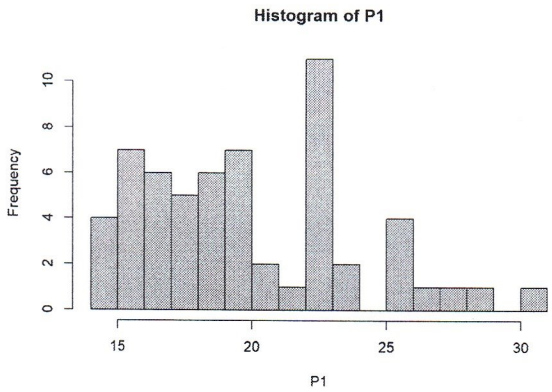
```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### Example of Univariate Predictive Intervals
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
X  <- read.table('parziali.txt')
P1 <- X$PI

# Let us focus on PI
hist(P1, breaks=15)
```

FULL CONFORMAL PREDICTION

Goal: build a prediction set (of a given probability) from the grade of the "primo parziale" of a new student




```
shapiro.test(P1)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  P1
## W = 0.94278, p-value = 0.00788
```

```
# Full Conformal
alpha          <- 0.1
x.obs          <- P1
x.new.grid     <- seq(min(x.obs) - 0.5*diff(range(P1)), max(x.obs) + 0.5*diff(range(P1)),
                  length = 1000)
p.value        <- numeric(length(x.new.grid))
random.number <- runif(1, 0, 33)
```

set of points for which we'll compute the **p-values**

```
NC <- function(z.aug, i){
  abs(z.aug[i] - mean(z.aug[-i]))     # sample mean as a predictor (i.e., T intervals)
  #abs(z.aug[i] - median(z.aug[-i]))   # a robust predictor (to deal with outliers)
  #abs(z.aug[i] - (mean(z.aug[-i])+10)) # a biased predictor
  #abs(z.aug[i] - 18)                  # a deterministic predictor
  #abs(z.aug[i] - random.number)       # a fully random predictor
}
```

← it receives the augmented sample and the index of the eliminated one

← "stupid" non-conformity measure

← even more stupid

← ...

still valid (not efficient)

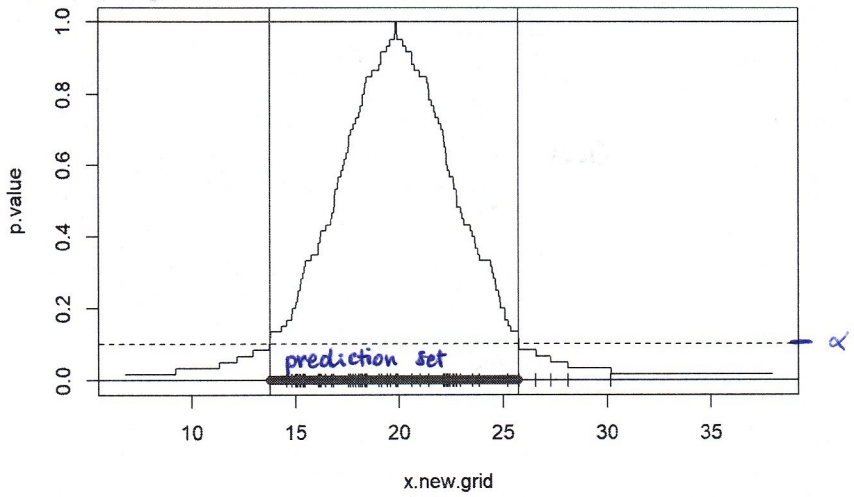
```
for(k in 1:length(x.new.grid)) {
  x.obs.aug <- c(x.obs, x.new.grid[k])
  scores    <- numeric(length(x.obs.aug))
  for (i in 1:length(x.obs.aug)) {
    scores[i] <- NC(x.obs.aug, i)
  }
  p.value[k] <- sum(scores >= scores[length(x.obs.aug)])/(length(x.obs.aug))
}
```

We move along the grid. For each point of the grid we build the augmented sample by concatenating the point of the grid with the sample. Then we compute the score for every point in the augmented sample. Once we have the scores we compute the p-value.

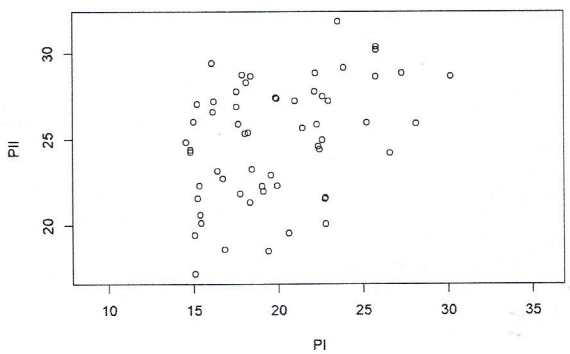
```
# Plot the p-values
plot(x.new.grid, p.value, type='l', ylim=c(0,1))
abline(h=c(0,1))
abline(h=alpha, col='red', lty=2)
points(x.obs, numeric(length(x.obs)), pch=3)

# Compute the Prediction Interval
PI.grid <- x.new.grid[which(p.value >= alpha)]
PI      <- c(min(PI.grid), max(PI.grid))
PI
```

```
## [1] 13.76683 25.74424
```

```
abline(v = PI, col='red')
points(PI.grid, numeric(length(PI.grid)), pch=16, col='red')
```




```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### Example of Multivariate Predictive Regions
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
X <- read.table('parziali.txt')

# Let us focus on (PI, PII)
plot(X, asp=1)
```



Goal: build a prediction region for predicting a new observation (new student) (1)

since dim = 2 we'll move in the cartesian plane instead of the real axes.

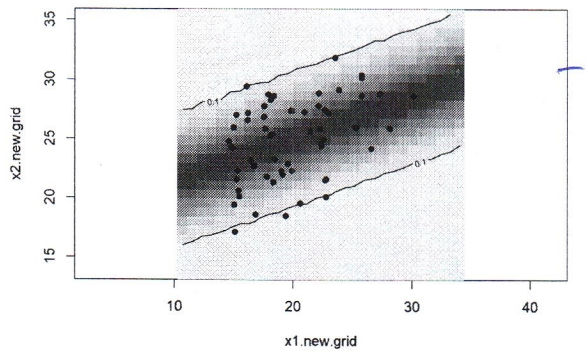
Goal: regression (2)

```
# Full Conformal
alpha       <- 0.1
x.obs       <- X
x1.new.grid <- seq(min(x.obs[,1]) - 0.25*diff(range(x.obs[,1])),
                   max(x.obs[,1]) + 0.25*diff(range(x.obs[,1])), length = 30)
x2.new.grid <- seq(min(x.obs[,2]) - 0.25*diff(range(x.obs[,2])),
                   max(x.obs[,2]) + 0.25*diff(range(x.obs[,2])), length = 30)
p.value     <- matrix(nrow = length(x1.new.grid), ncol = length(x2.new.grid))

NC <- function(z.aug, i){
  #sum((z.aug[i,] - colMeans(z.aug[-i,]))^2) # Euclidean distance
  #as.numeric( as.matrix(z.aug[i,] - colMeans(z.aug[-i,])) %*%
  #              solve(cov(z.aug[-i,])) %*%
  #              as.matrix(t(z.aug[i,] - colMeans(z.aug[-i,]))) ) # Mahalanobis Distance
  abs(z.aug[i,2]-sum(coefficients(lm(z.aug[-i,2]~ z.aug[-i,1]))*c(1, z.aug[i,1]))) # Regression
}

for(k in 1:length(x1.new.grid)) {
  for(h in 1:length(x2.new.grid)) {
    x.obs.aug <- rbind(x.obs, c(x1.new.grid[k],x2.new.grid[h]))
    scores    <- numeric(dim(x.obs.aug)[1])
    for (i in 1:dim(x.obs.aug)[1]) {
      scores[i] <- NC(x.obs.aug, i)
    }
  p.value[k,h] <- sum(scores >= scores[dim(x.obs.aug)[1]])/(dim(x.obs.aug)[1])
  print(c(k,h))
  }
}
```

