Final

2. Metropolis-Hastings Sampler

indicator= function(i,j){ ##this function computes the product of Xi's

if(i==j)

return(1)

else

return(-1)

}

gnuf = function(e,eta){

theta\_sum1 = 0

for(i in 1:2){ ##for US and Russia

for (j in 1:6){##for other countries

theta\_sum1 = theta\_sum1 +indicator(e[i],e[2+j])\*eta[((i-1)\*6)+j]

}

theta\_sum1 = theta\_sum1 + indicator(e[1],e[2])\*eta[13]##for us and russia edge

theta\_sum2 = 0

for(i in 1:2){##for issue edges to us and russia

for(j in 1:2){

theta\_sum2 = theta\_sum2 + indicator(e[i],e[j+8])\*eta[13+i+j]

}

}

theta\_sum = theta\_sum1 +theta\_sum2

alpha\_sum = 0

for(i in 1:10){

alpha\_sum = alpha\_sum + e[i]\*eta[17+i]

}

return(exp(-theta\_sum-alpha\_sum))##z will cancel in MH algorithm

}

}

get\_proposal = function(e){

digit = DiscreteSampler(rep(1/10,10),seq(1,10))

e[digit] = e[digit]\*-1 ##flips 1 to -1 and vice versa

if(e[9]==e[10]){

e[9]=e[9]\*-1 ##makes sure we only have one issue at a time

}

return(e)

}

eta =rep(1,27)

##us and ru

eta[13]=-20

eta[14]=-10#us and human

eta[15]=-10

eta[16]=10

eta[17]=-10

etest = c(1,1,-1,1,-1,1,-1,1,-1,1)

gnuf(etest,eta)

##MH sampler

mh\_sampler = function(e,t){

##initial state

alpha = 0

for (i in 1:t){

proposal = get\_proposal(e)

choice = (gnuf(proposal,eta)/gnuf(e,eta))##symmetric proposal so q's cancel

alpha = min(1,choice)

if(runif(1)<alpha)

e = proposal

else

e = e

}

return(e)

}

2a)

sample\_path <- function(e,N)

{

stat <- rep(0, N)

for (i in 1:N){

proposal = get\_proposal(e)

choice = (gnuf(proposal,eta)/gnuf(e,eta))##symmetric proposal so q's cancel

alpha = min(1,choice)

if(runif(1)<alpha)

e = proposal

else

e = e

stat[i] = e[1]\*e[2]

}

plot(1:N, stat, type="l")

#return (stat)

return(sum(stat)/N)

}

print(sample\_path(etest,1000))



This graph tracks the progress of X\_us\*X\_ru. As we can see, we “forget” the original state relatively quickly, meaning the chain moves fast. A burn-in time of 1000 iterations should do.

Expected\_value = function(e){

burn = mh\_sampler(e,1000)

return(sample\_path(burn,100000))

}

print(Expected\_value(etest))

[1] -0.56056

The expected value of X\_us \*X\_ru is about -.6.

2b)

Given that we assigned an eta which stipulates that the US and Russia both vote 1 on Economic

Development issue, we would expect this probability to be relatively high. The burn-in time does not need to be large, because the stationary distribution in this case very highly favors X\_us=X\_ru when the issue is Economic Development. However, for consistency we will use 1000 again.

get\_proposal\_modified = function(e){

digit = DiscreteSampler(rep(1/8,8),seq(1,8))##x\_eco is last index on e

e[digit] = e[digit]\*-1 ##flips 1 to -1 and vice versa

return(e)

}

mh\_sampler\_modified = function(e,t){

##initial state

alpha = 0

for (i in 1:t){

proposal = get\_proposal\_modified(e)

choice = (gnuf(proposal,eta)/gnuf(e,eta))##symmetric proposal so q's cancel

alpha = min(1,choice)

if(runif(1)<alpha)

e = proposal

else

e = e

}

return(e)

}

etest = c(rep(-1,9),1) ##x\_eco will always be 1 and x\_hu will always be -1

print(mh\_sampler\_modified(etest,1000))

sample\_path\_modified <- function(e,N)

{

stat <- rep(0, N)

for (i in 1:N){

proposal = get\_proposal\_modified(e)

choice = (gnuf(proposal,eta)/gnuf(e,eta))##symmetric proposal so q's cancel

alpha = min(1,choice)

if(runif(1)<alpha)

e = proposal

else

e = e

if(e[1]==e[2])

stat[i]=1

else

stat[i]=0

}

plot(1:N, stat, type="l")

#return (stat)

return(sum(stat)/N)

}

print(sample\_path\_modified(etest,1000))

prob\_us\_ru = function(e){

burn = mh\_sampler\_modified(etest,1000)

return(sample\_path\_modified(burn,10000))

}

print(prob\_us\_ru(etest))

[1] 0.9999



As expected, it unlikely that we will see anything other than 1 in this case. Also, as expected, the probability approaches 1 that the US and Russia will vote the same on this issue.

