642_HW2

Aidan Lytle

2022-09-24

Problem 1

The CDF of the two-parameter exponential is $F_X(x) = 1 - e^{-\lambda(x-v)}$; $x \geq v$, hich gives us $-log(F_X(x) - 1) = \lambda(x - v)$. So $x = -\frac{log(F_X(x)-1)}{\lambda} + v$, and the inverse becomes $F_X^{-1}(U) = -\frac{log((U)-1)}{\lambda} + v$. The function becomes (using the tolerance package's r2exp):

```
exponDist = function(lmb,v,n){
return(r2exp(n, rate = 1/lmb, shift = v))
}
```

where r2exp is the 2 parameter exponential family. The quantiles for the sample are

```
test = exponDist(1,2,500)
print(quantile(test))
```

```
##       0%      25%      50%      75%     100%
## 2.004224 2.284822 2.724758 3.507077 8.959868
```

While the theoretical quantiles are $F^{-1}(p, \lambda, v) = v - \frac{ln(1-x)}{\lambda}$ which give us (for lambda = 1, v = 2)

```
q0 = 2
q1 = 2-(log(1 - (1/4))/2)
q2 = 2-(log(1-(1/2))/2)
q3 = 2-(log(1-(3/4))/2)
q4 = 2 - (log(1-1)/2)
q0
```

```
## [1] 2
```

```
q1
```

```
## [1] 2.143841
```

```
q2
```

```
## [1] 2.346574
```

```
q3
```

```
## [1] 2.693147
```

```
q4
```

```
## [1] Inf
```

This matches well, because the true quantiles cannot be infinite for a finite sample. The theoretical quantiles match very closely to the experimental ones in the non-limiting regime.

Problem 2

We see that $F(x)$ can be given by the table

```
library(knitr)
tab <- matrix(c(0,1,2,3,4, .1, .3, .5, .7, 1), ncol = 5, byrow= TRUE, )
colnames(tab) <- c("___","___","___","___","_")
rownames(tab) <- c('x','F(x)')
tab<-as.table(tab)
kable(tab, caption = "cdf(x)")
```

Table 1: cdf(x)

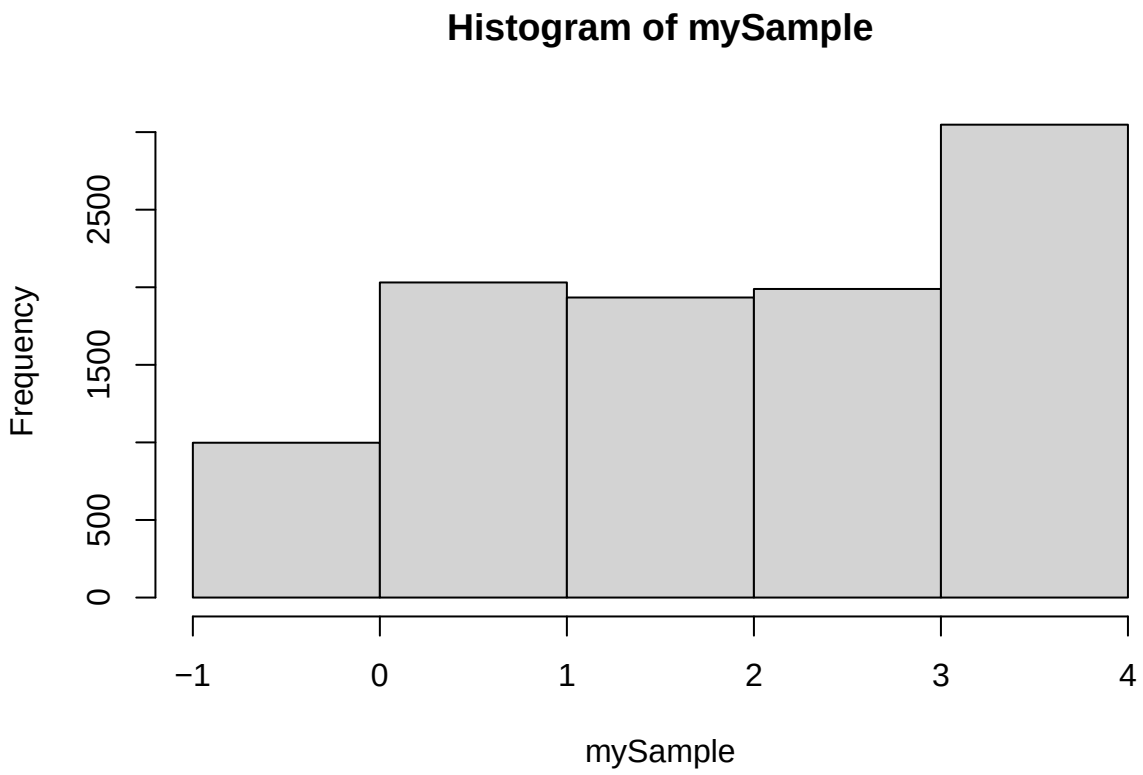
	___	___	___	___	_
x	0.0	1.0	2.0	3.0	4
F(x)	0.1	0.3	0.5	0.7	1