(1) optimal if dataset are bivariate gaussian

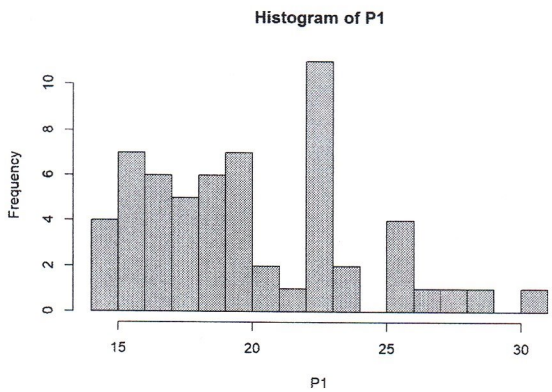
(2) we can use anything (even nonlinear)

```
# Plot the p-values and the prediction region
image(x1.new.grid, x2.new.grid, p.value, zlim=c(0,1), asp=1)
points(X, pch=16)
contour(x1.new.grid, x2.new.grid, p.value, levels = alpha, add=T)
```



prediction region shaped like a band

```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### Example of Univariate Predictive sets not based on discrepancy (e.g., KNN)
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
X  <- read.table('parziali.txt')
P1 <- X$PI

# Let us focus on PI
hist(P1, breaks=15)
```

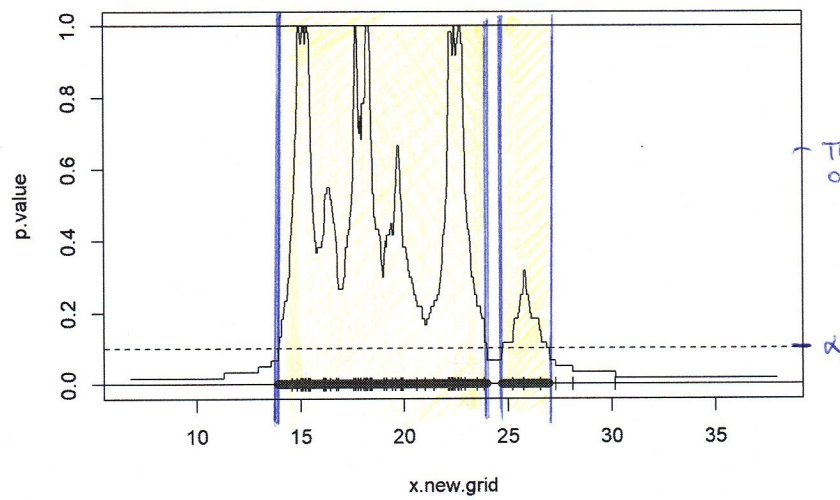



```
shapiro.test(P1)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  P1
## W = 0.94278, p-value = 0.00788
```

```
# Full Conformal
alpha      <- 0.1
x.obs      <- P1
x.new.grid <- seq(min(x.obs) - 0.5*diff(range(P1)),
                  max(x.obs) + 0.5*diff(range(P1)), length = 1000)
p.value    <- numeric(length(x.new.grid))
K          <- 5

NC <- function(z.aug, i){
  distances2 <- ( as.matrix(dist(z.aug))[i,-i] )^2
  mean(sort(distances2)[1:K]) # average linkage
  #min(sort(distances2)[1:K])  # single linkage
  #max(sort(distances2)[1:K])  # complete linkage
}

for(k in 1:length(x.new.grid)) {
  x.obs.aug <- c(x.obs, x.new.grid[k])
  scores    <- numeric(length(x.obs.aug))
  for (i in 1:length(x.obs.aug)) {
    scores[i] <- NC(x.obs.aug, i)
  }
  p.value[k] <- sum(scores >= scores[length(x.obs.aug)])/(length(x.obs.aug))
}

# Plot the p-values
plot(x.new.grid, p.value, type='l', ylim=c(0,1))
abline(h=c(0,1))
abline(h=alpha, col='red', lty=2)
points(x.obs, numeric(length(x.obs)), pch=3)

# Compute the Prediction Set
PI.grid <- x.new.grid[which(p.value >= alpha)]
points(PI.grid, numeric(length(PI.grid)), pch=16, col='red')
```

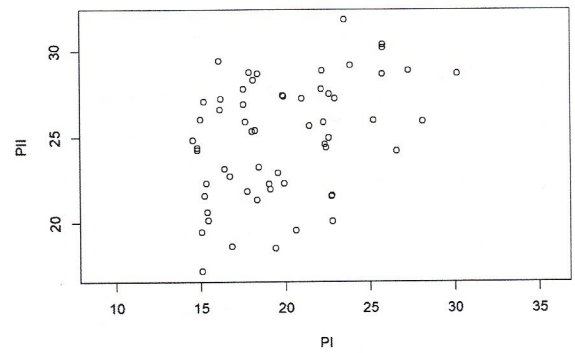
average squared distance to the 5 closest points (whatever linkage)



The prediction region is made of two intervals in this case

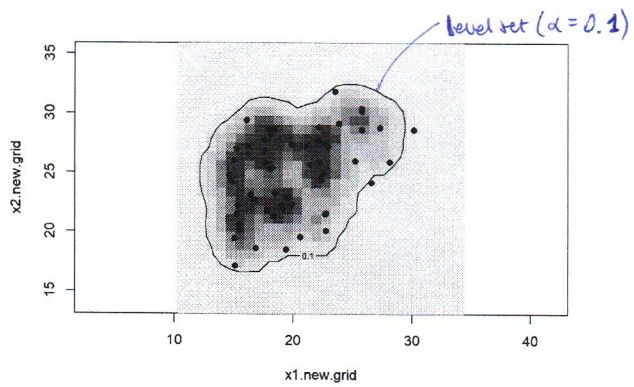
α

```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### Example of Multivariate Predictive sets not based on discrepancy (e.g., KNN)
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
X <- read.table('parziali.txt')

# Let us focus on (PI, PII)
plot(X, asp=1)
```



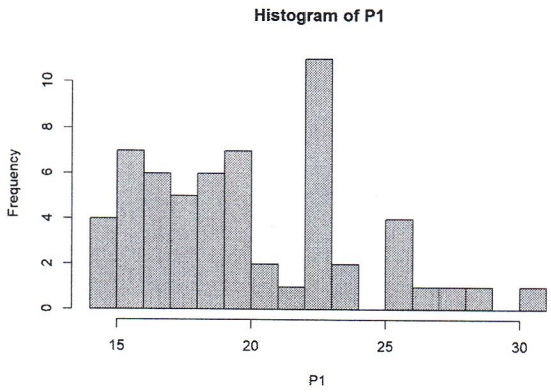
```
# Full Conformal
alpha       <- 0.1
x.obs       <- X
x1.new.grid <- seq(min(x.obs[,1]) - 0.25*diff(range(x.obs[,1])),
                   max(x.obs[,1]) + 0.25*diff(range(x.obs[,1])), length = 30)
x2.new.grid <- seq(min(x.obs[,2]) - 0.25*diff(range(x.obs[,2])),
                   max(x.obs[,2]) + 0.25*diff(range(x.obs[,2])), length = 30)
p.value     <- matrix(nrow = length(x1.new.grid), ncol = length(x2.new.grid))
K           <- 5

NC <- function(z.aug, i){
  distances2 <- ( as.matrix(dist(z.aug))[i,-i] )^2
  mean(sort(distances2)[1:K]) # average linkage
  #min(sort(distances2)[1:K])  # single linkage
  #max(sort(distances2)[1:K])  # complete linkage
}

for(k in 1:length(x1.new.grid)) {
  for(h in 1:length(x2.new.grid)) {
    x.obs.aug <- rbind(x.obs, c(x1.new.grid[k],x2.new.grid[h]))
    scores    <- numeric(dim(x.obs.aug)[1])
    for (i in 1:dim(x.obs.aug)[1]) {
      scores[i] <- NC(x.obs.aug, i)
    }
    p.value[k,h] <- sum(scores >= scores[dim(x.obs.aug)[1]])/(dim(x.obs.aug)[1])
    print(c(k,h))
  }
}
```

```
# Plot the p-values and the prediction region
image(x1.new.grid, x2.new.grid, p.value, zlim=c(0,1), asp=1)
points(X, pch=16)
contour(x1.new.grid, x2.new.grid, p.value, levels = alpha, add=T)
```




```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### Example of Univariate Predictive Intervals
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
X  <- read.table('parziali.txt')
P1 <- X$PI

# Let us focus on PI
hist(P1, breaks=15)
```

