DM-SERPOLLET-VALETTE

Created	@9 mai 2022 14:51
Class	CS332
Type	DM
Materials	

Compte Rendu

Tout le code source commenté est fourni en complément de ce compte rendu.

Exercice 1 :

Question 1 :

On fait une simulation grâce à la fonction suivante. Cette fonction renvoi au choix, un tableau avec l'instant d'arrivée de toutes les voitures ou un tableau avec les intervalles inter-arrivées.

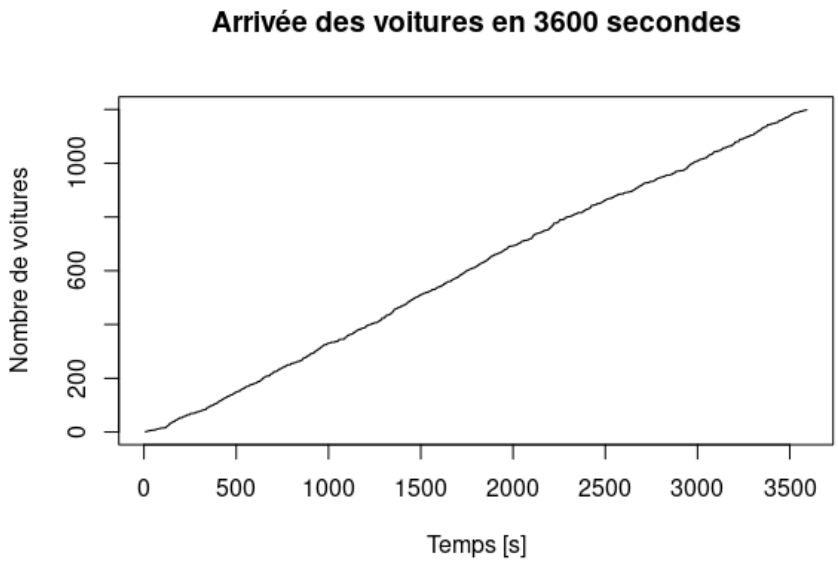
```
#simulation d'arrivée des voitures en t secondes
nb_voiture_fct <- function(t, type_return) {
  duree <- 0
  index <- 1
  tableau <- list()
  inter <- list()
  inter_ = rexp(1,lambda)
  duree <- duree + inter_
  while(duree < t){
    tableau[index] <- duree
    inter[index] <- inter_
    index <- index + 1
    inter_ = rexp(1,lambda)
    duree <- duree + inter_
  }
  if(type_return == "duree"){
    return(tableau)
  }
  if(type_return == "inter"){
    return(inter)
```

```
  }
}
```

On crée une fonction affichant le graphe des instants d'arrivée et le lambda expérimental obtenu :

```
Dessine_N <- function (t){
  tableau = nb_voiture_fct(t,"duree")
  list = 1:length(tableau)
  plot(tableau, list
       ,type ="s"
       ,xlab="Temps [s]"
       ,ylab="Nombre de voitures"
       ,main= paste("Arrivée des voitures en", t, "secondes\n"))
  #calcul et affichage de l'estimation du lambda (lambda experimental)
  cat("Lambda experimental = ", length(tableau)/t, "\n\n")
}
```

Le résultat est le suivant :



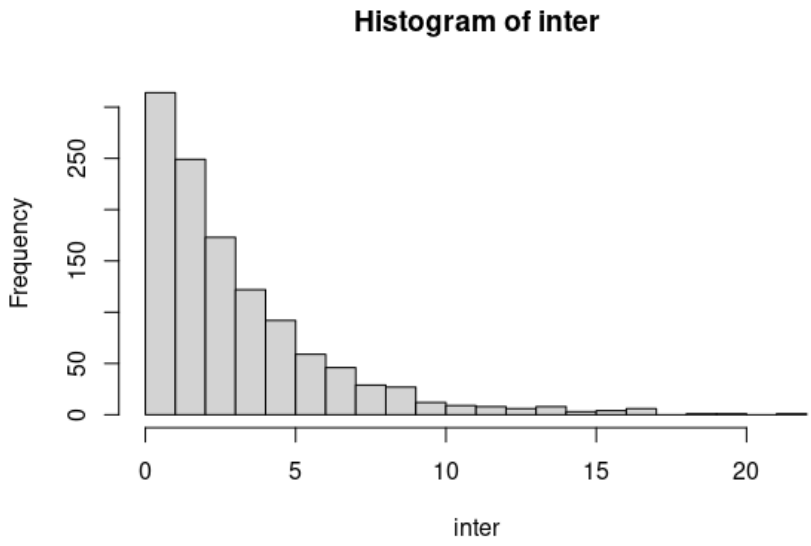

```
> Dessine_N(3600)
Lambda experimental =  0.3375
```

On obtient un lambda expérimental qui correspond assez bien avec le lambda théorique attendu (1/3).

On crée aussi une fonction qui affiche l'histogramme des intervalles inter-arrivées :

```
#calcul et affichage de l'histogramme, verification de la distribution exponentielle
Dessine_Histo <- function(){
  inter = unlist(nb_voiture_fct(duree_max,"inter"))
  histo <- hist(inter, breaks = max(inter), plot = T)
}
```

Le résultat est le suivant :



On reconnait bien une distribution d'une loi exponentielle.

Question 2 :

On crée une fonction comptant le nombre moyen d'arrivées pour une durée et des répétitions voulue :

```
#calcule le nombre moyen d'arrive pour un temps en seconde (moyenne calculée en nb_repet répétitions)
moyenne_arrive <- function (temps, nb_repet){
  nb_arrive <- 0
  for (i in (1:nb_repet)) {
    nb_arrive <- nb_arrive + length(nb_voiture_fct(temps,"duree"))
  }
  estimation = nb_arrive / nb_repet
  return(estimation)
}

Compare_moyenne <- function(t, n){
  m_e = moyenne_arrive(t,n)
  m_t = lambda*t
  diff = abs((m_e - m_t)*100/m_t)
  cat("Moyenne experimentale d'arrivé sur", t, "secondes (",n, "répétitions):\t", m_e ,"\n")
  cat("Moyenne théorique d'arrivé sur", t, "secondes :\t\t\t\t", m_t,"\n")
  cat("Différence de",diff,"%\n\n")
  #L'espérance dune variable de Poisson est λ
  return(diff)
}
```

Le résultat est le suivant :

```
> Compare_moyenne(3600,1000)
Moyenne experimentale d'arrivé sur 3600 secondes ( 1000 répétitions):    1199.454
Moyenne théorique d'arrivé sur 3600 secondes :                           1200
Différence de 0.0455 %
```

On remarque que pour 1000 répétitions on obtient un écart de seulement 0.045 % par rapport au résultat théorique attendu (1200).

On souhaite maintenant trouver des résultats similaires avec moins de répétitions, mais une durée plus longue.

C’est possible car (d'après le cours) l’estimateur λˆ ne dépend que du nombre total d'arrivées, donc avec un T plus grand, ce nombre grandi aussi et c'est plus précis.

Voici la fonction permettant de comparer :

```
test_quest_ii <- function(n){
  t1 <- 0
  t2 <- 0
  t3 <- 0
  for (i in (1:n)) {
    t1 <- t1 + Compare_moyenne(3600,100)
    t2 <- t2 + Compare_moyenne(3600,5)
    t3 <- t3 + Compare_moyenne(72000,5)
    print(i)
  }
  t1 = t1 / n
  cat("Différence pour t = 3600s et n = 100 :\t",t1,"%\n")
  t2 = t2 / n
  cat("Différence pour t = 3600s et n = 5 :\t",t2,"%\n")
  t3 = t3 / n
  cat("Différence pour t = 72000s et n = 5 :\t",t3,"%\n\n")
}
```

Et on obtient le résultat suivant :

```
Différence pour t = 3600s et n = 100 :   0.2301417 %
Différence pour t = 3600s et n = 5 :     1.035667 %
Différence pour t = 72000s et n = 5 :    0.2149417 %
```

On constate bien que lorsque qu’on réduit le nombre de répétitions sans augmenter la durée, on perd en précision (presque 5 fois moins bien).

