Computer Exercises 1-8

Mads Esben Hansen (s174434) & Gustav Als (s184400)

6/13/2022

Computer exercise 1

1

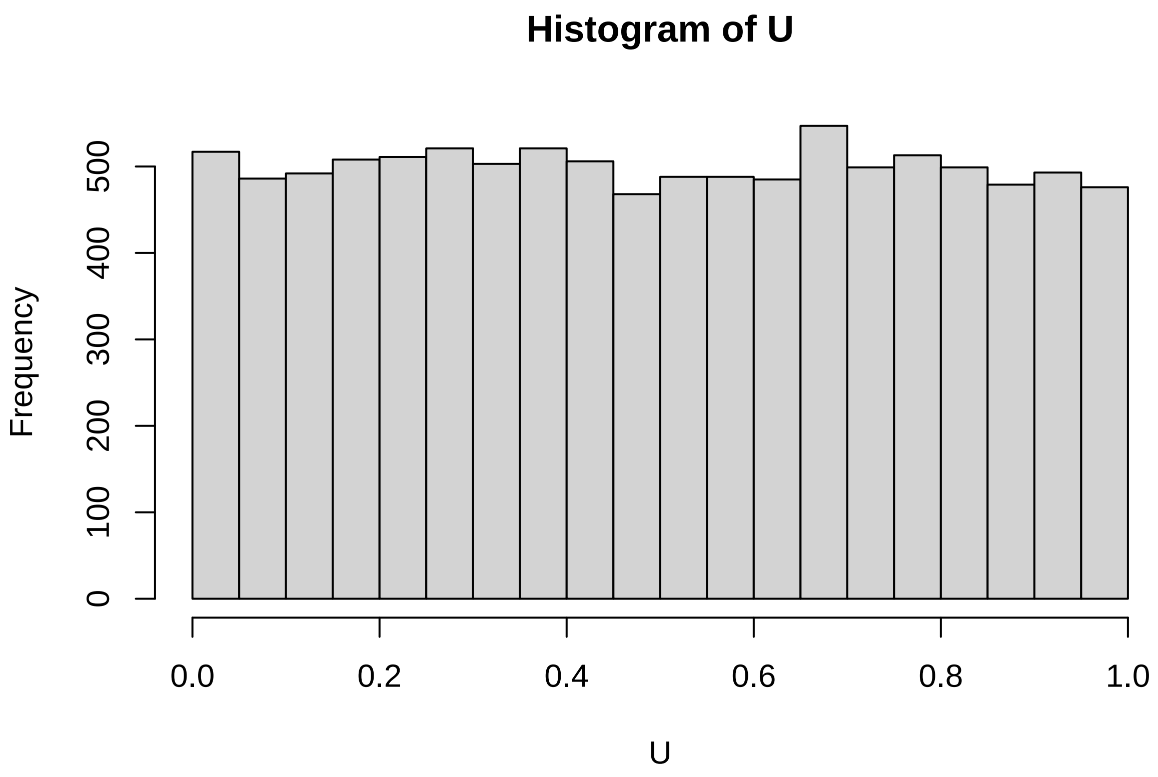
In this exercise, we wish to genarate uniformly distributed random numbers, $U_i \sim U(0,1)$. To this end, we start by defining a linear congruential generator:

```
LCG <- function(X,a,c,M,N) {
  sol = integer(N)
  stopifnot(all(sapply(c(X,a,c,M), function(x) x==round(x,0) )))
  sol[1] = X
  for (ii in 2:N) {
    sol[ii] = (a*sol[ii-1]+c) %% M
  }
  return ( sol/M )
}
```

a)

We now want to evaluate our pseudo-random variabel. We start by visual inspection, therefore we make a histogram:

```
N = 10000
M = 8388607      # 2^29 - 1
c = 1234568
a = 2047   # 2^11 - 1
X = as.integer(runif(1, 0, M))
U = LCG(X,a,c,M,N)

# Hist
hist(U, breaks = 20, xlab = "U")
```

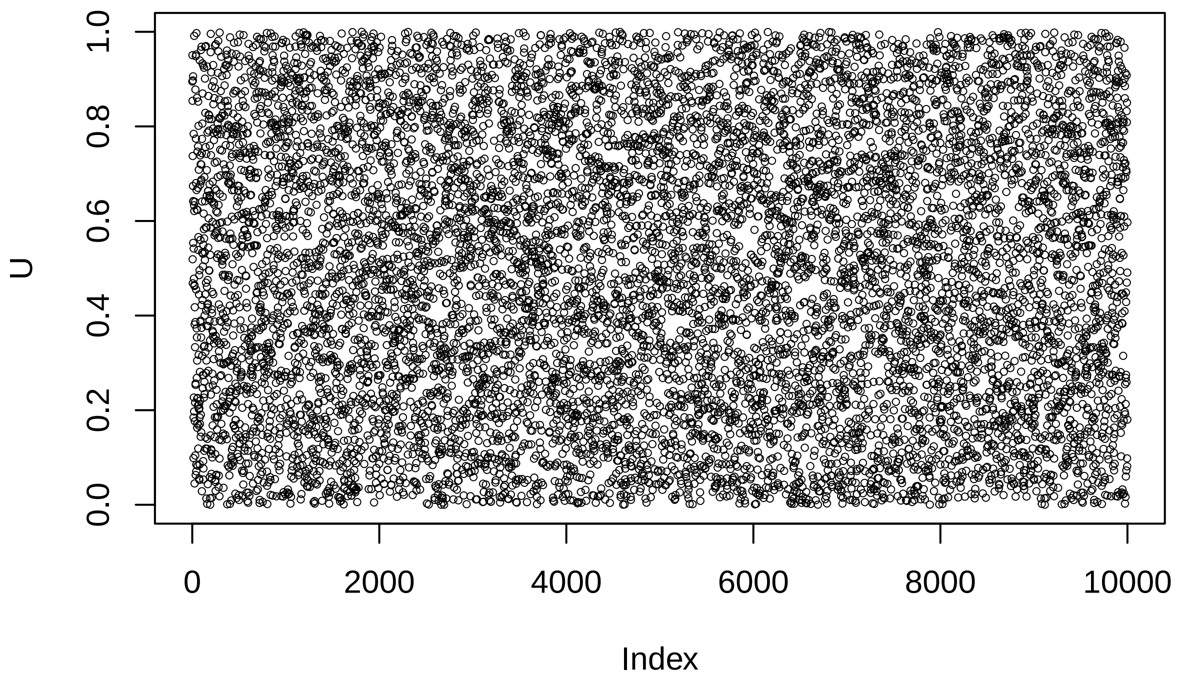


By the looks of it, its looks good. However, it is quite hard to determine exactly what we would expect, since we humans have a difficult time comprehending the effect of number of draws on the confidence intervals.

b)

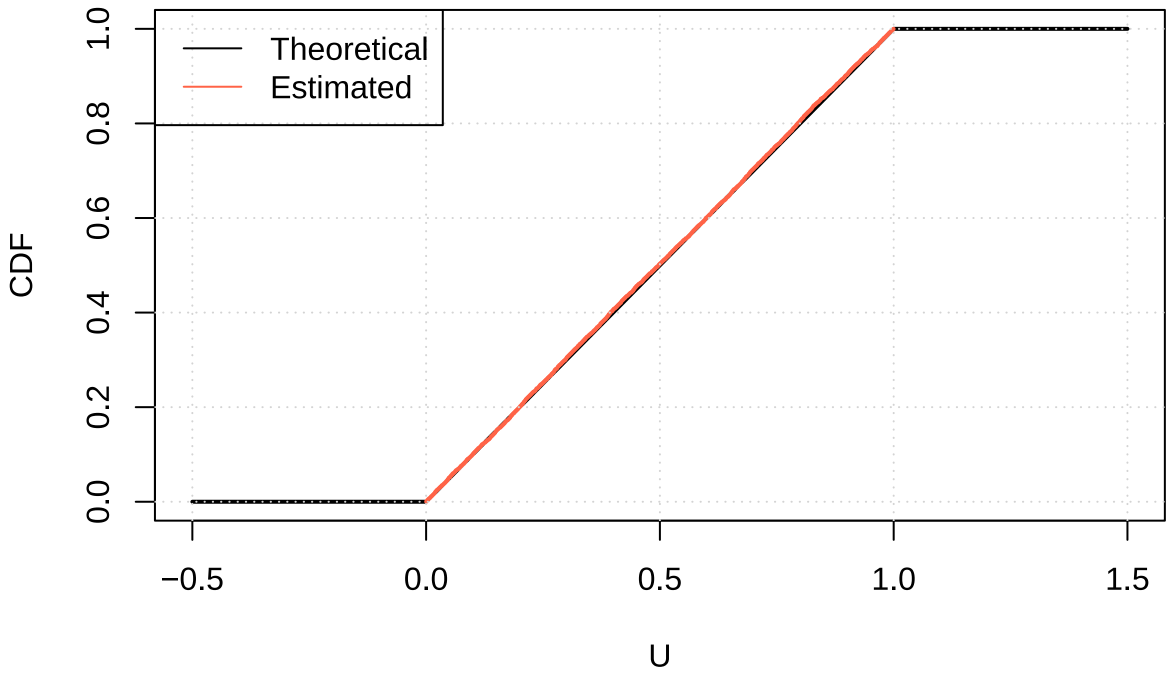
We now want to extend the quality control of our pseudo-random number generator. We start by looking at a simple scatter plot:

```r
plot(U, xlab = 'Index', ylab = 'U', cex = .5, lwd = .5)
```



As expected, the number all lie in the interval (0,1) with what seems to be equal distribution. We now want to compare the empirical cumulative distribution function (ECDF) with the theoretical cumulative distribution function (CDF):

```
ECDF <- function(X, na.rm = F){
  x.min = min(X,na.rm = na.rm)
  x.max = max(X,na.rm = na.rm)
  x = seq(x.min,x.max, length.out = length(X))
  list(
  x = x,
  y = sapply(x, function(x) sum(X<=x)) / length(X))
}
CDF.unif <- function(x){
  ifelse(x<0,0,ifelse(x<1,x,1))
}

e.cdf = ECDF(U)

plot(seq(-.5,1.5,length.out=N), CDF.unif(seq(-.5,1.5,length.out=N)), type = 'l', xlab = "U", ylab = 'CDI
lines(e.cdf$x, e.cdf$y, col = 'tomato', lwd = 2)
grid()
legend('topleft', c('Theoretical', 'Estimated'),
       col = c('black','tomato'), lty = 1)
```



We now perform a χ^2-test, as described in the lecture slides. We split random variable into bins and we can now model it as a multinomial distribution. Each bin will have X_j observed members and from the theoretical distribution we can calculate the expected number of members in each bin, np_j. We then divide by the standard deviation to normalize. By the central limit theorem we have:

$$\frac{X_j - np_j}{\sqrt{np_j(1-p_j)}} \sim N(0,1)$$

We can now square this, and use a usual χ^2-test, i.e.,

$$\frac{(X_j - np_j)^2}{np_j(1-p_j)} \sim \chi(1)$$

For all bins this gives the test

$$\sum_{j=1}^{k} \frac{(X_j - np_j)^2}{np_j} \sim \chi(k-1)$$

```r
nBin <- function(X,bins){
  aux.seq = seq(0,1,length.out = (bins+1))
  sapply(1:bins, function(i) sum( X>=aux.seq[i] & X<aux.seq[i+1]) )
}
bins = 20
p = 1/bins
eps = nBin(U,bins) - N*p
```

```r
chi.vals = (eps)^2/(N*p*(1-p))

## total p value
print(1 - pchisq( sum(eps^2/(p*N)), bins - 1 ))
```

```
## [1] 0.8235066
```

We get a p-value of 0.14, i.e., according to this test, there is a 14% chance of seeing less uniform numbers given that the underlying distribution is in fact uniform. In other words, we cannot reject that U does follow a uniform distribution.

We now proceed to the Kolmogorov-Smirnov (KS) test. The KS test compared the ECDF with the CDF.

```r
ks.test(U, 'punif', 0, 1)
```

```
## Warning in ks.test(U, "punif", 0, 1): ties should not be present for the
## Kolmogorov-Smirnov test
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  U
## D = 0.0087149, p-value = 0.4333
## alternative hypothesis: two-sided
```

We get a p-value of 0.65, i.e., according to this test, there is a 65% chance of seeing less uniform numbers given that the underlying distribution is in fact uniform. In other words, we cannot reject that U does follow a uniform distribution. Notice that the KS test complains that ties should not exist. This is becasue we have some numbers which are exactly the same - an unfortunate property of our limited memory pseudo-random number generator.

c)

We now proceed to run tests. We start by performing the above/below test. The idea of the test is to look at the number of draws above/below the median. Let D_a denote the number of draws above, and D_b the number of draws below. Then these asymptotically (follows when $n \to \infty$) follow:

$$(D_a + D_b) \sim N\left(2\frac{n_1 n_2}{n_1 + n_2} + 1, 2\frac{n_1 n_2(2n_1 n_2 - n_1 - n_2)}{(n_1 + n_2)^2(n_1 + n_2 - 1)}\right),$$

where $n_1 = n_2 = 5000$.

```r
n1 = 5000
n2 = 5000
m = 2*n1*n2/(n1+n2)+1
s = 2*n1*n2*(2*n1*n2-n1-n2)/((n1+n2)^2*(n1+n2-1))
ci = c(qnorm(0.025, m, sqrt(s)), qnorm(0.975, m, sqrt(s)))
ci
```

```
## [1] 4903.007 5098.993
```

```r
library('randtests')
runs.test(U, threshold = median(U), plot = F)
```

```
##
##  Runs Test
##
## data:  U
## statistic = 2.7201, runs = 5137, n1 = 5000, n2 = 5000, n = 10000,
## p-value = 0.006526
## alternative hypothesis: nonrandomness
```

So we find that we expect (95% confidence interval) that $(D_a + D_b) \in_{0.95\%} [4903.007, \quad 5098.993]$. For our draws we have $D_a + D_b = 4953$, we are therefore happy.

Ups/downs test: Here we use a slightly different approach where we go through the sequence and split into increasing runs. We, therefore, start a new run each time we encounter an x_i where $x_{i-1} > x_i$. As described in the lecture slides, we group the number of runs of length i into 6 bins, where the bin 6 contains all runs with length more than or equal to 6. We have additionally have

$$Z = \frac{1}{n-6}(R - nB)^T A(R - nB).$$

Then cf the slides, we get

$$Z \sim \chi(6).$$

```r
A = matrix(data=c(4529.4,9044.9,13568,18091,22615,27892,
9044.9,18097,27139,36187,45234,55789,
13568,27139,40721,54281,67852,83685,
18091,36187,54281,72414,90470,111580,
22615,45234,67852,90470,113262,139476), nrow = 6, ncol = 6)
b = t(c(1/6, 5/24,11/120, 19/720, 29/5040, 1/840))
r  = list()
ii = 2
jj = 1
r[[1]] = 1
while (ii <= N) {
  if (U[ii]>U[ii-1]) {
    r[[jj]] = r[[jj]]+1
  }  else {
    jj = jj + 1
    r[jj] = 1
  }
  ii = ii + 1
}
r = unlist(r)
R = t(c(sapply(1:5, function(i) sum(r==i)),sum(r>=6)))
Z = 1/(N-6) * (R - N*b) %*% A %*% t(R-N*b)
1 - pchisq(Z, 6)
```

```
##              [,1]
## [1,] 0.009579339
```

We get a p-value of 0.45, i.e., according to this test, there is a 45% chance of seeing less uniform numbers given that the underlying distribution is in fact uniform. In other words, we cannot reject that U does follow a uniform distribution.

The-Up-and-Down Test: This test is the natural next step to the "ups/downs test". This time we count both increasing and decresing runs. Let X denote the numbers of runs, i.e., the number of runs where it is either decreasing or increasing. Further let

$$Z = \frac{X - \frac{2n-1}{3}}{\sqrt{\frac{16n-29}{90}}}$$

then $Z \sim N(0,1)$.

```
r  = list()
ii = 2
jj = 1
r[[1]] = 0
over = U[ii] > U[ii-1]
while (ii <= N) {
  if (U[ii]>U[ii-1] && over) {
    r[[jj]] = r[[jj]]+1
  }
  else if (U[ii]<U[ii-1] && !over){
    r[[jj]] = r[[jj]]+1
  }
  else {
    over = U[ii] > U[ii-1]
    jj = jj + 1
    r[jj] = 1
  }
  ii = ii + 1
}
r = unlist(r)
print(1 - 2*pnorm( (length(r) - (2*N-1)/3)/sqrt((16*N-29)/90), 0, 1))
```

```
## [1] -0.9601854
```

We get a p-value of 0.73, i.e., according to this test, there is a 73% chance of seeing less uniform numbers given that the underlying distribution is in fact uniform. In other words, we cannot reject that U does follow a uniform distribution.

Correlation coefficient: Finally, we compute the correlation coefficient given by

$$c_h = \frac{1}{n-h} \sum_{i=1}^{n-h} U_i U_{i+h},$$

where $c_h \sim N\left(\frac{1}{4}, \frac{7}{144n}\right)$.

