

An introduction to Mathematical Statistics

Exam 2025

All the exercises are independent and may be done in the order you want. You may use R software at any time it may help you. You can use the result stated in a previous question even if you have not been able to prove it. Please include all the commands you want to use (even the codes given during the lecture) in your script. At the end of the exam, send your script by email at **laurent.delsol@univ-orleans.fr**.

The given marking scheme may change a little. The total is greater than 20 to take into account the length of the exercises with respect to the exam duration. Every **BONUS** question gives 1 additional point.

Exercise 1: Catch mark release recapture

Our aim is to estimate the unknown number of fishs living in a lake, denoted θ hereafter. because we have to take care of the fishs we have one additional constraint, we can only catch fishs one by one. Hence we have to put into the lake before the next catch. However, we may add one mark on the fish.

The method is the following:

- **[CATCH and MARK]** First catch and mark a known number **M** of fishs
- **[RELEASE]** Then wait a few days to let time to the marked fishs to be uniformly distributed in the whole lake
- **[RECAPTURE]** Finally start fishing again and observe for every caught fish if it is marked or not. The probability to catch a marked fish is then $p_\theta = \frac{M}{\theta}$.

Start by loading the **CMR.RData workspace**. It contains two variables:

- **x**: the observations (of X) collected at the last **RECAPTURE** step (marked: 1, unmarked: 0).
- **y**: the observations corresponding to Y, the number of unmarked fishs caught before the next marked one, collected at the last **RECAPTURE** step.

Note that in that experiment we have set **M=20**.

```
load('CMR.RData')
```

Part A: X and Bernoulli [Total: 3.5 pts]

The variable **x** contains the observations (of X) collected at the last **RECAPTURE** step (marked: 1, unmarked: 0). Now clearly, $X_1, \dots, X_n \stackrel{i.i.d.}{\sim} Ber(p_\theta)$, with $p_\theta = \frac{M}{\theta}$.

2. Construct an estimator $\hat{\theta}_1$ of θ from the method of moments. [0.75 pt]
3. The bias of $\hat{\theta}_1$ is very difficult to compute. Have a look to $\hat{\theta}_2 = \frac{M(n+1)}{n\bar{X}+1} = \frac{M(1+\frac{1}{n})}{\bar{X}+\frac{1}{n}}$. Prove $\mathbb{E}[\hat{\theta}_2] = \theta(1 - (1 - p_\theta)^n)$ and compute the bias of $\hat{\theta}_2$. Is $\hat{\theta}_2$ asymptotically unbiased? [2 pts]

Hint: $n\bar{X} \sim Bin(n, p_\theta)$ hence $\mathbb{E}[\hat{\theta}_2] = \sum_{k=0}^n \frac{M(n+1)}{k+1} \frac{n!}{k!(n-k)!} p_\theta^k (1 - p_\theta)^{n-k}$.

4. Compute the values of $\hat{\theta}_1$ and $\hat{\theta}_2$ from the observations. [0.75 pt]

```
20/mean(x)
```

```
## [1] 100
```

```
20*(length(x)+1)/(sum(x)+1)
```

```
## [1] 96.19048
```

Part B: Y and geometric [Total: 3 pts]

The variable **y** contains the observations corresponding to **Y**, the number of unmarked fish caught before the next marked one, collected at the last **RECAPTURE** step. Now clearly, $Y_1, \dots, Y_r \stackrel{i.i.d.}{\sim} \mathcal{G}(p_\theta)$, with $p_\theta = \frac{M}{\theta}$. **Hint:** if $Z \sim \mathcal{G}(p)$ then $\forall k \in \mathbb{N}, \mathbb{P}_1(Z = k) = (1-p)^k p$, $\mathbb{E}_p[Z] = \frac{1-p}{p}$ and $Var_p(Z) = \frac{1-p}{p^2}$. [1.5 pt]

2. Construct an estimator $\hat{\theta}_3$ of θ from $Y_1, \dots, Y_r$ and the method of moments. [1 pt]

3. Compute the bias and variance of $\hat{\theta}_3$.

4. Compute the value of $\hat{\theta}_3$ from the observations. [0.5 pt]

```
20*(mean(y)+1)
```

```
## [1] 100
```

Exercise : Archery competition

A great archery competition have organised between three legends of that sport: Rebel, Robin and Legolas.

Start by loading the **Archery2025_2.RData** workspace.

It contains three variables:

- **Archer**: shooter's name
- **X**: distance to the center of the target
- **Pos**: position (in 2 dimensions) of the arrow on the target
- **Pos0**: position (in 2 dimensions) of the 4 final arrows on the target

A. Distance to the center of the target

We observe first **X**, the distance between the arrows and the center of the target.