On regagne cette précision en augmentant la durée. Ici, on divise par 20 le nombre de répétitions et on multiplie par 20 la durée, la précision est ainsi la même.

Question 3 :

A)

On cherche à vérifier que, dans le cas général, les instants d’arrivée sont distribués selon une loi Gamma.

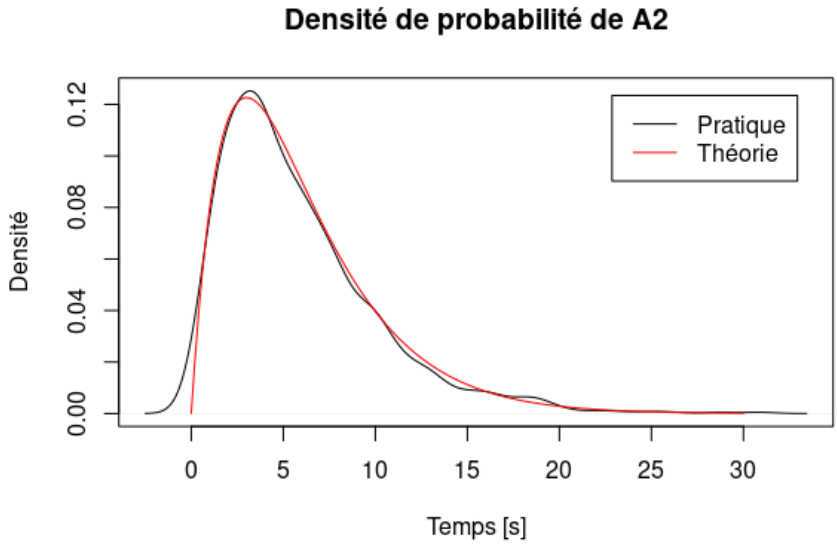
On choisit d’étudier les instants d’arrivée 2, 5, 50 et 500.

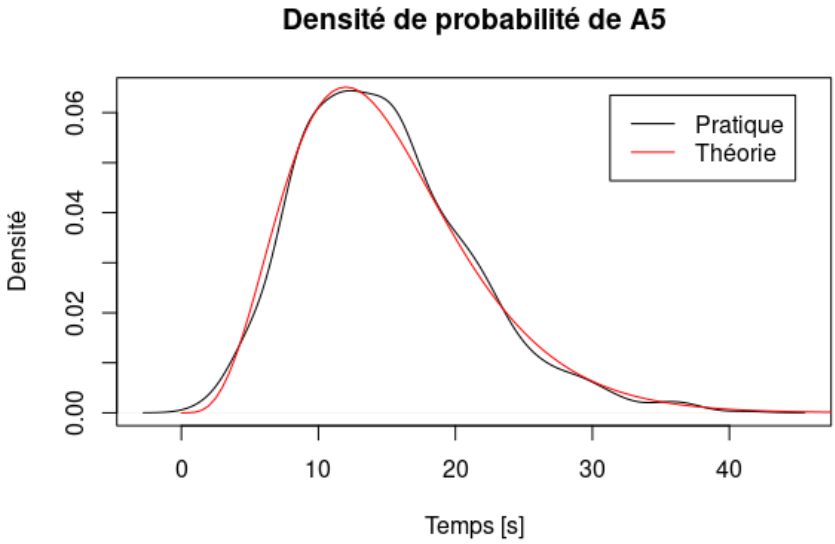
Voici la fonction permettant de récolter ces instants d’arrivée sur un grand nombre de simulations et d’afficher les densités de probabilité pour les comparer aux lois Gamma correspondantes :

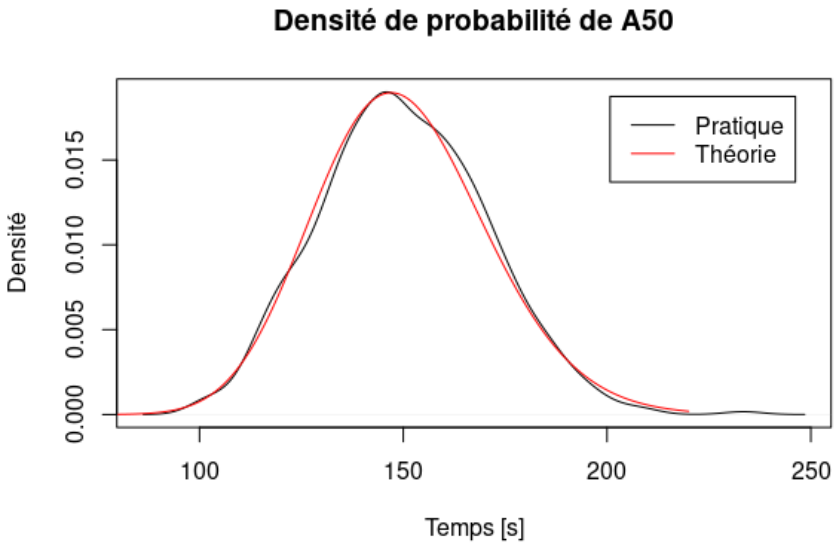
```
#on prend les arrivée 2 5 50 et 500 et on compare leur densité de probabilité à une loi gamma censée correspondre
compare_loi_gamma <- function(){
  A2 <- c()
  A5 <- c()
  A50 <- c()
  A500 <- c()
  for (i in 1:1000) {
    tab = nb_voiture_fct(3600,"duree")
    A2[i] = tab[2]
    A5[i] = tab[5]
    A50[i] = tab[50]
    A500[i] = tab[500]
  }

  densite_A2 = density(unlist(A2))
  plot(densite_A2
       ,xlab="Temps [s]"
       ,ylab="Densité"
       ,main= "Densité de probabilité de A2")
  x_vect = seq(0,30,by = 0.2)
```

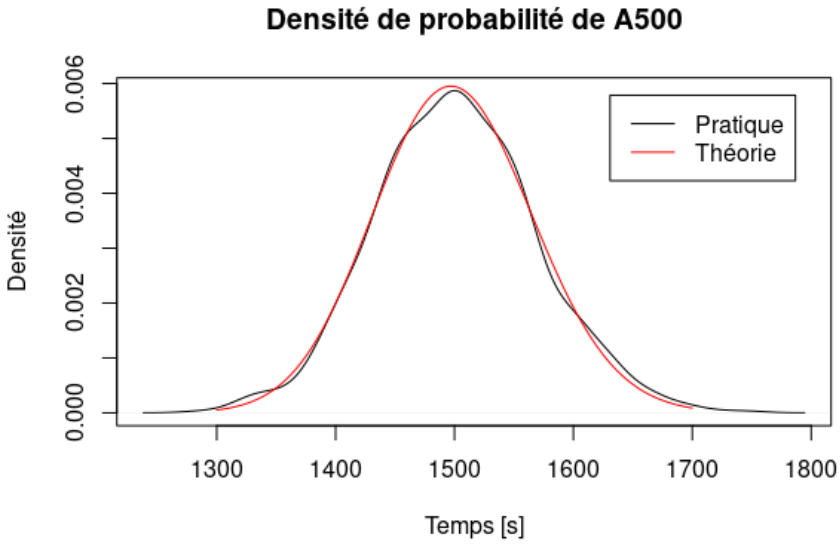
```
  lines(x_vect,dgamma(x_vect, 2, lambda), col = "red")
  legend("topright", inset = .05, legend=c("Pratique", "Théorie"),
         col=c("black", "red"), lty = 1)


  densite_A5 = density(unlist(A5))
  plot(densite_A5
       ,xlab="Temps [s]"
       ,ylab="Densité"
       ,main= "Densité de probabilité de A5")
  x_vect = seq(0,50,by = 0.5)
  lines(x_vect,dgamma(x_vect, 5, lambda), col = "red")
  legend("topright", inset = .05, legend=c("Pratique", "Théorie"),
         col=c("black", "red"), lty = 1)


  densite_A50 = density(unlist(A50))
  plot(densite_A50
       ,xlab="Temps [s]"
       ,ylab="Densité"
       ,main= "Densité de probabilité de A50")
  x_vect = seq(80,220,by = 1)
  lines(x_vect,dgamma(x_vect, 50, lambda), col = "red")
  legend("topright", inset = .05, legend=c("Pratique", "Théorie"),
         col=c("black", "red"), lty = 1)


  densite_A500 = density(unlist(A500))
  plot(densite_A500
       ,xlab="Temps [s]"
       ,ylab="Densité"
       ,main= "Densité de probabilité de A500")
  x_vect = seq(1300,1700)
  lines(x_vect,dgamma(x_vect, 500, lambda), col = "red")
  legend("topright", inset = .05, legend=c("Pratique", "Théorie"),
         col=c("black", "red"), lty = 1)
}
```

