Bayesian Rethinking: Week 2 Notes

Varun Nayyar

25/04/2019

Conditional Probability and Monte Carlo

Questions a. Hiroshi has two children. If you know one of them was a girl, what's the probability that the other child is a girl? (Counting & Bayes Rule)

Via the counting argument, we have 4 possible ways of having 2 children:

1. BB
2. BG
3. GB
4. GG

Given that 1 of the children is a girl, this means that we have only 3 ways of this scenario happening (since 1 can no longer happen). Hence the probability of the other child being a girl is 1/3

Via Bayes Rule, we have

P(both girls | at least one girl) = P(at least one girl | both girls) * P(both girls) / P(at least one girl)

Now P(at least one girl | both girls) = 1 P(both girls) = 1/4 P(at least one girl) = 3/4

Hence P(both girls | at least one girl) = (1/4) / (3/4) = 1/3

b. Prove this with a monte carlo experiment.

Using simple for loops and if statements

```
N = 1000
atleast1g = 0
bothg = 0
for (i in 1:N){
  c1 = sample(1:2, 1)
  c2 = sample(1:2, 1)
  if (c1 == 1 | c2 == 1){
    atleast1g = atleast1g +1
  }
  if (c1 == 1 & c2 == 1){
    bothg = bothg+1
  }
}

bothg/atleast1g
```

```
## [1] 0.3324538
```

For larger N, I do this using data.frames as for loops are slow in R (and python). However, this may be less intuitive

```
# sample size
N = 100000

# child 1 and child 2.
c1 = sample(1:2, N, replace=TRUE)
```

```
c2 = sample(1:2, N, replace=TRUE)
# girl 1 and boy 2.
# tibbles are data.frames which have nice properties
children = tibble(c1, c2)

# Note the %>% is the pipe operator
# it takes the results of the left
# and passes it to the next function
# so instead of f1(f2(X)), you can do
# X %>% f2 %>% f1
# which is more readable

# number of families with at least 1 girl
atleast1g = children %>% filter(c1==1 | c2==1)
# count the number of families with at least 1 girl
ngirls = nrow(atleast1g)
# count the number of families with both girls
nbothg = atleast1g %>% filter(c1==1 & c2 == 1) %>% nrow
# what's the result?
nbothg/ngirls
```

```
## [1] 0.332907
```

c. Tien has two children. You know that one of them is a daughter, who was born on a Thursday. Simulate this problem to identify the probability that the other child is also a daughter. Make sure to generate a sex, day pair and not just assume independence.

```
# sample size
N = 100000

# child 1 and child 2.
# day1 and day2
c1 = sample(1:2, N, replace=TRUE)
d1 = sample(1:7, N, replace=TRUE)
c2 = sample(1:2, N, replace=TRUE)
d2 = sample(1:7, N, replace=TRUE)
# girl 1 and boy 2.
# tibbles are data.frames which have nice properties
children = tibble(c1, d1, c2, d2)

# number of families with at least 1 girl born on thursday
atleast1g = children %>% filter((c1==1 & d1 == 4)| (c2==1 & d2 ==4))

# left as an exercise for the reader.
```

d. Prove the Monte Hall problem via Monte Carlo simulations.

Also left as an exercise for the reader

Queueing Problem

Simulate a queuing problem, as adapted from Gelman. There are 3 doctors at a clinc, new customers arrive Poisson distributed with lambda ~ 10 minutes. Each person takes anywhere from 5-20 minutes (uniformly distributed) occupying the doctor. The clinic opens at 9am and last patient is allowed in at 4pm.