SPLIT CONFORMAL PREDICTION




```
shapiro.test(P1)
```

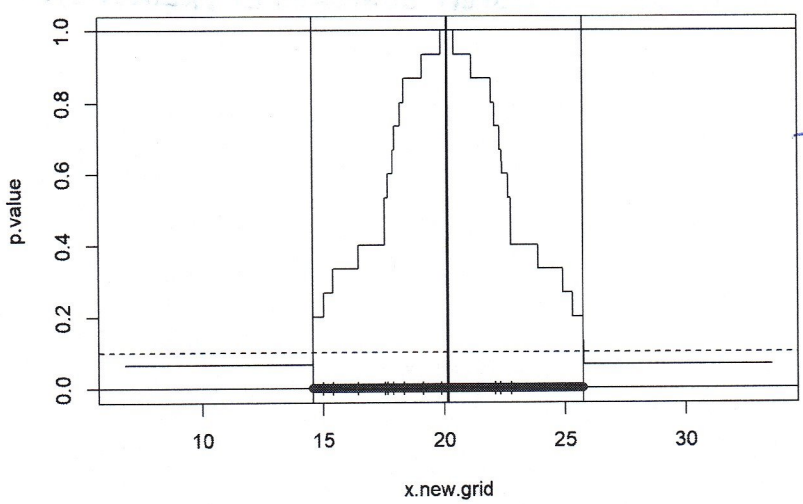
```
##
##  Shapiro-Wilk normality test
##
## data:  P1
## W = 0.94278, p-value = 0.00788
```

```
# Split Conformal
alpha          <- 0.1
n              <- length(P1)
training.prop <- 0.75
```
(we're splitting 25% and 75%.)

```
set.seed(240279)
#set.seed(240278)
training.id   <- sample(1:n, ceiling(n*training.prop), replace = F)
training.set  <- P1[training.id]   # Proper Training Set
x.obs         <- P1[-training.id]  # Calibration set
x.new.grid    <- seq(min(x.obs) - 0.5*diff(range(P1)),
                     max(x.obs) + 0.5*diff(range(P1)), length = 1000)
p.value       <- numeric(length(x.new.grid))
training.mean <- mean(training.set)

NC <- function(z.aug, i){
  abs(z.aug[i] - training.mean)
  # Sample mean of the training set as a predictor (i.e., T intervals)
}

for(k in 1:length(x.new.grid)) {
  x.obs.aug <- c(x.obs, x.new.grid[k])
  scores    <- numeric(length(x.obs.aug))
  for (i in 1:length(x.obs.aug)) {
    scores[i] <- NC(x.obs.aug, i)
  }
  p.value[k] <- sum(scores >= scores[length(x.obs.aug)])/(length(x.obs.aug))
}

# Plot the p-values
plot(x.new.grid, p.value, type='l', ylim=c(0,1))
abline(h=c(0,1))
abline(h=alpha, col='red', lty=2)
abline(v=mean(training.set), lwd=2)
points(x.obs, numeric(length(x.obs)), pch=3)

# Compute the Prediction Interval
PI.grid <- x.new.grid[which(p.value >= alpha)]
PI      <- c(min(PI.grid), max(PI.grid))
PI
```

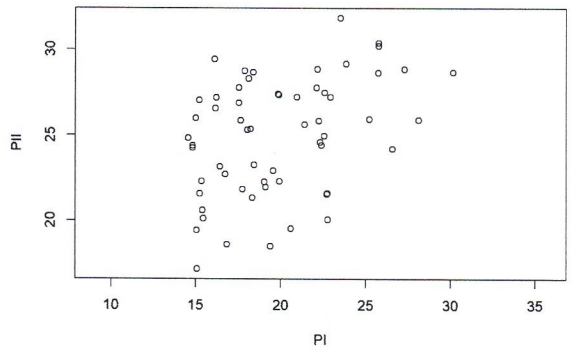
```
## [1] 14.57787 25.75213
```

```
abline(v = PI, col='red')
points(PI.grid, numeric(length(PI.grid)), pch=16, col='red')
```



— much more discretized

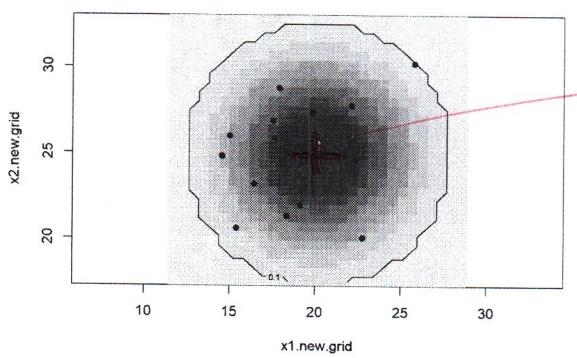
```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### Example of Multivariate Predictive Regions
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
X <- read.table('parziali.txt')

# Let us focus on (PI, PII)
plot(X, asp=1)
```




```
# Split Conformal
alpha          <- 0.1
n              <- dim(X)[1]
training.prop <- 0.75

set.seed(240279)
#set.seed(240278)
training.id   <- sample(1:n, ceiling(n*training.prop), replace = F)
training.set  <- X[training.id,]   # Proper Training Set
x.obs         <- X[-training.id,]  # Calibration set

x1.new.grid   <- seq(min(x.obs[,1]) - 0.25*diff(range(x.obs[,1])),
                     max(x.obs[,1]) + 0.25*diff(range(x.obs[,1])), length = 30)
x2.new.grid   <- seq(min(x.obs[,2]) - 0.25*diff(range(x.obs[,2])),
                     max(x.obs[,2]) + 0.25*diff(range(x.obs[,2])), length = 30)
p.value       <- matrix(nrow = length(x1.new.grid), ncol = length(x2.new.grid))
training.mean <- colMeans(training.set)

NC <- function(z.aug, i){
  sum((z.aug[i,] - training.mean)^2)
}

for(k in 1:length(x1.new.grid)) {
  for(h in 1:length(x2.new.grid)) {
    x.obs.aug <- rbind(x.obs, c(x1.new.grid[k],x2.new.grid[h]))
    scores    <- numeric(dim(x.obs.aug)[1])
    for (i in 1:dim(x.obs.aug)[1]) {
    scores[i] <- NC(x.obs.aug, i)
    }
  p.value[k,h] <- sum(scores >= scores[dim(x.obs.aug)[1]])/(dim(x.obs.aug)[1])
  print(c(k,h))
  }
}
```

← we use Euclidean but we can use anything

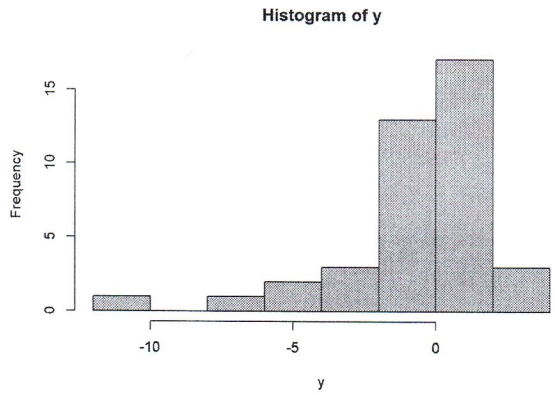
```
# Plot the p-values and the prediction region
image(x1.new.grid, x2.new.grid, p.value, zlim=c(0,1), asp=1)
points(x.obs, pch=16)
contour(x1.new.grid, x2.new.grid, p.value, levels = alpha, add=T)
points(training.mean[1], training.mean[2], pch=3, cex=4)
```



sample mean of the training set

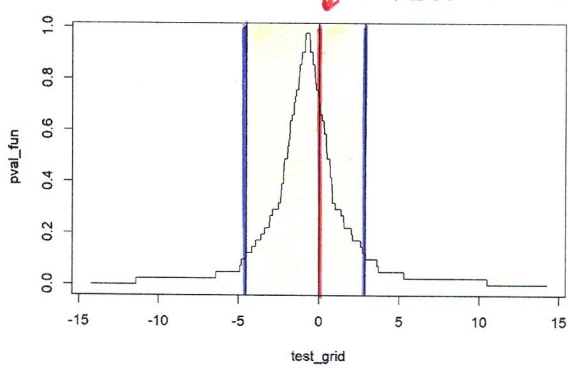
```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### Full Conformal
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
# Let's generate some data from a ill-behaved distribution
n = 40
y = rt(n,2)
hist(y)
```

