Problem Set 6

Ian McGroarty

08MAR2020

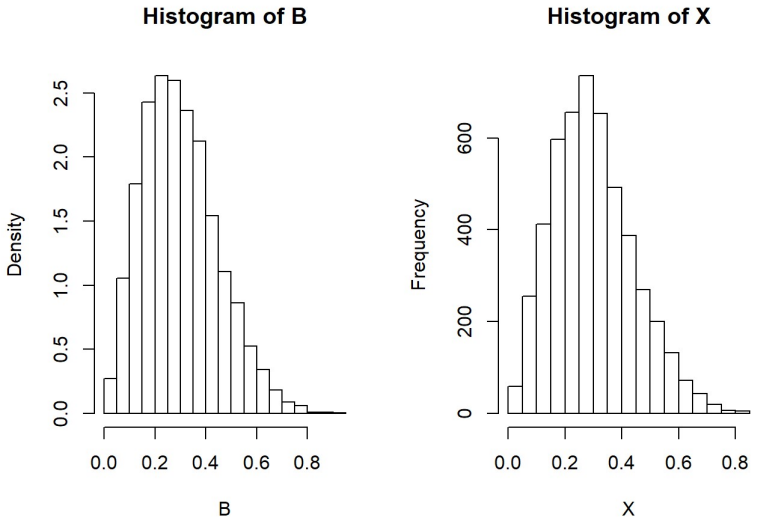
Problem 1

Consider a $Be(2.7, 6.3)$ target density:

(a)

Generate MH samples using a range of independent beta candidates from $Be(1, 1)$ to a beta distribution with a small variance. Compare the acceptance rates: The code below does exactly this. First I carefully build out and explain the procedure for Be(1,1). I then functionalize the procedure and compare the result of the function to the result of Be(1,1) to ensure that the function works properly.In doing this I also create a function to calculate the variance of the beta distribution so that I know that the new parameters lead to smaller distributions. I then vary the beta distribution parameters to compare acceptence rates. We can see that the further we get from the true distribution the lower the acceptance rate. What perplexes me is that if I use the true distribution, the acceptance rate is not 100%. I can't seem to deduce why this is and I must return to it when I have the chance.

```
# Initial Values
  a <- 2.7
  b <- 6.3
  Nsim <- 5000
  y1 <- 1:Nsim   # for plots
  AccRate <- 1
# Get the True Beta(2.7,6.3) distribution for comparisson
  B <- rbeta(Nsim,2.7,6.3)

# Initialize the chain (X CANDIDATES FROM A UNIFORM DISTRIBUTION)
   X <- rep(runif(1),Nsim)

# Run the Chain
   for (i in 2:Nsim){
     Y <- rbeta(1,1,1)  # Canidate values of X[i+1]
      # NOTE: Only need one value from this distribution since it is just the candidate
     rho <- dbeta(Y,a,b)/dbeta(X[i-1],a,b)
      # Note for the same distribution you don't need the q()
    # If runif(1)<rho then X[i]=X[i-1] if not it is Y
     X[i] <- X[i-1] + (Y-X[i-1])*(runif(1)<rho)
      # Still use the runif for the acceptance criteria
  # Acceptance Rate
     AccRate[i] <- (runif(1)<rho)
   }

## Plot the iterations
  # plot(y1,X,type="S", xlim=c(100,300))

## Plot Beta
  par(mfrow = c(1,2))
  hist(B, freq=FALSE)
  hist(X)
```



```
## Acceptance Rate
  (sum(as.numeric(AccRate)))/(Nsim-1)
```

```
## [1] 0.4490898
```

```
## Create A function for this process which allows us to vary the Beta Distribution
  # Variance Function
   variance.beta <- function(betaa,betab){
      var <- (betaa*betab)/(((betaa+betab)^2)*(betaa+betab+1))
     print(c("variance is",var))
   }
  # Metropolis Hastings function
      MHMC.AccRate.Beta <- function(simulations,betaa,betab){
      variance.beta(betaa,betab)
        F.Nsim <- simulations
      F.AccRate <- 1
      F.X <- rep(runif(1),F.Nsim)
      for (i in 2:F.Nsim){
         F.Y <- rbeta(1,betaa,betab)
        # (g(candidate)*q(X[i-1]))/(g(x[i-1])*q(candidate)
         F.rho <- (dbeta(F.Y,a,b)*dbeta(F.X[i-1],betaa,betab))/
                   (dbeta(F.X[i-1],a,b)/dbeta(F.Y,betaa,betab))
        F.X[i] <- F.X[i-1] + (F.Y-F.X[i-1])*(runif(1)<F.rho)
         F.AccRate[i] <- (runif(1)<F.rho)
      }
      print(c("Acc. Rate = ",(sum(as.numeric(F.AccRate)))/(F.Nsim-1)))
    }

MHMC.AccRate.Beta(Nsim,1,1)
```

```
## [1] "variance is"        "0.0833333333333333"
## [1] "Acc. Rate = "      "0.446089217843569"
```

```
MHMC.AccRate.Beta(Nsim,2,2)
```

```
## [1] "variance is" "0.05"
## [1] "Acc. Rate = "      "0.618123624724945"
```

```
MHMC.AccRate.Beta(Nsim,2.7,6.3)
```

```
## [1] "variance is" "0.021"
## [1] "Acc. Rate = "      "0.919583916783357"
```

```
MHMC.AccRate.Beta(Nsim,3,6)
```

```
## [1] "variance is"        "0.0222222222222222"
## [1] "Acc. Rate = "      "0.907381476295259"
```

```
MHMC.AccRate.Beta(Nsim,8,2)
```

```
## [1] "variance is"        "0.0145454545454545"
## [1] "Acc. Rate = "       "0.0002000400080016"
```

```
MHMC.AccRate.Beta(Nsim,10,10)
```

```
## [1] "variance is"        "0.0119047619047619"
## [1] "Acc. Rate = "      "0.676935387077416"
```

```
MHMC.AccRate.Beta(Nsim,10,50)
```

```
## [1] "variance is"         "0.00227686703096539"
## [1] "Acc. Rate = "      "0.992998599719944"
```

```
MHMC.AccRate.Beta(Nsim,50,50)
```

```
## [1] "variance is"         "0.00247524752475248"
## [1] "Acc. Rate = "       "0.0002000400080016"
```

