Stochastisch modelleren van biologische processen project

Wietse Vaes (1848522) & Amber Huybrechts (1953107)

Academiejaar: 2020-2021

Oefening 1: Markov ketens

a) eigenwaarden: $\lambda_1 = 1,\ \lambda_2 = 0.026 + 0.075i,\ \lambda_3 = 0.026 + 0.075i,\ \lambda_4 = 0.079,\ \lambda_5 = 1.14 \cdot 10^{-7} - 1.97 \cdot 10^{-7},\ \lambda_6 = 1.14 \cdot 10^{-7} - 1.97 \cdot 10^{-7},\ \lambda_7 = -2.28 \cdot 10^{-7}$

eigenvectoren: $v_1 = \begin{pmatrix} 0.378 \\ 0.378 \\ 0.378 \\ 0.378 \\ 0.378 \\ 0.378 \\ 0.378 \end{pmatrix}$, $v_2 = \begin{pmatrix} 0.107 - 0.1i \\ -0.608 + 0i \\ 0.105 + 0.232i \\ 0.227 - 0.059i \\ 0.105 + 0.232i \\ -0.608 + 0i \\ 0.227 - 0.059i \end{pmatrix}$, $v_3 = \begin{pmatrix} 0.107 + 0.1i \\ -0.608 + 0i \\ 0.105 - 0.232i \\ 0.227 + 0.059i \\ 0.105 - 0.232i \\ -0.608 + 0i \\ 0.227 + 0.059i \end{pmatrix}$, $v_4 = \begin{pmatrix} 0.414 \\ -0.529 \\ 0.270 \\ -0.248 \\ 0.270 \\ -0.529 \\ -0.248 \end{pmatrix}$, $v_5 = \begin{pmatrix} -0.31 + 3 \cdot 10^{-7}i \\ 0.66 - 1.1 \cdot 10^{-7}i \\ -0.094 - 5.4 \cdot 10^{-7}i \\ -0.094 + 2.1 \cdot 10^{-7}i \\ 0.023 - 8.4 \cdot 10^{-7}i \\ 0.66 + 0i \\ -0.094 + 2.1 \cdot 10^{-7}i \end{pmatrix}$, $v_6 = \begin{pmatrix} -0.31 - 3 \cdot 10^{-7}i \\ 0.66 + 1.1 \cdot 10^{-7}i \\ -0.094 + 5.4 \cdot 10^{-7}i \\ -0.094 - 2.1 \cdot 10^{-7}i \\ 0.023 + 8.4 \cdot 10^{-7}i \\ 0.66 + 0i \\ -0.094 - 2.1 \cdot 10^{-7}i \end{pmatrix}$, $v_7 = \begin{pmatrix} -0.31 \\ 0.66 \\ -0.094 \\ -0.094 \\ 0.02 \\ 0.66 \\ -0.094 \end{pmatrix}$

Al deze eigenwaarden hebben een absolute waarde kleiner dan of gelijk aan 1 met onderling verschillende eigenvectoren dus T is diagonaliseerbaar en bijgevolg bestaat er een stationaire verdeling. We berekenen deze op dezelfde manier als oefening 4 in de leidraad.

b) We weten dus dat de stationaire verdeling bestaat en gebruiken de werkwijze van oefening 4.

```
A <- t(T)-diag(7)
A <- rbind(A,c(1,1,1,1,1,1,1))
b <- c(0,0,0,0,0,0,0,1)
b <- t(A)%*%b #A is niet vierkant -> we lossen systeem A^TAx=A^Tb op.
A <- t(A)%*%A
pexact <- solve(A, b)
```

De stationaire verdeling ziet er dan als volgt uit:

$$\begin{pmatrix} 0.299 \\ 0.185 \\ 0.135 \\ 0.293 \\ 0.069 \\ 0.014 \\ 0.0051 \end{pmatrix}$$