Then the function needs to see what the ceiling of u is w/r/t to the cdf:

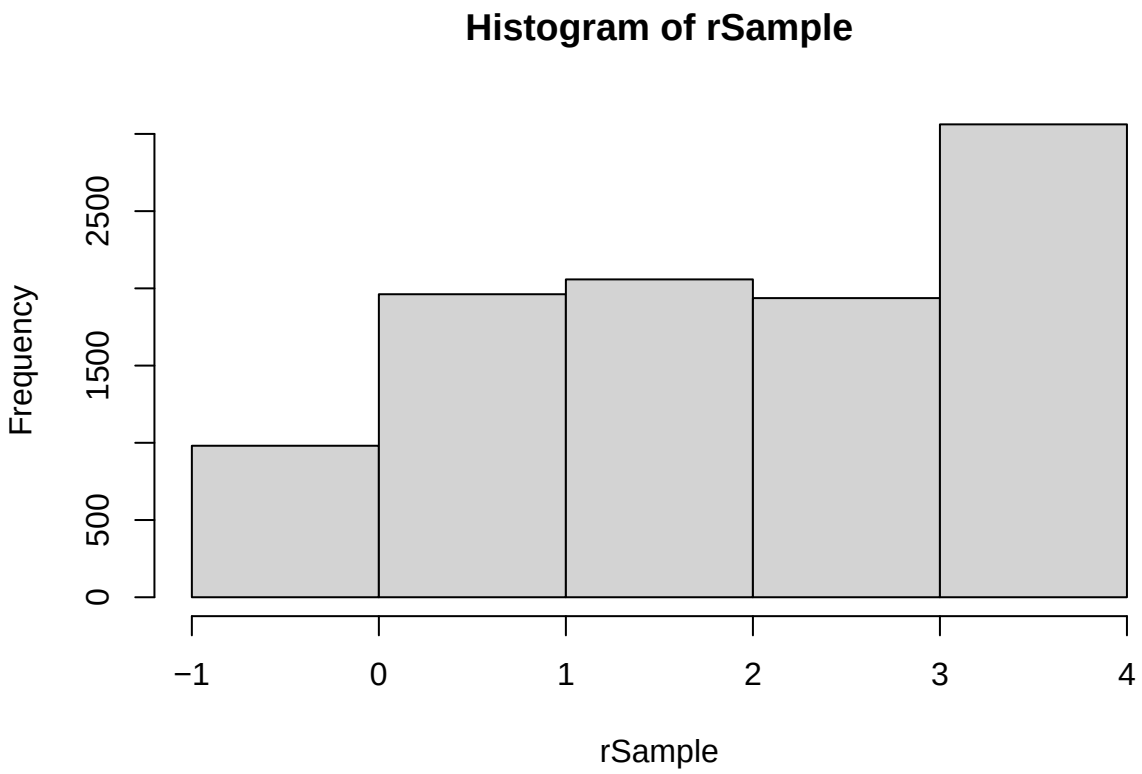
```
inverseDisc = function(u)
{
  if (u <= .1 ){
    return(0)
  }
  else if (.1 < u && u <= .3){
      return(1)
  }
  else if (.3 < u && u <= .5){
    return(2)
  }
  else if (.5 < u && u <= .7){
    return(3)
  }
  else if(.7 < u && u <= 1){
    return(4)
  }
}
```