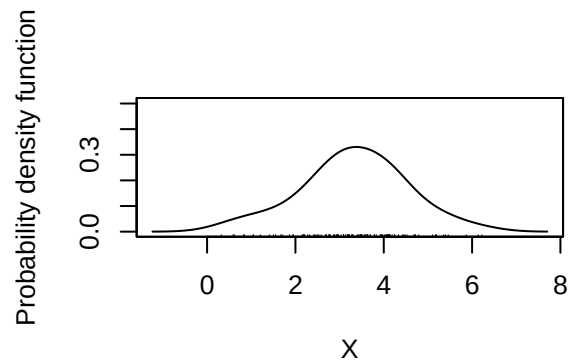
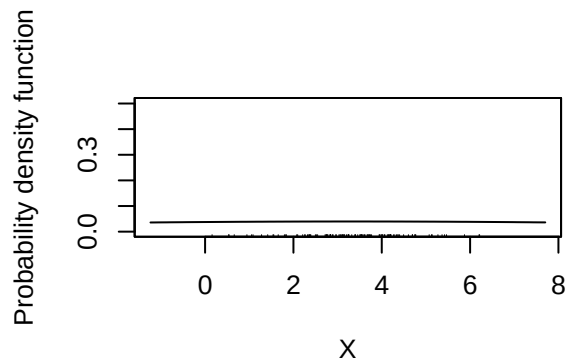
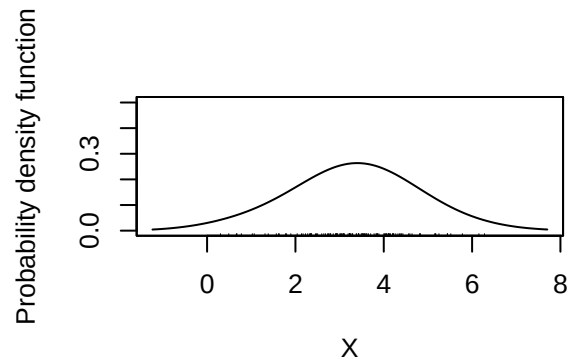
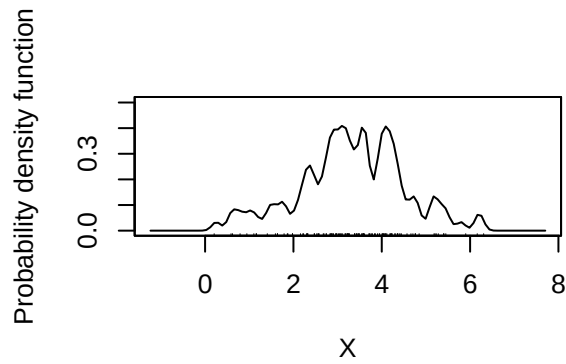
Part 1: Non parametric estimation: [Total: 3 pts +1pt BONUS]

1. Prove that $\hat{f}$ is a density if K is a density. [1 pt]
2. Give a nonparametric estimation of the density f of **X** using the kernel density estimator $\hat{f}$ using as smoothing parameter 0.1, 1, 10 and the value given by cross-validation criterion (hint: share the graphic window into four plots). Comment. [1.5 pts]

```
load('Archery2025_2.RData')
library(sm)
```

```
## Package 'sm', version 2.2-6.0: type help(sm) for summary information
```

```
par(mfrow=c(2,2))
sm.density(X,nbins=0,h=0.1)
sm.density(X,nbins=0,h=1)
sm.density(X,nbins=0,h=10)
sm.density(X,nbins=0,method='cv')
```

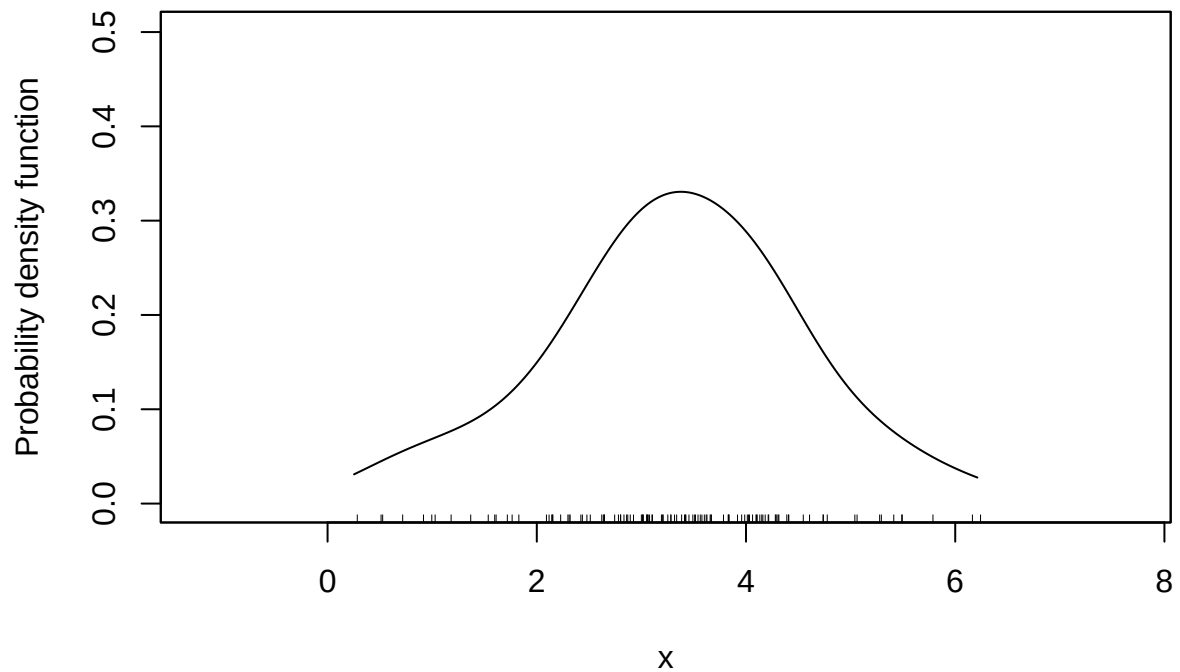


1.