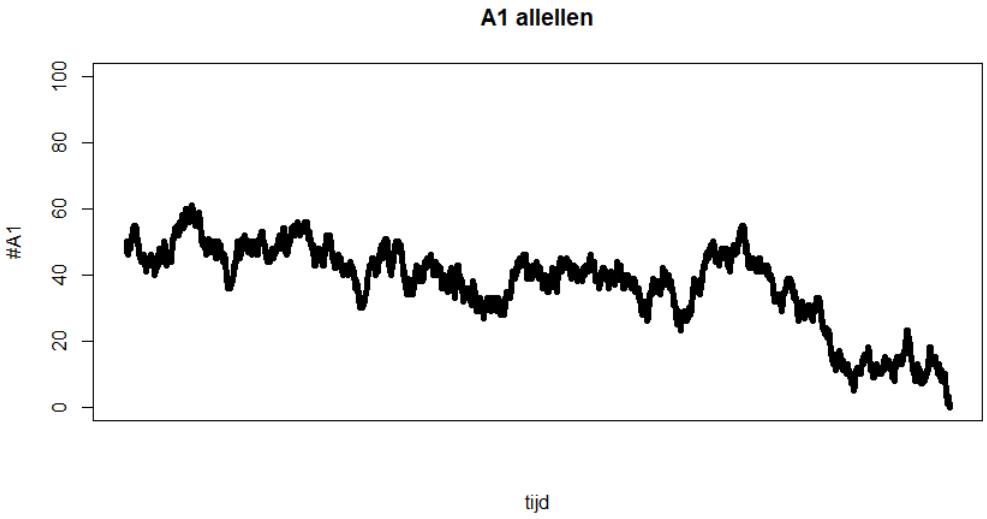
We interpreteren dit als volgt: ongeveer 29.9% van de keten bevat de sequentie A basen, 18.5% sequentie C basen, 13.5% sequentie G basen, 29.3% sequentie T basen, 6.9 % sequentie AG basen, 1.4 % sequentie AGC basen en 0.51% sequentie AGCT basen.

c) Nu genereren we een keten met de functie 'padgenereren' met de gegeven overgangsmatrix, begin base A en lengte 100: "AAAGATCCTGACCATGAAGTGTCAAGAGCAGCTTATAAAGGTGGAAGGACGC-TAACACACAATCCAAACTGCAAGAGCCATAACTCCAAACTGGTTCTATGAATATTCATCTCA"

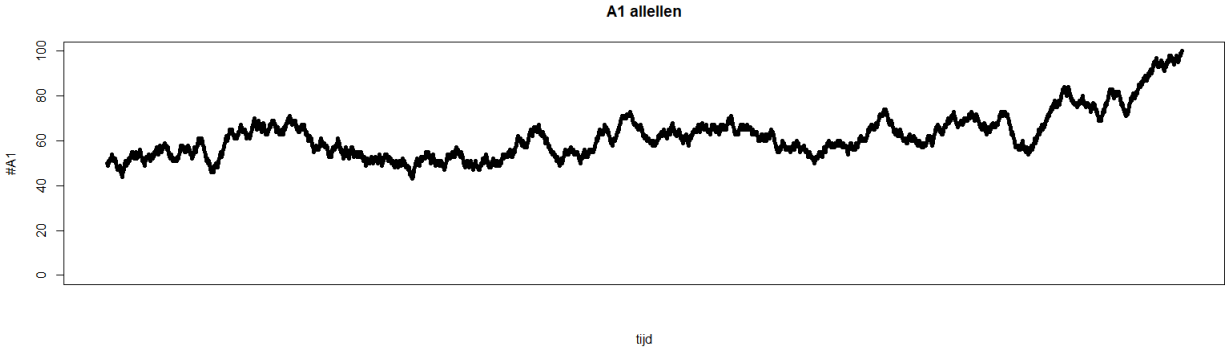
Oefening 2: Geboorte- en sterfteprocessen

Bij deze oefening beschouwen we de death rate als de rate waarbij een A1 allel sterft en vervangen wordt door een A2 allel. De geboorte rate zien we als de rate waarbij een A1 allel een A2 allel zal vervangen. De gebeurtenissen A1 vervangt A1 of A2 vervangt A2 worden niet als gebeurtenissen beschouwt.

a) We zien dat er evenwicht situaties voorkomen (op verschillende tijdstippen). Oftewel zal het A1 allel uitsterven, oftewel zal het A1 allel overheersen. Onderstaande grafieken tonen twee mogelijke verlopen van de populatie A1 allellen:

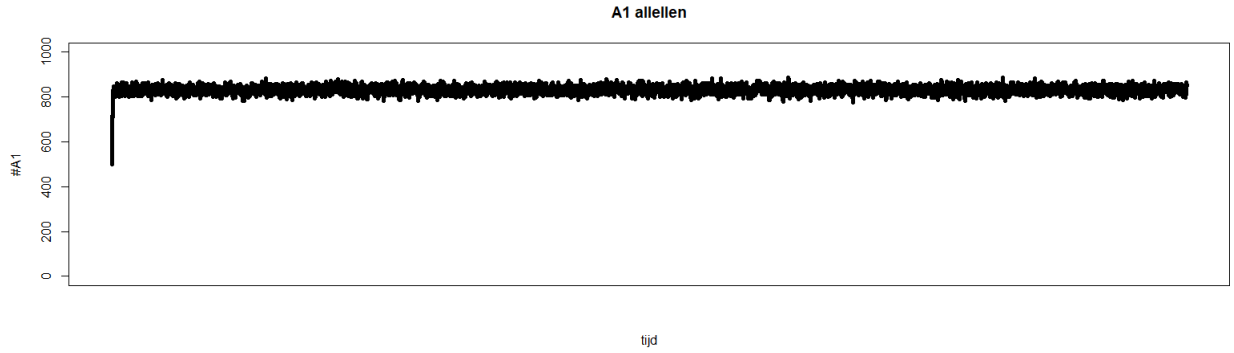


Figuur 1: mogelijke situatie: A1 allel sterft uit



Figuur 2: mogelijke situatie: A1 allel overheerst

b) Hier zien we dat er geen echt evenwicht voorkomt. We zien echter wel dat de populatie zal fluctueren. Indien we de gemiddelde waarde berekenden van de grootte van de populatie A1, vonden we dat het rond 832 fluctueerde. Ook zagen we dat, ookal begon de populatie met niets of overheerste het, het ging fluctueren rond deze waarde. Dit resultaat is vrij logisch aangezien de populaties nu van elkaar afhingen. Als een populatie groot werd ging de kans da het ging muteren ook stijgen.



Figuur 3: mutatie is mogelijk tussen A1 en A2 allellen

Oefening 3: Wachtrij theorie

1. a) Eerst hebben we theoretisch de gemiddelde bedieningstijd berekend voor de verschillende bedienden.
   - $E(S_1) = \frac{n}{\alpha} = 4$
   - $E(S_2) = \frac{n}{\alpha} = 3.5$
   - $E(S_3) = \frac{1}{\lambda} = 2.5$
   - $E(S_4) = \frac{a+b}{2} = 4.5$

   Daarna berekenden we dit met R en kwamen we uit op volgende resultaten:
   - $E(S_1) = 4.011301$
   - $E(S_2) = 3.496487$
   - $E(S_3) = 2.497434$
   - $E(S_4) = 4.508263$

   b) De gemiddelde tijd die een telefoongesprek in het systeem doorbrengt (vector 'time_spent') is gelijk aan 8.26 minuten.

   c) Het gemiddelde aantal telefoongesprekken in de wachtrij (vector 'w') is 4.67.

   d) De kans dat een binnenkomend gesprek niet moet wachten is 0.3084 ($\approx 30.84\%$).

   e) De verwachtte wachttijd voor een willekeurig gesprek (vector 'waiting_time') is 4.78 minuten.

   f) De verwachte maximum wachttijd gedurende een dag (vector 'wtmax') is 19.52 minuten.

   g) De verwachte maximum lengte van een wachtrij gedurende een dag (vector 'wmax') is 20.11.

   h) De verwachte tijd waarin een bediende niets doet gedurende een dag voor elke bediende apart (vector 'free_time...'):
   - bediende 1 heeft gemiddeld 69.64 minuten van de dag niets te doen.
   - bediende 2 heeft gemiddeld 71.47 minuten van de dag niets te doen.
   - bediende 3 heeft gemiddeld 78.67 minuten van de dag niets te doen.
   - bediende 4 heeft gemiddeld 54.627 minuten van de dag niets te doen.