Les résultats sont les suivants :




Densité de probabilité de A5
Pratique
Théorie
Densité
0.00
0.02
0.04
0.06
0
10
20
30
40
Temps [s]


Densité de probabilité de A50
Pratique
Théorie
Densité
0.000
0.005
0.010
0.015
100
150
200
250
Temps [s]



On constate donc que les courbes correspondent. On peut alors dire que dans le cas général, la distribution des instants d'arrivées suit une loi Gamma.

B)

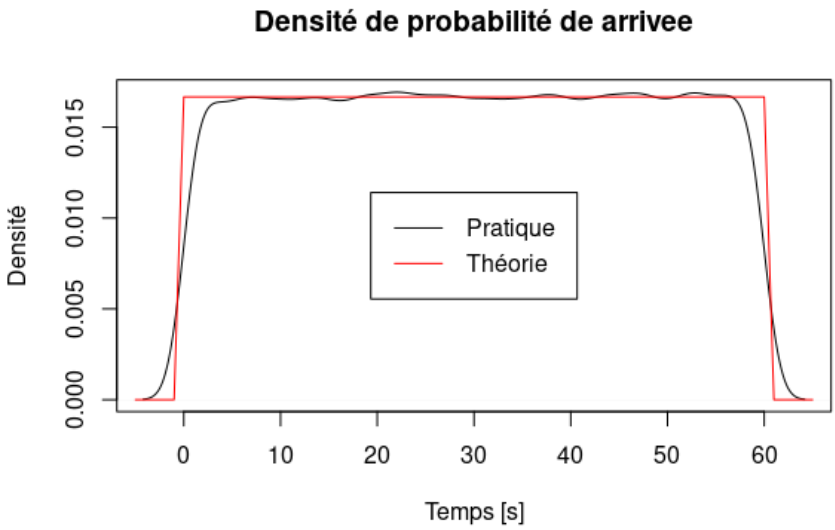
On se place maintenant dans le cas particulier ou N(60) = 20.

On veut vérifier que les instants d'arrivées sont distribués de manière uniforme sur l'intervalle [0,60].

Voici la fonction permettant de le faire :

```
#verifie que les interval inter-arrivée vérifie une loi uniforme quand on connait le nombre totale d'arrivée
verif_uniforme <- function(n){
  arrivee <- c()
  index <- 0
  for (i in 1:n) {
    tab = unlist(nb_voiture_fct(60,"duree"))
    if (length(tab)==20){
      arrivee <- c(arrivee,tab)
      index <- index +1
    }
  }
  densite_arrivee = density(arrivee)
  plot(densite_arrivee
       ,xlab="Temps [s]"
       ,ylab="Densité"
       ,main= "Densité de probabilité de arrivee")
  x_vect = seq(-5,65)
  lines(x_vect,dunif(x_vect, min = 0, max =60 ), col = "red")
  legend("center", inset = .05, legend=c("Pratique", "Théorie"),
         col=c("black", "red"), lty = 1)
}
```

Les résultats :

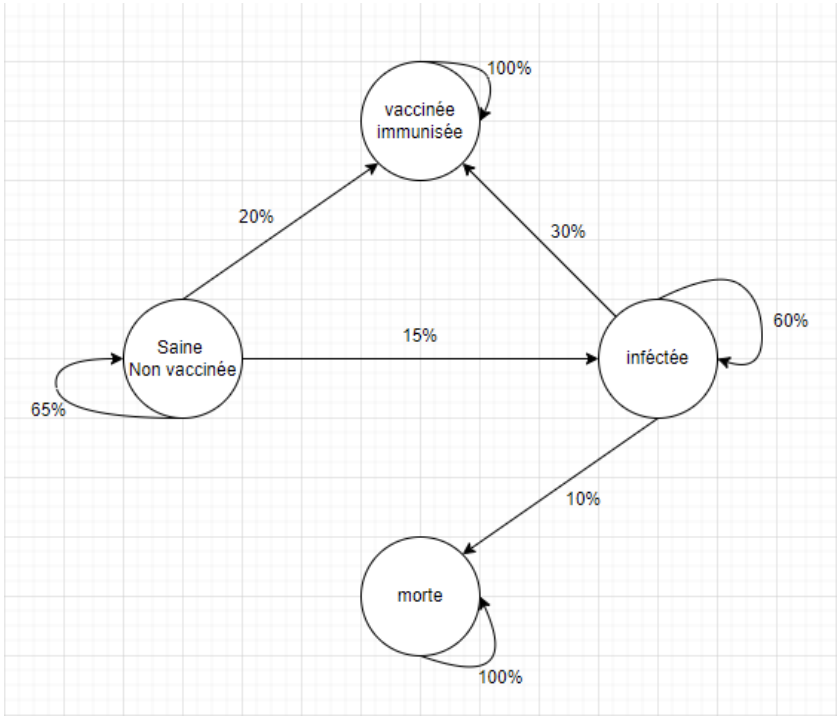


Comme les courbes correspondent, on peut dire que les instants d'arrivées sont distribués de façon uniforme sur l'intervalle [0,60].

Exercice 2 :

Question 1 :

Chaine de Markov :



Matrice de transition :

$$P = \begin{array}{c} \begin{matrix} \text{Sains, non vaccinés} & \text{Vaccinés / immunisés} & \text{Infectés} & \text{Morts} \end{matrix} \\ \begin{pmatrix} 0.65 & 0.2 & 0.15 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0.3 & 0.6 & 0.1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} \text{sains, non vaccinés} \\ \text{Vaccinés / immunisés} \\ \text{infectés} \\ \text{morts} \end{matrix}$$

Dans la matrice de transition, nous considérons les "chances" de passer d'un état à un autre au bout de la semaine. Nous considérons que toutes les personnes sont saines et non vaccinées. Pour déterminer une case de la matrice, nous regardons la probabilité qu'une personne avec l'état "ligne" passe à l'état "colonne" durant la semaine suivante.
L'unité est la semaine

Question 2 :

Théorie :

On suppose que lors de la première semaine, toutes les personnes sont saines et non vaccinées, on a donc : $\Pi_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$

On veut simuler une évolution de 5 semaines, on va donc calculer $\Pi_0 \times P^5$

$$P^5 = \begin{pmatrix} 0.116029 & 0.703154 & 0.114807 & 0.066009 \\ 0 & 1 & 0 & 0 \\ 0 & 0.86907 & 0.4815 & 0.28969 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

On aura donc : $\Pi_0 \times P^5 = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0.116029 & 0.703154 & 0.114807 & 0.066009 \\ 0 & 1 & 0 & 0 \\ 0 & 0.86907 & 0.4815 & 0.28969 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

et donc : $\Pi_0 \times P^5 = \begin{pmatrix} 0.116 & 0.703 & 0.114 & 0.066 \end{pmatrix}$