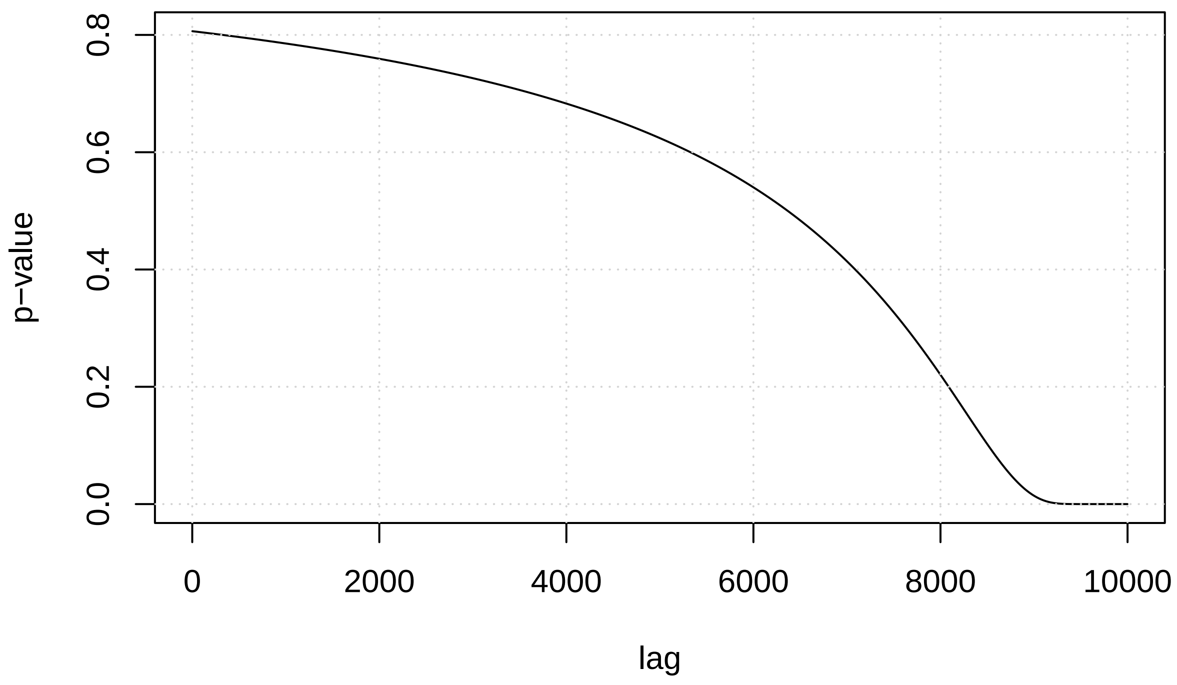
```
ch <- function(U,h) {
  1/(N-h) * sum( U[1:(N-h)] * U[(h+1):N])
}
hs = seq(1,9999,1)
```

```r
plot(hs, 2*pnorm(-abs(ch(U,hs)), 7/(144*N)), ylab = 'p-value',
     xlab = 'lag', type = 'l')
```

```
## Warning in 1:(N - h): numerical expression has 9999 elements: only the first
## used
```

```
## Warning in (h + 1):N: numerical expression has 9999 elements: only the first
## used
```

```r
grid()
```



From the plot above we see that the for the first many lags, the p-value is non-significant at a 5% level. When the lag gets sufficiently large, the p-value becomes significant, this is because our implementation deos not have an infinite cycle length.

Computer exercise 2

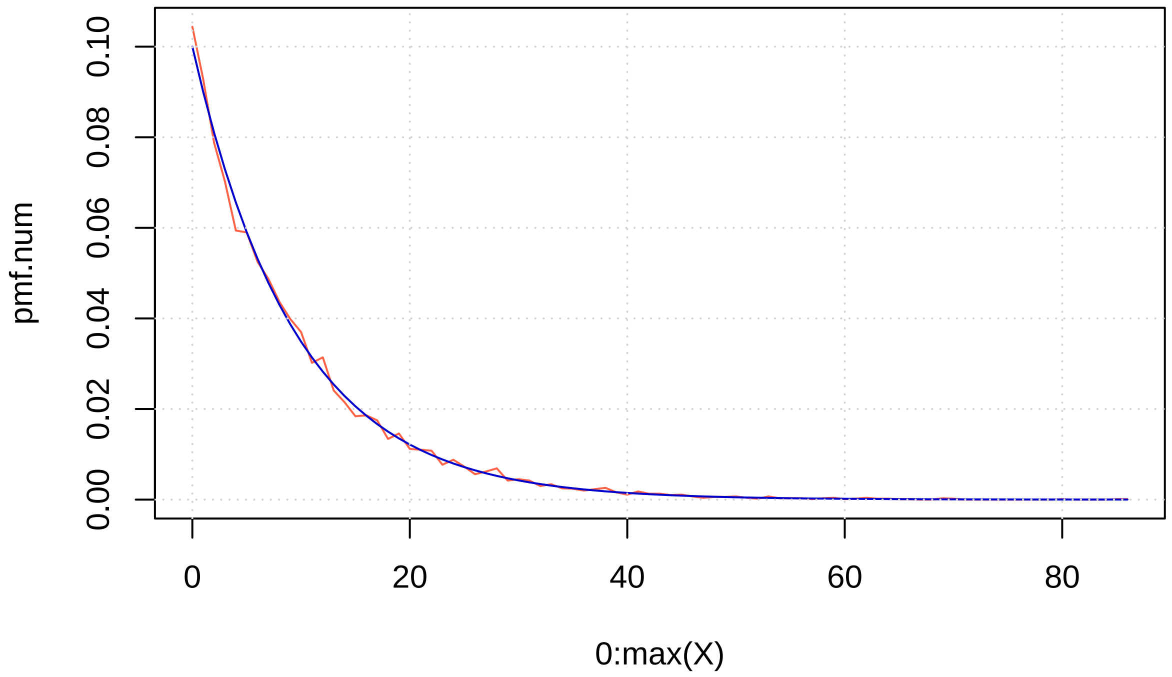
1

We start by looking at geometric distributions. The geometric distribution is a probability distribution of n independent Bernoulli trails. To get samples from a geometric distribution using a uniform distribution we use the transformation applied in the slides. We choose a probability of p = 0.1 and simulate N = 10000 samples.

```r
set.seed(1)
N = 10000
U = runif(N,0,1)
p = 0.1
X = floor(log(U)/log(1-p))
pmf.num = sapply(0:max(X), function(i) sum(X==i)) / N
cdf.geom = pgeom(0:max(X),p)
```

The samples can be compared the expected results for a geometric distribution using R sampling. This is shown visually below:

```r
plot(0:max(X),pmf.num, ylim = c(0,max(pmf.num)), col = 'tomato', type ='l')
lines(0:max(X), dgeom(0:max(X),p), col = 'blue3')
grid()
```



Quantitatively the simulated data looks very similar to the expected result. To do a more formal test of their equivalence a $\mathcal{X}^2$ test for the samples and the expected results.

```r
## Chisq test
chi.t = sum((pmf.num*N - dgeom(0:max(X),p)*N)^2/ (dgeom(0:max(X),p)*N))
1 - pchisq(chi.t, max(X)+1)
```

```
## [1] 0.3663582
```

From this we can conclude that the simulated points follow a geometric distribution.

2

In this exercise we want to simulate 6 points distributions by applying different sampling techniques, for the following probabilities:

```r
p = c(7/48, 5/48, 1/8, 1/16, 1/4, 5/16)
```

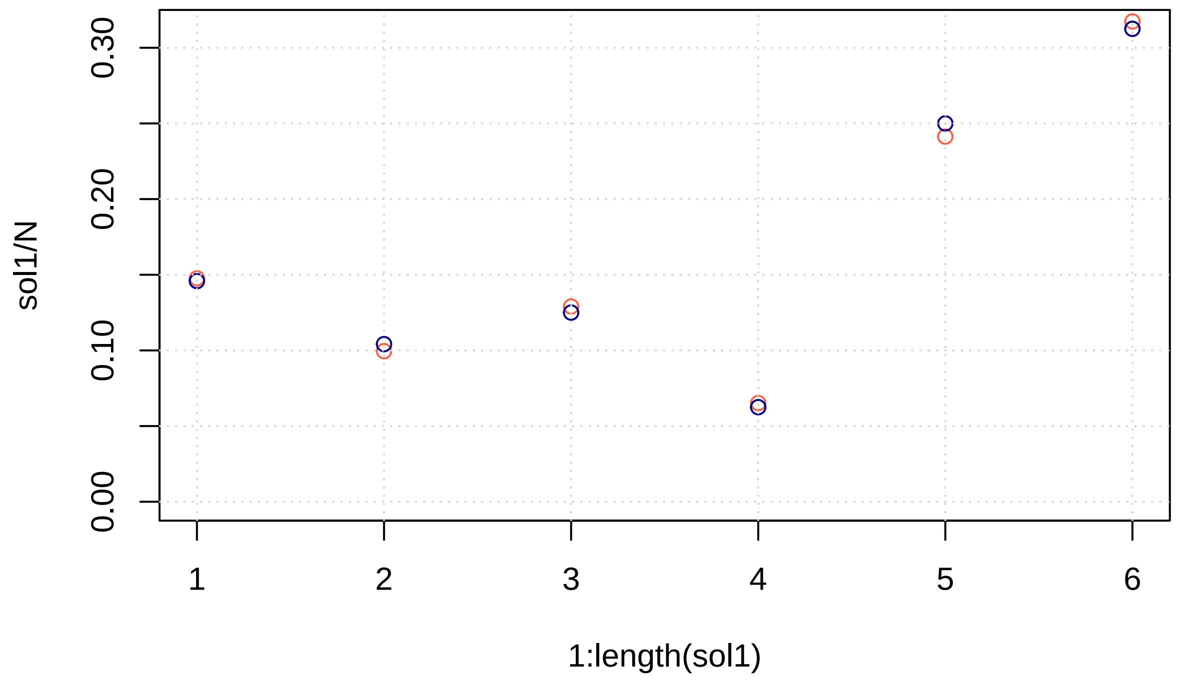
Firstly the direct (crude) method is applied.

The value of X is determined using the cumulative density function where the uniform sample needs to fall in the interval of the sample and previous sample

$$X = x_i \quad if \quad F(x_{i-1} < U \leq F(x_i))$$

The crude method is applied to the six point discrete distribution and the simulated result is plotted:

```r
crude <- function(p, U){
  p = cumsum(p)
  sapply(1:length(p), function(i) sum( U>=ifelse(i==1, 0, p[i-1]) & U<p[i] ))
}

sol1 = crude(p,U)
plot(y = sol1/N, x = 1:length(sol1), ylim = c(0,max(p)), col = 'tomato')
points(y = p, x = 1:length(p), col = 'blue4')
grid()
```



The simulated points seem to mathc the true distribution relatively well

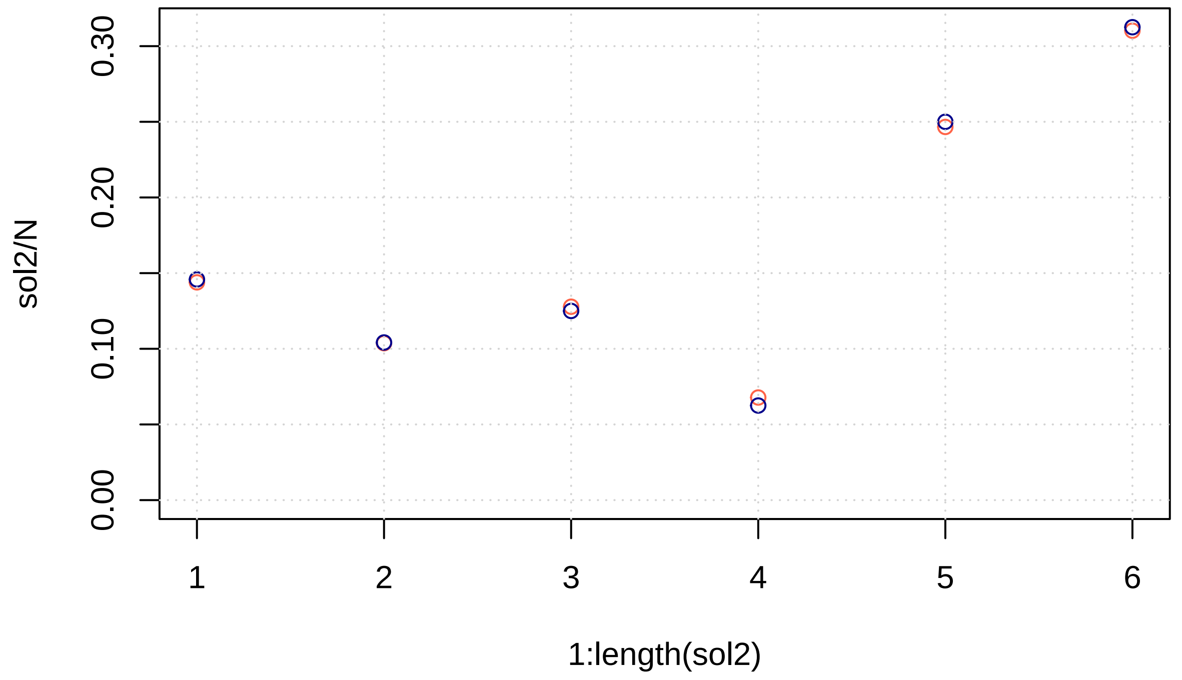
Rejection method:

We let $c \geq p_i$ then $\frac{p_i}{c} \geq 1$.

For each U we compute: $I = \lfloor kU_1 \rfloor + 1$ $U_2 < \frac{pI}{c}$

The rejection method is applied to our distribution:

```r
rejection <- function(p){
  c = max(p)
  k = length(p)

  ii = 1
  while (ii < length(U)) {
    I = floor(k*runif(1)) + 1
    if (runif(1) <= p[I]/c) return(I)
    ii = ii+1
  }
}

aux = replicate(N, rejection(p))
sol2 = sapply(1:length(p), function(i) sum(aux==i))
plot(y = sol2/N, x = 1:length(sol2), ylim = c(0,max(p)), col = 'tomato')
points(y = p, x = 1:length(p), col = 'blue4')
grid()
```



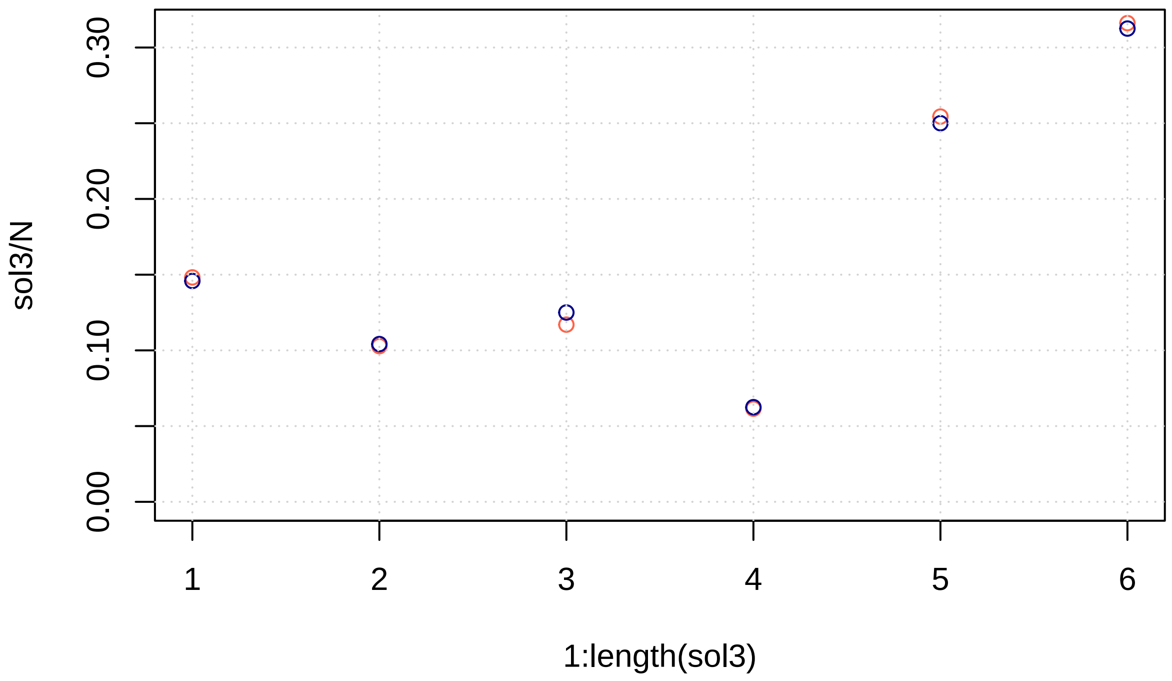
The simulated points again match the probability distribution relatively well.

Alias method

Here we setup tables $F(I)$ and $L(I)$ and on generation the following is run: $I = \lfloor kU_1 \rfloor + 1$ if $U_2 \leq F(I)$ output I else output $L(i)$

This is applied:

```r
alias <- function(p){
  k = length(p)
  L = 1:length(p)
  FF = p*length(p)
  G = which(FF>=1)
  S = which(FF<=1)

  while (length(S)>0) {
    i = G[1]
    j = S[1]
    L[j] = i
    FF[i] = FF[i] - (1-FF[j])
    if (FF[i]<(1-1e-9)){
      G = G[-1]
      S = c(S, i)
    }
    S = S[-1]
  }

  I = floor(k*runif(1)) + 1
  if (runif(1) <= FF[I]) {
    return(I)
  } else return(L[I])
}

aux = replicate(N, alias(p))
sol3 = sapply(1:length(p), function(i) sum(aux==i))
plot(y = sol3/N, x = 1:length(sol3), ylim = c(0,max(p)), col = 'tomato')
points(y = p, x = 1:length(p), col = 'blue4')
grid()
```



3

The three methods: crude, Rejection and Alias are now compared using $\mathcal{X}^2$ and Kolmogorov testing:

Testing the crude method.

```
## Chisq test
chi.t = sum((sol1 - p*N)^2/ (p*N))
1 - pchisq(chi.t, length(p))
```

```
## [1] 0.2007022
```

```r
ks.test(sol1/N, p)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  sol1/N and p
## D = 0.16667, p-value = 1
## alternative hypothesis: two-sided
```

Testing the rejection method.

```r
## Chisq test
chi.t = sum((sol2 - p*N)^2/ (p*N))
1 - pchisq(chi.t, length(p))
```

```
## [1] 0.4197831
```

```r
ks.test(sol2/N, p)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  sol2/N and p
## D = 0.16667, p-value = 1
## alternative hypothesis: two-sided
```

$\mathcal{X}^2$ test for the Alias method.

```r
## Chisq test
chi.t = sum((sol3 - p*N)^2/ (p*N))
1 - pchisq(chi.t, length(p))
```

```
## [1] 0.3178518
```

```r
ks.test(sol3/N, p)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  sol3/N and p
## D = 0.16667, p-value = 1
## alternative hypothesis: two-sided
```

It is clear that all 3 methods closely approximates the true solution, where the quality of their fit ranges worst to best: crude, rejection, alias.

