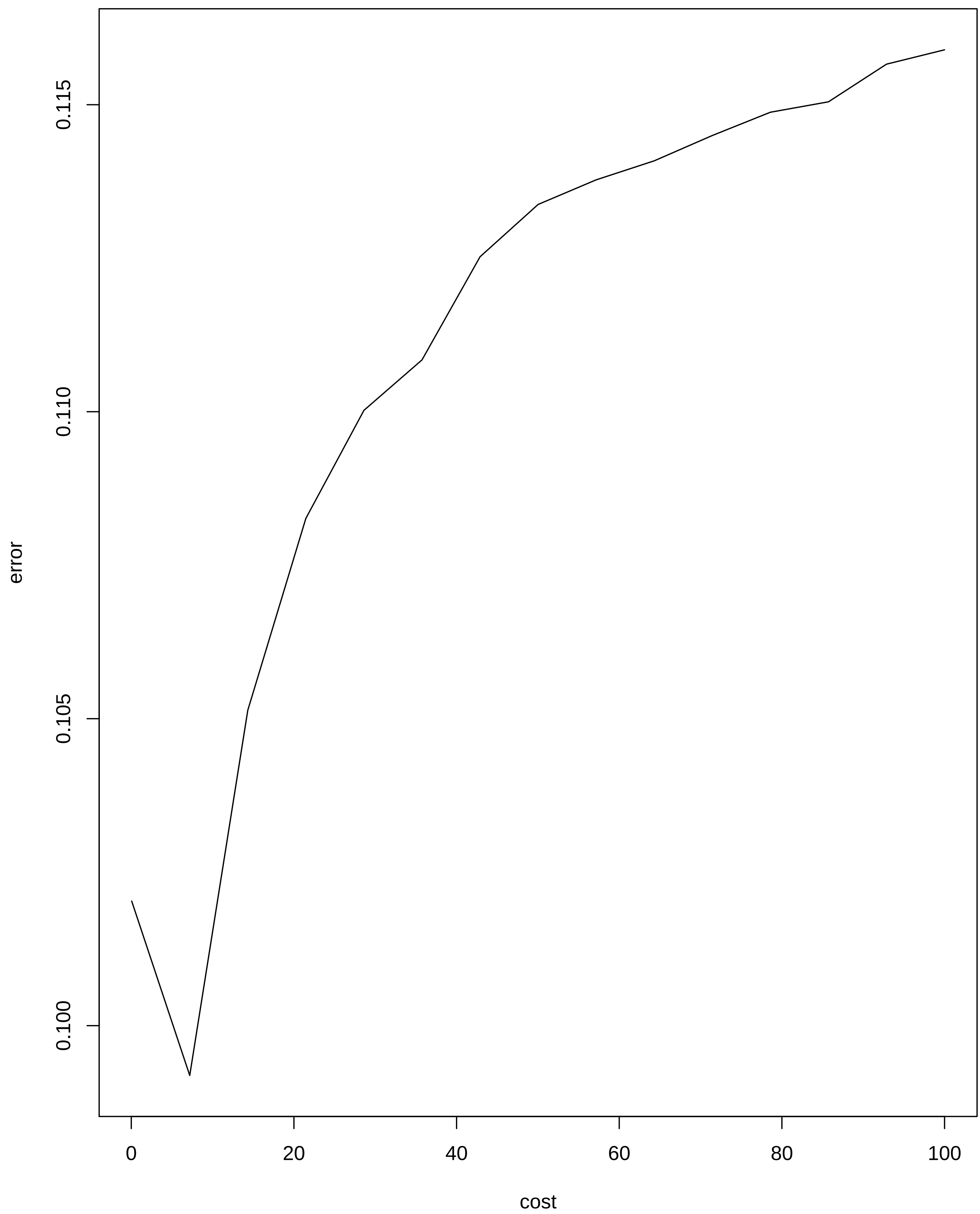
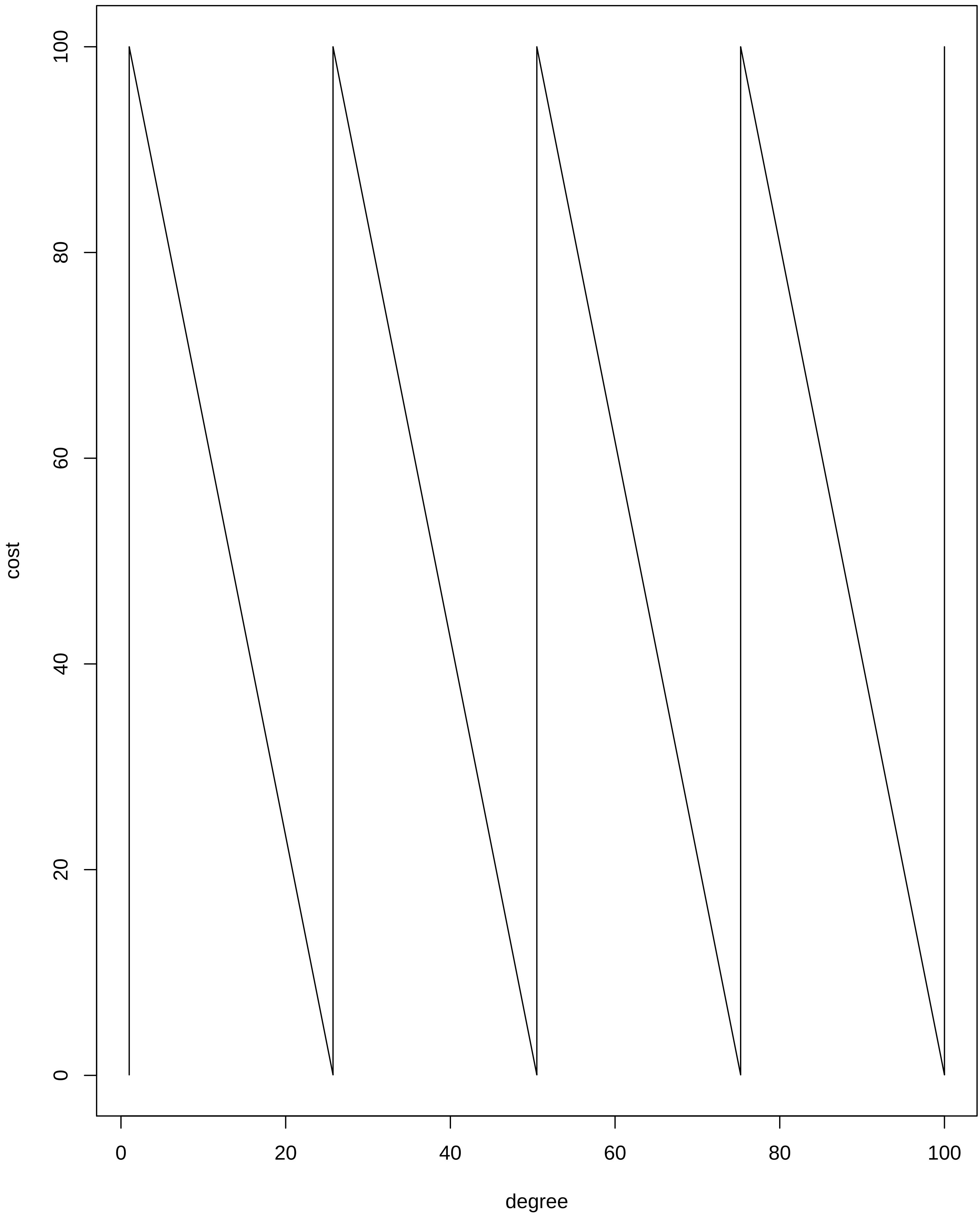
```
# Probelm 9.7
library(ISLR)
# a
Auto$mpg=ifelse(Auto$mpg>median(Auto$mpg),1,0)
table(Auto$mpg)
#   0   1
# 196 196

# b
#install.packages("e1071")
library(e1071)
costs=data.frame(cost=seq(0.05,100,length.out = 15))               # tuning
grid for the cost parameter.
svm.tune=tune(svm,mpg~.,data=Auto,ranges=costs,kernel='linear')     # 10-fold
cross validation.
svm.tune

# Parameter tuning of ‘svm’:
#
#   - sampling method: 10-fold cross validation
#
# - best parameters:
#   cost
# 7.189286
#
# - best performance: 0.09918917
#
# - Detailed performance results:
#          cost      error dispersion
# 1    0.050000 0.10202964 0.03361002
# 2    7.189286 0.09918917 0.03261848
# 3   14.328571 0.10513783 0.03433267
# 4   21.467857 0.10826243 0.03387225
# 5   28.607143 0.11002209 0.03365457
# 6   35.746429 0.11084365 0.03392947
# 7   42.885714 0.11252353 0.03420288
# 8   50.025000 0.11337569 0.03417114
# 9   57.164286 0.11377659 0.03427571
# 10  64.303571 0.11408622 0.03453361
# 11  71.442857 0.11449827 0.03448589
# 12  78.582143 0.11487757 0.03433948
# 13  85.721429 0.11504715 0.03439838
# 14  92.860714 0.11566034 0.03452640
# 15 100.000000 0.11589444 0.03428601

# c
params=data.frame(cost=seq(0.05,100,length.out =
5),degree=seq(1,100,length.out = 5))
svm.poly=tune(svm,mpg~.,data=Auto,ranges=params,kernel='polynomial')
```