Give a nonparametric estimation of the mode of **X**. [0.5 pt]

```
sm.mode=function(x,a,b,M,method='cv',plot=T)
{
  disc=seq(a,b,length.out=M)
  display="line"
  if (plot) {display="none"}
  dens=sm.density(x,eval.points=disc,method=method,nbins=0)$estimate
  mod=disc[(order(dens))[M]]
  max=max(dens)
  return(list(mode=mod,max=max))
}

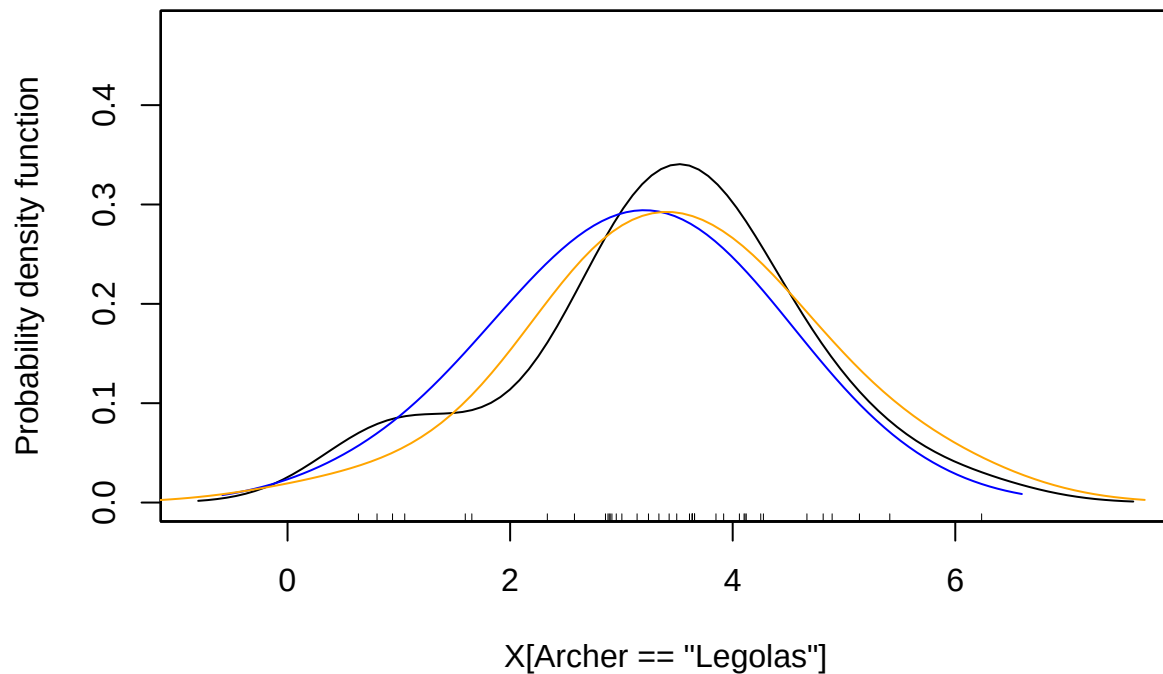
sm.mode(X,min(X),max(X),M=1000,method='cv')
```



```
## $mode
## [1] 3.378783
##
## $max
## [1] 0.3306078
```

1. [BONUS] Plot on the same graph the estimated densities obtained for the three archers and compare them. Do you expect supervised classification from **X** and **Archer** could be efficient?

```
sm.density(X[Archer=='Legolas'],nbins=0,method='cv')
sm.density(X[Archer=='Robin'],nbins=0,method='cv',add=TRUE,col='blue')
sm.density(X[Archer=='Rebel'],nbins=0,method='cv',add=TRUE,col='orange')
```



Part 2: Potential parametric estimation: [Total: 7 pts +2pts BONUS]

If the position of the arrows follow a bidimensional gaussian distribution $\mathcal{N}((0,0)^T, \sqrt{\theta}I_2)$, then f should have the following expression :

$$f(x, \theta) = \frac{x}{\theta} e^{-\frac{x^2}{2\theta}} 1_{\{x \geq 0\}},$$

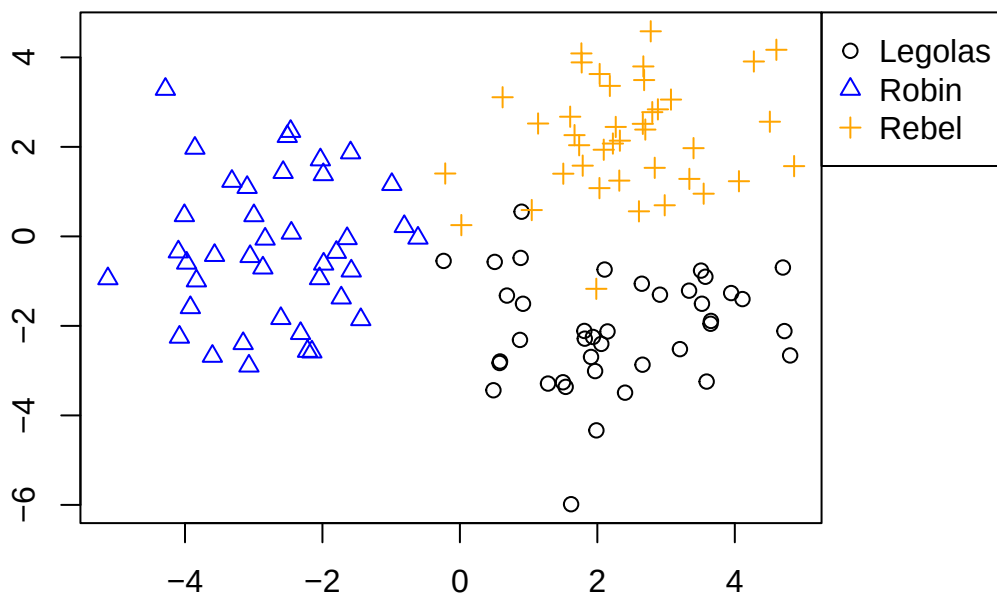
whith $\theta > 0$.