Dans notre cas, au bout de 5 semaines (c'est-à-dire 1 mois) le bilan est de :

- 11.6 % des personnes qui sont saines et non vaccinées
- 70.3 % des personnes qui sont vaccinées ou immunisées
- 11.4 % des personnes qui sont infectées
- 6.6 % des personnes qui sont mortes

Simulation :

On utilise une fonction pour calculer un nouvel état à partir d'un état précédent grâce à une loi de probabilité uniforme.

```
# fonction pour calculer un etat a partir d'un autre
nouvel_etat <- function(etat){
  proba <- runif(1,min = 0, max = 1)
  etat_temp <- 0
  if(etat == 0){ #si on est sain et pas vacciné
    if(proba <= 0.65){
      etat_temp <- etat
    }else if(proba > 0.65 && proba <= 0.85){
      etat_temp <- 1
    }else{
      etat_temp <- 2
    }
  }else if(etat == 1){#Si on est vacciné ou immunisé
    etat_temp <- 1
  }else if(etat == 2){#Si on est infécté
    if(proba <= 0.60){
      etat_temp <- etat
    }else if(proba > 0.60 && proba <= 0.90){
      etat_temp <- 1
    }else{
      etat_temp <- 3
    }
  }else{#si on est mort
    etat_temp <- etat
  }
  return (etat_temp)
}
```

On utilise une fonction pour calculer le nouvel état de chaque individu d'une population donnée.

```
changement_etat_population <- function(population){
  for(i in seq(1:length(population))){
    population[i] <- nouvel_etat(population[i])
  }
  return(population)
}
```

On utilise une fonction pour répéter les simulations sur un nombre de semaines donné.

```
simulation <- function(nb_semaine, nb_population){
  population <- rep.int(0, nb_population)
  for(i in seq(1:nb_semaine)){
    population <- changement_etat_population(population)
  }
  return(bilan_population(population = population))
}
```

Et on utilise une fonction pour retourner le taux de chaque état présent dans la population.

```
bilan_population <- function(population){
  sains <- 0
  vaccine <- 0
  infecte <- 0
  morts <- 0
  for(i in population){
    if(i == 0){
      sains <- sains + 1
    }else if(i == 1){
      vaccine <- vaccine + 1
    }else if(i == 2){
      infecte <- infecte + 1
    }else{
      morts <- morts + 1
    }
  }
  sains <- sains / length(population)
  vaccine <- vaccine / length(population)
  infecte <- infecte / length(population)
  morts <- morts / length(population)
  cat(sprintf("Sain : %s Vaccine : %s Malade : %s Mort: %s \n", sains, vaccine, infecte, morts))
}
```

En lançant la simulation sur une population de 100000 personnes et sur 5 semaines, on obtient les résultats suivants :

- 11.5 % des personnes qui sont saines et non vaccinées
- 70.3 % des personnes qui sont vaccinées ou immunisées
- 11.6 % des personnes qui sont infectées
- 6.6 % des personnes qui sont mortes

Ces résultats sont très proches de notre calcul théorique.

Question 3 :

Notre CMTD est composée de 3 classes irréductibles :

$$C_1 = \{0, 2\}$$
$$C_2 = \{1\}$$
$$C_3 = \{0\}$$

Nous devons calculer le temps de séjour dans la classe C_1 pour savoir combien de temps les personnes restent dans les états de transitions avant de mourir ou d'être immunisé.

On a l'équation suivante :

$$\begin{pmatrix} T_0 \\ T_1 \end{pmatrix} = (1) + \begin{pmatrix} 0,65 & 0,15 \\ 0 & 0,6 \end{pmatrix} \times \begin{pmatrix} T_0 \\ T_2 \end{pmatrix}$$

On a donc les équations suivantes :

$$T_0 = 1 + 0,65T_0 + 0,15T_2$$
$$T_2 = 1 + 0,6T_2$$

Ce qui nous donne :

$$0,35T_0 = 1 + \frac{0,15}{0,4}$$
$$T_2 = \frac{1}{0,4}$$

Nous avons donc :

$$T_0 \approx 3,9286 \approx 4$$

Il faut donc environ quatre semaines pour que la population soit morte ou vaccinée.

Simulation :

On crée une fonction qui retourne le nombre de semaines qu'il faut avant d'atteindre un des états de fin.
Puis, on lance une simulation sur 10000 personnes et on trouve une moyenne de 3,9366, ce qui valide notre étude théorique.

```
# calculer le temps qu'il faut avant qu'une personne ne soit morte ou vaccinée:
nb_semaine <- function(){
  etat <- 0
  compteur <- 0
  while ((etat != 1) && (etat != 3)){
    etat <- nouvel_etat(etat)
    compteur <- compteur + 1
  }
  return (compteur)
}
simulation_nb_semaine <- function(taille_population){
  somme <- 0
  for(i in seq(1:taille_population)){
    somme <- somme + nb_semaine()
  }
  return (somme / taille_population)
}

# lancer la simulation pour 10000 personnes
# simulation_nb_semaine(10000)
```

Question 4 :

Calcul théorique :

$$f_{0,3} = p_{0,3} + p_{0,0}.f_{0,3} + p_{0,1}.f_{1,3} + p_{0,2}.f_{2,3}$$

On a donc :

$$f_{0,3} = \frac{0.15}{0.35} f_{2,3}$$

Et :

$$f_{2,3} = p_{2,3} + p_{2,0}.f_{0,3} + p_{2,1}.f_{1,3} + p_{2,2}.f_{2,3}$$

Donc :

$$f_{2,3} = \frac{0.1}{0,4}$$

Et donc :

$$f_{0,3} = \frac{0.15}{0.35}.\frac{0.1}{0,4} = 0.107$$

En théorie, chaque habitant a 10,7 % de "chance" de mourir de la maladie

Calcul pratique :

On utilise une fonction pour calculer n fois le cas de Robert puis on fait la moyenne de ses morts :

```
robert <- function(nb_test){
  nb_mort <- 0
  for(i in seq(1:nb_test)){
    etat <- 0
    while(etat != 1 && etat != 3){
      etat <- nouvel_etat(etat)
    }
    if(etat == 3){
      nb_mort <- nb_mort + 1
    }
  }
  return ((nb_mort / nb_test) * 100)
}
# simulation pour 100000 fois sur notre Robert
# robert(100000)
```

Avec ceci, on trouve une moyenne de 10,6 % contre un résultat théorique de 10,7 %. Notre marge d'erreur est très fine.

Pour conclure, on peut dire que chaque citoyen a 10,6 % de "chance" de mourir de cette maladie.