   i) De kans dat geen enkel gesprek in de wachtrij staat of geholpen wordt aan het einde van de dag is 0.01 ($\approx 1\%$). We interpreten dit als dat er op het einde van de dag niemand in het systeem zit. Indien we meer dagen beschouwen ging dit naar 0.027.

   j) Het verwachte aantal gesprekken dat dagelijks per bediende uitgevoerd wordt (vector 'served...'):
   - bediende 1 voert gemiddeld 104.03 gesprekken uit per dag.
   - bediende 2 voert gemiddeld 119.05 gesprekken uit per dag.
   - bediende 3 voert gemiddeld 160.55 gesprekken uit per dag.
   - bediende 4 voert gemiddeld 95.5 gesprekken uit per dag.

   k) De kans dat ten minste 2 bediendes vrij zijn op het moment dat een telefoongesprek aankomt is 0.1869 ($\approx 18.69\%$).

2. a) b) De gemiddelde tijd die een telefoongesprek in het systeem doorbrengt is 6.05 minuten.

   c) Het gemiddelde aantal telefoongesprekken dat in de wachtrij staat is 2.69.

d) De kans dat een binnenkomend gesprek niet moet wachten is 0.3620 ($\approx 36.20\%$).

e) De verwachte wachttijd voor een willekeurig gesprek 2.69 minuten.

f) De verwachte maximum wachttijd gedurende een dag is 9.91 minuten.

b) Het verwachte percentage gesprekken dat de wachtrij vroegtijdig verlaat is 0.0332 ($\approx 3.32\%$).

De waardes die we uitkomen zien er mogelijk uit. Ook de verschillen tussen oefening 1 en 2 zien er logisch uit. Men zit minder lang in het systeem, er zijn er minder in de wachtrij, men kan vaker meteen geholpen worden, de wachttijd is lager en de verwachte maximum wachttijd is lager en onder 10 minuten.

Appendix

Oefening 1:

```
    T <- matrix(c(0.32, 0.18, 0, 0.27, 0.23, 0, 0,
            0.37, 0.23, 0.05, 0.35, 0, 0, 0,
            0.30, 0.21, 0.25, 0.24, 0, 0, 0,
            0.23, 0.19, 0.25, 0.33, 0, 0, 0,
            0.30, 0, 0.25, 0.24, 0, 0.21, 0,
            0.37, 0.23, 0.05, 0, 0, 0, 0.35,
            0.23, 0.19, 0.25, 0.33, 0, 0, 0), byrow=T, nrow = 7, ncol = 7)

## 1.a
ewv <- eigen(T) #we berekenen de eigenwaarde van T via de functie eigen()
ew <- ewv$values
ev <- ewv$vectors
abs(ew)

## 1.b  (alle abs(ew)<=1 en ev =/ ev-> er is een stationaire verdeling)
A <- t(T)-diag(7)  #We maken de matrix T in een stelsel zodat we hieruit de stationaire
                    verdeling vinden.
A <- rbind(A,c(1,1,1,1,1,1,1))
b <- c(0,0,0,0,0,0,0,1)
b <- t(A)%*%b #A is niet vierkant -> we lossen systeem A^TAx=A^Tb op.
A <- t(A)%*%A

pexact <- solve(A, b)

## 1.c
padgenereren <- function(Tr, E_n, n){
  bases <- c('A','C','G','T','AG','AGC','AGCT') #de basen
  pad <- c() #Begin van pad (zonder de start base erin)
  for (i in 1:n) {
    overgangskans <- Tr[E_n,] #De kansen van waar de huidige base naar toe kan
    E_n <- sample(x = 1:nrow(Tr), prob = overgangskans, size = 1) #Randomsample ervan
    pad <- c(pad, bases[E_n]) #Base aan pad toegevoegd
  }
  return( pad )
}

k <- 100
genketen <- padgenereren(T,1,k) #Creeer een pad met 100 bases
p <- c(sum(genketen=="A")/k,sum(genketen=="C")/k,
       sum(genketen=="G")/k,sum(genketen=="T")/k,
       sum(genketen=="AG")/k,sum(genketen=="AGC")/k,
       sum(genketen=="AGCT")/k) #controle kansen

keten <- "" #Hier voegen we elke string apart tot 1 string samen
for (i in 1:length(genketen)){
  keten <- paste(keten,genketen[i], sep = "")
}
```