5.

This function outputs expected values based on eta in the order corresponding to eta.

expected\_value\_vector = function(eta){

#browser()

etest = c(1,-1,1,-1,1,-1,1,-1,1,-1)

burn = mh\_sampler(etest,1000,eta)

e = burn

theta\_list = vector(mode="list", length=27) ##this will store all of the values that need summming

for(i in 1:27){

theta\_list[[i]] = rep(0,10000)##creates a vector at each place in the list

}

for (x in 1:10000){##this loop will fill vectors with datto be averaged

#browser()

gnu\_e = gnuf(e,eta)

proposal = get\_proposal(e)

gnu\_prop = gnuf(proposal,eta)

choice = exp(log(gnu\_prop)-log(gnu\_e))##symmetric proposal so q's cancel

##exponential deals with floating point error

alpha = min(1,choice)

if(runif(1)<alpha)

e = proposal

else

e = e

##compute xi\*xj

for (i in 1:2) {##for US and Russia

for (j in 1:6) {##for other countries

#browser()

theta\_list[[(i-1)\*6+j]][x] = e[i]\*e[2+j]

}

#browser()

}

#browser()

theta\_list[[13]][x] = e[1]\*e[2]

for(i in 1:2){##for issue edges to us and russia

for(j in 1:2){

#browser()

theta\_list[[13+(i-1)\*2+j]][x] = e[i]\*e[j+8]

}

}

#browser()

##compute xi

for(i in 18:27){

theta\_list[[i]][x]=e[i-17]

}

#browser()

}

expected\_values = rep(0,27)

for (y in 1:27){

expected\_values[y] = sum(theta\_list[[y]])/10000

}

return(expected\_values)

}

Now we compute the gradient of log(eta). Before we can do that, though, we need a function that makes the data into combinations of -1’s and 1’s that can be interpreted by our expected value

calculator. I chose to randomly assign a -1 or a 1 with equal probability to missing data points.

yes\_no = function(a){

if(a=="yes")

return(1)

if(a=="no")

return(-1)

create\_data = function(){

vote\_id = unique(dd$rcid)##creates vector that has all vote id's

country\_id = unique(dd$country\_code)

#browser()

new\_data = matrix(rep(0,10190),nrow = 1019)

for (i in 1:length(vote\_id)){

rcid\_index = which(dd$rcid==vote\_id[i])

##stores lines of data in which data for this vote is

##if statements to identify issue

issue = as.integer(dd[rcid\_index[1],6])

if(issue==4){

new\_data[i,9]=1

new\_data[i,10]=-1

}

else{

new\_data[i,10]=1

new\_data[i,9]=-1

}

for(j in 1:length(rcid\_index)){##for country's votes

#going to do seperate if statements cause only 8 countries

if(dd[rcid\_index[j],3]=="US"){

new\_data[i,1] = yes\_no(dd[rcid\_index[j],4])

}

if(dd[rcid\_index[j],3]=="RU"){

new\_data[i,2] = yes\_no(dd[rcid\_index[j],4])

}

if(dd[rcid\_index[j],3]=="CA"){

new\_data[i,3] = yes\_no(dd[rcid\_index[j],4])

}

if(dd[rcid\_index[j],3]=="CH"){

new\_data[i,4] = yes\_no(dd[rcid\_index[j],4])

}

if(dd[rcid\_index[j],3]=="ME"){

new\_data[i,5] = yes\_no(dd[rcid\_index[j],4])

}

if(dd[rcid\_index[j],3]=="SD"){

new\_data[i,6] = yes\_no(dd[rcid\_index[j],4])

}

if(dd[rcid\_index[j],3]=="ss"){

new\_data[i,7] = yes\_no(dd[rcid\_index[j],4])

}

if(dd[rcid\_index[j],3]=="TZ"){

new\_data[i,8] = yes\_no(dd[rcid\_index[j],4])

}

}

}

##now we need to run through the matrix and fill in zeroes

##with -1 and 1 with equal probability

for(i in 1:1019){

for(j in 1:10){

if(new\_data[i,j]==0)

new\_data[i,j]=DiscreteSampler(c(.5,.5),c(-1,1))

}

}

return(new\_data)

}

#vote\_matrix = create\_data()

grad\_Log\_eta = function(eta){

expected = 10000\*expected\_value\_vector(eta)

theta\_partial = rep(0,17)

alpha\_partial = rep(0,10)

