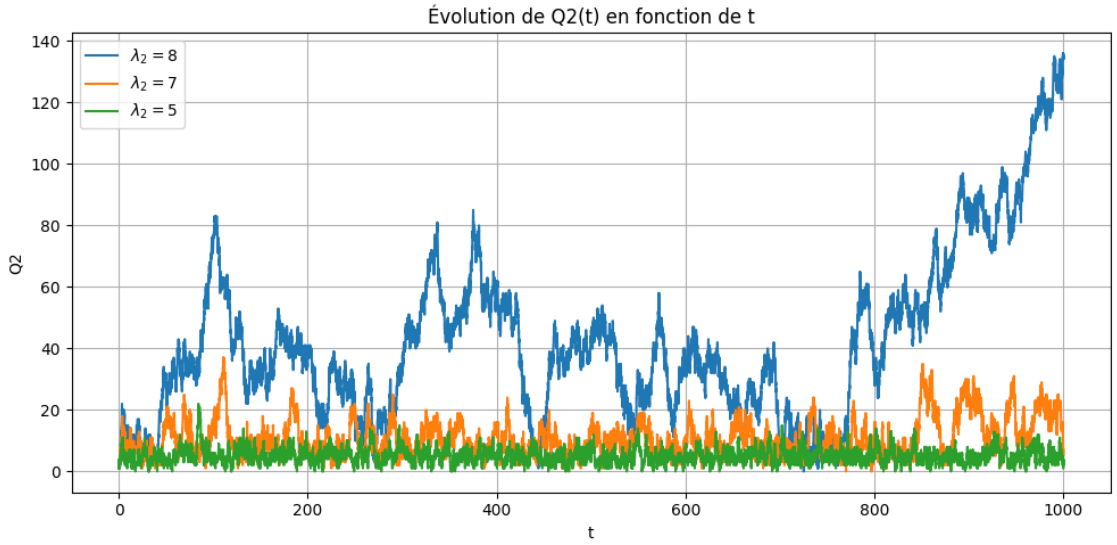
- $\lambda_1 = 4$
- $\lambda_2 = 8$
- $\mu_1 = 2$
- $\mu_2 = 1$
- $S = 10$

Voici le code et le graphe qui permettent de vérifier cette inégalité avec le nombre de clients de type 2 dans le système :

[11]:
```
#Settings de l'exo 3.4
S = 10
mu1 = 2
mu2 = 1

couple_lambda= [
    (4, 8),  #\rho_2 + \mathbb{E}[Q_1] = S
    (4, 7),  #\rho_2 + \mathbb{E}[Q_1] < S (tout juste inferieur)
```

```
    (4, 5),  #\rho_2 + \mathbb{E}[Q_1] < S (Stationnaire)

]

plt.figure(figsize=(10, 5))

#test le systeme sur les differents couple de \lambda_{i,j}
for _, (lambda1, lambda2) in enumerate(couple_lambda):
    second_terme = mean_number_waiting_customers(lambda1, mu1, S) #Calcul le␣
↪nombre moyen de client type 1 (\mathbb{E}[Q_1])
    charge_systeme = (lambda2 / mu2) + second_terme #Partie de gauche de␣
↪l'innegalité
    print(f"Charge {charge_systeme}")

    _, temps, q2, _ , _= simulation(1000, S, mu1, mu2, lambda1, lambda2)␣
↪#Resultats de Q(temps) pour le \lambda_2 choisi
    label = f"$\\lambda_2 = {lambda2}$"
    plt.plot(temps, q2, label=label)

# Partie plot
plt.xlabel("t")
plt.ylabel("Q2")
plt.title("Évolution de Q2(t) en fonction de t")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
Charge 9.999923619664118
Charge 8.999923619664118
Charge 6.999923619664118
```



Avec cette simulation, on peut clairement vérifier que cette inégalité est bien respectée.
On remarque que, pour la valeur de λ_2 au seuil d'égalité, le système diverge complètement avec le temps.

Par contre, pour des valeurs légèrement inférieures au seuil (même très proches), on commence à observer un régime stationnaire, comme sur la courbe orange et surtout sur la verte.