Oefening 2:

```
#2.a
#Standaard birthfunctie die de rate terug geeft
birth <- function(x, M, lambda2, alpha1, alpha2) {
  p <- (x/M)*(1-alpha1)+(1-x/M)*alpha2
  b <- lambda2*p*(M-x)
```

```
  return(b)
}


#Standaard deathfunctie die de rate terug geeft
death <- function(x, M, lambda1, alpha1, alpha2) {
  q <- 1-(x/M*(1-alpha1)+(1-x/M)*alpha2)
  d <- lambda1*q*x
  return(d)
}

Genallel <- function(M, lambda1, lambda2, alpha1, alpha2) {
  t <- 0
  T <- 999
  X <- c(ceiling(M/2)) #We mogen hier kiezen wat we invullen, maar kiezen de helft, 0 of alles vaak.
  k <- 1
  while ( t < T ){
    t <- t + rexp(1,rate=(death(X[k],M,lambda1,alpha1,alpha2)+birth(X[k],M,lambda2,alpha1,alpha2)))
    #tijd tot nieuwe gebeurtenis
    if (is.na(t)){
      return(X)   #Als t not a number is, komt er geen nieuwe gebeurtenis. Dus dit einde is
                  hoe het altijd zal blijven.
    }
    pbd <- birth(X[k],M,lambda2,alpha1,alpha2)/(death(X[k],M,lambda1,alpha1,alpha2)+
           birth(X[k],M,lambda2,alpha1,alpha2))
    #De kans dat de gebeurtenis een geboorte is
    b <- (runif(1)<=pbd) #Dit is de kans uitgeoefend op een random getal
    incdec <- 2*b-1 #Als het een geboorte was, increaset het met 1. Als het een sterfte was,
                    decreaset het met 1
    X[k+1] <- X[k] + incdec
    k <- k +1
  }
  return(X)
}

set.seed(12345)
allellen1 <- Genallel(100, 1, 1, 0, 0) #Genereren de evolutie van de populatie met gegeven waarden
x1 <- 1:length(allellen1)
plot(x1,allellen1, pch = 20, main ="A1 allellen",xaxt='n',
     xlab ="tijd", ylab ="#A1", ylim = c(0, 100)) #Plotten populatie


set.seed(12345)
allellen2 <- Genallel(1000,3,5,0.2,0.5) #Genereren de evolutie van de populatie met gegeven waarden
mean(allellen2) #Berekenen waarrond het zal schommelen

deltax = ceiling(length(allellen2)/200000) #De vector allellen2 is te lang om te plotten
x <- 1:length(allellen2) #We korten het dus in
x <- x[x%%deltax==0]

allellen2 <- allellen2[x]
x2 <- 1:length(allellen2)
plot(x2,allellen2, pch = 20, main = "A1 allellen",xaxt='n',
     xlab = "tijd", ylab = "#A1", ylim = c(0, 1000))
```