4

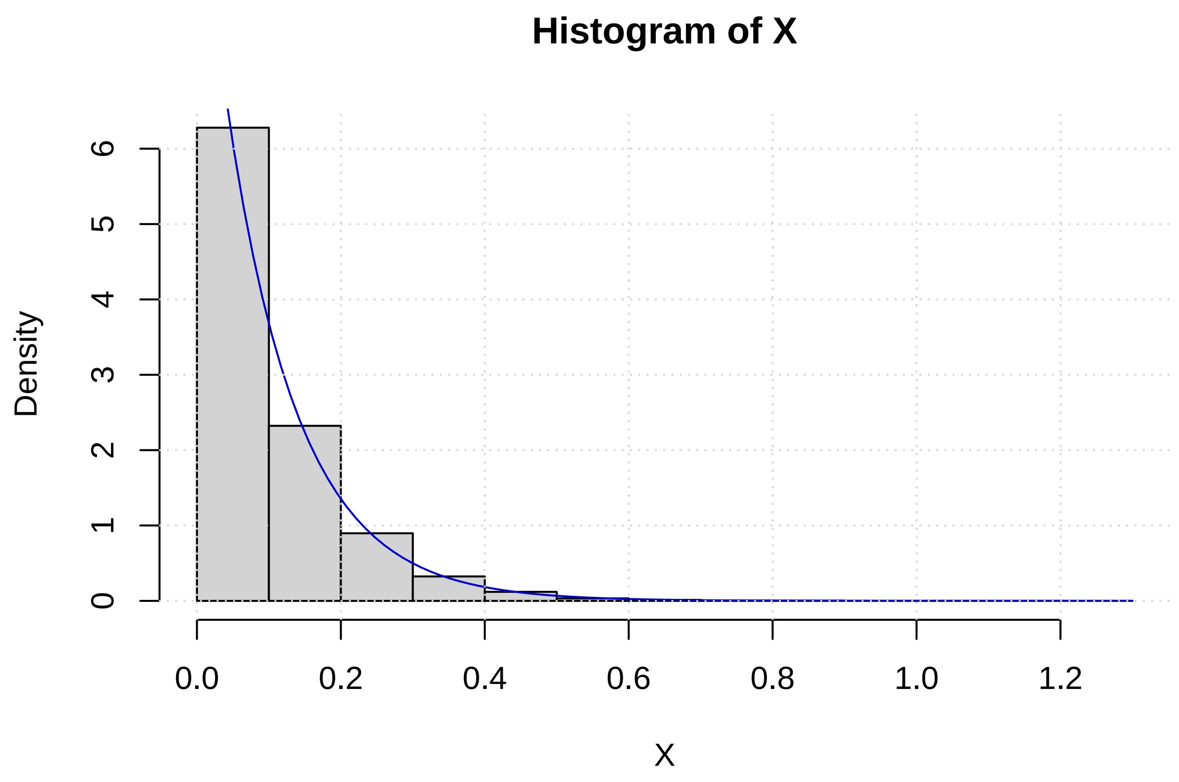
Crude method is easy to implement and works well relatively well but is slow compared to the other methods. Rejection sampling is more precise but does not scale well with higher dimensional problems. Alias method is generally the most precise method and has good efficiency but can be very sub optimal in cases where the probability distributions that are particularly well balanced

Computer exercise 3

1

Initially we sample from an exponential distribution using the inversion method and plotting the sampled values vs. a true exponential continous distribution:

```
N = 10000
U = runif(N,0,1)
lambda = 10
X = - log(U)/lambda
hist(X,probability = T)
curve(expr = dexp(x = x,rate = lambda), col = 'blue3', add = T)
grid()
```



The quality of the simulated values is tested using Kolmogorov-Smirnov testing:

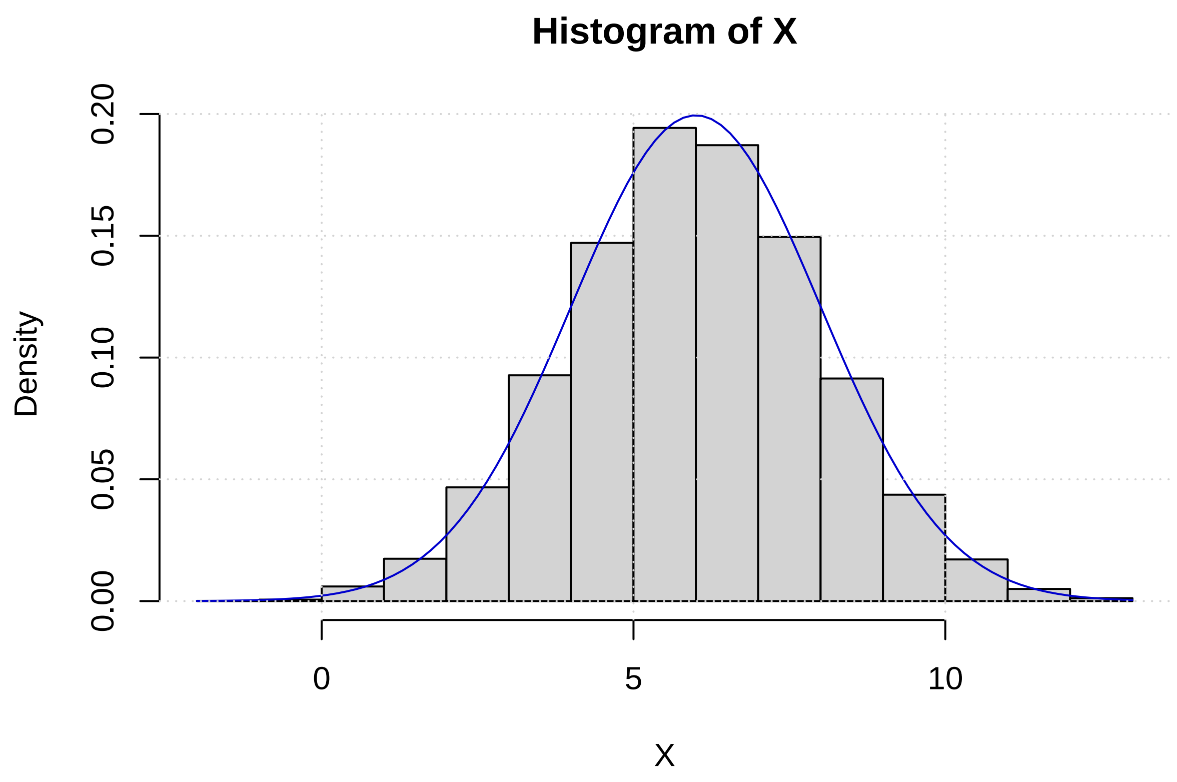
```
## Kolmogorov Smirnov
ks.test(X, 'pexp', lambda)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  X
## D = 0.0094184, p-value = 0.3376
## alternative hypothesis: two-sided
```

We can thus not say that the two distributions are dissimilar with regards to the KS test.

The inversion method is then applied to the sample gaussian values from a uniform distribution and the results are plotted against the true corresponding gaussian distribution:

```r
N = 10000
U = runif(N,0,1)
mu = 6
sigma = 2

X = qnorm(U, mean = mu, sd = sigma)
hist(X,probability = T)
curve(expr = dnorm(x,mu,sigma), col = 'blue3', add = T)
grid()
```



The simulated values seem to match the true distribution well, to test the fit a KS test is applied:

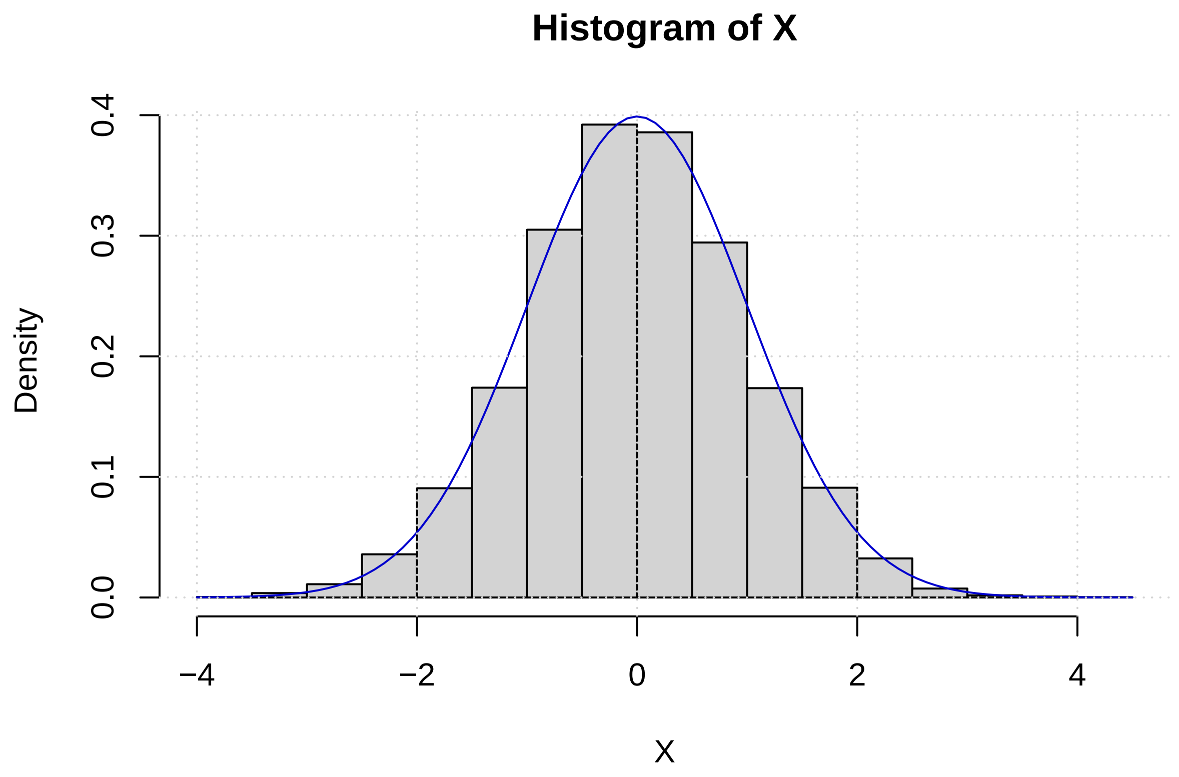
```r
## Kolmogorov Smirnov
ks.test(X, 'pnorm', mu,sigma)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  X
## D = 0.0095394, p-value = 0.3227
## alternative hypothesis: two-sided
```

Given the higher p-value the Gaussian inverse is thus a even better approximation than the exponential inverse.

The Box-Mueller is used for a standard normal here and plotted against the true distribution:

```r
Box.Muller <- function(){
  sqrt(-2*log(runif(1)))*cos(2*pi*runif(1))
}
X = replicate(N,Box.Muller())
hist(X,probability = T)
curve(expr = dnorm(x,0,1), col = 'blue3', add = T)
grid()
```



KS testing:

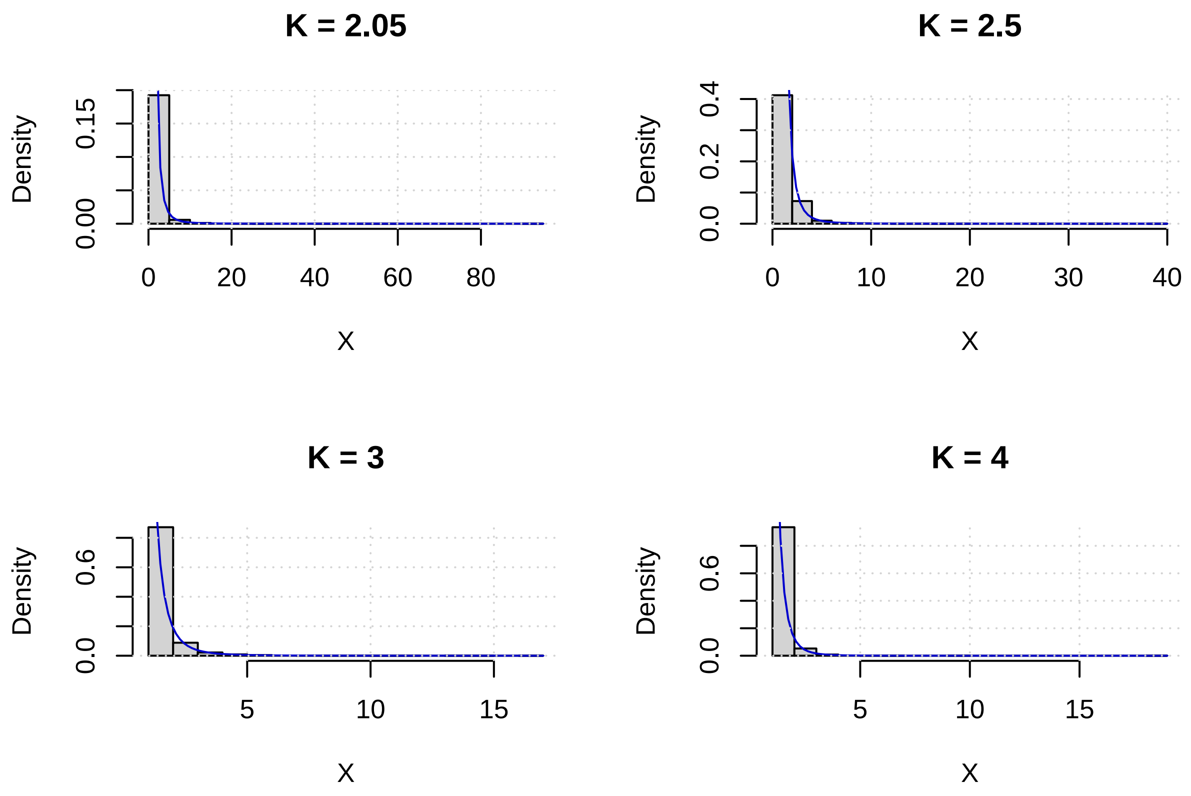
```r
## Kolmogorov Smirnov
ks.test(X, 'pnorm', 0,1)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  X
## D = 0.010017, p-value = 0.2682
## alternative hypothesis: two-sided
```

The simulated values can thus not be said to be different from the true distribution.

Pareto distributed values are now simualted for different values of k: $k = \{2.05, 2.5, 3, 4\}$:

```r
dpareto <- function(x,k,beta){
  k*beta^k/(x^(k+1))
}
ppareto <- function(x,k,beta){
  1-(beta/x)^k
}
N = 10000
ks = c(2.05, 2.5, 3, 4)
beta = 1
par(mfrow = c(2,2))
p.vals = sapply(1:4, function(i) {
  X = beta*runif(N,0,1)^(-1/ks[i])
  hist(X,probability = T, main = paste("K =",ks[i]))
  curve(expr = dpareto(x,ks[i],beta), col = 'blue3', add = T)
  grid()
  ks.test(X, 'ppareto', ks[i], beta)
})
```



KS testing for each of the k values are done:

```r
p.vals
```

```
##                [,1]
## statistic    0.01147648
## p.value     0.1435009
## alternative "two-sided"
## method      "One-sample Kolmogorov-Smirnov test"
## data.name   "X"
##             [,2]
## statistic   0.006180461
## p.value     0.8395242
## alternative "two-sided"
## method      "One-sample Kolmogorov-Smirnov test"
## data.name   "X"
##             [,3]
## statistic   0.01246121
## p.value     0.08958394
## alternative "two-sided"
## method      "One-sample Kolmogorov-Smirnov test"
## data.name   "X"
##             [,4]
## statistic   0.005877773
## p.value     0.8800413
## alternative "two-sided"
## method      "One-sample Kolmogorov-Smirnov test"
## data.name   "X"
```

Generally we see that the approximations become better with higher values of k. This is due to the high variance (heavy tails) contained in the Pareto distribution, which is especially prominent for low values of k.

2

We now wish to compare Pareto numerical expectations and variance with analytical expectations.

```r
par(mfrow = c(2,2))
num.e = mean(X)
sol = lapply(1:4, function(i) {
  X = beta*runif(N,0,1)^(-1/ks[i])
  list(
  num.e = mean(X),
  num.var = mean(abs(X - num.e)^2),
  analyt.e = ks[i]*beta/(ks[i]-1),
  analyt.var = ks[i]/((ks[i]-1)^2*(ks[i]-2)))
})

names(sol) = paste0('k_',ks)
sol
```

```
## $k_2.05
## $k_2.05$num.e
## [1] 1.904924
##
## $k_2.05$num.var
## [1] 7.365938
##
## $k_2.05$analyt.e
## [1] 1.952381
```

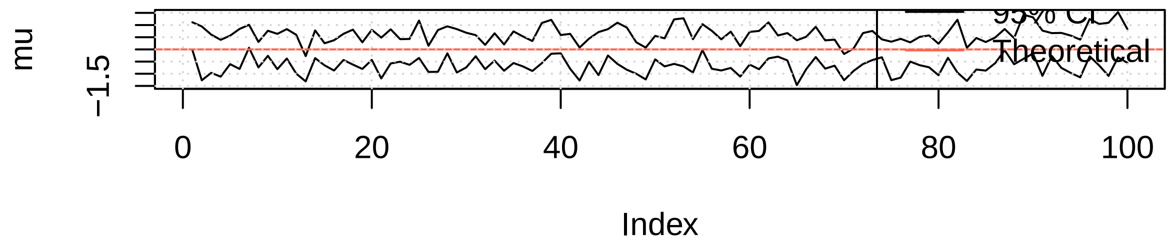
```
##
## $k_2.05$analyt.var
## [1] 37.18821
##
##
## $k_2.5
## $k_2.5$num.e
## [1] 1.678256
##
## $k_2.5$num.var
## [1] 4.395547
##
## $k_2.5$analyt.e
## [1] 1.666667
##
## $k_2.5$analyt.var
## [1] 2.222222
##
##
## $k_3
## $k_3$num.e
## [1] 1.496613
##
## $k_3$num.var
## [1] 2.918116
##
## $k_3$analyt.e
## [1] 1.5
##
## $k_3$analyt.var
## [1] 0.75
##
##
## $k_4
## $k_4$num.e
## [1] 1.337326
##
## $k_4$num.var
## [1] 2.13789
##
## $k_4$analyt.e
## [1] 1.333333
##
## $k_4$analyt.var
## [1] 0.2222222
```

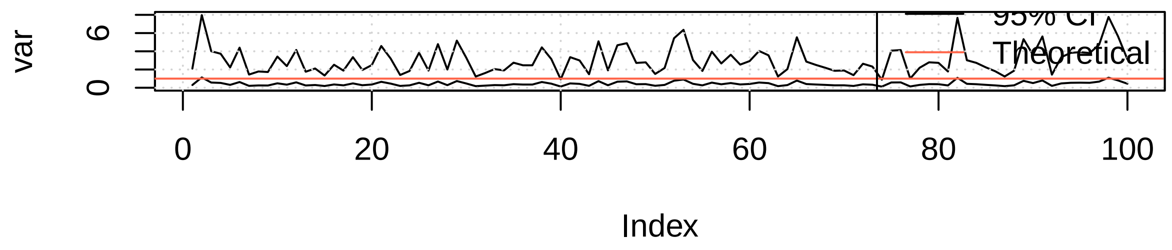
It seems the expectation is OK, but the variance is off. The reason could be that we are fairly close to something that is not well defined (var is inf for k=2). Pareto has quite a heavy tail - it is very hard to account for that in simulations (very few points there).

