Project 1 - Code

By: Sanne Jamila Razmara Olsen & Einride Brodahl Osland

Problem 1c

The following code consists of the function *simulate()* which simulates the Markov chain given by the probability matrix **P** for iter amount of days.

```
simulate <- function(P, iter){
  #Finding the number of rows of P
  n <- nrow(P)
  #Initializing vector of the states, iter states
  states <- numeric(iter)

  #Assumption X_0 = 0
  states[1] <- 1

  #Simulating Markov chain
  for(t in 2:iter){
    #Probability vector to simulate next state (X_t+1)
    p <- P[states[t-1],]
    #Draw from multinomial distrubution to choose next state
    states[t] <- which(rmultinom(1,1,p) == 1)
  }
  return(states)
}
```

Constructing **P** by the values given in the project description we test our implementation of the simulation for iter = 7300 days.

```
beta = 0.01
gamma = 0.1
alpha  = 0.005

P = matrix(c(1-beta,beta,0,0,1-gamma,gamma,alpha,0,1-alpha),nrow = 3, byrow = TRUE)

#State 0,1,2 transformed to state 1,2,3
n = 7300

states <- simulate(P,n)
```

We want to estimate the limiting probabilities by the function *limiting_probs()*.

```
limiting_probs <- function(states){
  n = length(states)
  p1=0
  p2 = 0
```

```
  p3 = 0

  for(i in states){
    if(i == 1){
      p1 = p1 +1
    }
    else if(i == 2){
      p2 = p2 +1
    }
    else{
      p3 = p3 +1
    }
  }
  p1 = p1/n
  p2 = p2/n
  p3 = p3/n

  pi <- c(p1,p2,p3)
  return(pi)
}
```

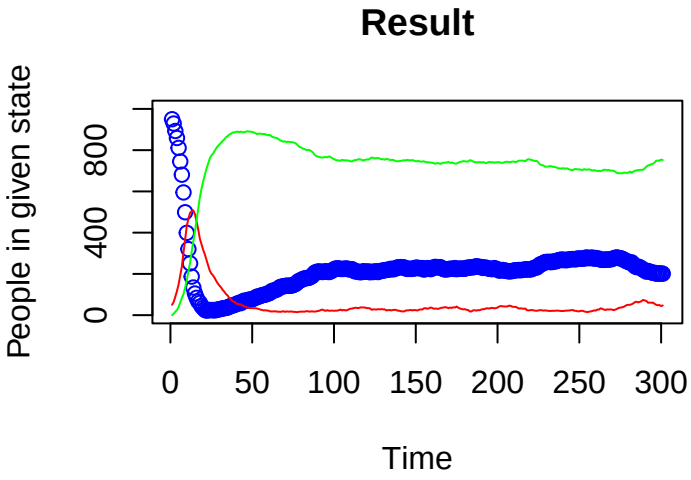
The following code of the implementation *confidence_interval()* constructs a 95% confidence interval for the j´th limiting probability.

```
confidence_interval <- function(P,N,j){
  pi = numeric(N)

  for(i in 0:N){
    states = simulate(P,7300)
    states = states[3650:7300]

    pi[i] <- limiting_probs(states)[j]

  }
  n = N

  t = qt(0.95,df = n-1)

  variance = var(pi)
  mean = mean(pi)

  S = sqrt(n/(n-1)*variance)

  a = mean + t*S/sqrt(n)
  b = mean - t*S/sqrt(n)

  return(c(a,b))
}
```