the idea is to map the space where the response lives with a p-value function



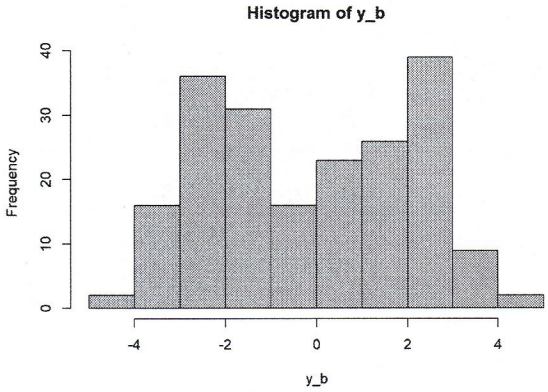

```
library(dbscan)

# We want to calculate the prediction set (in this case it's actually an interval)
# for a new obs drawn from F_y.
# Full Approach: I need to calculate the discrete p-value function over a discrete grid of points

grid_factor = 1.25
n_grid      = 1000
test_grid   = seq(-grid_factor*max(abs(y)),+grid_factor*max(abs(y)),length.out = n_grid)
pval_fun    = numeric(n_grid)

for(index in 1:n_grid){
  aug_y           = c(test_grid[index],y)
  mu              = mean(aug_y)
  ncm             = abs(mu - aug_y)
  pval_fun[index] = sum((ncm[-1]>=ncm[1]))/(n+1)
}

plot(test_grid,pval_fun,type='l')
abline(v=0,col='red')

alpha    = 0.1 # I want alpha to be 10%
index_in = pval_fun>alpha
#in this case the P-value function is unimodal, so I can just compute the range
PI       = range(test_grid[index_in])
abline(v=PI,col='blue')
```

theoretical mean of the distribution

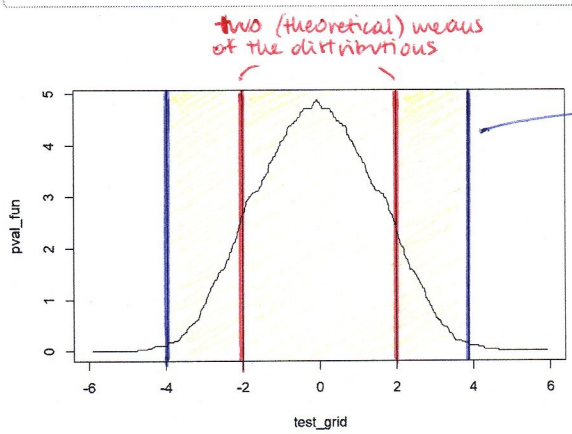


faster way

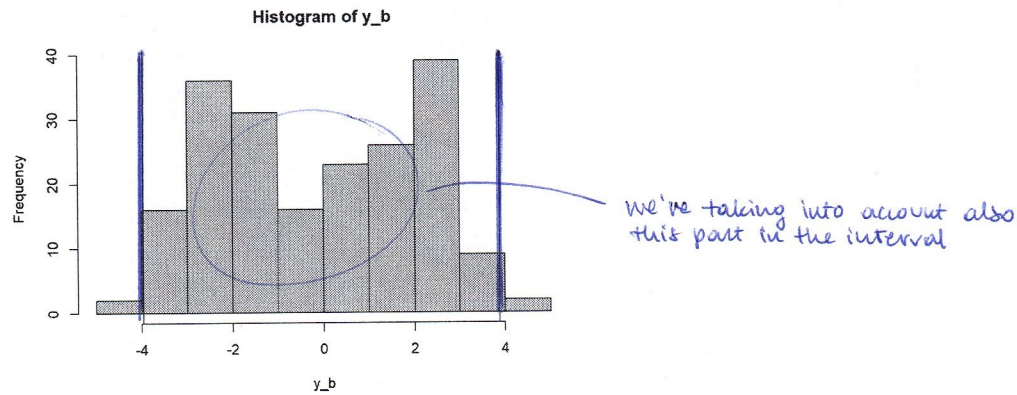
```
# I can of course think about using an apply structure to solve this
wrapper_full = function(grid_point){
    aug_y = c(grid_point,y)
    mu    = mean(aug_y)
    ncm   = abs(mu - aug_y)
    sum((ncm[-1]>=ncm[1]))/(n+1)
}

pval_fun = sapply(test_grid,wrapper_full)

# What about multimodal distributions?
n_b = 200
y_b = c(rnorm(n_b/2,mean = -2),rnorm(n_b/2,mean=2))
hist(y_b)
```



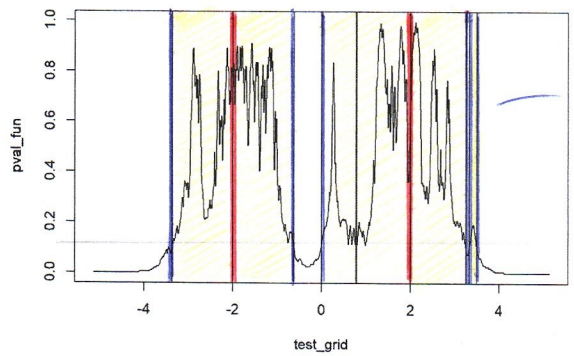

```
# What happens if I use the NCM as before?
wrapper_full = function(grid_point){
  aug_y = c(grid_point,y_b)
  mu    = mean(aug_y)
  ncm   = abs(mu - aug_y)
  sum((ncm[-1]>=ncm[1]))/(n+1)
}

test_grid = seq(-grid_factor*max(abs(y_b)),+grid_factor*max(abs(y_b)),length.out = n_grid)
pval_fun  = sapply(test_grid,wrapper_full)
plot(test_grid,pval_fun,type='l')
abline(v=c(-2,2),col='red')

index_in = pval_fun>alpha
# in this case the P-value function is unimodal, so I can just compute the range
PI       = range(test_grid[index_in])
abline(v=PI,col='blue')
```




```
hist(y_b)
abline(v=PI,col='blue')
```

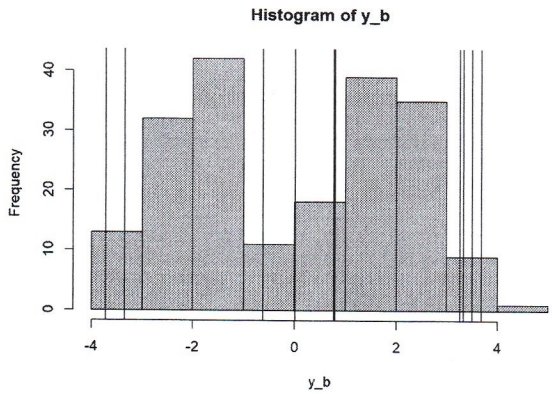


```
# Can I do better?

# Average euclidean distance from k-nn
pval_fun = numeric(n_grid)
k_s      = 0.2*n

wrapper_knn = function(grid_point){
  aug_y = c(grid_point,y_b)
  ncm   = kNNdist(matrix(aug_y),k_s)
  sum((ncm[-1]>=ncm[1]))/(n_b+1)
}

# This is gonna take a bit more, so
library(pbapply)
pval_fun = pbsapply(test_grid,wrapper_knn)
plot(test_grid,pval_fun,type='l')
abline(v=c(-2,2),col='red')
index_in = pval_fun>alpha
# Let's see when I switch from true to false:
breaks = test_grid[as.logical(c(0,abs(diff(index_in))))]

# In this case the P-value function is bimodal
abline(v=breaks,col='blue')
```