3

100 normal distributions are done based on 10 observations and the mean and variance 95% confidence intervals are plotted for each to gain insights into the fluctuations of randomly simulated normal distributions.

```r
set.seed(42)
ci.norm <- function(N){
  U = rnorm(N)
  mu.hat = mean(U)
  s = sqrt(mean( abs(U - mu.hat)^2 ))
  svend = list(
  mu.ci = mu.hat + qt(0.025,N-1)*sqrt(s^2/N)*c(-1,1),
  s.ci = (N-1)*s^2/c(qchisq(1-0.025,N-1),qchisq(0.025,N-1)))
}

sol = replicate(100, ci.norm(10))
par(mfrow=c(2,1))
plot(sapply(sol[1,], function(x) x[1]), ylim = c(-1.5, 1.5), type = 'l', ylab = "mu")
lines(sapply(sol[1,], function(x) x[2]))
abline(h=0, col = 'tomato')
grid()
legend('bottomright', c('95% CI', 'Theoretical'), lty = 1, col = c('black', 'tomato'))

plot(sapply(sol[2,], function(x) x[1]), ylim = c(0, 8), type = 'l', ylab = "var")
lines(sapply(sol[2,], function(x) x[2]))
abline(h=1, col = 'tomato')
grid()
legend('bottomright', c('95% CI', 'Theoretical'), lty = 1, col = c('black', 'tomato'))
```





As expected a large proportion of the sampled confidence intervals have the theorectical mean/variance contained in them. We investigate the precise number:

```
set.seed(42)
ci.m <- function(N){
  U = rnorm(N)
  mu.hat = mean(U)
  s = sqrt(mean( abs(U - mu.hat)^2 ))
  mu.ci = mu.hat + qt(0.025,N-1)*sqrt(s^2/N)*c(-1,1)
  return (mu.ci)
}
ci.s <- function(N){
  U = rnorm(N)
  mu.hat = mean(U)
  s = sqrt(mean( abs(U - mu.hat)^2 ))
  s.ci = (N-1)*s^2/c(qchisq(1-0.025,N-1),qchisq(0.025,N-1))
  return (s.ci)
}

solm = replicate(100, ci.m(10))
sols = replicate(100, ci.s(10))

print(paste("Theorectical mean in CI: ",sum(solm[1,] > 0 & solm[2,] < 0)))
```

```
## [1] "Theorectical mean in CI:  97"
```

```r
print(paste("Theorectical variance in CI: ", sum(sols[1,] < 1 & sols[2,] > 1)))
```

```
## [1] "Theorectical variance in CI:  95"
```

We see that 97 of the sampled CIs have the mean contained in them and 95 of the sampled variance CIs have the theoretical variance contained. As we apply a 95% confidence interval we expect this number to be 95, where for $reps : 100 \to \infty$ this should converge to be the case (frequentist perspective). This follows naturally from the central limit theorem.

Computer exercise 4

1

We define a discrete event simulation program for a blocking system, i.e. a system with m service units and no waiting room.

```r
q.blocks <- function(arr.time, serv.time, m){
  ## arr.times: vector of absolute times when a user arrives (cummulative waiting times)
  ## serv.times: vector service time for users (not cummulative)
  ## m: number of service units

  blocks = 0
  in.service = rep(0,m)
  for (ii in 1:N) {
    if (all(arr.time[ii] < in.service)){
      blocks = blocks + 1
    }
    else {
      in.service[ which.min( in.service >= arr.time[ii] ) ] = arr.time[ii]+serv.time[ii]
    }
  }
  return (blocks)
}
```

We now simulate a M/M/10/0 system. The mean of the arrival process is 1 and the mean service time is 8. For each simulation we do 10000 draws. We repeat the simulation 10 times.

To estimate confidence intervals for the fraction blocked, we assume the numerical solutions to follow a normal distribution, i.e.,

$$\theta_{95\%} = \bar{\theta} \pm \frac{\hat{\sigma}_\theta}{\sqrt{n}} t_{\alpha/2}(n-1),$$

where n is the number of repeated simulations.

```r
n.sim = 10
N = 10000
m = 10

num.prob = sapply(1:n.sim, function(i) {
  set.seed(i)
  at = cumsum(rexp(n = N, rate = 1))
  st = rexp(n=N, rate = 1/8)
  q.blocks(at, st, m)
}) / N

num.e = mean(num.prob)
num.s = sqrt(sum(abs(num.prob - num.e)^2) / (n.sim - 1))
num.ci = num.e + num.s/sqrt(n.sim) * qt(0.975,n.sim-1)*c(-1,1)
num.ci
```

```
## [1] 0.118441 0.126459
```

As given in the slides, the analytical blocking probability is given by

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

with arrival intensity λ and mean service time s.

```
lambda = 1
s = 8
A = lambda*s

(A^m/factorial(m)) / do.call('sum',lapply(1:m, function(i) A^i / factorial(i)))
```

```
## [1] 0.1217111
```

We are in luck! It seems that the analytical solution lies within our estiamted confidence interval.

2

a)

We now wish to simulate a K/M/10/0 system, where the arrival system is given by an Erlang distribution with mean 1. The first thing we realize is that if the shape parameter of the Erlang distribution is 1, then it is in fact an exponential distribution! When the shape parameters increases, the variance in the Erlang distribution decreases, and as it goes to infinity, the variance goes to zero. Therefore, the Erlang distribution lies within the exponential and the constant distribution for arrival proccesses.

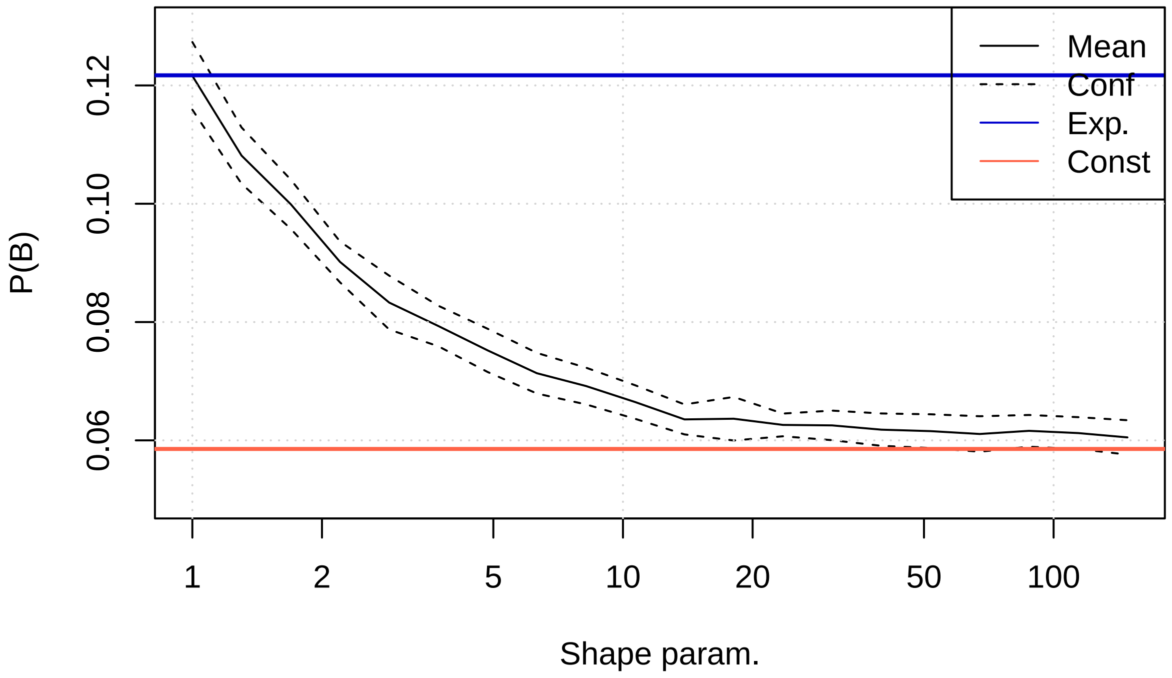
We therefore start by finding the expected number of blockings when the arrival process is constant.

```
num.prob = sapply(1:100, function(i) {
  set.seed(i)
  at = cumsum(rep(1,N)) # mean = shape/rate
  st = rexp(n=N, rate = 1/8)
  q.blocks(at, st, m)
}) / N
num.e = mean(num.prob)
num.e
```

```
## [1] 0.058546
```

We can now find the expectation of blokings as a function of the shape parameter:

```
x = exp(seq(0,5,length.out = 20))
sol = lapply(x, function(theta){
num.prob = sapply(1:n.sim, function(i) {
  set.seed(i)
  shape = theta
  rate = shape
  at = cumsum(rgamma(N,shape,rate)) # mean = shape/rate
  st = rexp(n=N, rate = 1/8)
  q.blocks(at, st, m)
}) / N
num.e = mean(num.prob)
num.s = sqrt(sum(abs(num.prob - num.e)^2) / (n.sim - 1))
num.upr = num.e + num.s/sqrt(n.sim) * qt(0.975,n.sim-1)
num.lwr = num.e - num.s/sqrt(n.sim) * qt(0.975,n.sim-1)
list(mean = num.e,
     upr = num.upr,
     lwr = num.lwr)
})

plot(x, sapply(sol,function(x) x$mean), ylim = c(0.05,0.13),log = 'x',
     type = 'l', ylab = 'P(B)', xlab = 'Shape param.')
lines(x, sapply(sol,function(x) x$upr), lty = 2)
lines(x, sapply(sol,function(x) x$lwr), lty = 2)
abline(h = 0.058546, col = 'tomato', lwd = 2)
abline(h = 0.1217111, col = 'blue3', lwd = 2)
grid()
legend('topright', c('Mean', 'Conf', 'Exp.', 'Const'), lty = c(1,2,1,1),
       col = c(1,1,'blue3', 'tomato'))
```



As expected, the probability of being blocked converges towards the constant arrivel process! Notice that it is because a higher variance means that more users will get blocked.

b)

In the previous, we looked at methods, where the highest variance was the same as an exponential. Now we will look at distribution with higher variance, i.e., hyperexponential distributions.

```
num.prob = sapply(1:n.sim, function(i) {
  set.seed(i)
  at = cumsum(rexp(N,ifelse(runif(N)<0.8, 0.8333, 5.0)))
  st = rexp(n=N, rate = 1/8)
  q.blocks(at, st, m)
}) / N

num.e = mean(num.prob)
num.s = sqrt(sum(abs(num.prob - num.e)^2) / (n.sim - 1))
num.ci = num.e + num.s/sqrt(n.sim) * qt(0.975,n.sim-1)*c(-1,1)
num.ci
```

```
## [1] 0.1407988 0.1482812
```

We see that this time the confidence interval lies above the one from the expoenential arrivel process, i.e., more users are blocked. This aligns perfectly with the discussion in the previous exercise.

3

We will now go back to a Poisson arrival process. Specifically, we will simulate a M/K/10/0 system.

a)

We start by a constant service time:

```
num.prob = sapply(1:n.sim, function(i) {
  set.seed(i)
  at = cumsum(rexp(n = N, rate = 1))
  st = rep(8, N)
  q.blocks(at, st, m)
}) / N

num.e = mean(num.prob)
num.s = sqrt(sum(abs(num.prob - num.e)^2) / (n.sim - 1))
num.ci = num.e + num.s/sqrt(n.sim) * qt(0.975,n.sim-1)*c(-1,1)
num.ci
```

```
## [1] 0.1179417 0.1260583
```

This time we see that we get result very similar to M/M/10/0 system. It seems that the variance of the service time does not have a great influence on the blocking probability.

b)

Now we proceed to a pareto distributed service time. Note that for this distribution, we have extremely fat tails, epspecially if the paramter is low (close to 1). In fact, for $k = 1$ the variance diverges! We assume that the mean service time is 8, and compute the remaining parameter from that.

```
k = 1.05
beta = 8 / (k/(k-1))
num.prob = sapply(1:n.sim, function(i) {
  set.seed(i)
  at = cumsum(rexp(n = N, rate = 1))
  st = beta*runif(N,0,1)^(-1/k) #pareto from unif
  q.blocks(at, st, m)
}) / N

num.e = mean(num.prob)
num.s = sqrt(sum(abs(num.prob - num.e)^2) / (n.sim - 1))
num.ci = num.e + num.s/sqrt(n.sim) * qt(0.975,n.sim-1)*c(-1,1)
num.ci
```

```
## [1] 0.0002458869 0.0013541131
```

```
## poisson / pareto / m / 0
k = 2.05
beta = 8 / (k/(k-1))
num.prob = sapply(1:n.sim, function(i) {
  set.seed(i)
  at = cumsum(rexp(n = N, rate = 1))
  st = beta*runif(N,0,1)^(-1/k) #pareto from unif
  q.blocks(at, st, m)
}) / N

num.e = mean(num.prob)
num.s = sqrt(sum(abs(num.prob - num.e)^2) / (n.sim - 1))
num.ci = num.e + num.s/sqrt(n.sim) * qt(0.975,n.sim-1)*c(-1,1)
num.ci
```

```
## [1] 0.1154298 0.1315902
```

Interesting... We see that for $k = 1.05$ we get strange results. This is because our simulation fails - we simply underestimate the importance of the tails in our simulations. For $k = 2.05$, we get somewhat sensible results, which indicates that the simulation is ok here.

c)

Finally, we want to model the arrival process as a scaled $\chi^2(2)$.

```
num.prob = sapply(1:n.sim, function(i) {
  set.seed(i)
  at = cumsum(rexp(n = N, rate = 1))
  st = rchisq(N,df=2)*4
  q.blocks(at, st, m)
}) / N

num.e = mean(num.prob)
num.s = sqrt(sum(abs(num.prob - num.e)^2) / (n.sim - 1))
num.ci = num.e + num.s/sqrt(n.sim) * qt(0.975,n.sim-1)*c(-1,1)
num.ci
```

```
## [1] 0.1152115 0.1237685
```

This time we again get something fairly close to the M/M/10/0 solution. This makes sense, since the $\chi^2(2)$ is fairly close to $\chi^2(1)$, which is equal to the exponential distribution.

4

Looking at the confidence intervals, it generally seems that the CLT does work, and we get fairly sensible results. However, for the pareto distribution with fat tails, we do get very strange results. This implies that while the CLT usually work, we need to be carefull that the asymptotic behaviour is OK, and that our simulations are able to represent the distribution sufficiently.

Computer exercise 5

Computer exercise 6

In this computer exercise, we will investigate the properties of Markov Chain Monte Carlo (MCMC).

1

The first thing we will do, is to generate values from an Erlang system using Metropolis-Hastings. The system is given by

$$P(i) = c\frac{A^i}{i!}, \quad i = 0, .., m$$

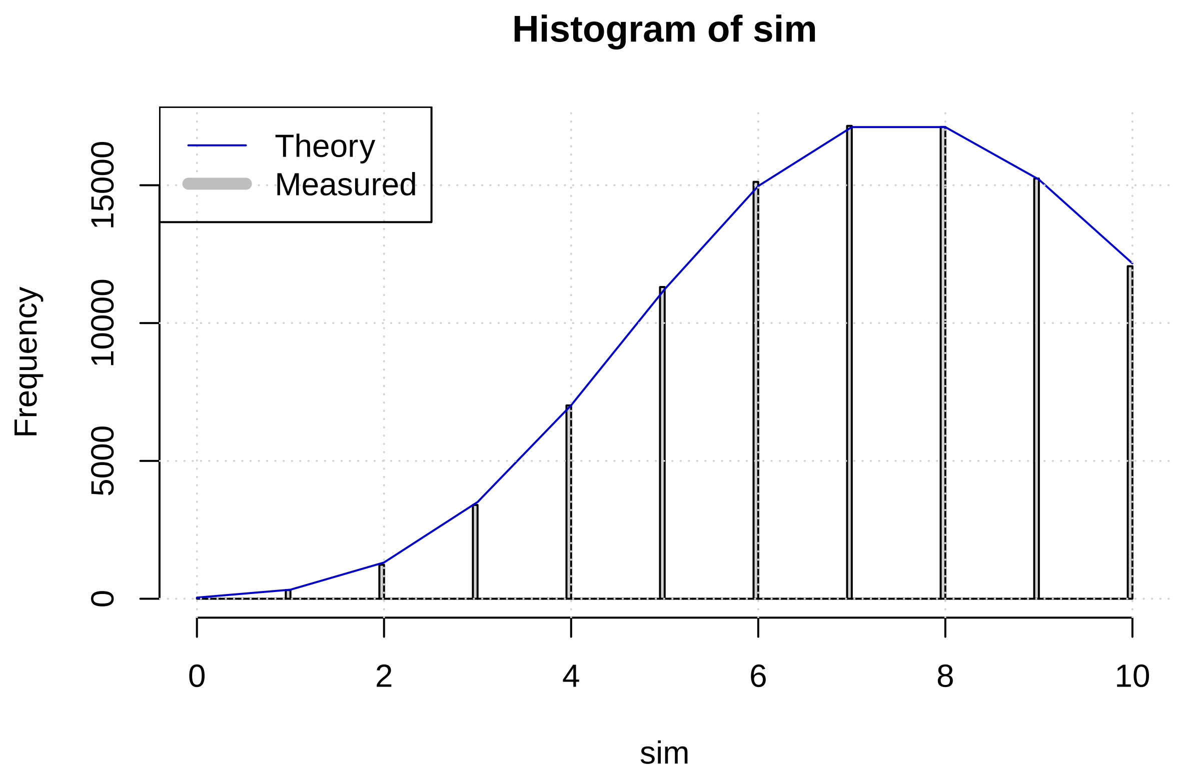
where $A = 8$ and $m = 10$.

First we define Metropolis-Hastings using random walk. Notice that since we are dealing with an Erlang system, the steps have to be integer.

```r
## Random Walk Metropolis-Hastings
RW.MH <- function(x0, n, g){
  X = numeric(n)
  X[1] = x0

  for (ii in 2:n) {
    y = X[ii-1] + ifelse(rbinom(1,1,0.5), 1, -1)
    X[ii] = ifelse((g(y) >= g(X[ii-1])) || (runif(1)<(g(y)/g(X[ii-1]))), y, X[ii-1])
  }
  X
}
```

We can now generate the distribution.

```r
m = 10
A <- 8
g <- function(A,i) ifelse(i<0 || i>m,0,A^i/factorial(i))
g.pdf <- function(x) g(A,x) / sum(sapply(0:m, function(i) g(A,i)))

set.seed(1)
N = 100000
sim = RW.MH(8, N, function(i) g(A,i))

## plotting
hist(sim, breaks = 200)
lines(0:m,N*sapply(0:m, function(i) g.pdf(i)), col = 'blue3')
grid()
legend('topleft', c('Theory', 'Measured'), lty = c(1,1),col=c('blue3', 'grey'), lwd = c(1,6))
```



From the visual inspection, it seems that our simulated distribution lies fairly close to the theoretical.

