Simulation project 2

Authors

Joachim Secher, s183973	Nicklas Bruun-Andersen, s183979	Gustav Als, s184400
Mads Esben Hansen, s174434	Magne Egede Rasmussen, s183963	Peter Kampen, s183979

Contribution table:

```
##        Nicklas Joachim Magne Gustav Mads Peter
## Task 1     0.1     0.2   0.1    0.1  0.3   0.2
## Task 2     0.2     0.1   0.3    0.1  0.2   0.1
## Task 3     0.2     0.2   0.1    0.1  0.3   0.1
## Task 4     0.1     0.1   0.3    0.2  0.1   0.2
## Task 5     0.3     0.1   0.1    0.2  0.2   0.1
## Task 6     0.1     0.1   0.1    0.3  0.1   0.3
## Task 7     0.2     0.2   0.1    0.1  0.2   0.2
## Task 8     0.2     0.1   0.1    0.2  0.2   0.2
## Task 9     0.1     0.1   0.1    0.3  0.2   0.2
```

Packages

```
library("stats")
library("lmerTest")
```

```
## Indlæser krævet pakke: lme4

## Indlæser krævet pakke: Matrix

##
## Vedhæfter pakke: 'lmerTest'

## Det følgende objekt er maskeret fra 'package:lme4':
##
##     lmer

## Det følgende objekt er maskeret fra 'package:stats':
##
##     step
```

Intro

In this study we investigate the patient flow in hospitals in hope of eventually optimizing the planning decision of distributing beds between wards using stochastic simulation. The problem is designed such that we have the same amount of patient types and wards, i.e:

$$\text{Patients: } \mathcal{P} = \{A, B, \ldots, F\}, \quad \text{Wards: } \mathcal{W} = \{A, B, \ldots, F\}$$

We denote it as primary hospitalization, when a patient is stored at the the initial correct ward, $\mathcal{P} = \mathcal{W}$. If there is no empty beds in the ward, we consider secondary hospitalization, where we allow stochastic reallocation of patients between wards. The reallocation probabilities associated with patients of type $\mathcal{P}$ to wards of type $\mathcal{W}$ are presented in the table below.

$\mathcal{P}$ \ $\mathcal{W}$	A	B	C	D	E	F^*
A	-	0.05	0.10	0.05	0.80	0.00
B	0.20	-	0.50	0.15	0.15	0.00
C	0.30	0.20	-	0.20	0.30	0.00
D	0.35	0.30	0.05	-	0.30	0.00
E	0.20	0.10	0.60	0.10	-	0.00
F^*	0.20	0.20	0.20	0.20	0.20	-

Figure 1: Probability, p_{ij} , of relocating a patient of type $i \in P$ to an alternative Ward $j \in W$. Includes the new Ward $F*$.

Likewise, blocking a patient at the initial ward is called a primary blocking, as well as patient blocking at the alternative ward is denoted a secondary blocking. If a patient is blocked twice, the patient will leave the hospital and will not get any kind of service.

In general we assume patients from different patient types to arrive according to Poisson processes with different rates. Further, we assume that their inter-service time distributions are exponentially distributed also with different rates. To gain an overview for our case we refer to the following table:

Ward and patient type	Bed capacity	Arrivals per day (λ_i)	Mean length-of-stay ($1/\mu_i$)	Urgency points
A	55	14.5	2.9	7
B	40	11.0	4.0	5
C	30	8.0	4.5	2
D	20	6.5	1.4	10
E	20	5.0	3.9	5
F^*	*To be decided*	13.0	2.2	*Not relevant*

Figure 2: Parameters associated with each ward and patient type. Ward $F*$ denotes the new ward and the urgency points (column 5) reflects the "penalty" if a patient of type i is not admitted in Ward i

Notice that here we have also included a maximum bed capacity emphasising the need of reallocation.

Primary tasks

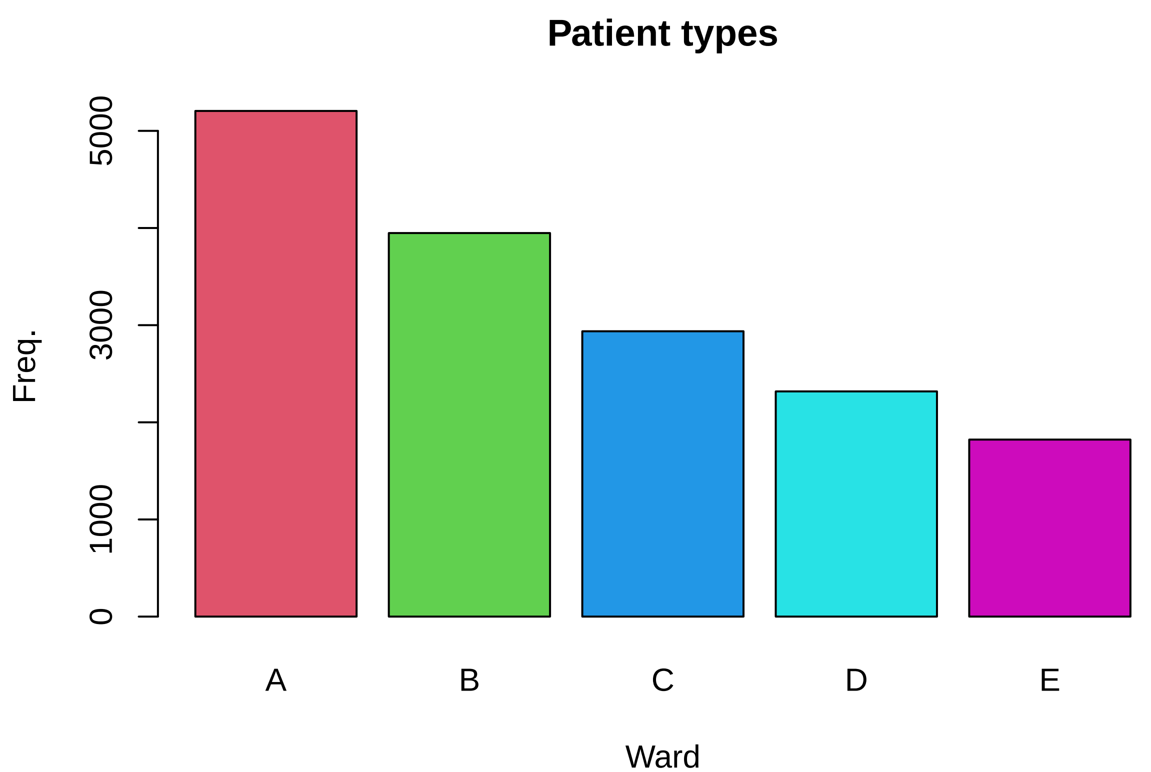
1) Build a simulation model that simulates the patient flow in the hospital

We start by building a simulation model that simulates the patient flow in the hospital as a function of the bed distribution and the mentioned parameters.

To do so, we start by drawing arrival and service times for sufficiently many patients (1 years worth) for the different types of patients.

```
a.rates = c(14.5, 11.0, 8.0, 6.5, 5.0)
s.rates = 1/c(2.9,4.0, 4.5, 1.4, 3.9)
a.times <- s.times <- type <- list()  ## arrival times / service times / types

set.seed(69)
for (ii in 1:length(a.rates)) {
  N = 1000
  a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
```

```
  while (tail(a.times[[ii]],1) < 365){
    N = N*2
    a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
  }
  a.times[[ii]] = a.times[[ii]][a.times[[ii]] <= 365]

  s.times[[ii]] = rexp(n = N, rate = s.rates[ii])
  type[[ii]] = rep(ii, length(a.times[[ii]]))
}

## unlist
a.times = unlist(a.times)
s.times = unlist(s.times)
type = unlist(type)

## order
s.times = s.times[order(a.times)]
type = type[order(a.times)]
a.times = a.times[order(a.times)]

barplot(sapply(1:5, function(i) sum(type==i)), names.arg = c("A","B","C","D","E"),
        ylab = "Freq.", xlab = "Ward", main = "Patient types", col = 2:6)
```

