Homework 5

Max Wagner

April 2, 2016

1a.

```
X <- matrix(c(0.9,0,0,1,0.05,0.85,0,0,0.03,0.09,0.9,0,0.02,0.06,0.1,0), nrow = 4)
rownames(X) <- c("Low","Med","High","Fail")
colnames(X) <- c("Low","Med","High","Fail")
X
```

```
##      Low  Med High Fail
## Low  0.9 0.05 0.03 0.02
## Med  0.0 0.85 0.09 0.06
## High 0.0 0.00 0.90 0.10
## Fail 1.0 0.00 0.00 0.00
```

1b.

```
b <- matrix(c(1,0,0,0), nrow = 1)
b %*% X %*% X %*% X
```

```
##        Low      Med     High   Fail
## [1,] 0.771 0.115875 0.085425 0.0277
```

1c.

Rearrange the probabilities, then calc.

```
Xc <- X
Xc[4,4] <- 1
Xc[4,1] <- 0
b %*% Xc %*% Xc %*% Xc
```

```
##        Low      Med     High   Fail
## [1,] 0.729 0.114875 0.084825 0.0713
```

1d.

A function to iterate, instead of doing what i did above.

```
matp <- function(matrix, power){
  a <- matrix
  if (power >= 1) {
    for (i in 1:power) {
      matrix <- a %*% matrix
    }
    return (matrix)
  }
}
```

And now check some states by using the above function. Should we consider a failure to occur

```
n <- 1
d <- Xc
while (d[1,4] < .5) {
  d <- matp(Xc,n)
  n <- n + 1
}
print (paste("There is a 50% chance of failure at week:", (n+1)))
```

```
## [1] "There is a 50% chance of failure at week: 17"
```

1e.

With a one week turnaround:

```
round(52/17,0)
```

```
## [1] 3
```

1f.

Get a steady state, mult it by the profit/loss margins.

```
prof <- c(1000, 500, 400, -700)
steady <- matp(X, 100)
ep <- b %*% steady %*% prof;ep
```

```
##          [,1]
## [1,] 657.3771
```

1g.

Same idea, but with new policy:

```
X.2 <- matrix(c(0.9,0,1,1,0.05,0.85,0,0,0.03,0.09,0,0,0.02,0.06,0,0), nrow = 4)
prof <- c(1000, 500, -600, -700)
steady <- matp(X.2, 100)
ep <- b %*% steady %*% prof;ep
```

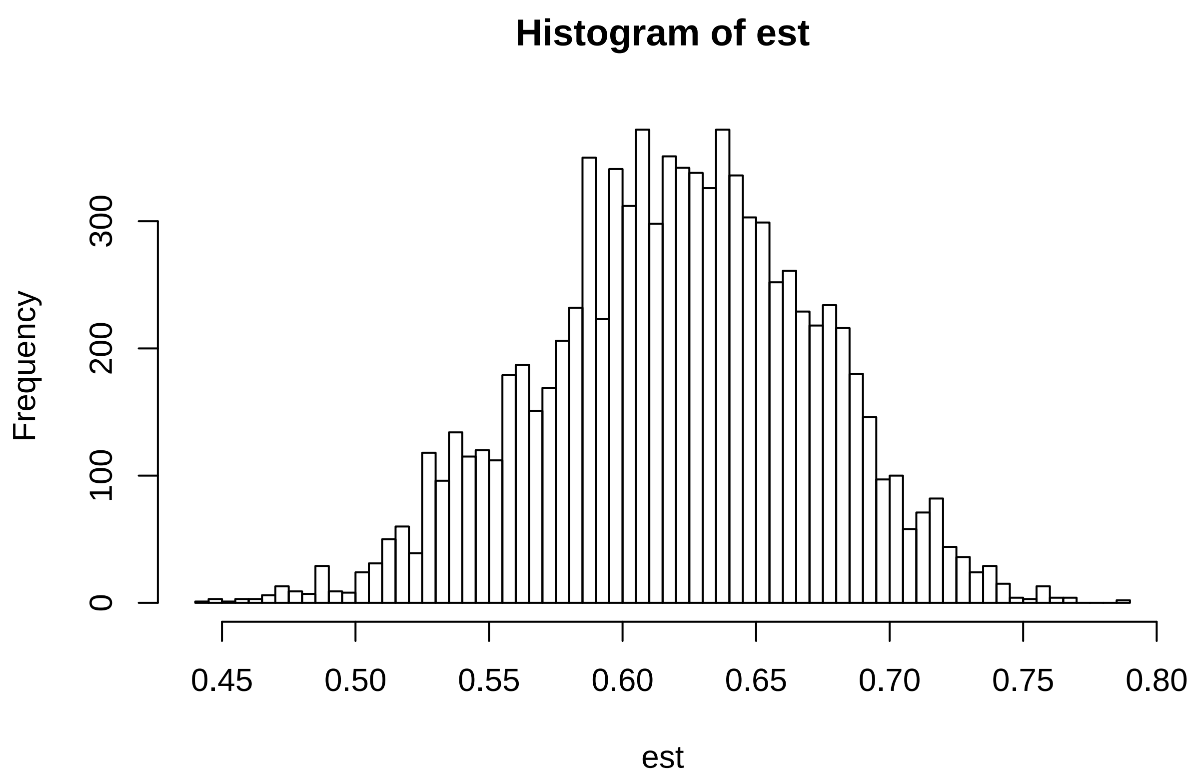
```
##          [,1]
## [1,] 769.3023
```

