Blatt1

Tobias

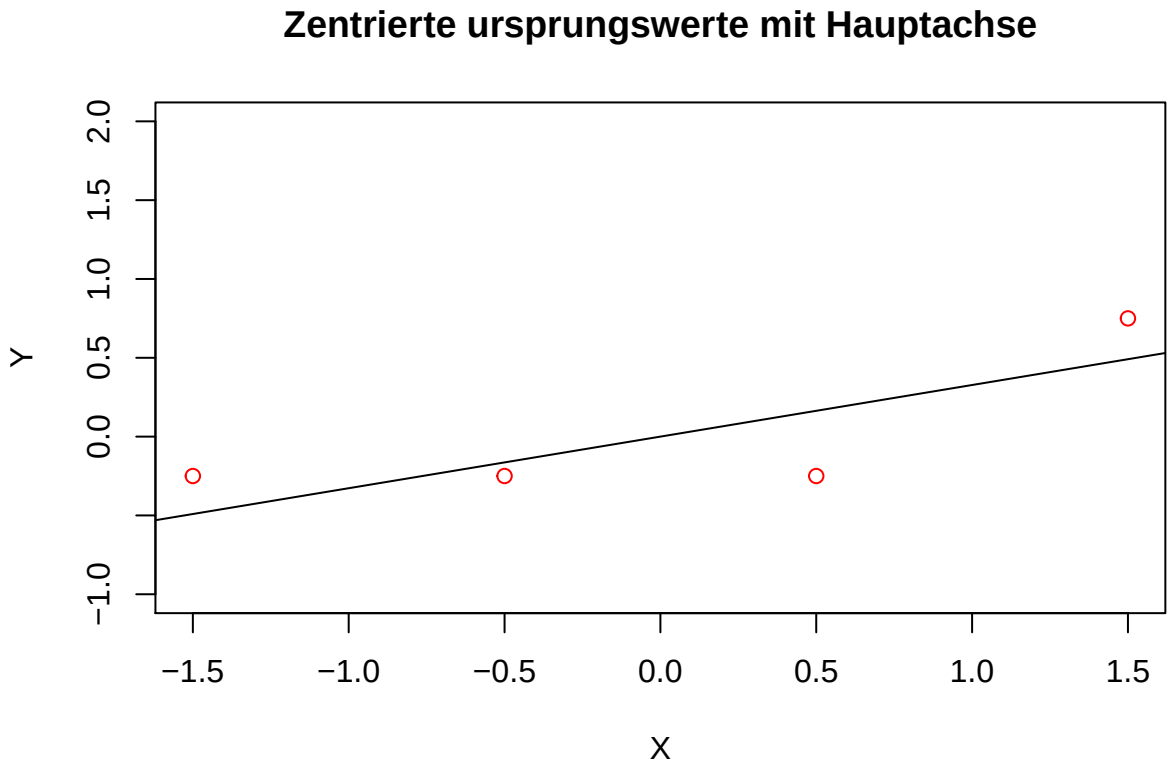
17 5 2021

```
#library('jpeg')
#Bild = jpeg::readJPEG(source = "Aufgabe1cut.JPG",native = TRUE)
##knitr::include_graphics(Bild)  #Funktioniert nicht
#plot(0:1,0:1,type = 'n',ann=FALSE,axes = FALSE)
#rasterImage(Bild,0,0,1,1)
```

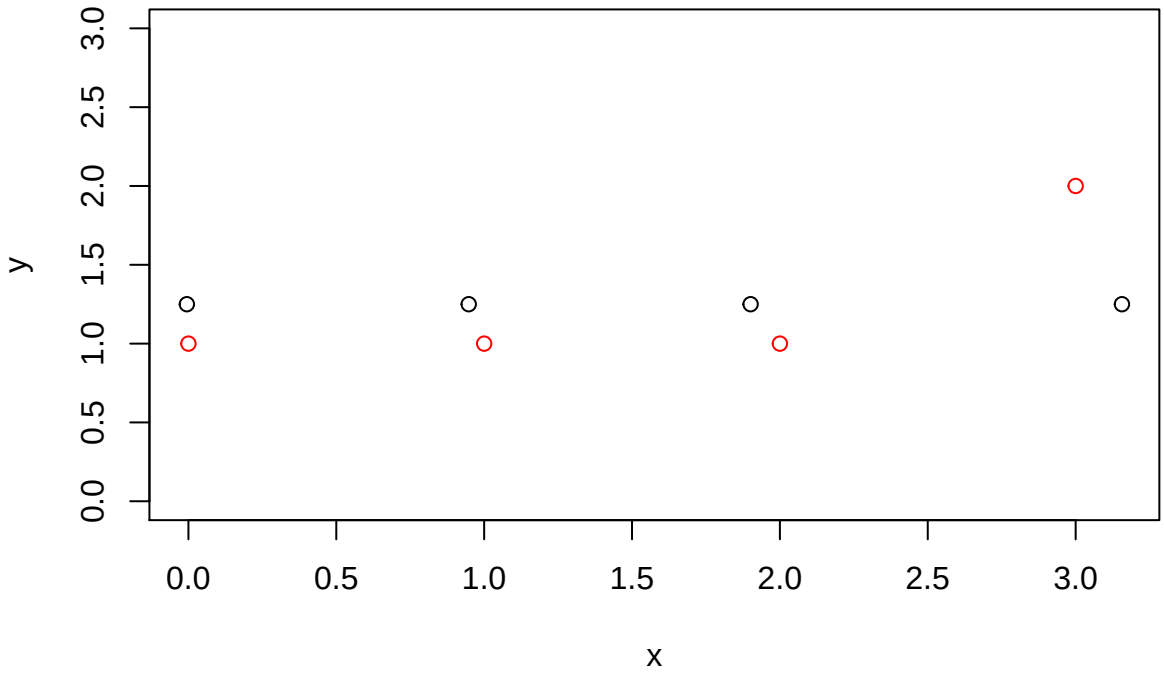
Aufgabe 1.1

```
x = c(0,1,2,3)
y = c(1,1,1,2)
A = rbind(x,y)  #gegebene Matrix A


X = x-mean(x)
Y = y-mean(y)
A_ = rbind(X,Y)   #Mittelpunkt der Matrix A in den Ursprung verschoben


u = eigen(A_ %*% t(A_))
v = eigen(t(A_)%*%A_)
EV = u$vec  #Eigenvektoren der Matrix
a_i = EV[,1] %*% A_   #Eigenvektor vom größten Eigenwert mal der verschobenen Matrix
```

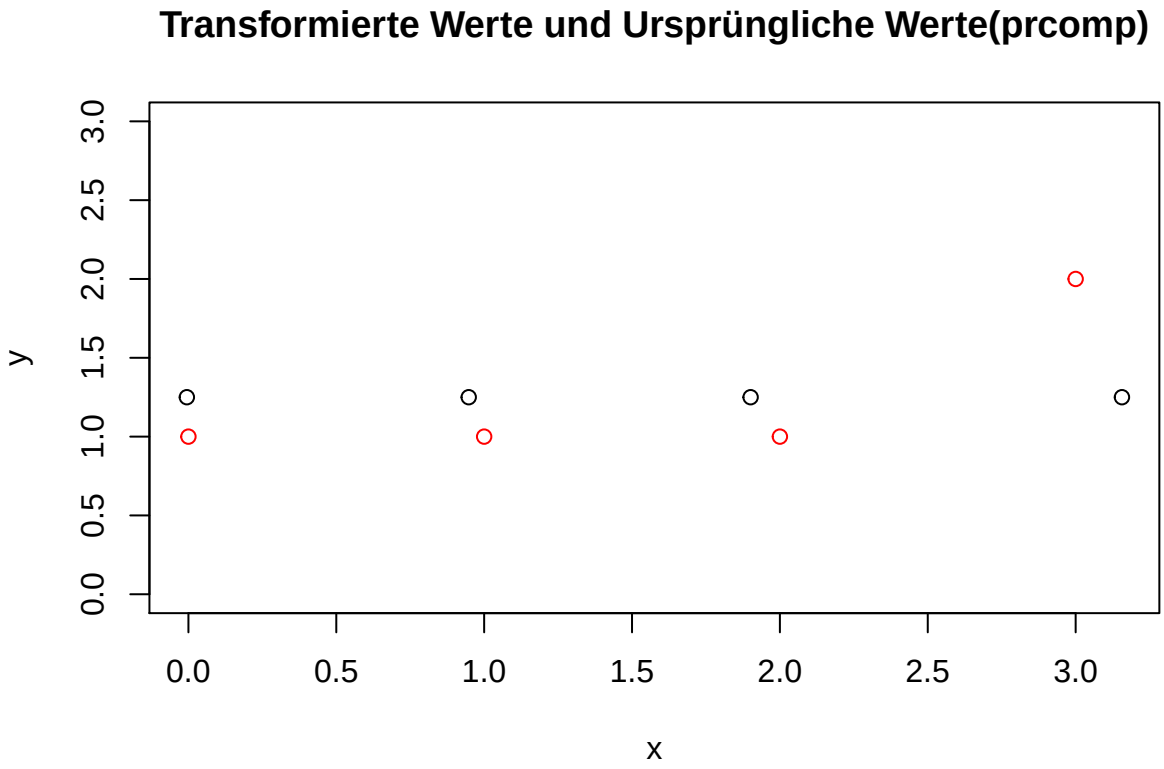
```
plot(X,Y,main = 'Zentrierte ursprungswerte mit Hauptachse',col = 2,ylim = c(-1,2))
lines(c(-2,2),(0.312/0.953)*c(-2,2)) # Achses des größten Eigenvektors
```




```
plot(-1*a_i+mean(x),c(mean(y),mean(y),mean(y),mean(y)),xlab = 'x',ylab = 'y',
     main = 'Transformierte Werte und Ursprüngliche Werte', ylim = c(0,3))
points(x,y,col = 2)
```




```
pr = prcomp(t(A_))  #principal components analysis (prcomp()$rotation -> Eigenvektoren)
plot(pr$rotation[,1]%*%A_+mean(x),c(mean(y),mean(y),mean(y),mean(y)),xlab = 'x',ylab = 'y'
     ,main = 'Transformierte Werte und Ursprüngliche Werte(prcomp)', ylim = c(0,3))
points(x,y,col = 2)
```



Die selbstberechneten Werte stimmen mit den Werten aus der prcomp-Funktion überein. Es muss lediglich auf die Vorzeichen der Eigenvektoren geachtet werden.

