Final Project Chloe&Mandy

Tianyu Zhan, Mandie Liu

2015.11.5

Problem 1 Metropolis-Hastings

```
set.seed(730)
mh.beta <- function(c,n.sims) {
  x.update <- function(x.cur,c) {
    # choosing candidate
    x.can <- rbeta(1, c*x.cur, c*(1-x.cur))
    # likelihood(Whether I should use d or p)
    a1 <- dbeta(x.can, 6,4)/dbeta(x.cur,6,4)
    # jumping distribution
    a2_num <- dbeta(x.cur, c*x.can, c*(1-x.can))
    a2_den <- dbeta(x.can, c*x.cur, c*(1-x.cur))
    a2 <- a2_num / a2_den
    # adjusted acceptance probability
    accept.prob <- (a2 * a1)
    if (runif(1) <= accept.prob) x.cur=x.can
    else x.cur
    return(x.cur)
  }
  draws <- c()
  x.cur <- runif(1)
  for (i in 1:n.sims) {
    draws[i] <- x.cur
    x.cur <- x.update(x.cur,c)
  }
  #draws <-draws[seq((burnIn+1),n.sims,by=30)]
  par(mfrow=c(1,3))
  plot(draws,ylim=c(0,1),main = paste("Trace plot: c = ", c, sep=""))
  acf(draws,ylim=c(0,1),main= paste("Autocorrelation Plot: c = ", c, sep=""))
  hist(draws, main=paste("Autocorrelation Plot: c = ", c, sep=""),freq=FALSE)
  comp <- ks.test(draws[!duplicated(draws)], "pbeta",6,4)
  print (comp)
  return()
}
```

To commit a Metropolis Hastling alogorithm we start from a random variable from uniform distribution. Then we jump to the candidate based on our proposal function

$$\phi_{prop}|\phi_{old} \sim Beta\left(c\phi_{old}, c\left(1-\phi_{old}\right)\right)$$

But here we should be aware of the fact that Beta distribution is not symmmetric, so we need to add a correction to the posterior factor, that is

$$\frac{J_t\left(\theta^{t-1}|\theta^*\right)}{J_t\left(\theta^*|\theta^{t-1}\right)}$$

For example,

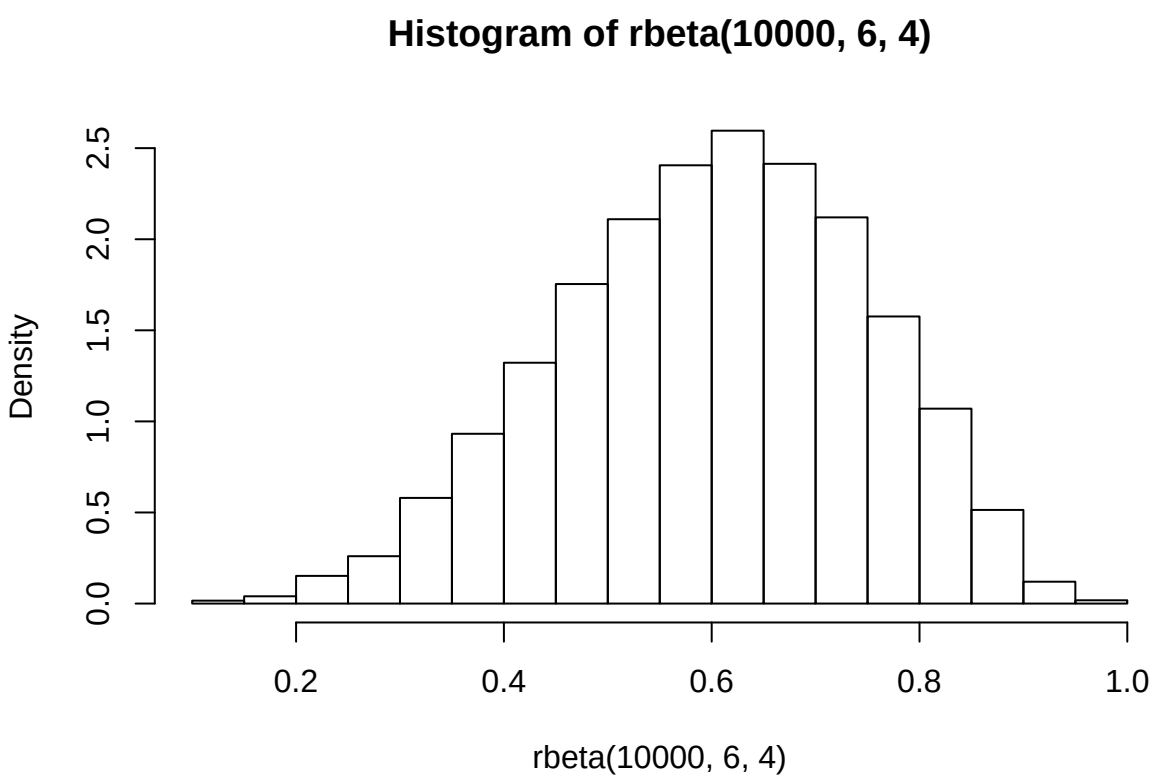
$$J_t\left(\theta^*|\theta^{t-1}\right) \propto Beta\left(c\theta^{t-1}, (1-c)\,\theta^{t-1}\right)$$

After that, we get acceptance ratio r,

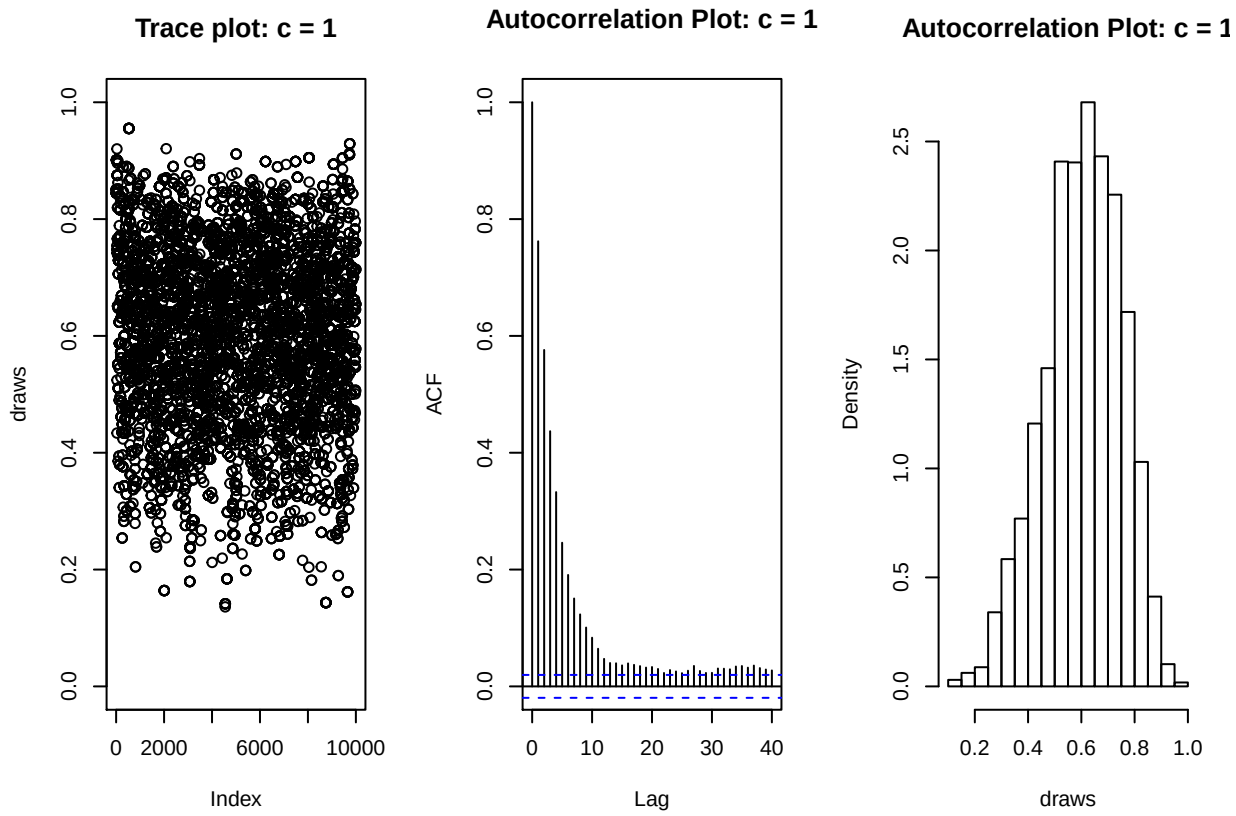
$$r = \frac{p\left(\theta^{*}|y\right)/J_t\left(\theta^{*}|\theta^{t-1}\right)}{p\left(\theta^{t-1}|y\right)/J_t\left(\theta^{t-1}|\theta^{*}\right)}$$

Then we compare r to u~ Uniform(0,1). If r > u we accept it and jump to the new point, or we remain unmoved at current state. The whole process loops designated times as iterations, and normally we need to do thining and burn in to reduce autocorrelation and get desired distribution.

```
hist(rbeta(10000,6,4),freq=FALSE)
```




```
mh.beta(1,10000)
```

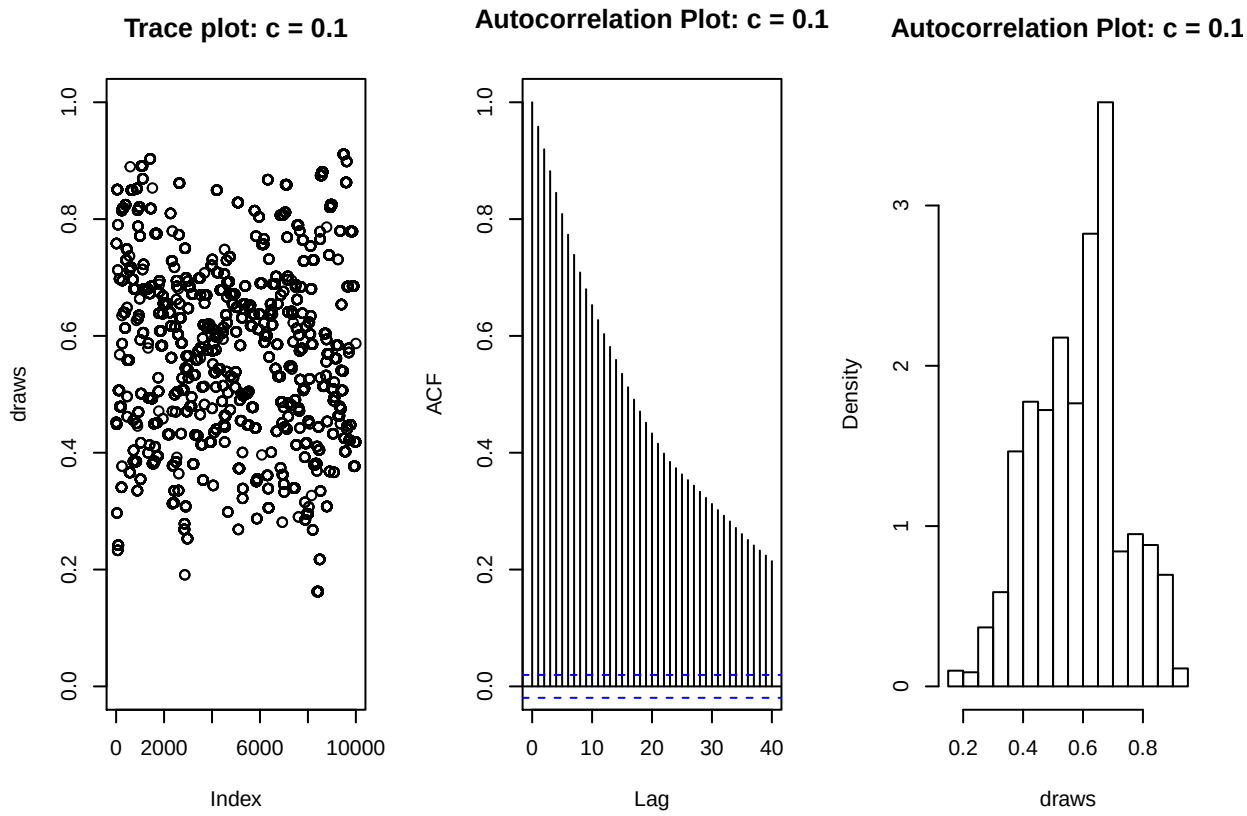



```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  draws[!duplicated(draws)]
## D = 0.025179, p-value = 0.08347
## alternative hypothesis: two-sided

## NULL
```

The histogram of our simulation is more sken to the left compared to beta(6,4) and p-value from K-S Test equals 2.027e-06, which rejects that our simulation comes from Beta(6,4).

