Lab 6

Stephen R. Proulx

Bayesian Statistical Modeling Winter 2024

Lab Exercise, Week 7

When is this lab due? Labs are due on the Thursday after they are assigned. However, in many cases you can complete them during the lab period itself. This assignment is due on Thursday, 2/29/2024.

The Metropolis algorithm

Here is the code from the book to run a simple implementation of the algorithm. It requires a function that we are trying to measure, which is the way that population size varies by island. In general, it is often possible to evaluate our function of interest (the posterior probability density function) for specific input values, but not easy to find the regions where the function has high values, especially when the function is multidimensional. Here we will be working with functions that we define and know everything about already, just to demonstrate that the algorithm can find their shape.

A simple linear function

We will start with the example from the book, one where the size of the islands we are visiting is equal to the index number of that island. We will setup a tibble that holds the island index number `location` and the island `size`

```
popsize=tibble(location=seq(1:10),size=seq(1:10))
num_islands=nrow(popsize)
```

Let's give it a look:

```
ggplot(popsize, aes(x=location,y=size)) + geom_line()
```

Great, we know what it looks like :) But we also want to re-scale it so that it can represent frequencies, which means sum to one. The trick to this is to find the normalizing constant which is the sum of the `size` column.

```
norm=sum(popsize$size)

popsize <- mutate(popsize,frequency = size/norm)

sum(popsize$frequency)
```

So we can see what it would look like expressed as frequency, this is the frequency we want our algorithm to give us back when it explores the landscape.

```
ggplot(popsize, aes(x=location,y=frequency)) + geom_line()
```

Walking through the algorithm by single steps

Start off at island 10 and set the value of current to the current position:

```
positions <- tibble(week=1,position=10)

current=positions$position
cur_week=2
```

First decide to test moving to the left (move of -1) or right (move of +1)

```
set.seed(378484)
(flip<-sample( c(-1,1) , size=1 ))
```

With this random seed, we move right, i.e. from 10 to 11 Now get the position of the proposal move

```
 (proposal <- current + flip)
```

But 11 might not be a real position, because we have wrapped around the top of our circle of islands, so we check this and figure out that we are actually testing island 1:

```
 if ( proposal < 1 ) proposal <- num_islands
    if ( proposal > num_islands ) proposal <- 1

proposal
```

Now we calculate the probability of moving, which is one if the proposal island has a larger size, and is the ratio of the population sizes on the islands we are comparing otherwise. First we calculate the ratio of populations sizes:

```
 (prob_move <- popsize[popsize$location==proposal,]$size/popsize[popsize$location==current,]$size)
```

Island 1 is 0.1 as big as island 10, so we have a 10% chance of moving to that island.

We move if a random draw (uniform(0,1)) is less than the value of `prob_move`

```
(current <- ifelse( runif(1) < prob_move , proposal , current ))
```

Our random draw was bigger than 0.1, so we stay on island 10.

And now we record our position and add it to the list:

```
(positions <- rbind(positions,c(cur_week,current)))

cur_week<-cur_week+1
```

Question 1 Run through this process again, you can cut and paste most of it, but make sure you both understand what each bit of code is doing, and the specific outcome you get. Write in text explaining what and why the algorithm is doing.

Question 2 See how many samples it takes to get the a distribution that is similar to our expectation. Here is code to do it for a 10 week sampling period.

This generates the sequence of moves:

```
num_weeks<-5000
positions<- tibble(week=seq(num_weeks),location=rep(0,num_weeks)) # make an tibble to record our progre
current <- 10 # starting location

for ( i in 1:num_weeks ) {
  ## record current position
    positions$location[i] <- current
  ## flip coin to generate proposal
    proposal <- current + sample( c(-1,1) , size=1 )
  ## now make sure he loops around the archipelago
    if ( proposal < 1 ) proposal <- num_islands
    if ( proposal > num_islands ) proposal <- 1
  ## move?
    prob_move <- popsize[popsize$location==proposal,]$size/popsize[popsize$location==current,]$size
    current <- ifelse( runif(1) < prob_move , proposal , current )
}
```

This plots the sequence path:

```
ggplot(data= positions , aes(x=week,y=location))+
  geom_line()
```

And this compares the histogram of the sampled islands to the frequency in the original function:

```
ggplot(data=positions, aes(x=location))+
  geom_histogram(bins=10)+
  geom_line(data=tibble(location=seq(1:10),count=seq(1:10))%>%
              mutate(count=count*num_weeks/norm),inherit.aes = F, aes(x=location,y=count))
```

Re-create these plots for 100, 500, 1000, and 5000 weeks. How well does it match? How could you quantify the match?

Question 3: A more difficult archipelago to study Here we define an island chain that is going to be more difficult for the algorithm to map, because it has two separate islands that are way bigger than their neighbors. I'm giving you the code to run multiple markov chains and then combine them.

Your job for this question is to interpret the results verbally.

First we set up the islands with two very large islands:

```
popsize=tibble(location=seq(1,14),size=c(0.1,1,10,100,10,1,0.1,0.1,0.1,1,10,100,10,0.1))

norm=sum(popsize$size)
```

And we will run 5 different Markov chains, starting right in between the two big islands.

```
num_weeks <- 1e4
reps=5
chains=list()

for(j in 1:reps){
positions<- tibble(week=seq(num_weeks),location=rep(0,num_weeks))
current <- 8
for ( i in 1:num_weeks ) {
  ## record current position
    positions$location[i] <- current
  ## flip coin to generate proposal
    proposal <- current + sample( c(-1,1) , size=1 )
  ## now make sure he loops around the archipelago
    if ( proposal < 1 ) proposal <- 14
    if ( proposal > 14 ) proposal <- 1
  ## move?
    prob_move <- popsize[popsize$location==proposal,]$size/popsize[popsize$location==current,]$size
    current <- ifelse( runif(1) < prob_move , proposal , current )
}

chains[[j]]  <- positions
}
```

We can plot the sequence of islands visited for each chain:

```
ggplot(data=chains[[1]] , aes(x=week,y=location))+
  geom_line(color="red",alpha=0.5)+
  geom_line(data=chains[[2]],color="blue",alpha=0.5)+
  geom_line(data=chains[[3]],color="green",alpha=0.5)+
  geom_line(data=chains[[4]],color="yellow",alpha=0.5)+
  geom_line(data=chains[[5]],color="purple",alpha=0.5)
```

We can also look at the histograms:

```
ggplot(data=chains[[1]] , aes(x=location))+
  geom_histogram(bins = 14,fill="red",alpha=0.5)+
  geom_histogram(data=chains[[2]],bins = 14,fill="blue",alpha=0.5)+
  geom_histogram(data=chains[[3]],bins = 14,fill="green",alpha=0.5)+
  geom_histogram(data=chains[[4]],bins = 14,fill="yellow",alpha=0.5)+
  geom_histogram(data=chains[[5]],bins = 14,fill="purple",alpha=0.5)
```

From the sequences or the histograms, what do you see? How do they compare to each other?

And now we'll put the chains together and summarize them:

```
combined_chains=chains[[1]]
for(i in 2:5){
  combined_chains=bind_rows(combined_chains,chains[[i]])
}

ggplot(data=combined_chains , aes(x=location))+
```

```
  geom_histogram(bins=14 )+
  geom_line(data=popsize%>%
              mutate(count=size*num_weeks*reps/norm),inherit.aes = F, aes(x=location,y=count))
```

How do you think the algorithm did on this surface?

Question 4: Try your own surface Make up your own "population size surface". You can make it longer or shorter. Create a set of plots as we did for the other surfaces and guage how well the algorithm was able to match your function.