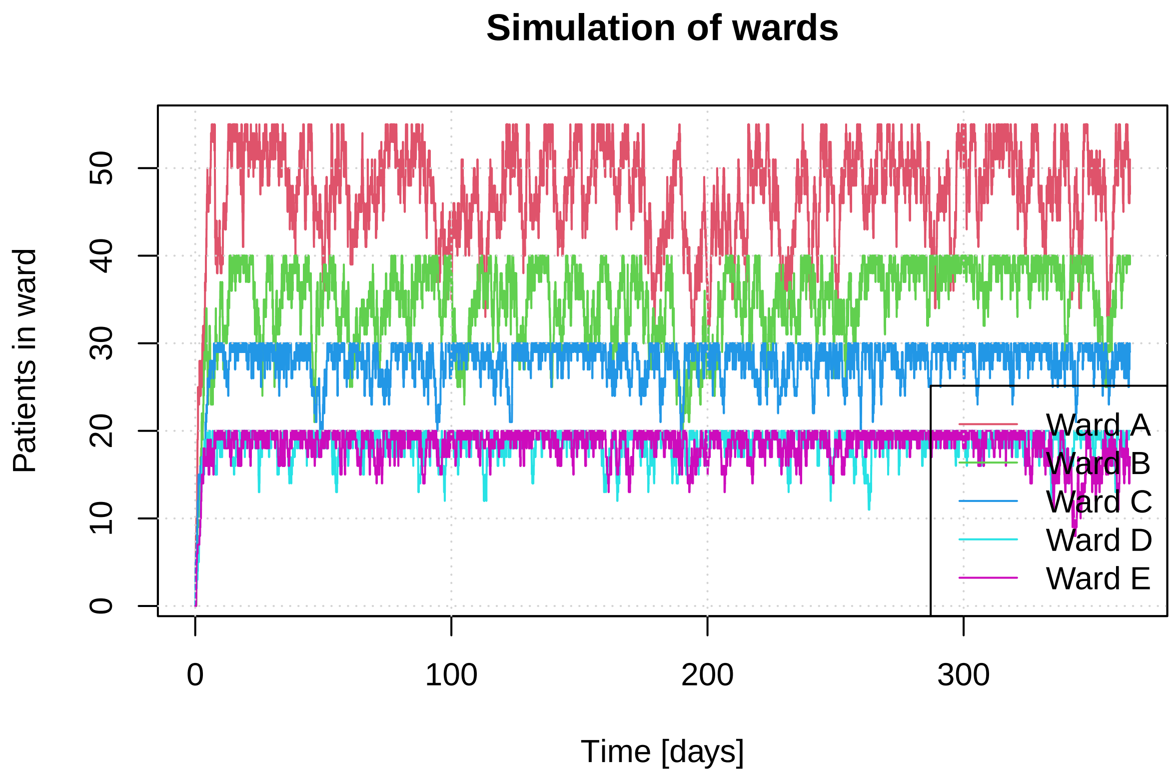


We can now allocate patients iteratively to desired wards. If there is no more room in a given ward, we follow the rules as described in the exercise, i.e., we allocate to another ward with probabilities as given in the table. If there is no room in the chosen ward for reallocation either, the patients will not be admitted at all.

```r
tm = t(matrix(c(0,0.05, 0.10, 0.05, 0.8,
       0.2, 0, 0.5, 0.15, 0.15,
       0.3, 0.2, 0, 0.2, 0.3,
       0.35, 0.30, 0.05, 0, 0.3,
       0.2, 0.1, 0.6, 0.1, 0),5,5))
beds = c(55,40,30,20,20)

allocate.patients <- function(beds, a.times, s.times, type, tm){
    ## init variables
    in.service = lapply(beds, function(b) rep(0, b))
    status = matrix(NA, length(a.times), length(in.service))
    total.block = rep(0, length(in.service))
    re.alloc = matrix(0, length(in.service), length(in.service))

    ## Allocate patients
    for (ii in 1:length(a.times)) {
      if (all(a.times[ii] < in.service[[type[ii]]])){
        try.type = sample(1:length(in.service), 1, prob = tm[type[ii], ])
        if (all(a.times[ii] < in.service[[try.type]])){
          ## no free beds :(
          total.block[type[ii]] = total.block[type[ii]] + 1
        }
        else {
          tmp_min = which.min( in.service[[ try.type ]] >= a.times[ii] )
          in.service[[ try.type ]][ tmp_min ] = a.times[ii]+s.times[ii]
          re.alloc[type[ii], try.type] = re.alloc[type[ii], try.type] + 1
        }
      }
      else {
        tmp_min = which.min( in.service[[type[ii]]] >= a.times[ii] )
        in.service[[type[ii]]][ tmp_min ] = a.times[ii]+s.times[ii]
      }
      status[ii, ] = sapply(1:length(in.service), function(jj) sum(in.service[[jj]]>= a.times[ii]))
    }
    list(
      status = status,
      re.alloc = re.alloc,
      total.block = total.block
    )
}
```

We can now assess how patients are distributed over times at the 5 wards.

```r
sol = allocate.patients(beds, a.times, s.times, type, tm)

plot(a.times,sol$status[,1], type = 'l', col = 2, main = "Simulation of wards",
     ylab = "Patients in ward", xlab = "Time [days]")
aux <- sapply(2:5, function(ii) lines(a.times,sol$status[,ii], col = ii+1))
grid()
legend('bottomright', paste("Ward", c("A","B","C","D","E")), col = 2:6, lty = 1)
```



Additionally, we can assess how people are reallocated:

```
sol$re.alloc
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0   12   15   14  101
## [2,]   88    0  135   33   43
## [3,]  214  142    0   87  155
## [4,]  309  237   36    0  181
## [5,]  124   52  245   34    0
```

Notice that especially patient of type D are reallocated to ward A, when ward D is full. Additionally, patients of type A are reallocated to ward E far more than to any of the other wards. This is consistent with the probability table given in the exercise.

Finally, we can assess the distributed of people who are not admitted at all by type.

```
sol$total.block
```

```
## [1] 150 191 246 207 195
```

It seems that the type has little impact on if a patient is admitted to the hospital. Maybe there is a slight over-representation of patients of type C.

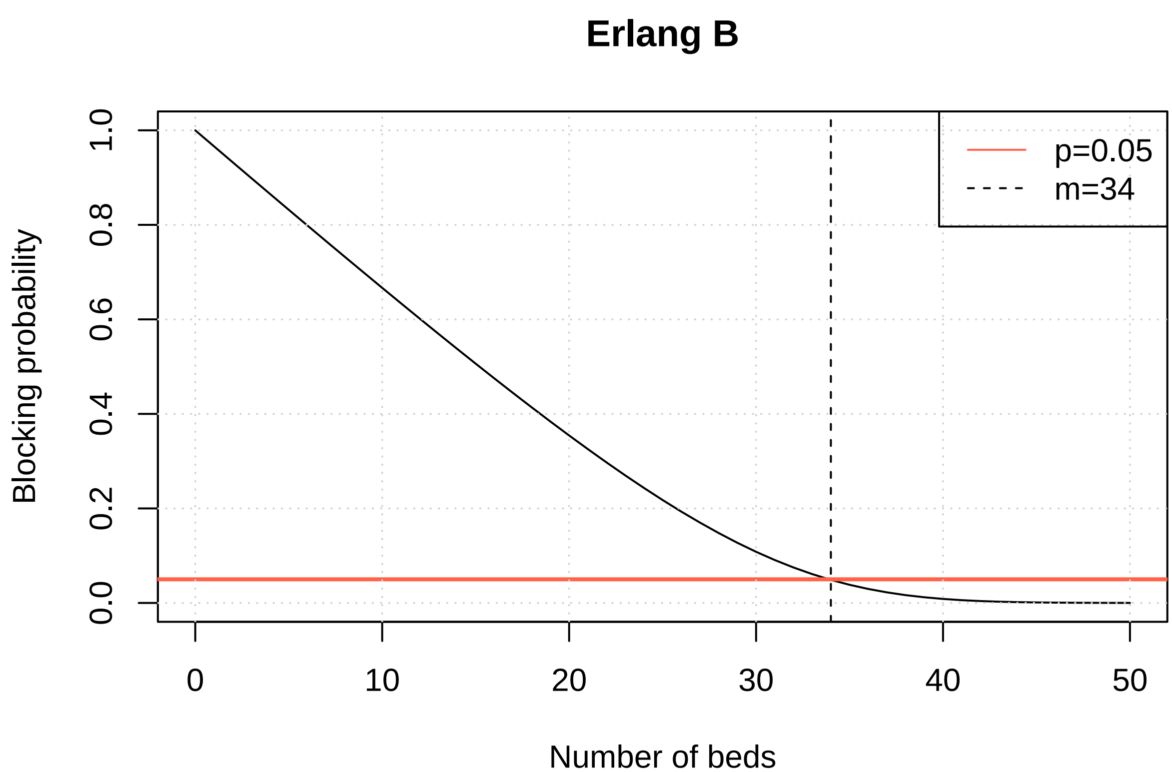
2) and 3) Create a new ward F, and asses the implcations of this.

We are now asked to create a ward F, with number of beds such that at least 95% of patients of type F are allocated to ward F. We immediately notice that no patients from other wards are reallocated to F. This means that F only contains patients of type F. We can therefore use Erlang-B to estimate the blocking probability. Let m denote the number of beds, λ the arrival intensity, s the mean service time, and $A = \lambda \cdot s$. Let B denote the blocking probability, then

$$B = \frac{\frac{A^m}{m!}}{\sum_{i=0}^{m} \frac{A^i}{i!}}.$$

We want the lowest number of beds such that $B \leq 0.05$.

```r
## Find number of beds in F using erlang B
lambda = 13.0; s = 2.2; A = lambda*s
erlang.b <- function(A,m){
  A^m/factorial(m)/sum(sapply(0:m, function(i) A^i/factorial(i)))
}
plot(0:50, sapply(0:50, function(i) erlang.b(A,i)), type = 'l',
     xlab = 'Number of beds', ylab = 'Blocking probability',
     main = "Erlang B")
abline(h=0.05, col = "tomato", lwd = 2)
abline(v = 34, lty = 2)
grid()
legend('topright', c("p=0.05", "m=34"), lty = c(1,2), col = c("tomato", 1))
```



We find that

$$m^* = \min\{m | B(m) \leq 0.05\} = 34,$$

i.e., there must be at least 34 beds in ward F.