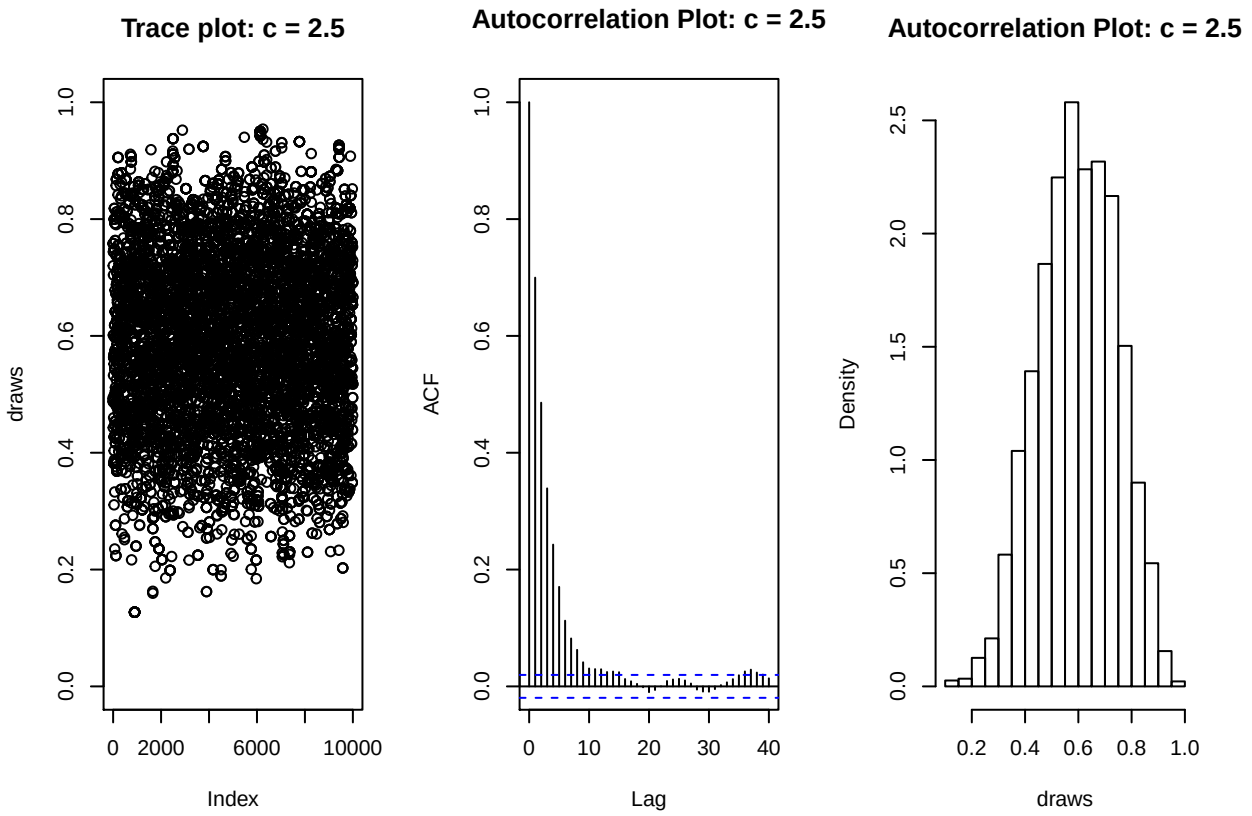
```
mh.beta(0.1,10000)
```




```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  draws[!duplicated(draws)]
## D = 0.12879, p-value = 1.556e-06
## alternative hypothesis: two-sided

## NULL
```

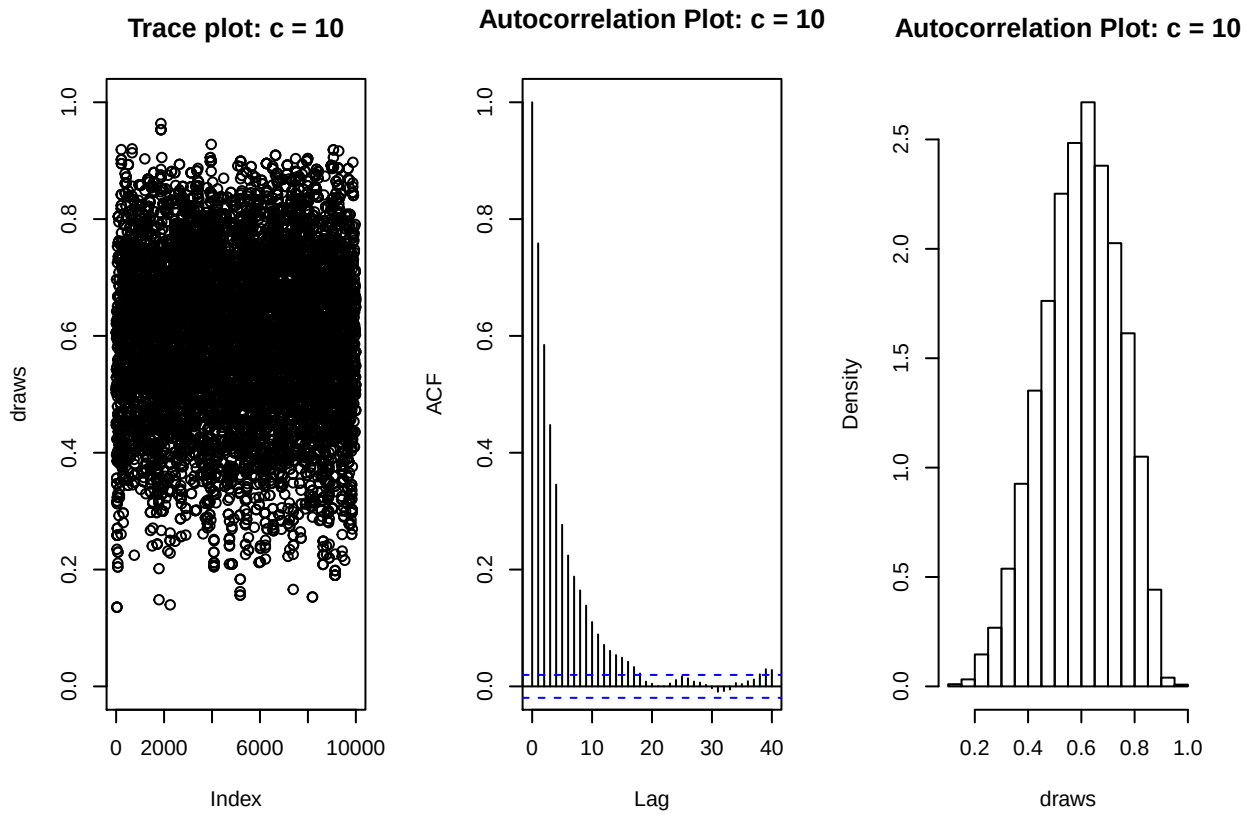
```
mh.beta(2.5,10000)
```




```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  draws[!duplicated(draws)]
## D = 0.054125, p-value = 3.387e-11
## alternative hypothesis: two-sided

## NULL
```

```
mh.beta(10,10000)
```




```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  draws[!duplicated(draws)]
## D = 0.03789, p-value = 8.488e-09
## alternative hypothesis: two-sided

## NULL
```

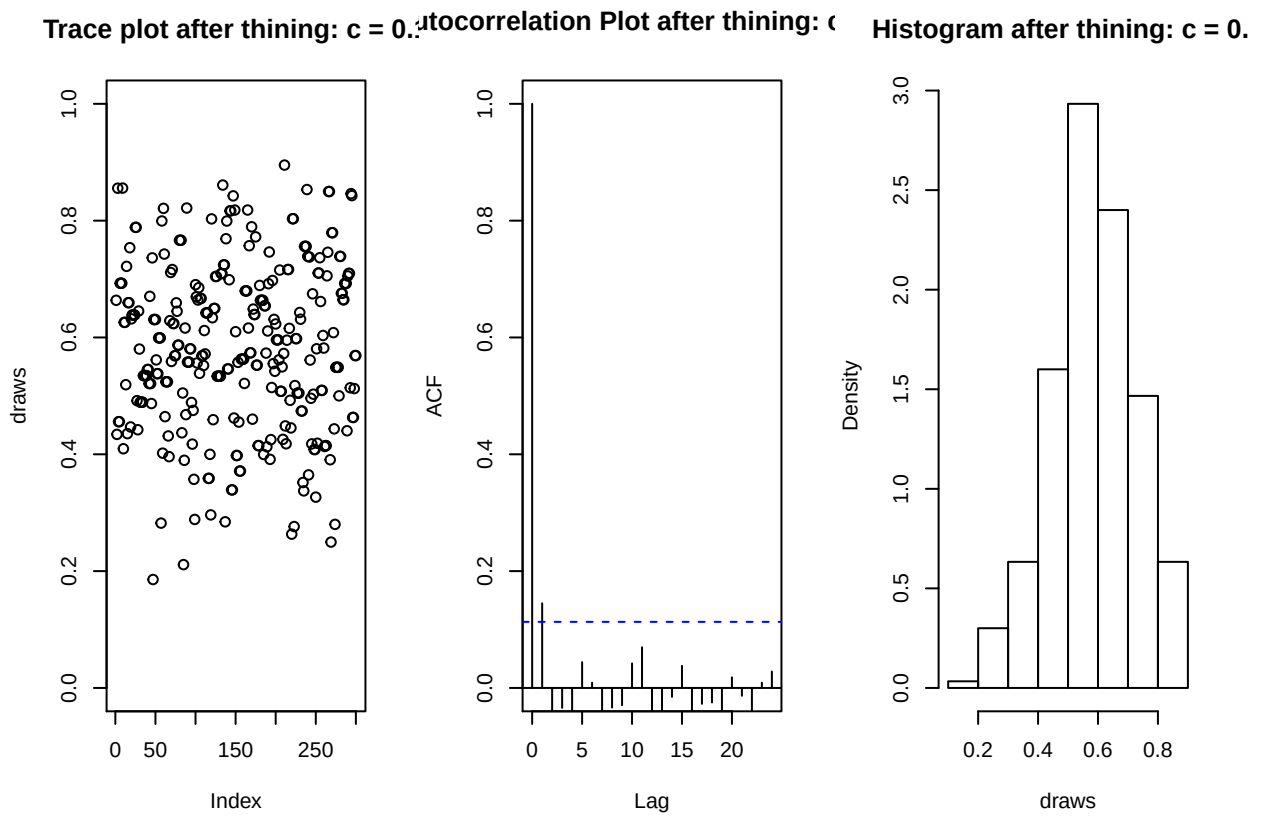
Overall, c=2.5 is the most effective. It has a nicer histogram and an autocorrelation plot. While for c=0.1 and c=10, we could see clearly that we need to do systematic sampling to thin the simulation. The histogram for c=0.1 is really wired, and overall distribution is far more unstable and fat-tail than beta distribution. C=10 is better than c=0.1 in histogram and the overall shape assembles Beta distribution. Overall c=2.5 is the best.