1. Now use the parametric model to propose an unbiased estimator $\hat{\theta}$ of θ from maximum likelihood method. [1.5 pts]
2. Compute its bias and quadratic error (**hint**: $Y = \frac{X^2}{\theta} \sim \mathcal{E}(\frac{1}{2})$) [1 pt]
3. Prove that this parametric model defines an exponential family and deduce a sufficient and complete variable. What does this imply for $\hat{\theta}$? [1.75 pt]
4. Compute the Fisher Information of the sample. [1.25 pt]
5. Is $\hat{\theta}$ an efficient estimator of θ ? [0.5 pt]
6. Plot on the same graph the empirical distribution (as in Part 1, question 3 with the best choice of h) and the estimated parametric distribution. Comment the result. [1 pt]
7. [BONUS] Give a parametric estimation of the mode of x from the parametric model and $\hat{\theta}$. **Hint**: You should optimize the function $x \mapsto \frac{x}{\theta} e^{-\frac{x^2}{2\theta}} 1_{\{x \geq 0\}}$ with respect to x .
8. [BONUS] Compare the values of $\hat{\theta}$ obtained for the three archers. Do you expect parametric supervised classification from **X** and **Archer** could be efficient?

B. Discrimination from the position (in 2 dimensions) of the arrow on the target [Total: 3 pts]

The competitors fire their final arrows simultaneously, so it's impossible to know whose arrows they belong to. On the target, the four arrows land at positions (1.4, 1.3), (1.4, -1.2), (-2, 0), and (-0.1, 0.2). How could you identify the archers who fired them?

1. Compare the positions of the arrows for the different archers on the graph below. What do you observe? Why is this promising? [0.5 pt]



1. Use the **Classif_NP** function seen in class to put into practice the method explained in question 2 using the position and Archer variables. Comment on the result. This may take around one minute to run on your laptop. [1.5 pt]

```
##### ROC
ROC=function(z,p1,plot=FALSE)
{
  p1=pmin(pmax(p1,0),1)
  p=c(-0.0001,sort(unique(p1)),1.0001)#
  #p=seq(-0.01,1.01,0.01)#
  #c(-0.0001,sort(unique(p1))-0.0000000001,1.0001)
  lp=length(p)
  ROC=matrix(NA,lp,2)
  for (s in 1:lp)
  {
    ROC[s,1]=mean(p1[z==0]>=p[s])
    ROC[s,2]=mean(p1[z==1]>=p[s])
  }
  #AUC=sum((ROC[-(lp),2])*(ROC[-(lp),1]-ROC[-(1),1]))
  AUC=sum((ROC[-(lp),2]+ROC[-1,2])*(ROC[-(lp),1]-ROC[-(1),1])/2)

  colnames(ROC)=c("FPR", "TPR")
  if (plot)
  {
    plot(ROC,type='l',main=paste('AUC =', AUC),xlim=c(0,1),ylim=c(0,1))
    abline(0,1,col='orange')
  }

  return(list(ROC=ROC,AUC=AUC))
}
Classif_NP=function(X,Y,X0=NA,plot=FALSE)
{
  X=as.matrix(X)
  n=length(Y)
  V=sort(unique(Y))
  n_V=length(V)
  Prob=matrix(NA,n,n_V)
  colnames(Prob)=V
  Class=rep(NA,n)
  if (!is.na(max(X0)))
  {
    X0=as.matrix(X0)
    P0=matrix(NA,nrow(X0),n_V)
    colnames(P0)=V
    Class0=rep(NA,n)
  }
  for (v in 1:n_V)
  {
    z=as.numeric(Y==V[v])
    for (i in 1:n)
    {
      value=sm.regression(X[-i,],z[-i], eval.points=X,eval.grid=FALSE,display='none',method='cv',nbins=
      if (!is.na(value)) {Prob[i,v]=value}
      if (is.na(value)) {Prob[i,v]=mean(z[-i])}
    }
    if (!is.na(max(X0))) {P0[,v]=sm.regression(X,z,eval.points=X0,eval.grid=FALSE,display='none',method=
    P0[is.na(P0[,v]),v]=mean(z)}
  }
}
```

```

}
if (n_V==2) {Roc=ROC(Y==V[2],Prob[,2],plot)}
Class=V[apply(Prob,1,which.max)]
V_est=sort(unique(Class))
if (length(V_est)==n_V){M_table=table(Y,Class)}
else {
  M_table=matrix(0,n_V,n_V)
  M_table0=table(Y,Class)
  for (j in 1:length(V_est)) {M_table[,which(V==V_est[j])]=M_table0[,j]}
}
Err=1-(sum(diag(M_table))/sum(M_table))
if (!is.na(max(X0))) {Class0=V[apply(P0,1,which.max)]}
if (!is.na(max(X0))) {return(list(Class=Class, Prob=Prob, M_table=M_table, Err=Err, Class0=Class0, Prob0=P0, AUC=AUC))}
else {return(list(Class=Class, Prob=Prob, M_table=M_table, Err=Err, AUC=ifelse(n_V==2,Roc$AUC,NA)))}
}

RE=Classif_NP(Pos,Archer,X0=Pos0)
RE$Err

```

```
## [1] 0.05
```

```
RE$M_table
```

```
##           Class
## Y           Legolas Rebel Robin
## Legolas       38      1      1
## Rebel         2      36      2
## Robin         0       0     40
```

1. Using what you did previously, for each of the last 4 arrows, can you correctly identify the archer who shot it? Explain why? [1 pt]

```
RE$Prob0
```

```
##           Legolas      Rebel      Robin
## [1,] 0.130998123 0.82910535 0.001054179
## [2,] 0.836653185 0.14129303 0.000229799
## [3,] 0.007004315 0.02623667 0.994878561
## [4,] 0.267315290 0.31095693 0.360491553
```

```
RE$Class0
```

```
## [1] "Rebel" "Legolas" "Robin" "Robin"
```

Exercise 3: Escape [Total: 4.5 pts]

Your friend Edmont Dantes has been unjustly imprisoned for a crime he didn't commit. You find a job as a prison gardener, hoping to help him escape by finding the location of the potential underground tunnel leading to the exit. Under the pretext of working on landscaping the courtyard, you manage, with the utmost discretion, to measure in places how deep you can sink your crowbar into the ground.