It makes more sense to repair when in the high state, as the expected profit is higher.

2.

Sorry for the lack of comments, it worked, and I didn't want to break it more than it already kind of was.

```r
haste <- function(theta, group) {
  return ((2 + theta)^group[1] * (1 - theta)^(group[2] + group[3]) * theta^group[4])
}

group <- c(125,18,20,34)
w <- .25
m <- 10000
x <- numeric(m)
b <- 1000
samp <- 1000
u <- runif(m)
v <- runif(m, -w, w)
x[1] <- w

for (i in 2:m) {
  z <- x[i-1] + v[i]
  if (u[i] <= haste(z, group) / haste(x[i-1], group)) {
    x[i] <- z
  } else {
    x[i] <- x[i-1]
  }
}

est <- x[(b+1):m]
hist(est, 100)
```

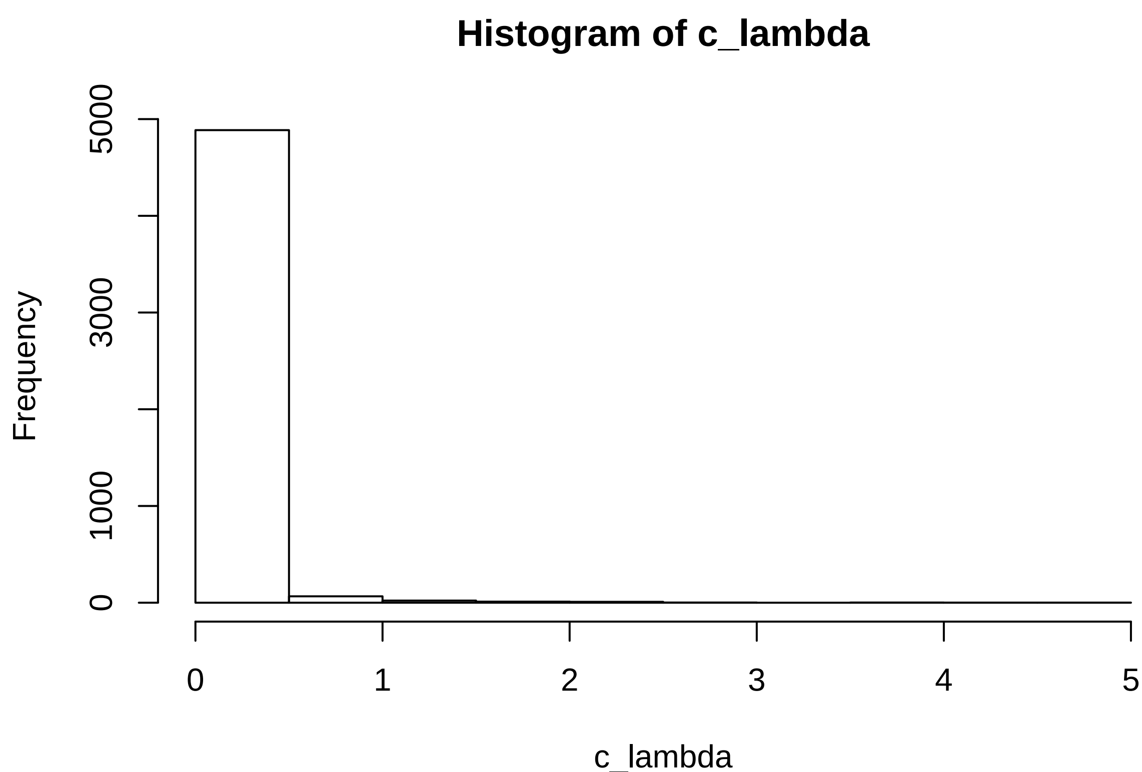


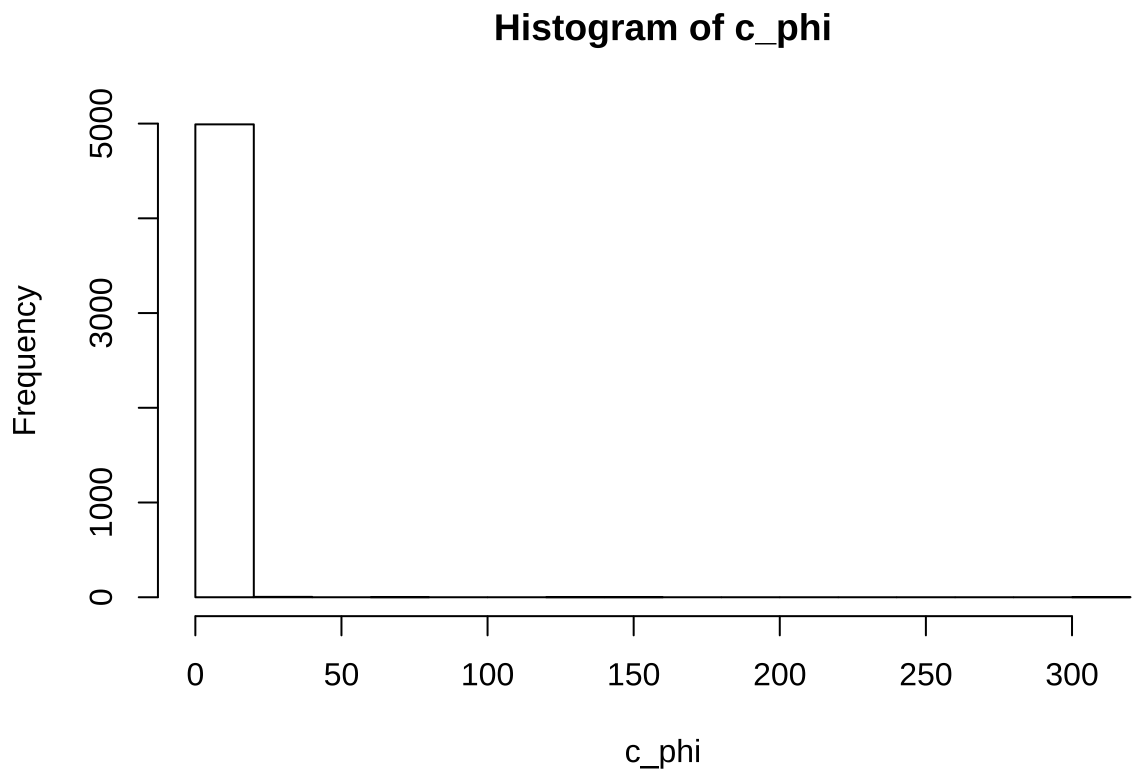
The estimation for theta is 0.6204498

3a.