For further improve the performance for c=0.1 and c=10 I modify the code to add one line of burn-in and systematic sampling as follows:

```
set.seed(730)
thinmh.beta <- function(c,n.sims,burnIn) {
  x.update <- function(x.cur,c) {
    # choosing candidate
    x.can <- rbeta(1, c*x.cur, c*(1-x.cur))
    a1 <- dbeta(x.can, 6,4)/dbeta(x.cur,6,4)
    a2_num <- dbeta(x.cur, c*x.can, c*(1-x.can))
    a2_den <- dbeta(x.can, c*x.cur, c*(1-x.cur))
    a2 <- a2_num / a2_den
```

```
    # adjusted acceptance probability
    accept.prob <- (a2 * a1)
    if (runif(1) <= accept.prob) x.cur=x.can
    else x.cur
    return(x.cur)
  }
  draws <- c()
  x.cur <- runif(1)
  for (i in 1:n.sims) {
    draws[i] <- x.cur
    x.cur <- x.update(x.cur,c)
  }
  # Add an line of thining and burnIn.
  draws <-draws[seq((burnIn+1),n.sims,by=30)]
  par(mfrow=c(1,3))
  plot(draws,ylim=c(0,1),main = paste("Trace plot after thining: c = ", c, sep=""))
  acf(draws,ylim=c(0,1),main= paste("Autocorrelation Plot after thining: c = ", c, sep=""))
  hist(draws, main=paste("Histogram after thining: c = ", c, sep=""),freq=FALSE)
  comp <- ks.test(draws[!duplicated(draws)], "pbeta",6,4)
  print (comp)
  return()
}
```

```
thinmh.beta(0.1,10000,1000)
```

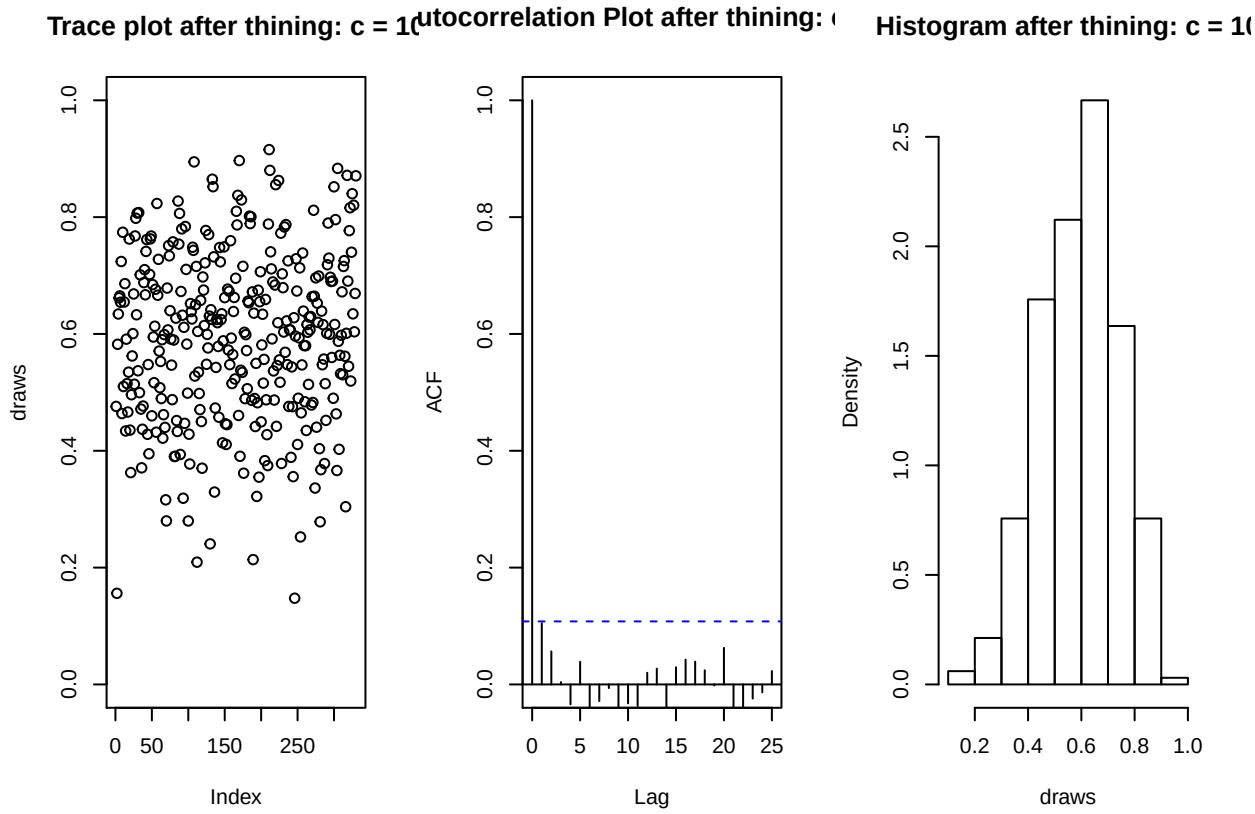



```
##
##  One-sample Kolmogorov-Smirnov test
```

```
##
## data:  draws[!duplicated(draws)]
## D = 0.089459, p-value = 0.0705
## alternative hypothesis: two-sided

## NULL
```

```
thinmh.beta(10,10000,100)
```

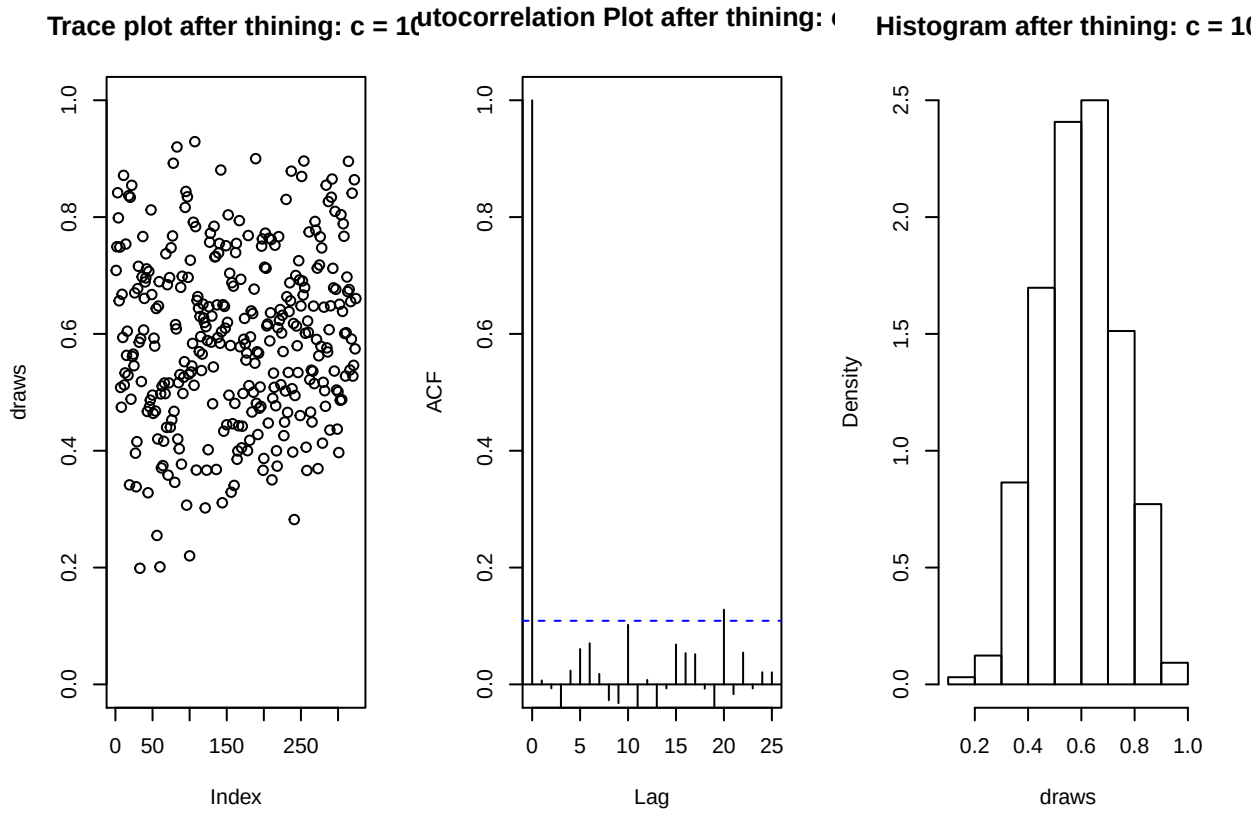



```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  draws[!duplicated(draws)]
## D = 0.037963, p-value = 0.7284
## alternative hypothesis: two-sided

## NULL
```

We could see from ACF plot that to reduce the dependency of variables, the case that c=0.1 needs more burnIn than c=10 to become stable. Also the p-value from K-S test when c=10 are higher compared to c=0.1.

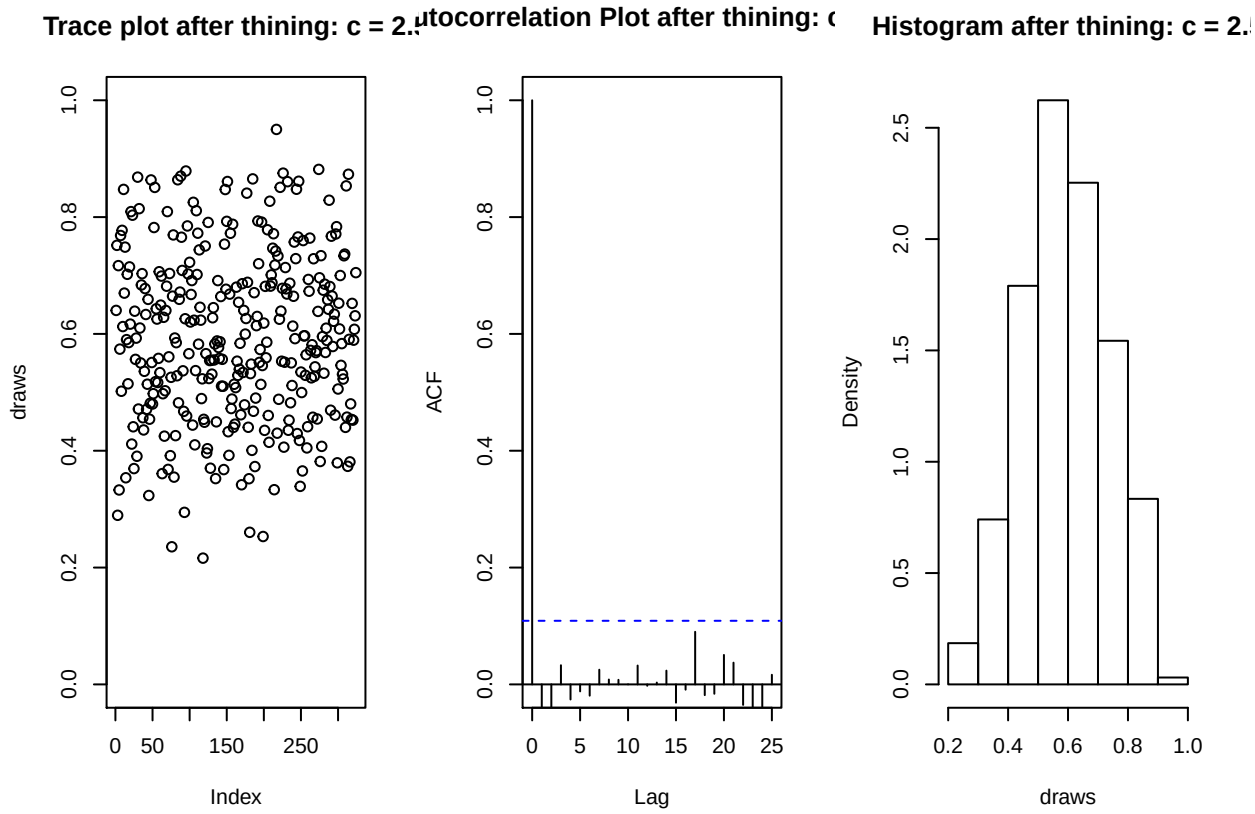
```
thinmh.beta(10,10000,300)
```




```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  draws[!duplicated(draws)]
## D = 0.051167, p-value = 0.3644
## alternative hypothesis: two-sided

## NULL
```

```
thinmh.beta(2.5,10000,300)
```




```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  draws[!duplicated(draws)]
## D = 0.056533, p-value = 0.2516
## alternative hypothesis: two-sided

## NULL
```