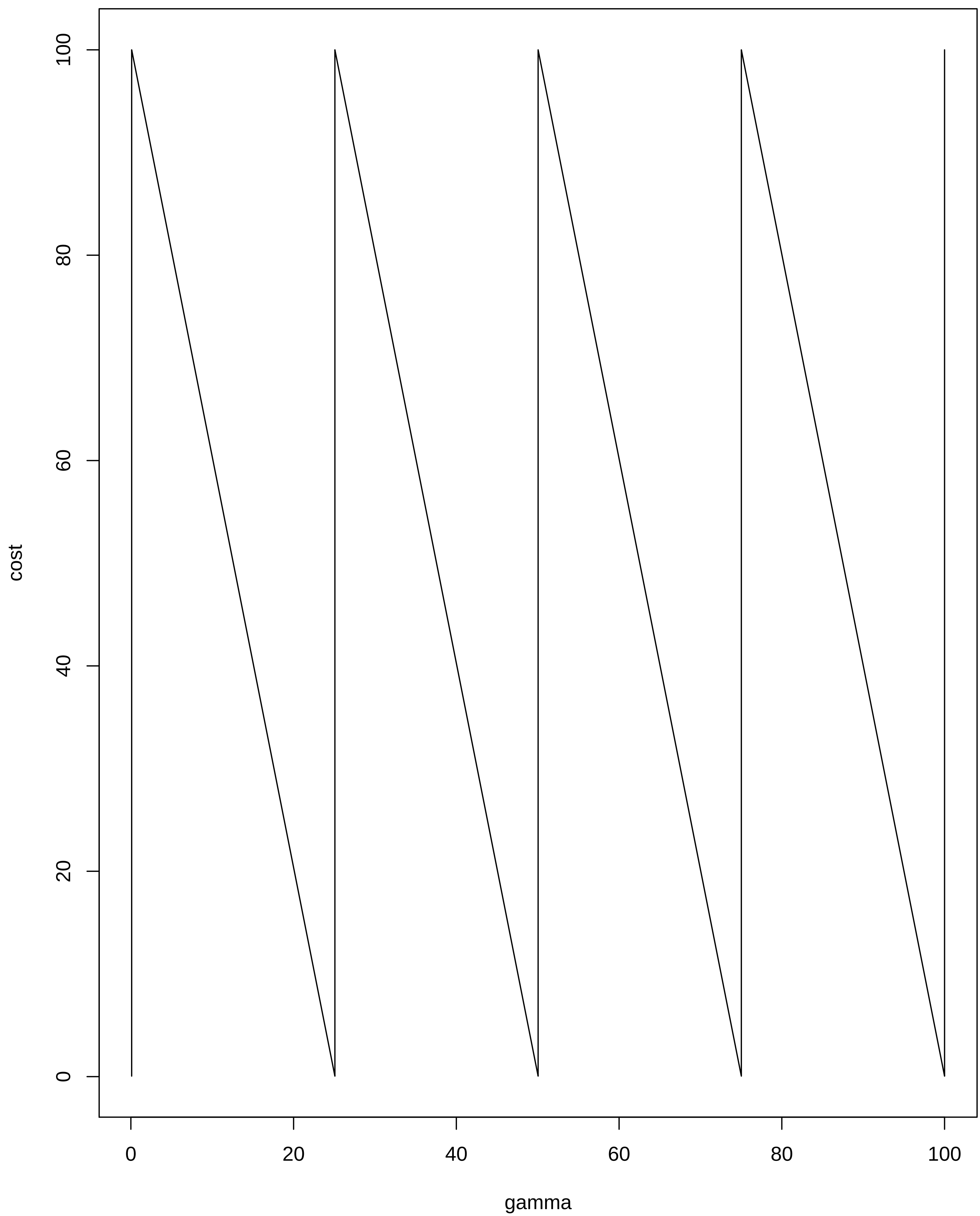
```
summary(svm.poly)
#
# Parameter tuning of ‘svm’:
#
#   - sampling method: 10-fold cross validation
#
# - best parameters:
#   cost degree
# 100      1
#
# - best performance: 0.0974665

# - Detailed performance results:
#       cost degree      error dispersion
# 1    0.0500   1.00 0.31122882 0.04302732
# 2   25.0375   1.00 0.10085624 0.05523499
# 3   50.0250   1.00 0.09941058 0.05527120
# 4   75.0125   1.00 0.09868831 0.05421817
# 5  100.0000   1.00 0.09805010 0.05295848
# 6    0.0500  25.75 0.52345824 0.05422446
# 7   25.0375  25.75 0.52345824 0.05422446
# 8   50.0250  25.75 0.52345824 0.05422446
# 9   75.0125  25.75 0.52345824 0.05422446
# 10 100.0000  25.75 0.52345824 0.05422446
# 11   0.0500  50.50 0.52345824 0.05422446
# 12  25.0375  50.50 0.52345824 0.05422446
# 13  50.0250  50.50 0.52345824 0.05422446
# 14  75.0125  50.50 0.52345824 0.05422446
# 15 100.0000  50.50 0.52345824 0.05422446
# 16   0.0500  75.25 0.52345824 0.05422446
# 17  25.0375  75.25 0.52345824 0.05422446
# 18  50.0250  75.25 0.52345824 0.05422446
# 19  75.0125  75.25 0.52345824 0.05422446
# 20 100.0000  75.25 0.52345824 0.05422446
# 21   0.0500 100.00 0.52345824 0.05422446
# 22  25.0375 100.00 0.52345824 0.05422446
# 23  50.0250 100.00 0.52345824 0.05422446
# 24  75.0125 100.00 0.52345824 0.05422446
# 25 100.0000 100.00 0.52345824 0.05422446

params=data.frame(cost=seq(0.05,100,length.out =
5),gamma=seq(0.1,100,length.out = 5))
svm.radial=tune(svm,mpg~.,data=Auto,ranges=params,kernel='radial')
summary(svm.radial)
# Parameter tuning of ‘svm’:
#
#   - sampling method: 10-fold cross validation
#
# - best parameters:
```

```
#   cost gamma
# 25.0375   0.1
#
# # - best performance: 0.07497467
#     cost   gamma      error  dispersion
# 1    0.0500   0.100 0.07553539 0.020885918
# 2   25.0375   0.100 0.07070107 0.009828754
# 3   50.0250   0.100 0.07682088 0.009985928
# 4   75.0125   0.100 0.08041295 0.012046698
# 5  100.0000   0.100 0.08180043 0.013067447
# 6    0.0500  25.075 0.48294554 0.050646841
# 7   25.0375  25.075 0.24964179 0.002840959
# 8   50.0250  25.075 0.24964179 0.002840959
# 9   75.0125  25.075 0.24964179 0.002840959
# 10 100.0000  25.075 0.24964179 0.002840959
# 11   0.0500  50.050 0.48311419 0.050693482
# 12  25.0375  50.050 0.25124322 0.002629829
# 13  50.0250  50.050 0.25124322 0.002629829
# 14  75.0125  50.050 0.25124322 0.002629829
# 15 100.0000  50.050 0.25124322 0.002629829
# 16   0.0500  75.025 0.48313952 0.050694461
# 17  25.0375  75.025 0.25142340 0.002641727
# 18  50.0250  75.025 0.25142340 0.002641727
# 19  75.0125  75.025 0.25142340 0.002641727
# 20 100.0000  75.025 0.25142340 0.002641727
# 21   0.0500 100.000 0.48314265 0.050694752
# 22  25.0375 100.000 0.25144617 0.002644836
# 23  50.0250 100.000 0.25144617 0.002644836
# 24  75.0125 100.000 0.25144617 0.002644836
# 25 100.0000 100.000 0.25144617 0.002644836

# d
plot(svm.tune$performance[,c(1,2)],type='l')
# From the plot, we can see that while the cost of about 5 achieves the most
reduction in the miss classification error.

plot(svm.poly$performance[,c(2,1)],type='l')
# From the plot and from the summary, the best performacne is achived by
lowest degree of 1 and cost of 100.

plot(svm.radial$performance[,c(2,1)],type='l')
# From the plot and the summary, the best performance is achived by a gamma of
0.1.
```