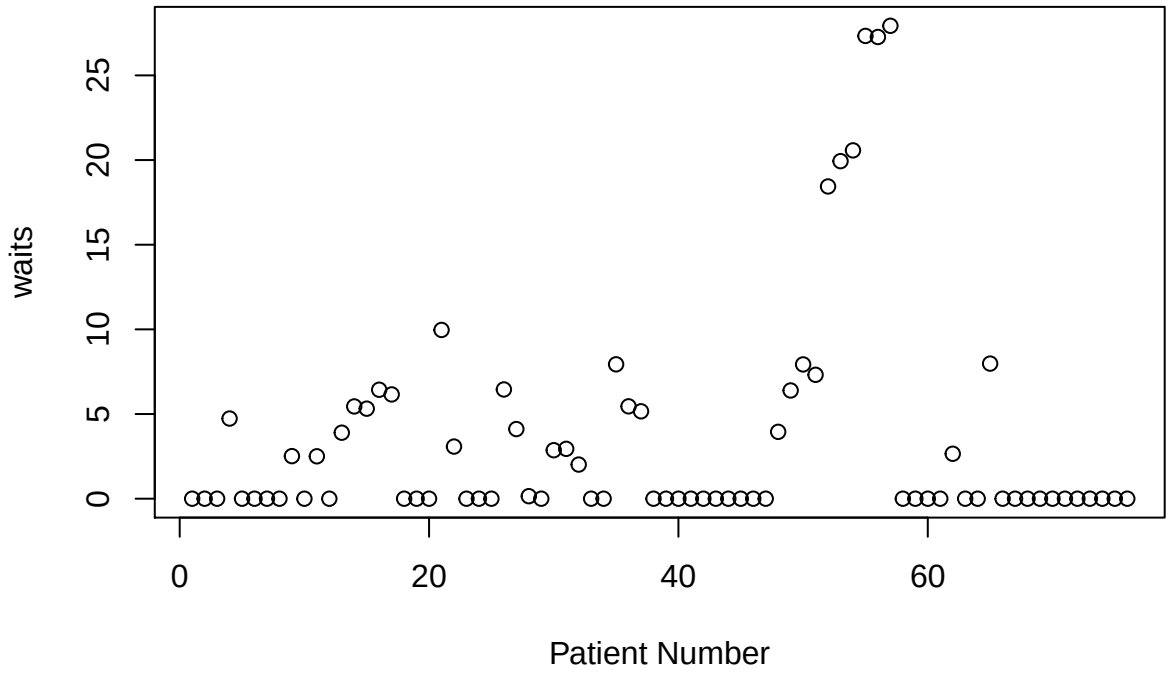
Poisson distributed means the number of events you expect to happen every time perdod. So if patients arrive every hour, this can be modelled as Poisson(6) where the time period is an hour or Poisson(0.1) where the time period is 1 minute. In this case, I'm going to model the time between patients instead, which is exponentially distributed. I.e. time between patients ~ Exp(0.1) in minutes (Note Exp(6) would be time per patient in hours)

a. Simulate this process for a 100 days and identify patient average wait times.