Kontrolle -> nicht perfekt

```
x = c(1.2,1.9,2.5,4.4)
y = c(1.6,3.9,5.8,7.9)
A = rbind(x,y)  #gegebene Matrix A


X = x-mean(x)
Y = y-mean(y)
A_ = rbind(X,Y)


u = eigen(A_ %*% t(A_)) # $values(Eigenwerte) $vector(Eigenvektoren)
v = eigen(t(A_)%*%A_)
a_i = u$vec[,1] %*% A_

print('Werte')
```

```
## [1] "Werte"
```

```
v$vec
```

```
##            [,1]       [,2]        [,3]        [,4]
## [1,]  0.6636342  0.4212372  0.61818188  0.00000000
## [2,]  0.2067743 -0.2183956 -0.07315991  0.95089188
## [3,] -0.1724986 -0.7155422  0.67276150 -0.07507041
```

```
## [4,] -0.6979099  0.5127006  0.39986342  0.30028165
```

```
sqrt(rbind(c(u$values[1],0),c(0,u$values[2]),c(0,0),c(0,0)))
```

```
##          [,1]      [,2]
## [1,] 5.189613 0.0000000
## [2,] 0.000000 0.6228325
## [3,] 0.000000 0.0000000
## [4,] 0.000000 0.0000000
```

```
u$vec
```

```
##           [,1]       [,2]
## [1,] 0.4456629 -0.8952009
## [2,] 0.8952009  0.4456629
```

```
print('Probe')
```

```
## [1] "Probe"
```

```
v$vec %*% sqrt(rbind(c(u$values[1],0),c(0,u$values[2]),c(0,0),c(0,0))) %*% u$vec
```

```
##            [,1]       [,2]
## [1,]  1.7697302 -2.9661516
## [2,]  0.3564626 -1.0212416
## [3,] -0.7979156  0.6027691
## [4,] -1.3282771  3.3846240
```

```
print('svdPC')
```

```
## [1] "svdPC"
```

```
print('Werte')
```

```
## [1] "Werte"
```

```
(s <- svd(A_))
```

```
## $d
## [1] 5.1896127 0.6228325
##
## $u
##            [,1]       [,2]
## [1,] -0.4456629 -0.8952009
## [2,] -0.8952009  0.4456629
##
## $v
##            [,1]       [,2]
## [1,]  0.6636342 -0.4212372
## [2,]  0.2067743  0.2183956
## [3,] -0.1724986  0.7155422
## [4,] -0.6979099 -0.5127006
```

```
D = rbind(c(s$d[1],0),c(0,s$d[2])) # Als Matrix
print('Probe')
```

```
## [1] "Probe"
```

```
s$u %*% D %*% t(s$v)    # Funktioniert
```

```
##      [,1] [,2]          [,3] [,4]
```

```
## [1,] -1.3 -0.6 -5.551115e-17  1.9
## [2,] -3.2 -0.9  1.000000e+00  3.1
```

```
#d = t(s$u[,1]) %*% A_    # Größter Eigenvektor
#plot(d[1,]+mean(x),c(mean(y),mean(y),mean(y),mean(y)))
```

Aufgabe 1.2

a) Es werden zufällig k Punkte in die Messung eigefügt.

Die Abstände zwischen den Punkten und allen Messwerten wird ermittelt. Die Messwerten werden den Punkten zugeordnet. Berechnung des Mittelpunktes der zugeordneten Messwerte. Verschiebung der k Punkte in die jeweiligen Mittelwerte. Beginne von vorne bis Abbruchsbedingung erfüllt ist.

```
library(ggplot2)
library(datasets)
data(iris)
```

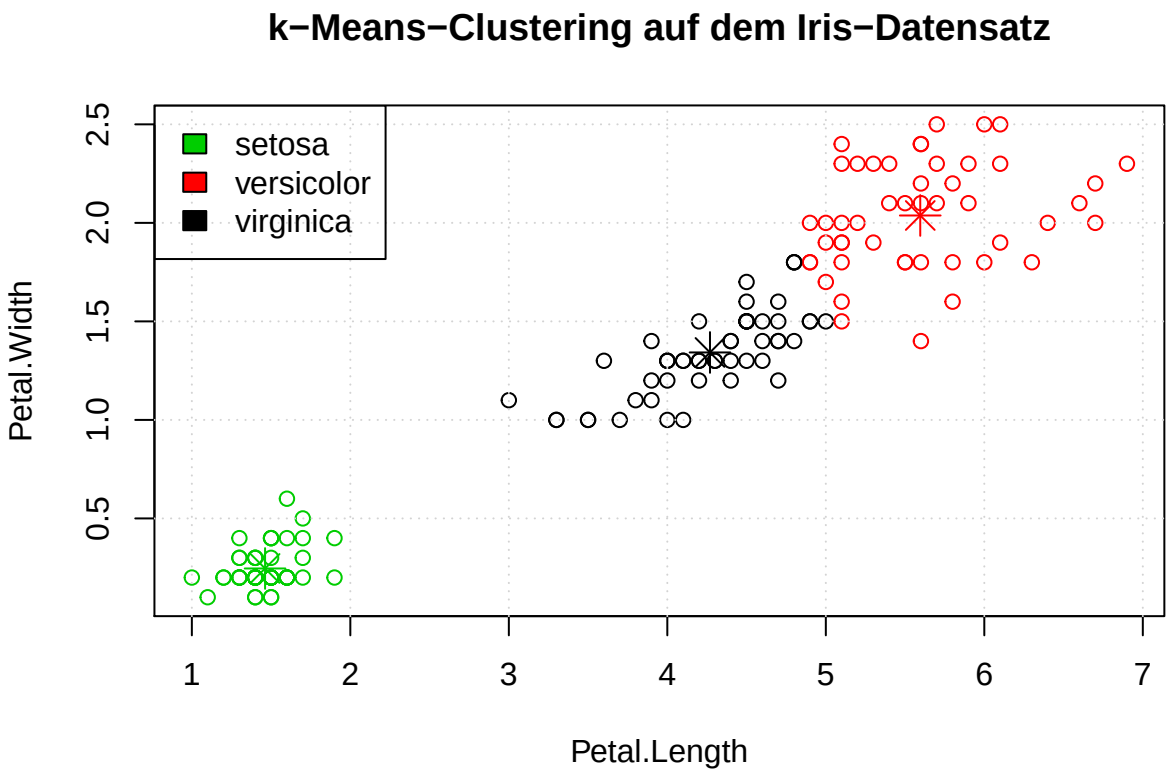
b)

c) Da es sich um 3 Irisklassen handelt solten auch bei k-means 3 Klassen gewählt werden. Da die Lage der eigefügten k Punkte zufällig ist, kann es zu unterschiedlichen Ergebnissen zwischen mehreren durchläufen kommen.

```
data = t(rbind(iris$Petal.Length,iris$Petal.Width))
colnames(data) <- c("Petal.Length", "Petal.Width")

kmeans = kmeans(data,centers = 3,nstart = 20,iter.max = 50)
```