(b)

Suppose that we want to generate a truncated beta restricted to the interval (c,d) with $(c, d) \in (0, 1)$. Compare the performance of a MHA based on B(2,6) proposal with one based on a $U(c, d)$ proposal. Take c = 0.1,0.25, d=0.9,0.75.

It seems that for both iterations of (c,d) the performace measured by run time is faster for the Uniform distribution. Generally, the performances on the range (c,d)=(0.1,0.9) are *slower* than for their respective processes over (c,d)=(0.25,0.75). This makes sense since it is a smaller range. However, the larger range (c,d)=(0.1,0.9) seems to be much more efficient for the beta distribution than the uniform for the same range and for both distribution on the smaller range. While for the smaller range (0.25,0.75) the uniform was more efficient than the beta, it was mostly due to a drop in efficiency of the beta distribution (relative to the larger range) rather than an increase in efficiency for the uniform distribution.

```
# Initial Values
  a <- 2.7
  b <- 6.3
  Nsim <- 5000
  y1 <- 1:Nsim   # for plots
  AccRate <- 1
# Get the True Beta(2.7,6.3) distribution for comparisson
  B <- rbeta(Nsim,2.7,6.3)

# Adapt the function from (a) to sample from a beta(2,6) with truncation from c to d
 MHMC.AccRate.Beta.Trunc <- function(simulations,betaa,betab,f.c,f.d){
      F.Nsim <- simulations
      F.AccRate <- 1
      F.X <- rep(runif(1),F.Nsim)
      for (i in 2:F.Nsim){
         F.Y <- rbeta(1,betaa,betab)
        # (g(candidate)*q(X[i-1]))/(g(x[i-1])*q(candidate)
         F.rho <- (dbeta(F.Y,a,b)*dbeta(F.X[i-1],betaa,betab))/
                  (dbeta(F.X[i-1],a,b)/dbeta(F.Y,betaa,betab))
        # Mutliply by the T/F statement that F.Y is in the bounds
        F.X[i] <- F.X[i-1] + (F.Y-F.X[i-1])*(runif(1)<F.rho)*(F.Y<=f.d)*(F.Y>=f.c)
         F.AccRate[i] <- (runif(1)<F.rho)*(F.Y<=f.d)*(F.Y>=f.c)
      }
      print(c("Acc. Rate = ",(sum(as.numeric(F.AccRate)))/(F.Nsim-1)))
 }
# Adapt the function from (a) to sample from a uniform from c,d
 MHMC.AccRate.Unif.Trunc <- function(simulations,betaa,betab,f.c,f.d){
      F.Nsim <- simulations
      F.AccRate <- 1
      F.X <- rep(runif(1),F.Nsim)
      for (i in 2:F.Nsim){
         F.Y <- runif(1,f.c,f.d)