5 Calcul de la probabilité stationnaire

5.1) Je fixe le vecteur $\Pi = (x_0, x_1, x_2, \ldots, x_j, x_{j+1}, \ldots, x_L)$.
Pour le générateur infinitésimal donné en 6.1 de l'article, les équations d'équilibre sont données par $\Pi \cdot Q = 0$, soit :

$$x_{j-1}C + x_jB_j + A_{j+1}x_{j+1} = 0$$

Je veux montrer que la suite $x_j = x_S R^{j-S}$ est solution des équations d'équilibre au-delà du rang S.

Je remplace alors :

$$\begin{aligned} x_{j-1} &= x_S R^{j-S} R^{-1} \\ x_j &= x_S R^{j-S} \\ x_{j+1} &= x_S R^{j-S} R \end{aligned}$$

En injectant dans l'équation d'équilibre :

$$x_S R^{j-S} R^{-1} C + x_S R^{j-S} B_j + A_{j+1} x_S R^{j-S} R = 0$$

En factorisant :

$$x_S R^{j-S} \left(R^{-1}C + B_j + A_{j+1}R \right) = 0$$

Donc, pour que cette équation soit satisfaite, il suffit que la matrice :

$$R^{-1}C + B_j + A_{j+1}R = 0$$

Et d'après l'énoncé, il existe bien une matrice R_S telle que cette équation matricielle soit vérifiée.

J'en conclus donc que la suite $x_j = x_S R^{j-S}$ est bien solution des équations d'équilibre pour tout $j \geq S$.

5.2) Pour montrer

$$x_{S-1} = x_S(\tilde{B}_S + R\tilde{A}_S)$$

Je vais repartir de l'équation du dessus et je vais fixer $j = S$ car elle est valable pour $j \geq S$.
Ce qui s'écrit :

$$x_{S-1}C + x_S B_j + A_{S+1}x_{S+1} = 0$$

Je suppose que les matrices A_j sont les mêmes pour $j \geq S$, donc $A_j = A_S$ pour tous $j \geq S$.

En isolant x_{S-1} et en remplaçant $x_{S+1} = x_S R$, j'obtiens :

$$x_{S-1} = x_S(B_S + A_{S+1}R)C^{-1}$$

Je sais depuis l'article que $C = \lambda_2$ et que $\tilde{B}_S = \frac{1}{\lambda_2}B_S$ (de même pour A).

Donc en remplaçant, je trouve l'égalité :

$$\boxed{x_{S-1} = x_S(\tilde{B}_S + \tilde{A}_{S+1}R)}$$

5.3) (Pour être honnête, je me suis aidé de ChatGPT pour le début de la question, étant donné que j'étais un peu bloqué.)

En partant de l'article, j'ai identifié la correspondance $T_j = R^{(j+1)}$ grâce à la formule donnée :

$$\pi_j = \pi_{j-1}R^{(j)} \implies T_j = R^{(j+1)}$$

Via la formule de l'article :

$$R^{(j)} = -C\left(B_j + R^{(j+1)}A_{j+1}\right)^{-1}$$

Je remplace dans ma formule :

$$T_{j-1} = -C\left(B_j + T_j A_{j+1}\right)^{-1}$$

Ce que j'ai compris, c'est qu'à partir de x_S (avec $T_S = R$), on peut reconstruire tous les x_j pour $j < S$ en appliquant cette récurrence, en utilisant uniquement les matrices A_j, B_j et C.

5.4) Pour montrer que : $x_0\left((M - \lambda_2 I) + T_0 A_1\right) = 0$

Je pars de l'équation d'équilibre et je fixe $j = 0$, ce qui donne :

$$x_{-1}C + x_0 B_0 + A_1 x_1 = 0$$

Puis, en utilisant la récurrence obtenue dans la question précédente, on a :

$$x_1 = x_0 T_0$$

En injectant cette expression dans l'équation :

$$x_0(B_0 + T_0 A_1) = 0 \quad \text{tel que :} \quad B_0 = M - \lambda_2 I$$

Je me retrouve bien avec :

$$\boxed{x_0\left((M - \lambda_2 I) + T_0 A_1\right) = 0}$$

5.5) La première approche que j'ai eue, est celle-ci :

En sachant que nous avons un $L \to \infty$ pour le vecteur Π, par définition, on doit avoir :

$$\sum_{j=0}^{\infty} x_j = 1$$

- Pour $j \geq S$, on a $x_j = x_S R^{j-S}$:

$$\sum_{j=S}^{\infty} x_j = x_S \sum_{k=0}^{\infty} R^k = \sum_{j=S}^{\infty} x_j = x_S (I - R)^{-1}$$