#for US edges

for(k in 3:8){

sum1 = 0

for(j in 1: nrow(vote\_matrix)){

sum1 = sum1 + (vote\_matrix[j,1]\*vote\_matrix[j,k])

}

theta\_partial[k-2] = -sum1 + expected[k-2]

}

##for RU edges

for(k in 3:8){

sum1 = 0

for(j in 1: nrow(vote\_matrix)){

sum1 = sum1 + (vote\_matrix[j,2]\*vote\_matrix[j,k])

}

theta\_partial[4+k] = -sum1 + expected[4+k]

}

# ##for US-RU edges

sum1 = 0

for(j in 1: nrow(vote\_matrix)){

sum1 = sum1 + (vote\_matrix[j,1]\*vote\_matrix[j,2])

}

theta\_partial[13] = -sum1 +expected[13]

##for issue edges

for(i in 1:2){

for(k in 9:10){

sum1 = 0

for(j in 1: nrow(vote\_matrix)){

sum1 = sum1 + (vote\_matrix[j,i]\*vote\_matrix[j,k])

}

theta\_partial[k+3+2\*i] = -sum1 +expected[k+3+2\*i]

}

#

}

for(i in 1:10){

sum1 = 0

for(j in 1: nrow(vote\_matrix)){

sum1 = sum1 + vote\_matrix[j,i]

}

alpha\_partial[i] = -sum1 + expected[i+17]

}

return(c(theta\_partial,alpha\_partial))

}

#print(grad\_Log\_eta(eta))

mag = function(a){

sum1 = 0

for(i in length(a)){

sum1 = sum1 + (a[i])^2

}

return(sqrt(sum1))

}

log\_eta\_estimate = function(eta){

sum1 = 0

for(i in 1:nrow(vote\_matrix)){

sum1 = sum1 + log(gnuf(vote\_matrix[i, ],eta))

}##note- not going to compute z-just want to test steepest ascent

return(sum1/1000)#to deal with its growth

}

steepest\_ascent = function(t){

s = .5

eta = rep(0,27)

#etavec = rep(0,t)

for(i in 1:t){

gradient = grad\_Log\_eta(eta)

gradient = gradient/mag(gradient)##normalizes

eta = eta + s\*gradient

if(i%%50==0){##tracks it every 50 iterations

est = log\_eta\_estimate(eta)

print(est)

}

#etavec[i] = log\_eta\_estimate(eta)

}

#plot(seq(1,t),etavec)

return(eta)

}



This graph shows that our steepest ascent algorithm is indeed working. Although it is not strictly increasing, it shows a general upward trend. It would have been possible to ensure that likelihood always increased with the use of backtracking, but the algorithm would have taken a very long time if we had stopped to compute log(eta) at each step. Now we can run the steepest ascent for a long number of iterations:

eta = steepest\_ascent(1100)

print(eta)

[1] -0.0808

[1] -1.791834

[1] -0.9429828

[1] 3.640724

[1] 7.621669

[1] 14.20757

[1] 16.5137

[1] 15.15591

[1] 16.23442

[1] 17.58198

[1] 22.46061

[1] 25.1068

[1] 25.91475

[1] 30.96852

[1] 30.07832

[1] 36.3049

[1] 39.55387

[1] 40.86027

[1] 39.92495

[1] 41.68771

[1] 49.21348

[1] 46.02296

[1] 46.24907

This vector tracked the value every 50 iterations of an estimate of the log likelihood function.

For the most part, these values increased, until about 1000 iterations, at which point we seemed

to have reached some sort of local maximum, which we will use for our eta vector.

[1] -6.676846e-02 -3.997280e-02 -1.779868e-02 -6.882413e-03 1.560022e-02 2.346860e-02

[7] 2.503506e+00 2.638180e+00 4.383113e+00 -2.775960e+01 -1.706677e-01 -2.916193e+01

[13] 4.965392e+00 -5.497529e+02 5.497529e+02 2.284750e+00 -2.284750e+00 1.092634e+01

[19] 8.884667e-04 -3.459478e-02 -1.899798e-02 2.162909e-02 -7.707820e-02 -3.717380e-03

[25] -3.729622e-02 -2.100000e+01 2.100000e+01

Here are several functions that compare actual values from the data set to expected values computed using eta:

> expected\_value\_vector(eta)

[1] 0.0100 0.0456 0.0294 -0.0024 -0.0192 -0.0724 -0.0300 -0.0008 -0.0078 0.0356 0.0064

[12] 0.0236 -0.4204 -1.0000 1.0000 0.4204 -0.4204 -1.0000 0.4204 -0.0100 -0.0456 -0.0294

[23] 0.0024 0.0192 0.0724 1.0000 -1.0000

The above function, which was utilized to calculate the gradient, computes the expected values of all pairs of Xi’s as well as the expected values of the individual Xi’s. For example, the 13th index of this vector corresponds to the expected value of X\_us\*X\_ru.