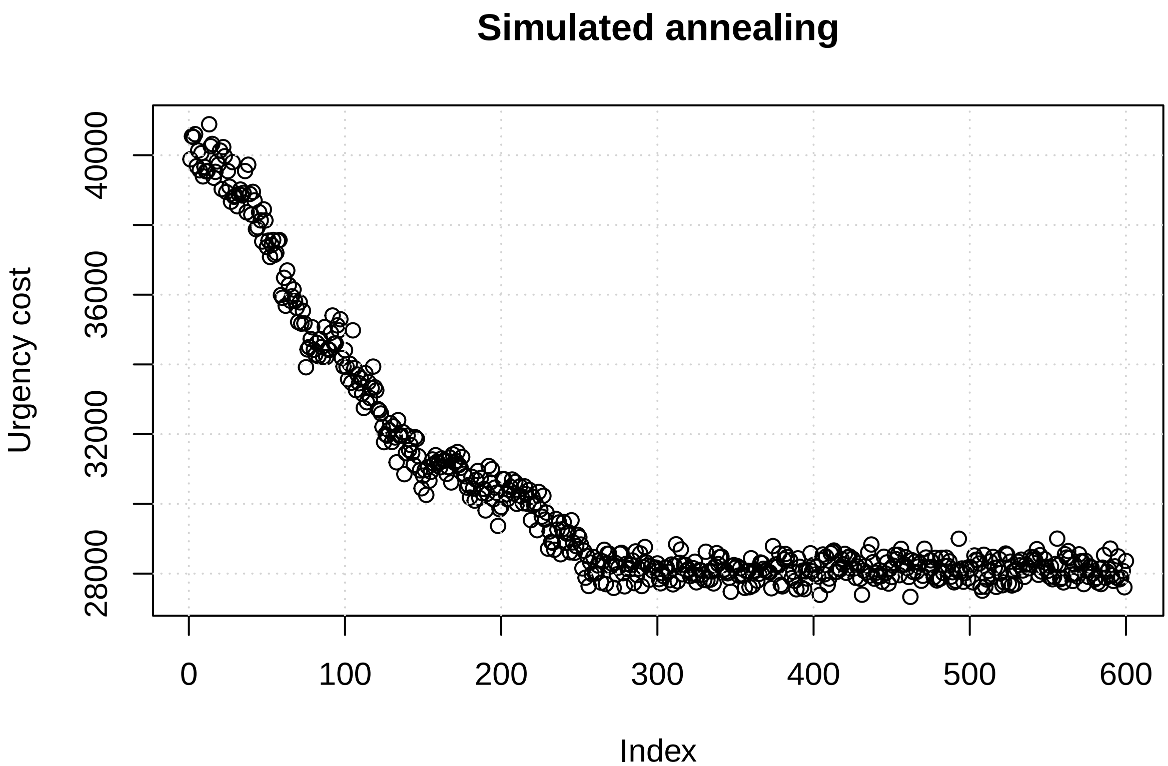
We now want to reallocate beds optimally, maintaining the total number of beds in the hospital. To do so we use the *urgency points*. We create a cost function that sums the total number of patients of type i that has not been allocated to ward i weighted by the urgency of ward i.

```
queue.cost <- function(beds) {
  in.service = lapply(beds, function(b) rep(0, b))
  cost = 0
  for (ii in 1:length(a.times)) {
    if (all(a.times[ii] < in.service[[type[ii]]])){
      cost = cost + ifelse(type[ii]!=6, urgency[type[ii]], 0)
      try.type = sample(1:length(in.service), 1, prob = tm[type[ii], ])
      if (all(a.times[ii] < in.service[[try.type]])){
        ## no free beds :(
      }
      else {
        tmp_min = which.min( in.service[[ try.type ]] >= a.times[ii] )
        in.service[[ try.type ]][ tmp_min ] = a.times[ii]+s.times[ii]
      }
    }
    else {
      tmp_min = which.min( in.service[[type[ii]]] >= a.times[ii] )
      in.service[[type[ii]]][ tmp_min ] = a.times[ii]+s.times[ii]
    }
  }
  cost
}
```

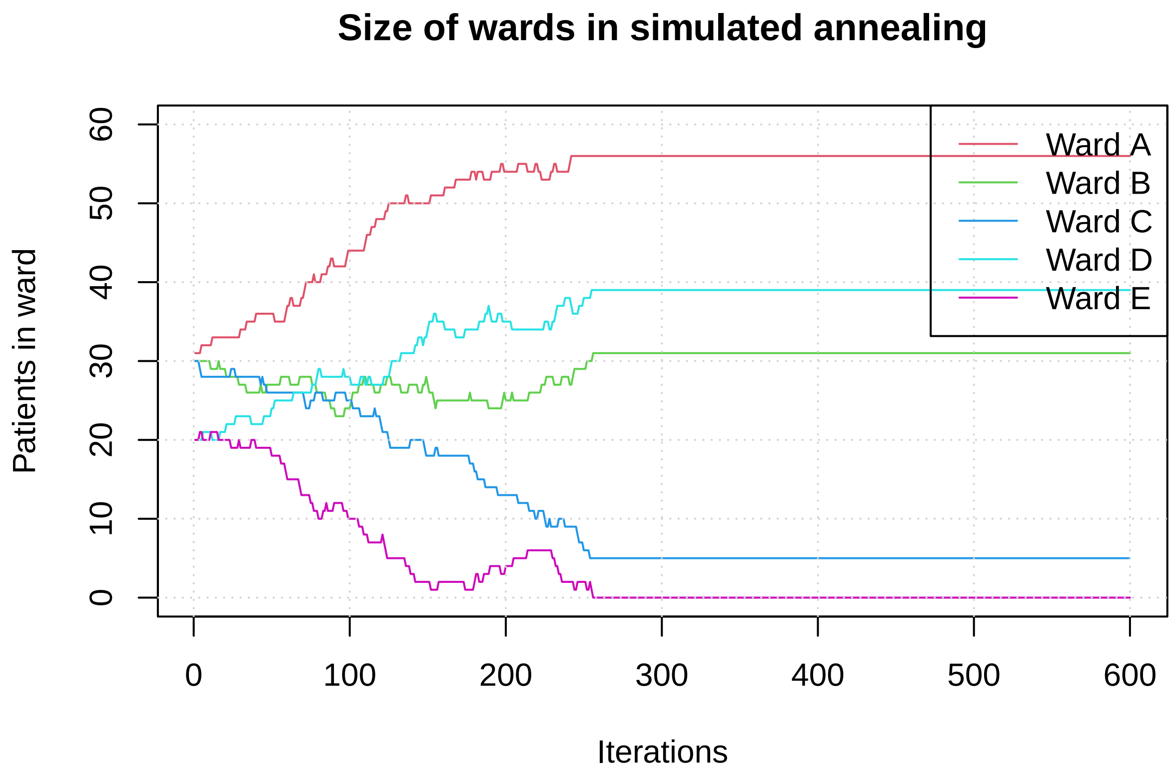
We can now use simulated annealing to optimize the distribution of beds according to the urgency points.

```
a.rates = c(14.5, 11.0, 8.0, 6.5, 5.0, 13.0)
s.rates = 1/c(2.9,4.0, 4.5, 1.4, 3.9, 2.2)
urgency = c(7,5,2,10,5)
a.times <- s.times <- type <- list()  ## arrival times / service times / types

set.seed(69)
for (ii in 1:length(a.rates)) {
  N = 1000
  a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
  while (tail(a.times[[ii]],1) < 365){
    N = N*2
    a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
  }
  a.times[[ii]] = a.times[[ii]][a.times[[ii]] <= 365]
  s.times[[ii]] = rexp(n = N, rate = s.rates[ii])
  type[[ii]] = rep(ii, length(a.times[[ii]]))
}

## unlist
a.times = unlist(a.times)
s.times = unlist(s.times)
type = unlist(type)

## order
s.times = s.times[order(a.times)]
type = type[order(a.times)]
a.times = a.times[order(a.times)]
```

```r
tm = t(matrix(c(0,0.05, 0.10, 0.05, 0.8, 0,
                0.2, 0, 0.5, 0.15, 0.15, 0,
                0.3, 0.2, 0, 0.2, 0.3, 0,
                0.35, 0.30, 0.05, 0, 0.3, 0,
                0.2, 0.1, 0.6, 0.1, 0, 0,
                0.2, 0.2, 0.2, 0.2, 0.2, 0),6,6))
T.fun <- function(k){
  1/sqrt(1+k)
}
optim.queue <- function(beds0, fun, N) {
  ## init route
  beds = matrix(NA,nrow = N, ncol = length(beds0))
  cost = length(beds0)
  beds[1,] = beds0
  cost[1] = fun(beds0)

  for (ii in 2:N) {
    # propose route
    beds.new = beds[ii-1, ]
    while ( (!any(beds[ii-1, ] != beds.new)) && all(beds.new>0) ){
      beds.new = beds[ii-1, ]
      idx = sample(1:5,2, replace = F)
      beds.new[idx[1]] = beds.new[idx[1]] - 1
      beds.new[idx[2]] = beds.new[idx[2]] + 1
    }
    c0 = fun(beds[ii-1,])
    c1 = fun(beds.new)
    ## accept/reject
    beds[ii, ] = `if`( c1<c0 || (exp(-(c1 - c0)/T.fun(ii))>runif(1)),
                        beds.new, beds[ii-1, ])
    cost[ii] = c1
    #print(ii)
  }
  list(beds = beds,
       cost = cost)
}

if (file.exists('sim.optim.beds.rds')){
  sim.beds = readRDS('sim.optim.beds.rds')
} else{
  sim.beds = optim.queue(c(31,30,30,20,20,34), queue.cost, 600)
  saveRDS(sim.beds,
        'sim.optim.beds.rds')
}
```


Simulated annealing
Urgency cost
28000
32000
36000
40000
0
100
200
300
400
500
600
Index


Distributed of beds
Initial
Optimal
Beds
0
10
20
30
40
50
A
B
C
D
E
F
Ward




```
## [1] "Optimal bed distribution:  56 31 5 39 0 34"
```

We see that the simulated annealing seems to converge after ~ 300 iterations. Notice that due to the generic stochasticity in the reallocation of patient, there is some generic noise in the cost function - even with the same distribution of beds. We see that ward A and B are large unchanged, ward C and E are basically closed and the number of beds in ward D is almost doubled! Looking at the urgency points, we see that ward D has the highest urgency, it therefore makes sense that the number of beds in this ward in particular is increased.

Primary performance measures

4) Estimate the probability that all beds are occupied on arrival, the expected number of admissions, and the expected number of relocated patients for each ward in the hospital.

We start by estimating the probability that all beds are occupied on arrival, the expected number of admissions, and the expected number of relocated patients for each ward in the hospital. To perform the estimates, we draw κ 1 year samples, and from each of these estimate the desired parameters. Via the CLT we can then assume these estimates to follow a normal distributed.