Oefening 3:

```
#a) gemiddelde bedieningstijd telefoonbediende
```

```
X1 <- rgamma(100000,shape=1/3,rate=1/12) #Gegenereerde populatie volgens gamma verdeling
mean(X1) #E(S1)=n/alfa=4 (theoretische waarde)
X2 <- rgamma(100000,shape=1/2,rate=1/7) #Gegenereerde populatie volgens gamma verdeling
mean(X2) #E(S2)=n/alfa=3.5 (theoretische waarde)
X3 <- rexp(100000,rate=0.4) #Gegenereerde populatie volgens exponentiele verdeling
mean(X3) #E(S3)=1/lambda=2.5 (theoretische waarde)
X4 <- runif(100000,min=2,max=7) #Gegenereerde populatie volgens uniforme verdeling
mean(X4) #E(S4)=(a+b)/2=4.5 (theoretische waarde)

####b-k)
set.seed(1234567)
simulate_wachtrij1 <- function(lambda, T) {
  Arrival <- c() #Vector van aankomsttijden van klant
  Server <- c() #Vector van server toegewezen aan klant
  Starttime <- c() #Starttijden met bediende van klant
  Finishtime <- c()#Eindtijden met bedienen van klant
  end_time_server <- c(0,0,0,0)  #De laatste finishtime van elke server
  Free_servers <- c(1,2,3,4) #Vector die gebruikt wordt om vrije servers bij te houden
  Hoeveel_vservers <- c() #Hoeveelheid vrije servers
  k <- 1
  tt <- rexp(1,rate=lambda) # Tijd eerste aankomst
  while( tt<T ) {
    Arrival[k] <- tt
    Free_server <- Free_servers[ end_time_server<tt ] #We kijken welke servers vrij zijn
    if ( length(Free_server) == 0 ) { #Indien er geen servers vrij zijn:
      stophelp <- min(end_time_server) #Kijken welke eerste klaar zal zijn
      if ( stophelp==end_time_server[1] ) { #Alloceren deze klant aan de bediende
                                            en genereren bedieningstijd
        Server[k] <- 1
        S <- rgamma(1, shape = 1/3, rate = 1/12)
      } else if ( stophelp==end_time_server[2] ) {
        Server[k] <- 2
        S <- rgamma(1,shape=1/2,rate=1/7)
      } else if ( stophelp == end_time_server[3] ) {
        Server[k] <- 3
        S <- rexp(1, rate=0.4)
      } else {
        Server[k] <- 4
        S <- runif(1, min=2, max=7)
      }
      Starttime[k] <- stophelp #Starttime van klant is tijd wanneer eerste server vrij is
      Finishtime[k] <- Starttime[k]+S #Endtime is Starttime + hoelang het duurt(S)
      end_time_server[stophelp==end_time_server] = Finishtime[k] #We houden bij dat nu de server
                                                                 een nieuwe laatste finish heeft
    } else { #Er is meer dan 1 server vrij
      if ( length(Free_server)==1 ) { #Er is 1 server vrij
        fs <- Free_server[1] #fs is de gekozen server voor de klant
      } else {
        fs <- sample(Free_server,1) #Hier kiezen we een random server als er meer vrije servers zijn
      }
      Starttime[k] <- tt
      if( fs==1 ) { #Alloceren deze klant aan de bediende en generen bedieningstijd
        Server[k] <- 1
        S <- rgamma(1,shape=1/3,rate=1/12)
      } else if ( fs==2 ) {
        Server[k] <- 2
        S <- rgamma(1,shape=1/2,rate=1/7)
      } else if ( fs==3 ) {
        Server[k] <- 3
        S <- rexp(1,rate=0.4)
```

```
      } else {
        Server[k] <- 4
        S <- runif(1,min=2,max=7)
      }
      Finishtime[k] <- tt+S
      end_time_server[fs] <- Finishtime[k]
    }
    Hoeveel_vservers <- c(Hoeveel_vservers, length(Free_server)) #Houden bij hoeveel vrije servers
                                                                 er waren voor allocatie
    k <- k+1
    tt <- tt+rexp(1, rate=lambda) #Generen aankomst tijd van volgende klant (we doen het hier opdat
                                  er geen klant na 17u komt)
  }