Exercice 3 :

Question 1 :

On crée plusieurs fonctions qui vont servir à déterminer les instants de départs et d'arrivée :

```
#1
#On commence par remplir 2 tableaux:
#Le premier compte les arrivées et le deuxièmes les temps de traitement
#On se base sur le premier exercice

fct_arrive <- function(temps,l, type_return){
  duree <- 0
  index <- 1
  tableau <- c()
  inter <- c()

  inter_ = rexp(1,l)

  while(duree < temps){
    duree <- duree + inter_

    tableau[index] <- duree
    inter[index] <- inter_

    index <- index + 1
    inter_ = rexp(1,l)
  }
  if(type_return == "duree"){
    return(tableau)
  }
```

```
  if(type_return == "inter"){
    return(inter)
  }
}

fct_service <- function(temps,m, type_return){
  duree <- 0
  index <- 1
  tableau <- c()
  inter <- c()

  inter_ = rexp(1,m)

  while(duree < temps){
    duree <- duree + inter_

    tableau[index] <- duree
    inter[index] <- inter_

    index <- index + 1
    inter_ = rexp(1,m)
  }
  if(type_return == "duree"){
    return(tableau)
  }
  if(type_return == "inter"){
    return(inter)
  }
```

Grâce à ces 2 tableaux, on peut calculer les instants de départ :

```
#prend en entree un temps max et 2 tableaux contenant les temps d'arrivé et les temps de service
fct_depart <- function(temps,arrivees, services, type_return){

  duree <- 0
  index <- 1
  depart_duree <- c()
  depart_inter <- c()

  interval = arrivees[1]+services[1]

  while ((duree < temps) && (index<length(arrivees))) {
    duree <- duree + interval
    depart_duree[index] <- duree
    depart_inter[index] <- interval

    index <- index +1

    if (arrivees[index] < depart_duree[index-1]){
      interval = services[index]
    }else{
      interval = (arrivees[index] - depart_duree[index-1]) + services[index]
    }
  }
  if(type_return == "duree"){
    return(depart_duree)
  }
  if(type_return == "inter"){
    return(depart_inter)
  }
}
```

Avec les instants de départ et d'arrivée, on peut déterminer le nombre de clients présent à chaque instant dans la file :

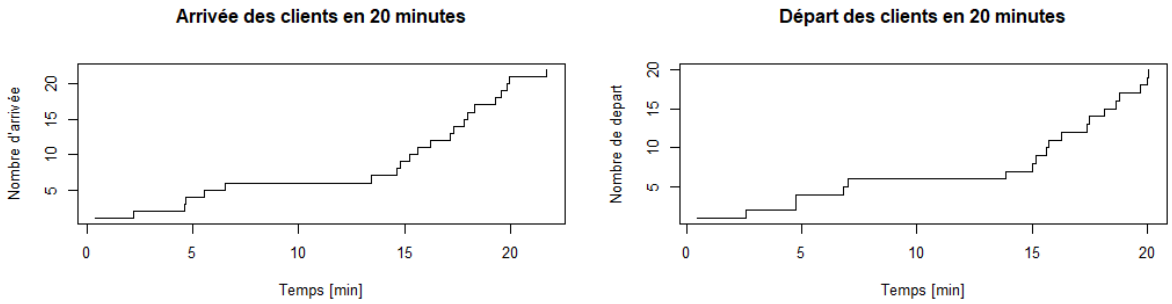
```
N <- function(t, arrivees, departs){
  temps <- c()
  nb_client <-c()

  index_arr <- 1
```

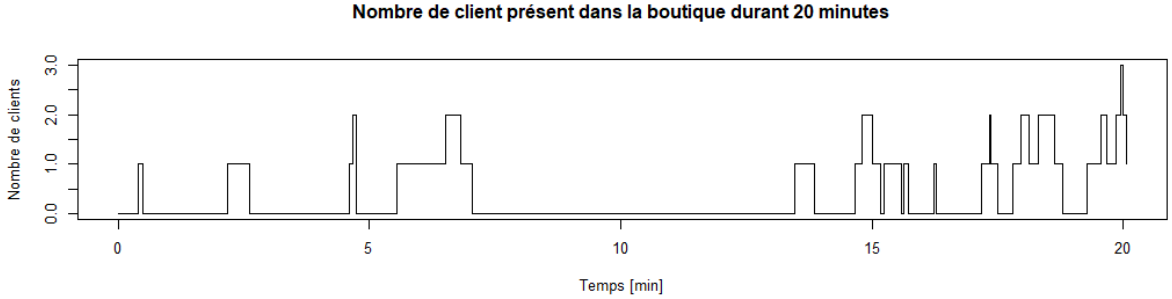
```
  index_dep <- 1
  index_temp <- 1

  temps[1] <- 0
  nb_client[1] <- 0

  while ((index_dep <= length(departs)) && (temps[index_temp]<t)) {

    if(arrivees[index_arr] < departs[index_dep]){
      temps[index_temp+1] = arrivees[index_arr]
      nb_client[index_temp+1] <- nb_client[index_temp] +1
      index_arr <- index_arr + 1
    }else{
      temps[index_temp+1] = departs[index_dep]
      nb_client[index_temp+1] <- nb_client[index_temp] - 1
      index_dep <- index_dep + 1
    }
    index_temp <- index_temp + 1
  }

  my_list <- list("x" = temps, "y" = nb_client)

  return(my_list)
}
```

On crée ensuite une fonction pour afficher les différents graphiques :

```
Dessine_Q1 <- function (t){
  #par(mfrow=c(2,2)) #Pour afficher sur la meme image
  layout(matrix(c(1,2,3,3), 2, 2, byrow = TRUE))

  arrivees <- fct_arrive(t,lambda,"duree")
  services <- fct_service(t,mu,"inter")

  depart = fct_depart(t,arrivees,services,"duree")

  nbclient = N(t,arrivees,depart)

  list_arr = 1:length(arrivees)
  plot(arrivees, list_arr
       ,type ="s"
       ,xlab ="Temps [min]"
       ,ylab ="Nombre d'arrivée"
       ,main = paste("Arrivée des clients en", t, "minutes\n"))

  list_dep = 1:length(depart)
  plot(depart, list_dep
       ,type ="s"
       ,xlab ="Temps [min]"
       ,ylab ="Nombre de depart"
       ,main = paste("Départ des clients en", t, "minutes\n"))

  plot(nbclient$x, nbclient$y
       ,type ="s"
       ,xlab ="Temps [min]"
       ,ylab ="Nombre de clients"
       ,main = paste("Nombre de client présent dans la boutique durant", t, "minutes\n"))
}
```

Le résultat est le suivant :





Question 2 :

Calcul théorique :

La formule du cours donne :

$$Q = \frac{\rho}{(1-\rho)}$$

avec : $\rho = \frac{\lambda}{\mu}$

On a donc :

$$\rho = \frac{2}{3}$$

$$Q = \frac{\frac{2}{3}}{1-\frac{2}{3}} = \frac{\frac{2}{3}}{\frac{1}{3}} = 2$$

Calcul pratique :

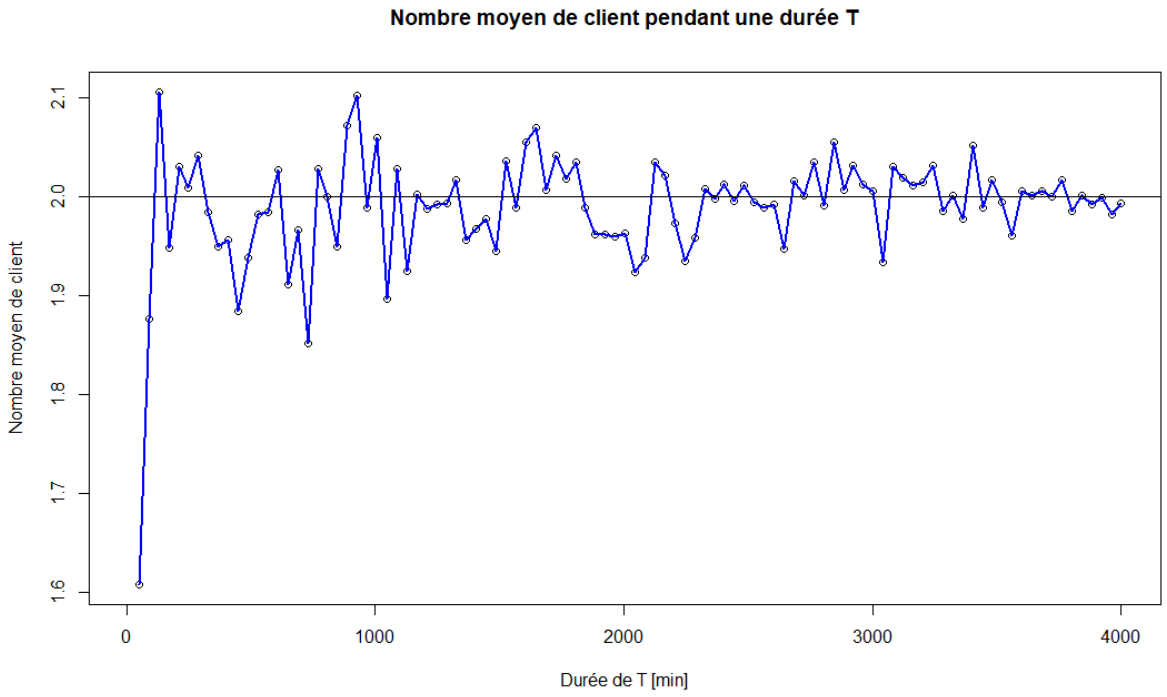
On crée une fonction qui fait varier T et qui à chaque fois calcule la moyenne du nombre moyen de clients sur plusieurs simulations :