We now want to do a statistical test to verify that our simulation is correct. Specifically, we will use a χ^2-test. One of the assumptions in this test is that all points are independent. However, since we are using a random walk apporach, we have to de-correlate our simulation by e.g. only considering every 10th point.

```
sim = sim[1:length(sim)%%10 == 1]
n.obs = sapply(0:m, function(i) sum(sim==i))
n.exp = sapply(0:m, function(i) g.pdf(i))*length(sim)
chi.t = sum((n.obs - n.exp)^2/n.exp)

1 - pchisq(chi.t, length(n.obs)-1)
```

```
## [1] 0.8085457
```

We get a p-value of 0.8, i.e., we cannot reject that the simulated and the theoretical distribution are equivalent based on the χ^2-test.

2

We will now consider the 2 dimensional system given by

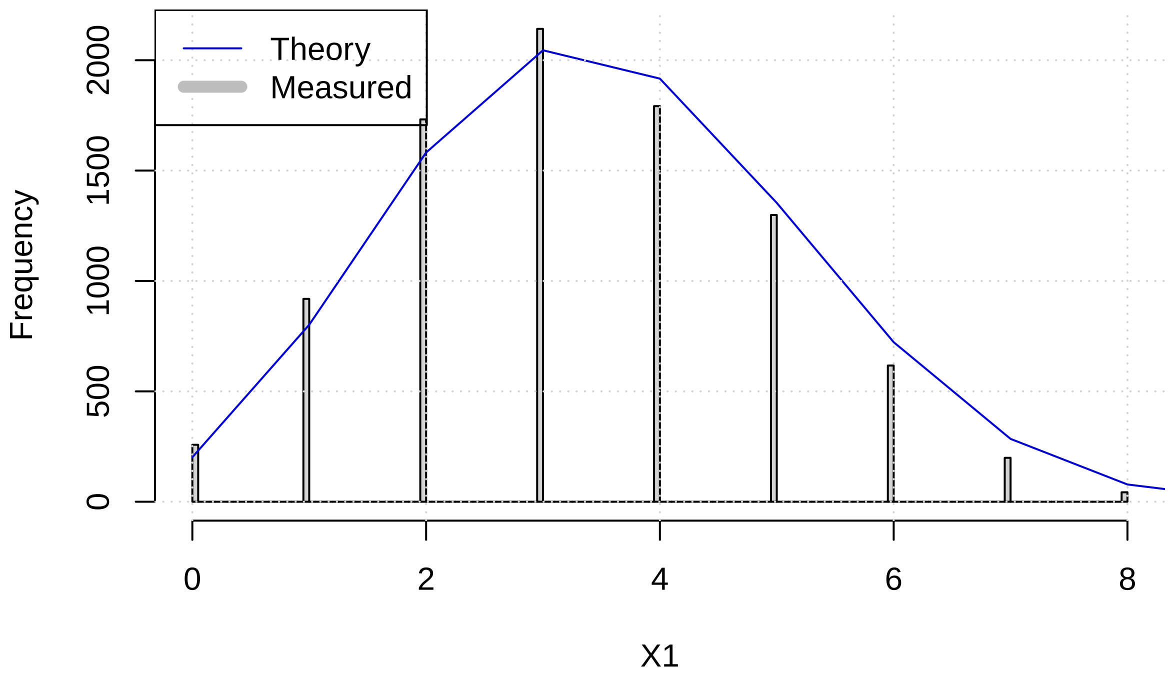
$$P(i,j) = c\frac{A_1^i}{i!}\frac{A_2^j}{j!}, \quad 0 < i,j \quad 0 \leq i+j \leq m$$

where $A_1 = A_2 = 4$ and $m = 10$.

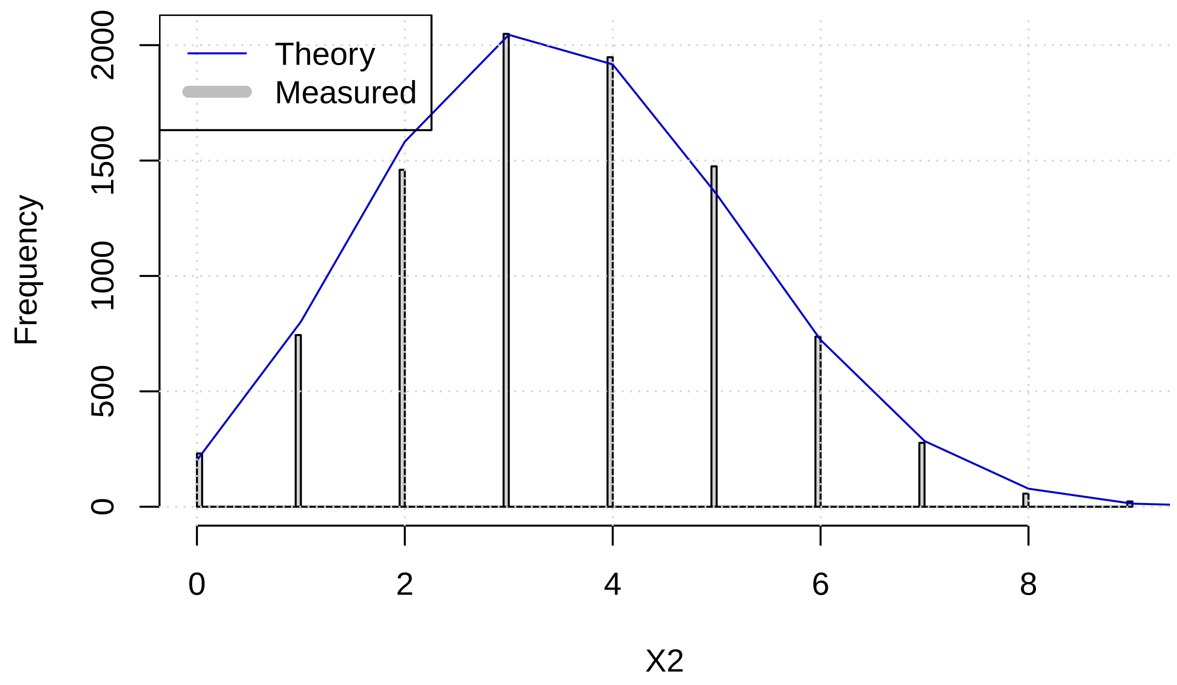
2a

We will now use Metropolis-Hastings directly to generate variates from this distribution.

```r
## usual
RW.MH2d <- function(x0, n, g){
  X = matrix(NA, nrow = n, ncol = length(x0))
  X[1,] = x0
  y = numeric(length(x0))

  for (ii in 2:n) {
    y = X[ii-1,] + ifelse(rbinom(length(x0),1,2/3), ifelse(rbinom(1,1,1/2),0,1), -1)
    while (all(y==X[ii-1])) y = X[ii-1,] + ifelse(rbinom(length(x0),1,2/3),
                                                  ifelse(rbinom(1,1,1/2),0,1), -1)
    X[ii,]  = `if` (( ((g(y) >= g(X[ii-1,])) || (runif(1)<(g(y)/g(X[ii-1,])))) ),  y, X[ii-1, ])
  }
  X
}
g <- function(A,x){
  stopifnot(length(A) == length(x))
  ifelse(any(x<0) || sum(x)>m, 0, prod(A^x/factorial(x)))
}

m = 10
A = rep(4,2)
burnin = 1000
set.seed(1)
N = 10000
sim = RW.MH2d(c(2,2), N, function(x) g(A,x))
sim = sim[burnin:nrow(sim), ]
x.marg = function(x) sum(sapply(0:m, function(zz) g(A,c(x,zz))))

## plotting
hist(sim[,1], breaks = 200, main = '', xlab = 'X1')
lines(0:m,(N-burnin)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x))),
      col = 'blue3')
grid()
legend('topleft', c('Theory', 'Measured'), lty = c(1,1),col=c('blue3', 'grey'), lwd = c(1,6))
```



```
hist(sim[,2], breaks = 200, main = '', xlab = 'X2')
lines(0:m,(N-burnin)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x))),
      col = 'blue3')
grid()
legend('topleft', c('Theory', 'Measured'), lty = c(1,1),col=c('blue3', 'grey'), lwd = c(1,6))
```



We have here plotted the marginal distribution from the simulation together with the theoretical marginal distribution for both i and j. They both seem to follow relatively well - maybe the distribution for X_1, i.e., i, is slighly skewed.

Once again, we perform a χ^2-test.

```
sim = sim[1:nrow(sim)%%10 == 1, ]

n.obs1 = sapply(0:m, function(i) sum(sim[,1]==i))
n.obs2 = sapply(0:m, function(i) sum(sim[,2]==i))
n.exp = nrow(sim)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x)))
chi.t = sum((n.obs1 - n.exp)^2/n.exp) + sum((n.obs2 - n.exp)^2/n.exp)

1 - pchisq(chi.t, length(n.obs1)*2-1)
```

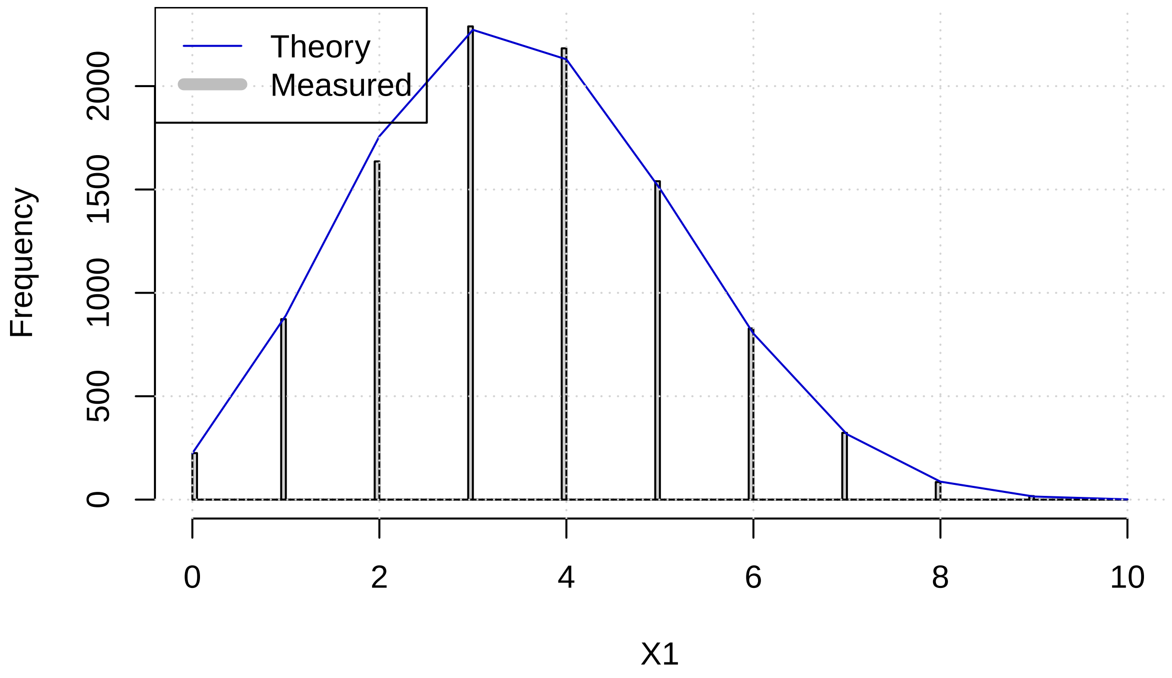
```
## [1] 0.2752178
```

We get a p-value of 0.28, i.e., we cannot reject that the simulated and the theoretical distribution are equivalent based on the χ^2-test.

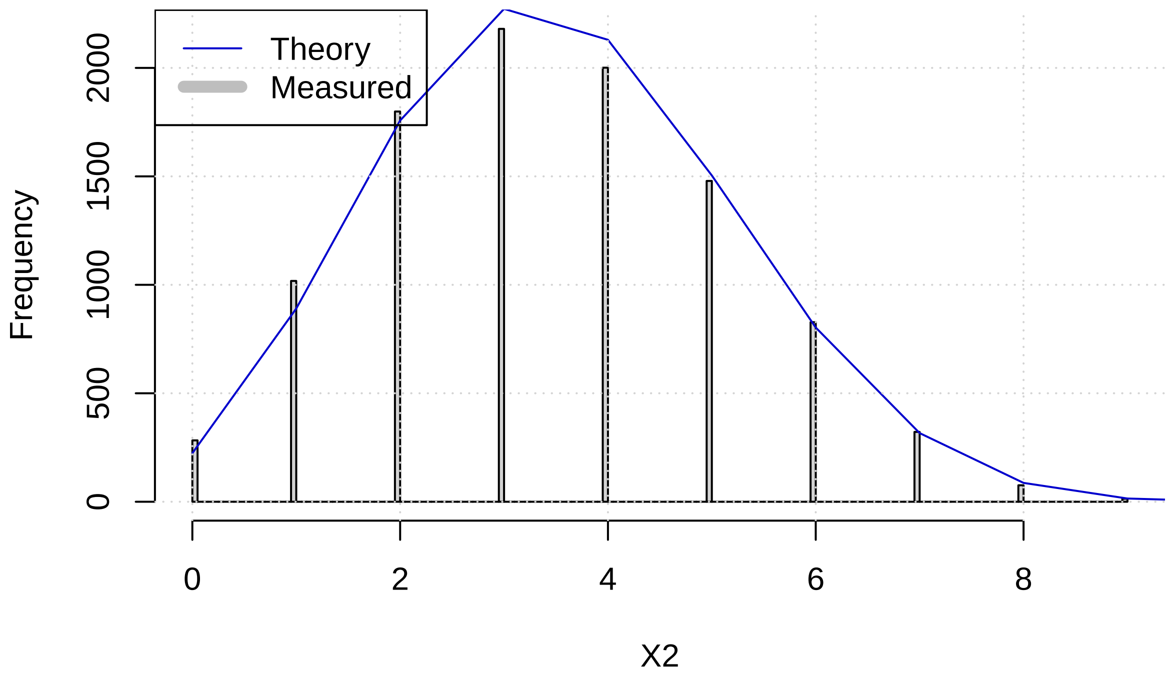
2b

We will now use Metropolis-Hastings coordinate wise to generate variates from this distribution.

```r
## coordinate wise
RW.MH2d.coord <- function(x0, n, g){
  X = matrix(NA, nrow = n, ncol = length(x0))
  X[1,] = x0
  y = numeric(length(x0))

  for (ii in 2:n) {
    X[ii,] = X[ii-1,]
    for (jj in 1:length(x0)) {
      y = X[ii,]
      y[jj] = X[ii-1,jj] + ifelse(rbinom(1,1,1/2), 1, -1)
      X[ii,jj] = ifelse((g(y) >= g(X[ii-1,])) || (runif(1)<(g(y)/g(X[ii-1,]))), y[jj], X[ii-1,jj])
    }
  }
  X
}

set.seed(1)
sim = RW.MH2d.coord(c(2,2), N, function(x) g(A,x))

## plotting
hist(sim[,1], breaks = 200, main = '', xlab = 'X1')
lines(0:m,nrow(sim)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x))),
      col = 'blue3')
grid()
legend('topleft', c('Theory', 'Measured'), lty = c(1,1),col=c('blue3', 'grey'), lwd = c(1,6))
```



```r
hist(sim[,2], breaks = 200, main = '', xlab = 'X2')
lines(0:m,nrow(sim)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x))),
      col = 'blue3')
grid()
legend('topleft', c('Theory', 'Measured'), lty = c(1,1),col=c('blue3', 'grey'), lwd = c(1,6))
```



We have here plotted the marginal distribution from the simulation together with the theoretical marginal distribution for both i and j. They both seem to follow relatively well.

Once again, we perform a χ^2-test.

```
sim = sim[1:nrow(sim)%%10 == 1, ]

n.obs1 = sapply(0:m, function(i) sum(sim[,1]==i))
n.obs2 = sapply(0:m, function(i) sum(sim[,2]==i))
n.exp = nrow(sim)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x)))
chi.t = sum((n.obs1 - n.exp)^2/n.exp) + sum((n.obs2 - n.exp)^2/n.exp)

1 - pchisq(chi.t, length(n.obs1)*2-1)
```

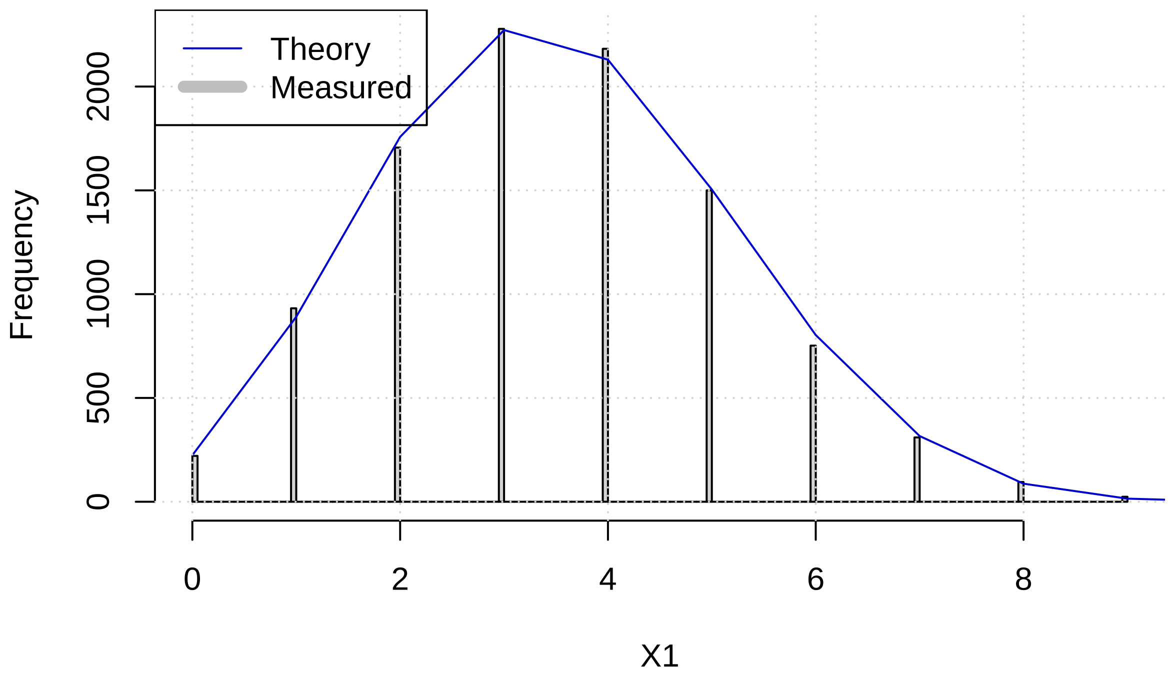
```
## [1] 0.5310416
```

We get a p-value of 0.53, i.e., we cannot reject that the simulated and the theoretical distribution are equivalent based on the χ^2-test.