0.115
0.110
0.105
0.100
error
0
20
40
60
80
100
cost



cost
0
20
40
60
80
100
degree
0
20
40
60
80
100

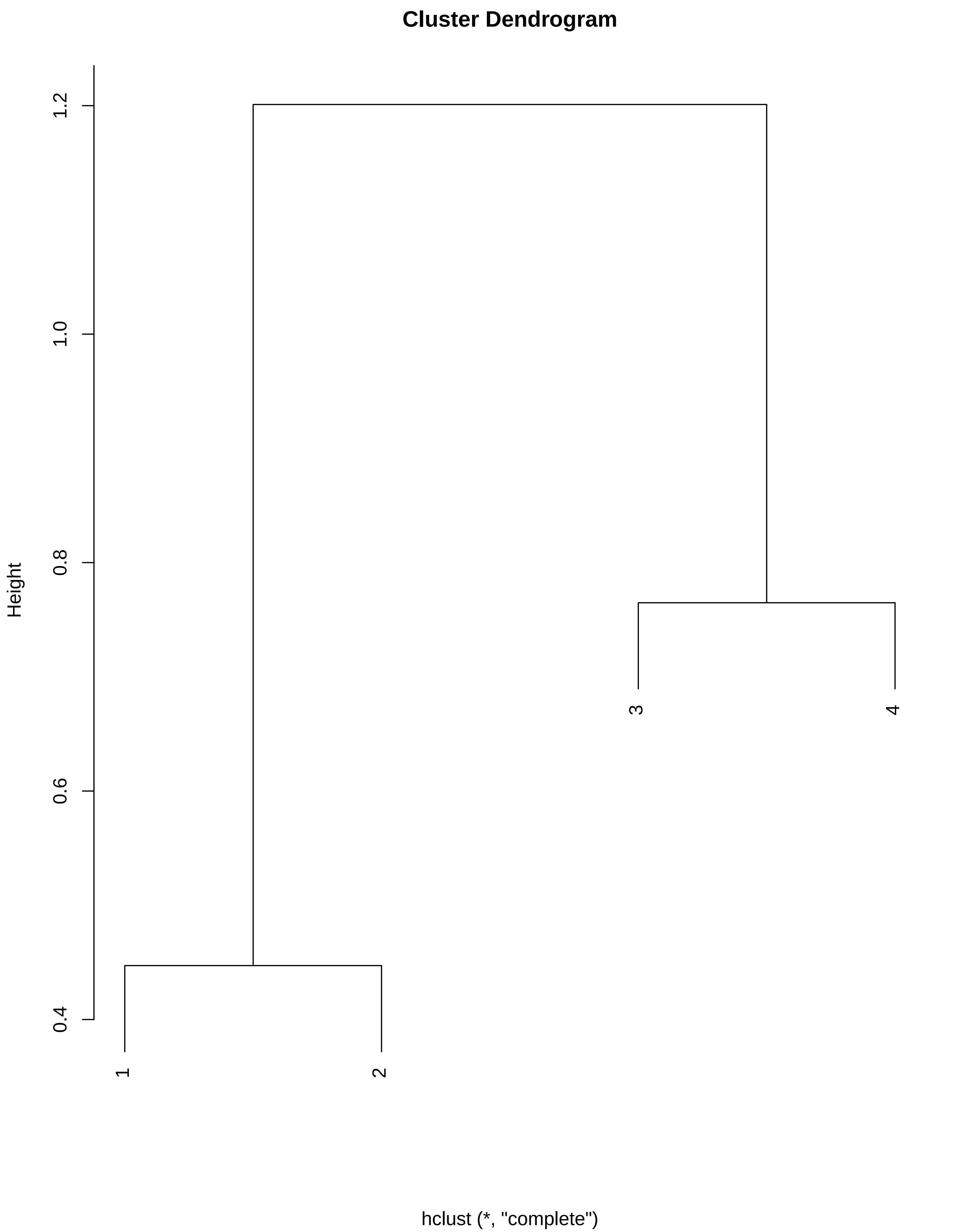


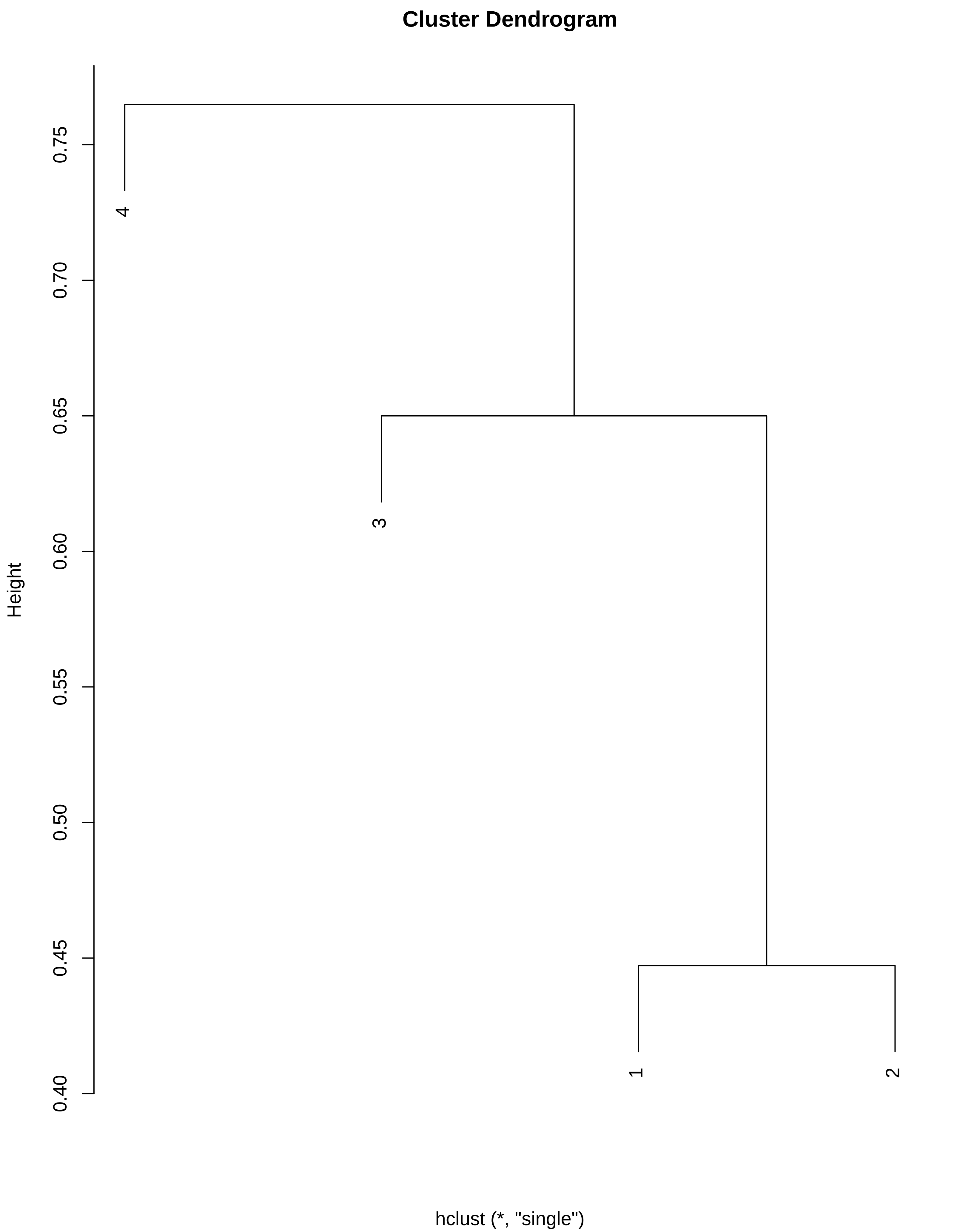
cost
0
20
40
60
80
100
gamma
0
20
40
60
80
100