We start by assuming that ward F does not exist.

```
a.rates = c(14.5, 11.0, 8.0, 6.5, 5.0)
s.rates = 1/c(2.9,4.0, 4.5, 1.4, 3.9)
tm = t(matrix(c(0,0.05, 0.10, 0.05, 0.8,
         0.2, 0, 0.5, 0.15, 0.15,
         0.3, 0.2, 0, 0.2, 0.3,
         0.35, 0.30, 0.05, 0, 0.3,
```

```
         0.2, 0.1, 0.6, 0.1, 0),5,5))
beds = c(55,40,30,20,20)
kappa = 20
b.total <- e.admin <- all.occ <- rep(NA, kappa)
alloc.total = matrix(NA, kappa, length(beds))
set.seed(69)
for (ll in 1:kappa) {
  a.times <- s.times <- type <- list()  ## arrival times / service times / types
  for (ii in 1:length(a.rates)) {
    N = 1000
    a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
    while (tail(a.times[[ii]],1) < 365){
      N = N*2
      a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
    }
    a.times[[ii]] = a.times[[ii]][a.times[[ii]] <= 365]
    s.times[[ii]] = rexp(n = N, rate = s.rates[ii])
    type[[ii]] = rep(ii, length(a.times[[ii]]))
  }
  a.times = unlist(a.times)
  s.times = unlist(s.times)
  type = unlist(type)
  s.times = s.times[order(a.times)]
  type = type[order(a.times)]
  a.times = a.times[order(a.times)]

  sol = allocate.patients(beds, a.times, s.times, type, tm)

  all.occ[ll] = sum(apply(sol$status, 1, function(X) sum(X)==165))/length(a.times)
  b.total[ll] = sum(sol$total.block) / length(a.times)
  alloc.total[ll, ] = apply(sol$re.alloc, 1, sum)
  e.admin[ll] = length(a.times) - sum(sol$total.block)
}

print('Probability that all beds are occupied:' )
```

```
## [1] "Probability that all beds are occupied:"
```

```
print(mean(all.occ) + qt(0.975, df = kappa)*sd(all.occ)/sqrt(kappa)*c(-1,1))
```

```
## [1] 0.003087141 0.004256552
```

```
print('Probability of not being admitted:' )
```

```
## [1] "Probability of not being admitted:"
```

```
print(mean(b.total) + qt(0.975, df = kappa)*sd(b.total)/sqrt(kappa)*c(-1,1))
```

```
## [1] 0.05285991 0.05809862
```

```
print('Reallocations from ward:' )
```

```
## [1] "Reallocations from ward:"
```

```
wards = c('A', 'B', 'C', 'D', 'E', 'F')
for (i in 1:ncol(alloc.total)) {
  print(paste('Ward', wards[i]))
```

```
  print(mean(alloc.total[,i])+qt(0.975, df=kappa)*sd(alloc.total[,i])/sqrt(kappa)*c(-1,1))
}
```

```
## [1] "Ward A"
## [1] 153.7802 177.6198
## [1] "Ward B"
## [1] 317.7473 344.1527
## [1] "Ward C"
## [1] 547.521 591.279
## [1] "Ward D"
## [1] 688.5698 758.8302
## [1] "Ward E"
## [1] 403.4525 438.1475
```

```
print('Expected number of admission (1st year)')
```

```
## [1] "Expected number of admission (1st year)"
```

```
print(mean(e.admin) + qt(0.975, df = kappa)*sd(e.admin)/sqrt(kappa)*c(-1,1))
```

```
## [1] 15406.55 15515.55
```

5) Use the urgency points

We can now assess the relocation of the patients with respect to the urgency points. Specifically, we can assess the expected contribution to a cost based on weighted contributions from each ward.

```
apply(alloc.total, 2, mean)*urgency
```

```
## [1] 1159.90 1654.75 1138.80 7237.00 2104.00
```

We see that the contribution from ward D to the total urgency cost is much larger than the rest. This indicates that it would be a good idea to reallocate beds to ward D. Since patients are reallocated based on a probability matrix, the system is quite complex - it is therefore difficult to determine exactly what the optimal re-distribution would be. As shown earlier, simulated annealing would be a way of getting a estimate.

6) Looking into when ward F is included.

We now repeat when F is included. We use distribution of beds found by means of simulated annealing, i.e., something close to the optimal distribution of beds.

```
a.rates = c(14.5, 11.0, 8.0, 6.5, 5.0, 13.0)
s.rates = 1/c(2.9,4.0, 4.5, 1.4, 3.9, 2.2)
tm = t(matrix(c(0,0.05, 0.10, 0.05, 0.8, 0,
                0.2, 0, 0.5, 0.15, 0.15, 0,
                0.3, 0.2, 0, 0.2, 0.3, 0,
                0.35, 0.30, 0.05, 0, 0.3, 0,
                0.2, 0.1, 0.6, 0.1, 0, 0,
                0.2, 0.2, 0.2, 0.2, 0.2, 0),6,6))
beds = opt.bed
kappa = 20
b.total <- e.admin <- all.occ <- rep(NA, kappa)
alloc.total = matrix(NA, kappa, length(beds))
set.seed(69)
for (ll in 1:kappa) {
  a.times <- s.times <- type <- list()  ## arrival times / service times / types
  for (ii in 1:length(a.rates)) {
```

```r
    N = 1000
    a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
    while (tail(a.times[[ii]],1) < 365){
      N = N*2
      a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
    }
    a.times[[ii]] = a.times[[ii]][a.times[[ii]] <= 365]
    s.times[[ii]] = rexp(n = N, rate = s.rates[ii])
    type[[ii]] = rep(ii, length(a.times[[ii]]))
  }
  a.times = unlist(a.times)
  s.times = unlist(s.times)
  type = unlist(type)
  s.times = s.times[order(a.times)]
  type = type[order(a.times)]
  a.times = a.times[order(a.times)]

  sol = allocate.patients(beds, a.times, s.times, type, tm)

  all.occ[ll] = sum(apply(sol$status, 1, function(X) sum(X)==165)) / length(a.times)
  b.total[ll] = sum(sol$total.block) / length(a.times)
  alloc.total[ll, ] = apply(sol$re.alloc, 1, sum)
  e.admin[ll] = length(a.times) - sum(sol$total.block)
}

print('Probability that all beds are occupied:' )
```

```
## [1] "Probability that all beds are occupied:"
```

```r
print(mean(all.occ) + qt(0.975, df = kappa)*sd(all.occ)/sqrt(kappa)*c(-1,1))
```

```
## [1] 0.004469514 0.007549163
```

```r
print('Probability of not being admitted:' )
```

```
## [1] "Probability of not being admitted:"
```

```r
print(mean(b.total) + qt(0.975, df = kappa)*sd(b.total)/sqrt(kappa)*c(-1,1))
```

```
## [1] 0.2089025 0.2341639
```

```r
print('Reallocations from ward:' )
```

```
## [1] "Reallocations from ward:"
```

```r
wards = c('A', 'B', 'C', 'D', 'E', 'F')
for (i in 1:ncol(alloc.total)) {
  print(paste('Ward', wards[i]))
  print(mean(alloc.total[,i])+qt(0.975, df=kappa)*sd(alloc.total[,i])/sqrt(kappa)*c(-1,1))
}
```

```
## [1] "Ward A"
## [1] 56.09975 67.80025
## [1] "Ward B"
## [1] 438.5212 485.5788
## [1] "Ward C"
## [1] 1458.946 1496.154
```

```
## [1] "Ward D"
## [1] 131.4054 151.0946
## [1] "Ward E"
## [1] 674.6375 698.4625
## [1] "Ward F"
## [1] 334.3912 607.8088
```

```
print('Expected number of admission (1st year)')
```

```
## [1] "Expected number of admission (1st year)"
```

```
print(mean(e.admin) + qt(0.975, df = kappa)*sd(e.admin)/sqrt(kappa)*c(-1,1))
```

```
## [1] 16190.23 16672.57
```

We see a quite dramatic increase in probability of not being admitted at all. In fact, we go from a probability of $[0.053 \quad 0.058]$ to $[0.209 \quad 0.234]$. For this part, it does not make sense to assess the urgency points, since we have already used simulated annealing to optimize this and since we do not have the urgency for ward F.

Sensitivity analysis

7) length-of-stay distribution

We now test the system's sensitivity to the length-of-stay distribution by replacing the exponential distribution with the log-normal distribution. We test the new distribution by gradually increasing the variance. Specifically, we use

$$\sigma_i^2 = \frac{k}{\mu_i^2}, \quad k \in \{1, 2, ..., 5\}.$$

Since we modify the variance, we also need to re-compute the μ parameter, μ^*, to ensure the same mean, i.e.,