        # (g(candidate)*q(X[i-1]))/(g(x[i-1])*q(candidate)
         F.rho <- (dbeta(F.Y,a,b)*dunif(F.X[i-1]))/
                  (dbeta(F.X[i-1],a,b)/dunif(F.Y))
        # Mutliply by the T/F statement that F.Y is in the bounds
        F.X[i] <- F.X[i-1] + (F.Y-F.X[i-1])*(runif(1)<F.rho)*(F.Y<=f.d)*(F.Y>=f.c)
         F.AccRate[i] <- (runif(1)<F.rho)*(F.Y<=f.d)*(F.Y>=f.c)
      }
      print(c("Acc. Rate = ",(sum(as.numeric(F.AccRate)))/(F.Nsim-1)))
 }

## (c,d)=(0.1,0.9)
    system.time(MHMC.AccRate.Beta.Trunc(10000,2,6,0.1,0.9))
```

```
## [1] "Acc. Rate = "      "0.796879687968797"
```

```
##    user  system elapsed
##    0.19    0.00    0.18
```

```
    system.time(MHMC.AccRate.Unif.Trunc(10000,2,6,0.1,0.9))
```

```
## [1] "Acc. Rate = "      "0.523252325232523"
```

```
##    user  system elapsed
##    0.17    0.00    0.17
```

```
## (c,d)=(0.25,0.75)
    system.time(MHMC.AccRate.Beta.Trunc(10000,2,6,0.25,0.75))
```

```
## [1] "Acc. Rate = "      "0.377837783778378"
```

```
##    user  system elapsed
##    0.15    0.00    0.16
```

```
    system.time(MHMC.AccRate.Unif.Trunc(10000,2,6,0.25,0.75))
```

```
## [1] "Acc. Rate = "      "0.573157315731573"
```

```
##    user  system elapsed
##    0.14    0.00    0.14
```

Problem 2

Calculate the mean and monitor the convergence of a gamma G(4.3,6.2) random variable using: ### (a) Accept-Reject with a G(4,7) candidate We can see with the R code below that accept reject works well but has an efficiency rate of only (about 0.32)

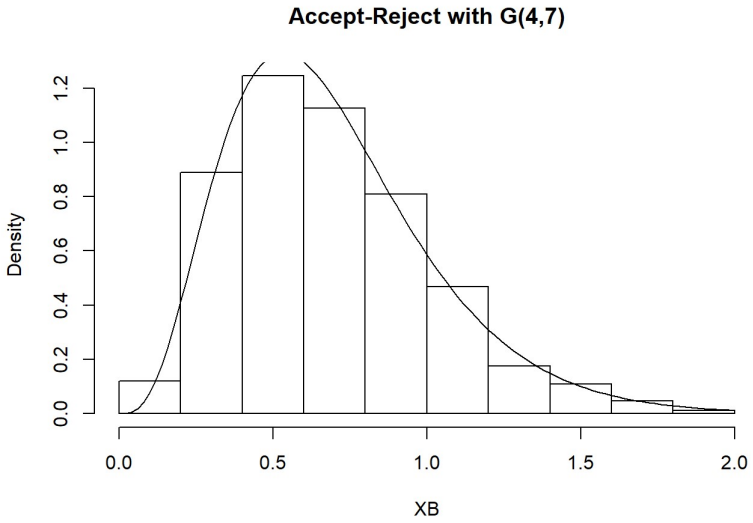
```
# Initial Values
Nsim <- 10000
n <- 4.3
b <- 6.2

# Set the max
  M <-  optimise(f=function(x){dgamma(x,4,7)},interval=c(0,1),maximum=T)$objective

# Set the function
   fx <- function(x){
     dgamma(x,n,b)
   }
# Generate x candidates over the range under condiseration
  Xcand <- runif(Nsim,min=0,max=2)
# Generate y candidates up to M
  Ycand <- runif(Nsim, min=0, max=M)
# Keep the x candidates for which the Y candidates are viable solutions
  XB <- Xcand[Ycand < fx(Xcand)]
# Acceptance Rate & Mean
   print(paste0("Acc. Rate = ",(length(XB)/Nsim)))
```

```
## [1] "Acc. Rate = 0.323"
```

```
    print(paste0("Mean = ",mean(XB)))
```

```
## [1] "Mean = 0.684904038817723"
```