Problem 1e

```r
set.seed(123)
# Number of trials, with vectors to save the values in the process
n<- 300
S <- vector(length=n+1)
I <- vector(length=n+1)
R <- vector(length=n+1)

#initial states of the population
S[1]<- 950
I[1]<- 50
R[1]<- 0

for (i in 1:n){
    #inducing the new
    S_n <- rbinom(1, R[i], .005)
    I_n <- rbinom(1, S[i], .5*I[i]/1000)
    R_n <- rbinom(1, I[i], .1)
    # Saving the new states into the vector introduced previously
    S[i+1]= S[i]+S_n-I_n
    I[i+1]= I[i]+I_n-R_n
    R[i+1]= R[i]+R_n-S_n
    #testing that the code works by seeing if the number of "peopl" sum to 1000
    #print(R[i]+I[i]+S[i])


}
step<- seq(301)
```

```r
plot(step, S,col="blue", main="Result",xlab="Time", ylab="People in given state",ylim=c(0,1000))
lines(step, I,col="red")
lines(step, R, col="green")
```



Problem 1f

```r
N<- 1000
Large_infect= vector(length= N)
Large_infect_step= vector(length= N)

for(j in 1:N){

    #using the same code as for e for the simulations
    n<- 300
    S <- vector(length=n+1)
    I <- vector(length=n+1)
    R <- vector(length=n+1)

    S[1]<- 950
    I[1]<- 50
    R[1]<- 0

    #initialize the maximum number of infected and its index
    max_I<-0
    index<-0


    for (i in 1:n){

        S_n <- rbinom(1, R[i], .005)
        I_n <- rbinom(1, S[i], .5*I[i]/1000)
        R_n <- rbinom(1, I[i], .1)
    # Saving the new states into the vector introduced previously
        S[i+1]= S[i]+S_n-I_n
        I[i+1]= I[i]+I_n-R_n
        R[i+1]= R[i]+R_n-S_n

        if(I[i+1]>max_I){
            max_I <- I[i+1]
            index <- (i+1)
        }
    }
    #pushing setting the maxvalue for the iteration
    Large_infect[j]= max_I
    Large_infect_step[j]= index

}
#plot(Large_infect, Large_infect_step)

#Expected value of max infected:
print(mean(Large_infect))
```

```
## [1] 522.611
```

```r
#Expected timestep to get to the max infected:
print(mean(Large_infect_step))
```

```
## [1] 12.882
```

```r
#Confidence intervals:

print(quantile(Large_infect,c(.025,.975)))
```

```
##    2.5%   97.5%
## 482.000 560.025
```

```r
print(quantile(Large_infect_step,c(.025,.975)))
```

```
##  2.5% 97.5%
##    11    14
```

Problem1g g) making a function to simulate with different number of vaccinated we assume that the vaccinated cant become sick, thus we just remove them from the S variable, and use the same code as in f

```r
set.seed(123)

Vacc_simulate= function(iteration, vaccinated){

N<- iteration
Large_infect= vector(length= N)
Large_infect_step= vector(length= N)

for(j in 1:N){

    #using the same code as for e for the simulations
    n<- 300
    S <- vector(length=n+1)
    I <- vector(length=n+1)
    R <- vector(length=n+1)

    S[1]<- 950- vaccinated
    I[1]<- 50
    R[1]<- 0

    #initialize the maximum number of infected and its index
    max_I<-0
    index<-0


    for (i in 1:n){

        S_n <- rbinom(1, R[i], .005)
        I_n <- rbinom(1, S[i], .5*I[i]/1000)
        R_n <- rbinom(1, I[i], .1)
    # Saving the new states into the vector introduced previously
        S[i+1]= S[i]+S_n-I_n
        I[i+1]= I[i]+I_n-R_n
        R[i+1]= R[i]+R_n-S_n

        if(I[i+1]>max_I){
            max_I <- I[i+1]
```

```
            index <- (i+1)
        }
    }
    #pushing setting the maxvalue for the iteration
    Large_infect[j]= max_I
    Large_infect_step[j]= index
}
print(mean(Large_infect))
print(mean(Large_infect_step))
return(I)
}

#Simulating the different cases, and one case where there are no vaccinated
No_vac<-Vacc_simulate(1000, 0)
```

```
## [1] 522.61
## [1] 12.884
```

```
I_case1<-Vacc_simulate(1000, 100)
```

```
## [1] 439.697
## [1] 13.642
```

```
I_case2<-Vacc_simulate(1000, 600)
```