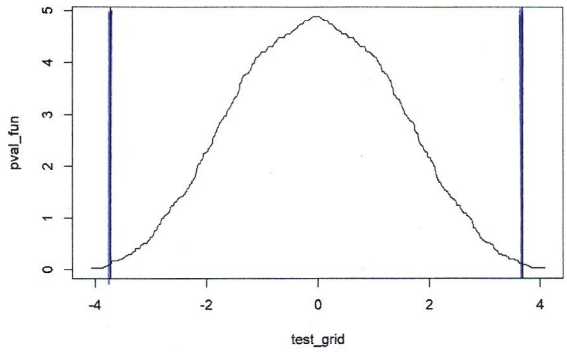
we get separate prediction intervals

```
hist(y_b)
abline(v=breaks,col='blue')
abline(v=PI,col='blue')
```




```
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
### Split Conformal
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
# No need of test grid, nor loops.
n         = 40
y         = rt(n,2)
test_grid = seq(-grid_factor*max(abs(y)),+grid_factor*max(abs(y)),length.out = n_grid)
pval_fun  = sapply(test_grid,wrapper_full)
plot(test_grid,pval_fun,type='l')
index_in = pval_fun>alpha
# In this case the P-value function is unimodal, so I can just compute the range
PI       = range(test_grid[index_in])
abline(v=PI,col='blue')
```

for comparison with full conformal

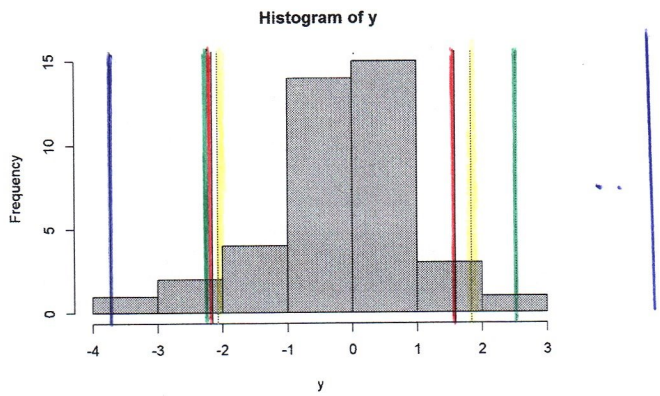



```
hist(y)
abline(v=PI,col='blue')

i1       = sample(1:n,n/2)
t_set    = y[i1]
c_set    = y[-i1]
mu       = mean(t_set)
ncm      = abs(c_set-mu)
ncm_sort = c(sort(ncm),Inf)
d        = sort(ncm)[ceiling((n/2 + 1)*(1-alpha))]
PI_split = c(mu-d,mu+d)
abline(v=PI_split,col='green')

# What happens if I run this multiple times?
i1       = sample(1:n,n/2)
t_set    = y[i1]
c_set    = y[-i1]
mu       = mean(t_set)
ncm      = abs(c_set-mu)
ncm_sort = c(sort(ncm),Inf)
d        = sort(ncm)[ceiling((n/2 + 1)*(1-alpha))]
PI_split = c(mu-d,mu+d)
abline(v=PI_split,col='orange')

i1       = sample(1:n,n/2)
t_set    = y[i1]
c_set    = y[-i1]
mu       = mean(t_set)
ncm      = abs(c_set-mu)
ncm_sort = c(sort(ncm),Inf)
d        = sort(ncm)[ceiling((n/2 + 1)*(1-alpha))]
PI_split = c(mu-d,mu+d)
abline(v=PI_split,col='red')
```

usually the splitting between training and calibration is 50,50

mean on the training

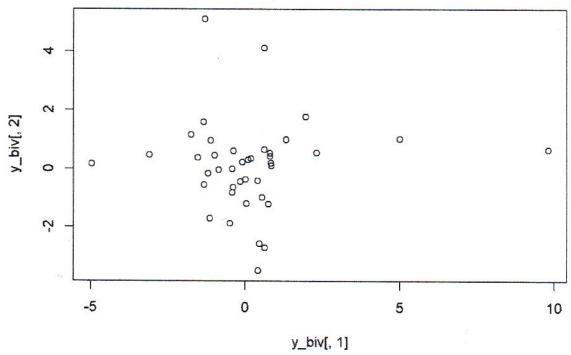
non conformity measure on the calibration (using the mean of the training)



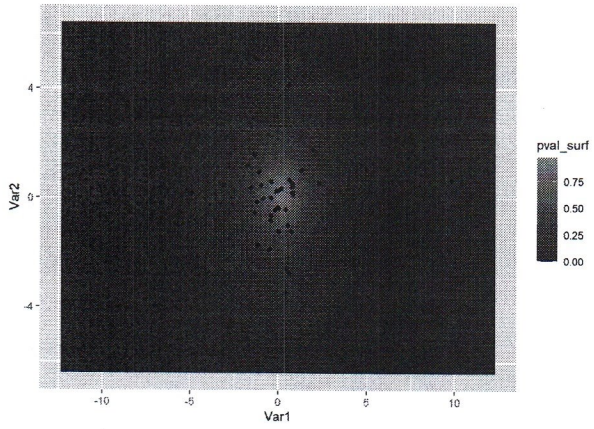
Notice that the Split Conformal prediction intervals are symmetric around the mean calculated on the training set

```
# They are, of course, all valid.
```

```
### --------------------------------------------------------------------------------
### --------------------------------------------------------------------------------
### Multivariate Conformal Prediction
### --------------------------------------------------------------------------------
### --------------------------------------------------------------------------------
# If we remember what we saw the very first tutorial session..
# Let's generate from a bivariate T
n           = 40
y_biv       = cbind(rt(n,2),rt(n,2))
n_grid      = 200
test_grid_x = seq(-grid_factor*max(abs(y_biv[,1])),+grid_factor*max(abs(y_biv[,1])),
                  length.out = n_grid)
test_grid_y = seq(-grid_factor*max(abs(y_biv[,2])),+grid_factor*max(abs(y_biv[,2])),
                  length.out = n_grid)
xy_surface  = expand.grid(test_grid_x,test_grid_y)
plot(y_biv[,1], y_biv[,2])

wrapper_multi_conf = function(test_point){
  newdata           = rbind(test_point,y_biv)
  depth_surface_vec = rowSums(t(t(newdata)-colMeans(newdata))^2)
  # In this case I am using the L^2 norm.
  sum(depth_surface_vec[-1]>=depth_surface_vec[1])/(n+1)
}

pval_surf=pbapply(xy_surface,1,wrapper_multi_conf)

# Let's plot this p-value surface:
data_plot = cbind(pval_surf,xy_surface)

library(ggplot2)
```

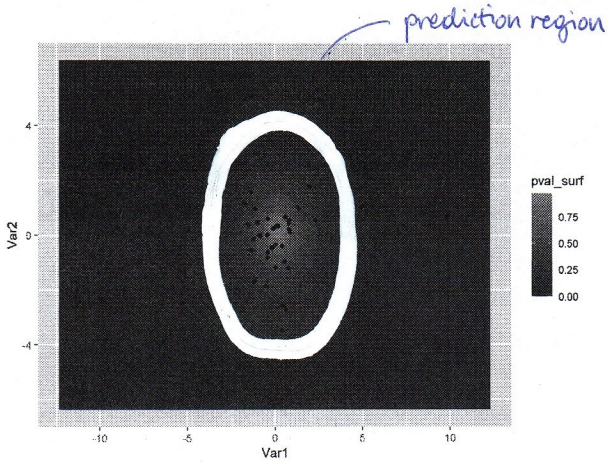



```
ggplot() + scale_color_continuous() +
  geom_tile(data=data_plot, aes(Var1, Var2, fill= pval_surf)) +
  geom_point(data=data.frame(y_biv), aes(X1,X2))
```

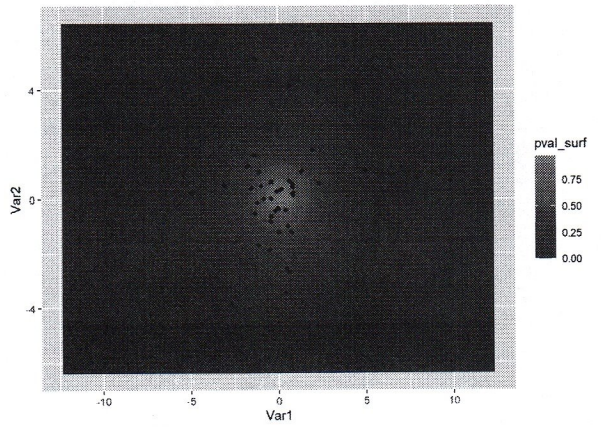



```
# Let's define the prediction set
p_set       = xy_surface[pval_surf>alpha,]
poly_points = p_set[chull(p_set),]

ggplot() +
  geom_tile(data=data_plot, aes(Var1, Var2, fill= pval_surf)) +
  geom_point(data=data.frame(y_biv), aes(X1,X2)) +
  geom_polygon(data=poly_points,aes(Var1,Var2),color='red',size=1,alpha=0.01)
```