(Cette fonction retourne aussi le paramètre opérationnel : $\bar{}q_T = \frac{1}{T}\int_0^T q(t)dt$)

```
nb_moy_client <- function(t,lambda_,mu_){

  arrivees <- fct_arrive(t,lambda_,"duree")
  services <- fct_service(t,mu_,"inter")
  depart = fct_depart(t,arrivees,services,"duree")

  nbclient = N(t,arrivees,depart)

  moy <- 0
  for (i in 2:(length(nbclient$x))) {
    moy <- moy + ((nbclient$x[i]-nbclient$x[i-1])*nbclient$y[i-1])
  }
  moy = moy / t
  return(moy)
}
```

```
#affiche le nombre moyen de client avec T qui varie
#retourne le paramètre operationnel, c-a-d l'integrale de 0 à T du nombre moyen de client divisé par T
q2 <- function(t_min,t_max,nb_rep,afficher,lambda_,mu_){
  nb_point = 125
  delta = (t_max - t_min)/nb_point
  temps <- seq(t_min,t_max,delta)
  nb_client <- c()

  param_ope <- 0

  for (i in (1:nb_point+1)) {
    temp<- 0
    for (j in 1:nb_rep) {
      temp<- temp + nb_moy_client(temps[i],lambda_,mu_)
    }
    nb_client[i] <- temp / nb_rep
    param_ope <- param_ope + (delta*nb_client[i])
  }

  param_ope <- param_ope / t_max

  if(afficher){
    layout(matrix(c(1), 1,1, byrow = TRUE))
    plot(temps, nb_client
         ,xlab ="Durée de T [min]"
         ,ylab ="Nombre moyen de client"
         ,main = paste("Nombre moyen de client pendant une durée T\n"))
    lines(temps, nb_client, col = 'blue', lwd = 2)
    abline(h=2)
  }

  return(param_ope)
}
```

Le résultat de q2(1,4000,15,TRUE,2,3) est :



On remarque plusieurs choses :

- Plus la durée T grandie, plus le résultat est précis

- Le résultat théorique converge bien vers 2, qui est la valeur théorique attendue.

Question 3 :

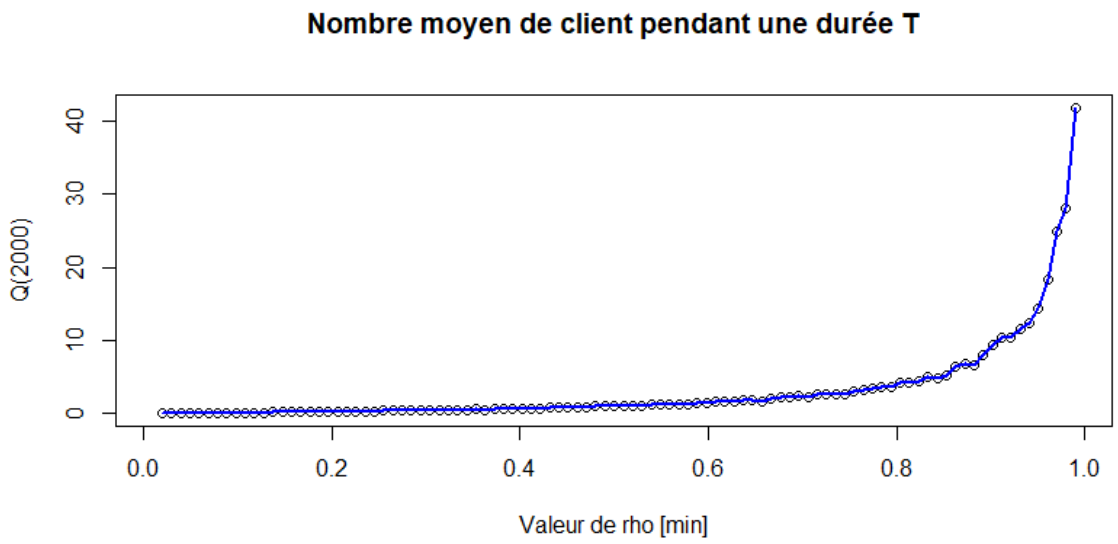
On choisit T = 2000. En effet, d'après les résultats de la question précédente, on constate que T = 2000 suffit pour pouvoir considérer que le régime stationnaire est établi.

A)

On crée une fonction qui fait varier rho de 0 à 1 et qui calcule sur plusieurs simulations le nombre moyen de clients :

```
evol_para_stocha <- function(nb_rep){
  nb_point = 100
  mu_ = 1
  delta = (0.99 - 0.01)/nb_point
  lambda_vect <- seq(0.01,0.99,delta)
  Q_vect <- c()

  for (i in (1:nb_point+1)) {
    lambda_ <- lambda_vect[i]
    temp<- 0
    for (j in 1:nb_rep) {
      temp<- temp + nb_moy_client(2000,lambda_,mu_)
    }
    Q_vect[i] <- temp / nb_rep
  }

  layout(matrix(c(1), 1,1, byrow = TRUE))
  plot(lambda_vect, Q_vect
       ,xlab ="Valeur de rho [min]"
       ,ylab ="Q(2000)"
       ,main = paste("Nombre moyen de client pendant une durée T\n"))
  lines(lambda_vect, Q_vect, col = 'blue', lwd = 2)
}
```

Le résultat de evol_para_stocha(10) est :

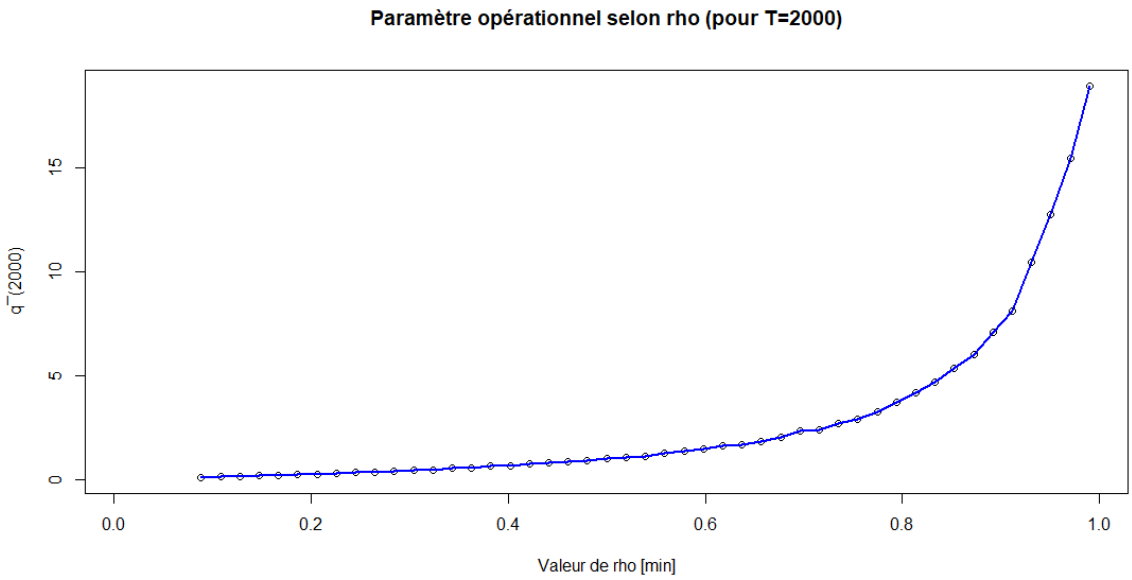


B)