  return (list("Arrival"=Arrival,"Starttime"=Starttime,"Finishtime"=Finishtime,"Server"=Server,
  "Vrijeservers" = Hoeveel_vservers))
}

numb_wait <- function(r) { #Berekent het aantal wachtende klanten
  n <- c()
  k <- length(r$Arrival)
  for(j in 2:(k-1)){
    wait <- 1*(r$Starttime[1:(j-1)]>r$Arrival[j])
    n[j-1] <- mean(wait)*(j-1)
  }
  return(n)
}

free_times <- function(r,k){ #Berekent hoelang bediende wachten tussen klanten (per server)
  free_time <- c()
  Finishtime <- r$Finishtime[r$Server==k]
  Starttime <- r$Starttime[r$Server==k]
  for (i in 1:length(Starttime)-1){
    free_time <- c(free_time,Starttime[i+1]-Finishtime[i])
  }
  return(free_time)
}

#Hier maken we alle vectoren die dingen bijhouden
w <- c() #Houdt bij hoeveel mensen wachten tussen gebeurtenissen
wmax <- c() #Houdt de max hoeveelheid wachtende mensen op een tijdstip per dag bij
wtmax <- c() #Houdt de max wachttijd van klant bij per dag
waiting_time <- c() #Houdt de wachttijd per klant bij
dayend <- c() #Houdt de tijd van laatste vertrek van klant bij per dag

Arrival <- c() #Houdt alle aankomsten bij van klanten
Starttime <- c() #Houdt alle starttijden bij van klanten
Finishtime <- c() #Houdt alle eindtijden bij van klanten
Server <- c() #Houdt alle toegewezen servers van klanten bij
Serversvrij <- c() #Houdt de hoeveelheid vrije servers op een tijdstip bij

#Houdt vrije tijd tussen klanten in totaal bij van server ...
free_time1 <- c()
free_time2 <- c()
free_time3 <- c()
free_time4 <- c()

#Houdt hoeveelheid klanten per dag bij van server ...
served1 <- c()
served2 <- c()
```

```
served3 <- c()
served4 <- c()

for (i in 1:100) { #100 dagen
  r <- simulate_wachtrij1(1, 8*60) #We simuleren hier 1 dag van 8*60 minuten
  #Nu voegen we elke dag in een vector samen. Dit doen we omdat we de max
  en #behandelde klanten moeten bijhouden
  wachtende = numb_wait(r)
  w <- c(w,wachtende)
  wmax <- c(wmax,max(wachtende))
  waiting_timer <- r$Starttime-r$Arrival
  wtmax <- c(wtmax,max(waiting_timer))
  waiting_time <- c(waiting_time, waiting_timer)
  dayend <- c(dayend, max(r$Finishtime))

  Arrival <- c(Arrival,r$Arrival)
  Starttime <- c(Starttime,r$Starttime)
  Finishtime <- c(Finishtime,r$Finishtime)
  Server <- c(Server,r$Server)
  Serversvrij <- c(Serversvrij,r$Vrijeservers)

  free_time1 <- c(free_time1,sum(free_times(r,1)))
  free_time2 <- c(free_time2,sum(free_times(r,2)))
  free_time3 <- c(free_time3,sum(free_times(r,3)))
  free_time4 <- c(free_time4,sum(free_times(r,4)))

  served1 <- c(served1, sum(r$Server==1))
  served2 <- c(served2, sum(r$Server==2))
  served3 <- c(served3, sum(r$Server==3))
  served4 <- c(served4, sum(r$Server==4))
}

#b
time_spent <- Finishtime-Arrival #We berekenen de tijd die de klant in het systeem doorbrengt
mean(time_spent)

#c
mean(w)

#d
sum(waiting_time==0)/length(waiting_time)

#e
mean(waiting_time)

#f
mean(wtmax)

#g
mean(wmax)

#h
mean(free_time1) #We berekenen de gemiddelde tijd dat de bediende vrij heeft
mean(free_time2)
mean(free_time3)
mean(free_time4)

#i
sum(dayend<60*8)/length(dayend) #Indien het laatste vertek voor tijdstip 60*8 gebeurt,
                                voldoet het aan de situatie van de vraag
```

```
#j
mean(served1)
mean(served2)
mean(served3)
mean(served4)

#k
sum(Serversvrij>=2)/length(Serversvrij)

#Alles in 1 lijst
oef3.1 = list("1.b" = mean(time_spent),
         "1.c" = mean(w),"1.d" = sum(waiting_time==0)/length(waiting_time),
         "1.e" = mean(waiting_time),
         "1.f" = mean(wtmax),
         "1.g" = mean(wmax),
         "1.h" = c(mean(free_time1[free_time1!=0]), mean(free_time2[free_time2!=0]),