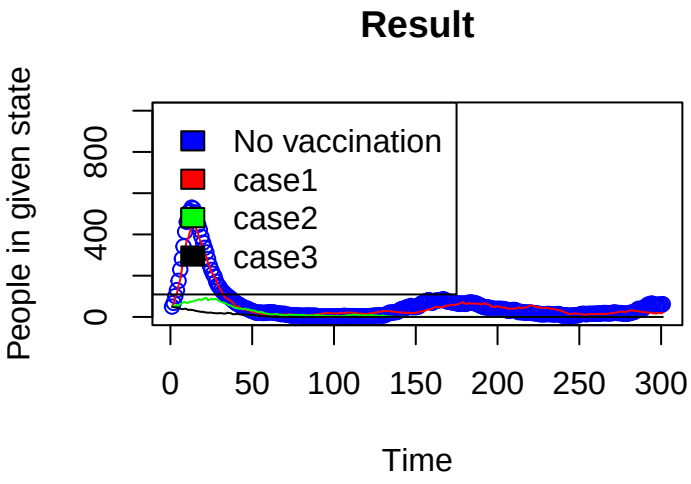
```
## [1] 97.619
## [1] 16.945
```

```
I_case3<-Vacc_simulate(1000, 800)
```

```
## [1] 50.033
## [1] 3.079
```

```
step<- seq(301)
```

```
plot(step, No_vac,col="blue", main="Result",xlab="Time", ylab="People in given state",ylim=c(0,1000))
lines(step, I_case1,col="red")
lines(step, I_case2, col="green")
lines(step, I_case3, col="black")
legend("topleft",legend=c("No vaccination","case1","case2","case3"), fill= c("blue","red","green","blac
```



Problem 2a

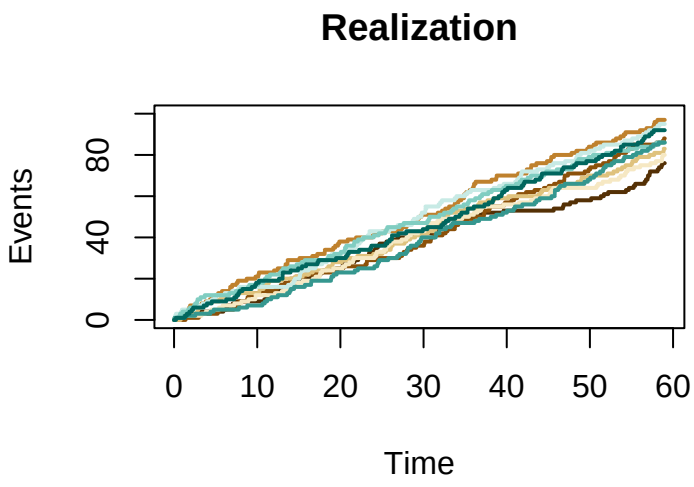
The function *Poisson_Process()* calculates n realizations of the Poisson Process.

```
Poisson_Process <- function(lambda,n,t){
  #Creating empty vector of size n+1
  v<-numeric(n+1)
  v[1] <-0


  for (k in 1:n){
    x <- rpois(1,lambda*t)
    if(x > 100){
      v[k] = 1
    }
    else{
      v[k] = 0
    }
  }
  return(v)
}

t <- 59
n <- 1000
lambda <-1.5

v = Poisson_Process(lambda,n,t)
mean(v)
```

```
## [1] 0.1218781
```