Let's write the code to simluate one day

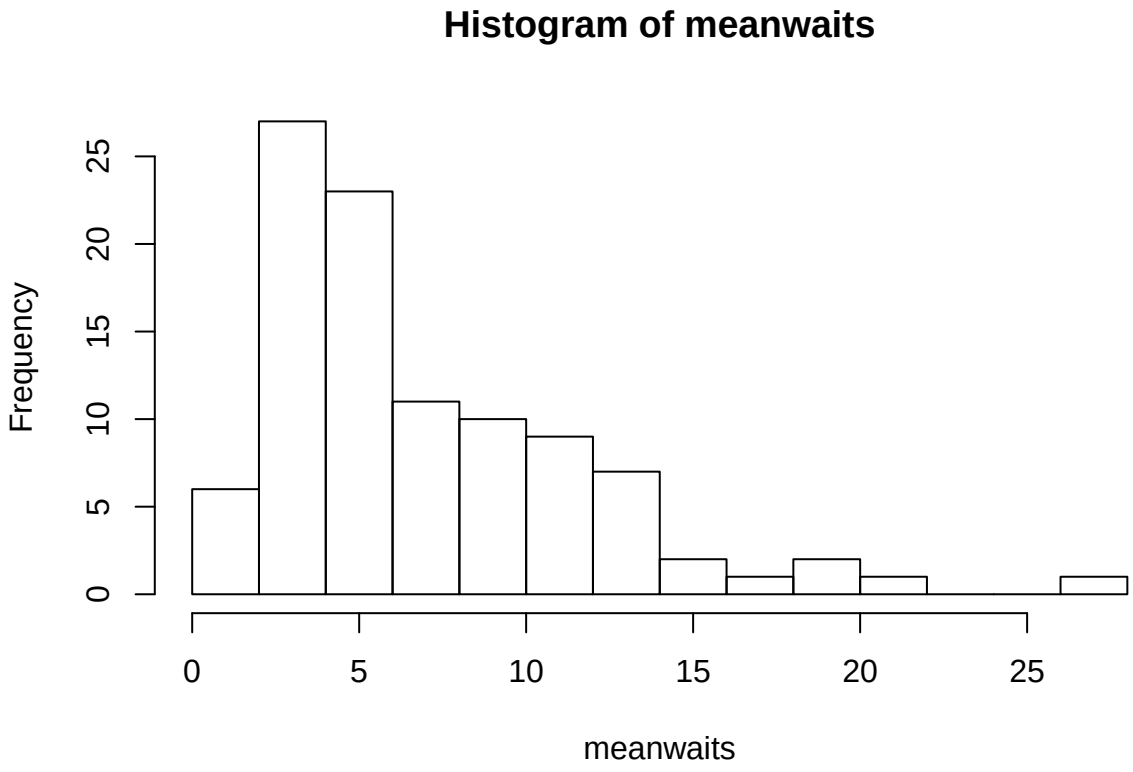
```
# Simulate a 100 arrival times and limit to those
# that arrive before 120


doctorq = function(rate, numDoctors, patientWaitDist) {
  # for 420 minutes, we expect 420 * rate patients
  expectL = 420 * rate
  # generate more patient arrival times than necessary
  ts = rexp(expectL * 2, rate) %>% cumsum
  # cutoff
  ts = ts[ts<=420]

  # numP = length(ts)
  # record when doctor is next available
  docAvail = rep(0, numDoctors)
  # record waittimes here. Pre allocation is good for R
  waittimes = rep(0, length(ts))
  # 1 indexed iteration
  i = 1

  for (patientarrival in ts){
    # find doctor who is next free
    nextAvailDoc = which.min(docAvail)
    timeAvail = docAvail[nextAvailDoc]
    # get index and when he's available
    #cat(i, patientarrival, "DA: ", docAvail, '\n')

    if (timeAvail <= patientarrival){
      # doctor's available
      waittimes[i] = 0
      docStart = patientarrival
    } else {
      # patient needs to wait. Record that
      waittimes[i] = timeAvail - patientarrival
      docStart = timeAvail
    }
    # update when the doctor is next available
    docAvail[nextAvailDoc] = docStart + patientWaitDist()

    # increase index of waiting
    i = i + 1
  }
  return(waittimes)
}
```

```
patientrate = 0.2 # .1 per minute poisson
numDoctors = 3
# I'm passing an arbitrary distribution across
patientWaitDist = function() runif(1, 5, 20)

waits = doctorq(patientrate, numDoctors, patientWaitDist)
nump = length(waits)
plot(1:nump, waits, xlab="Patient Number")
```