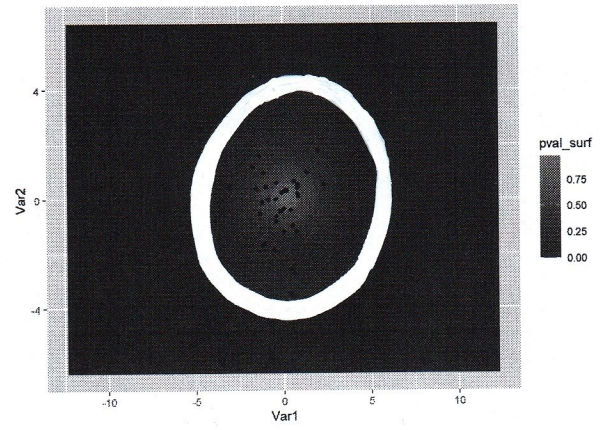
prediction region




```
# What happens if we change the NCM?

library(roahd)
wrapper_multi_conf_mah = function(test_point){
  newdata           = rbind(test_point,y_biv)
  depth_surface_vec = mahalanobis(newdata,colMeans(newdata),cov = cov(newdata))
  # In this case I am using the Mahalanobis distance
  sum(depth_surface_vec[-1]>=depth_surface_vec[1])/(n+1)
}

pval_surf = pbapply(xy_surface,1,wrapper_multi_conf_mah)

# Let's plot this p-value surface:
data_plot = cbind(pval_surf,xy_surface)

ggplot() +
  geom_tile(data=data_plot, aes(Var1, Var2, fill= pval_surf)) +
  geom_point(data=data.frame(y_biv), aes(X1,X2))
```

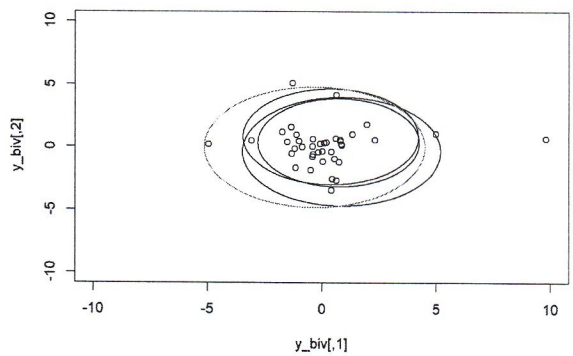



```
# Let's define the prediction set...
p_set       = xy_surface[pval_surf>alpha,]
poly_points = p_set[chull(p_set),]

ggplot() +
  geom_tile(data=data_plot, aes(Var1, Var2, fill= pval_surf)) +
  geom_point(data=data.frame(y_biv), aes(X1,X2)) +
  geom_polygon(data=poly_points,aes(Var1,Var2),color='red',size=1,alpha=0.01)
```



```r
# What about splitting? here is a bit more complex
i1       = sample(1:n,n/2)
t_set    = y_biv[i1,]
c_set    = y_biv[-i1,]
mu       = colMeans(t_set)
ncm      = sqrt(rowSums(c_set-mu)^2)
ncm_sort = c(sort(ncm),Inf)
d        = ncm_sort[ceiling((n/2 + 1)*(1-alpha))]

# What is the shape of the region for which d_euc(mu,point)<d? a circle!
# It's in general a problem to understand what is the shape of the level set of a given NCM
plot(y_biv,xlim = c(-10,10),ylim = c(-10,10))
angle_grid    = seq(0,(2*pi),length.out = 1000)
circle_points = data.frame(x=d*sin(angle_grid) + mu[1],y=d*cos(angle_grid) + mu[2])
polygon(circle_points)

# and, of course:
i1            = sample(1:n,n/2)
t_set         = y_biv[i1,]
c_set         = y_biv[-i1,]
mu            = colMeans(t_set)
ncm           = sqrt(rowSums(c_set-mu)^2)
ncm_sort      = c(sort(ncm),Inf)
d             = ncm_sort[ceiling((n/2 + 1)*(1-alpha))]
circle_points = data.frame(x=d*sin(angle_grid) + mu[1],y=d*cos(angle_grid) + mu[2])
polygon(circle_points,border='blue')

i1            = sample(1:n,n/2)
t_set         = y_biv[i1,]
c_set         = y_biv[-i1,]
mu            = colMeans(t_set)
ncm           = sqrt(rowSums(c_set-mu)^2)
ncm_sort      = c(sort(ncm),Inf)
d             = ncm_sort[ceiling((n/2 + 1)*(1-alpha))]
circle_points = data.frame(x=d*sin(angle_grid) + mu[1],y=d*cos(angle_grid) + mu[2])
polygon(circle_points,border='red')

i1            = sample(1:n,n/2)
t_set         = y_biv[i1,]
c_set         = y_biv[-i1,]
mu            = colMeans(t_set)
ncm           = sqrt(rowSums(c_set-mu)^2)
ncm_sort      = c(sort(ncm),Inf)
d             = ncm_sort[ceiling((n/2 + 1)*(1-alpha))]
circle_points = data.frame(x=d*sin(angle_grid) + mu[1],y=d*cos(angle_grid) + mu[2])
polygon(circle_points,border='green')
```

it's convenient to use the L^2 norm (for which the contours are circles) or the L^∞ (= max) norm (for which the contours are squares)




```r
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
### How to use it for "actual" purposes?
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
# ConformalInference package: not on CRAN, needs to be installed from github and
# since we're gonna use custom functions (made by me :P)
# We're gonna install it from my github

#devtools::install_github(repo="matteo-fontana/conformal", subdir="conformalInference")
library(conformalInference)
# Let's load our usual data
# I will now generate predictions for model of increasing complexity
# Let's start with easy
load('nlr_data.rda')
attach(Prestige)
```

```
## The following object is masked from package:datasets:
##
##     women
```

Let's see what happens with models of increasing complexity:

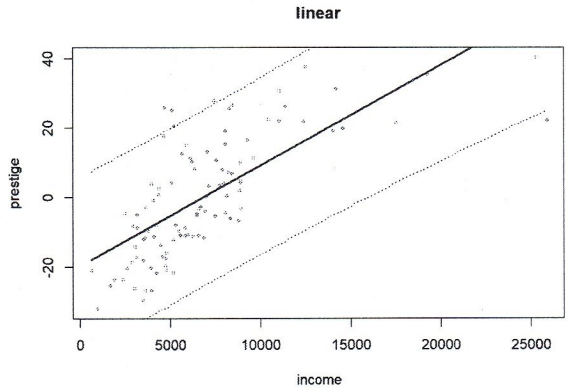
1.

```
model_poly  = lm(prestige ~ income)
income.grid = seq(range(income)[1],range(income)[2],by=100)
preds       = predict(model_poly,list(income=income.grid),se=T)
plot(income ,prestige ,xlim=range(income.grid) ,cex =.5, col =" darkgrey ",main='linear')
lines(income.grid,preds$fit ,lwd =2, col =" blue")

# How to generate conformal predictions?
lm_train   = lm.funs(intercept = T)$train.fun
lm_predict = lm.funs(intercept = T)$predict.fun

c_preds = conformal.pred(income,prestige,income.grid,alpha=0.05,verbose=T,
                         train.fun = lm_train,predict.fun = lm_predict,num.grid.pts = 200)
```

(design matrix, response, grid of points for which we want to run a prediction, alpha, verbose=T says at what point we are (it prints it), lm_train, lm_predict)

FULL

```
lines(income.grid,c_preds$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds$up,c_preds$lo) ,lwd =1, col =" blue",lty =3)
```