On crée une fonction qui fait varier rho de 0 à 1 et qui calcule sur plusieurs simulations le paramètre opérationnel :

```
evol_para_ope <- function(nb_rep){
  nb_point = 50
  mu_ = 1
  delta = (0.99 - 0.01)/nb_point
  lambda_vect <- seq(0.01,0.99,delta)
  Q_vect <- c()

  for (i in (1:nb_point+1)) {
    lambda_ <- lambda_vect[i]
    Q_vect[i] <- q2(1,2000,nb_rep,FALSE,lambda_,mu_)
    cat("Avancement :",(i/(nb_point+1))*100,"%\n" )
  }

  layout(matrix(c(1), 1,1, byrow = TRUE))
  plot(lambda_vect, Q_vect
       ,xlab ="Valeur de rho [min]"
       ,ylab ="q¯(2000)"
       ,main = paste("Paramètre opérationnel selon rho (pour T=2000)\n"))
  lines(lambda_vect, Q_vect, col = 'blue', lwd = 2)
}
```

Le résultat de evol_para_ope (10) est :



On remarque 2 choses :

- Les deux courbes ont la même forme, mais le paramètre stochastique croit plus vite que le paramètre opérationnel.
- Les courbes correspondent bien au résultat théorique, quand rho tend vers 1 le nombre de clients dans la file tend vers l'infini.

Question 4 :

Dans cette question, on reprend le code des 3 questions précédentes et on le modifie afin de correspondre à un modèle de file M/M/2.

Il suffit de modifier les fonctions générant les tableaux contenant les instants d'arrivée et de départ :

- fct_arrivee ne change pas.
- fct_depart change radicalement, en effet nous nous sommes rendus compte que l'organisation de notre fonction d'origine n'était pas adapté pour rajouter des serveurs.

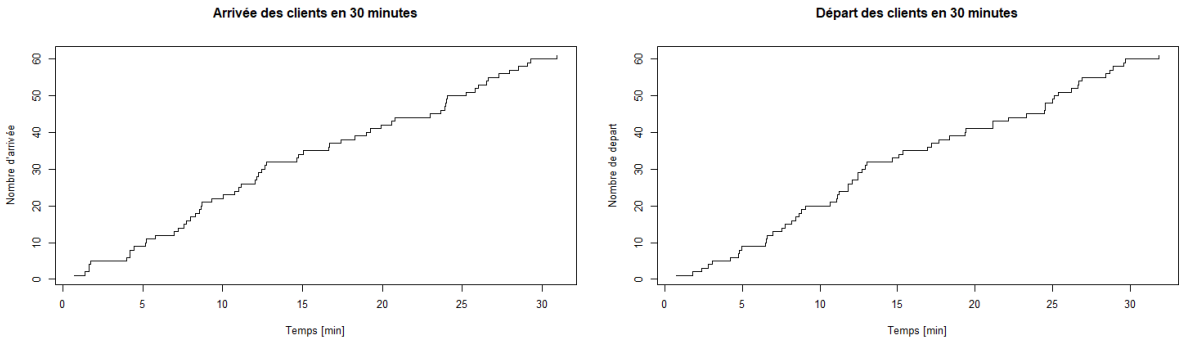
  On a donc modifié la fonction afin de simplifier le code.

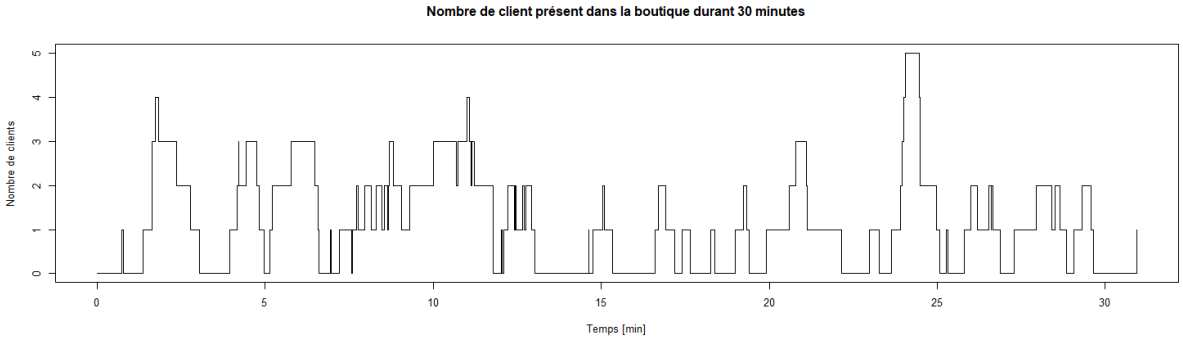
```
fct_depart <- function(temps,arrivees, mu_){

  duree <- 0
  index <- 1
  depart_duree <- c()
  serveur_1 <- 0
  serveur_2 <- 0

  for (i in 1:(length(arrivees))) {

    if(arrivees[i]>serveur_1){#si le client arrive après que le serveur 1 ai fini de servir le client precedent
      depart_duree[index]= arrivees[i] + rexp(1,mu_)
      serveur_1 = depart_duree[index]
    }else if(arrivees[i]>serveur_2){#si le client arrive après que le serveur 2 ai fini de servir le client prece
dent
      depart_duree[index]= arrivees[i] + rexp(1,mu_)
      serveur_2 = depart_duree[index]
    }else if(serveur_1 < serveur_2){#sinon on attend qu'un des deux serveur ai fini et on passe avec le premier l
ibre
      depart_duree[index]= serveur_1 + rexp(1,mu_)
      serveur_1 = depart_duree[index]
    }else{
      depart_duree[index]= serveur_2 + rexp(1,mu_)
      serveur_2 = depart_duree[index]
    }
    index <- index + 1
  }
  depart_duree <- sort(depart_duree)
  return(depart_duree)
}
```

- On ne modifie le reste des fonctions que pour s'adapter aux changements de paramètre que requiert la nouvelle fonction fct_depart.

Voici les résultats attendus des questions 1 à 3 :

Question 1 :





Question 2:

Calcul théorique :

La formule du cours donne :

$$Q = \frac{\rho^{C+1}}{(C-1)!(C-\rho)^2}\pi_0 + \rho$$

avec :

$$\rho = \frac{\lambda}{\mu}$$

et :

$$\pi_0 = \frac{1}{\sum_{n=0}^{C-1} \frac{\rho^n}{n!} + \frac{\rho^C}{(C-1)!(C-\rho)}}$$