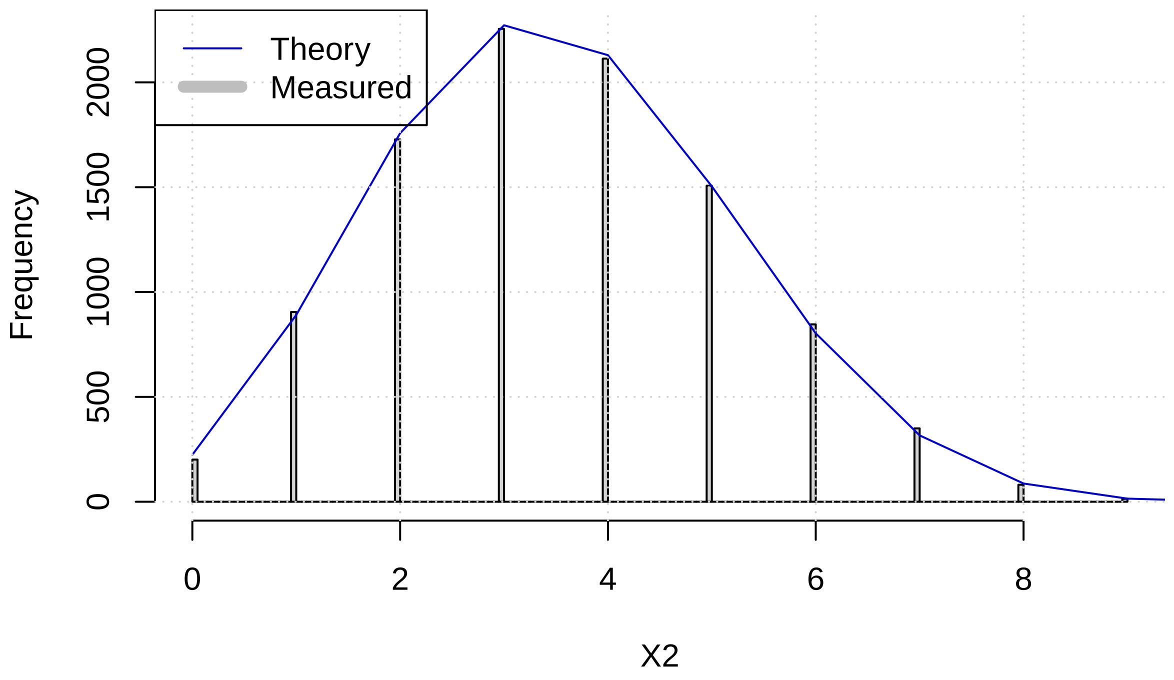
2c

We will now use Gibbs sampling to generate variates from this distribution.

```
## Gibbs sampling
gibbs <- function(x0, n, g){
  X = matrix(NA, nrow = n, ncol = length(x0))
  X[1,] = x0
  y = numeric(length(x0))

  for (ii in 2:n) {
    X[ii,] = X[ii-1,]
    for (jj in 1:length(x0)) {
      g.cond = sapply(0:m, function(i) {
        aux = X[ii,]
        aux[jj] = i
        g(aux)
      })
      f.cond = g.cond / sum(g.cond) ## Conditional distribution
      X[ii,jj] = sample(0:m, 1, prob = f.cond)
    }
  }
  X
}


set.seed(1)
sim = gibbs(c(2,2), N, function(x) g(A,x))

## plotting
hist(sim[,1], breaks = 200, main = '', xlab = 'X1')
lines(0:m,nrow(sim)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x))),
      col = 'blue3')
grid()
legend('topleft', c('Theory', 'Measured'), lty = c(1,1),col=c('blue3', 'grey'), lwd = c(1,6))
```



```r
hist(sim[,2], breaks = 200, main = '', xlab = 'X2')
lines(0:m,nrow(sim)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x))),
      col = 'blue3')
grid()
legend('topleft', c('Theory', 'Measured'), lty = c(1,1),col=c('blue3', 'grey'), lwd = c(1,6))
```



We have here plotted the marginal distribution from the simulation together with the theoretical marginal distribution for both i and j. They both seem to follow relatively well.

Once again, we perform a χ^2-test. Notice that this time, we do not have to de-correlate the data, since the Gibbs sampler does not use a random walk.

```
n.obs1 = sapply(0:m, function(i) sum(sim[,1]==i))
n.obs2 = sapply(0:m, function(i) sum(sim[,2]==i))
n.exp = nrow(sim)*sapply(0:m, function(i) x.marg(i))/sum(sapply(0:m, function(x) x.marg(x)))
chi.t = sum((n.obs1 - n.exp)^2/n.exp) + sum((n.obs2 - n.exp)^2/n.exp)

1 - pchisq(chi.t, length(n.obs1)*2-1)
```

```
## [1] 0.185179
```

We get a p-value of 0.19, i.e., we cannot reject that the simulated and the theoretical distribution are equivalent based on the χ^2-test.

3

We consider a Bayesian statistical problem. The observations are $X_i \sim N(\Theta, \Psi)$, where the prior distribution of the pair $(\Xi, \Gamma) = (\log(\Theta), \log(\Psi))$ is standard normal with correlation $\rho = 1$. The joint density $f(x, y)$ of (Θ, Ψ) is

$$f(x,y) = \frac{1}{2\pi xy\sqrt{1-\rho^2}} \exp\left[-\frac{\log(x)^2 - 2\rho\log(x)\log(y) + \log(y)^2}{2(1-\rho^2)}\right].$$

3a

We start by generating a pair (θ, psi), i.e., a realization of parameters for the distribution of X.

```
library(MASS)
set.seed(42)
sol = mvrnorm(1, rep(0,2), Sigma = matrix(c(1,0.5,0.5,1), 2,2))
eps = sol[1]
gamma = sol[2]

theta = exp(eps)
psi = exp(gamma)
print(paste("theta:", theta))
```

```
## [1] "theta: 4.34764330123791"
```

```
print(paste("psi:", psi))
```

```
## [1] "psi: 2.47177312399365"
```

3b

We can now generate samples of X.

```
set.seed(1)
n = 10
X = rnorm(n, mean = theta, sd = sqrt(psi))
X
```

```
##  [1] 3.362741 4.636365 3.033879 6.855724 4.865691 3.057713 5.113973 5.508428
##  [9] 5.252879 3.867516
```

```
print(paste('Sample mean:', mean(X)))
```

```
## [1] "Sample mean: 4.55549084416965"
```

```
print(paste('Sample variance:', var(X)))
```

```
## [1] "Sample variance: 1.506087049589"
```

3c

We will now try to derive the posterior distribution of (Θ, Ψ) given the samples. To do so we use Bayes theorem, specifically, we condition on the samples just drawn, i.e.,

$$f_{(\Theta,\Psi)|\boldsymbol{X}=\boldsymbol{x}}(\theta,\psi) = \frac{f_{\boldsymbol{X}|(\Theta,\Psi)=(\theta,\psi)}(\boldsymbol{x}) f_{(\Theta,\Psi)}(\theta,\psi)}{f_{\boldsymbol{X}}(\boldsymbol{x})}.$$

Let us take it one step at the time.

$f_{\boldsymbol{X}|(\Theta,\Psi)=(\theta,\psi)}(\boldsymbol{x})$ follow a normal distribution with mean θ and variance ψ, i.e.

$$f_{\boldsymbol{X}|(\Theta,\Psi)=(\theta,\psi)}(\boldsymbol{x}) = \frac{1}{\sqrt{2\psi\pi}} \exp\left[-\frac{(\boldsymbol{x}-\theta)^2}{2\psi}\right].$$

$f_{(\Theta,\Psi)}(\theta,\psi)$ we already know - it was given in the problem description, i.e.,

$$f_{(\Theta,\Psi)}(\theta,\psi) = \frac{1}{2\pi\theta\psi\sqrt{\frac{3}{4}}} \exp\left[-\frac{\log(\theta)^2 - \log(\theta)\log(\psi) + \log(\psi)^2}{\frac{3}{2}}\right].$$

Finally, we need the marginal density of X, $f_{\boldsymbol{X}}(\boldsymbol{x})$.

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \int_{(\Theta,\Psi)} f_{\boldsymbol{X}|(\Theta,\Psi)=(\theta,\psi)}(\boldsymbol{x}) f_{(\Theta,\Psi)}(\theta,\psi) d(\theta,\psi).$$

Inserting we get

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \int_0^\infty \int_0^\infty \frac{1}{\sqrt{2\psi\pi}} \exp\left[-\frac{(\boldsymbol{x}-\theta)^2}{2\psi}\right] \frac{1}{2\pi\theta\psi\sqrt{\frac{3}{4}}} \exp\left[-\frac{\log(\theta)^2 - \log(\theta)\log(\psi) + \log(\psi)^2}{\frac{3}{2}}\right] d\theta d\psi.$$

Hmm, that seems like a difficult integral to solve… In general, integral of the kernel of a normal distribution does not have a closed form solution, and now we even have "log" in the numerator. We do not know any nice tricks to solve this - some might exist. We tried to solve it using computer algrebra system (CAS) (Maple), but unfortunately it was also unable to determine a solution.

It seems that we might be in a situation where the normalization constant is very hard to determine. This was exactly the motivation behind using MCMC, so maybe we should try to do that?

3d

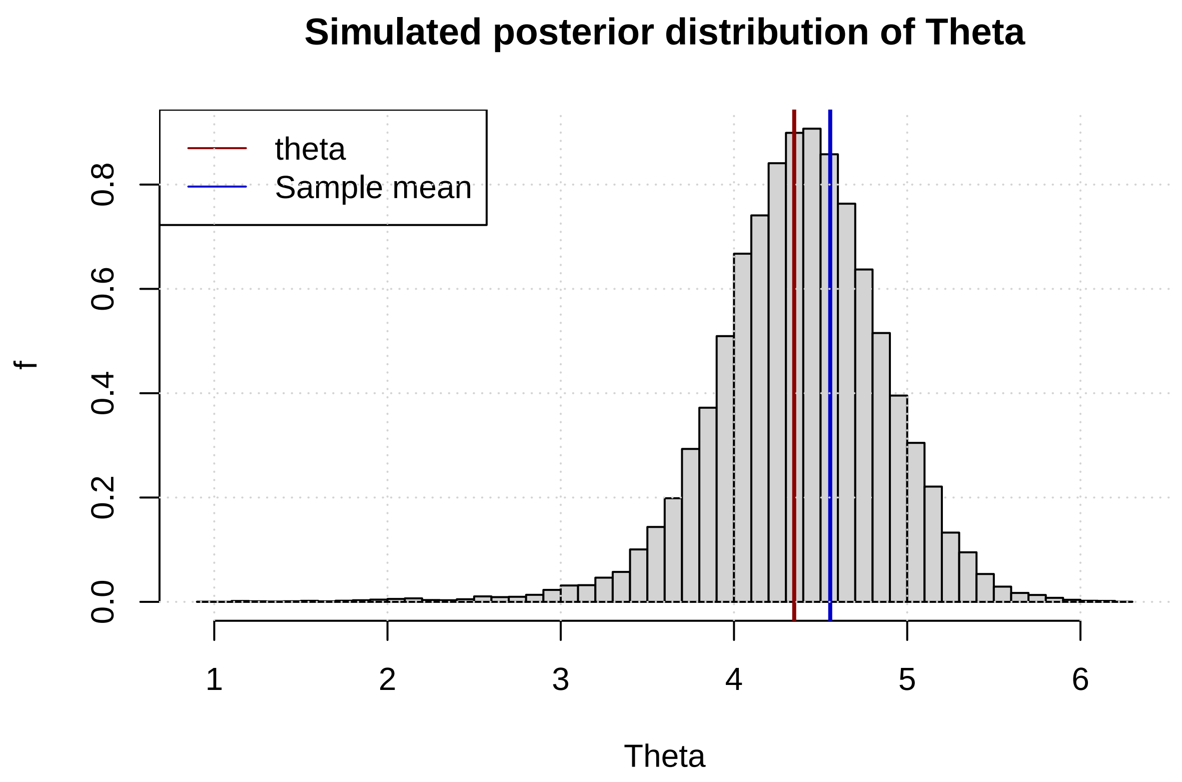
We now want to generate MCMC samples from the posterior distribution of (Θ,Ψ) using the Metropolis Hastings method. We start by implementing a continous Metropolis Hastings using random walk. As symmetric distribution we will use a scaled t distribution with 10 degrees of freedom (should be better as investigating the tails than a simple normal). Notice also that we have implemented the algorithm using a log transform of the density. The reason is that we had some numerical issues for $n = 1000$ which we suspected was caused by catastrophic cancelation.

```r
## Random Walk Metropolis-Hastings
MH.C <- function(x0, n, g){
  X = matrix(NA, nrow = n, ncol = length(x0))
  X[1,] = x0
  y = numeric(length(x0))

  for (ii in 2:n) {
    X[ii,] = X[ii-1,]
    for (jj in 1:length(x0)) {
      y = X[ii,]
      y[jj] = X[ii-1,jj] + rt(1,10)/15
      #X[ii,jj] = ifelse((g(y) >= g(X[ii-1,])) || (runif(1)<(g(y)/g(X[ii-1,]))), y[jj], X[ii-1,jj])
      X[ii,jj] = ifelse((g(y) >= g(X[ii-1,])) || (runif(1)< exp(g(y) - g(X[ii-1,]))), y[jj], X[ii-1,jj]
    }
  }
  X
}
```

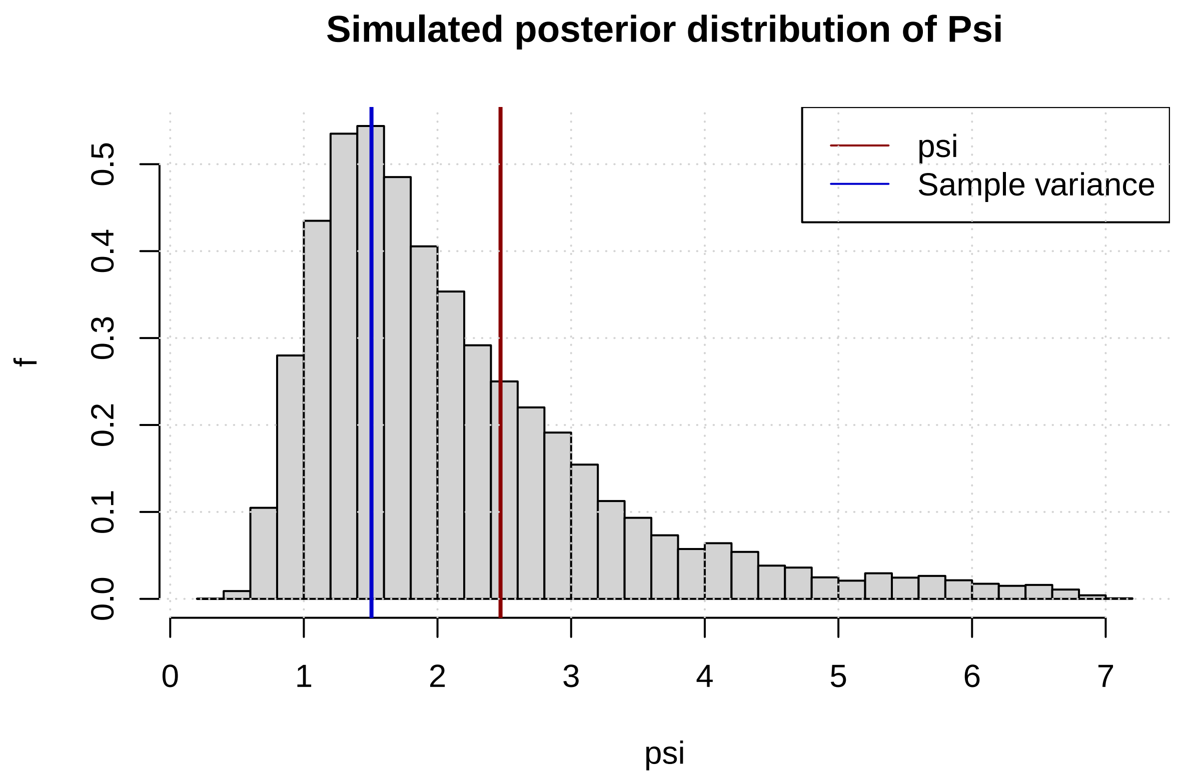
We can now use

$$g_{(\Theta,\Psi)}(\theta,\psi) = \frac{1}{c} f_{\boldsymbol{X}|(\Theta,\Psi)=(\theta,\psi)}(\boldsymbol{x}) f_{(\Theta,\Psi)}(\theta,\psi).$$

```
fx.cond <- function(X, theta, psi) sum(log(1/sqrt(2*psi*pi) * exp(-(X-theta)^2/(2*psi))))
ftp <- function(theta,psi){
  log(1/(2*pi*theta*psi*sqrt(3/4))*exp(-(log(theta)^2 - log(theta)*log(psi) + log(psi)^2)/(3/2)))
}
g <- function(X, param) {
  ifelse( all(param>0), fx.cond(X,param[1],param[2]) + ftp(param[1],param[2]), 0)
}

set.seed(1)
sim = MH.C(x0 = c(1,1), n = 100000, function(param) g(X, param) )

hist(sim[,1], main = "Simulated posterior distribution of Theta", xlab = "Theta",
     ylab = "f", freq = F, breaks = 40)
abline(v=theta, lwd = 2, col = 'red4')
abline(v=mean(X), lwd = 2, col = 'blue3')
legend('topleft', c('theta', 'Sample mean'), lty = 1, col = c('red4','blue3'))
grid()
```



```r
hist(sim[,2], main = "Simulated posterior distribution of Psi", xlab = "psi",
     ylab = "f", freq = F, breaks = 40)
abline(v=psi, lwd = 2, col = 'red4')
abline(v=var(X), lwd = 2, col = 'blue3')
legend('topright', c('psi', 'Sample variance'), lty = 1, col = c('red4','blue3'))
grid()
```



We have now simulated posterior distributions for Θ and Ψ. Notice that they both have their mode very close to the sample mean and variance respectively.