You can retrieve the observations collected in the **Evasion.RData** dataset.

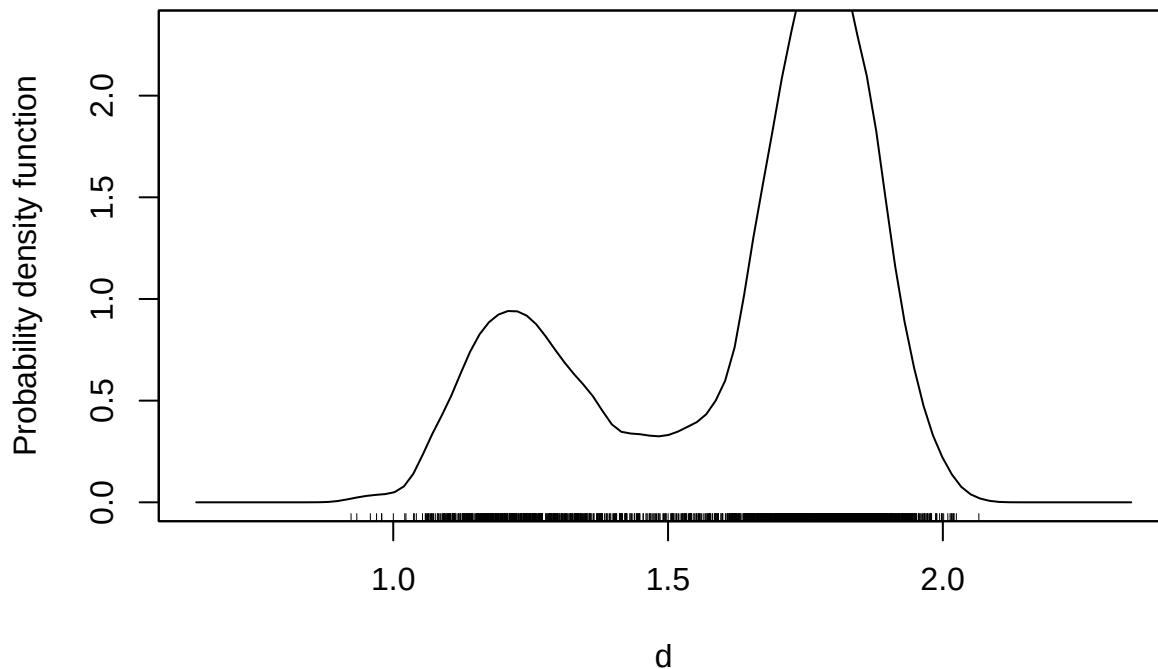
- **x1** and **x2** indicate the position (x-axis and y-axis)
- **d** indicates the depth

```
library(sm)
load('Evasion.RData')
length(d)
```

```
## [1] 2000
```

1. Estimate the density of the variable **d** using a suitable smoothing parameter. How many clusters can you may expect from the estimated density? [1 pt]

```
sm.density(d,method='cv',nbins=0)
```



2. Give the two (local and global) modal values, the threshold (valley point between them) and the extract the clusters from the estimated density. Why is this encouraging regarding the hope of finding a hidden underground tunnel? [1.5 pt]

```
sm.clustering.extrema=function(x,tau=10,a=min(x),b=max(x),M=1000,method='sj',plot=T)
{
  disc=seq(a,b,length.out=M)
  n=length(x)
  dens=sm.density(x,eval.points= disc,method=method,nbins=0)$estimate
  inc=rep(0,M)
  valleys=rep(0,M)
  k=1
  for (j in 1:M)
  {
    i1=max(1,j-tau)
    i2=min(M,j+tau)
    ratio=(dens[i2]-dens[i1])/(disc[i2]-disc[i1])
    if (ratio>0) {inc[j]=1}
    if (j>1) {if (inc[j]==0 & inc[j-1]==1 ) {k=k+1}}
    valleys[j]=k
  }
  if (plot) {plot(disc,dens,type='l')}
}
```

```

k=max(valleys)
thresholds=rep(NA,k)
mins=rep(NA,k)
for (j in 1:k)
{
  candidates=disc[valleys==j]
  thresholds[j]=candidates[which.min(dens[valleys==j])]
  mins[j]=min(dens[valleys==j])
  #if (plot) {segments(thresholds[j],0,thresholds[j],mins[j],col='orange')}
}
n=length(x)
cluster=rep(0,n)
for (j in 1:k) {if (j==1) {cluster[x==thresholds[j]]=j}
cluster[x>thresholds[j]]=j}
cluster.disc=rep(0,M)
for (j in 1:k) {if (j==1) {cluster.disc[disc==thresholds[j]]=j}
cluster.disc[disc>thresholds[j]]=j}
K=max(cluster.disc)
palette=rainbow(K+1)
if (plot) {for (j in 0:K) {if (sum(cluster.disc==j)!=0) {
disc.j=disc[cluster.disc==j]
dens.j=dens[cluster.disc==j]
n.j=sum(cluster.disc==j)
polygon(c(disc.j[1],disc.j,disc.j[n.j]),c(0,dens.j,0),
col = palette[j+1])}
} }
#cluster.on.data[cluster.on.data==0]=NA
return(list(cluster=cluster,thresholds=thresholds))
}
finding.extrema=function(x,tau=10,a=min(x),b=max(x),M=1000,method='cv',plot=T)
{
  disc=seq(a,b,length.out=M)
  n=length(x)
  dens=sm.density(x,eval.points= disc,method=method,nbins=0,display='none')$estimate
  inc=rep(0,M)
  valleys=rep(0,M)
  hills=rep(0,M)
  kv=1
  kh=1
  for (j in 1:M)
  {
    i1=max(1,j-tau)
    i2=min(M,j+tau)
    ratio=(dens[i2]-dens[i1])/(disc[i2]-disc[i1])
    if (ratio>0) {inc[j]=1}
    if (j>1) {if (inc[j]==0 & inc[j-1]==1 ) {kv=kv+1}}
    if (j>1) {if (inc[j]==1 & inc[j-1]==0 ) {kh=kh+1}}
    valleys[j]=kv
    hills[j]=kh
  }
  if (plot) {plot(disc,dens,type='l')}

  kv=max(valleys)

