Examples of using QueuesFunction

Sunday, October 30, 2016

This file contains examples showing how we can use our function Queues_Function for different simulation studies

Example 1

Suppose we want to compare two Queing Systems, one of them with uniform distributions and the second one with exponential ones. In this example we want to keep 'nonlinear correlation structure' (use the inverse transorfm method we discussed on Friday). Our Queueing Systems consist of 6 queues each. Serving times in the first system follow $U(0,6)$, $U(1,7)$, $U(2,10)$, $U(0,2)$, $U(0,2)$ and $U(0,4)$. respectively, while in the second system they folow $Exp\left(\frac{1}{3}\right)$, $Exp\left(\frac{1}{4}\right)$, $Exp\left(\frac{1}{6}\right)$,$Exp(1)$, $Exp(1)$ and $Exp\left(\frac{1}{2}\right)$, since we want to have comparable means.

```
source('Queues_Function.R')
# fix T and lambda
T <- 100
lambda <- 1

# fix also parameters min and max (of uniform dist.) for the 3 queues:
# in or case unif(0,6), unif(1,7), unif(2,9)
param_unif_df <- data.frame(min = c(0, 1, 2, 0, 0, 0),
                            max = c(6, 7, 10, 2, 2, 4))

# now fix parameters of corresponding parameters for the exponential distributions
# normally we want to have the same mean values of the corresponding queues
param_exp <- sapply(1:nrow(param_unif_df), function(i){
  2/(param_unif_df$max[i] + param_unif_df$min[i])
})

# create a sample from the Poisson process
set.seed(13)
n <- rpois(1, lambda*T)

# remember to make sure n > 0, otherwise there would be no patients
n
```

```
## [1] 105
```

```
vector_arrival_times <- sort(c(0, runif(n - 1, min = 0, max = T)))
```

Now we create a matrix of $U(0,1)$ random variables and we later transform them to 'our' uniform distributions (on different intervals). We do an analogous thing to obtain corresponding exponentially distributed serving times, but here we use the following transformation

$$F^{-1}(u) = -\frac{\log(1-u)}{\lambda}$$

```r
# create a matrix of unif(0,1) random variables
temp_df <- matrix(runif(n*nrow(param_unif_df)), nrow=n)
serving_times_unif <- sapply(seq_len(ncol(temp_df)), function(i){
  param_unif_df$min[i] + temp_df[,i]*(param_unif_df$max[i] - param_unif_df$min[i])
})

serving_times_exp <- sapply(seq_len(ncol(temp_df)), function(i){
  - log(1-temp_df[,i])/param_exp[i]
  })

# comparison of means
colMeans(serving_times_unif)
```

```
## [1] 3.0513373 3.6898595 6.1459105 0.9457434 0.9803129 2.0468782
```

```r
colMeans(serving_times_exp)
```

```
## [1] 3.171505 3.342612 6.908679 0.907033 1.007503 2.244448
```

```r
# performing QueuesFunction on serving_times_unif and serving_times_exp
queue_unif <- QueuesFunction(serving_times_unif, vector_arrival_times)
queue_exp <- QueuesFunction(serving_times_exp, vector_arrival_times)
```

Example 2

Suppose we cannot use inverse transform method, so we don't care about using the same u's from $U(0,1)$. However, we want to have a 6-element queue consisting of serving times with 6 different distributions.

```r
set.seed(13)
T <- 50
lambda <- 1
# create a sample from the Poisson process
n <- rpois(1, lambda*T)
n
```

```
## [1] 53
```

```r
vector_arrival_times <- sort(c(0, runif(n - 1, min = 0, max = T)))
serving_times_different <- as.data.frame(cbind(V1 = abs(rcauchy(n, 0, 1)),
                                    V2 = runif(n, 3, 4),
                                    V3 = abs(rnorm(n, 0, 1)),
                                    V4 = rgamma(n, 5, 6),
                                    V5 = rgamma(n, 7, 100),
                                    V6 = abs(rt(n, 1))))

queue_different <- QueuesFunction(serving_times_different, vector_arrival_times)
summary(queue_different)
```

```
##                Length Class      Mode
## queues_list    6      -none-     list
## leaving_times 7      data.frame list
```