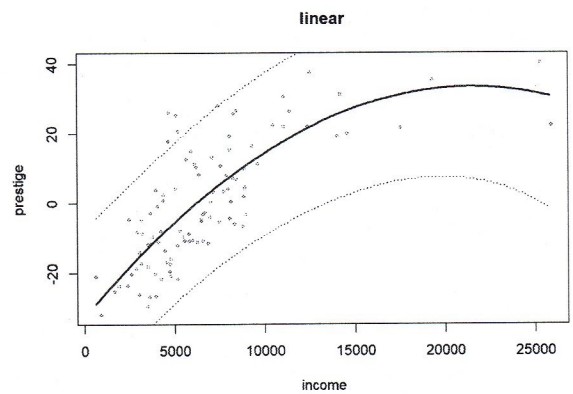


2.

```
# Let's go a bit more complex:
model_poly  = lm(prestige ~ poly(income,degree=2))
income.grid = seq(range(income)[1],range(income)[2],by=100)
preds       = predict(model_poly,list(income=income.grid),se=T)
plot(income ,prestige ,xlim=range(income.grid) ,cex =.5, col =" darkgrey ",main='linear')
lines(income.grid,preds$fit ,lwd =2, col =" blue")

design_matrix = matrix(poly(income,degree=2),ncol=2)
pred_grid = matrix(poly(income.grid,degree=2,coefs = attr(poly(income,degree=2),"coefs") ),ncol=2)

c_preds = conformal.pred(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                         train.fun = lm_train,predict.fun = lm_predict,num.grid.pts = 200)
```

FULL

```
lines(income.grid,c_preds$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds$up,c_preds$lo) ,lwd =1, col =" blue",lty =3)
```




```
# How to it in a split fashion?
plot(income ,prestige ,xlim=range(income.grid) ,cex =.5, col =" darkgrey ",main='linear')
lines(income.grid,preds$fit ,lwd =2, col =" blue")

c_preds_split = conformal.pred.split(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                                     train.fun = lm_train,predict.fun = lm_predict)
```

SPLIT

```
## Splitting data into parts of size 51 and 51 ...
## Training on first part ...
## Computing residuals and quantiles on second part ...
```

```
lines(income.grid,c_preds_split$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds_split$up,c_preds_split$lo) ,lwd =1, col =" red",lty =3)

c_preds_split = conformal.pred.split(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                                     train.fun = lm_train,predict.fun = lm_predict)
```

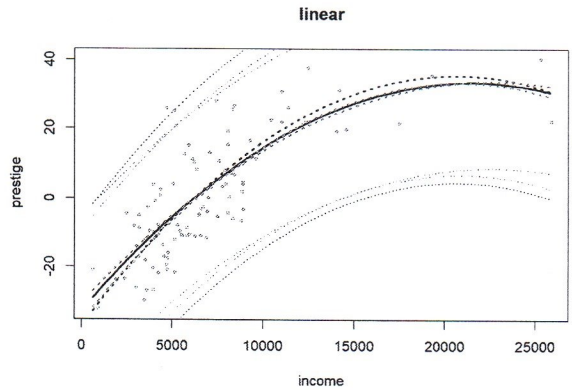
2nd round of split conformal

```
## Splitting data into parts of size 51 and 51 ...
## Training on first part ...
## Computing residuals and quantiles on second part ...
```

```
lines(income.grid,c_preds_split$pred ,lwd =2, col ="green",lty=3)
matlines(income.grid ,cbind(c_preds_split$up,c_preds_split$lo) ,lwd =1, col =" green",lty =3)

c_preds_split = conformal.pred.split(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                                     train.fun = lm_train,predict.fun = lm_predict)
```

```
## Splitting data into parts of size 51 and 51 ...
## Training on first part ...
## Computing residuals and quantiles on second part ...
```

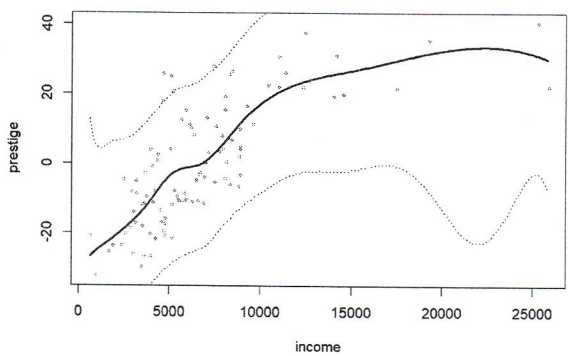
```
lines(income.grid,c_preds_split$pred ,lwd =2, col ="orange",lty=3)
matlines(income.grid ,cbind(c_preds_split$up,c_preds_split$lo) ,lwd =1, col =" orange",lty =3)
```



3.

```
# As Long as I am using the Lm framework, everything tends to be very easy
library(splines)
br         = c(quantile(income,probs = c(0.2,0.4,0.6,0.8)),15000)
model_cut = lm(prestige ~ bs(income, degree=3,knots=br))
preds      = predict(model_cut,list(income=income.grid),se=T)
plot(income ,prestige ,xlim=range(income.grid) ,cex =.5, col =" darkgrey " )
lines(income.grid,preds$fit ,lwd =2, col =" blue")

design_matrix = bs(income, degree=3,knots=br)
pred_grid      = matrix(bs(income.grid,degree=3,knots=br),nrow=length(income.grid))

c_preds = conformal.pred(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                         train.fun = lm_train,predict.fun = lm_predict,num.grid.pts = 200)
```

```
lines(income.grid,c_preds$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds$up,c_preds$lo) ,lwd =1, col =" blue",lty =3)
```



4.

```
# What if we cannot use the lm framework? we need to come up with something different
fit = smooth.spline(income,prestige,cv=T)
```

```
## Warning in smooth.spline(income, prestige, cv = T): cross-validation with non-
## unique 'x' values seems doubtful
```

```
plot(income ,prestige,cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
opt = fit$df
opt
```

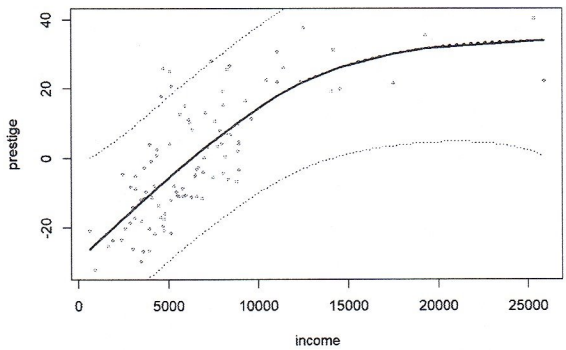
```
## [1] 3.720986
```

```
fit$lambda
```

```
## [1] 0.01474169
```

```
# a procedure like this takes also into account the choice of the correct smoothing value
train_ss = function(x,y,out=NULL){
  smooth.spline(x,y,df=opt)
}

predict_ss = function(obj, new_x){
  predict(obj,new_x)$y
}

c_preds = conformal.pred(income,prestige,income.grid,alpha=0.05,verbose=T,
                         train.fun = train_ss ,predict.fun = predict_ss,num.grid.pts = 200)
```

```
lines(income.grid,c_preds$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds$up,c_preds$lo) ,lwd =1, col =" blue",lty =3)
```



5.

```
# Let's try to do it in a more complex case: GAMS
library(mgcv)
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-33. For overview type 'help("mgcv-package")'.
```

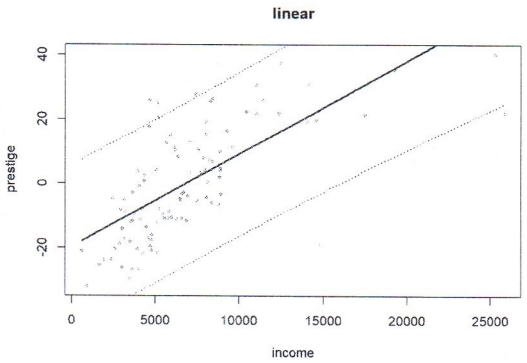
```
# I provide a VERY quick, and VERY dirty solution: it can be done of course in a way
# better and more general fashion:
model_gam      = gam(prestige ~ s(education,bs='cr') + s(income,bs='cr'))
education.grid = seq(range(education)[1],range(education)[2],length.out = 100)
income.grid    = seq(range(income)[1],range(income)[2],length.out = 100)
grid           = expand.grid(education.grid,income.grid)
names(grid)    = c('education','income')
pred           = predict(model_gam,newdata=grid)

library(rgl)
options(rgl.useNULL = TRUE)
persp3d(education.grid,income.grid,pred,col='yellow')
points3d(education,income,prestige,col='black',size=5)

train_gam = function(x,y,out=NULL){
  colnames(x) = c('var1','var2')
  train_data  = data.frame(y,x)
  model_gam   = gam(y ~ s(var1,bs='cr') + s(var2,bs='cr'),data=train_data)
}

predict_gam = function(obj, new_x){
  new_x           = data.frame(new_x)
  colnames(new_x) = c('var1','var2')
  predict.gam(obj,new_x)
}

c_preds = conformal.pred(cbind(education,income),prestige,c(median(education),median(income)),
                         alpha=0.05,verbose=T,train.fun = train_gam ,
                         predict.fun = predict_gam,num.grid.pts = 200)
```

```
c_preds
```

```
#############################################################################################
###                                  CONFORMAL PREDICTION                                 ###
###                               Library(conformalInference)                             ###
#############################################################################################
#devtools::install_github(repo="matteo-fontana/conformal", subdir="conformalInference")
library(conformalInference)

load('nlr_data.rda')
attach(Prestige)
```