                 mean(free_time3[free_time3!=0]), mean(free_time4[free_time4!=0])),
         "1.i" = sum(dayend<60*8)/length(dayend),
         "1.j" = c(mean(served1),mean(served2),mean(served3),mean(served4)),
         "1.k" = sum(Serversvrij>=2)/length(Serversvrij))



set.seed(1234567)
simulate_wachtrij2 <- function(lambda, T) { #Deze functie is zo goed als exact hetzelfde
                                            van de vorige functie
  Arrival <- c()
  Server <- c()
  Starttime <- c()
  Finishtime <- c()
  end_time_server <- c(0,0,0,0)
  Free_servers <- c(1,2,3,4)
  k <- 1
  tt <- rexp(1,rate=lambda)
  while (tt<T) {
    Arrival[k] <- tt
    Free_server <- Free_servers[end_time_server<tt]
    if (length(Free_server) == 0) {
      stophelp <- min(end_time_server)
      if ((stophelp-Arrival[k])>=10){ #We kijken hier of de wachttijd groter zal zijn dan 10
        Server[k] <- 0 #Als dit zo is, gaat de klant naar server 0 (weg)
        Starttime[k] <- Arrival[k]+10 #De klant is 10 minuten erin gebleven
        Finishtime[k] <- Arrival[k]+10 #Dit dient voor #mensen in systeem
      } else {
        if (stophelp==end_time_server[1]) {
          Server[k] <- 1
          S <- rgamma(1,shape=1/3,rate=1/12)
        } else if (stophelp==end_time_server[2]){
          Server[k] <- 2
          S <- rgamma(1,shape=1/2,rate=1/7)
        } else if (stophelp==end_time_server[3]){
          Server[k] <- 3
          S <- rexp(1,rate=0.4)
        } else {
          Server[k] <- 4
          S <- runif(1,min=2,max=7)
        }
        Starttime[k] <- stophelp
        Finishtime[k] <- Starttime[k]+S
```

```
        end_time_server[stophelp==end_time_server] <- Finishtime[k]
      }
    } else {
      if (length(Free_server)==1) {
        fs <- Free_server[1]
      } else {
        fs <- sample(Free_server,1)
      }
      Starttime[k] <- tt
      if(fs==1) {
        Server[k] <- 1
        S <- rgamma(1,shape=1/3,rate=1/12)
      } else if (fs==2) {
        Server[k] <- 2
        S <- rgamma(1,shape=1/2,rate=1/7)
      } else if (fs==3) {
        Server[k] <- 3
        S <- rexp(1,rate=0.4)
      } else {
        Server[k] <- 4
        S <- runif(1,min=2,max=7)
      }
      Finishtime[k] <- tt+S
      end_time_server[fs] <- Finishtime[k]
    }
    k <- k+1
    tt <- tt+rexp(1,rate=lambda)
  }
  return (list("Arrival"=Arrival,"Starttime"=Starttime,"Finishtime"=Finishtime,"Server"=Server))
}

#Deze vectoren hebben dezelfde betekenis als voordien
w <- c()
wtmax <- c()
waiting_time <- c()

Arrival <- c()
Starttime <- c()
Finishtime <- c()
Server <- c()

for (i in 1:100) { #Doet exact hetzelfde als voordien (we berekenen wel minder)
  r <- simulate_wachtrij2(1, 8*60 )
  wachtende <- numb_wait(r)
  w <- c(w,wachtende)
  waiting_timer <- r$Starttime-r$Arrival
  wtmax <- c(wtmax,max(waiting_timer))
  waiting_time <- c(waiting_time, waiting_timer)

  Server <- c(Server,r$Server)
  Arrival <- c(Arrival,r$Arrival)
  Starttime <- c(Starttime,r$Starttime)
  Finishtime <- c(Finishtime,r$Finishtime)
}

#a.b
time_spent <- Finishtime-Arrival
mean(time_spent)

#a.c
```

```
mean(w)

#a.d
sum(waiting_time==0)/length(waiting_time)

#a.e
mean(waiting_time)

#a.f
mean(wtmax)

#b
sum(Server==0)/length(Server) #kijkt ratio van klanten in server 0 (weg) en klanten

#Alles in 1 vector
oef3.2 = list("2.a.b" = mean(time_spent),
         "2.a.c" = mean(w),
         "2.a.d" = sum(waiting_time==0)/length(waiting_time),
         "2.a.e" = mean(waiting_time),
         "2.a.f" = mean(wtmax),
         "2.b"= sum(Server==0)/length(Server))
```