```

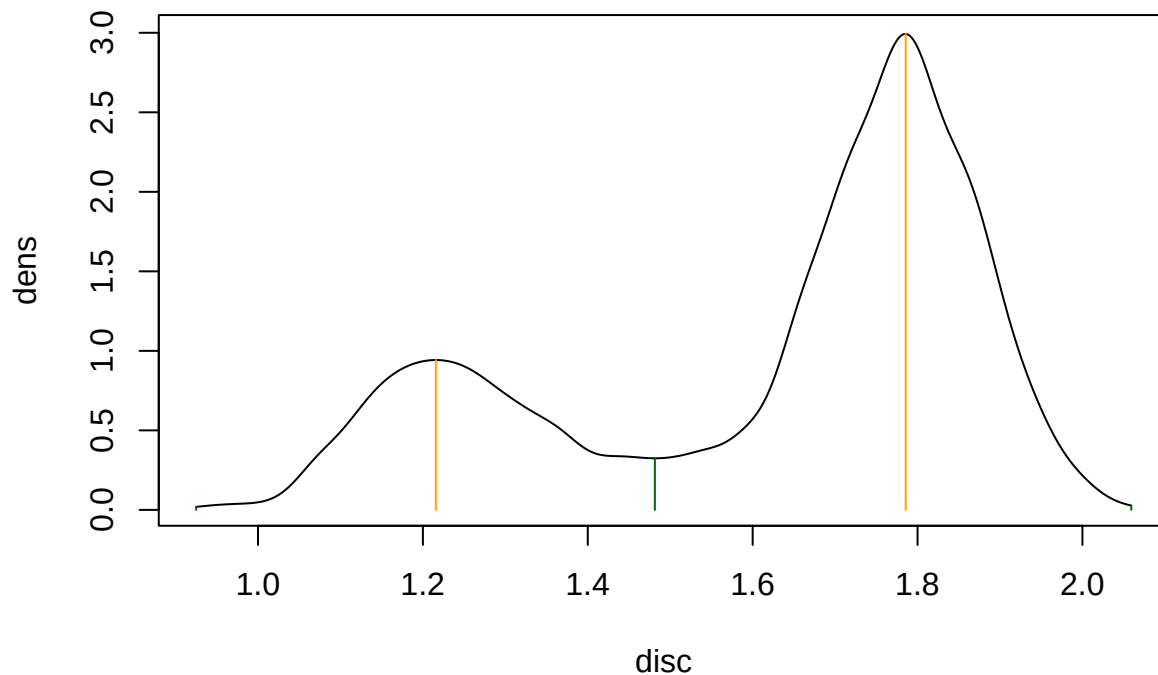
```

argmin=rep(NA,kv)
min=rep(NA,kv)
kh=max(hills)
argmax=rep(NA,kh)
max=rep(NA,kh)

for (j in 1:kv)
{
  candidates=disc[valleys==j]
  argmin[j]=candidates[which.min(dens[valleys==j])]
  min[j]=min(dens[valleys==j])
  if (plot) {segments(argmin[j],0,argmin[j],min[j],col='darkgreen')}
}
for (j in 1:kh)
{
  candidates=disc[hills==j]
  argmax[j]=candidates[which.max(dens[hills==j])]
  max[j]=max(dens[hills==j])
  if (plot) {segments(argmax[j],0,argmax[j],max[j],col='orange')}
}

return(list(argmax=argmax,max=max,argmin=argmin,min=min))
}
finding.extrema(d,10)