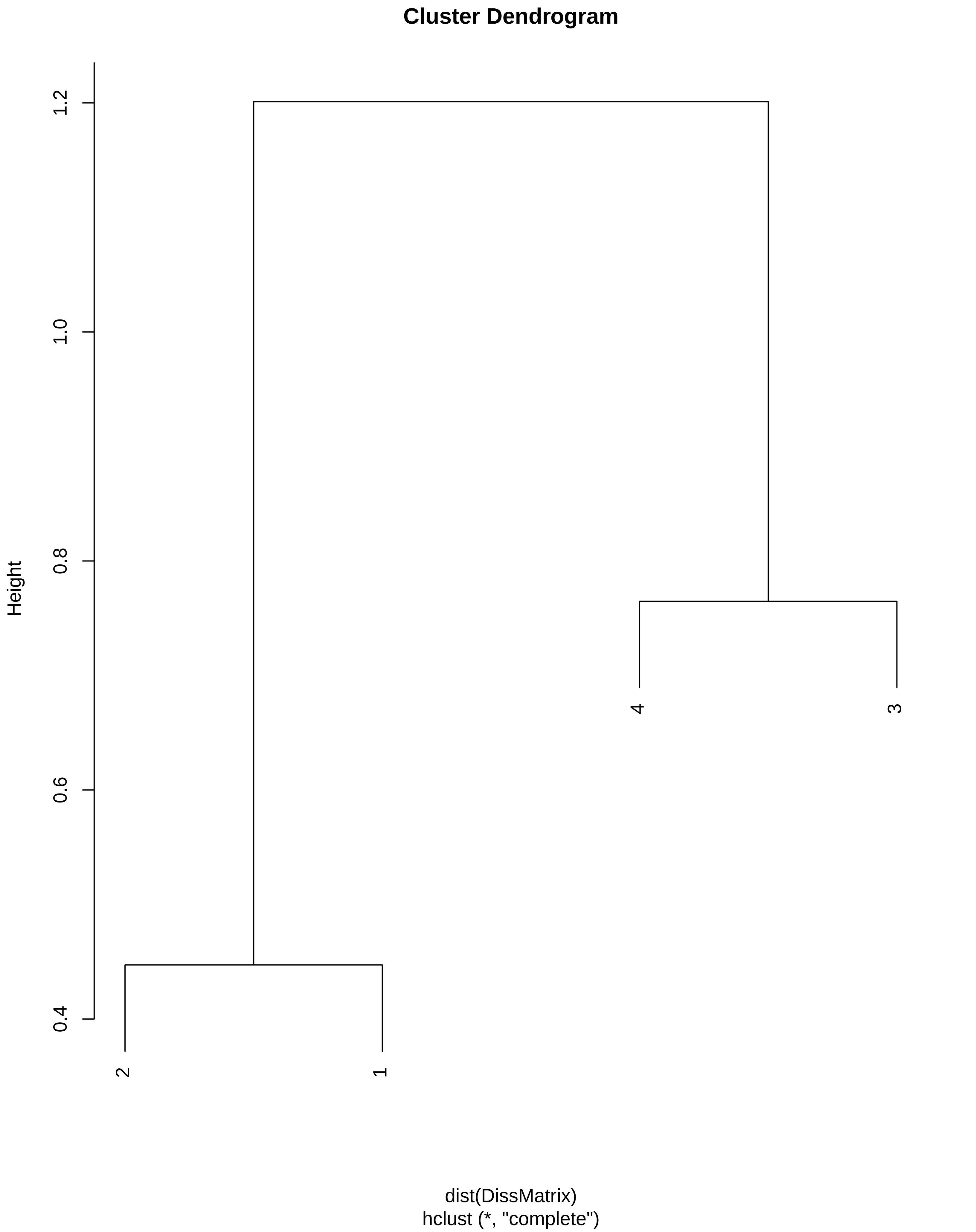
```
#Problem 10.2
#install.packages("knitr")
library(knitr)
# a
DissMatrix=data.frame(c(0,0.3,0.4,0.7),c(0.3,0,0.5,0.8),c(0.4,0.5,0,0.45),c(0.7,0.8,0.45,0
colnames(DissMatrix)=c(paste('Col',1:4))
kable(DissMatrix)
  # | Col 1| Col 2| Col 3| Col 4|
  # |-----:|-----:|-----:|-----:|
  # |   0.0|   0.3|  0.40|  0.70|
  # |   0.3|   0.0|  0.50|  0.80|
  # |   0.4|   0.5|  0.00|  0.45|
  # |   0.7|   0.8|  0.45|  0.00|

plot(hclust(dist(DissMatrix)),xlab='')
#The table gives the dissimilarity between classes, meaning that in order to
produce clusters we need only select pairs whose values are small. The
smallest value in the table is for the entries (1,2) and (2,1), followed by
the pair (3,4),(4,3).


# b
plot(hclust(dist(DissMatrix),method='single'),xlab='')

# c
#If we cut the first dendogram obtain at the value of 0.7, then we obtain the
clusters (1,2) and (3,4). Cutting below this value produces 3 clusters since
the observations 3 and 4 are in their own cluster.

# d
#The clusters obtained here are (4) and (1,2,3).

# e
row.names(DissMatrix)=c(2,1,4,3)
plot(hclust(dist(DissMatrix)))
#A dendogram is read bottom up, where the height indicates where clusters are
fused. Thus there is no horizontal meaning, the leafs are be swapped but they
still represent clusters that are fused at the same height.



# problem 10.7
USArrests.scaled=scale(USArrests)
correlation=as.dist(1-cor(t(USArrests.scaled)))
euclidean=dist(USArrests.scaled)^2

#If the quantities are approximately proportional then euclidean ≈
K·correlation for a constant K.
summary(correlation/euclidean)
```