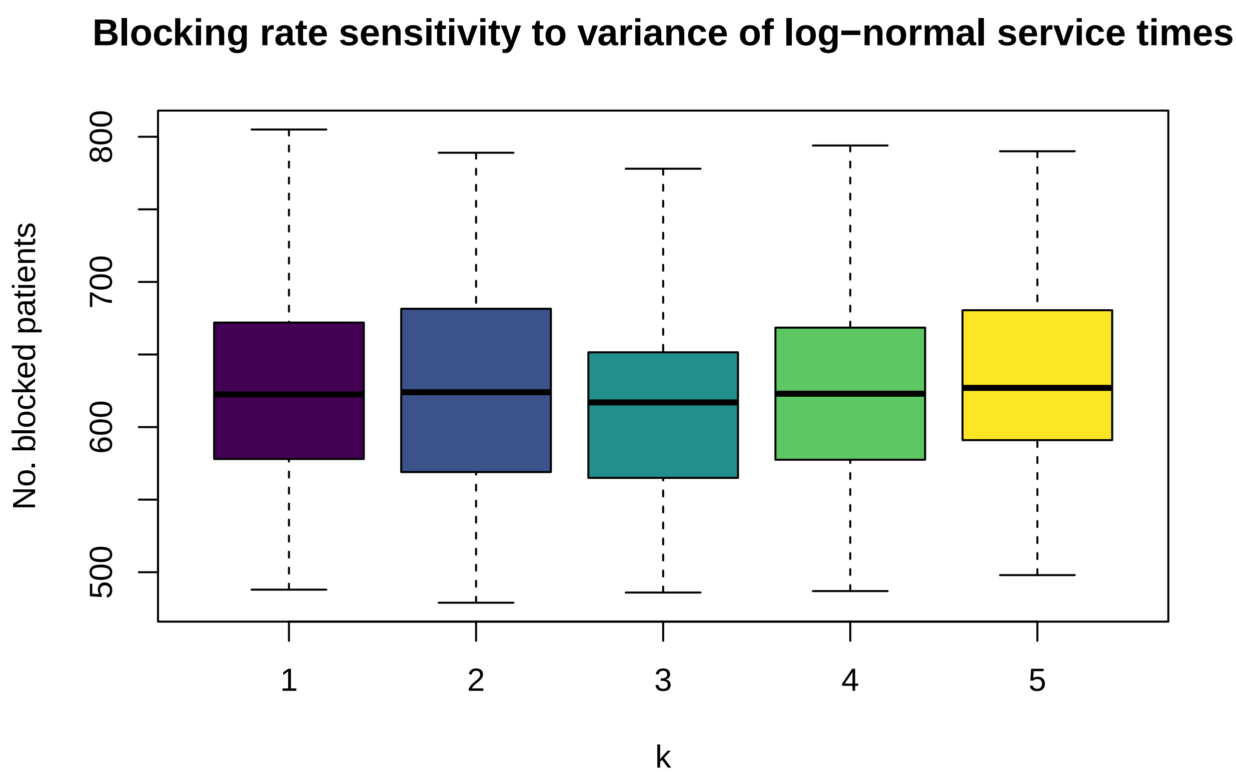
$$\mu_i^* = \log(1/\mu_i) - \frac{\sigma_i^2}{2}.$$

```
permutations = 1:5
if (file.exists('varSens.rds')){
  aux = readRDS('varSens.rds')
  beds.block.all = aux[[1]]
  choice.first.all = aux[[2]]
  beds.block.sum.all = aux[[3]]
  choice.first.sum.all = aux[[4]]
} else {
  beds.org = c(55,40,30,20,20)
  a.rates = c(14.5, 11.0, 8.0, 6.5, 5.0)
  s.rates = 1/c(2.9,4.0, 4.5, 1.4, 3.9)
  a.times <- s.times <- type <- list()  ## arrival times / types

  beds.block.allo = array(0, c(length(permutations), 100, length(beds.org)))
  beds.block.all = array(0, c(length(permutations), 100, length(beds.org)))
  beds.block.sum.all = array(0, c(length(permutations), 100, 1))
  choice.first.all = array(0, c(length(permutations), 100, length(beds.org)))
  choice.first.sum.all = array(0, c(length(permutations), 100, 1))
  set.seed(1)
  for (perm in permutations){
    print(sprintf("perm = %i", perm))
```

```r
  for (bi in 1:100){
    print(sprintf("     b = %i", bi))
    beds = beds.org

    beds.block.allo[perm, bi, ] = beds

    ii = 1
    a.times <- s.times <- type <- list()
    for (ii in 1:length(a.rates)) {
      N = 10000
      a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))

      while (tail(a.times[[ii]],1) < 365){
        N = N*2
        a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
      }
      a.times[[ii]] = a.times[[ii]][a.times[[ii]] <= 365]

      s.times[[ii]] = rlnorm(n = N, meanlog = log(1/s.rates[ii])-perm/(2*(1/s.rates[ii])^2),
                             sdlog = sqrt(perm/(1/s.rates[ii])^2))
      type[[ii]] = rep(ii, length(a.times[[ii]]))
    }


    ## unlist
    a.times = unlist(a.times)
    s.times = unlist(s.times)
    type = unlist(type)

    ## order
    s.times = s.times[order(a.times)]
    type = type[order(a.times)]
    a.times = a.times[order(a.times)]

    tm = t(matrix(c(0,0.05, 0.10, 0.05, 0.8,
                    0.2, 0, 0.5, 0.15, 0.15,
                    0.3, 0.2, 0, 0.2, 0.3,
                    0.35, 0.30, 0.05, 0, 0.3,
                    0.2, 0.1, 0.6, 0.1, 0),5,5))

    sol = allocate.patients(beds, a.times, s.times, type, tm)

    beds.block.all[perm, bi, ] = sol$total.block
    tmp_apply = sapply(1:length(beds), function(i) sum(type==i))-rowSums(sol$re.alloc)
    choice.first.all[perm, bi, ] = tmp_apply
    beds.block.sum.all[perm, bi, 1] = sum(sol$total.block)
    choice.first.sum.all[perm, bi, 1] = sum(choice.first.all[perm, bi, ])
  }
}

saveRDS(object = list(beds.block.all,
          choice.first.all,
          beds.block.sum.all,
```

```
            choice.first.sum.all),
          file = 'varSens.rds')
}
```

We can now assess the number of people not admitted to the hospital as a function of variance in service time.

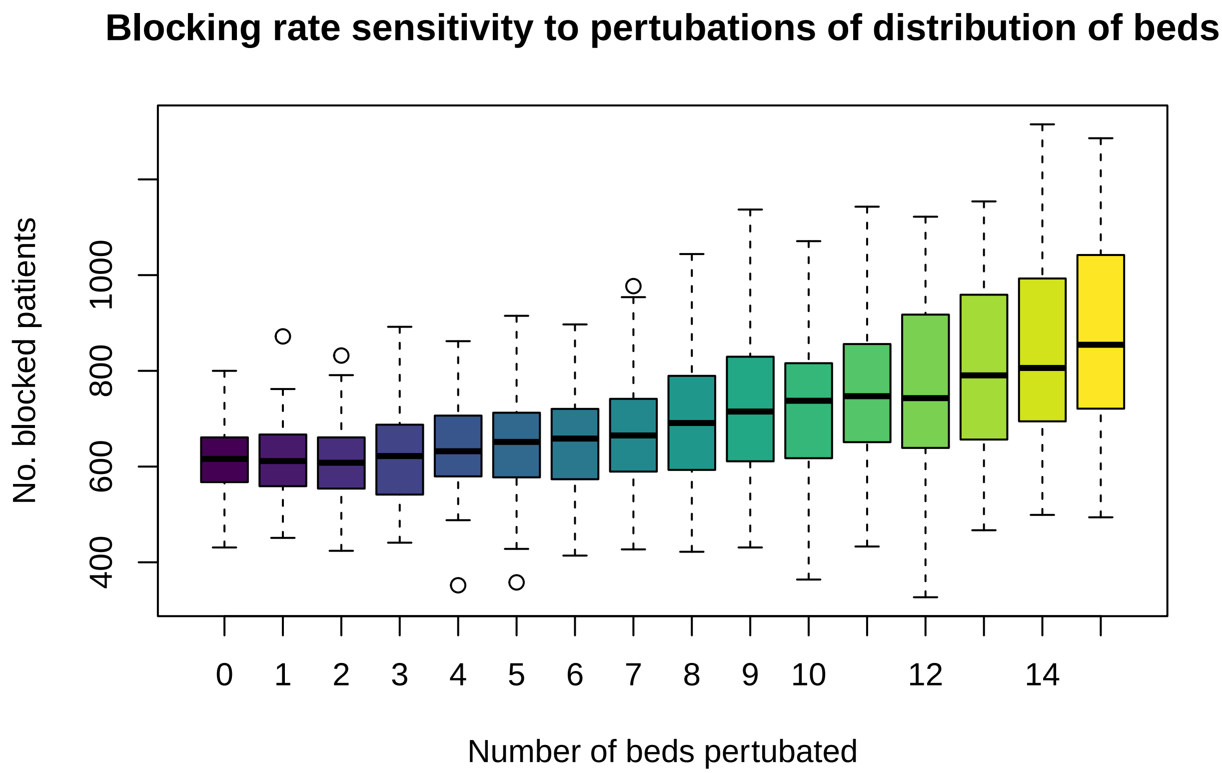


Overall, we see that the variance in the service time, i.e., time spent in hospital, does not have a large effect on the number of patients not admitted (deduced directly from the plot). We previously saw, with the introduction of Erlang-B, that the blocking probability is independent on the distribution of the service time (only depends on the mean service time). Therefore, it is not particularly surprising that we see no difference here.

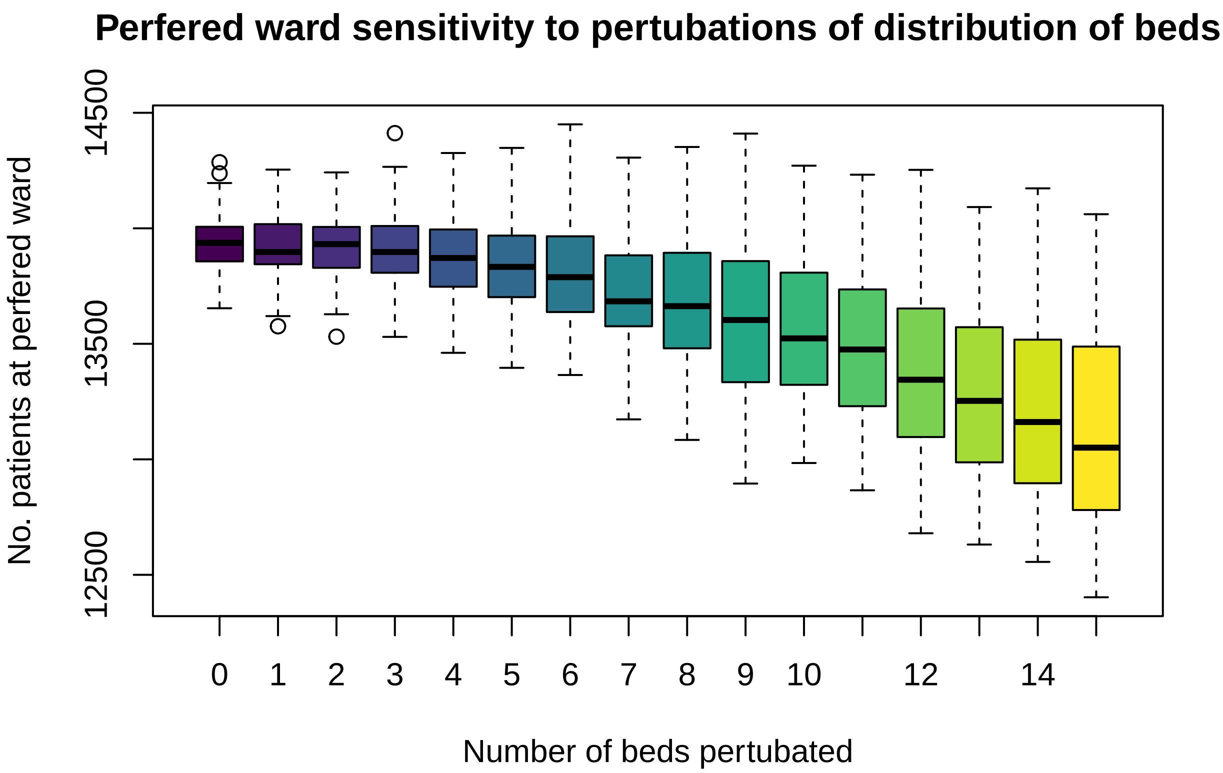
8) Sensitivity to the distribution of beds in the hospital

We will now look at how the total number of blocked patients variances as the distribution of beds is perturbed. For simplicity we will look at the system, without the new $F*$ ward. This will be done by redistribution between 0 and 15 beds between two wards in the hospital.