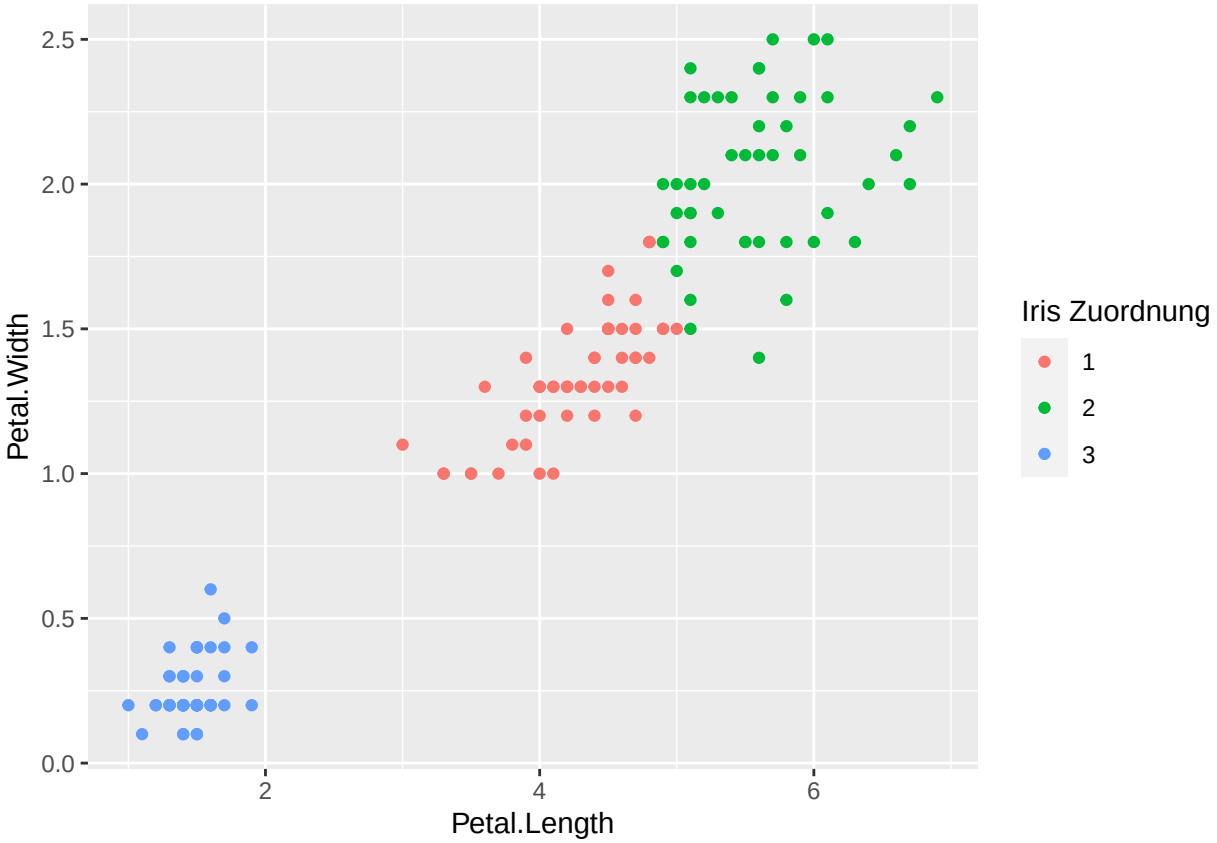
Darstellung mit polt()

```
plot(data,col = kmeans$cluster,main = "k-Means-Clustering auf dem Iris-Datensatz",
     xlab = "Petal.Length", ylab = "Petal.Width")
grid()
legend("topleft", c("setosa","versicolor","virginica"), fill=c("3","2","1"))
points(kmeans$centers,col = 1:3,pch = 8,cex = 2)
```



Darstellung mit ggpolt()

```
df = data.frame(data,grup = (kmeans$cluster))
ggplot(df,aes(Petal.Length,Petal.Width,colour = factor(kmeans$cluster)))+
  geom_point()+
  labs(colour = "Iris Zuordnung")
```



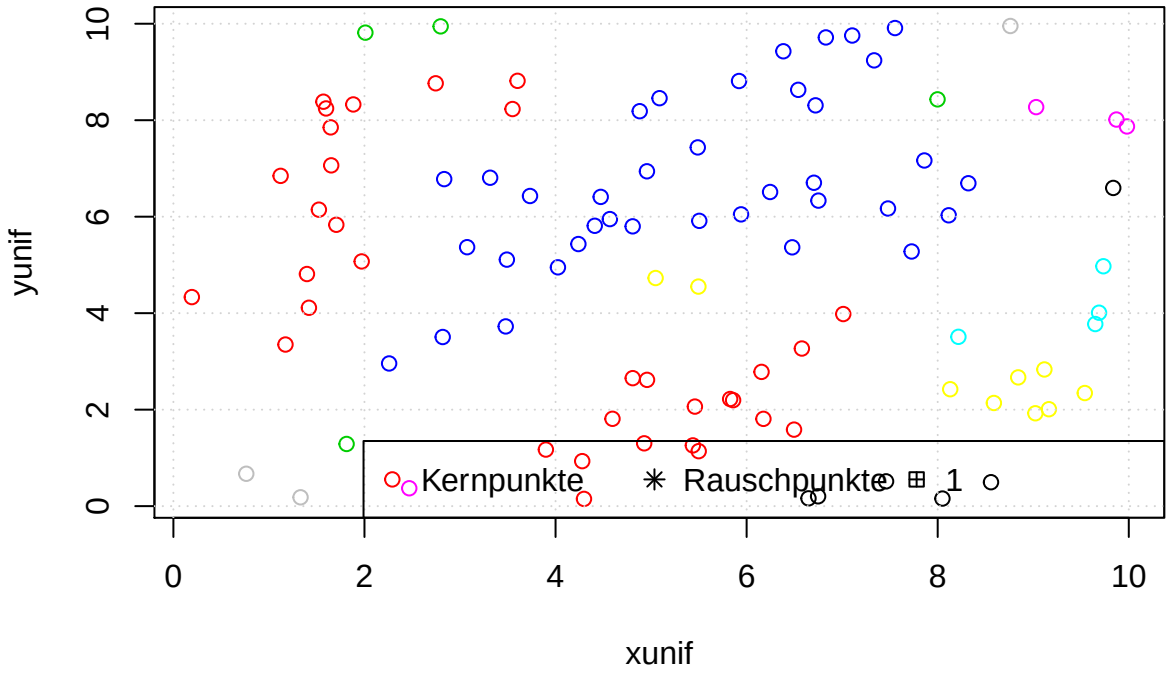
Aufgabe 1.3

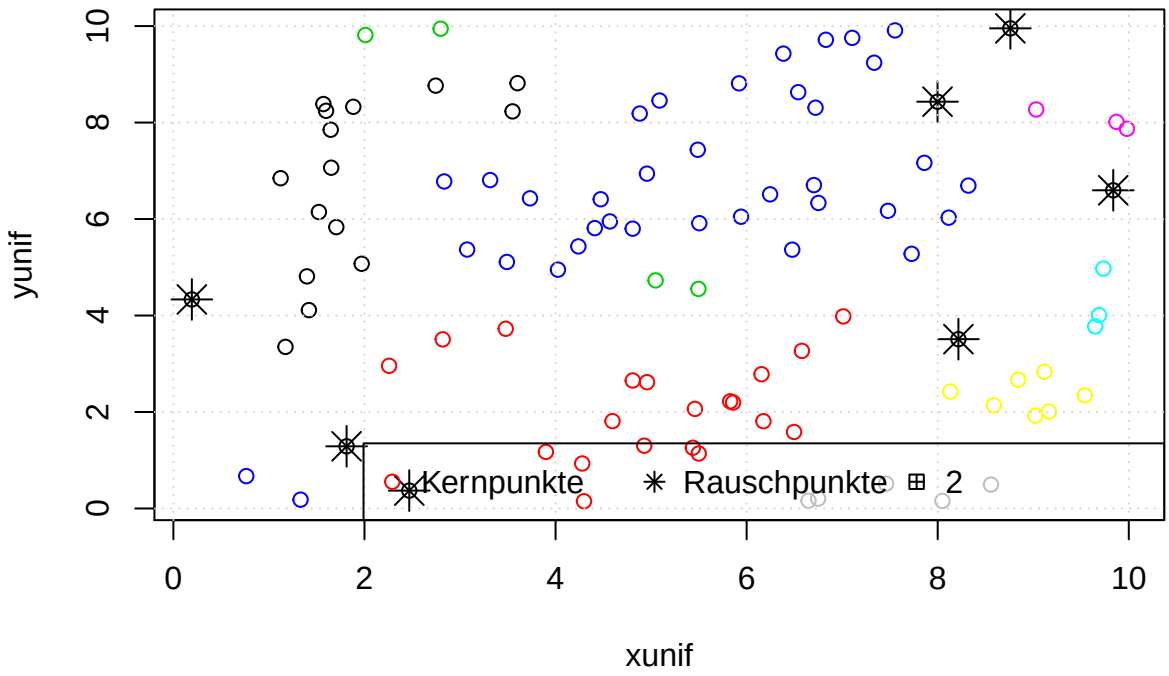
a) Es wird eine Epsilonumgebung und die minimale Anzahl der Punkte angegeben, welche sich in der Epsilonumgebung befinden müssen, damit es sich im einen Kernpunkt handelt. Befinden sich weniger Punkte in der Ubgebung, handelt es sich um einen Randpunkt. Sollten keine Punkte in der Umgebung sein, ist es ein Rauschpunkt.

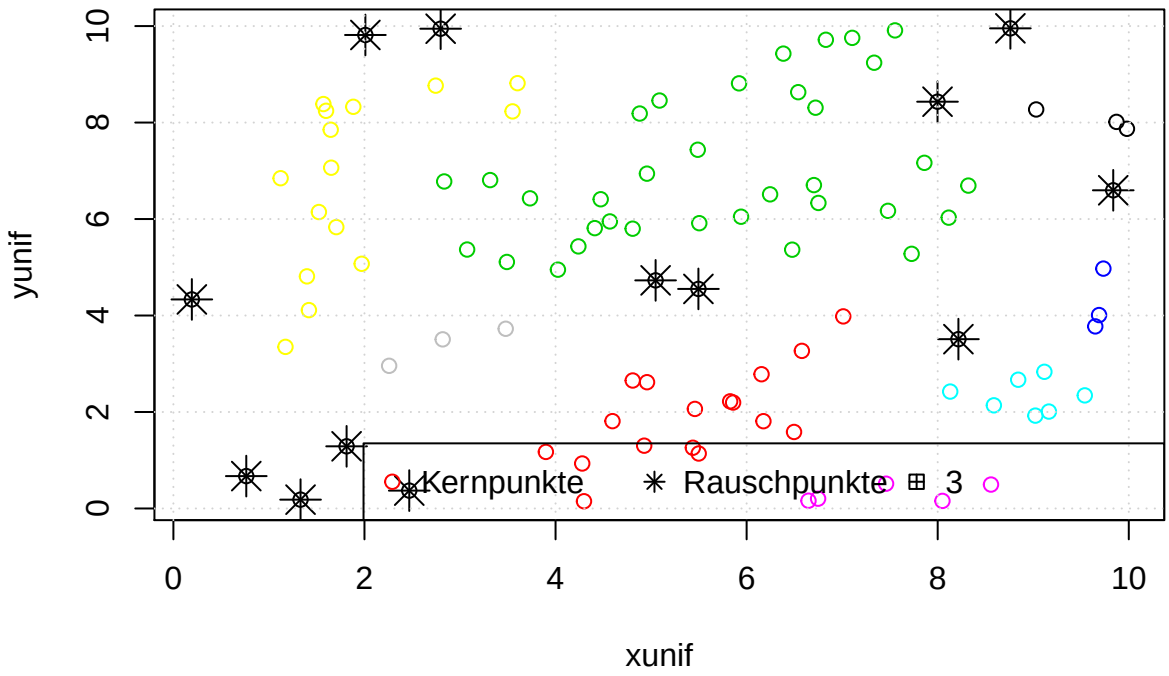
```
library(dbscan)

xunif = runif(100,min = 0,max = 10)
yunif = runif(100,min = 0,max = 10)
unif = data.frame(t(rbind(xunif,yunif)))

xnorm = rnorm(100,mean = 5,sd = 1.5)
ynorm = rnorm(100,mean = 5,sd = 1.5)
norm = data.frame(t(rbind(xnorm,ynorm)))

dbunif = function(dist,minPts){
  dbunif = dbscan(dist,1,minPts = minPts)
   #dist=Verteilung(unif,norm),epsilon=1,minPunkte in Epsilon für Kern-Randpunkt

  rausp = which(dbunif$cluster %in% 0)

  plot(dist,col = dbunif$cluster+1)
  points(dist[rausp,],pch = 8,cex = 2)
```