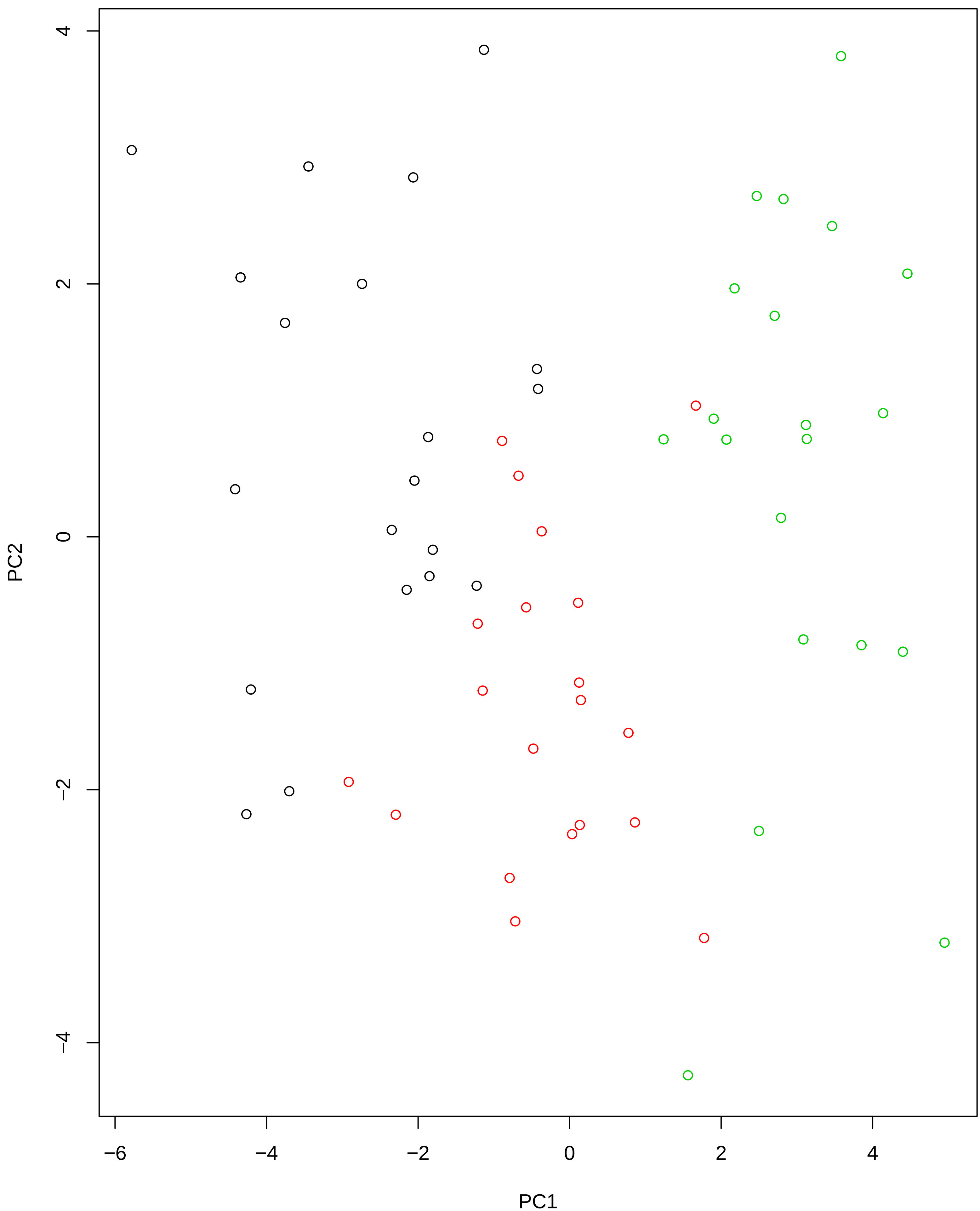
```
#    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#0.000086 0.069135 0.133943 0.234193 0.262589 4.887686

summary(correlation-0.1339*euclidean)
#    Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
# -4.569956 -0.459715  0.000393 -0.059043  0.557616  1.808901

#If K=0.1339 then they are approximately equal, different only 0.05 on
average.



# Problem 10.10
# a
set.seed(42)
data= matrix(sapply(1:3,function(x){ rnorm(20*50,mean = 10*sqrt(x))
}),ncol=50)    # 20 obs. in each class with 50 features.
class=unlist(lapply(1:3,function(x){rep(x,20)}))

# b
pr.out=prcomp(data)
plot(pr.out$x[,c(1,2)],col=class)

# c
set.seed(1)
kmeans.out=kmeans(data,3)
table(kmeans.out$cluster)
#   1  2  3
#   20 19 21
table(class)
# class
# 1  2  3
# 20 20 20

plot(pr.out$x[,c(1,2)],col=kmeans.out$cluster)
# Comparing to the graph from 10b, we can see that there is only one
observation that is miss classified.

# d
set.seed(1)
kmeans.out=kmeans(data,2)
table(kmeans.out$cluster)
#1  2
#24 36

table(class)
# class
# 1  2  3
# 20 20 20
```

```
plot(pr.out$x[,c(1,2)],col=kmeans.out$cluster)
#K-means seem to find a single cluster that is the same as before.
# We can see that k-mean seperated the green middle section from the plot
10.d2, and put all the point the right of about 0.5 (PC1) to one cluster, and
left points to one cluster.




# e
set.seed(1)
kmeans.out=kmeans(data,4)
table(kmeans.out$cluster)
# 1  2  3  4
#19 10 17 14

plot(pr.out$x[,c(1,2)],col=kmeans.out$cluster)

#However, by examining the plot we can see that it again find the original
green cluster with some overlap between it and the remaining ones. Overlap
between clusters in the two principal components is also clear,
#as should be expected since they may be close in the remaining dimensions.


# f
set.seed(1)
kmeans.out=kmeans(pr.out$x[,c(1,2)],3)
table(kmeans.out$cluster)
# 1  2  3
#20  8 32
plot(pr.out$x[,c(1,2)],col=kmeans.out$cluster)
# This clustering seems to seperated the plot where a clear path can be seen.


# g
set.seed(1)
kmeans.out=kmeans(scale(data,center = T,scale = T),3)
table(kmeans.out$cluster)
#1  2  3
#32 14 14
plot(pr.out$x[,c(1,2)],col=kmeans.out$cluster)

#There is significant overlap in the first two clusters, and the algorithm
performs poorly compare to b.
```