We start by simply looking at how the number of blocked patients varies as we increase the number of beds we pertubate in the distribution. A total of 1600 simulations from $t = 0$ to $t = 365$ were performed.



The same plot, but for the number of patients getting their preferred ward.



To quantify our observations we consider a standard analysis of variance for the model $b = \beta p^2 + \alpha p + c$, where b is the number of blocked patients and p is the size of the pertubations.

```
## Analysis of Variance Table
##
## Response: beds
##             Df   Sum Sq  Mean Sq F value    Pr(>F)
## perb         1 10434001 10434001  560.78 < 2.2e-16 ***
## Residuals 1598 29732755    18606
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The same model is considered for the number of patients getting their preferred ward ans a function of the size of the pertubation.

```
## Analysis of Variance Table
##
## Response: choice
##             Df    Sum Sq   Mean Sq F value    Pr(>F)
## perb         1 104930451 104930451  1379.5 < 2.2e-16 ***
## Residuals 1598 121549277     76063
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

For both models we see that both the first, second and intercept terms are significant, which gives that the number of patients who are blocked increases as size of the pertubations of the system increases, while the number of patients not getting their preferred ward decreases.

We can also test if there is a significant change in the variance as the size of the pertubation increases, for the number of blocked patients and number of patients getting their preferred ward.

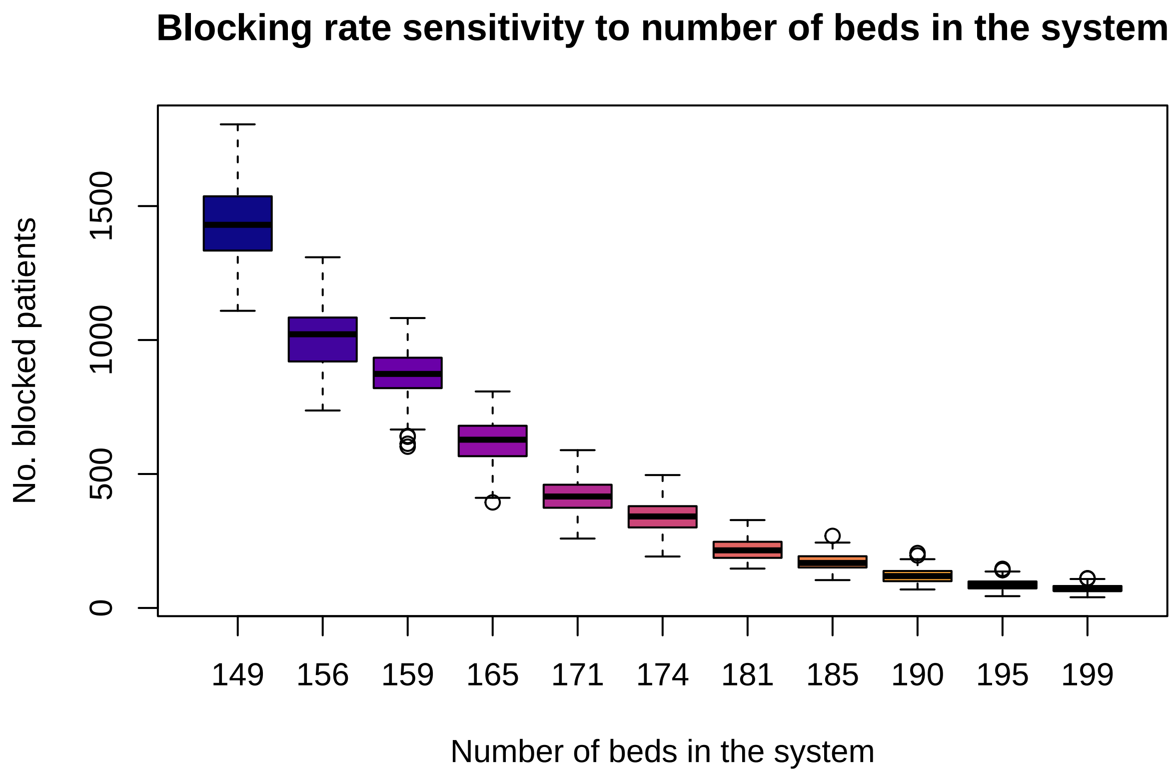
```
## [1] "Analysis for blocked patients"

## Analysis of Variance Table
##
## Response: beds.vars
##            Df     Sum Sq    Mean Sq F value    Pr(>F)
## perb.sizes  1 1769076764 1769076764  165.95 3.744e-09 ***
## Residuals  14  149245431   10660388
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

## [1] ""

## [1] "Analysis for preferred ward"

## Analysis of Variance Table
##
## Response: choice.vars
##            Df     Sum Sq    Mean Sq F value    Pr(>F)
## perb.sizes  1 3.2572e+10 3.2572e+10  321.54 4.697e-11 ***
## Residuals  14 1.4182e+09 1.0130e+08
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From where we conclude that there is a significant change in the variance as the size of the pertubation increases.

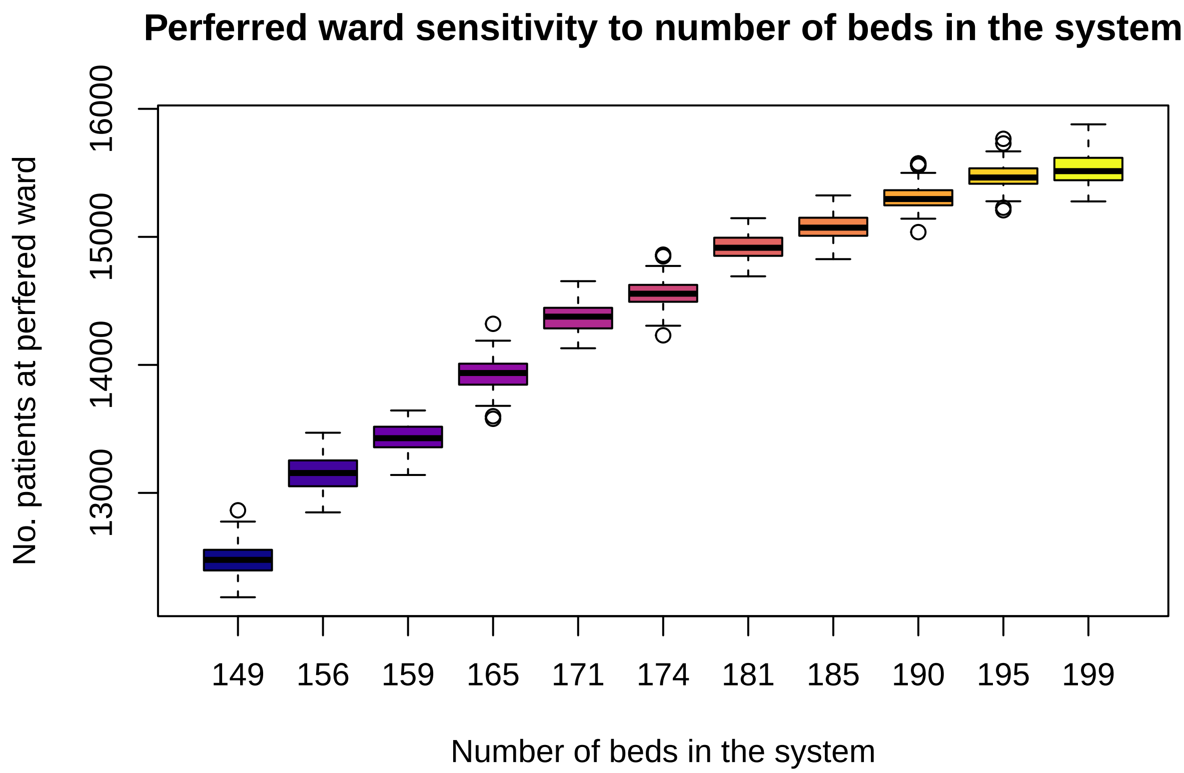
9) Changing the total number of beds in the system

We will now look at how total number of blocked patients variances as the total number of beds in the system is de- or increased. For simplicity we will look at the system, without the new $F*$ ward, and the number the beds will de distribution as close to the original distribution as integer round-off allows for. During our simulations we found that the mean number of patients during 1 year was ≈ 16425, so in a system without any blocking this the number of patients that should get their preferred ward.

We start by simply looking at how the number of blocked patients varies as function of the total number of beds in the system. A total of 100 simulations from $t = 0$ to $t = 365$ were performed for each size.

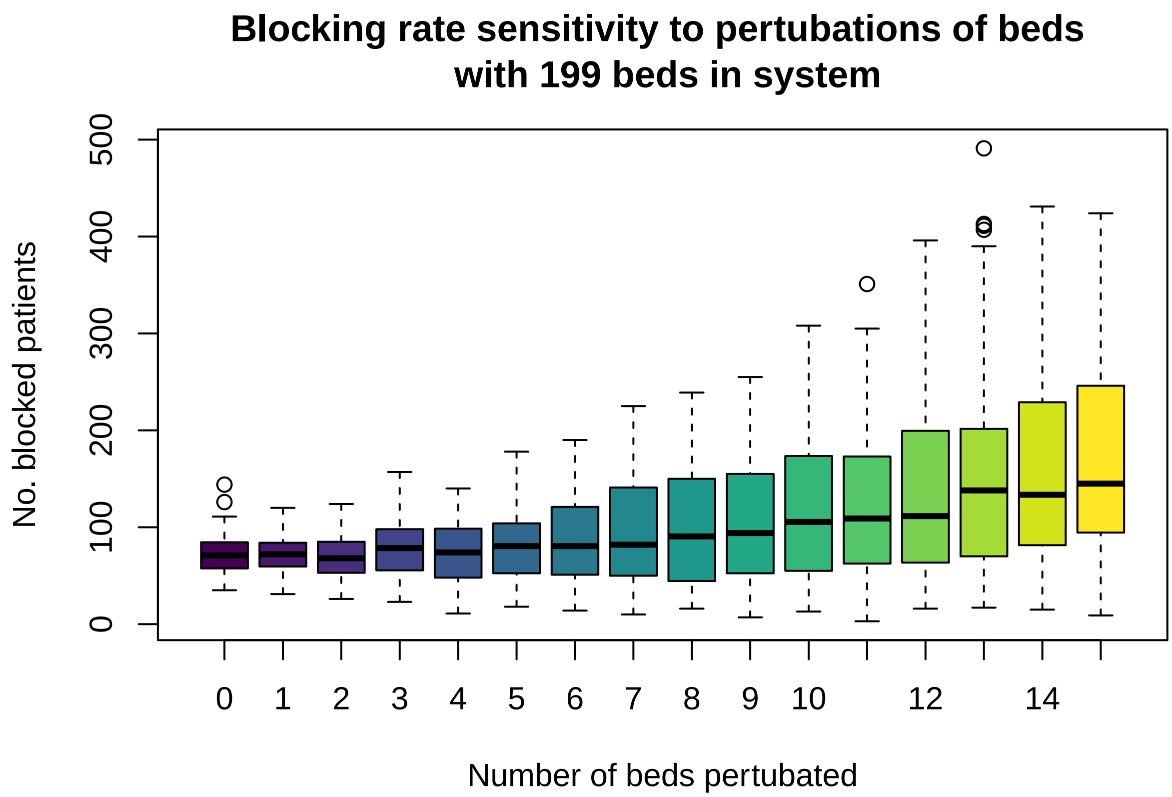


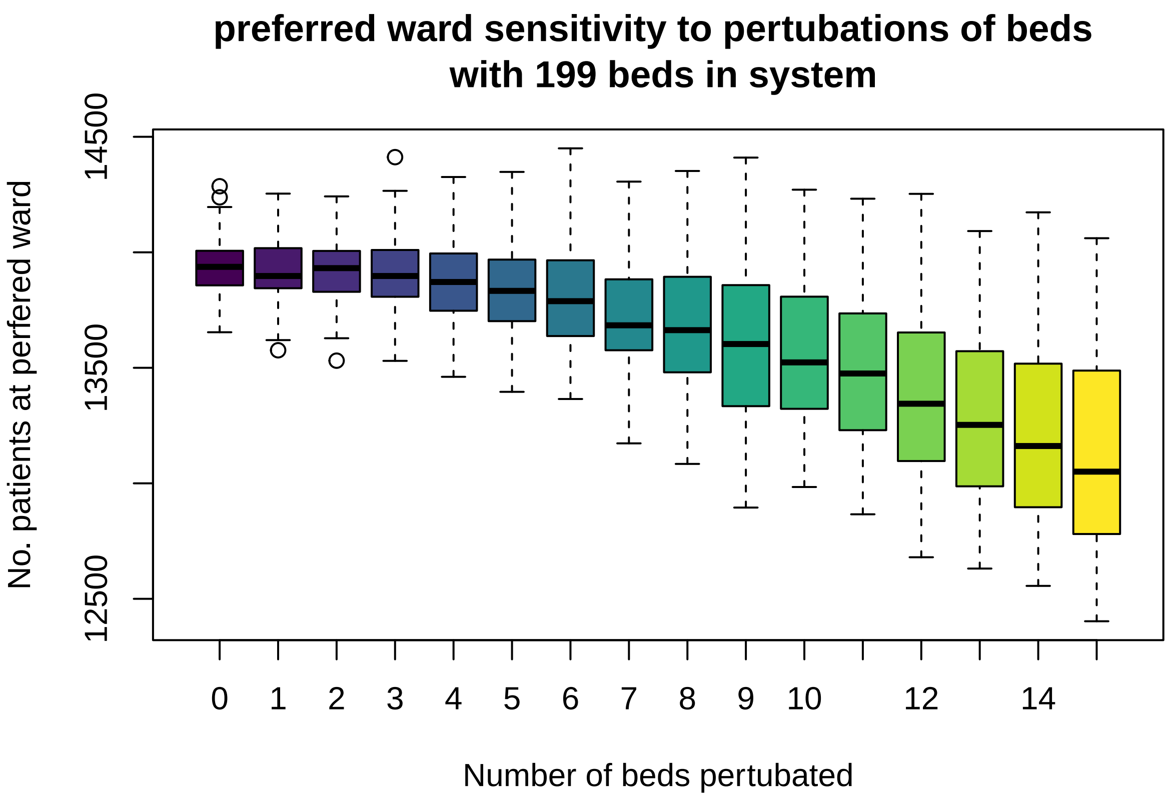
The same plot, but for the number of patients getting their preferred ward is made.



From these two plots becomes clear that there is the number of beds in the system affects the number of blocked patients, and the number of patients getting their preferred ward, which would also be expected.

Lets go back and investigate the sensitivity of a system with a larger number of total beds in the system. The same method as in the previous exercise is used, but now with a system with 199 total beds.


Blocking rate sensitivity to pertubations of beds
with 199 beds in system
No. blocked patients
0
100
200
300
400
500
0
1
2
3
4
5
6
7
8
9
10
12
14
Number of beds pertubated



We can now test if there is a significant difference in the change of mean and variance if the original system is pertubated versus the new system with 199 beds.

We compare the confidence interval of the parameters in the two linear model, i.e. the linear model for the normal system and the model for the system with 199 beds.