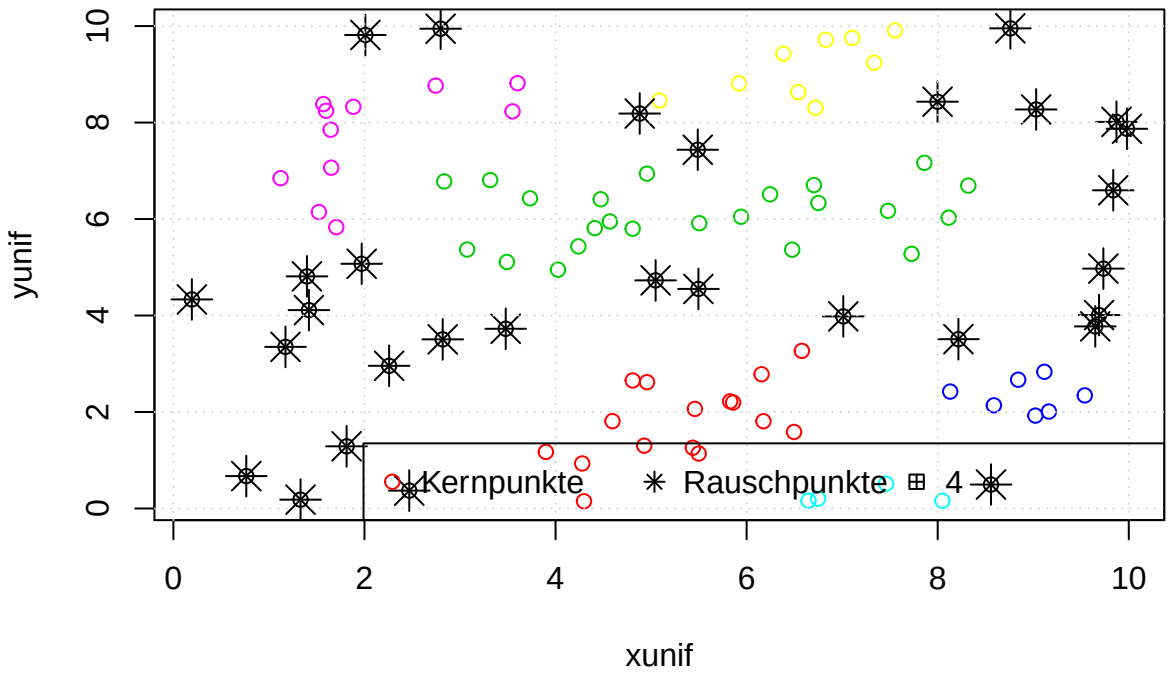
```
  grid()
  legend("bottomright", c("Kernpunkte","Rauschpunkte",minPts),
         pch = c(1,8,12),horiz = TRUE,col = c(2,1,1))
}
```

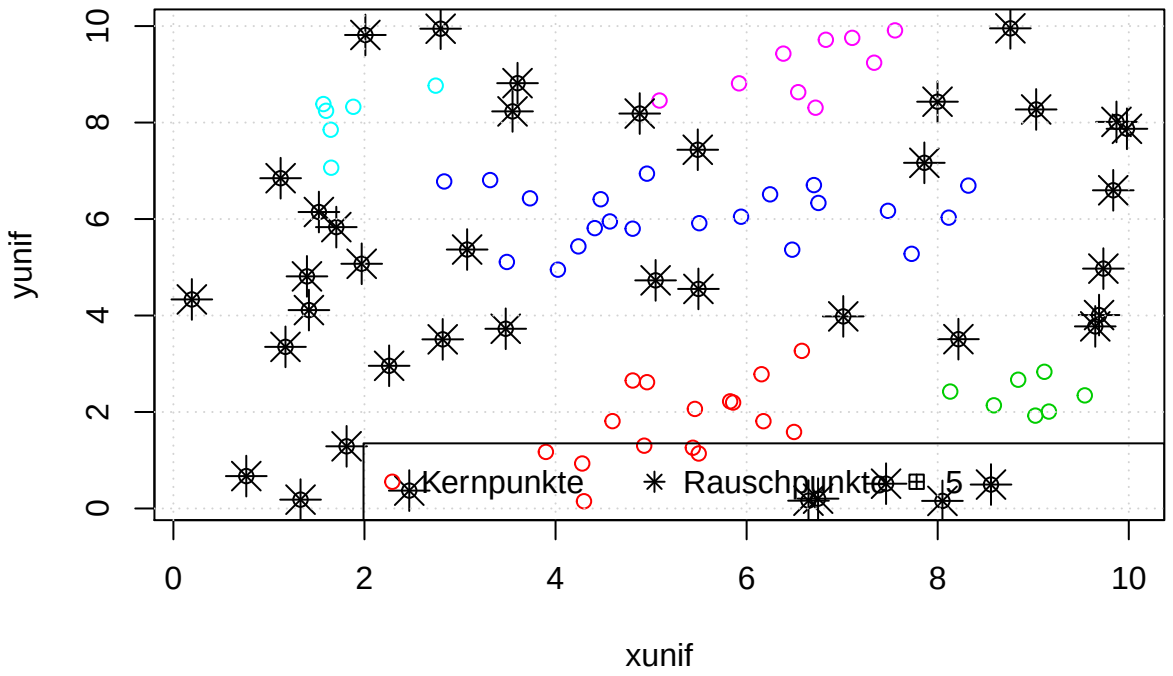
```
for (i in c(1:10)) {
  dbunif(unif,i)
  i=i+1
}
```