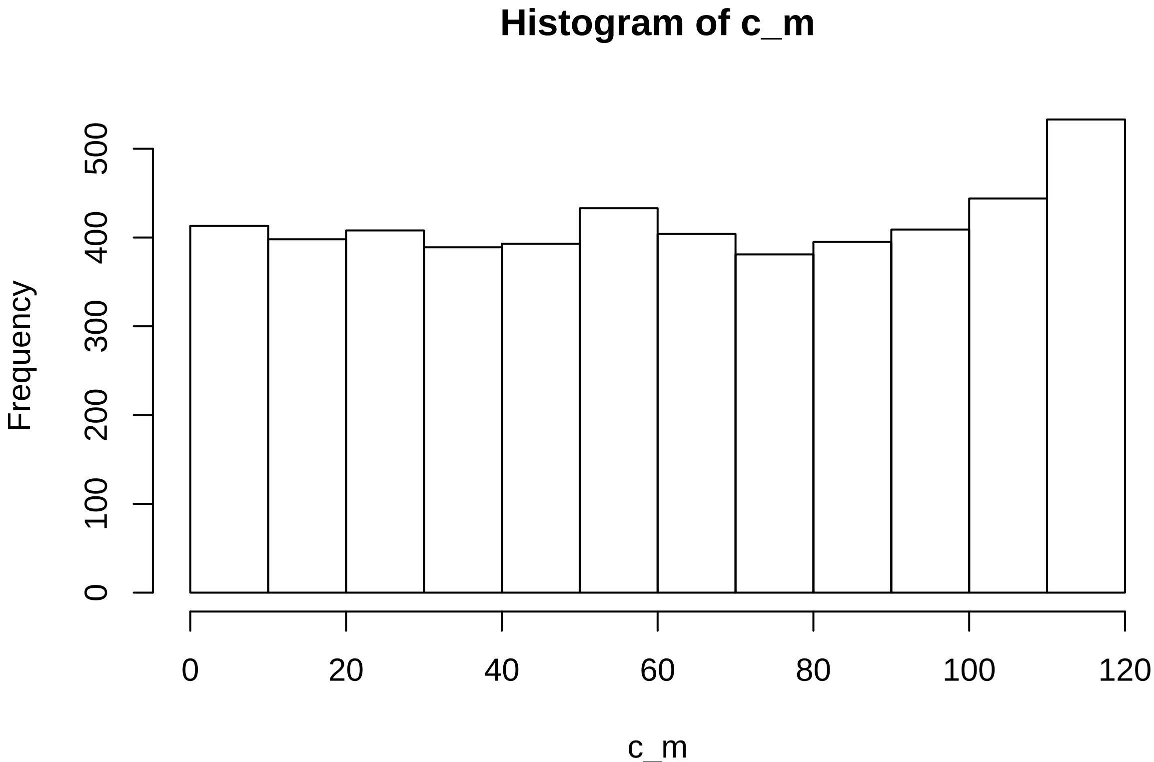
```
# add the vars i need, inital guesses
Y <- c(4,5,4,1,0,4,3,4,0,6,3,3,4,0,2,6,3,3,5,4,5,3,1,4,4,1,5,5,3,4,2,5,2,2,3,4,2,1,3,2,2,1,1,1,1,3,0,0,

mainthing <- function(i) {
  c_lambda <- c()
  c_phi <- c()
  c_beta <- c()
  c_delta <- c()
  c_m <- c()
  Mprob <- numeric(n)

  n <- i
  N <- length(Y)
  m <- sample(1:N, 1)

  alpha <- 1
  beta <- 1
  gamma <-1
  delta <- 1

  # fxn from psuedo code
  fullcondm <- function(m,lambda,phi,Y,N,alpha0,beta0,gamma0,delta0) {
    lamexp <- if (m >1)sum(Y[1:m]) else 0
    phiexp <- if (m<N) sum(Y[(m+1):N]) else 0
    prob <- lambda^(alpha0-1 + lamexp)*exp(-(beta0+m)*lambda)*phi^(gamma0-1+phiexp)*exp( -(delta0+n-m)*
    return(ifelse(is.nan(prob), 0, prob))
  }

  # some looping
  for (i in 1:n) {
    sum1 <- sum(Y[1:m])
    if (m == N) {
      sum2=0
    } else {
      sum2 <- sum( Y[ (m+1):N])
    }

    lambda <- sample(rgamma(sum1+alpha, beta/(beta*m + 1)),1)
    phi <- sample(rgamma(sum2 + gamma, delta/ (delta*(N-m) + 1)), 1)

    for (j in 1:N) {
      Mprob[j] <- fullcondm(j,lambda, phi, Y,N,alpha,beta,gamma,delta)
    }

    if(sum(Mprob) == 0) {
      m <- sample(1:N, 1)
    } else {
      m <- sample(1:N, prob = Mprob, size = 1)
    }

    beta <- 1/ sample(rgamma(alpha, lambda+1), 1)
```