```



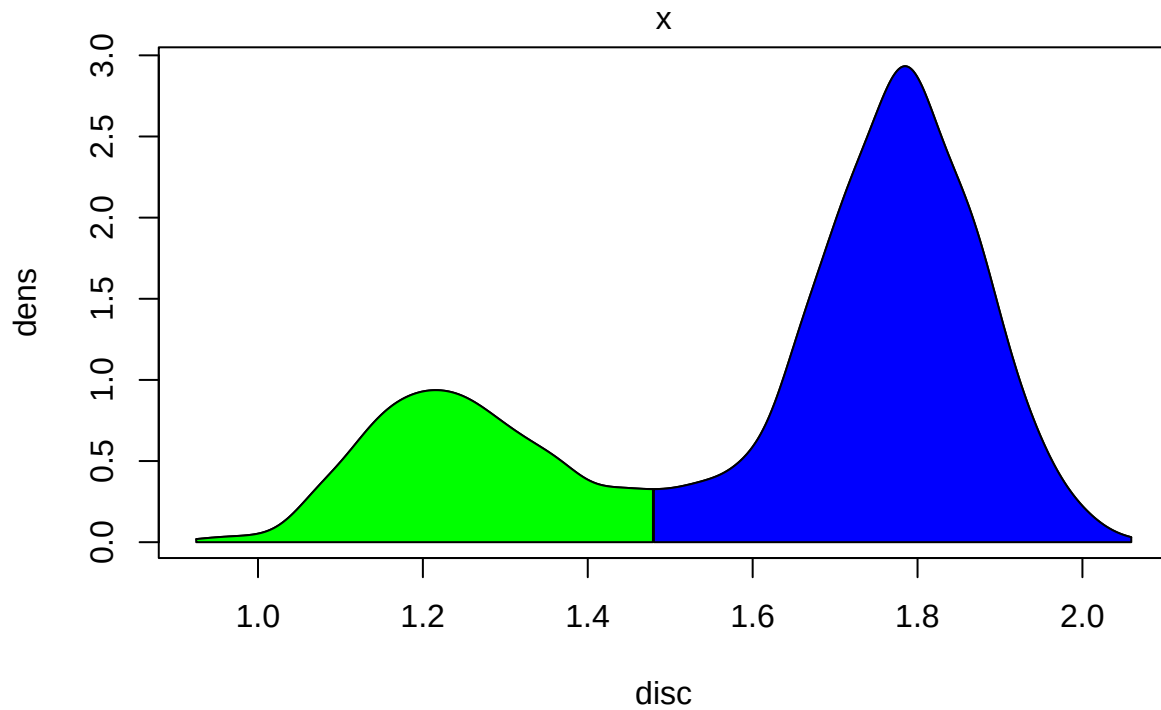
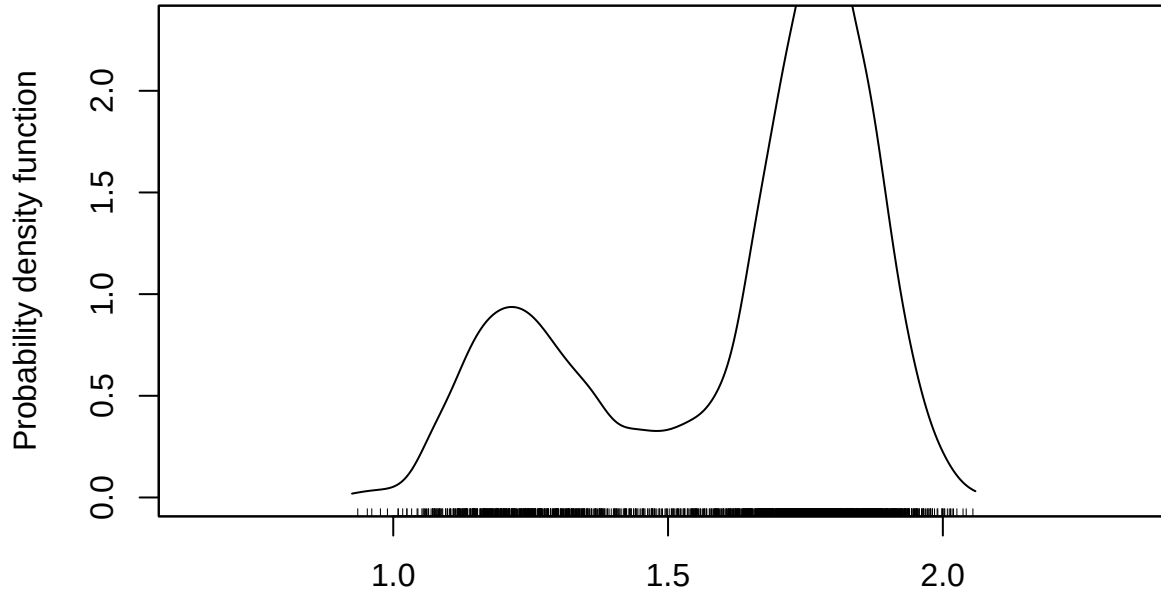
```

## $argmax
## [1] 1.215669 1.785644
##
## $max
## [1] 0.9422981 2.9926070
##
## $argmin

```

```
## [1] 0.9250038 1.4813546 2.0592782
##
## $min
## [1] 0.01902836 0.32443699 0.02718937
```

```
sm.clustering.extrema(d,10)
```



```
## $cluster
## [1] 2 1 1 2 2 2 2 1 2 2 2 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 1 2
## [38] 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 2 2 2 2 2 1 2 2 1 2 1 1
## [75] 2 2 1 2 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 2 2 1 1 1 2
## [112] 2 2 1 1 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 1 1 1 1 1 2 2 2 2 1 2 1 1 2 1 2 2 1
```