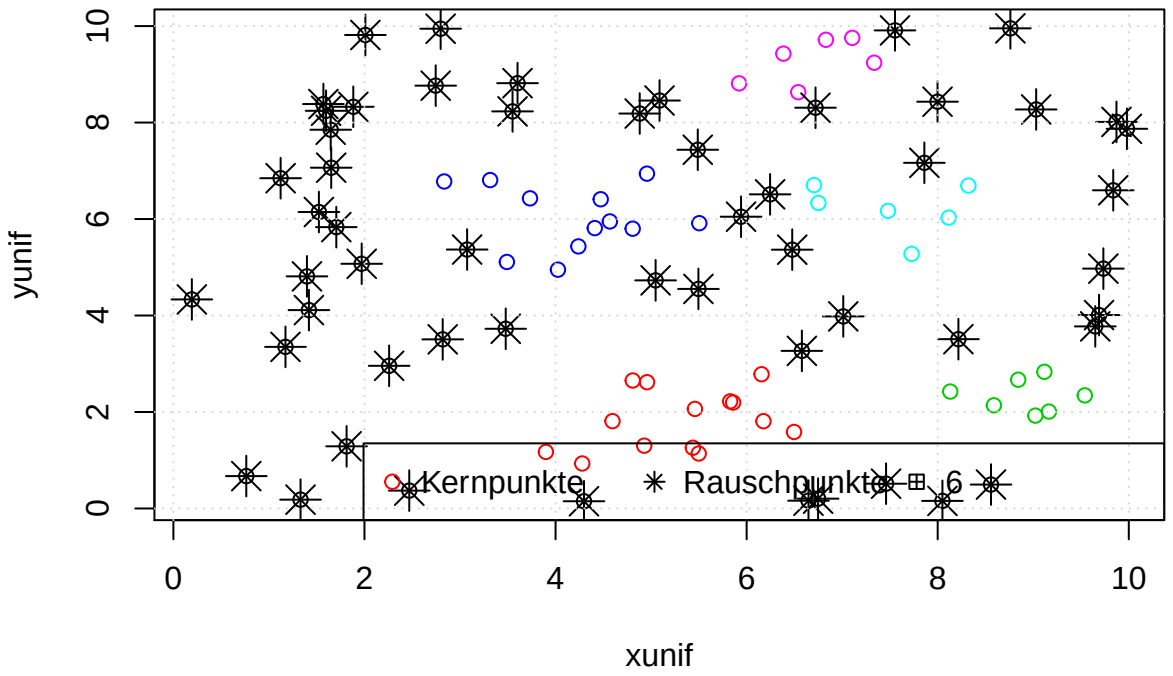


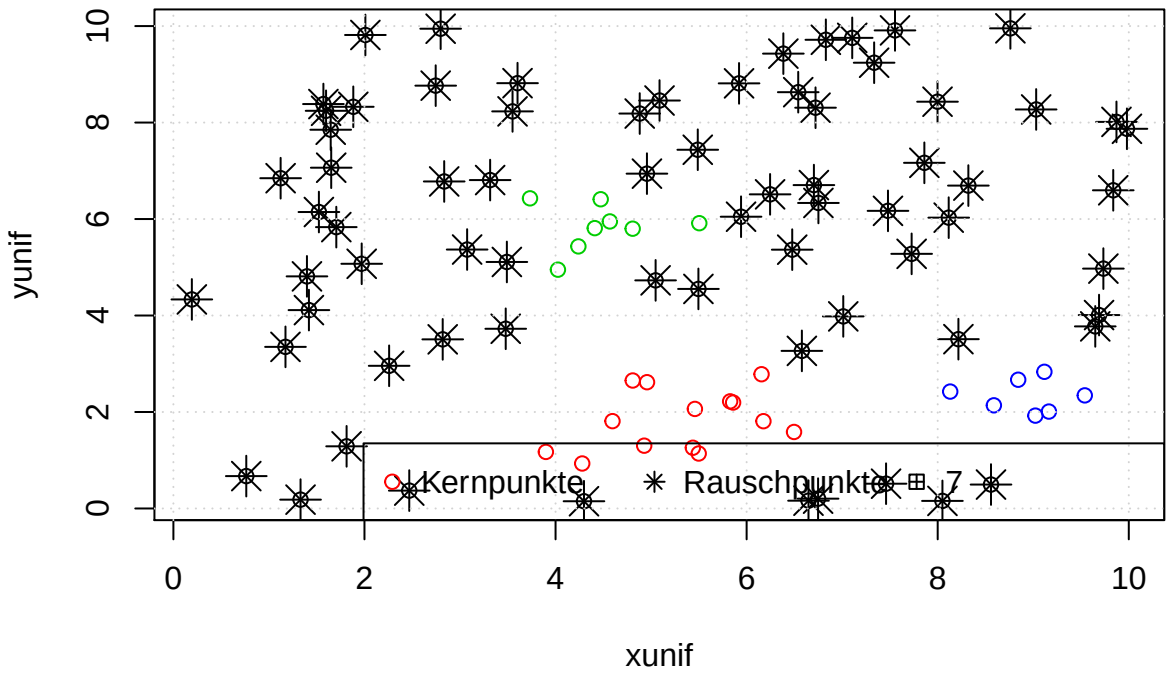

yunif
xunif
0
2
4
6
8
10
Kernpunkte
Rauschpunkte
2

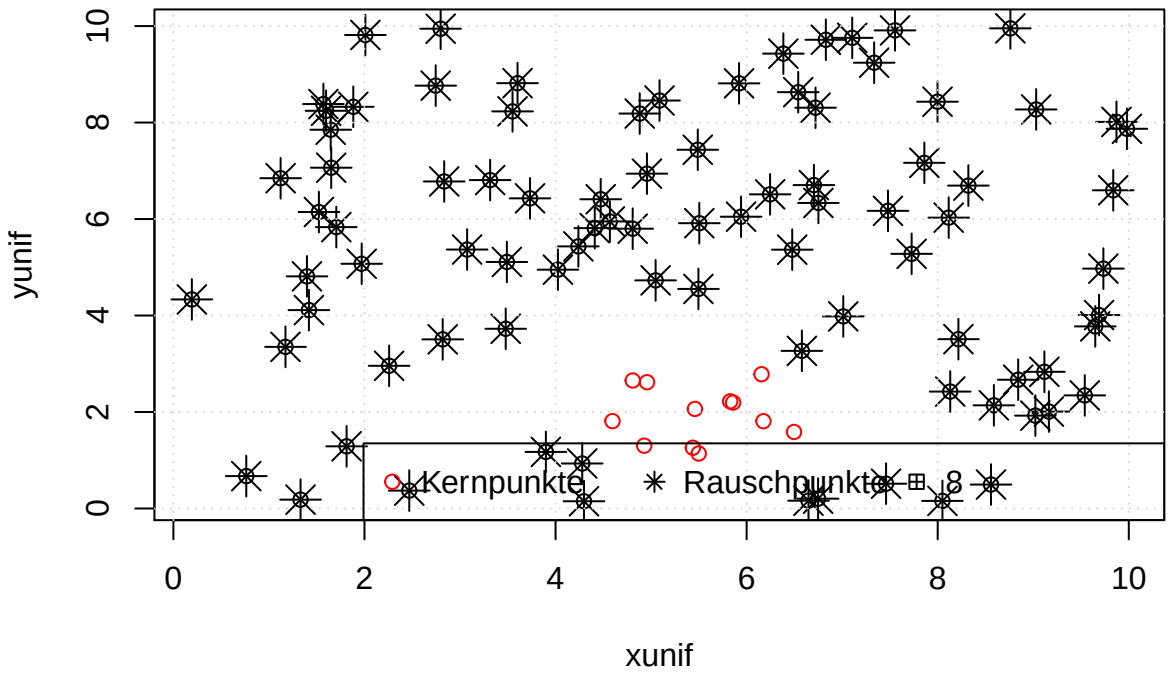

yunif
xunif
0
2
4
6
8
10
Kernpunkte
Rauschpunkte
3

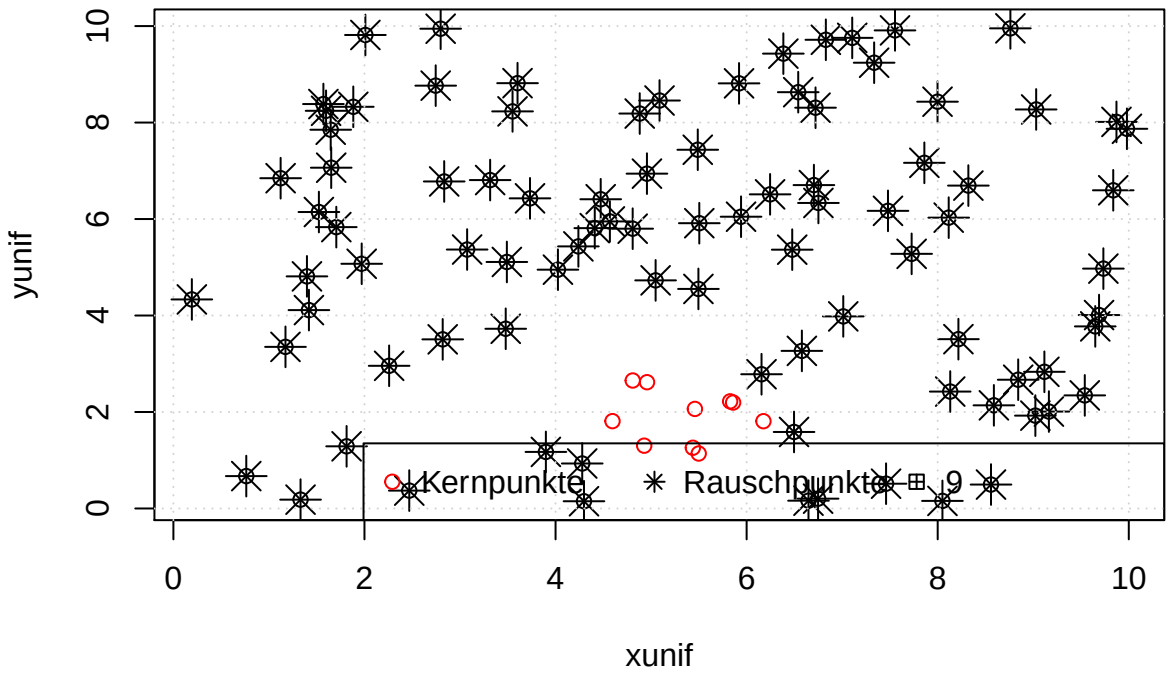

yunif
xunif
0
2
4
6
8
10