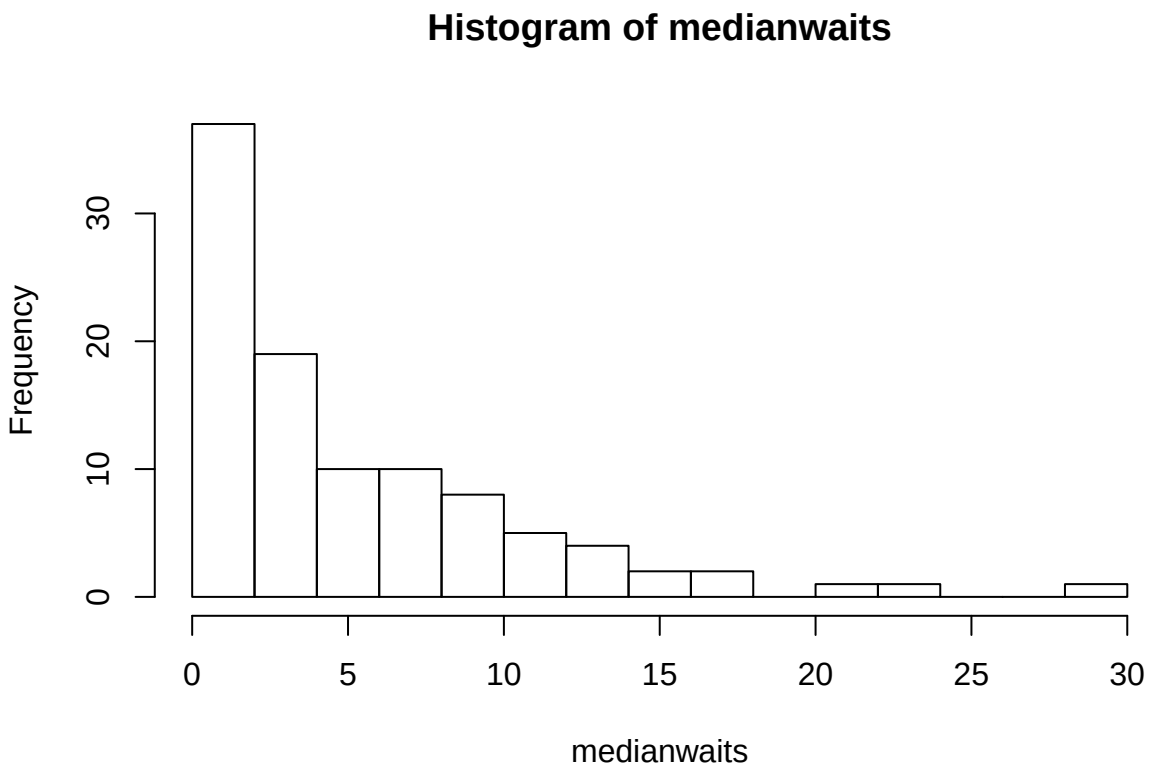


Now for 100 days

```
Ndays = 100
meanwaits = rep(0, Ndays)
medianwaits = rep(0, Ndays)


patientrate = 0.2 # .1 per minute poisson
numDoctors = 3
# I'm passing an arbitrary distribution across
patientWaitDist = function() runif(1, 5, 20)


for (day in 1:Ndays) {
  waits = doctorq(patientrate, numDoctors, patientWaitDist)
  meanwaits[day] = mean(waits)
  medianwaits[day] = median(waits)
}

hist(meanwaits, 15)
```




```
hist(medianwaits, 15)
```



b. Simulate this problem as multi queue, i.e. each patient chooses a doctor to see when they arrive at the clinic.

Left as an exercise for the reader, will require a change in the function provided

c. Simulate where the time to solve problems is heavy tailed (I suggest using a shifted log normal)

Use `patientWaitDist = function() rlnorm(1, 1.11) + 5` Depending on how heavy tailed you make it, you'll see changes.

Monte Carlo, convergence and accuracy

a. Calculate pi using Buffon's Needle using Monte Carlo. Use 4 different techniques to model how a human would drop the needles and report on any bias estimates.

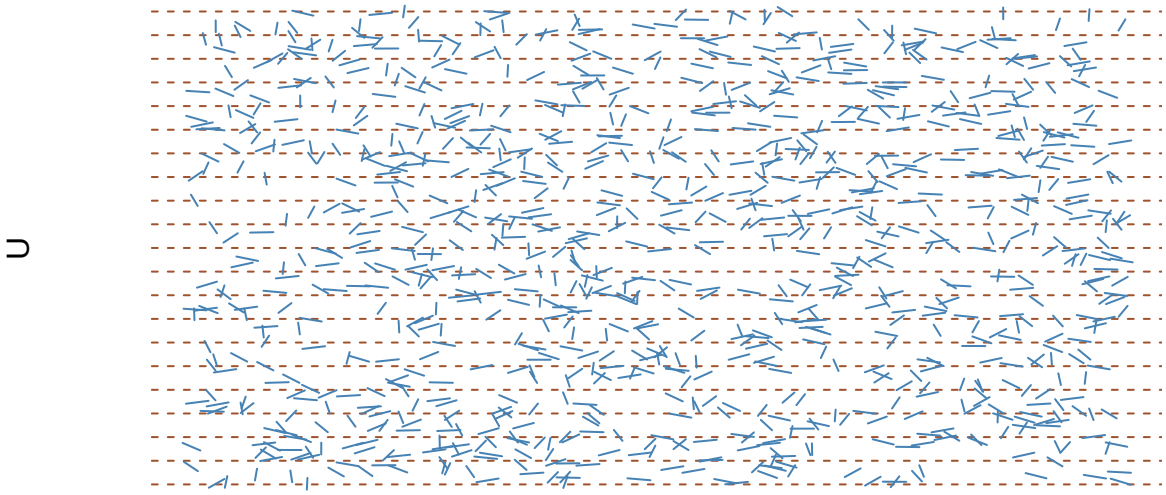
Note this code is quite old, so feel free to change things