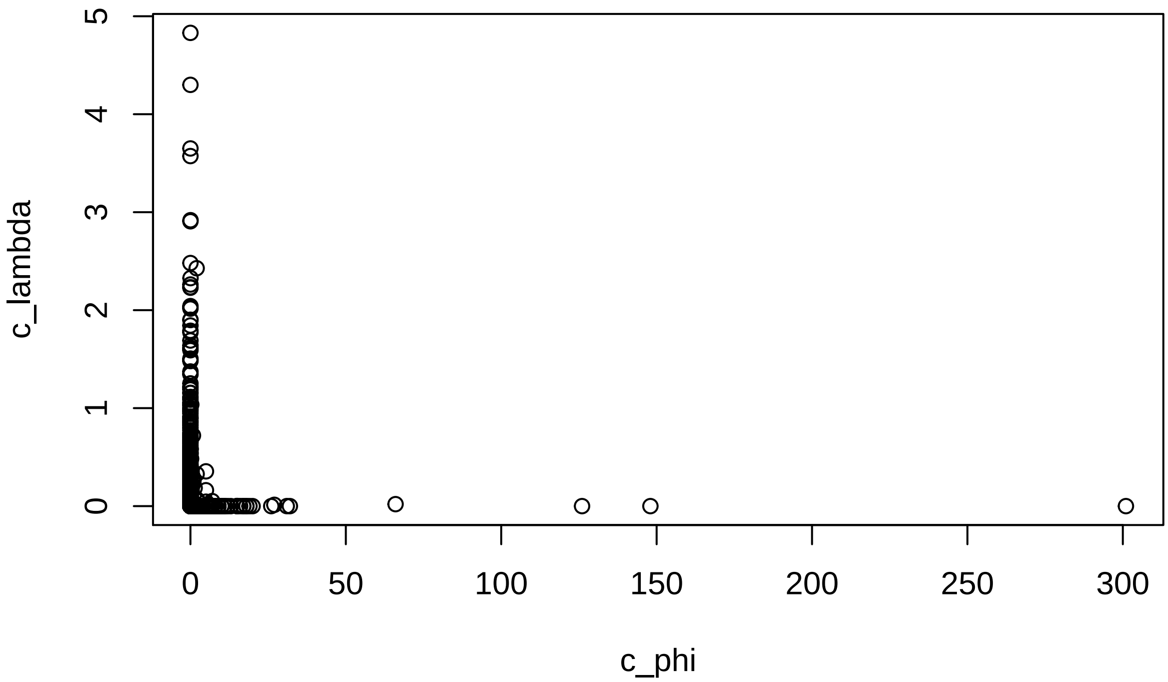
```
    delta <- 1/ sample(rgamma(gamma, phi+1), 1)

    c_lambda <- c(c_lambda, lambda)
    c_phi <- c(c_phi, phi)
    c_m <- c(c_m, m)
    c_beta <- c(c_beta, beta)
    c_delta <- c(c_delta, delta)
  }

  # plot the things for part a
  hist(c_lambda)
  hist(c_phi)
  hist(c_m)
  plot(c_lambda~c_phi)
  plot(c_lambda~c_m)
  plot(c_beta~c_delta)
}

# run 5000 iterations
mainthing(5000)
```