3e

We will now repeat what we did in the previous exercise, but now with $n = 100$ and $n = 1000$.

```r
print("n = 100...")
```

```
## [1] "n = 100..."
```

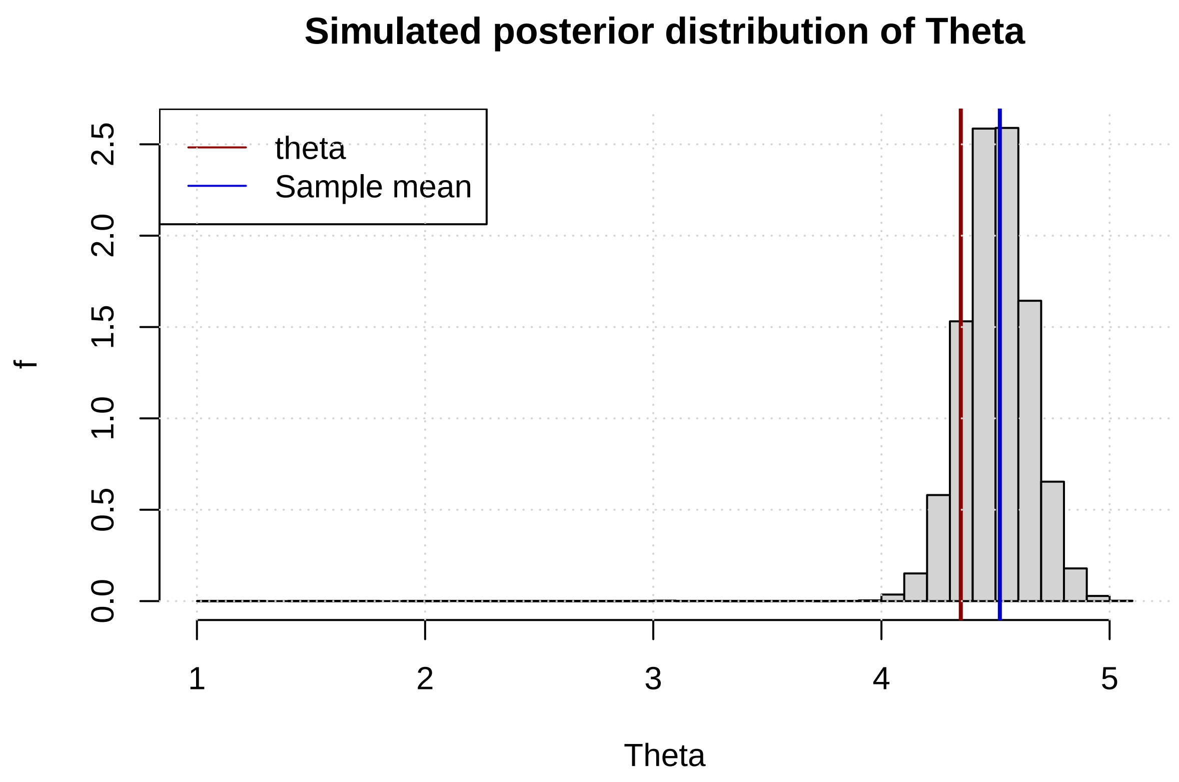
```r
set.seed(1)
n = 100
X = rnorm(n, mean = theta, sd = sqrt(psi))
print(paste('Sample mean:', mean(X)))
```

```
## [1] "Sample mean: 4.51883464417079"
```

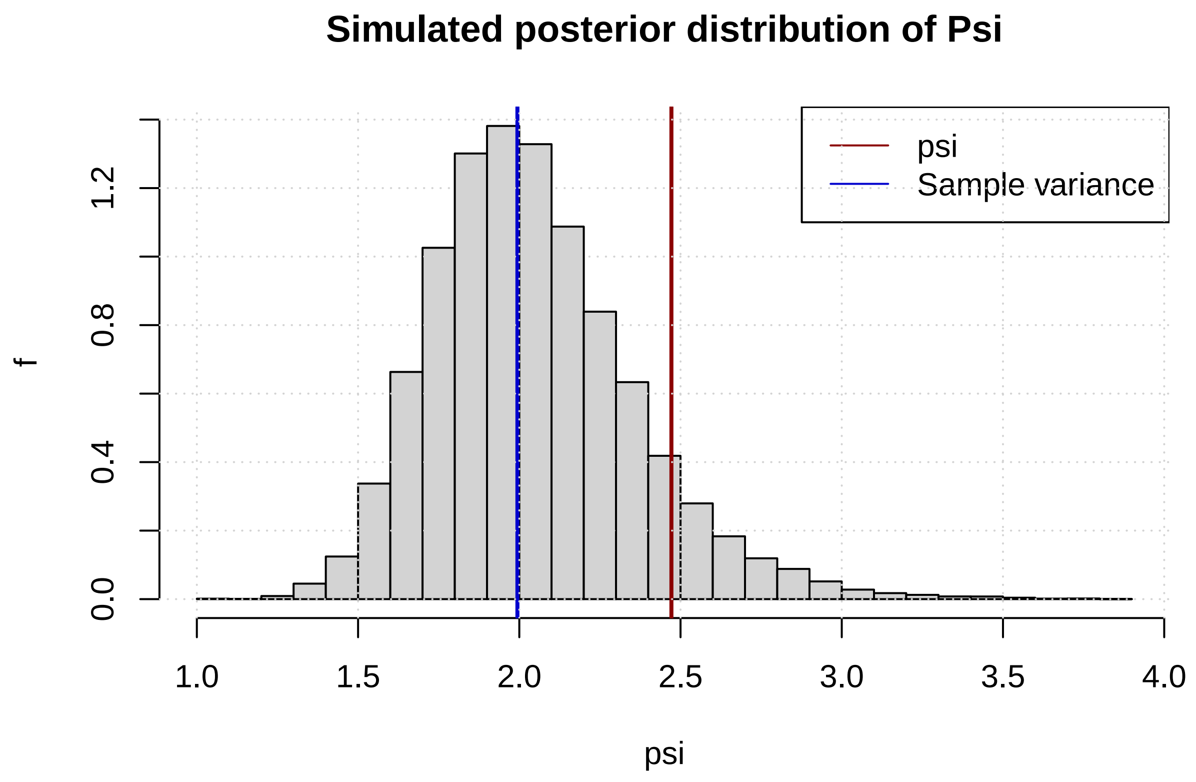
```r
print(paste('Sample variance:', var(X)))
```

```
## [1] "Sample variance: 1.99413285076186"
```

```r
sim = MH.C(x0 = c(1,1), n = 100000, function(param) g(X, param) )

hist(sim[,1], main = "Simulated posterior distribution of Theta", xlab = "Theta",
     ylab = "f", freq = F, breaks = 40)
abline(v=theta, lwd = 2, col = 'red4')
abline(v=mean(X), lwd = 2, col = 'blue3')
legend('topleft', c('theta', 'Sample mean'), lty = 1, col = c('red4','blue3'))
grid()
```



```r
hist(sim[,2], main = "Simulated posterior distribution of Psi", xlab = "psi",
     ylab = "f", freq = F, breaks = 40)
abline(v=psi, lwd = 2, col = 'red4')
abline(v=var(X), lwd = 2, col = 'blue3')
legend('topright', c('psi', 'Sample variance'), lty = 1, col = c('red4','blue3'))
grid()
```



```r
print("n = 1000...")
```

```
## [1] "n = 1000..."
```

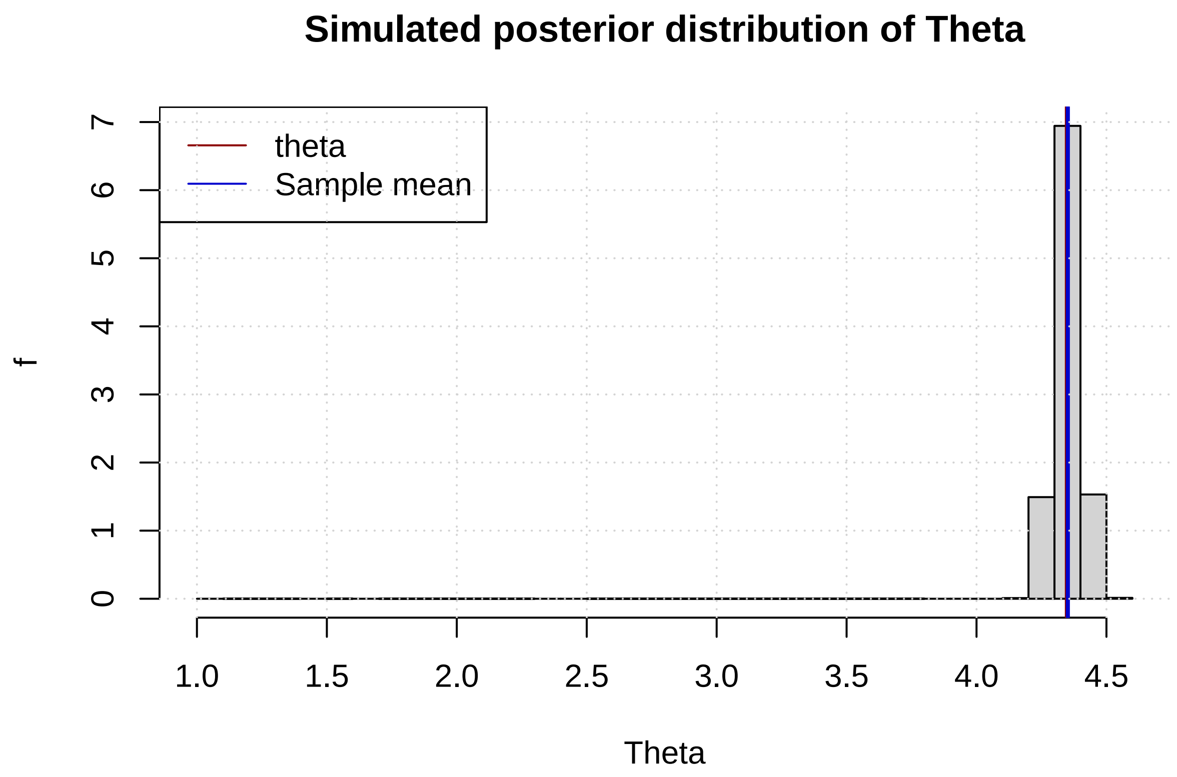
```r
n = 1000
X = rnorm(n, mean = theta, sd = sqrt(psi))
print(paste('Sample mean:', mean(X)))
```

```
## [1] "Sample mean: 4.35155347562585"
```

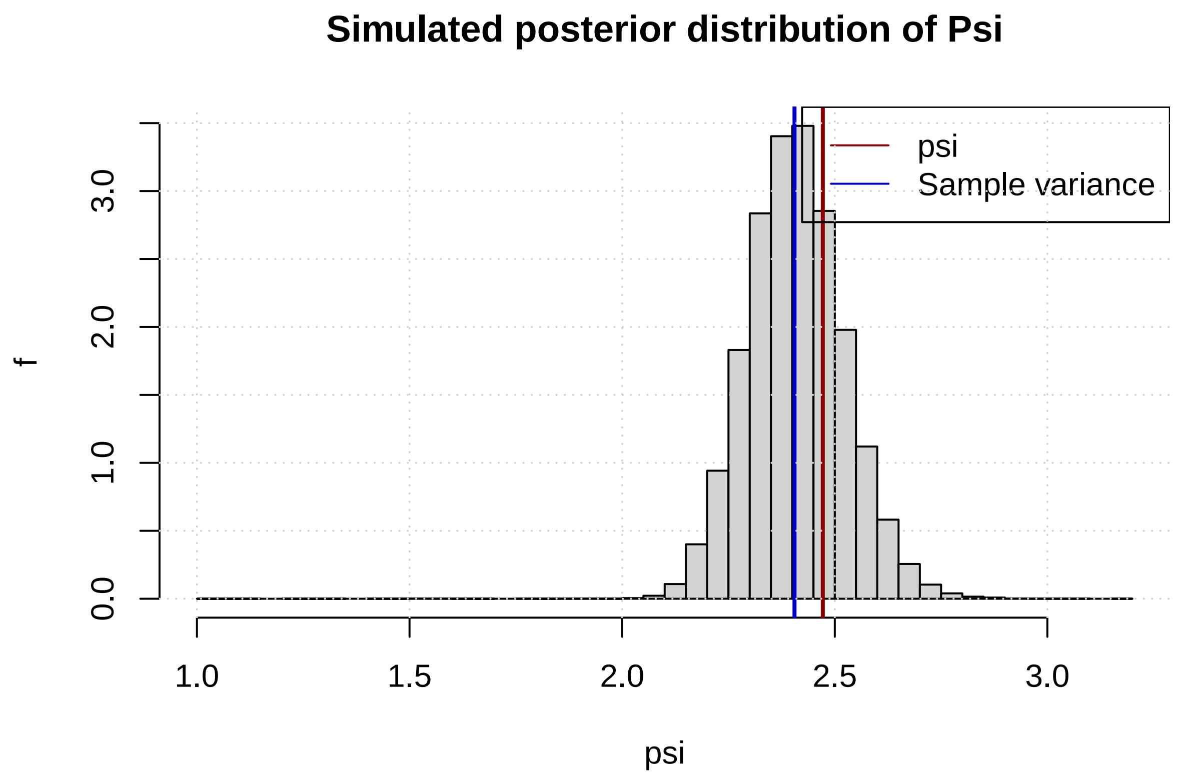
```r
print(paste('Sample variance:', var(X)))
```

```
## [1] "Sample variance: 2.40533526372934"
```

```r
sim = MH.C(x0 = c(1,1), n = 100000, function(param) g(X, param) )

hist(sim[,1], main = "Simulated posterior distribution of Theta", xlab = "Theta",
     ylab = "f", freq = F, breaks = 40)
abline(v=theta, lwd = 2, col = 'red4')
abline(v=mean(X), lwd = 2, col = 'blue3')
legend('topleft', c('theta', 'Sample mean'), lty = 1, col = c('red4','blue3'))
grid()
```



```r
hist(sim[,2], main = "Simulated posterior distribution of Psi", xlab = "psi",
     ylab = "f", freq = F, breaks = 40)
abline(v=psi, lwd = 2, col = 'red4')
abline(v=var(X), lwd = 2, col = 'blue3')
legend('topright', c('psi', 'Sample variance'), lty = 1, col = c('red4','blue3'))
grid()
```



As expected, when we increase n, the sample mean and variance will be closer to the underlying parameter, i.e., θ and ψ. Furthermore, the increased number of samples means that the prior means "less" relatively speaking. Therefore, we see a increasingly narrow posterior distribution.

Computer exerices 7

In this exericse we will investigate simulated annealing.

1 a

We start by implementing a traveling salesman problem (TSP) solver using simulated annealing. Initially, we consider the input given as a list of coordinates in 2d. We assume the distance between two points are at all time given as their Euclidian distance. For cooling function we use

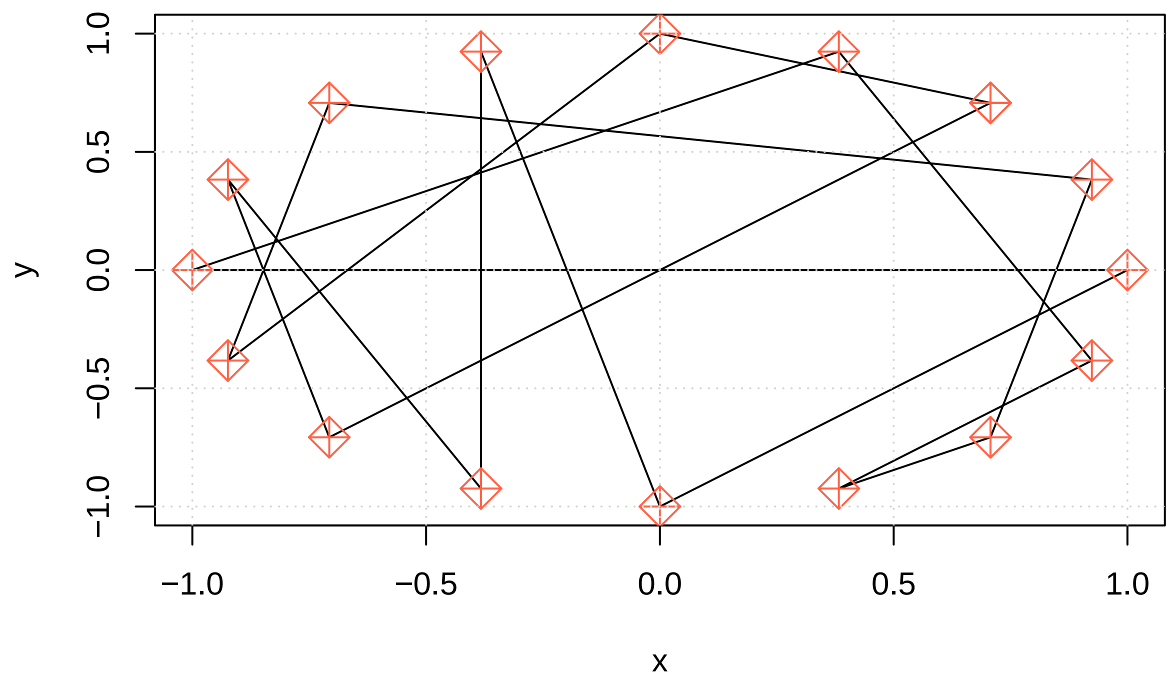
$$T_k = 1/\log k + 1$$

as given in the lecture slides.

```r
euclid.dist <- function(c1,c2){
  sqrt((c1$x-c2$x)^2 + (c1$y-c2$y)^2)
}
cost <- function(coord, route){
  sum(sapply(1:length(coord), function(ii) euclid.dist(coord[[route[ii]]], coord[[route[ii+1]]])))
}
T.fun <- function(k){
  #1/(sqrt(1+k)/10)
  1/log(k+1)
}

TSP.euclid <- function(coord, N) {
  ## init route
  route = matrix(NA,nrow = N, ncol = length(coord)+1)
  route[1,] = c(1:length(coord),1)

  for (ii in 2:N) {
    # propose route
    route.new = route[ii-1, ]
    while ( (!any(route[ii-1, ] != route.new)) ){
      perm = sample.int(length(coord)-1,2) + 1
      route.new[perm[1]] = route[ii-1, perm[2]]
      route.new[perm[2]] = route[ii-1, perm[1]]
    }

    ## accept/reject
    route[ii, ] = `if`( cost(coord,route.new)<cost(coord,route[ii-1,]) || (exp(-(cost(coord,route.new) -
                        route.new, route[ii-1, ])
  }
  route
}
```