After thining c=10 rather than c=2.5 performs the best. To improve the efficiency of MCMC, it's better to choose a proposal function that is close to our target distribution. As for Beta distribution mean = 6/(6+4)=0.6, when c=10 we will sample the most portion of beta(10*0.6, 10*(1-0.6)), so in this sense when it reaches stationary, c=10 should perform the best. Thus when we have larger burnIn c=10 is the best, for example when burnIn=300.It is supported by nice histogram and a larger p-value. However it takes longer for c=10 than c=2.5 to reach stationary, for example when burnin=100, p-value(c=10)=0.33 < p-value(c=2.5)=0.63, but the histogram for c=10 is still the best. Overall to ensure that we are sampling from Beta(6,4), we could add burnin to our code and choose c=10.

Problem 2 Gibbs Sampling

Since $p\left(x|y\right) \propto ye^{-yx}, 0 < x < B < \infty$, we can get

$$\int_0^B ye^{-yx}\, dx = 1 - e^{-yB} \propto 1$$

In order to get an integral that equals 1, we can get the conditional probability of x as

$$p(x|y) = \frac{ye^{-yx}}{1-e^{-yB}}$$

Calculate the CDF:

$$F(x|y) = \int_0^x p(x|y)\ dx = \frac{1-e^{-yx}}{1-e^{-yB}}, 0 < x < B < \infty$$

Then, get the inverse function of CDF:

$$F^{-1}(u) = -\frac{\log\left(1-u\left(1-e^{-yB}\right)\right)}{y}$$

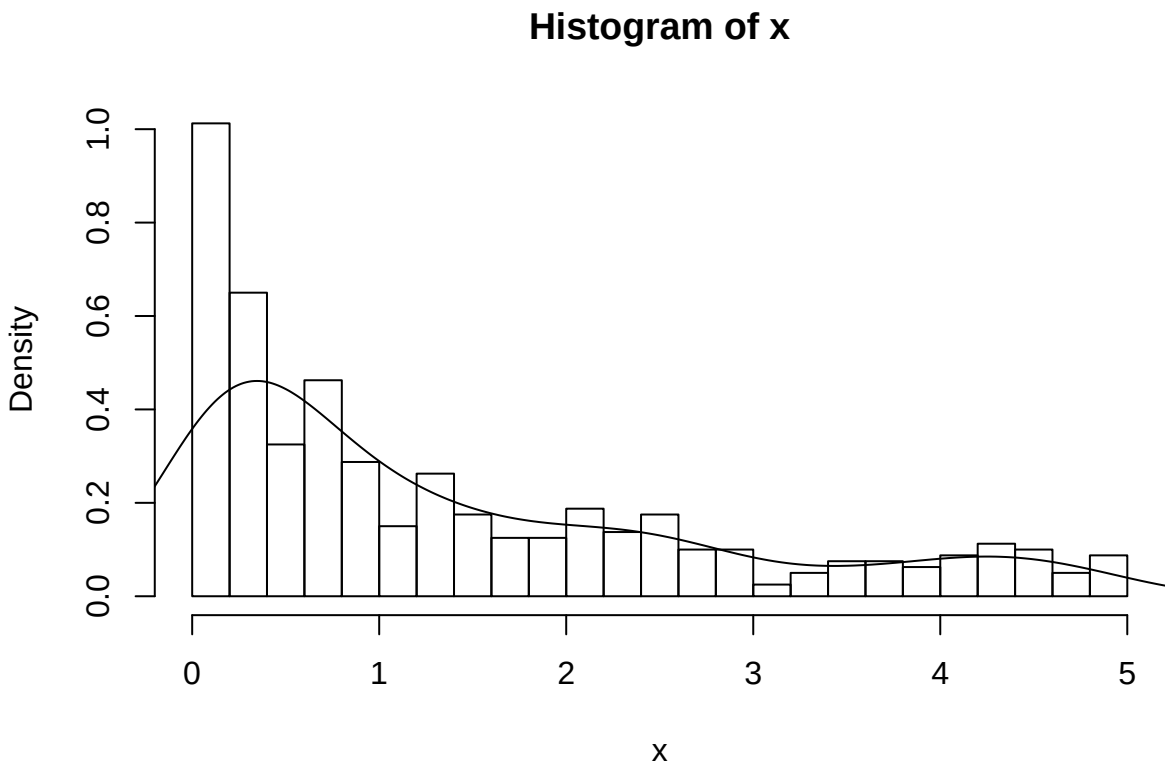
We can generate samples of x using this formula. The generation of samples of y is of the same logic, since $p(x|y)$ and $p(y|x)$ are symmetric.
The followring R code shows how Gibbs Sampling works.

```
Gibbs=function(n,B,burnin){
  x.update=c()
  y.update=c()
  y.update[1]=runif(1)
  # randomly generate a starting value of y
  for (i in 1:n){
    x.update[i]=-log(1-runif(1)*(1-exp(-y.update[i]*B)))/y.update[i]
    # using the new y value to update x
    y.update[i+1]=-log(1-runif(1)*(1-exp(-x.update[i]*B)))/x.update[i]
    # using the new x value to update y
  }
  x=x.update[-(1:burnin)]
  y=y.update[-(1:burnin)]
  # drop the initial values
  hist(x,freq=F,breaks=20)
  lines(density(x))
  return(mean(x))
  # caulculate the estimate of expectation of x
}
```

If we choose B=5, then for sample sizes T=500, 5000, 50000, we can get three histograms. The estimate of the expectation of x is

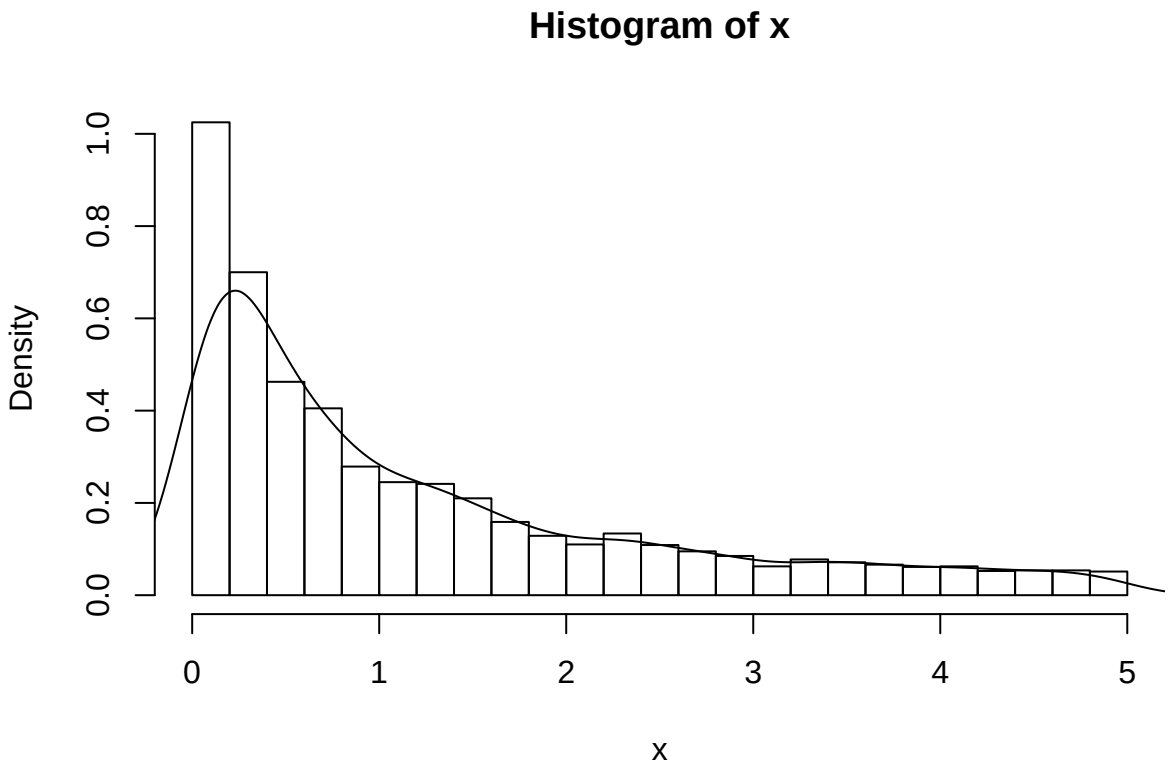
$$E_{p(X)}(X) = \frac{1}{n}\sum_{i=1}^{n} X_i = \bar{X}$$

```
set.seed(526)
# choose 20% of the sample size as "burnin"
Gibbs(500,5,100)
```



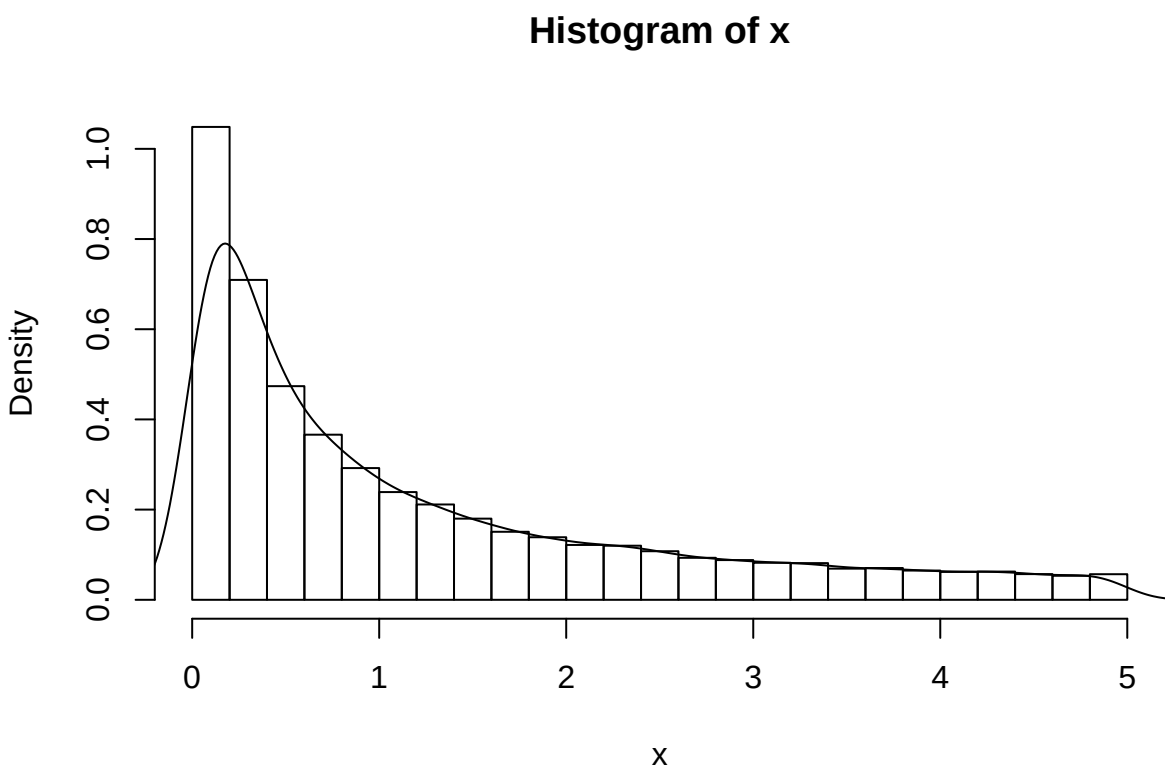

```
## [1] 1.385384
```

```
Gibbs(5000,5,1000)
```




```
## [1] 1.238268
```

```
Gibbs(50000,5,10000)
```