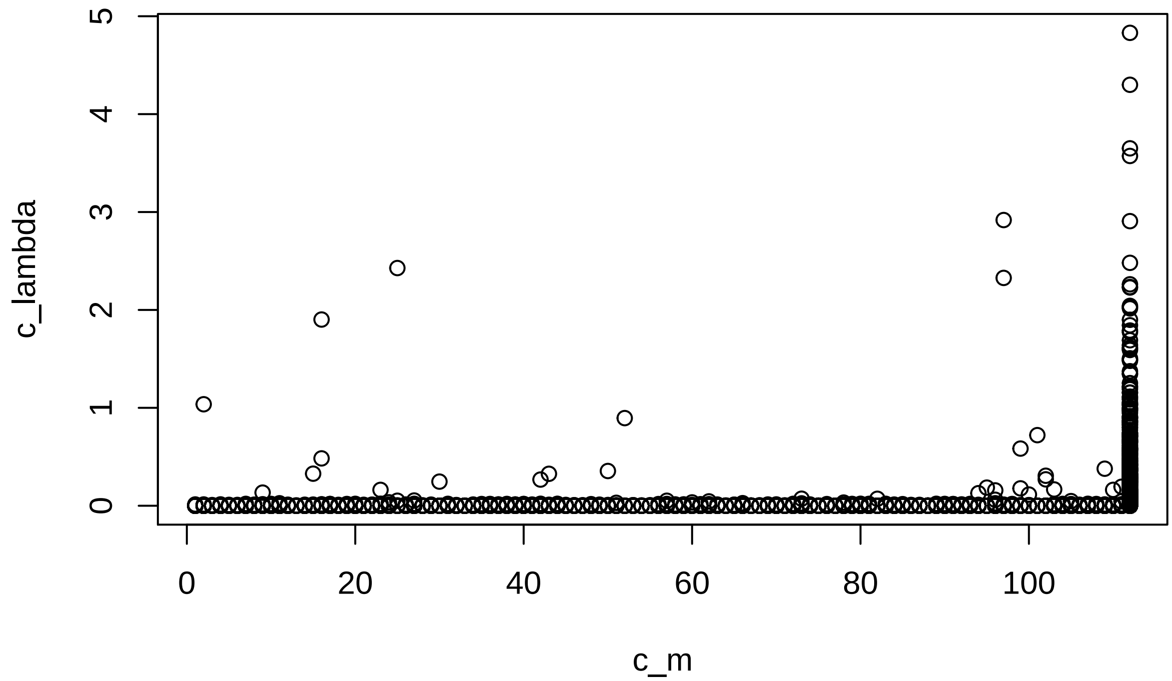


Histogram of c_phi
Frequency
0
1000
3000
5000
0
50
100
150
200
250
300
c_phi

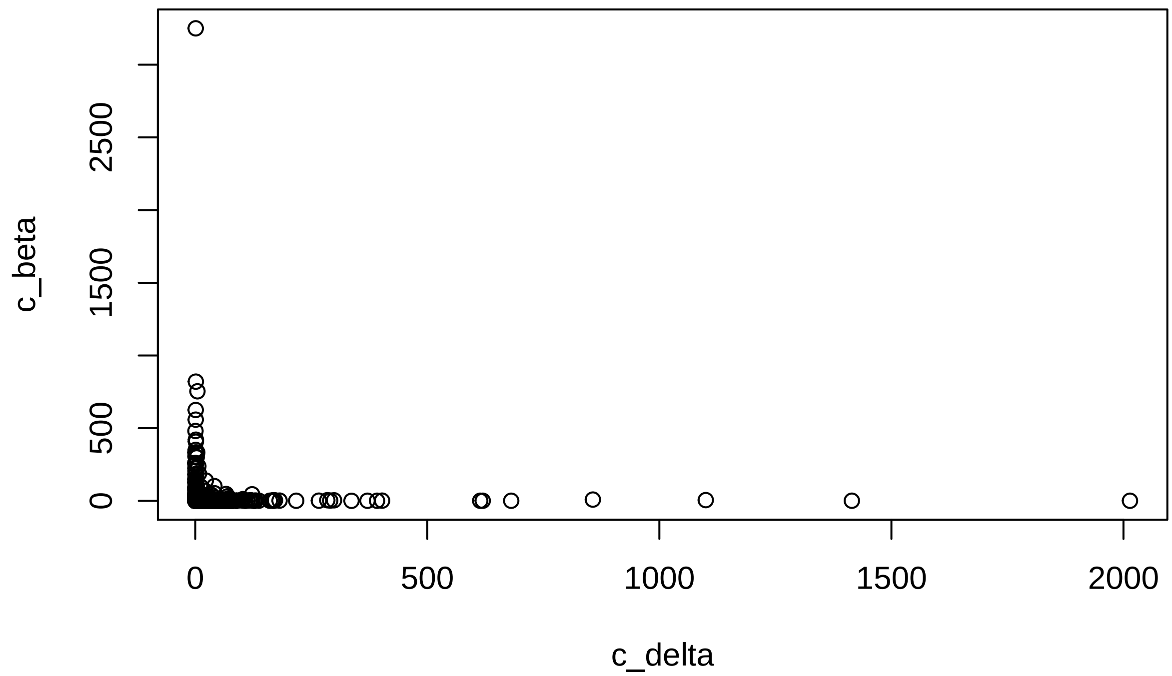

Histogram of c_m
Frequency
0
100
200
300
400
500
0
20
40
60
80
100
120
c_m



c_lambda
0
1
2
3
4
5
0
50
100
150
200
250
300
c_phi



c_lambda
0
1
2
3
4
5
0
20
40
60
80
100
c_m



3b.

The change point is around 40, or year 1891.

```
error <- qnorm(0.95)*sd(Y)/sqrt(length(Y))
print(paste('95% CI:', 40-error, 40+error))
```

```
## [1] "95% CI: 39.7439412679073 40.2560587320927"
```

```
before40 <- mean(Y[1:40]);before40
```

```
## [1] 3.125
```

```
after40 <- mean(Y[41:length(Y)]);after40
```

```
## [1] 0.9166667
```