The mean is a representation of the probability that there are more than 100 claims on day 59.

```r
#install.packages("RColorBrewer")
library(RColorBrewer)


plot_poisson <-function(t_value,lambda,N){
  #Creating colour palette to visualize simulations
  colour_palette = brewer.pal(n=10, name = "BrBG")
  #Generating plot to fill with simulation plots
  plot(NULL, NULL, xlim = c(0, t_value), ylim = c(0, 100), xlab = "Time", ylab = "Events", main = "Reali



  #Simulating N Poisson processes and plotting them by using lines() function
  for (n in 0:N) {
    #Picking from Poisson distribution
    X <- rpois(1, lambda * t_value)
    x = c(0:X, X)

    t <- runif(X, 0, t_value)
    t = c(0, sort(t), t_value)
    l = length(x)

    for (i in 1:(l - 1)){
      lines(t[i:(i + 1)], rep(x[i], 2), lwd = 2, col = colour_palette [n]) #Adding simulation to plot
    }
  }
}


#Initializing given values
t_value <- 59
lambda <- 1.5
N <- 9

plot_poisson(t_value,lambda,N)
```



Problem 2b

```
#Defining parameters
N <- 1000
gamma <- 10
lambda <- 1.5

simulate_poisgam <- function(N,gamma,lambda){
  t_value <- 59
  v <- numeric(N)

  #Simulating
  for (n in 1:N) {
    #Poisson distribution
    X <- rpois(1, lambda * t_value)
    z <- numeric(N)

    for (i in 1:X){
      #Exponential distribution
      c <- rexp(1, gamma)
      z[i] = c
    }
    #Finding probs
    if (sum(z) > 8) {
      v[n] = 1
    } else {
      v[n] = 0
    }
  }

  return(v)
}
```

```r
#Probability
mean(simulate_poisgam(N,gamma,lambda))
```

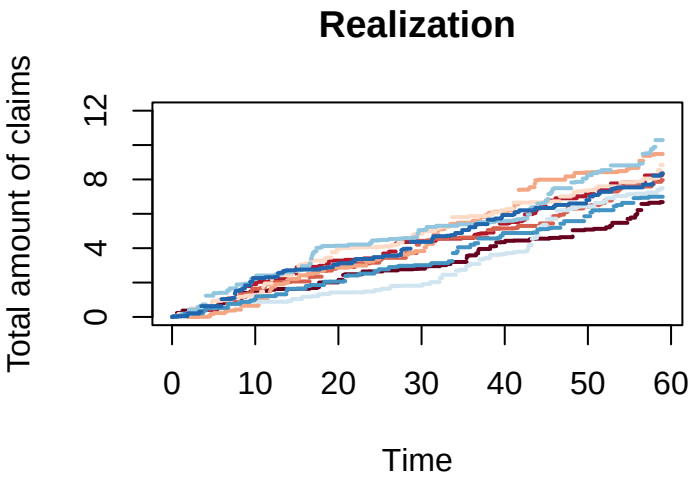
```
## [1] 0.71
```

The estimated probability of the total claim amount exceeding 8 mill kr. after 59 days is 0.971.

We now want to provide the estimated probability by making a figure that shows 10 realizations of $Z(t)$, for $t \in [0, 59]$ in the same figure.

```r
color_palette <- brewer.pal(n = 10, name = 'RdBu')

lambda <- 1.5
gamma <- 10
t_value <- 59
N <- 9

final_simulation <-function(lambda,gamma,t_value,N){
  # Plot
  plot(NULL, NULL, xlim = c(0, t_value), ylim = c(0, 12), xlab = "Time", ylab = "Total amount of claims"
  #Simulation
  for (n in 0:N) {
    X <- rpois(1, lambda * t_value)

    t <- runif(X, 0, t_value)
    t = c(0, sort(t), t_value)

    x= c(0:X, X)
    z <- numeric(X)


    for (i in 0:X){
      C <- rexp(1, gamma)
      z[i] <- C
    }


    cumulative_z <- c(0, cumsum(z))
    l = length(x)
    for (i in 1:(l - 1)){
      lines(t[i:(i + 1)], rep(cumulative_z[i], 2), lwd = 2, col = color_palette[n])
    }
  }
}
```

```r
final_simulation(lambda,gamma,t_value,N)
```


Realization
Total amount of claims
0
4
8
12
0
10
20
30
40
50
60
Time