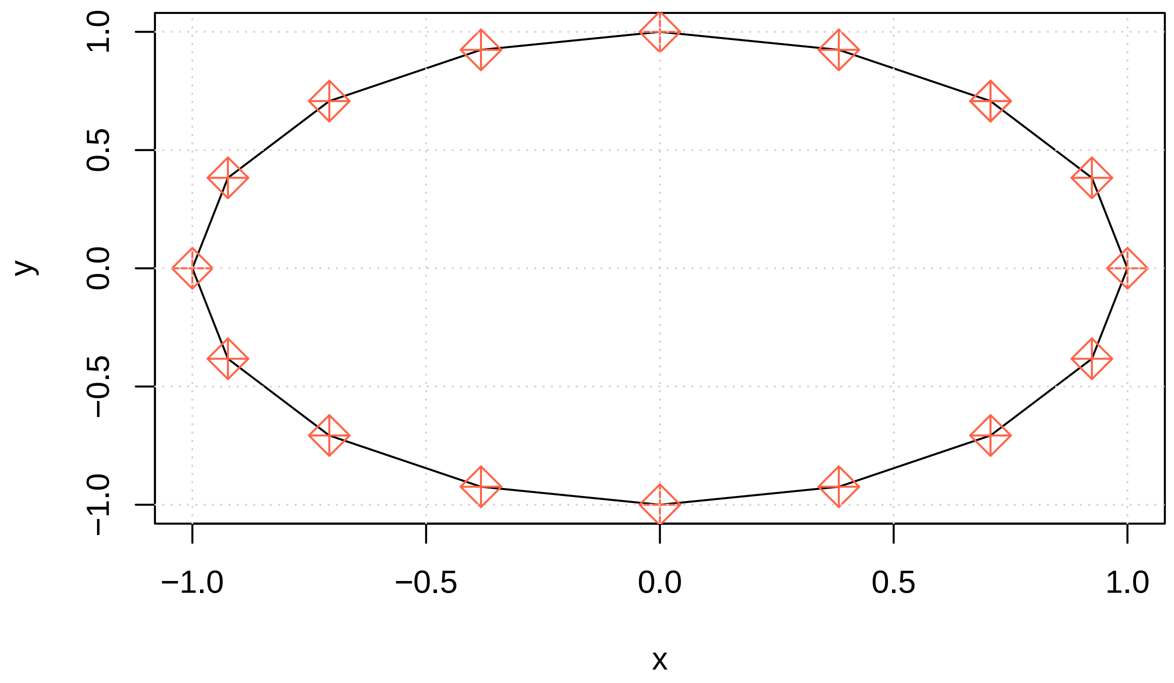
To test our implemenetation, we generate coordinates on a unit circle and shuffle their order.

```r
## circle TSV
set.seed(12)
m = 16
coord = lapply(1:m, function(i) list(x=cos(2*pi*i/m), y=sin(2*pi*i/m)))
coord = lapply(sample(m), function(i) coord[[i]]) # shuffle
route = c(1:m,1)

plot(sapply(route, function(i) coord[[i]]$x ),
     sapply(route, function(i) coord[[i]]$y ), type = 'l', xlab = 'x', ylab = 'y')
points(sapply(route, function(i) coord[[i]]$x ),
       sapply(route, function(i) coord[[i]]$y ), pch = 9, col = 'tomato', cex = 2)
grid()
```



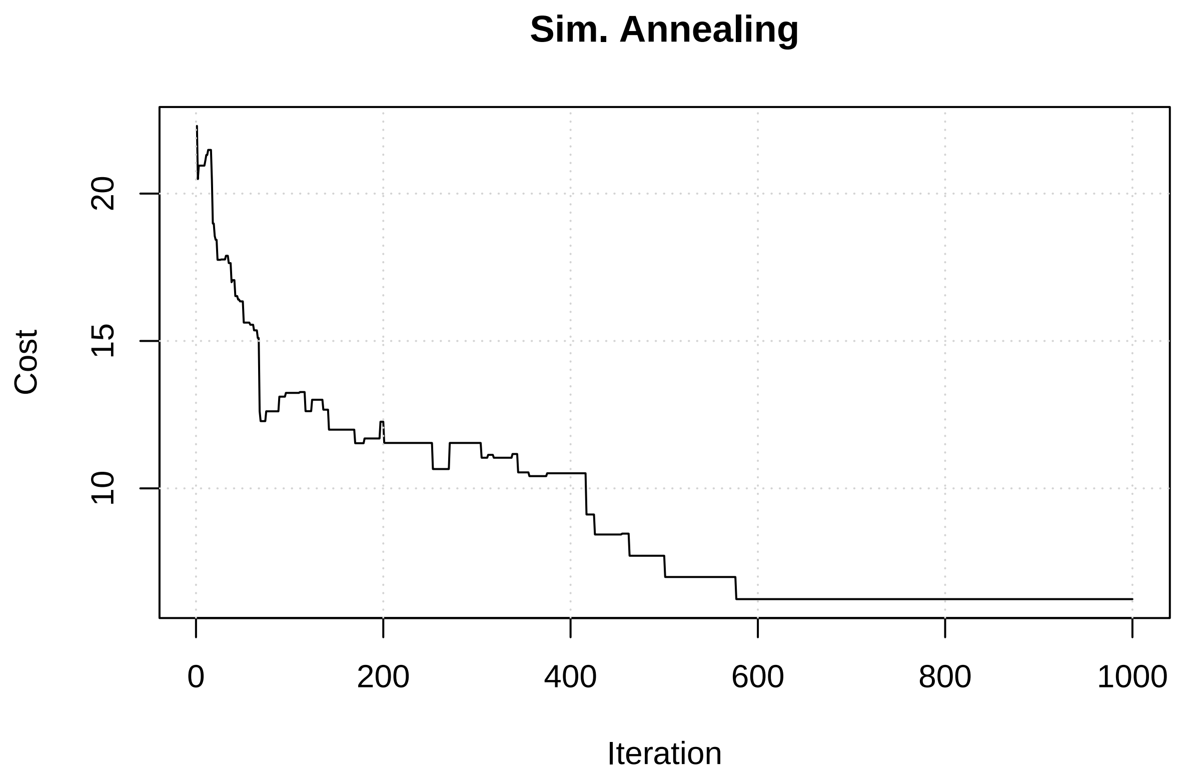
As we can see the order of the coordinates are shuffled, and hence the route mapped out is quite bad. We now want to find the shortest route that passes through all points. Below we can see the final state of simulated annealing. Notice that we have completely ordered the coordinates.

```
## Solve TSP
set.seed(60)
N = 1000
sim.routes = TSP.euclid(coord, N)
plot(sapply(sim.routes[N,], function(i) coord[[i]]$x ),
     sapply(sim.routes[N,], function(i) coord[[i]]$y ), type = 'l', xlab = 'x', ylab = 'y')
points(sapply(route, function(i) coord[[i]]$x ),
       sapply(route, function(i) coord[[i]]$y ), pch = 9, col = 'tomato', cex = 2)
grid()
```



To investigate how the problem converges, we can look at the total cost, i.e., total distance travelled as a function of iterations. As can be seen from the plot below, the decrease in distance travelled comes both in the form of small and large steps, corresponding to smaller and larger untanglements.

```
plot(1:N, apply(sim.routes,1,function(x) cost(coord, x)), type = 'l',
     xlab = 'Iteration', ylab = 'Cost', main = 'Sim. Annealing')
grid()
```



1 b

We now turn our attention to work with costs directly and apply it to the cost matrix from the course homepage.

The matrix is given by

```r
mat = matrix(c(0,225,110,8,257,22,83,231,277,243,94,30,4,265,274,250,87,83,271,86,
               255,0,265,248,103,280,236,91,3,87,274,265,236,8,24,95,247,259,28,259,
               87,236,0,95,248,110,25,274,250,271,9,244,83,250,248,280,29,26,239,7,
               8,280,83,0,236,28,91,239,280,259,103,23,6,280,244,259,95,87,230,84,
               268,87,239,271,0,244,275,9,84,25,244,239,275,83,110,24,274,280,84,274,
               21,265,99,29,259,0,99,230,265,271,87,5,22,239,236,250,87,95,271,91,
               95,236,28,91,247,93,0,247,259,244,27,91,87,268,275,280,7,8,240,27,
               280,83,250,261,4,239,230,0,103,24,239,261,271,95,87,21,274,255,110,280,
               247,9,280,274,84,255,259,99,0,87,255,274,280,3,27,83,259,244,28,274,
               230,103,268,275,23,244,264,28,83,0,268,275,261,91,95,8,277,261,84,247,
               87,239,9,103,261,110,29,255,239,261,0,259,84,239,261,242,24,25,242,5,
               30,255,95,30,247,4,87,274,242,255,99,0,24,280,274,259,91,83,247,91,
               8,261,83,6,255,29,103,261,247,242,110,29,0,261,244,230,87,84,280,100,
               242,8,259,280,99,242,244,99,3,84,280,236,259,0,27,95,274,261,24,268,
               274,22,250,236,83,261,247,103,22,91,250,236,261,25,0,103,255,261,5,247,
               244,91,261,255,28,236,261,29,103,9,242,261,244,87,110,0,242,236,95,259,
               84,236,27,99,230,83,7,259,230,230,22,87,93,250,255,247,0,9,259,24,
               91,242,28,87,250,110,6,271,271,255,27,103,84,250,271,244,5,0,271,29,
               261,24,250,271,84,255,261,87,28,110,250,248,248,22,3,103,271,248,0,236,
               103,271,8,91,255,91,21,271,236,271,7,250,83,247,250,271,22,27,248,0),20,20)
```

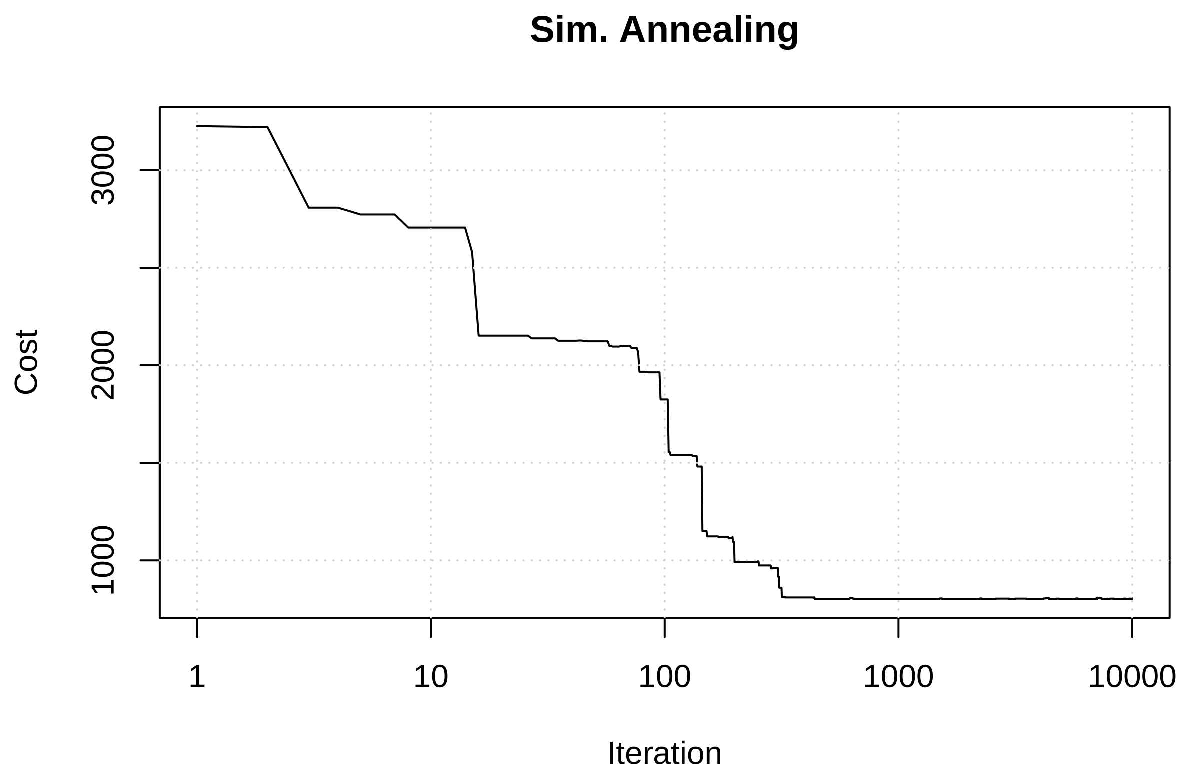
To work solve the TSP with a cost matrix, we have to modify our code a bit, specifically, we modify the cost function. Addtionally, we have some issues regarding premature convergence, i.e., the cooling function is too aggresive. Therefore we modify the cooling function also, and now use

$$T_k = 10/\log(k+1).$$

```
T.fun <- function(k){
  #1/(sqrt(1+k))
  #1/log(sqrt(k+1))
  10/log(k+1)
}
#### Now with matrix
cost.mat <- function(mat, route){
  sum( sapply(1:ncol(mat), function(i) mat[ route[i], route[i+1] ] ))
}

TSP.matrix <- function(mat, N) {
  ## init route
  route = matrix(NA,nrow = N, ncol = ncol(mat)+1)
  route[1,] = c(1:ncol(mat),1)
  ii = 2
  for (ii in 2:N) {
    # propose route
    route.new = route[ii-1, ]
    while ( (!any(route[ii-1, ] != route.new)) ){
      perm = sample.int(ncol(mat)-1,2) + 1
      route.new[perm[1]] = route[ii-1, perm[2]]
      route.new[perm[2]] = route[ii-1, perm[1]]
    }

    ## accept/reject
    route[ii, ] = `if`( cost.mat(mat,route.new)<cost.mat(mat,route[ii-1,]) ||
        (exp(-(cost.mat(mat,route.new) - cost.mat(mat,route[ii-1, ]))/T.fun(ii))>runif(1)),
        route.new, route[ii-1, ])
  }
  route
}
```

We can now solve the TSP. It is now a bit difficult to graphically show before/after, so we just look at the effect on the total cost. In our simulation, we find the minimum cost to be 802. Since simulated annealing is method of probabilitistic optimization, it is impossible to tell if this stay around this area for a while, so the actual min is most likely in this vecinity.

```
set.seed(42)
N = 10000
sim.routes = TSP.matrix(mat, N)
plot(1:N, apply(sim.routes,1,function(x) cost.mat(mat, x)), type = 'l',
   xlab = 'Iteration', ylab = 'Cost', main = 'Sim. Annealing', log = 'x')
grid()
```



```r
print(paste('Min cost:', min(apply(sim.routes,1,function(x) cost.mat(mat, x)))) )
```

```
## [1] "Min cost: 802"
```

Computer exericse 8

13

We are given n independent and identically distributed random variables, $X_1, ..., X_n$, with unknown mean, μ. For given constants $a < b$, we are interested in estimating

$$p = P\{a < \sum_{i=1}^{n} X_i/n - \mu < b\}.$$

a

We want to solve this by means of bootstrapping for realizations of X and constants a, b. To do so we draw m distributions "like" X with (!) replacement, i.e., what could have been. For each draw, R_j, we compute the mean $Y = E[R_j]$. We then generate a new vector given by the difference between the sample mean and the mean of the sample means, $Z = Y - E[Y]$. We can now find p by

$$p = \frac{\sum 1_{a<z_i<b}}{|Z|}.$$

We do this k times to achive k estimates of p, which we expect to follow an asymptotic normal distribution.

b

We are now given a realization of X and constants a, b. We then find P by means of bootstrapping

```r
set.seed(1)
X = c(56, 101, 78, 67, 93, 87, 64, 72, 80, 69)
a = -5
b =  5
m = 10
k = 1000

p = rep(NA,k)
for (ii in 1:k) {
  Y = sapply(1:m, function(j)  mean(sample(X, replace = T)))
  Z = Y - mean(Y)
  p[ii] = sum(Z>a & Z<b) / length(Z)
}

mean(p) + sqrt(var(p)/length(p))*qt(0.975,k)*c(-1,1)
```

```
## [1] 0.7762833 0.7935167
```

On a 95% confidence level, we find

$$p = [0.7763 \quad 0.7935].$$

15

We now wish to estimate $Var(S^2)$. To do so, we use the same approach as in the previous exercise.

```r
set.seed(1)
X = c(5,4,9,6,21,17,11,20,7,10,21,15,13,16,8)

m = 15
k = 1000

s = rep(NA,k)
for (ii in 1:k) {
  Y = sapply(1:m, function(j) var(sample(X, replace = T)))
  s[ii] = var(Y)
}
mean(s) + sqrt(var(s)/length(s))*qt(0.975,k)*c(-1,1)
```

```
## [1] 56.67867 59.29858
```

On a 95% confidence level, we find

$$Var(S^2) = [56.6859.30].$$

3

In this final exercise we want to estimate the variance of means and medians of a pareto distribution with parameters, $\beta = 1$ and $k = 1.05$. We immediately notice that k is close to 1, therefore we expect to have very fat tails. To get good estimates we would therefore need to do very many draws from the distribution.

We are specifically asked to do 200 draws from the distribution (which is far to few), and compute the estimates from these by means of bootstrapping (with 100 replicates).

```r
library(EnvStats)
```

```
##
## Vedhæfter pakke: 'EnvStats'

## De følgende objekter er maskerede _af_ '.GlobalEnv':
##
##     dpareto, ppareto

## Det følgende objekt er maskeret fra 'package:MASS':
##
##     boxcox

## De følgende objekter er maskerede fra 'package:stats':
##
##     predict, predict.lm

## Det følgende objekt er maskeret fra 'package:base':
##
##     print.default
```

```r
set.seed(1)
m = 200
X = rpareto(m, 1, 1.05)
k = 100

var.mean = rep(NA,k)
var.median = rep(NA,k)
for (ii in 1:k) {
  b.mean = sapply(1:m, function(j) mean(sample(X, replace = T)))
  b.median = sapply(1:m, function(j) median(sample(X, replace = T)))

  var.mean[ii] = var(b.mean)
  var.median[ii] = var(b.median)
}

mean(var.mean) + sqrt(var(var.mean)/length(var.mean))*qt(0.975,k)*c(-1,1)
```

```
## [1] 0.8254713 0.8581198
```

```r
mean(var.median) + sqrt(var(var.median)/length(var.median))*qt(0.975,k)*c(-1,1)
```

```
## [1] 0.01841746 0.01945409
```

On a 95% confidence level, we find

$$var(\mu) = [0.8255 \quad 0.8581]$$
$$var(median) = [0.01842 \quad 0.01945].$$

Generally, we see that if we repeat the whole thing, the mean is much more sentitive to the initial draws than the median. The reason is that while a very large draw (from the fat tail) has a large impact on the mean, it does not have so on median.