Setup

```
numbhits=function(A,B){
 sum(abs(floor(A[,2])-floor(B[,2]))>0)}

L=0.25 #half-length of the needle
D=20  #length of the room
N=10^6
```

Uniform Centre and Uniform Angle

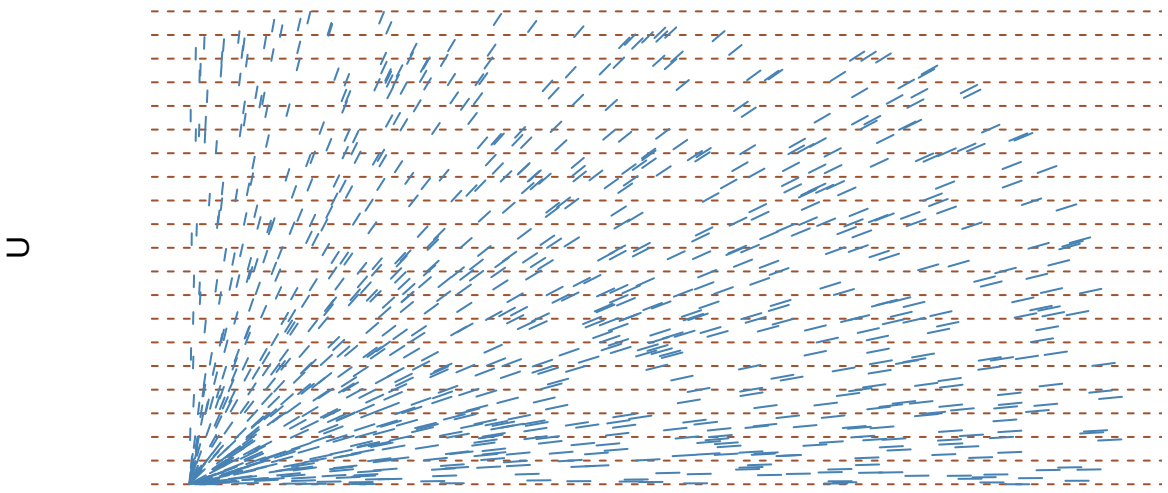
```
#version #1: uniform location of the centre
U=runif(N,min=0,max=D) #centre
O=runif(N,min=0,max=pi) #angle
C=cbind(runif(N,0,D),U)
A=C+L*cbind(cos(O),sin(O))
B=C-L*cbind(cos(O),sin(O))



plot(C,type="n",axes=F,xlim=c(0,D),ylim=c(0,D))
#title(main=paste(numbhits(A,B),"hits",sep=" "))
for (i in 0:(D))
 abline(h=i,lty=2,col="sienna")
for (t in 1:1000)
 lines(c(A[t,1],B[t,1]),c(A[t,2],B[t,2]),col="steelblue")
```