```
## [1] "Normal model"
```

```
##                 2.5 %    97.5 %
## (Intercept) 559.29208 584.83954
## perb         16.06706  18.96906
```

```
## [1] "Model with 199 beds"
```

```
##                 2.5 %    97.5 %
## (Intercept) 49.228743 61.500080
## perb         6.154447  7.548377
```

Thus we see that the pertubations to the system has a significantly smaller effect when the total number of beds is increased to 200 This makes perfect sense as we reject significantly fewer patients, hence it is much more likely that there is availability in the desired ward even when a number of beds have been relocated to another ward. One could also simply see this as testing whether a relatively large pertubation affects more than a relatively smaller pertubation. Thus we just confirm the idea that this is the case. Ideally we would have looked at the effect of the pertubations as a function of the number of beds to figure out when the system would be unaffected by changes of these sizes.

Code appendix

Code for *Sensitivity to the distribution of beds in the hospital*

```
beds.org = c(55,40,30,20,20)
a.rates = c(14.5, 11.0, 8.0, 6.5, 5.0)
s.rates = 1/c(2.9,4.0, 4.5, 1.4, 3.9)
a.times <- s.times <- type <- list()  ## arrival times / types

permutations = 1:16



beds.block.allo = array(0, c(length(permutations), 100, length(beds.org)))
beds.block.all = array(0, c(length(permutations), 100, length(beds.org)))
beds.block.sum.all = array(0, c(length(permutations), 100, 1))
choice.first.all = array(0, c(length(permutations), 100, length(beds.org)))
choice.first.sum.all = array(0, c(length(permutations), 100, 1))

set.seed(1)

for (perm in permutations){
print(sprintf("perm = %i", perm))
bi = 0
for (b in 1:5){
  for (bs1 in 1:length(beds.org)){
    for (bs2 in 1:length(beds.org)){
      if (bs1 != bs2) {
        bi = bi + 1
        print(sprintf("     b = %i", bi))
        beds = beds.org
        # permute beds
        idxs = sample(1:length(beds), size=2, replace = FALSE)
        beds[bs1] = beds[bs1] + (perm-1)
        beds[bs2] = beds[bs2] - (perm-1)

        beds.block.allo[perm, bi, ] = beds

        ii = 1
        a.times <- s.times <- type <- list()
        for (ii in 1:length(a.rates)) {
          N = 10000
          a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))

          while (tail(a.times[[ii]],1) < 365){
            N = N*2
            a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
          }
```

```r
    a.times[[ii]] = a.times[[ii]][a.times[[ii]] <= 365]

    s.times[[ii]] = rexp(n = N, rate = s.rates[ii])
    type[[ii]] = rep(ii, length(a.times[[ii]]))
  }


  ## unlist
  a.times = unlist(a.times)
  s.times = unlist(s.times)
  type = unlist(type)

  ## order
  s.times = s.times[order(a.times)]
  type = type[order(a.times)]
  a.times = a.times[order(a.times)]


  #### Ex 1 ####
  ## trans-matrix
  tm = t(matrix(c(0,0.05, 0.10, 0.05, 0.8,
                  0.2, 0, 0.5, 0.15, 0.15,
                  0.3, 0.2, 0, 0.2, 0.3,
                  0.35, 0.30, 0.05, 0, 0.3,
                  0.2, 0.1, 0.6, 0.1, 0),5,5))

  in.service = lapply(beds, function(b) rep(0, b))

  status = matrix(NA, length(a.times), length(in.service))
  total.block = rep(0, length(in.service))
  total.first.choice = rep(0, length(in.service))
  re.alloc = matrix(0, length(in.service), length(in.service))
  ii = 1
  for (ii in 1:length(a.times)) {
    if (all(a.times[ii] < in.service[[type[ii]]])){
      try.type = sample(1:length(in.service), 1, prob = tm[type[ii], ])
      if (all(a.times[ii] < in.service[[try.type]])){
        ## no free beds :(
        total.block[type[ii]] = total.block[type[ii]] + 1
      }
      else {
        in.service[[ try.type ]][ which.min( in.service[[ try.type ]] >= a.times[ii] ) ] = a.time
        re.alloc[type[ii], try.type] = re.alloc[type[ii], try.type] + 1
      }
    }
    else {
      in.service[[type[ii]]][ which.min( in.service[[type[ii]]] >= a.times[ii] ) ] = a.times[ii]+
      total.first.choice[type[ii]] = total.first.choice[type[ii]] + 1
    }

    status[ii, ] = sapply(1:length(in.service), function(jj) sum(in.service[[jj]]>= a.times[ii]))
  }
```