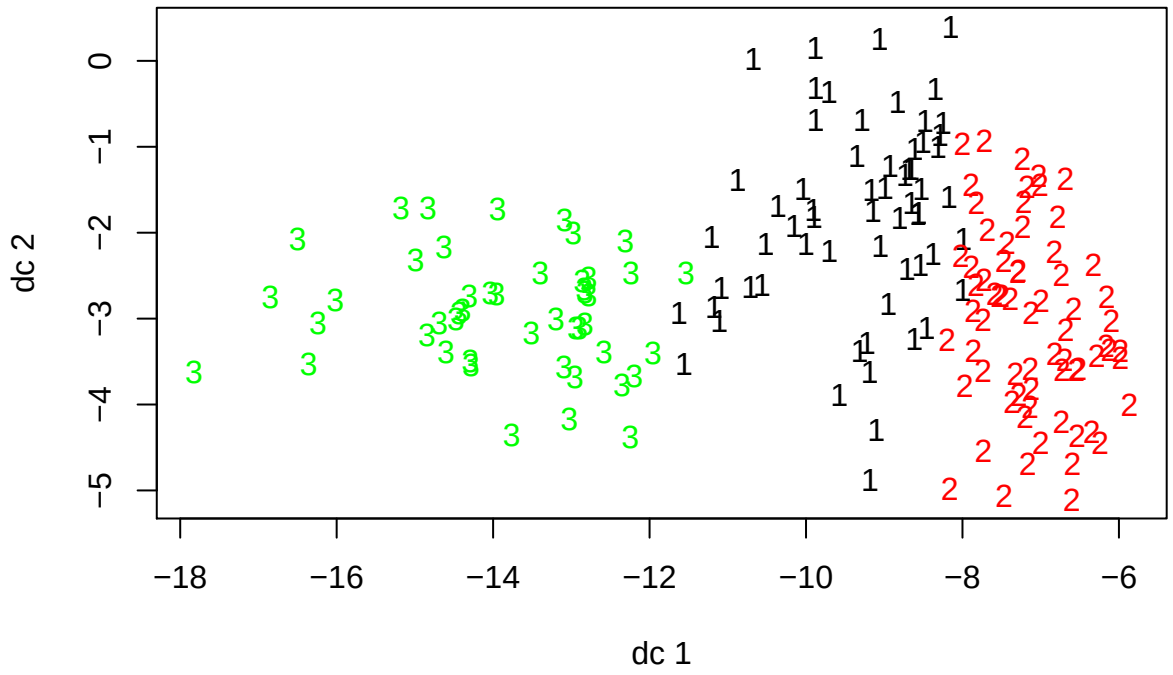



```
## [1] 1.254668
```

K-Means Part I

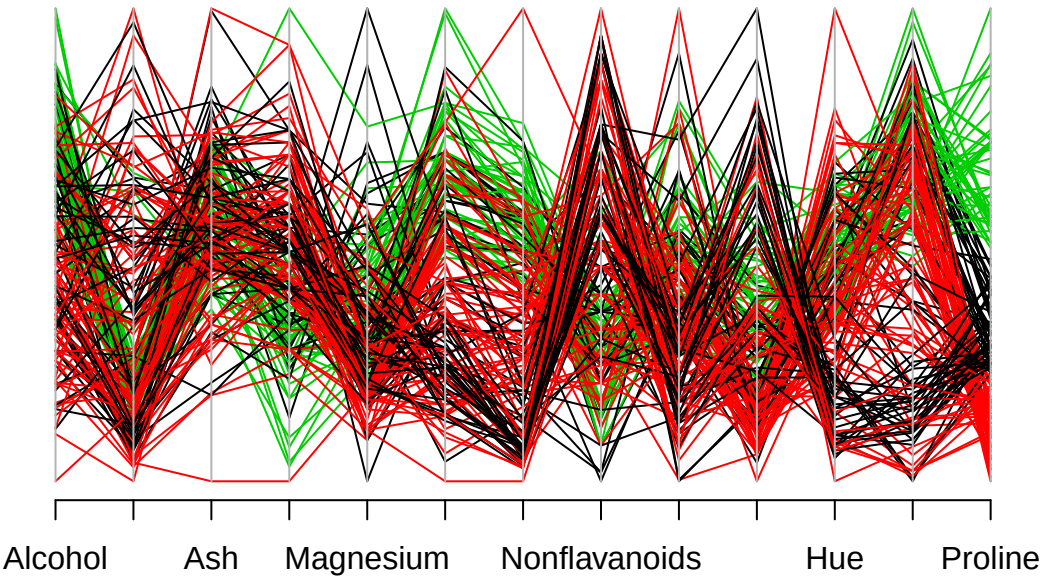
Use k-means method to cluster the wines into 3 groups.

```
# the default measure to calculate k-means is Euclidean distance
library('fpc')
plotcluster(wine[-1],fit$cluster)
```



From the graph we can see that the boundaries are not very clear, which indicates that some observations can be classified into either groups. Draw parallel coorinates plot to see how variables contributed in each cluster.

```
library(MASS)
parcoord(wine[-1],fit$cluster)
```



We cannot observe a clear pattern from this graph.

Take a look at the misclassification rate (number of misclassified obs divided by total number of obs).

```
##
##       1  2  3
##    1 13  0 46
##    2 20 50  1
##    3 29 19  0
```

There are 145(=46+20+29+50) out of 178 obs that have been misclassified, so the misclassification rate is 0.81.

To take a further look at how well the kmeans algorithm did, we use ARI (Adjusted Rand Index) as an evaluation criteria.

$$ARI = \frac{\Sigma_{ij}\binom{n_{ij}}{2} - [\Sigma_i\binom{a_i}{2}\Sigma_j\binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\Sigma_i\binom{a_i}{2} + \Sigma_j\binom{b_j}{2}] - [\Sigma_i\binom{a_i}{2}\Sigma_j\binom{b_j}{2}]/\binom{n}{2}}$$

where n_{ij} denotes for the ith row and jth column in a contigency table; a_i denotes for the sum of ith row and b_j denotes for the sum of jth column.

```
## Loading required package: grid
## Loading required package: lattice
## Loading required package: modeltools
## Loading required package: stats4
```

```
##       ARI
## 0.3711137
```

A small ARI indicates a poor classification.

We also want to use another internal index **Silhouette** to evaluate the clustering result. Silhouette s(x):

$$s(x) = \frac{b(x) - a(x)}{max\{a(x), b(x)\}}$$

where cohesion a(x) is the average distance of x to all other vectors in the same cluster, and separation b(x) is the average distance of x to the vectors in other clusters. Its values range from -1 to 1. A high silhouette value indicates that i is well-matched to its own cluster, and poorly-matched to neighboring clusters. If most points have a high silhouette value, then the clustering solution is appropriate. If many points have a low or negative silhouette value, then the clustering solution may have either too many or too few clusters.

```
library(clusterCrit)
# an open-source package to compute all the clustering indices
intIdx=intCriteria(as.matrix(wine[-1]),fit$cluster,"all")
ccfit=cclust(as.matrix(wine[-1]),3)
```

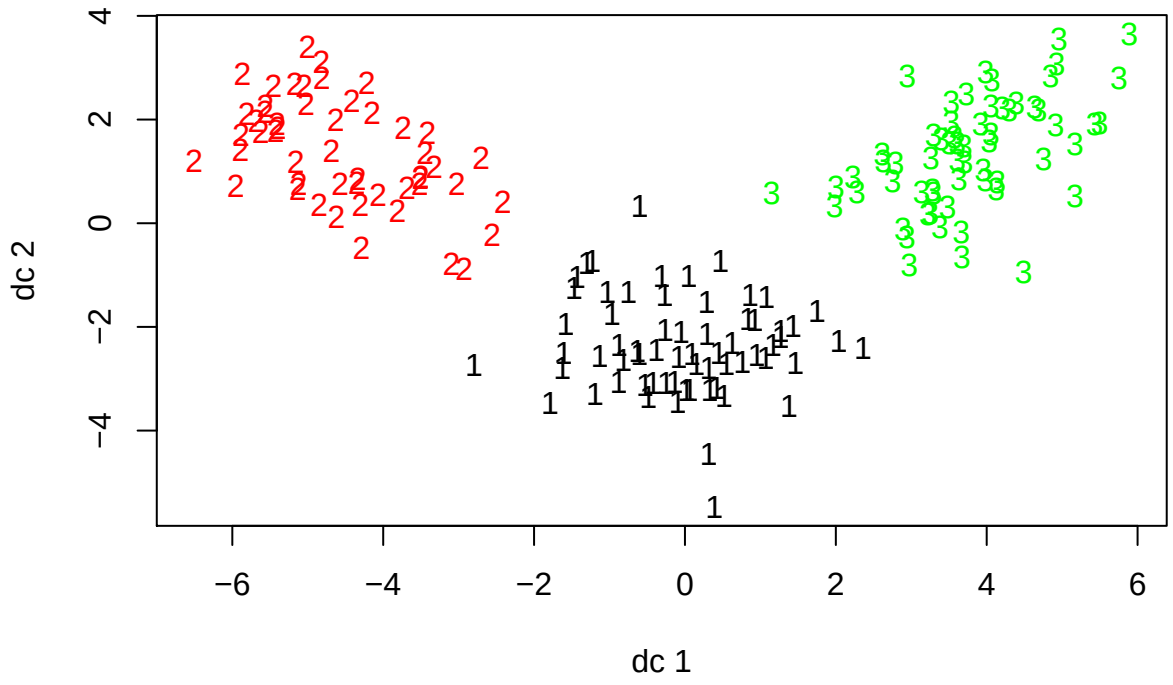
```
## Found more than one class "kcca" in cache; using the first, from namespace 'kernlab'
## Found more than one class "kcca" in cache; using the first, from namespace 'kernlab'
```

```
set.seed(526)
plot(shadow(ccfit))
```

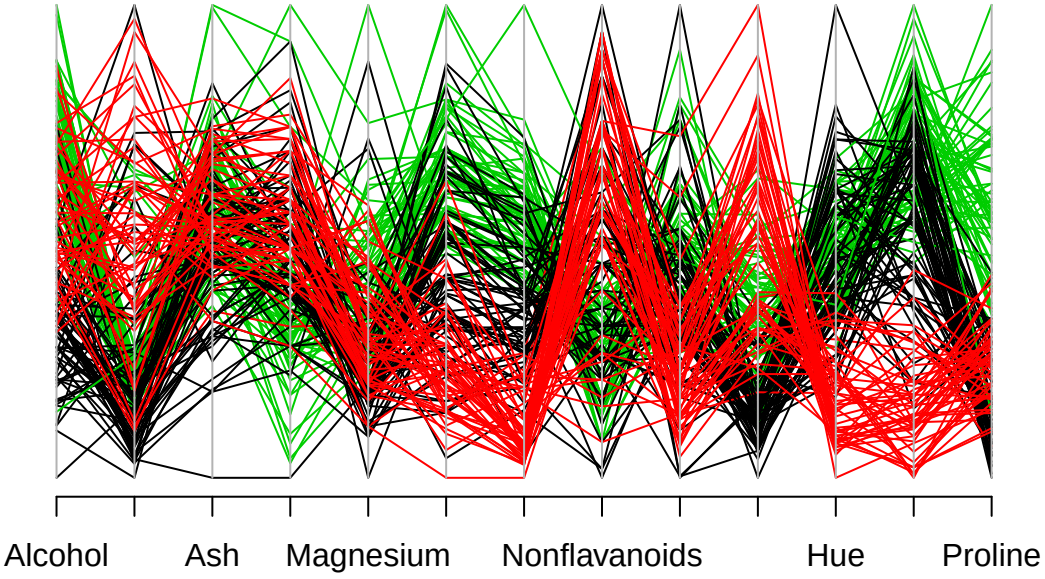
From the plot we can see that in each cluster, there are some observations that have small silhouette value, which indicates a not-so-good classification.