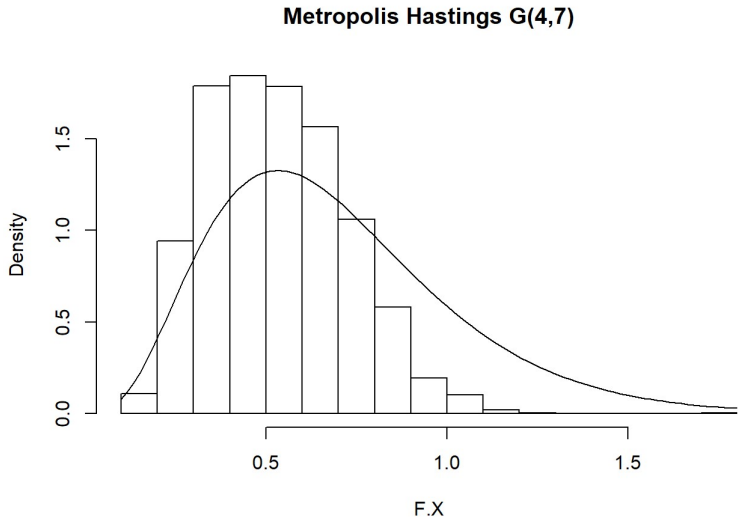
```
hist(XB,freq=FALSE, main="Accept-Reject with G(4,7)")
curve(dgamma(x,4.3,6.2),add=T)
```



(b) Metropolis Hastings wit G(4,7)

For the Metropolis Hastings algorism with proposal G(4,7) there is a much better acceptance rate of about 0.79. But the curve does not fit as well.

```
# Initial Values
  n <- 4.3
  b <- 6.2
  Nsim <- 5000
  AccRate <- 1

MHMC.AccRate.Gamma <- function(simulations,gammaa,gammab){
      F.Nsim <- simulations
      F.AccRate <- 1
      F.X <- rep(runif(1),F.Nsim)
      for (i in 2:F.Nsim){
         F.Y <- rgamma(1,gammaa,gammab)
        # (g(candidate)*q(X[i-1]))/(g(x[i-1])*q(candidate)
         F.rho <- (dgamma(F.Y,n,b)*dgamma(F.X[i-1],gammaa,gammab))/
                  (dgamma(F.X[i-1],n,b)/dgamma(F.Y,gammaa,gammab))
        # Mutliply by the T/F statement that F.Y is in the bounds
        F.X[i] <- F.X[i-1] + (F.Y-F.X[i-1])*(runif(1)<F.rho)
         F.AccRate[i] <- (runif(1)<F.rho)
      }
      hist(F.X, freq=FALSE, main=paste0("Metropolis Hastings G(",gammaa,",",gammab,")"))
      curve(dgamma(x,n,b),add=T)
      print(paste0("Acc. Rate = ",(sum(as.numeric(F.AccRate)))/(F.Nsim-1)))
      print(paste0("Mean = ",mean(F.X)))
}
MHMC.AccRate.Gamma(10000,4,7)
```

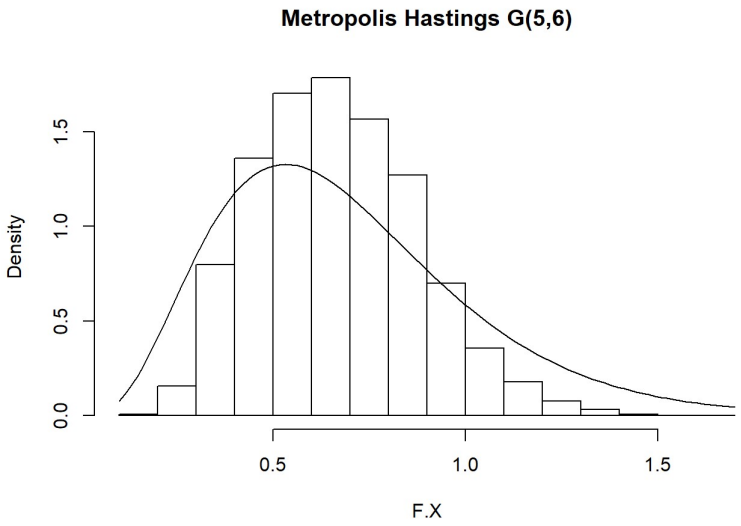



```
## [1] "Acc. Rate = 0.801880188018802"
## [1] "Mean = 0.532687230524437"
```