On a donc :

$$\rho = \frac{1}{1} = 1$$

$$\pi_0 = \frac{1}{\frac{1^0}{0!} + \frac{1^1}{1!} + \frac{1^2}{(2-1)!(2-1)}}$$

$$\pi_0 = \frac{1}{1+1+1} = \frac{1}{3}$$

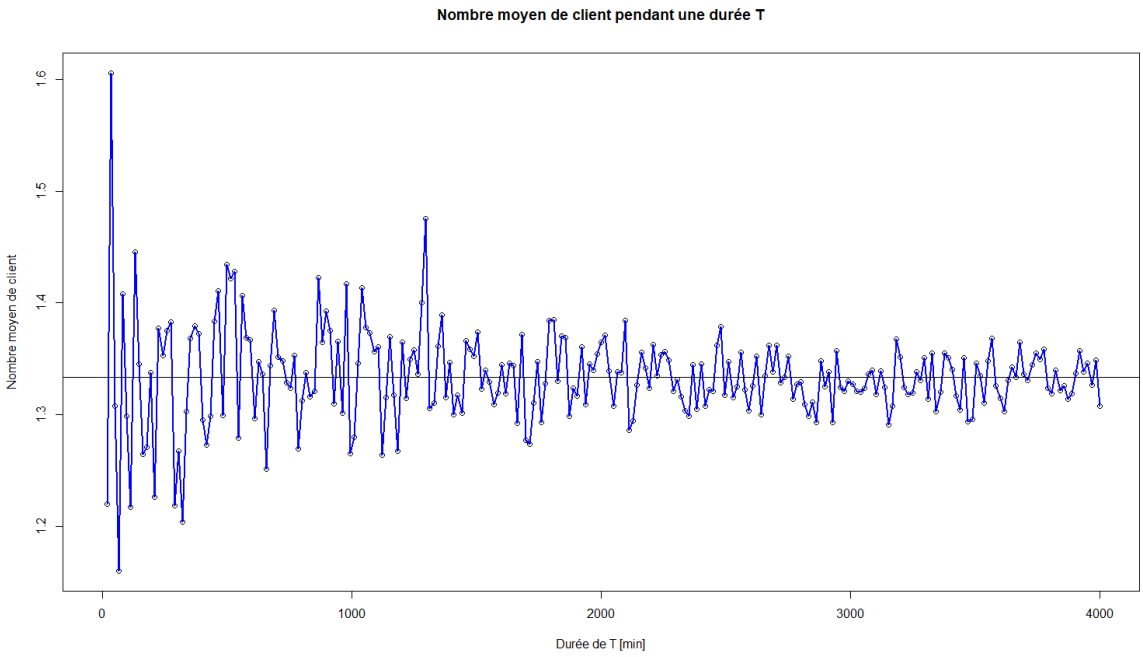
$$Q = \frac{1^{2+1}}{(2-1)!(2-1)^2}\frac{1}{3} + 1$$

$$Q = \frac{1}{1*1} * \frac{1}{3} + 1 = 1 + \frac{1}{3} = \frac{4}{3} \approx 1,33$$

Calcul pratique :

On réutilise la fonction q2 en traçant une ligne à y=4/3 au lieu de y=2.

On a donc :



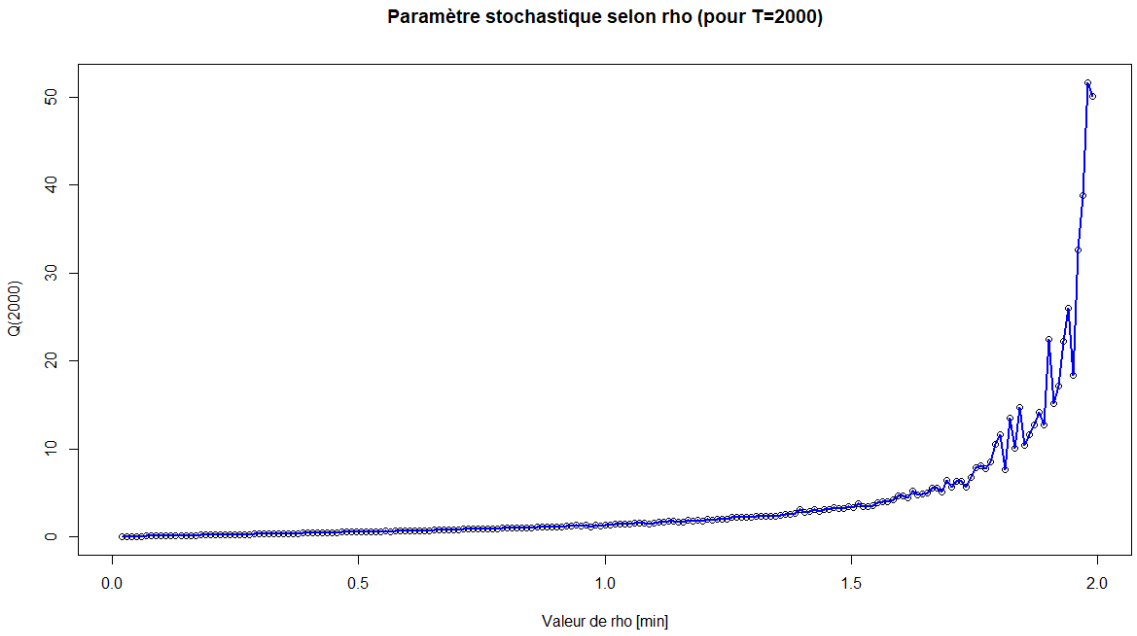
Comme avec un seul serveur, on remarque que plus le temps est long, plus le résultat devient précis. De plus, le résultat obtenu correspond au résultat théorique.

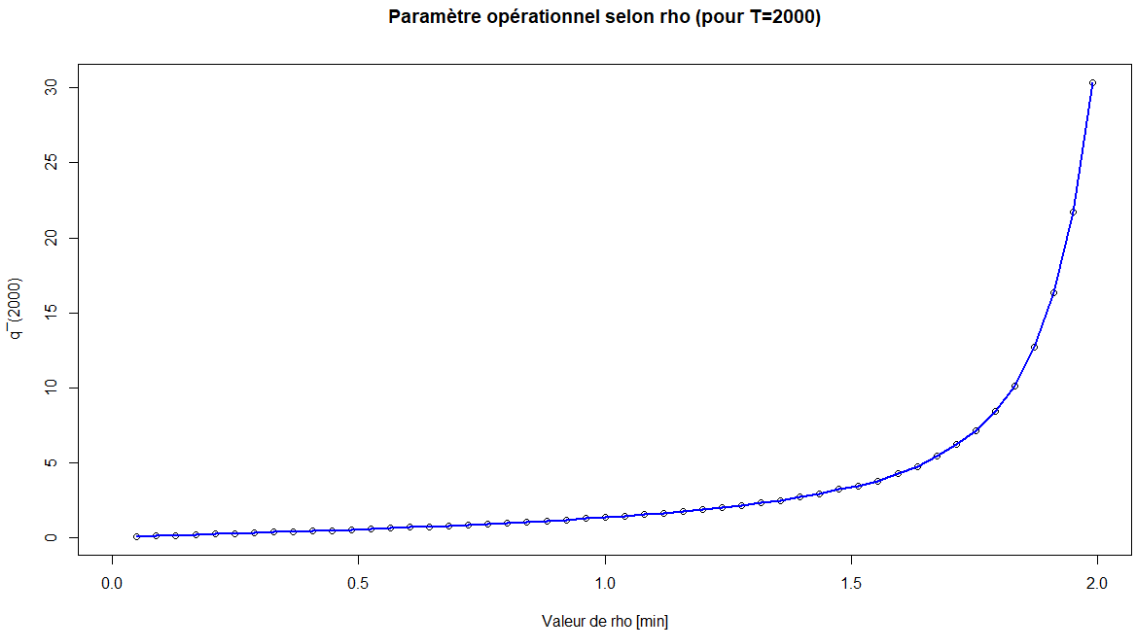
Question 3 :

On choisit encore T = 2000, pour les mêmes raisons que précédemment.

On réutilise les fonctions préalablement définies, mais en faisant varier rho entre 0 et C (donc entre 0 et 2).

Le résultat est le suivant :





On remarque 3 choses :

- Les deux courbes ont la même forme, mais le paramètre stochastique croit plus vite que le paramètre opérationnel.
- La courbe du paramètre opérationnel est plus lisse que l'autre, cela est dû au fait que le calcul du paramètre opérationnel se base sur une intégrale (moyenne).
- Les courbes correspondent bien au résultat théorique, quand rho tend vers 2 le nombre de clients dans la file tend vers l'infini.

Conclusion :

Ce DM nous a permis de revoir les chapitres important du programme et de découvrir le langage R.