Kernpunkte
Rauschpunkte
4

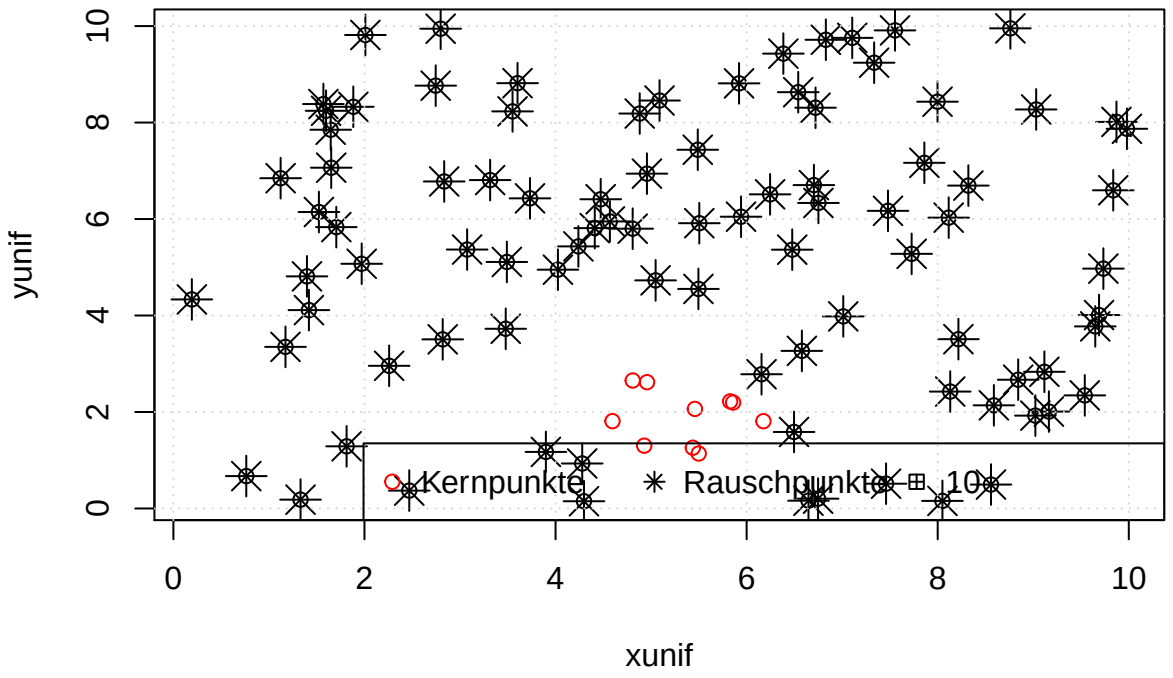

yunif
xunif
0
2
4
6
8
10
Kernpunkte
Rauschpunkte
5


yunif
xunif
0
2
4
6
8
10
Kernpunkte
Rauschpunkte
6

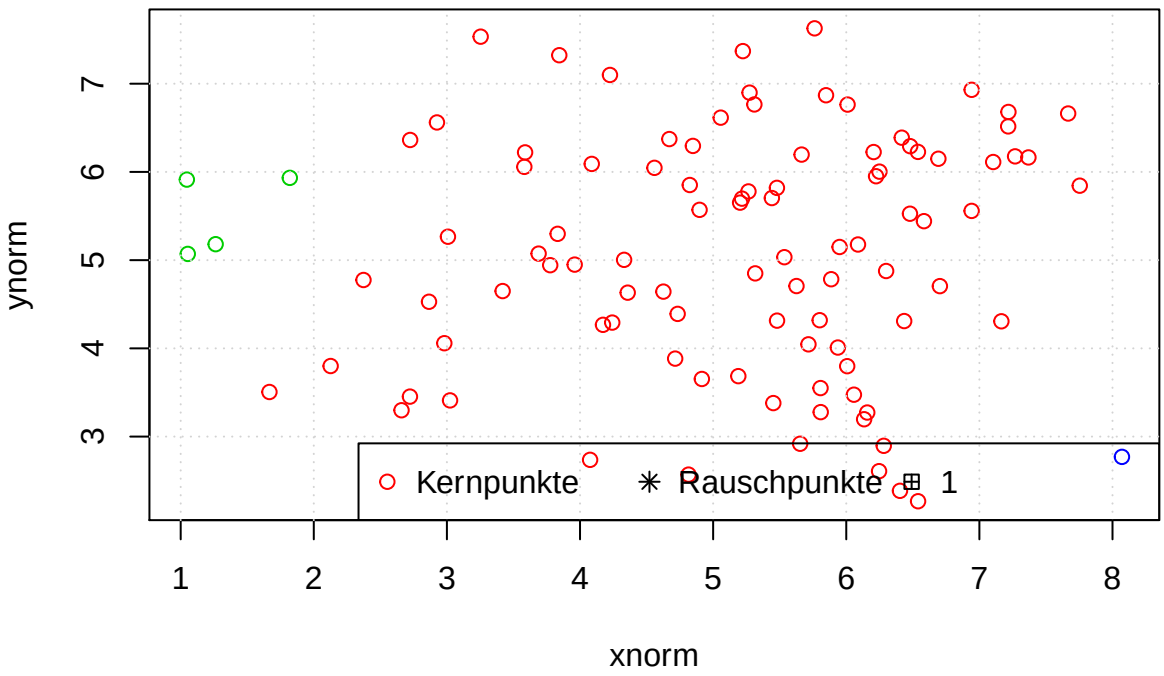

yunif
xunif
0
2
4
6
8
10
Kernpunkte
Rauschpunkte
7

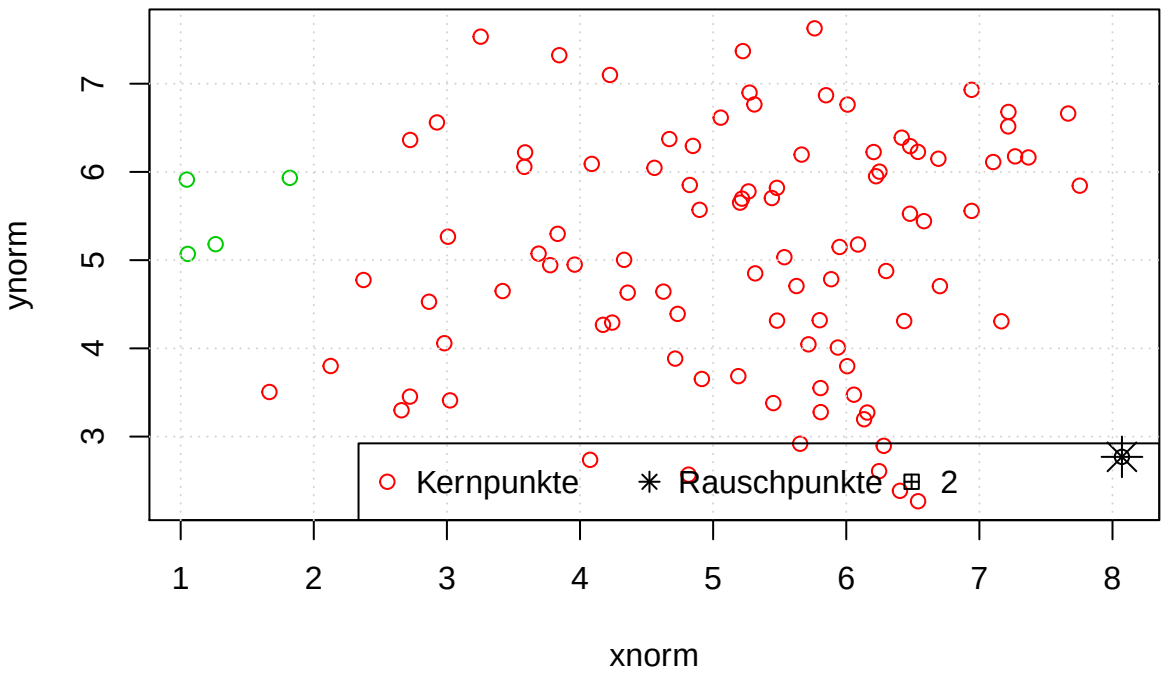