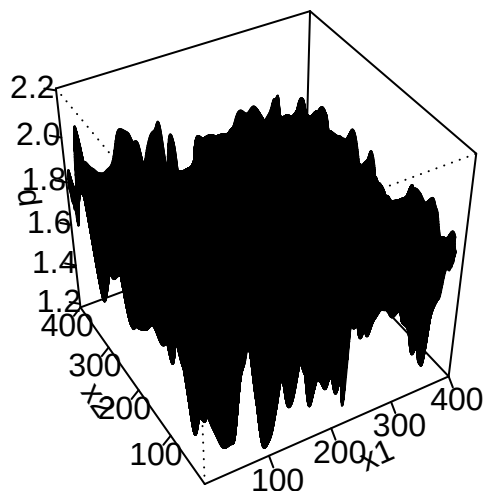
```

## [149] 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2
## [186] 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 2 2 2 2 1 1 2 1 2 2 2 2 2 2 1 2
## [223] 2 1 1 2 1 2 2 2 1 2 2 2 1 2 2 1 2 2 2 1 1 2 1 1 2 2 2 2 2 2 2 2 1 1 1 1
## [260] 1 2 1 2 1 1 2 2 2 2 2 1 2 2 1 2 1 2 2 2 1 2 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2
## [297] 2 2 1 2 2 2 2 2 1 1 2 1 1 2 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2
## [334] 2 2 2 2 2 2 2 1 1 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2
## [371] 2 2 2 1 2 1 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 1 2 2 1 2 2 2 1 2 2 1 2 2 2
## [408] 1 1 1 2 1 2 1 1 1 2 1 1 2 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2 2 1 2
## [445] 2 1 2 1 1 1 2 2 2 2 1 1 2 2 1 2 1 2 2 2 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 2 1
## [482] 2 2 2 2 2 1 2 1 2 1 2 2 2 1 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2
## [519] 2 2 2 2 1 2 1 2 1 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2
## [556] 2 2 1 2 2 2 2 1 1 2 2 2 2 2 1 1 2 1 2 1 2 2 1 2 2 2 2 2 1 1 2 1 2 2 1 1 2
## [593] 1 2 2 2 2 1 2 1 1 2 2 2 1 2 2 2 2 1 2 1 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2
## [630] 2 1 2 2 1 1 1 2 1 1 2 2 2 1 2 2 2 2 2 1 2 1 1 1 2 1 2 1 2 2 2 2 1 2 2 2 1
## [667] 2 1 2 1 2 2 1 2 2 2 2 2 1 1 2 2 2 2 2 2 1 2 2 1 2 1 2 2 2 1 2 1 2 2 1 1 2
## [704] 2 1 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 1
## [741] 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 1 2 2 2 2 1 2 2 1 2 2 2 1
## [778] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2
## [815] 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
## [852] 2 2 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 2 2 2 1 1 2 1 1 2 1 2 1 1 2
## [889] 2 1 2 2 2 1 2 1 2 1 2 2 2 1 1 2 2 1 2 2 2 2 2 1 2 1 1 2 2 1 2 2 1 2 1 1 2
## [926] 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 2 2 2 1 2 2 2
## [963] 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 2 1 2 1 2 2 2 2 2 1 1 2 2 2 2 2 2 1 1 1 2 2
## [1000] 1 1 1 2 2 2 2 2 1 2 2 1 1 1 2 2 2 2 2 2 1 2 1 1 1 2 2 2 2 2 1 2 1 2 2 2 1
## [1037] 2 2 1 2 1 2 2 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2
## [1074] 2 2 1 1 1 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 2 2 2 2 2
## [1111] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 2 1 1 2 2 2 2 1 2 1
## [1148] 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 1 2 1 1 2 2 2 1
## [1185] 1 2 2 1 1 2 1 2 2 2 2 2 1 2 2 2 2 1 2 1 2 2 2 2 2 1 1 2 1 1 2 1 1 1 2 1 1
## [1222] 2 2 1 2 2 2 1 1 2 1 1 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 1 2 2
## [1259] 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1 2 1 2 1 2 2 2 1 2 2 2 1 2
## [1296] 2 2 2 2 2 1 2 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 2 1 2
## [1333] 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 1 2 2 2
## [1370] 2 1 2 2 2 2 1 1 2 2 1 1 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 1
## [1407] 2 1 2 2 1 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2
## [1444] 1 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2
## [1481] 1 2 1 2 2 2 2 2 1 2 1 2 2 2 2 1 2 2 1 2 2 2 2 1 2 2 1 2 2 1 2 1 1 2 1 2 1 2
## [1518] 2 2 2 1 2 2 1 1 2 1 2 2 2 1 1 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2
## [1555] 2 1 2 2 2 1 2 1 2 1 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2
## [1592] 2 2 2 2 1 2 1 2 2 2 1 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1
## [1629] 2 1 2 2 2 2 2 1 2 2 2 1 2 2 1 2 1 2 1 1 2 1 2 2 2 1 2 2 2 2 1 1 2 2 2 2 1
## [1666] 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 1 2 1 2 2 1 1 1 1 2 1 1 2 2 2 1 2 2 2 1 2 1
## [1703] 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 1 2 2 1 2 2 1 2 2 2 1 2 2 1 2 2 1 1
## [1740] 1 1 1 2 2 1 1 1 2 2 2 2 1 2 2 2 1 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2
## [1777] 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 2 2 2 1 1 2 2 2 1 2 2 2 2 1 2 2 2 2 1 2
## [1814] 2 2 1 2 2 2 2 1 2 1 1 2 2 1 2 2 2 2 2 2 1 2 1 1 1 1 2 2 2 2 2 1 2 2 2 1 2
## [1851] 1 2 2 1 2 1 2 1 2 2 2 2 2 2 1 1 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2
## [1888] 2 1 2 2 1 2 2 2 1 1 2 2 2 1 2 2 2 2 2 2 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2
## [1925] 2 2 1 2 2 2 1 1 1 2 2 1 1 2 2 1 2 2 1 1 2 2 2 1 1 2 2 2 2 1 2 2 1 2 2 2 2
## [1962] 1 2 2 1 2 2 1 2 1 1 2 1 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2
## [1999] 2 1
##
## $thresholds
## [1] 0.9250038 1.4790838 2.0592782

```

3. Estimate the regression function of the variable **d** on (**x1** , **x2**) on the grid **[1,2,..., 400]x[1,2,..., 400]** with a suitable smoothing parameter. You will use the following parameters: **eval.points=cbind(g1,g2)** and **nbins=0**, with **g1=1:400** and **g2=1:400**. Name **R** the matrix containing the estimated values of the regression function. This may take around one minute to run on your laptop. [1.5 pt]

```
g1=1:400
g2=1:400
R=sm.regression(cbind(x1,x2),d,nbins=0,eval.points=cbind(g1,g2),eval.grid=TRUE,method='cv')$estimate
```



4. Visualize the result obtained by applying the following commands. Which entry **E1**, **E2**, or **E3** will you indicate to your friend to allow his escape? [0.5 pt]

```
image(R,main='Freedom', xlab='Prison',yaxt="n",xaxt="n")
axis(1,at=c(0.22,0.53, 0.89), labels=c('E1','E2','E3'))
```

Freedom