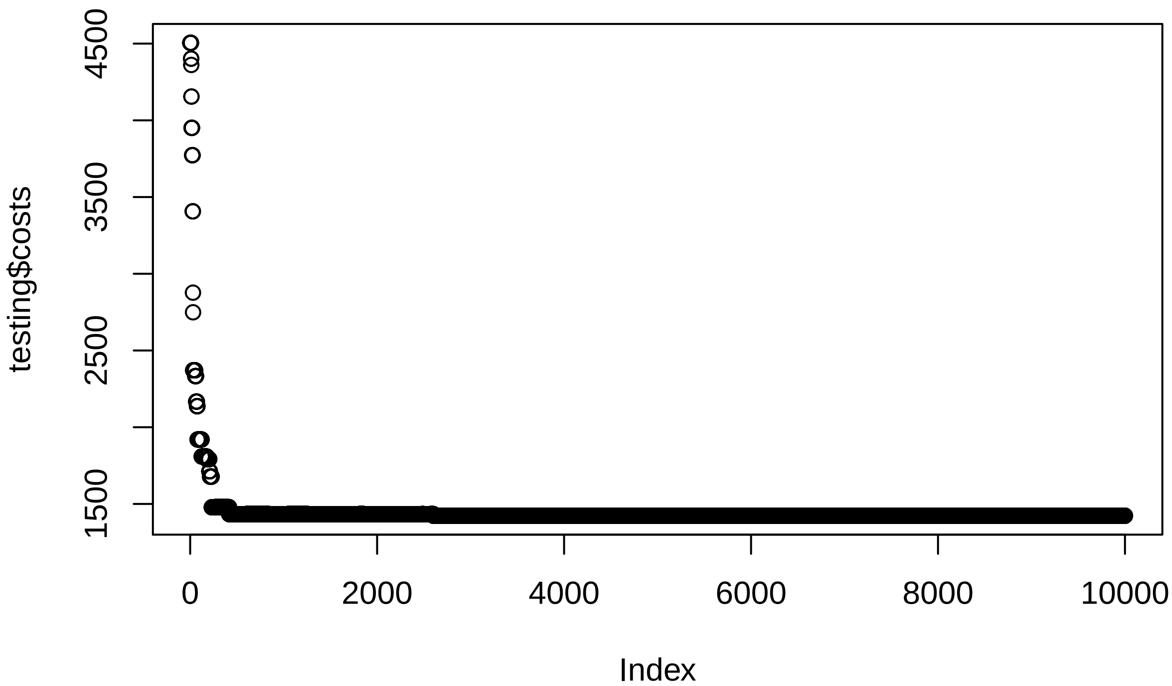
It seems that the average before 1891 is much lower than after, so it is consistant.

3c.

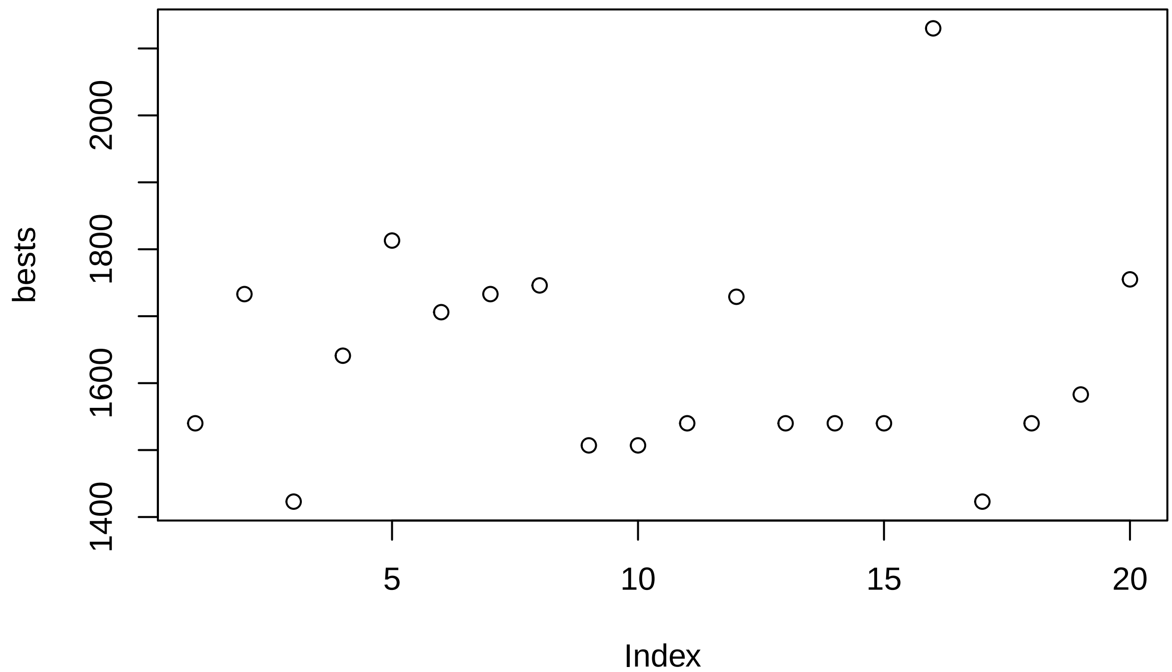
The main advantage of Metropolis is that is can be used when the posterior is unknown. Metropolis can also be more accurate due to the variables in Gibbs not jointly evolving.

4.