We want to improve the performance by **scaling and centering** (scale() function can do this work) the data.

```
fit2=kmeans(scale(wine[-1]),3)
plotcluster(scale(wine[-1]),fit2$cluster)
```




```
parcoord(scale(wine[-1]),fit2$cluster)
```




```
mis2=table(wine$Type,fit2$cluster)
mis2
```

```
##
##       1  2  3
##   1   0  0 59
##   2 65  3  3
##   3  0 48  0
```

```
randIndex(mis2)
```

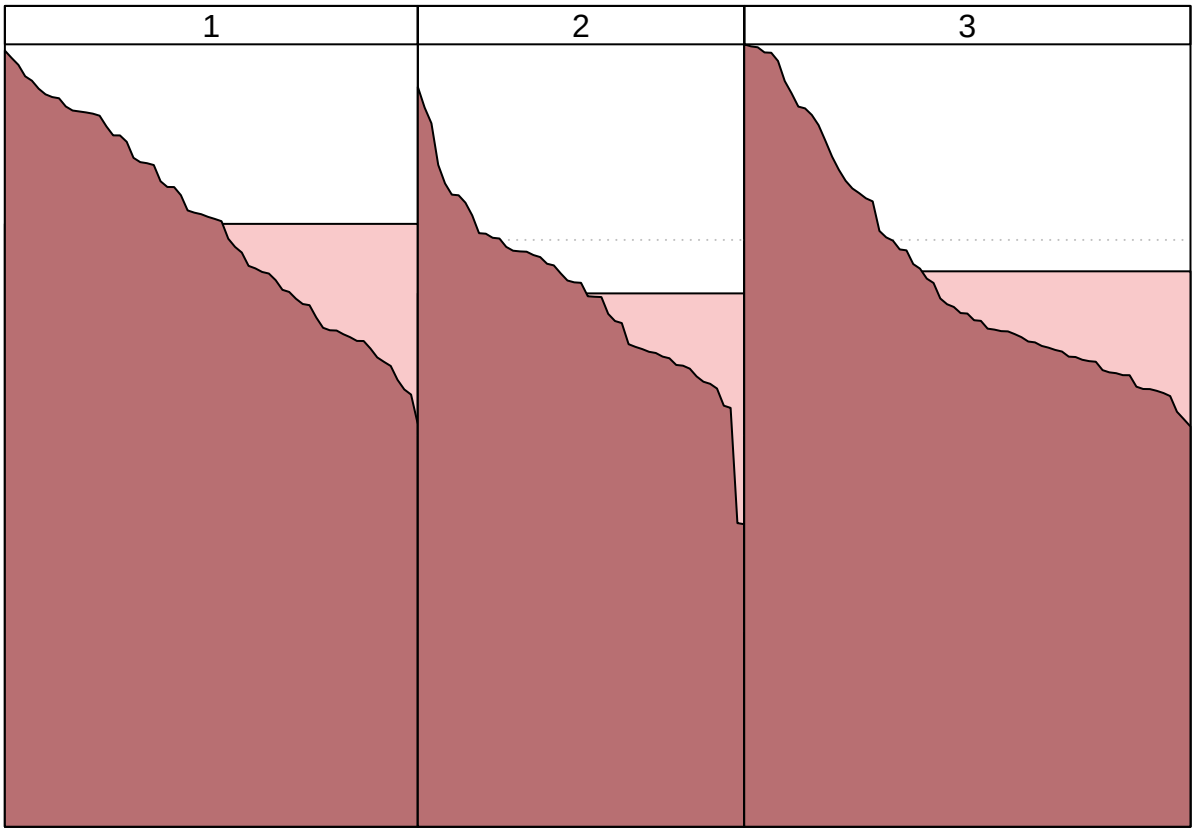
```
##      ARI
## 0.897495
```

We can see the data is clustered very well, with no collapse between clusters.

From the graph, we may consider that black cluster contains wine with low flavanoids value, low proanthocyanins value, low hue value; or green cluster contains wine which has dilution value higher than wine in red cluster.

This time, the misclassification rate is 0.034, much better than the previous result. ARI is 0.897, also increased a lot.
Therefore, scaling the data can improve the clustering quality.

```
intIdx2=intCriteria(as.matrix(scale(wine[-1])),fit2$cluster,"all")
ccfit2=cclust(as.matrix(scale(wine[-1])),3)
plot(shadow(ccfit2))
```



From the plot, we can see that most observations in each cluster have a large silhouette value, which suggests that the classification is very good.

In addition to K-means, we use **Hierarchial Clustering**–another commonly used clustering method–as an alternative method. First we consider the unscaled data.

```
wine_dist=dist(wine[-1])
fit_h=hclust(wine_dist)
```

```
h_cluster=cutree(fit_h,k=3)
mis3=table(wine$Type,h_cluster)
randIndex(mis3)
```

```
##      ARI
## 0.370833
```

Follow the same procedures as above, we find the ARI derived from Hierarchial Clustering is 0.371, almost the same as K-means.
Then we try on scaled data.

```
wine_dist2=dist(scale(wine[-1]))
fit_h2=hclust(wine_dist2)
h_cluster2=cutree(fit_h2,k=3)
mis4=table(wine$Type,h_cluster2)
randIndex(mis4)
```

```
##       ARI
## 0.5771436
```

The ARI is only 0.577, much smaller than the result (0.897) we get from K-means. Therefore, in terms of scaled data, we think K-means is better than Hierarchial Clustering, since it has a smaller misclassification rate and larger ARI.

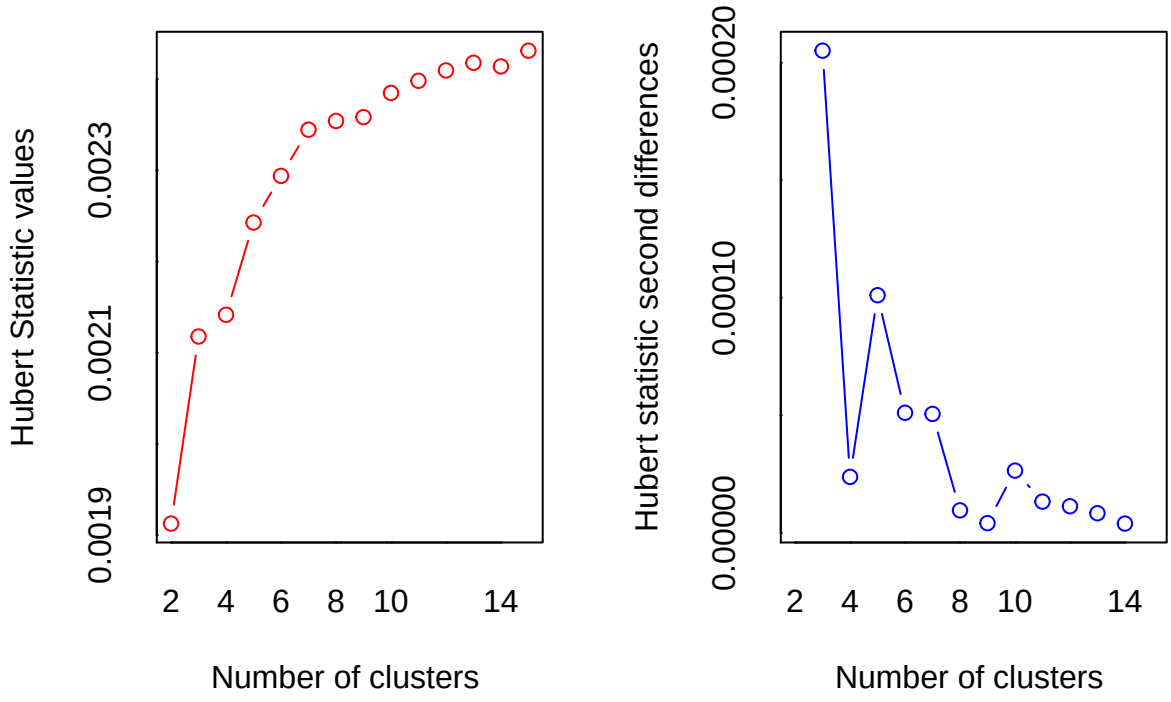
K-Means Part II Iris dataset

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

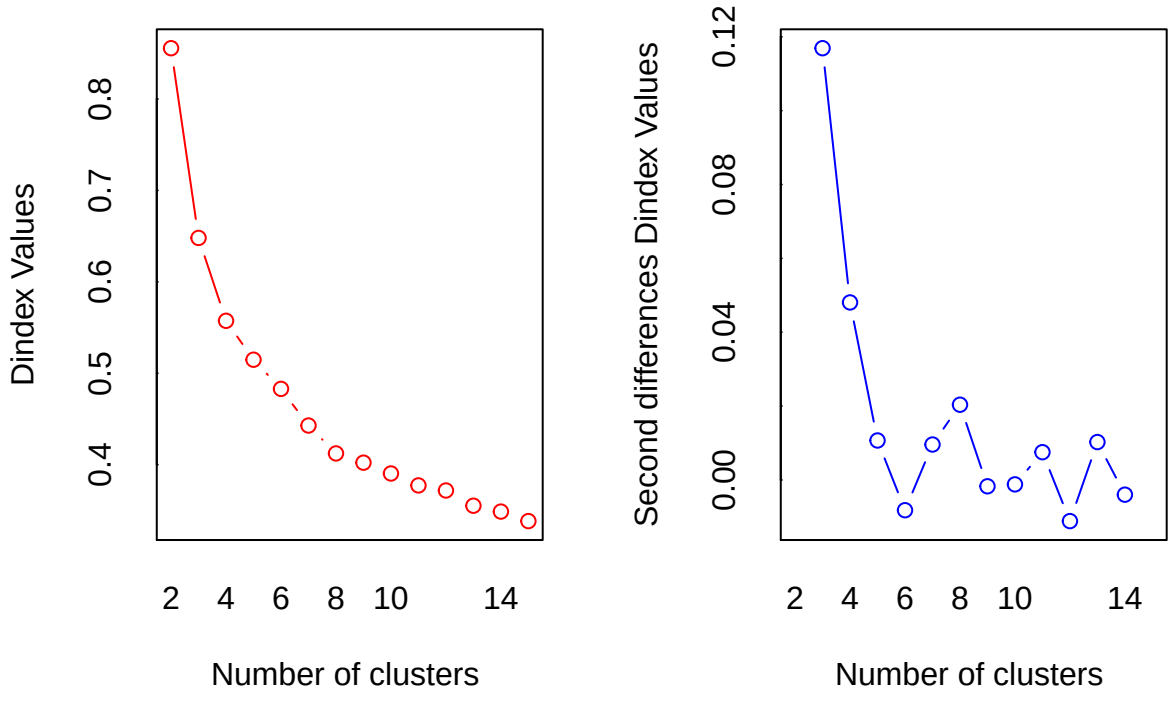
```
# choosing number of clusters
require("NbClust")
```

```
## Loading required package: NbClust
```

```
nc <- NbClust(irisnew, min.nc=2, max.nc=15, method="kmeans")
```