```
        beds.block.all[perm, bi, ] = total.block
        choice.first.all[perm, bi, ] = total.first.choice
        beds.block.sum.all[perm, bi, 1] = sum(total.block)
        choice.first.sum.all[perm, bi, 1] = sum(total.first.choice)
      }
    }
  }


}
}

# Restore the object
saveRDS(beds.block.all, file = "bedsSens-beds-blocked.rds")
saveRDS(choice.first.all, file = "bedsSens-choice-first-all.rds")
saveRDS(beds.block.allo, file = "bedsSens-beds-allo.rds")
saveRDS(beds.block.sum.all, file = "bedsSens-beds-blocked-sum.rds")
saveRDS(choice.first.sum.all, file = "bedsSens-choice-first-sum.rds")
```

Code for *Changing the total number of beds in the system*

```
# Main task
beds.org = c(55,40,30,20,20)
total.beds = sum(beds.org)
a.rates = c(14.5, 11.0, 8.0, 6.5, 5.0)
s.rates = 1/c(2.9,4.0, 4.5, 1.4, 3.9)
a.times <- s.times <- type <- list()  ## arrival times / types

ss = seq(150,200,5)
sizes = matrix(0, nrow = (length(ss)), ncol=length(beds.org))
for (s in 1:(length(ss))) {
  sizes[s, ] = round(beds.org/total.beds * ss[s])
}
ss = rowSums(sizes)


beds.block.allo = array(0, c(length(ss), 100, length(beds.org)))
beds.block.all = array(0, c(length(ss), 100, length(beds.org)))
beds.block.sum.all = array(0, c(length(ss), 100, 1))
choice.first.all = array(0, c(length(ss), 100, length(beds.org)))
choice.first.sum.all = array(0, c(length(ss), 100, 1))

set.seed(1)

for (j in 1:length(ss)){
print(sprintf("size = %i", j))
for (bi in 1:100){
  print(sprintf("     b = %i", bi))
  beds = sizes[j,]

  beds.block.allo[j, bi, ] = beds

  ii = 1
```

```r
a.times <- s.times <- type <- list()
for (ii in 1:length(a.rates)) {
  N = 10000
  a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))

  while (tail(a.times[[ii]],1) < 365){
    N = N*2
    a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
  }
  a.times[[ii]] = a.times[[ii]][a.times[[ii]] <= 365]

  s.times[[ii]] = rexp(n = N, rate = s.rates[ii])
  type[[ii]] = rep(ii, length(a.times[[ii]]))
}


## unlist
a.times = unlist(a.times)
s.times = unlist(s.times)
type = unlist(type)

## order
s.times = s.times[order(a.times)]
type = type[order(a.times)]
a.times = a.times[order(a.times)]


#### Ex 1 ####
## trans-matrix
tm = t(matrix(c(0,0.05, 0.10, 0.05, 0.8,
                0.2, 0, 0.5, 0.15, 0.15,
                0.3, 0.2, 0, 0.2, 0.3,
                0.35, 0.30, 0.05, 0, 0.3,
                0.2, 0.1, 0.6, 0.1, 0),5,5))

in.service = lapply(beds, function(b) rep(0, b))

status = matrix(NA, length(a.times), length(in.service))
total.block = rep(0, length(in.service))
total.first.choice = rep(0, length(in.service))
re.alloc = matrix(0, length(in.service), length(in.service))
ii = 1
for (ii in 1:length(a.times)) {
  if (all(a.times[ii] < in.service[[type[ii]]])){
    try.type = sample(1:length(in.service), 1, prob = tm[type[ii], ])
    if (all(a.times[ii] < in.service[[try.type]])){
      ## no free beds :(
      total.block[type[ii]] = total.block[type[ii]] + 1
    }
    else {
      in.service[[ try.type ]][ which.min( in.service[[ try.type ]] >= a.times[ii] ) ] = a.times[ii]+s
      re.alloc[type[ii], try.type] = re.alloc[type[ii], try.type] + 1
    }
```

```
    }
    else {
      in.service[[type[ii]]][ which.min( in.service[[type[ii]]] >= a.times[ii] ) ] = a.times[ii]+s.times
      total.first.choice[type[ii]] = total.first.choice[type[ii]] + 1
    }

    status[ii, ] = sapply(1:length(in.service), function(jj) sum(in.service[[jj]]>= a.times[ii]))
  }

  beds.block.all[j, bi, ] = total.block
  choice.first.all[j, bi, ] = total.first.choice
  beds.block.sum.all[j, bi, 1] = sum(total.block)
  choice.first.sum.all[j, bi, 1] = sum(total.first.choice)
  }
}

# Restore the object
saveRDS(beds.block.all, file = "sizeSens-beds-blocked.rds")
saveRDS(choice.first.all, file = "sizeSens-choice-first-all.rds")
saveRDS(beds.block.allo, file = "sizeSens-beds-allo.rds")
saveRDS(beds.block.sum.all, file = "sizeSens-beds-blocked-sum.rds")
saveRDS(choice.first.sum.all, file = "sizeSens-choice-first-sum.rds")
```

```
beds.org = c(67,48,36,24,24)
a.rates = c(14.5, 11.0, 8.0, 6.5, 5.0)
s.rates = 1/c(2.9,4.0, 4.5, 1.4, 3.9)
a.times <- s.times <- type <- list()  ## arrival times / types

permutations = 1:16



beds.block.allo = array(0, c(length(permutations), 100, length(beds.org)))
beds.block.all = array(0, c(length(permutations), 100, length(beds.org)))
beds.block.sum.all = array(0, c(length(permutations), 100, 1))
choice.first.all = array(0, c(length(permutations), 100, length(beds.org)))
choice.first.sum.all = array(0, c(length(permutations), 100, 1))

set.seed(1)

for (perm in permutations){
  print(sprintf("perm = %i", perm))
  bi = 0
  for (b in 1:5){
    for (bs1 in 1:length(beds.org)){
      for (bs2 in 1:length(beds.org)){
        if (bs1 != bs2) {
          bi = bi + 1
          print(sprintf("     b = %i", bi))
          beds = beds.org
          # permute beds
          idxs = sample(1:length(beds), size=2, replace = FALSE)
          beds[bs1] = beds[bs1] + (perm-1)
          beds[bs2] = beds[bs2] - (perm-1)
```

```
beds.block.allo[perm, bi, ] = beds

ii = 1
a.times <- s.times <- type <- list()
for (ii in 1:length(a.rates)) {
  N = 10000
  a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))

  while (tail(a.times[[ii]],1) < 365){
    N = N*2
    a.times[[ii]] = cumsum(rexp(n = N, rate = a.rates[ii]))
  }
  a.times[[ii]] = a.times[[ii]][a.times[[ii]] <= 365]

  s.times[[ii]] = rexp(n = N, rate = s.rates[ii])
  type[[ii]] = rep(ii, length(a.times[[ii]]))
}


## unlist
a.times = unlist(a.times)
s.times = unlist(s.times)
type = unlist(type)

## order
s.times = s.times[order(a.times)]
type = type[order(a.times)]
a.times = a.times[order(a.times)]


#### Ex 1 ####
## trans-matrix
tm = t(matrix(c(0,0.05, 0.10, 0.05, 0.8,
                0.2, 0, 0.5, 0.15, 0.15,
                0.3, 0.2, 0, 0.2, 0.3,
                0.35, 0.30, 0.05, 0, 0.3,
                0.2, 0.1, 0.6, 0.1, 0),5,5))

in.service = lapply(beds, function(b) rep(0, b))

status = matrix(NA, length(a.times), length(in.service))
total.block = rep(0, length(in.service))
total.first.choice = rep(0, length(in.service))
re.alloc = matrix(0, length(in.service), length(in.service))
ii = 1
for (ii in 1:length(a.times)) {
  if (all(a.times[ii] < in.service[[type[ii]]])){
    try.type = sample(1:length(in.service), 1, prob = tm[type[ii], ])
    if (all(a.times[ii] < in.service[[try.type]])){
      ## no free beds :(
      total.block[type[ii]] = total.block[type[ii]] + 1
    }
    else {
```

```
                in.service[[ try.type ]][ which.min( in.service[[ try.type ]] >= a.times[ii] ) ] = a.ti
                re.alloc[type[ii], try.type] = re.alloc[type[ii], try.type] + 1
              }
            }
            else {
              in.service[[type[ii]]][ which.min( in.service[[type[ii]]] >= a.times[ii] ) ] = a.times[ii
              total.first.choice[type[ii]] = total.first.choice[type[ii]] + 1
            }

            status[ii, ] = sapply(1:length(in.service), function(jj) sum(in.service[[jj]]>= a.times[ii]
          }

          beds.block.all[perm, bi, ] = total.block
          choice.first.all[perm, bi, ] = total.first.choice
          beds.block.sum.all[perm, bi, 1] = sum(total.block)
          choice.first.sum.all[perm, bi, 1] = sum(total.first.choice)
        }
      }
    }


  }
}

# Restore the object
saveRDS(beds.block.all, file = "bedsSens-200-beds-blocked.rds")
saveRDS(choice.first.all, file = "bedsSens-200-choice-first-all.rds")
saveRDS(beds.block.allo, file = "bedsSens-200-beds-allo.rds")
saveRDS(beds.block.sum.all, file = "bedsSens-200-beds-blocked-sum.rds")
saveRDS(choice.first.sum.all, file = "bedsSens-200-choice-first-sum.rds")
```