> sum(vote\_matrix[,1]\*vote\_matrix[,2])/nrow(vote\_matrix)

[1] -0.3954858

The expected value given eta is -.42, which is close to the data’s value of -.39.

> for(i in 3:8){

+ print(sum(vote\_matrix[,2]\*vote\_matrix[,i])/nrow(vote\_matrix))

+ }

[1] -0.02060844

[1] -0.04612365

[1] -0.0578999

[1] 0.6074583

[1] 0.0009813543

[1] 0.6192345

|  |  |  |
| --- | --- | --- |
|  | From Data set | From Expected value given eta |
| RU-CA | [1] -0.02060844 | -0.0300 |
| RU-CH | [1] -0.04612365 | -0.0008 |
| RU-ME | [1] -0.0578999 | -0.0078 |
| RU-SD | [1] 0.6074583 | 0.0356 |
| RU-SS | [1] 0.000981354 | 0.0064 |
| RU-TZ | [1] 0.6192345 | 0.0236 |

This table shows the values of X\_RU paired with other countries side by side with the expected values given eta. As we can see, some of the values are close and others are way off. This is possibly due to the fact that some countries, such as South Sudan, did not participate in many votes (only 1.1%), so there was no very much data to fit for that expected value. Eta is a decent fit in some ways, but lacks accuracy in many cases, as shown by this table. It produces expected values that are off by orders of magnitude.

7)

We just computed the expected value of X\_us\*X\_ru in problem 6:

> expected\_value\_vector(eta)

[1] 0.0100 0.0456 0.0294 -0.0024 -0.0192 -0.0724 -0.0300 -0.0008 -0.0078 0.0356 0.0064

[12] 0.0236 -0.4204 -1.0000 1.0000 0.4204 -0.4204 -1.0000 0.4204 -0.0100 -0.0456 -0.0294

[23] 0.0024 0.0192 0.0724 1.0000 -1.0000

The 13th index, -.4204, corresponds to the expected value of interest.

This function computes the probability that the US and Russia vote the same on economic issues:

prob\_eco = function(){

eco\_iss = which(vote\_matrix[,10]==1)

prob\_sum =0

for(i in 1:length(eco\_iss)){

if(vote\_matrix[eco\_iss[i],1]==vote\_matrix[eco\_iss[i],2]){

prob\_sum=prob\_sum+1

}

}

return(prob\_sum/length(eco\_iss))

}

print(prob\_eco())

> print(prob\_eco())

[1] 0.3344262

8)

[1] -6.676846e-02 -3.997280e-02 -1.779868e-02 -6.882413e-03 1.560022e-02 2.346860e-02

[7] 2.503506e+00 2.638180e+00 4.383113e+00 -2.775960e+01 -1.706677e-01 -2.916193e+01

[13] 4.965392e+00 -5.497529e+02 5.497529e+02 2.284750e+00 -2.284750e+00 1.092634e+01

[19] 8.884667e-04 -3.459478e-02 -1.899798e-02 2.162909e-02 -7.707820e-02 -3.717380e-03

The eta vector is shown above. The values of most interest in this vector are the largest ones in absolute value, because they will have the greatest effect on the Boltzmann model probability. Two large values are at indexes 14 and 15. These values correspond to theta between the US and human rights and the theta between the US and economic development. The first is large and negative, meaning that the US tends to vote “yes” on human rights issues. The second is large and positive, meaning that the US tends to vote “no” on economic development issues. The US and Russia tend to vote opposite each other, shown by the positive index 13. Indexes 16 and 17 of eta show further that Russia votes opposite the US, as Russia votes “no” on human rights and “yes” on economic development.

Although these eta values are smaller in absolute value than indexes 14 and 15, the thetas that correspond to voting pairs between the US and the other 6 countries, and between RU and the other 6 countries, reveal an interesting trend. Canada, Switzerland, Montenegro, and Sudan tend to vote the same as the US, while Sudan, South Sudan and Tanzania tend to vote the same as Russia. These values may indicate patterns, but it must be noted that many data points were randomly assigned as -1 or 1, so conclusions can only be tentative.

Some possible implications for voting patterns are that some countries tend to vote in blocs together. It is well known that the US and Russia often disagree, but knowledge of which countries typically side with the US and which side with Russia could aid those powers in passing resolutions that need support of the General Assembly.