yunif
xunif
0
2
4
6
8
10
Kernpunkte
Rauschpunkte
8

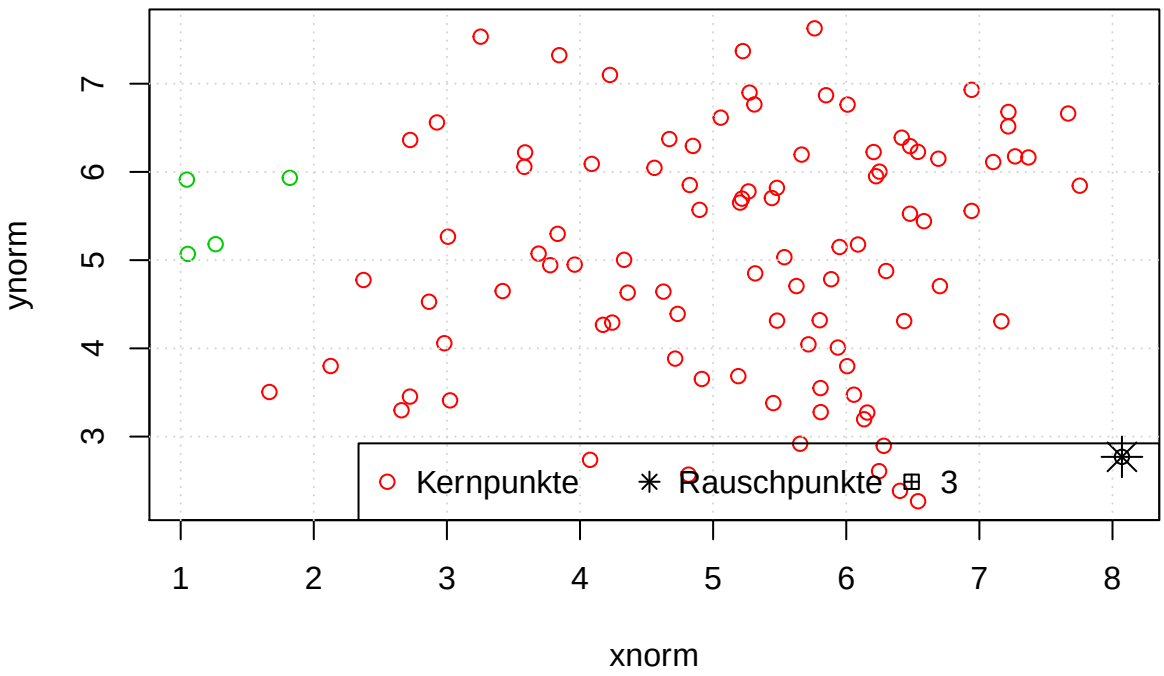

yunif
xunif
0
2
4
6
8
10
Kernpunkte
Rauschpunkte
9

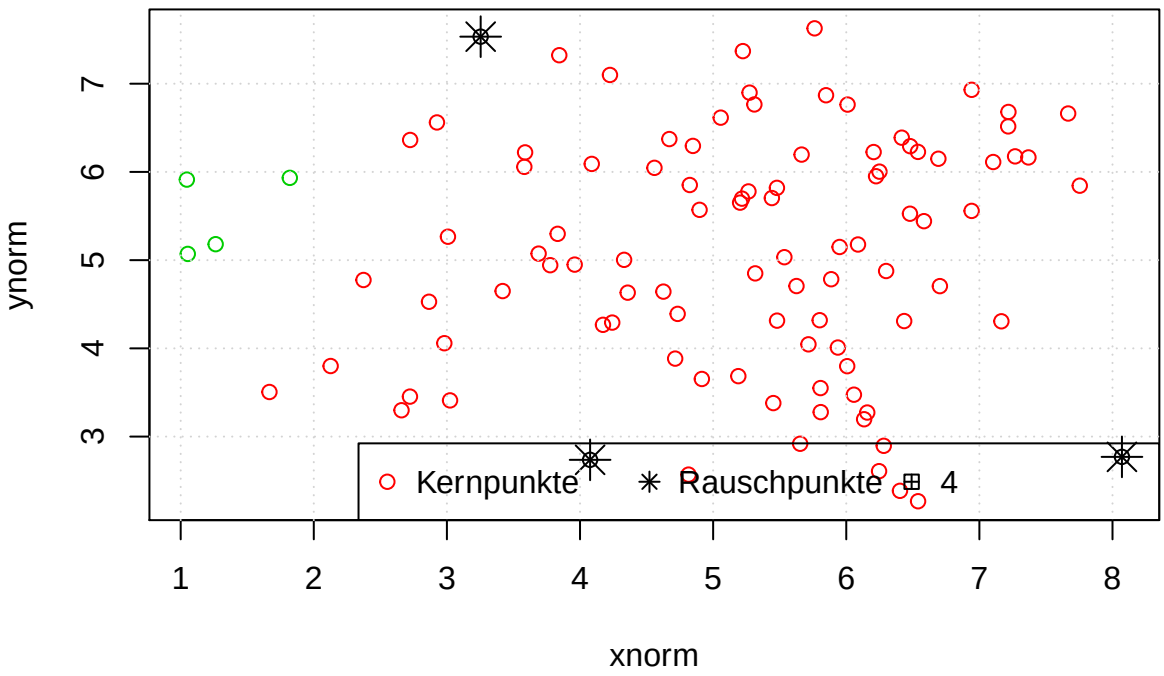


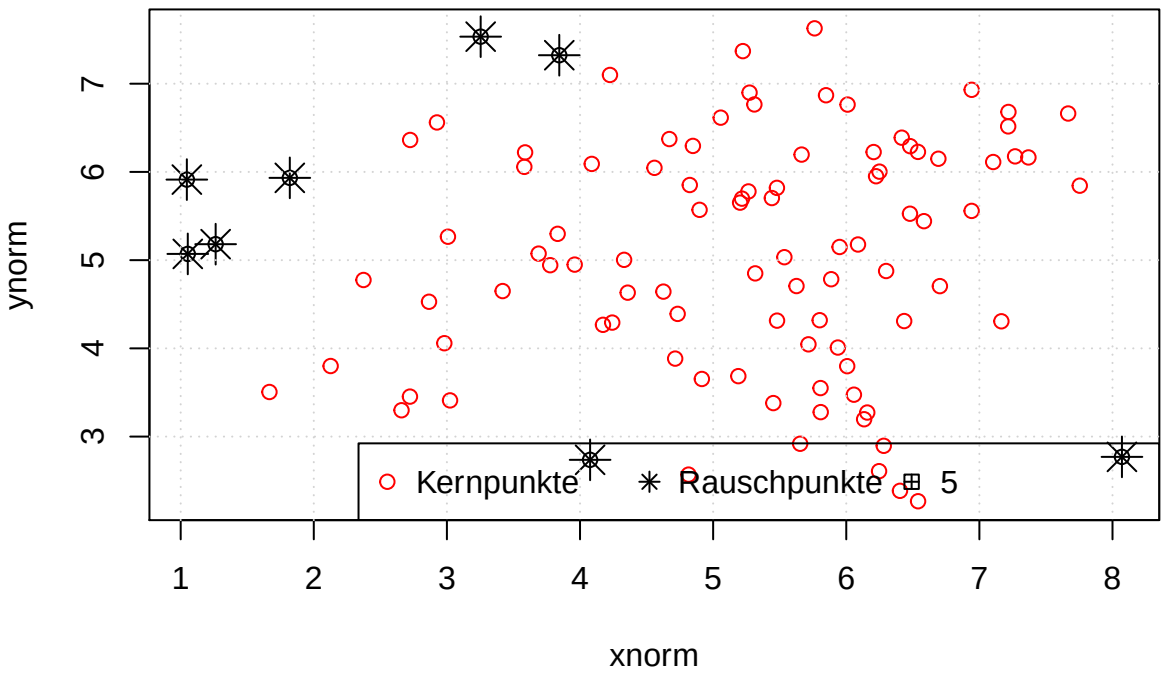

```
for (i in c(1:10)) {
  dbunif(norm,i)
  i=i+1
}
```