  Car on sait que : $\sum_{k=0}^{\infty} R^k = (I - R)^{-1}$

- Pour $j < S$, on sait $x_j = x_{j+1} T_j^{-1}$, donc en remplaçant par récurrence :

$$\sum_{j=0}^{S-1} x_j = \sum_{j=0}^{S-1} x_S \left(\prod_{k=j+1}^{S} T_k^{-1} \right)$$

Si on reprend la normalisation avec ces 2 termes, on a :

$$x_S \left[\sum_{j=0}^{S-1} \left(\prod_{k=j+1}^{S} T_k^{-1} \right) + (I - R)^{-1} \right] = x_s \cdot P = 1$$

Donc $x_s = P^{-1}$

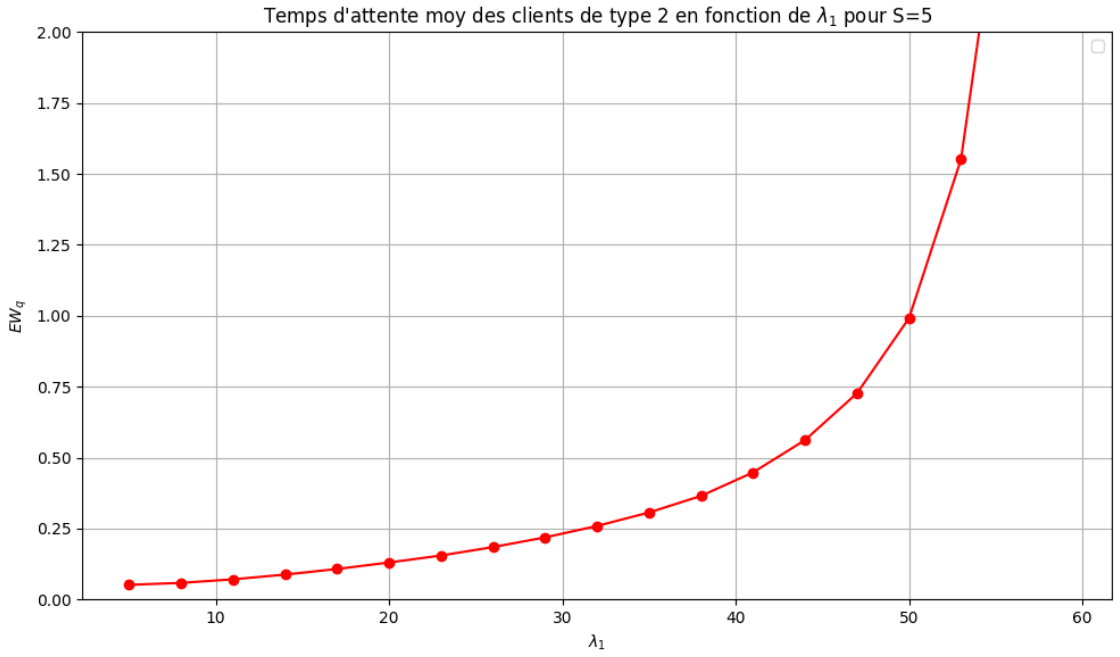
Mais cela m'a l'air compliqué, et grâce à l'exercice du dessus je peux aussi trouver :

$$x_S = x_0 \prod_{j=1}^{S} T_{j-1} = x_0 T_0 T_1 \cdots T_{S-1}$$

Honnêtement je suis un peu bloqué sur cette question.