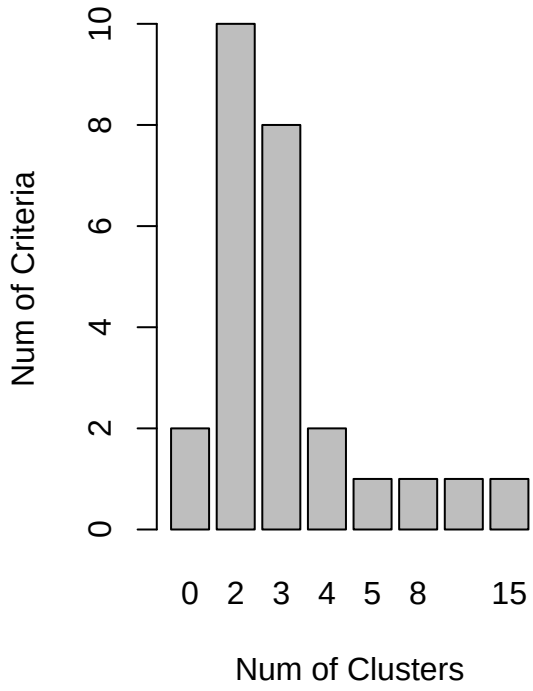

```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```




```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
```

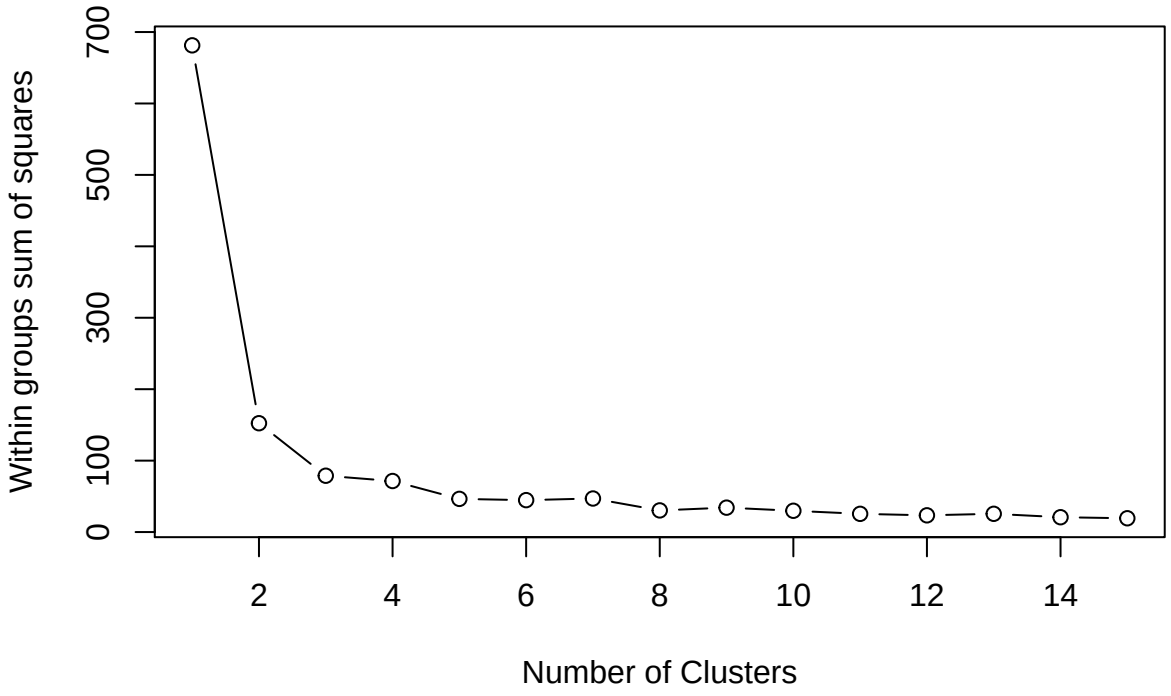
```
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## ******************************************************************
## * Among all indices:
## * 10 proposed 2 as the best number of clusters
## * 8 proposed 3 as the best number of clusters
## * 2 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 14 as the best number of clusters
## * 1 proposed 15 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## ******************************************************************
```

```
barplot(table(nc$Best.n[1,]),xlab="Num of Clusters", ylab="Num of Criteria")
```



We could see from the plot that 2 clusters are the best. This also agrees with the barplot results with 2 clusters being supported the most by 10 indicies. However, 3 clusters are the second suggested one with 8 proposed indices.

```
wss <- 0
for (i in 1:15){
  wss[i] <-sum(kmeans(irisnew, centers=i)$withinss)
}
plot(1:15,wss,type="b", xlab="Number of Clusters", ylab="Within groups sum of squares")
```



The sum of square error (SSE) screen plot also shows that cluster=2 is the significant bend. Both methods suggest k=2 is the best, but as there are 3 types in iris, we continue to try 2 and 3 clusters respectively.

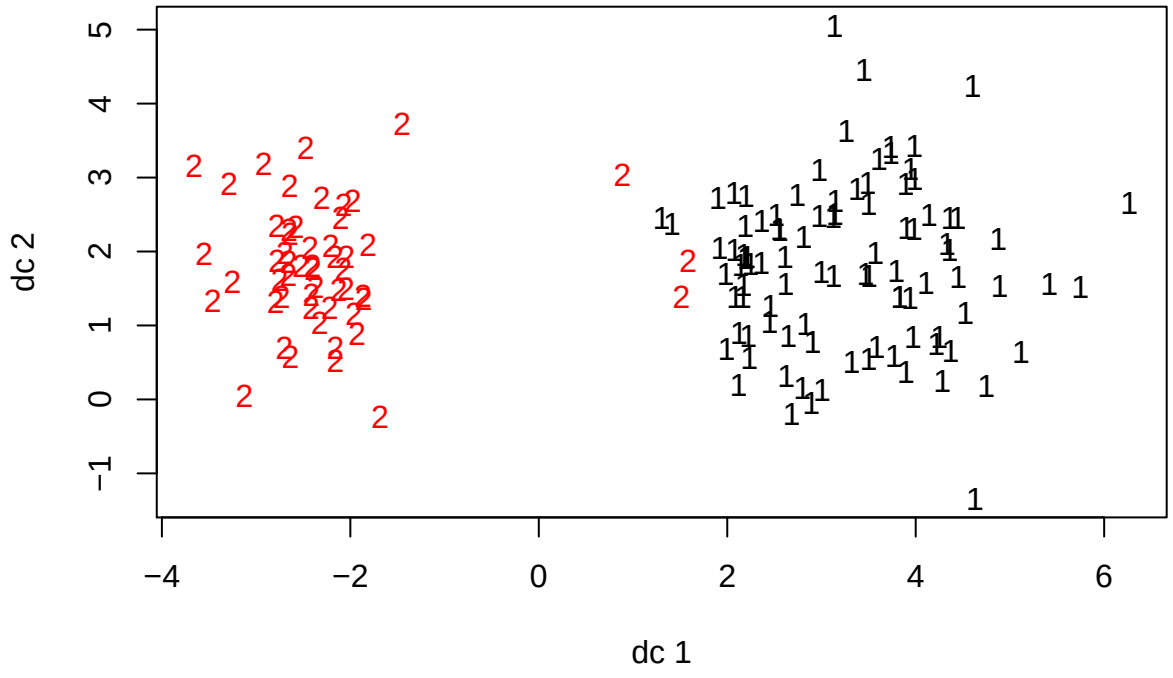
```
fit.kmiris2 <- kmeans(irisnew,2)
fit.kmiris2
```

```
## K-means clustering with 2 clusters of sizes 97, 53
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     6.301031    2.886598     4.958763    1.695876
## 2     5.005660    3.369811     1.560377    0.290566
##
## Clustering vector:
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 123.79588  28.55208
##  (between_SS / total_SS =  77.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

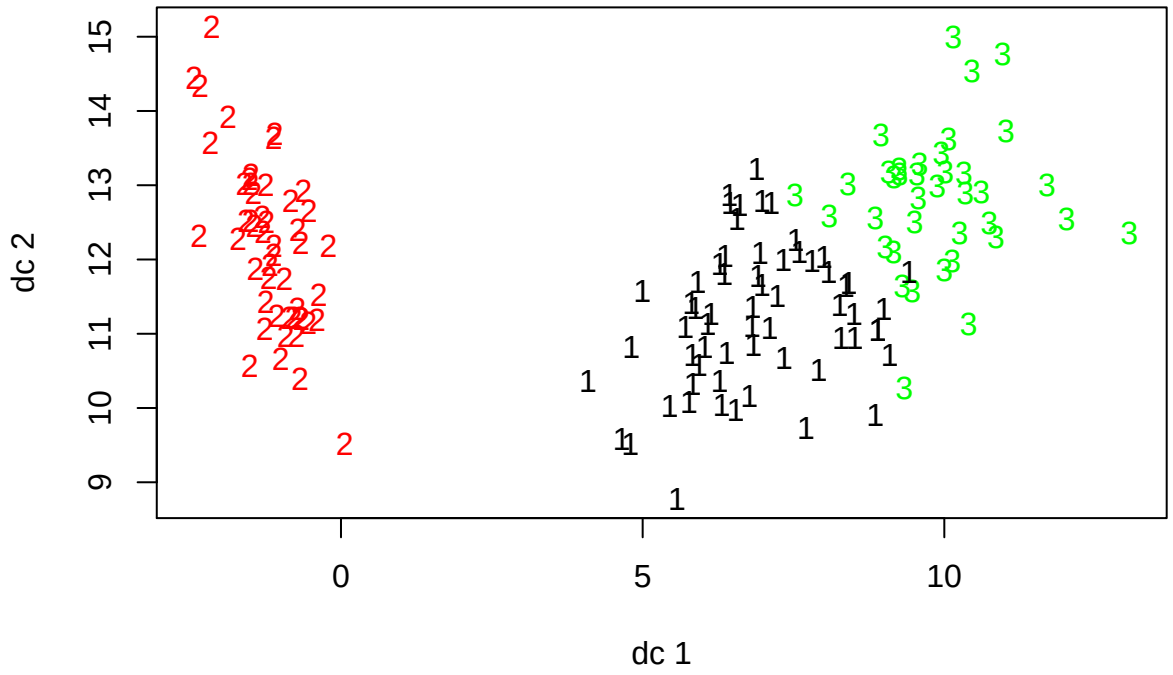
```
fit.kmiris3 <- kmeans(irisnew,3)
fit.kmiris3
```

```
## K-means clustering with 3 clusters of sizes 62, 50, 38
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.901613    2.748387     4.393548    1.433871
## 2     5.006000    3.428000     1.462000    0.246000
## 3     6.850000    3.073684     5.742105    2.071053
##
## Clustering vector:
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3 3 3
## [106] 3 1 3 3 3 3 3 3 1 1 3 3 3 3 1 3 1 3 1 3 3 1 1 3 3 3 3 3 1 3 3 3 3 1 3
## [141] 3 3 1 3 3 3 1 3 3 1
##
## Within cluster sum of squares by cluster:
## [1] 39.82097 15.15100 23.87947
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
library(fpc)
plotcluster(irisnew, fit.kmiris2$cluster)
```



```
plotcluster(irisnew, fit.kmiris3$cluster)
```



when cluster=2, (between_SS / total_SS = 77.6 %) thus the overall difference is very large between groups. Also, there are almost no collapse between clusters, but there are 3 data points in group 2 very close to group 1.
When cluster=3, (between_SS / total_SS = 88.4 %) yields a better difference across groups. There is some collapse between group1 and 3 but overall it makes more sense because it's more like the real case.
Also, it's noticeable that the two clustering numbers give very different results.

```
a=table(iris$Species,fit.kmiris2$cluster)
b=table(iris$Species,fit.kmiris3$cluster)
a
```

```
##
##                1  2
##    setosa      0 50
##    versicolor 47  3
##    virginica  50  0
```

```
b
```

```
##
##                1  2  3
##    setosa      0 50  0
##    versicolor 48  0  2
##    virginica  14  0 36
```

```
library(flexclust)
randIndex(a)
```