Then generating the sample using *sample*:

```
mySample = runif(10000)
for (i in 1:length(mySample)){
  mySample[i] = inverseDisc(mySample[i])
}
rSample = sample(c(0,1,2,3,4), size = 10000, replace = TRUE, prob=c(.1,.2,.2,.2,.3))
hist(mySample, breaks = seq(from = -1, to = 4, by = 1))
```




```
hist(rSample, breaks = seq(from = -1, to = 4, by = 1))
```



Then we can compare the three tables/pdfs. It is clear that the empirical probabilities are not exactly what the table predicts, however, they are extremely close. There is a slight uptick of points in the center of each distribution, for reasons I don't really understand, but I assume they have to do with the distribution of pseduo-random numbers from this particular seed and would change given other seeds.

Problem 3

We use the inverse method. The CDF of $LogN(0,1)$ is $\frac{1}{2}\left[1+erf\left(\frac{log(x)-\mu}{\sqrt{2}\sigma}\right)\right]$, so the inverse method returns $\mu+\left(\sqrt{2}\sigma erf^{-1}(2F(u)-1)\right)$. Then

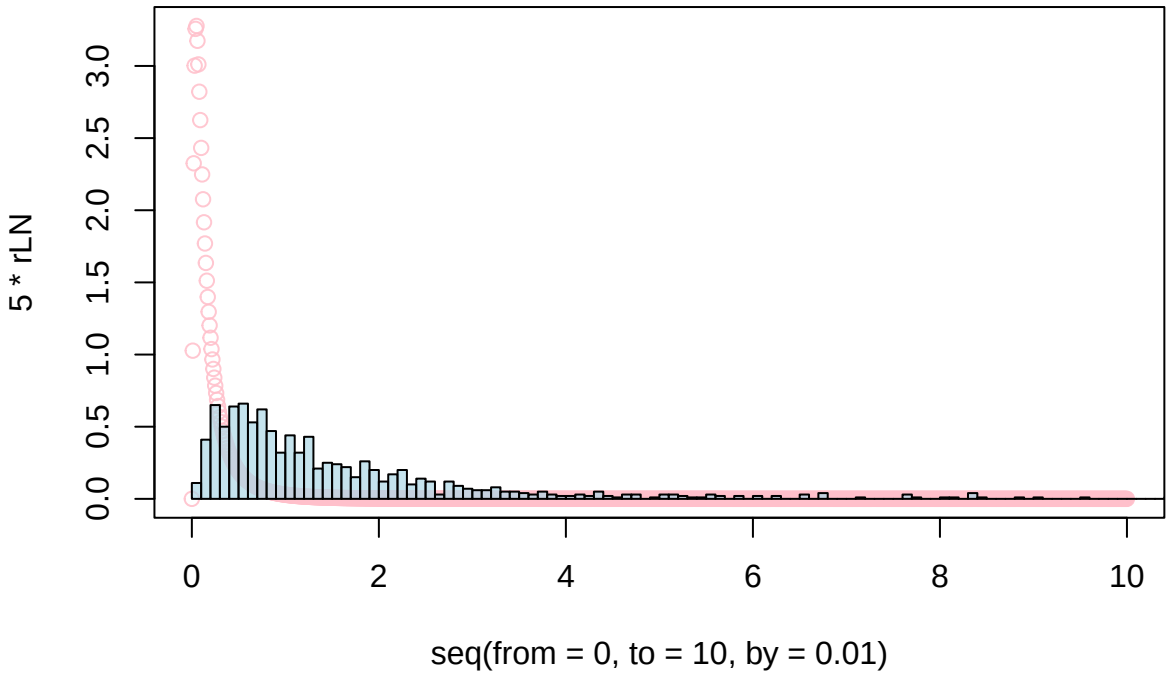
```
fInv = function(x,mu,sig){
  return(exp(mu + sqrt(2)*sig*erfinv(2*x - 1)))
}
u = runif(1000)
for (i in 1:1000){
  u[i] = fInv(u[i], mu = 0, sig = 1)
}

c1 = rgb(173,216,230,max = 255, alpha = 80, names = "lt.blue")
c2 = rgb(255,192,203, max = 255, alpha = 80, names = "lt.pink")

exper = hist(u, breaks = seq(from = 0, to = 80, by = .1), plot = FALSE)
rLN = dlnorm(seq(0,80, by = .08), meanlog = 0, sdlog = 1)
```

```
c1 = rgb(173,216,230,max = 255, alpha = 180, names = "lt.blue")
c2 = rgb(255,192,203, max = 255, alpha = 225, names = "lt.pink")


plot(seq(from = 0, to = 10, by = .01),5*rLN, col = c2)
plot(exper, col = c1, freq = FALSE, add = TRUE)
```