Pi Estimate from above is 3.1438335

Rays from a corner

```
#version #1: uniform location of the centre
O=runif(N,min=0,max=pi/2) #angle
U=runif(N)*(D*sqrt(1+apply(cbind(sin(O)^2,cos(O)^2),1,min))-2*L) #centre
A=cbind(U*cos(O),U*sin(O))
B=A+2*L*cbind(cos(O),sin(O))


plot(C,type="n",axes=F,xlim=c(0,D),ylim=c(0,D))
#title(main=paste(numbhits(A,B),"hits",sep=" "))
for (i in 0:(D))
 abline(h=i,lty=2,col="sienna")
for (t in 1:1000)
 lines(c(A[t,1],B[t,1]),c(A[t,2],B[t,2]),col="steelblue")
```

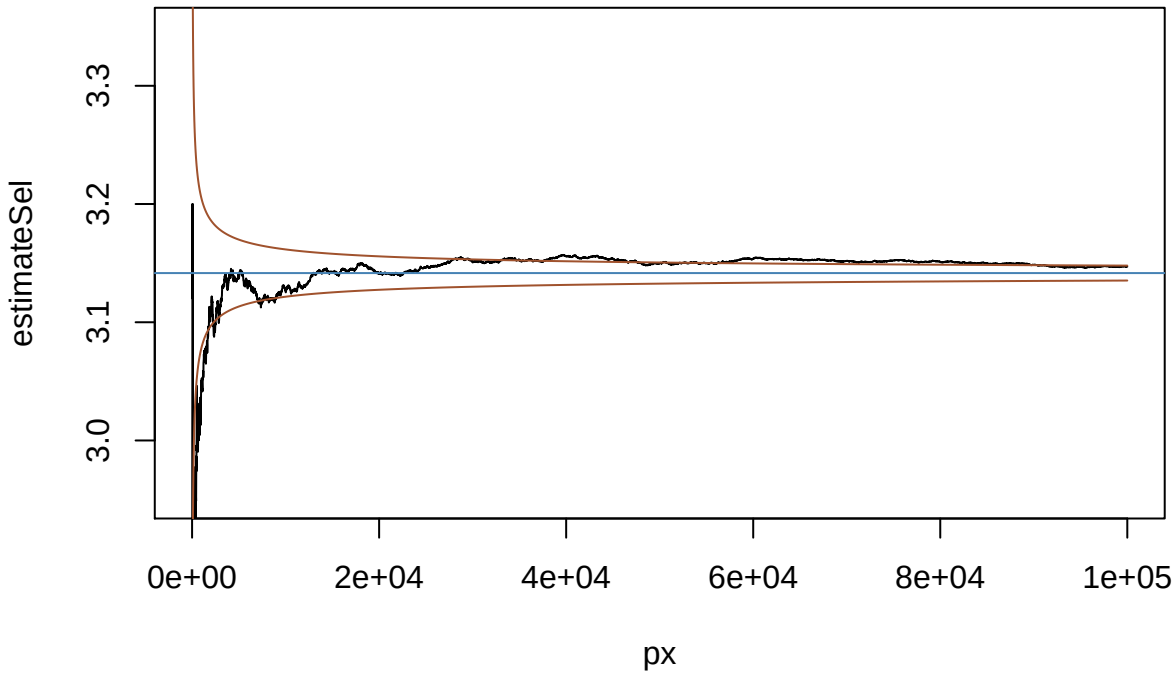


Pi Estimate from above is 3.2391602. We can see by changing how our data is generated, our results can change and our Monte Carlo results are no longer accurate

b. Calculate pi using the cannonball method. Report on the empirical error as a function of number of samples and explain your hypothesis.

Error in a Monte Carlo sense is 1/sqrt(N). This will be obvious from below.

```
# cannonball
N = 10^5
x = runif(N)
y = runif(N)

c = cumsum(x^2 + y^2  < 1)
estimate = 4*c/(1:N)


px = seq.int(10, N, 10)

estimateSel = estimate[px]
bound = sqrt(1/px)

plot(px, estimateSel, 'l', ylim=c(2.95, 3.35)); abline(pi, 0, col='steelblue'); lines(px, pi + 2*bound,
```



Now just the error plots in log of sample size

```
#px = seq(1, 6, length.out=1000)
estimateZer = (estimate[px] - pi)
plot(log10(px), estimateZer, 'l', ylim=c(-0.6, 0.6), ylab="Error", xlab="log10(samplesize)") ; abline(0
```