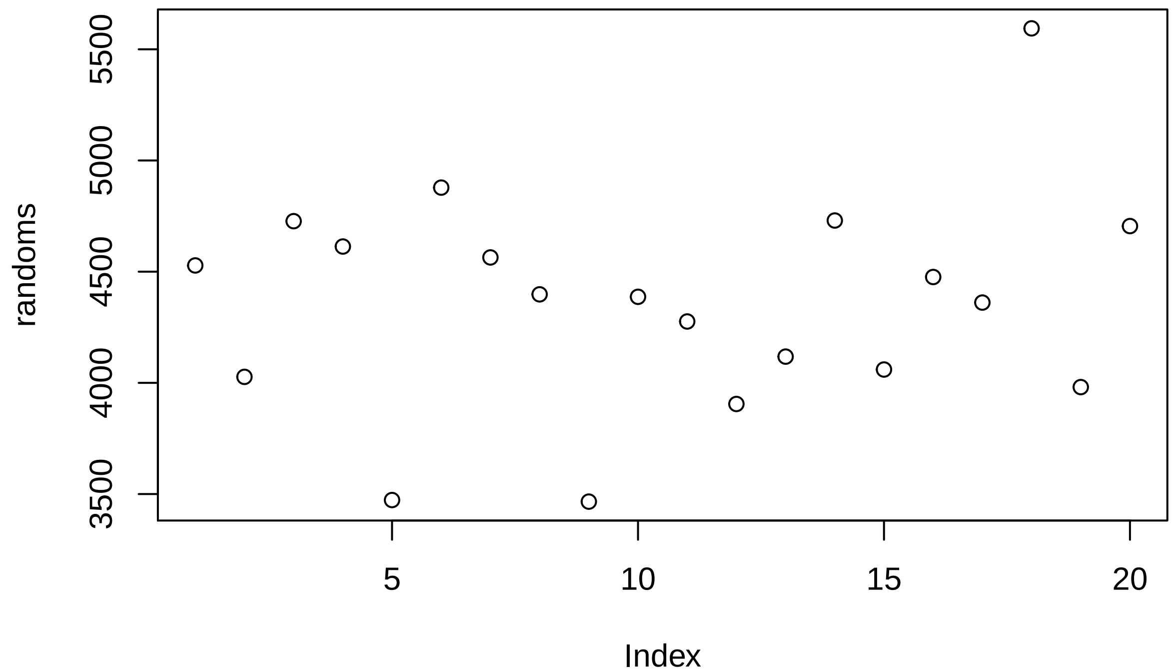
```
C <- matrix(c(0,633,257,91,412,150,80,134,259,505,353,324,70,211,268,246,121,633,0,390,661,227,488,572,5

cost.f <- function(x, C) {
  cost <- 0
  for (i in 1:(length(x) - 1)) {
    cost <- cost + C[x[i], x[i + 1]]
  }
  return(cost)
}

anneal <- function(T0, beta, N) {
  T <- T0
  C <- matrix(c(0,633,257,91,412,150,80,134,259,505,353,324,70,211,268,246,121,633,0,390,661,227,488,572
  n <- dim(C)[1]
  x <- sample(1:17)
  sx <- cost.f(x, C)
  xbest <- x
  sbest <- sx
  costs <- numeric(N)

  for(i in 1:N){
    x1 <- sample(1:17, 17)
    I <- sort(c(x1[1], x1[2]))

    if(I[2] == 17)
      y <- c(x[1:I[1]-1], x[I[2]:I[1]])
    else
      y <- c(x[1:I[1]-1], x[I[2]:I[1]], x[(I[2]+1):length(x)])

    sx <- cost.f(x, C)
    sy <- cost.f(y, C)

    if(sy < sx)
      alpha <- 1
    else
      alpha <- exp(-(sy-sx) / T)

    U <- runif(1)
    if(U < alpha){
      x <- y
      sx <- sy
    }

    T <- beta * T
    xbest <- x
    sbest <- sx
    costs[i] <- sbest
  }
  return(list(x = xbest, costs = costs))
}

testing <- anneal(1, .9999, 10000)
plot(testing$costs)
```




```
bests <- c()
for(i in 1:20){
  bests <- c(bests, anneal(1, .9999, 10000)$costs[10000])
}
plot(bests)
```




```
randoms <- c()
for(i in 1:20){
  randoms <- c(randoms, cost.f(sample(1:17,17), C))
}
plot(randoms)
```



The optimal pathing takes around 1600, while random takes around 4200, so the optimal path is significantly better. The min of bests is 1423 while the min of randoms is 3466. We can do it more times to get similar results.

```
for(i in 1:4) {
  bests <- c()
  for(i in 1:20){
    bests <- c(bests, anneal(1, .9999, 10000)$costs[10000])
  }
  print(min(bests))
}
```

```
## [1] 1423
## [1] 1423
## [1] 1423
## [1] 1507
```

```
for(i in 1:4) {
  randoms <- c()
  for(i in 1:20){
    randoms <- c(randoms, cost.f(sample(1:17,17), C))
  }
  print(min(randoms))
}
```

```
## [1] 3586
```

```
## [1] 3493
## [1] 3603
## [1] 3395
```