The comparison of the output of the inverse method with the theoretical pdf is quite stark in its difference. The peaks do not match, nor do the distribution's skew etc. It is not clear that the curves are even the same function! The reason for this difference is likely the floating point error introduced in the uniform distribution which we are inverting, which prevents the smaller values from being output. If we force the memory or datatypes to all be double, long, or longlong, it is likely that the empirical distribution would converge more closely to the pdf.

Problem 4

First, we find the mean of each column, and the sd, so that we can normalize by column. Because of the linearity of the expectation operator, this will return the desired mean for the whole set. Similarly, the standard devation will be uniform across the data, so the standard deviation will remain unchanged (as the variance is linear in the square, so the mean of the individual sqaured distance from the mean will be 1 if all standard deviations are 1).

```
#calculate the means and standard deviations for the datasets in the columns
Scores = scor
muMec = mean(scor[,1])
muVec = mean(scor[,2])
muAlg = mean(scor[,3])
muAna = mean(scor[,4])
muSta = mean(scor[,5])

mu = c(muMec, muVec, muAlg, muAna, muSta)

stddMec = sd(scor[,1])
stddVec = sd(scor[,2])
stddAlg = sd(scor[,3])
stddAna = sd(scor[,4])
stddSta = sd(scor[,5])
```

Next, we find the matrix of the z valued datasets:

```
sdd = c(stddMec, stddVec, stddAlg, stddAna, stddSta)

for (i in 1:length(scor[,1]))
{
  for (j in 1:5)
  {
    Scores[,j][i] = (scor[,j][i] - mu[j])/sdd[j]
  }
}
Z = Scores
```

Finally, we find the mean of the bivariate and trivariate sets (which will be one, as discussed above) and the covariance matrix. We can see that up to some floating point errors, the mean and standard deviation of any of the sets are limiting to zero. Thus, the entire dataset is normalized by the simple process outlined above.

```
mu_bi = c(muVec, muMec)
mu_tri = c(muAlg, muAna, muSta)

covMat = cov(Z)


print(covMat)
```

```
##           mec       vec       alg       ana       sta
## mec 1.0000000 0.5534052 0.5467511 0.4093920 0.3890993
## vec 0.5534052 1.0000000 0.6096447 0.4850813 0.4364487
## alg 0.5467511 0.6096447 1.0000000 0.7108059 0.6647357
## ana 0.4093920 0.4850813 0.7108059 1.0000000 0.6071743
## sta 0.3890993 0.4364487 0.6647357 0.6071743 1.0000000
```

```
print(mean(as.vector(unlist(as.vector(Z)))))
```

```
## [1] 3.873312e-17
```

```
print(sd(as.vector(unlist(as.vector(Z)))))
```

```
## [1] 0.9954338
```

and these are what we expected (~0,1).