(c) Metropolis Hastings with G(5,6)

The Metropolis Hastings Algorithm with proposal G(5,6) is slightly better fit than that of G(4,7) but slightly less efficient with an acceptance probability of about 0.62

```
MHMC.AccRate.Gamma(10000,5,6)
```




```
## [1] "Acc. Rate = 0.631263126312631"
## [1] "Mean = 0.66760863190446"
```

Problem 3

Show the calculation of the Kernal of the Metropolis Hastings Algorithm:

$$\rho(X_t, X_{t+1}) \cdot g(X_{t+1}|X_t) + \delta(X_{t+1} - X_t) \cdot (1 - r(X_t))$$

So if is clear that:

$$X_{t+1} = \{ \begin{matrix} X' & w/prob & \rho(X_t, X') \\ X_t & w/prob & 1 - \rho(X_t, X') \end{matrix}$$

It follows that the conditional probability density:

$$f(X_{t+1}|X', X_t) = \delta(X' - X_{t+1}) \cdot \rho(X_t, X') + \delta(X_t - X_{t+1}) \cdot (1 - \rho(X_t, X'))$$

The Transition Kernal wants the conditional probability of $X_{t+1}|X_t$

$$K(X_t, X_{t+1}) = \int f(X_{t+1}|X') \cdot g(X'|X_t) dX'$$

Let us first concider the case that X' is accepted. Since all $X_{t+1} \in f(X)$ we can evalute the integral using the definition of K. By construction of K we have

$$K(X_t, X_{t+1}) = \rho(X_t, X_{t+1}) \cdot g(X_{t+1}|X_t) \quad \forall X_{t+1}$$

Next condier the case that X' is rejected and thus $X_{t+1} = X_t$. For this we can not evaluate the integral without first knowing $q(X_t, X_{t+1})$ because the distribution of rejections with depend on q. So we leave it as theintegration of the probability $X_{t+1} = X_t$.

$$K(X_t, X_{t+1}) = [1 - \int \rho(X_t, X_{t+1}) \cdot g(X_{t+1}|X_t) dX'] \delta(X_{t+1} - X_t)$$

We can add the two probabilities together since it is peicewise to get:

$$K(X_t, X_{t+1}) = \rho(X_t, X_{t+1}) \cdot g(X_{t+1}|X_t) + [1 - \int \rho(X_t, X_{t+1}) \cdot g(X_{t+1}|X_t) dX'] \delta(X_{t+1} - X_t)$$

Problem 4

Baker (1965) suggested that the acceptance probability of the Metropolis-Hastings algorithm be:

$$p = \frac{f_X(x_{i+1})}{f_X(x_{i+1}) + f_X(x_i)}$$

In the case where $g(x_{i+1}|x_i) = g(x_i|x_{i+1})$ does the acceptance probability gaurentee detailed balance?

$$
\begin{aligned}
g(x_i|x_{i+1}) &= g(x_{i+1}|x_i) && \text{By Construction} \\
f_X(x_i) \cdot g(x_i|x_{i+1}) &= f_X(x_i) \cdot g(x_{i+1}|x_i) \\
\frac{f_X(x_{i+1})}{f_X(x_{i+1}) + f_X(x_i)} \cdot f_X(x_i) \cdot g(x_i|x_{i+1}) &= \frac{f_X(x_{i+1})}{f_X(x_{i+1}) + f_X(x_i)} \cdot f_X(x_i) \cdot g(x_{i+1}|x_i) \\
f_X(x_{i+1}) \cdot \frac{f_X(x_i)}{f_X(x_i) + f_X(x_{i+1})} \cdot g(x_i|x_{i+1}) &= f_X(x_i) \cdot \frac{f_X(x_{i+1})}{f_X(x_{i+1}) + f_X(x_i)} \cdot g(x_{i+1}|x_i) \\
P(X = X_{i+1}) \cdot P(X_{i+1} \to X_i) &= P(X = X_i) \cdot P(X_i \to X_{i+1}) && \text{Def. Detailed Balance}
\end{aligned}
$$