One of the ideas discussed on Friday was measuring correlation between the waiting time in subsequent queues. Technically, the output of our function contains 'waiting time + time of being served'. Hence, we can do the following thing:

```
leaving_times <- queue_different$leaving_times
waiting_times <- leaving_times[,2:ncol(leaving_times)] - leaving_times[,1:(ncol(leaving_times)-1)] - se

cor(waiting_times)
```

```
##                      Leaving_queue1_time Leaving_queue2_time
## Leaving_queue1_time          1.00000000         0.640391367
## Leaving_queue2_time          0.64039137         1.000000000
## Leaving_queue3_time          0.01433385         0.125590597
## Leaving_queue4_time         -0.10516491        -0.002236175
## Leaving_queue5_time         -0.03314208        -0.007474408
## Leaving_queue6_time          0.64213463         0.748065336
##                      Leaving_queue3_time Leaving_queue4_time
## Leaving_queue1_time          0.01433385        -0.105164905
## Leaving_queue2_time          0.12559060        -0.002236175
## Leaving_queue3_time          1.00000000        -0.040285691
## Leaving_queue4_time         -0.04028569         1.000000000
## Leaving_queue5_time         -0.34720197        -0.020064300
## Leaving_queue6_time          0.05173229        -0.009030629
##                      Leaving_queue5_time Leaving_queue6_time
## Leaving_queue1_time        -0.033142079         0.642134628
## Leaving_queue2_time        -0.007474408         0.748065336
## Leaving_queue3_time        -0.347201968         0.051732285
## Leaving_queue4_time        -0.020064300        -0.009030629
## Leaving_queue5_time         1.000000000         0.047044295
## Leaving_queue6_time         0.047044295         1.000000000
```

Example 3

The simplest example, in which all serving times are deterministic

```
set.seed(13)
T <- 50
lambda <- 1
# create a sample from the Poisson process
n <- rpois(1, lambda*T)

# serving times in subsequent queues: 10, 20, 30, 20, 40, 5
serving_times_deterministic <- as.data.frame(matrix(rep(c(10, 20, 30, 100, 40, 5), times = n),
                                                    ncol=6, byrow=TRUE))

queue_deterministic <- QueuesFunction(serving_times_deterministic, vector_arrival_times)
leaving_times <- queue_deterministic$leaving_times
waiting_times <- leaving_times[,2:ncol(leaving_times)] - leaving_times[,1:(ncol(leaving_times)-1)] -
serving_times_deterministic

colMeans(waiting_times)
```

```
## Leaving_queue1_time Leaving_queue2_time Leaving_queue3_time
##            234.8653            260.0000            260.0000
## Leaving_queue4_time Leaving_queue5_time Leaving_queue6_time
##           1820.0000              0.0000              0.0000
```

Example 4

Suppose we want our serving times to follow some heavy tailed distribution. In this example we use Pareto distribution https://en.wikipedia.org/wiki/Pareto_distribution with $x_m = 1$ and $\alpha = 1.5$. Its p.d.f. is given by:

$$f(x) = \frac{3}{2}x^{-2.5}$$

This distribution has expected value equal to 3. If we want to use a distribution with another mean, by multiply our variable by an appropriate number.
Similarly to Example 1, we use inverse transform method to obtain Pareto distributed samples from uniform distribution.

```
set.seed(13)
T <- 50
lambda <- 1
means_pareto <- c(1,2,3,4,5,3)

n <- rpois(1, lambda*T)

# remember to make sure n > 0, otherwise there would be no patients
n
```

```
## [1] 53
```

```
vector_arrival_times <- sort(c(0, runif(n - 1, min = 0, max = T)))
temp_df <- matrix(runif(n*length(means_pareto)), nrow=n)

serving_times_pareto <- sapply(seq_len(ncol(temp_df)), function(i){
  (1/(1-temp_df[,i])^(2/3))*means_pareto[i]/3
  })

queue_pareto <- QueuesFunction(serving_times_pareto, vector_arrival_times)
```