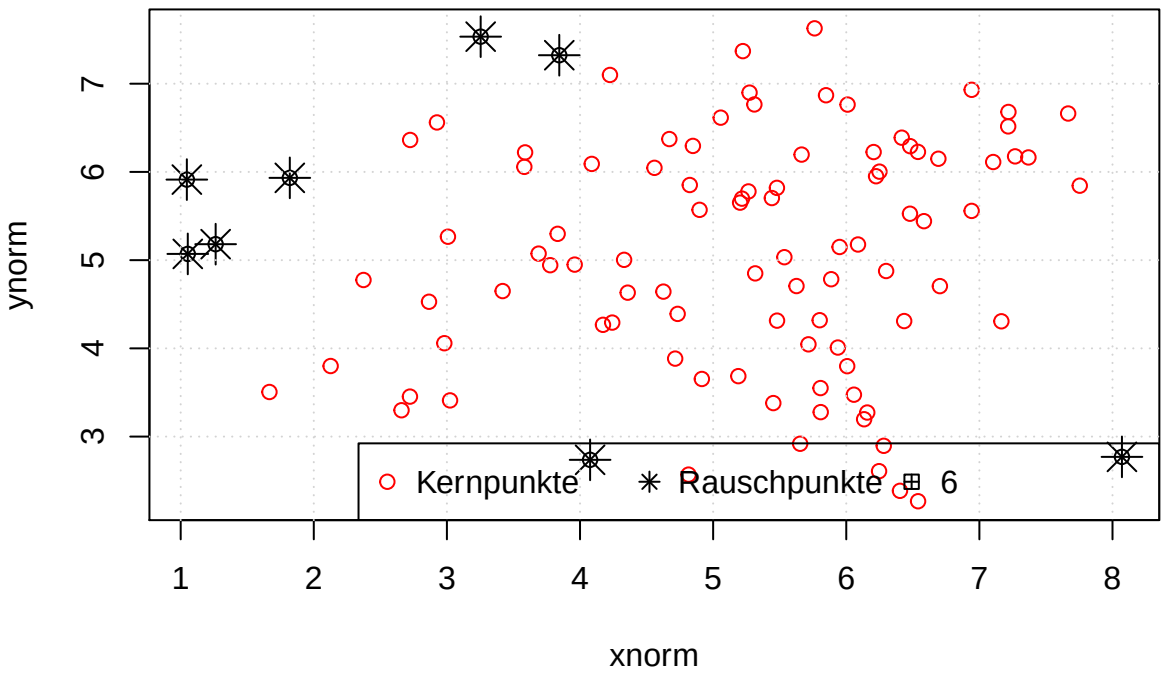

ynorm
xnorm
Kernpunkte
Rauschpunkte
1

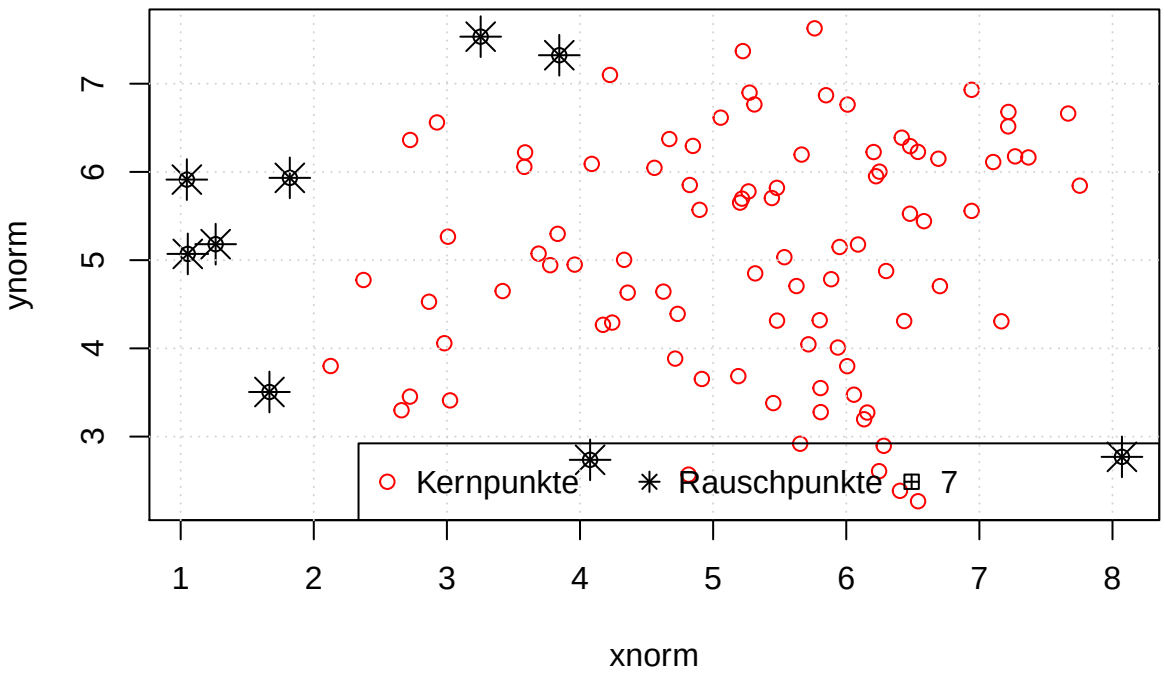

ynorm
xnorm
Kernpunkte
Rauschpunkte
2

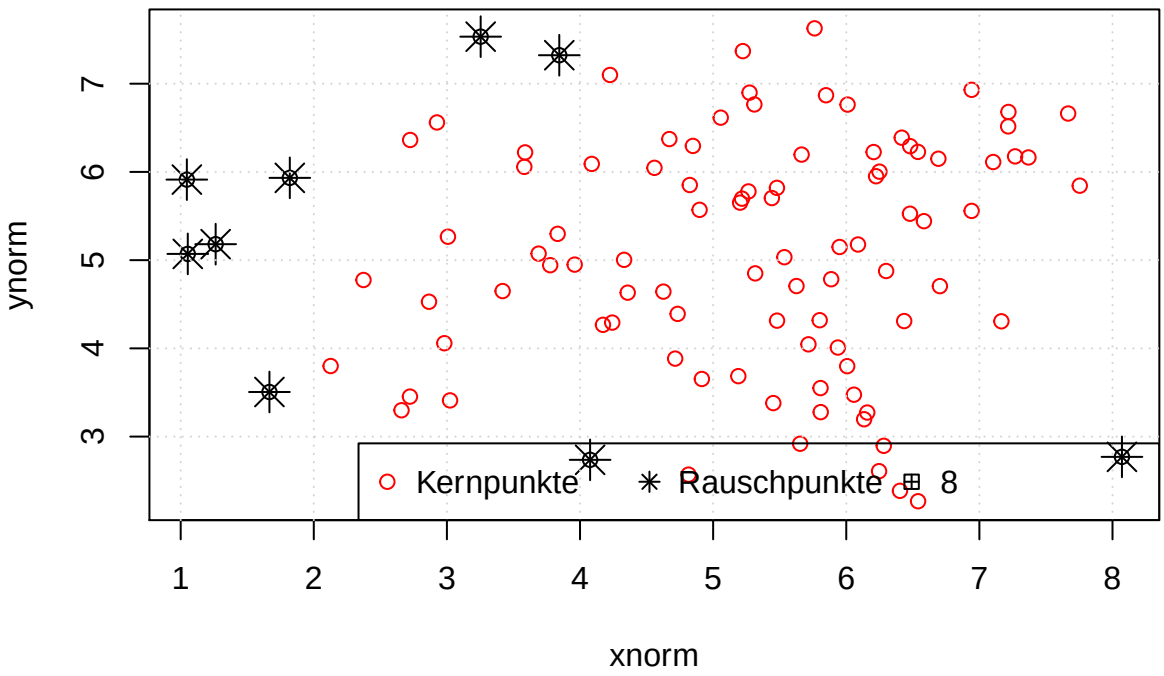

ynorm
xnorm
Kernpunkte
Rauschpunkte
3

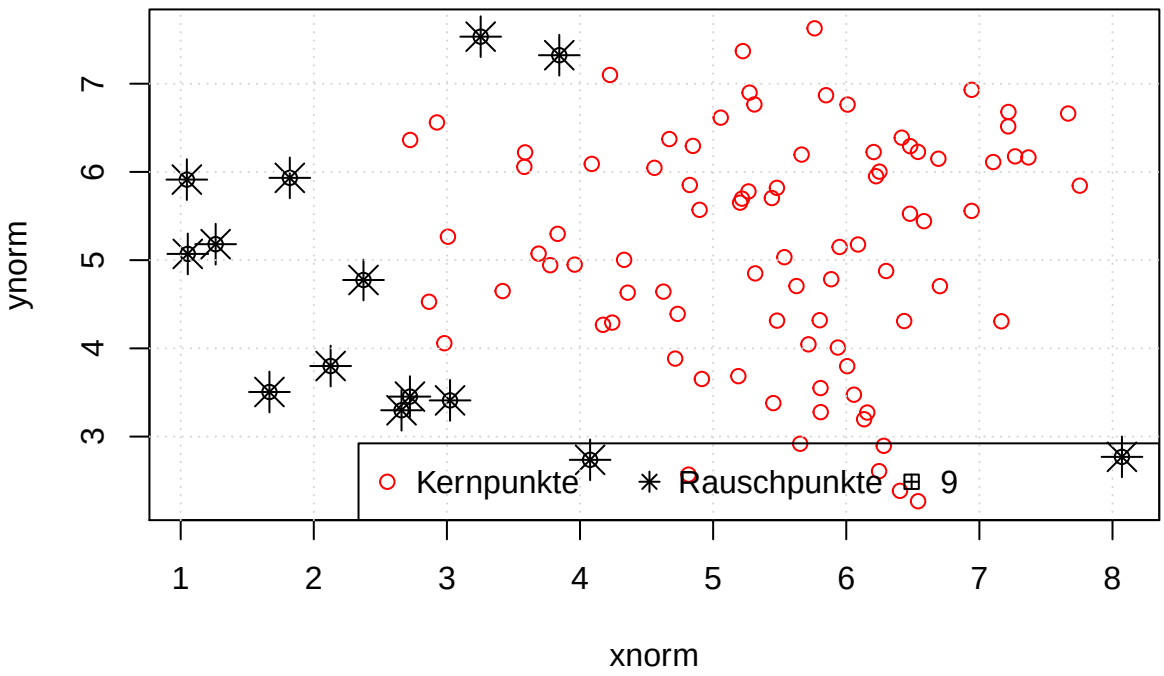

ynorm
xnorm
3
4
5
6
7
1
2
3
4
5
6
7
8
Kernpunkte
Rauschpunkte
4

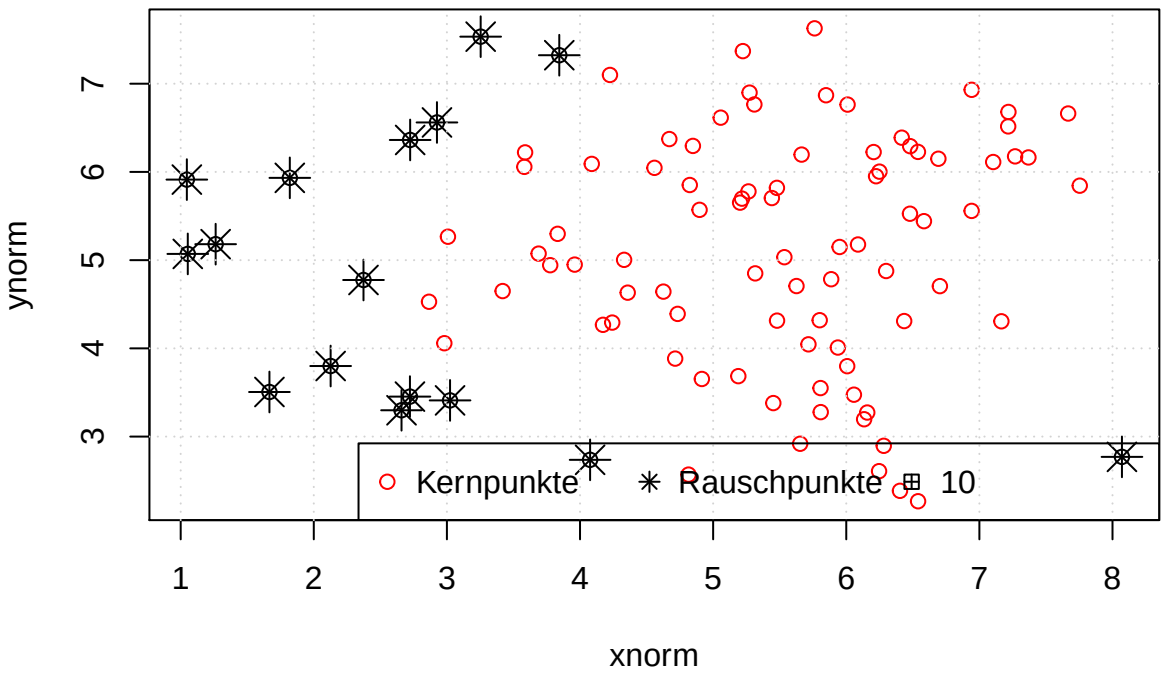

ynorm
xnorm
Kernpunkte
Rauschpunkte
5


ynorm
xnorm
Kernpunkte
Rauschpunkte
6


ynorm
xnorm
Kernpunkte
Rauschpunkte
7


ynorm
xnorm
Kernpunkte
Rauschpunkte
8


ynorm
xnorm
Kernpunkte
Rauschpunkte
9



Es ist deutlich zu erkennen, dass die Anzahl der Rauschpunkte mit steigendem minPts Wert steigt. Da die Punkte, welche genügend nahe Nachbarn in der Epsilonumgebung besitzen, mit steigendem minPts Wert sinkt, war das auch zu erwarten.