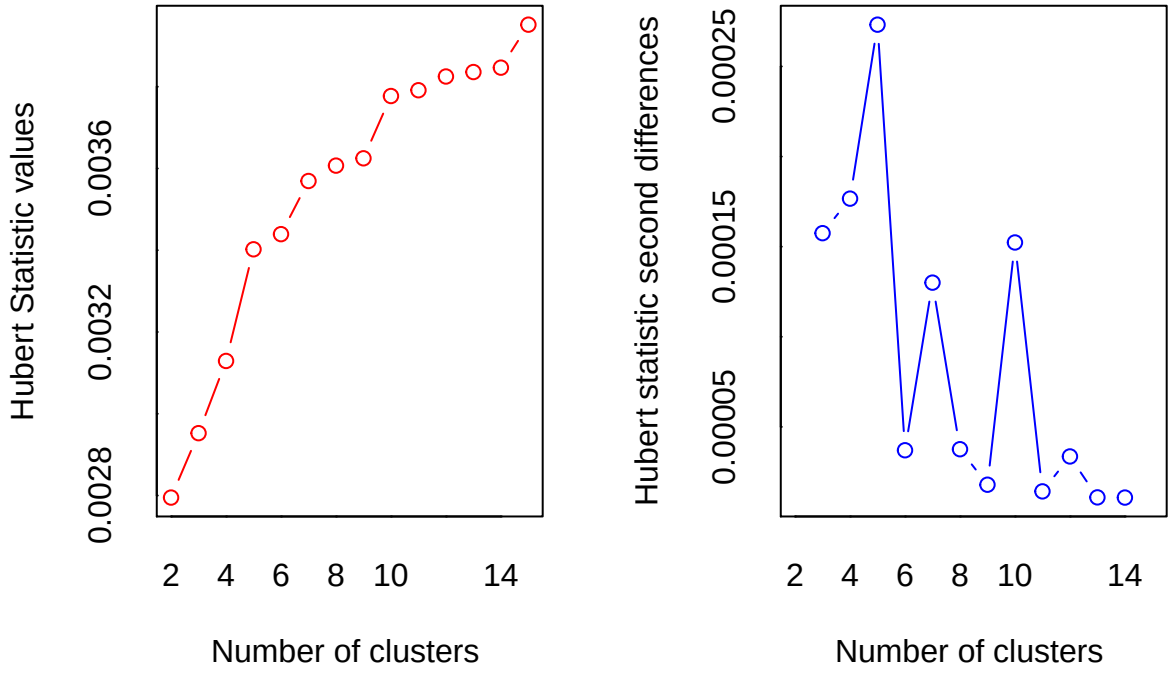
```
##       ARI
## 0.5399218
```

```
randIndex(b)
```

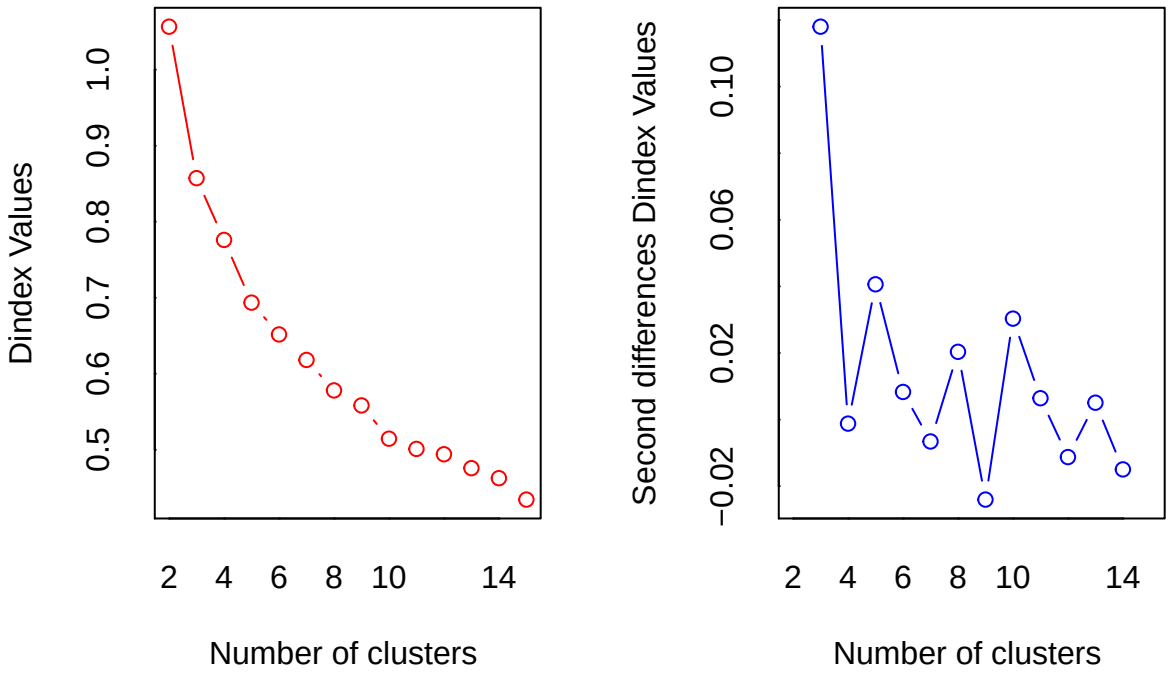
```
##       ARI
## 0.7302383
```

A small ARI indicates a poor classification.
When cluster=2, ARI is only 0.54, which is conceivable because there are only 2 clusters compared to the original 3 groups.
When cluster=3, ARI is 0.73, which is still not satisfatory so I continue to scale the data.

```
nc <- NbClust(scale(iris[-5]), min.nc=2, max.nc=15, method="kmeans")
```

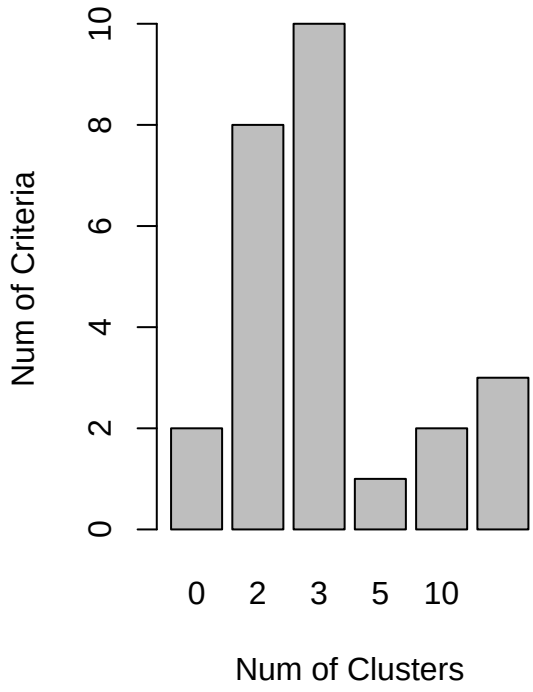



```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```



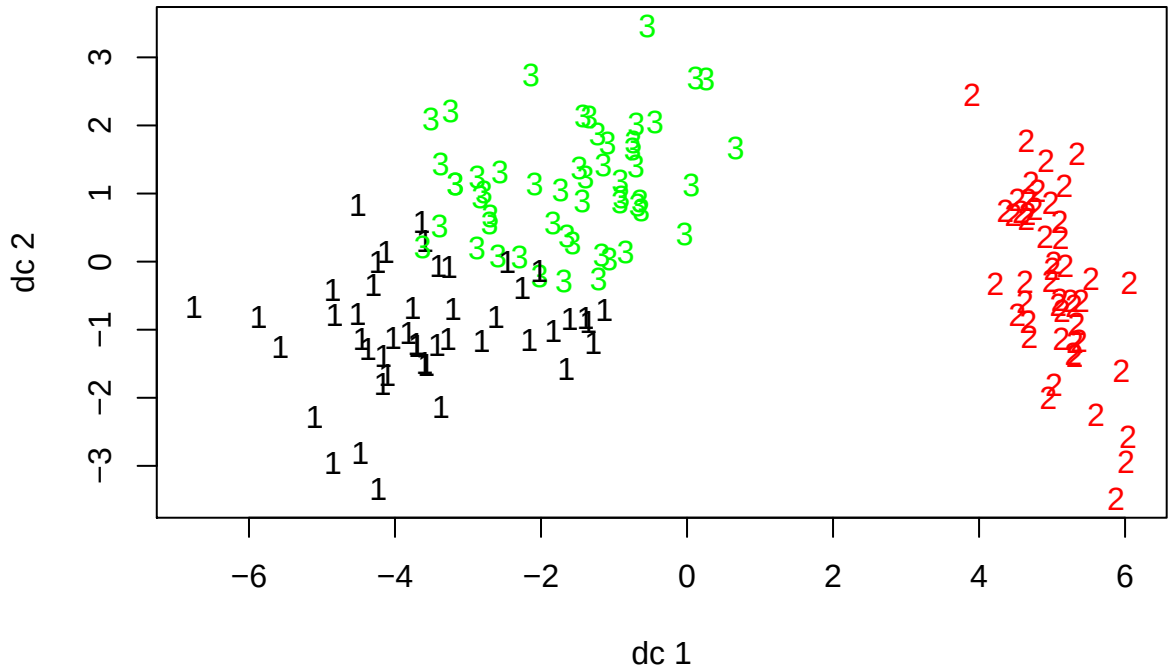

```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 8 proposed 2 as the best number of clusters
## * 10 proposed 3 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
## * 3 proposed 15 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  3
##
##
## *******************************************************************
```

```
barplot(table(nc$Best.n[1,]),xlab="Num of Clusters", ylab="Num of Criteria")
```



After scaling, the two methods all suggest cluster number to be 3.

```
fit2=kmeans(scale(iris[-5]),3)
plotcluster(scale(iris[-5]), fit2$cluster)
```



The plot looks like the unscaled plot with cluster=3, however the boundary is even more blur.

```
c=table(iris$Species,fit2$cluster)
library(flexclust)
randIndex(c)
```

```
##       ARI
## 0.6201352
```

After scaling, ARI drops to 0.62. K-means seems not a good classfication method for iris and the scaling doesn't help. It is because the predictor in iris are similar in terms of sacle.