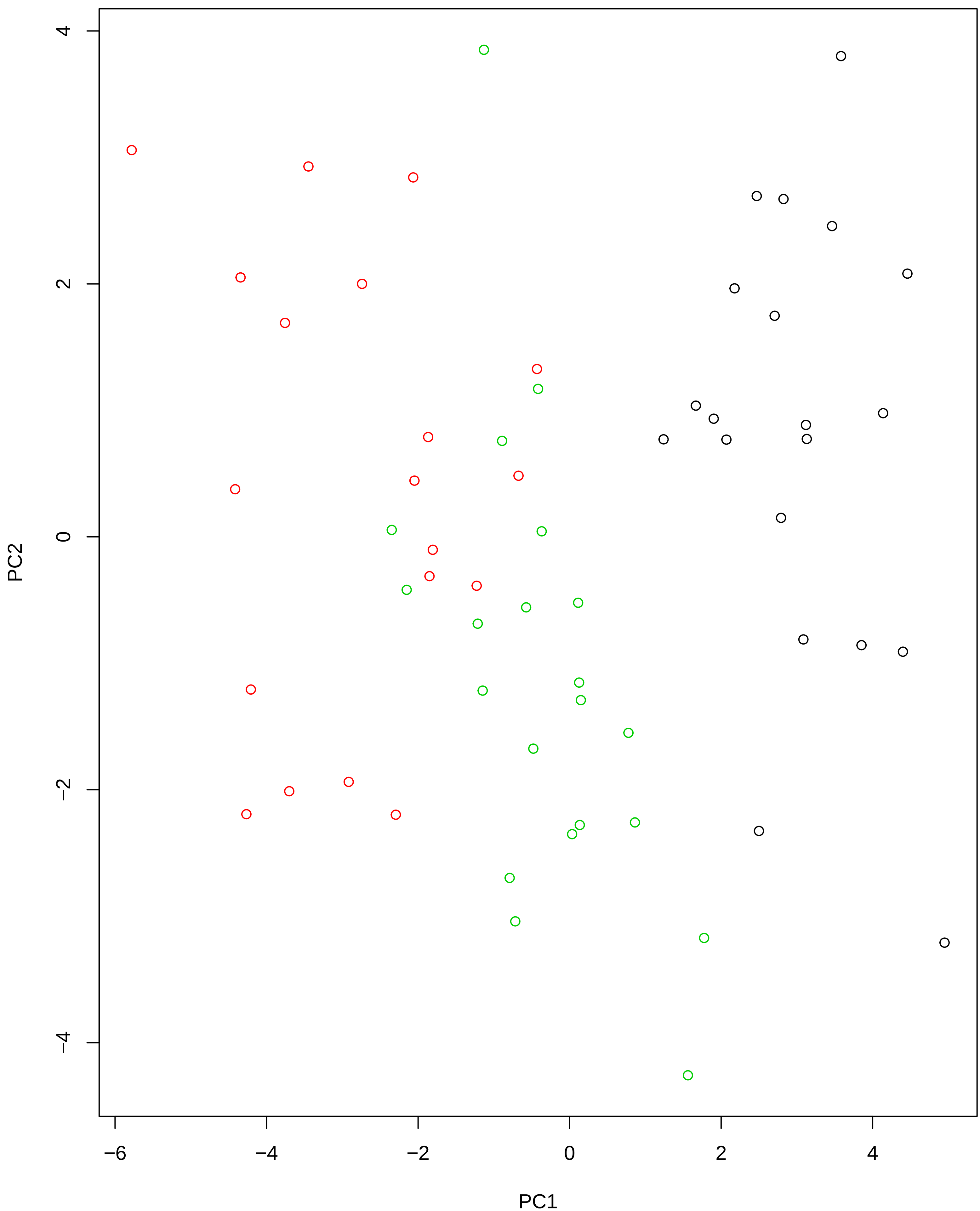

Cluster Dendrogram
Height
1.2
1.0
0.8
0.6
0.4
1
2
3
4
hclust (*, "complete")


Cluster Dendrogram
Height
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.75
4
3
1
2
hclust (*, "single")

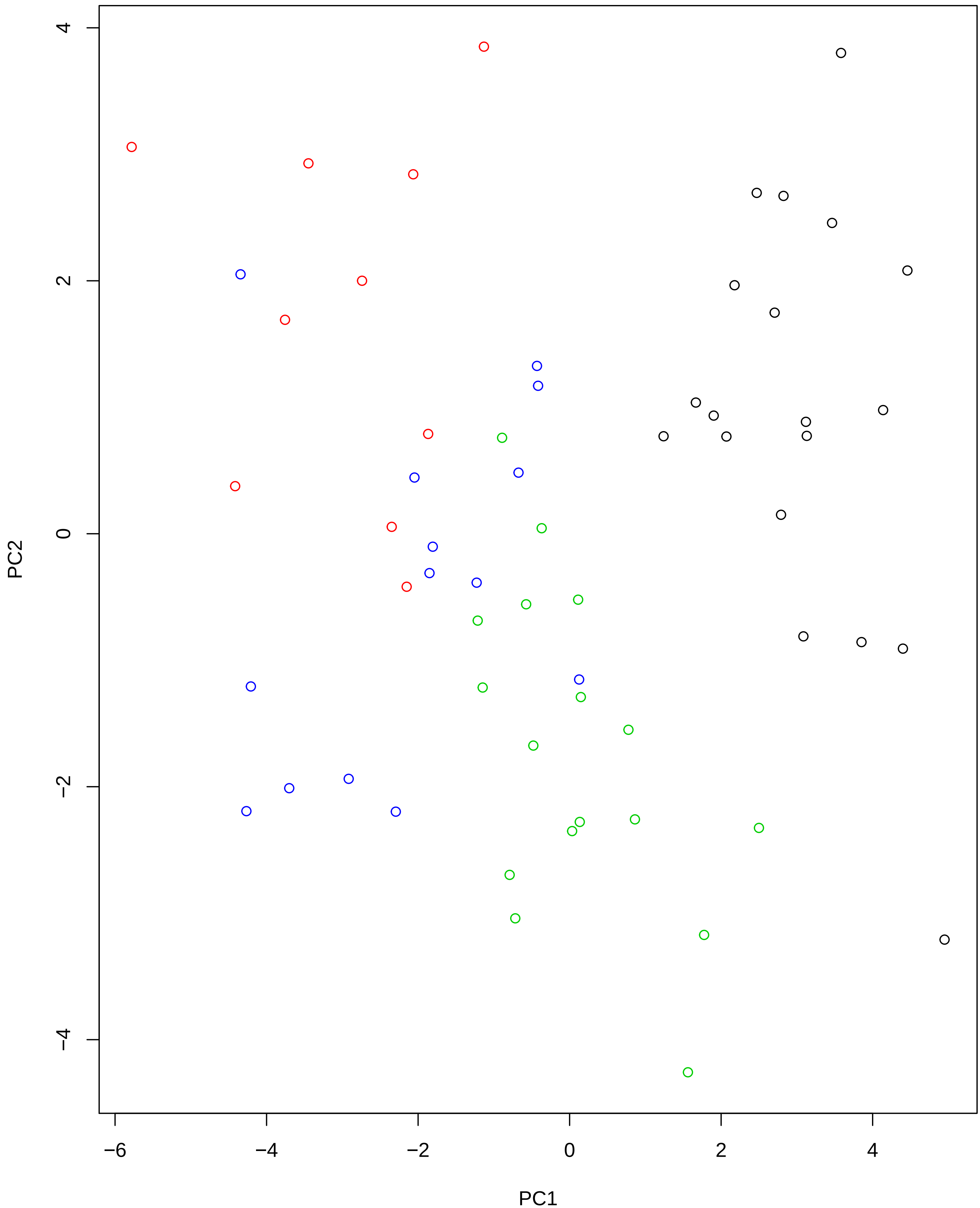

Cluster Dendrogram
Height
1.2
1.0
0.8
0.6
0.4
2
1
4
3
dist(DissMatrix)
hclust (*, "complete")