5.6) Pour reproduire la Figure 2 de [1], j'ai dû modifier le code de la fonction `simulation()` pour avoir EW_q = `moy_attente_t2`.
En ayant pris les paramètres donnés dans l'article, j'ai écrit le bout de code ci-dessous permettant de faire cette figure.

```
[ ]: #Parametres qui sont donnés dans m'article en 6.2
S = 5
mu1 = 4
mu2 = 20
lambda2 = 8
temps_simulation = 30000
lambda1_values = np.arange(5, 60, 3)

#Boucle sur les valeurs de \lambda_1 et ajoute le temps d'attente associé des␣
 ↪type 2.
ewq_each_lambda1 = []
for lambda1 in lambda1_values:
    _, _, _, _, ewq = simulation(temps_simulation, S, mu1, mu2, lambda1, lambda2)
    ewq_each_lambda1.append(ewq)

#Plot le graph
plt.figure(figsize=(10, 6))
plt.plot(lambda1_values, ewq_each_lambda1, 'o-', color='red')
plt.xlabel(r"$\lambda_1$")
plt.ylabel(r"$EW_q$")
plt.title(r"Temps d'attente moyen des clients de classe 2 en fonction de␣
 ↪$\lambda_1$ pour S=5")
plt.yticks(np.arange(0, 2.25, 0.25))
plt.ylim(0, 2)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



On voit clairement que la courbe est très proche de celle de la Figure 2 de [1].

- Pour $\lambda_1 = 50$, on a bien $EW_q = 1$
- Pour $\lambda_1 = 30$, on a bien $EW_q \approx 0.25$
- Pour $\lambda_1 \geq 50$, on a bien EW_q qui diverge $\implies$ saturation du système.

Cela montre que ma simulation est cohérente avec la Figure 2 de [1].

6 Canaux de garde

6.1) Dans la configuration avec des serveurs de garde, le processus (Q_1, Q_2) n'est pas un processus de Markov car, comme dans l'exemple vu en cours avec les canaux de garde pour les Handovers :

- Tant que #serveurs disponibles $> G$, les arrivées ont un taux $\lambda_1 + \lambda_2$
- Dès que #serveurs disponibles $< G$, les arrivées ont un taux λ_1

Dans ce cas, à chaque arrivée, le système doit connaître le type de client sur les serveurs.
Mais cette information n'est pas accessible uniquement via (Q_1, Q_2).

On ne peut donc pas définir le prochain état à partir de l'état courant (comparé au cas de préemption).
Par définition, ce n'est donc pas un processus de Markov.

6.2) Pour modéliser ce processus en processus de Markov (comme dans le cas MMPP vu en cours), je définis le processus $X(t) = (i, j)$, où :

- $i =$ #clients de type 1 en service,
- $j =$ #clients de type 2 en service.

Je note également : $n = i + j$: le nombre total de serveurs occupés

On définit les taux d'arrivée en fonction de n et en respectant la condition de G :

$$\lambda(n) = \begin{cases} \lambda_1 + \lambda_2 & \text{si } n \leq S - G \\ \lambda_1 & \text{si } S - G < n < S \end{cases}$$

On définit les taux de départ :

$$\mu(n) = n \cdot (\mu_1 + \mu_2)$$

Comme dans l'exercice 3.3, les transitions sont données par :

- $(i, j) \to (i + 1, j) = \lambda_1$
- $(i, j) \to (i - 1, j) = i \cdot \mu_1$
- $(i, j) \to (i, j + 1) = \lambda_2$
- $(i, j) \to (i, j - 1) = j \cdot \mu_2$

6.3) Dans le cas avec préemption, les clients de type 1 peuvent remplir tout le système car ils peuvent éjecter des clients de type 2.

Mais dans le cas des canaux de garde, les clients de type 1 ne peuvent pas éjecter un client de type 2.
Ils ne peuvent entrer que si un serveur est libre. Il y a donc plus de pertes, car il y a des clients de type 2 dans les serveurs.

Ça implique : $\mathbb{E}[Q_1] < \rho_1$ alors qu'avec préemption $\mathbb{E}[Q_1] = \rho_1$

En reprenant les équations d'un exercice plus haut :

- Avec préemption :
$$\rho_1 + \rho_2 < S$$
- Canaux de garde :
$$\mathbb{E}[Q_1] + \rho_2 < S$$

On a donc bien :

$$\boxed{\rho_2 + \rho_1 \geq \rho_2 + \mathbb{E}[Q_1]}$$

Cela montre que le modèle sans préemption laisse plus de marge pour $\rho_2 \implies$ moins contraignant.

References

[1] *Evsey Morozov et al. « Modified Erlang loss system for cognitive wireless networks »*, Disponible à : https://partage.imt.fr/index.php/s/a4SSkqYgXR88f5J

[2] *Martin Hairer, Ergodic Properties of Markov Processes*, Disponible à : https://www.hairer.org/notes/Markov.pdf

7 Note

Merci pour ce cours et ce projet, je les ai trouvés vraiment intéressants !