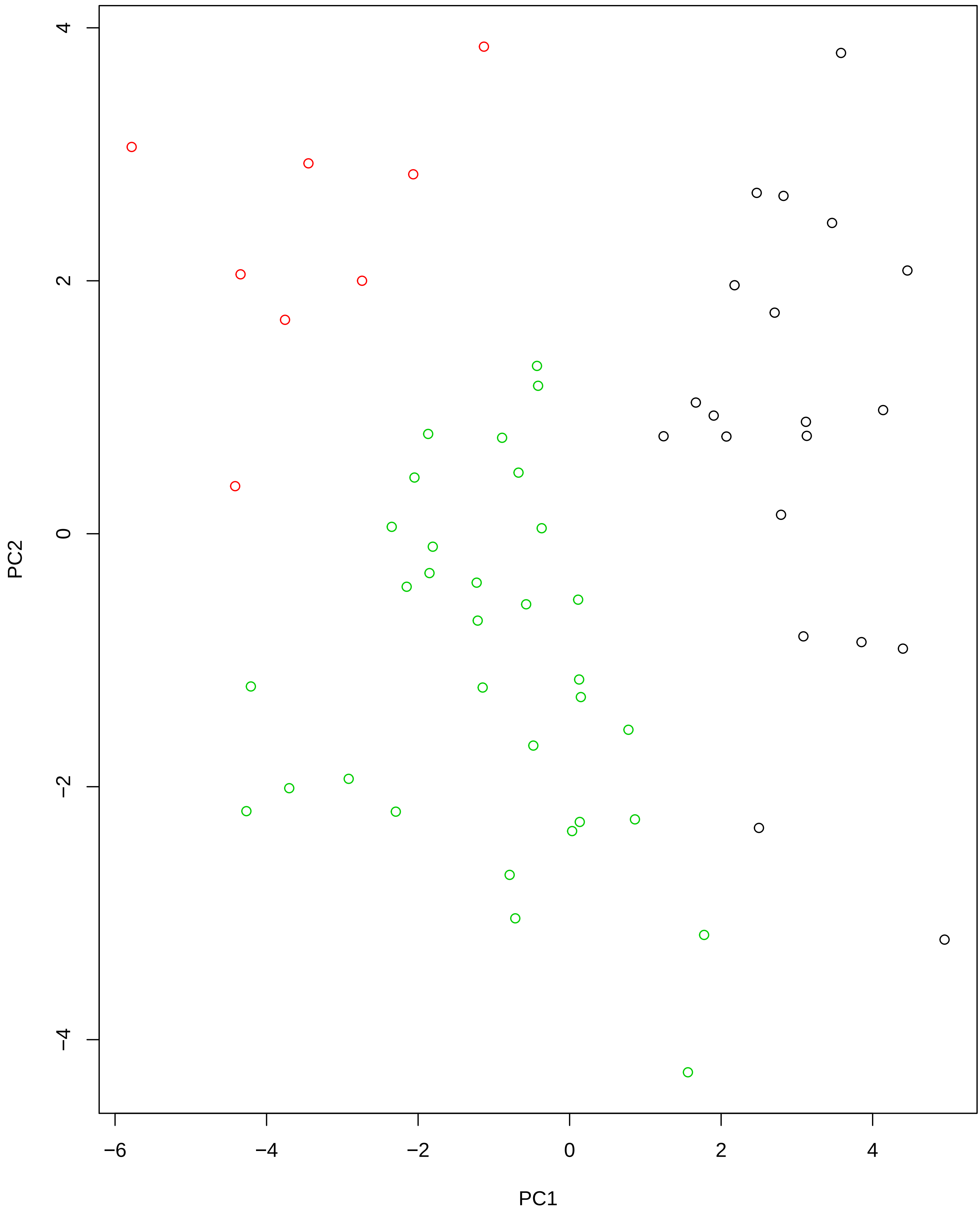
PC1
PC2
−6
−4
−2
0
2
4



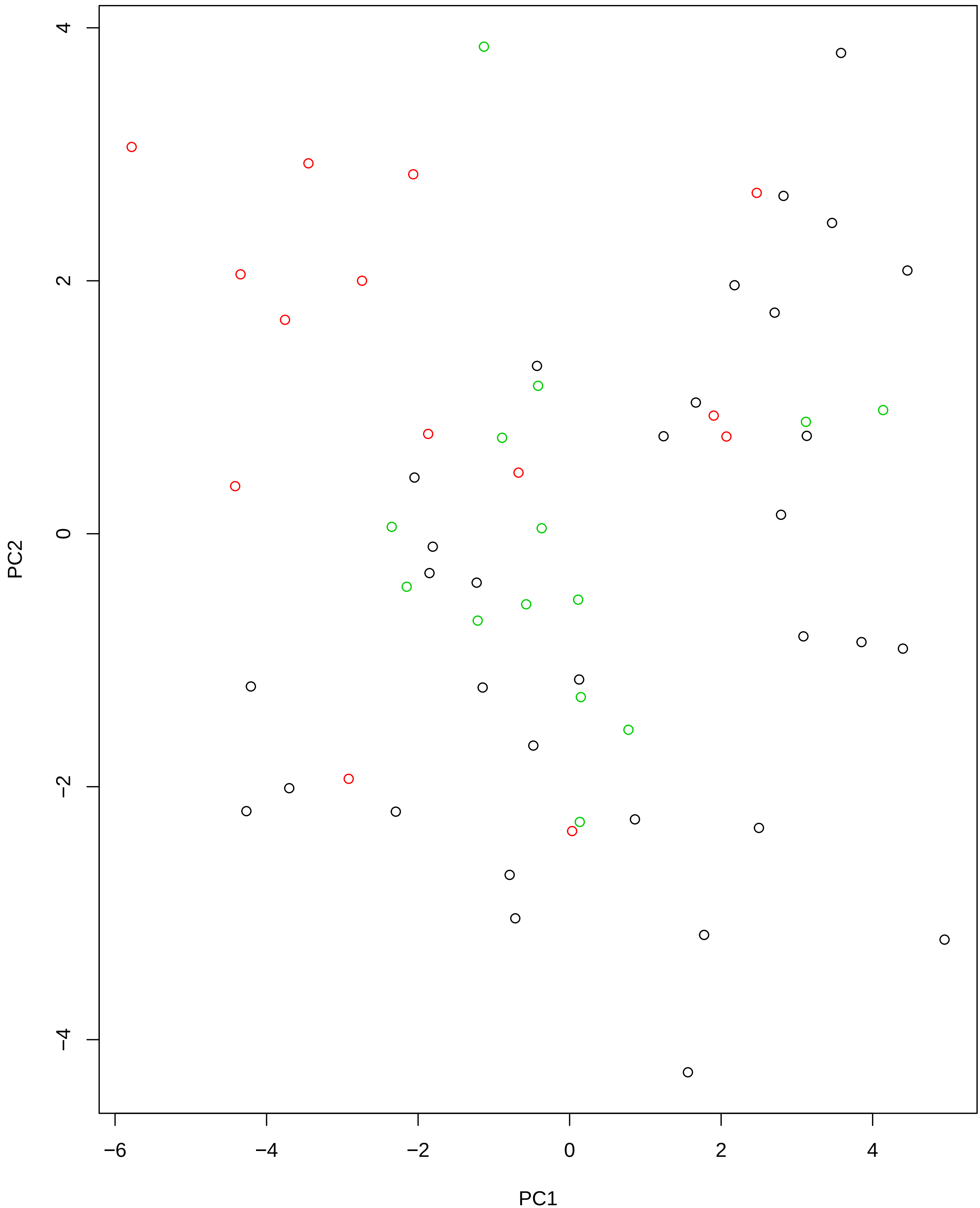
4
2
0
−2
−4
PC2
−6
−4
−2
0
2
4
PC1



PC2
4
2
0
−2
−4
−6
−4
−2
0
2
4
PC1



PC1
PC2
−6
−4
−2
0
2
4



4
2
0
−2
−4
PC2
−6
−4
−2
0
2
4
PC1