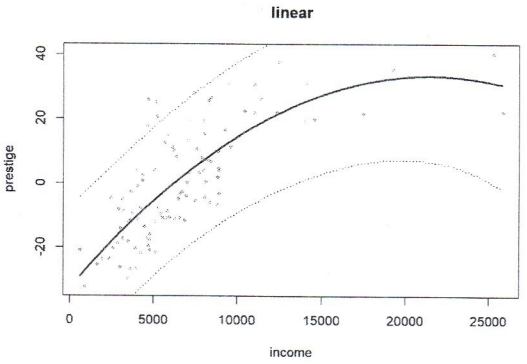
```
### ---------------------------------------------------------------------------------------
### 1.
### ---------------------------------------------------------------------------------------
model_poly  = lm(prestige ~ income)
income.grid = seq(range(income)[1],range(income)[2],by=100)
preds       = predict(model_poly,list(income=income.grid),se=T)
plot(income ,prestige ,xlim=range(income.grid) ,cex =.5, col =" darkgrey ",main='linear')
lines(income.grid,preds$fit ,lwd =2, col =" blue")

# How to generate conformal predictions?
lm_train   = lm.funs(intercept = T)$train.fun
lm_predict = lm.funs(intercept = T)$predict.fun

c_preds = conformal.pred(income,prestige,income.grid,alpha=0.05,verbose=T,
                         train.fun = lm_train,predict.fun = lm_predict,num.grid.pts = 200)
```

```
lines(income.grid,c_preds$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds$up,c_preds$lo) ,lwd =1, col =" blue",lty =3)
```




```
### ---------------------------------------------------------------------------------------
### 2.
### ---------------------------------------------------------------------------------------
model_poly  = lm(prestige ~ poly(income,degree=2))
income.grid = seq(range(income)[1],range(income)[2],by=100)
preds       = predict(model_poly,list(income=income.grid),se=T)
plot(income ,prestige ,xlim=range(income.grid) ,cex =.5, col =" darkgrey ",main='linear')
lines(income.grid,preds$fit ,lwd =2, col =" blue")

design_matrix = matrix(poly(income,degree=2),ncol=2)
pred_grid = matrix(poly(income.grid,degree=2,coefs = attr(poly(income,degree=2),"coefs") ),ncol=2)

c_preds = conformal.pred(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                         train.fun = lm_train,predict.fun = lm_predict,num.grid.pts = 200)
```

```
lines(income.grid,c_preds$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds$up,c_preds$lo) ,lwd =1, col =" blue",lty =3)
```

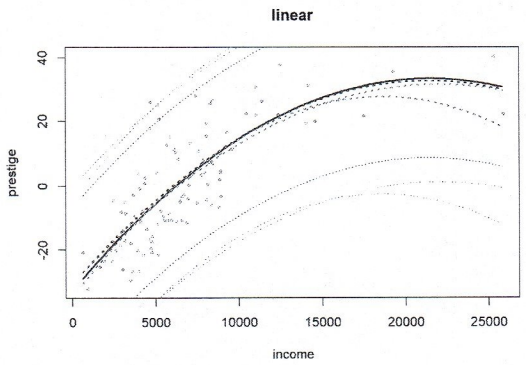



```
# How to it in a split fashion?
plot(income ,prestige ,xlim=range(income.grid) ,cex =.5, col =" darkgrey ",main='linear')
lines(income.grid,preds$fit ,lwd =2, col =" blue")

c_preds_split = conformal.pred.split(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                                     train.fun = lm_train,predict.fun = lm_predict)
```

```
lines(income.grid,c_preds_split$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds_split$up,c_preds_split$lo) ,lwd =1, col =" red",lty =3)

c_preds_split = conformal.pred.split(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                                     train.fun = lm_train,predict.fun = lm_predict)
```

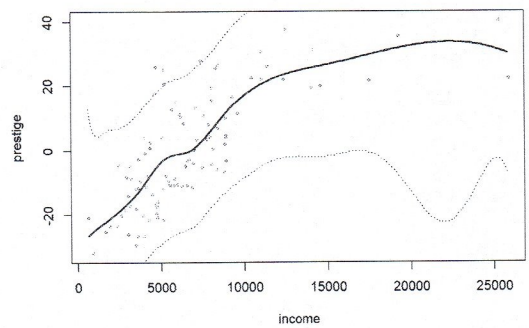
```
lines(income.grid,c_preds_split$pred ,lwd =2, col ="green",lty=3)
matlines(income.grid ,cbind(c_preds_split$up,c_preds_split$lo) ,lwd =1, col =" green",lty =3)

c_preds_split = conformal.pred.split(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                                     train.fun = lm_train,predict.fun = lm_predict)
```

```
lines(income.grid,c_preds_split$pred ,lwd =2, col ="orange",lty=3)
matlines(income.grid ,cbind(c_preds_split$up,c_preds_split$lo) ,lwd =1, col =" orange",lty =3)
```




```
### ------------------------------------------------------------------------------------
### 3.
### ------------------------------------------------------------------------------------
library(splines)
br         = c(quantile(income,probs = c(0.2,0.4,0.6,0.8)),15000)
model_cut = lm(prestige ~ bs(income, degree=3,knots=br))
preds      = predict(model_cut,list(income=income.grid),se=T)
plot(income ,prestige ,xlim=range(income.grid) ,cex =.5, col =" darkgrey " )
lines(income.grid,preds$fit ,lwd =2, col =" blue")

design_matrix = bs(income, degree=3,knots=br)
pred_grid     = matrix(bs(income.grid,degree=3,knots=br),nrow=length(income.grid))

c_preds = conformal.pred(design_matrix,prestige,pred_grid,alpha=0.05,verbose=T,
                         train.fun = lm_train,predict.fun = lm_predict,num.grid.pts = 200)
```

```
lines(income.grid,c_preds$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds$up,c_preds$lo) ,lwd =1, col =" blue",lty =3)
```




```
### ------------------------------------------------------------------------------------
### 4.
### ------------------------------------------------------------------------------------
fit = smooth.spline(income,prestige,cv=T)
```

```
plot(income ,prestige,cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
opt = fit$df
opt
```

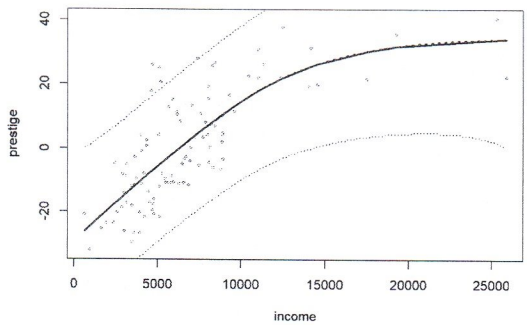
```
## [1] 3.720986
```

```
fit$lambda
```

```
## [1] 0.01474169
```

```
# a procedure like this takes also into account the choice of the correct smoothing value
train_ss = function(x,y,out=NULL){
  smooth.spline(x,y,df=opt)
}

predict_ss = function(obj, new_x){
  predict(obj,new_x)$y
}

c_preds = conformal.pred(income,prestige,income.grid,alpha=0.05,verbose=T,
                         train.fun = train_ss ,predict.fun = predict_ss,num.grid.pts = 200)
```

```
lines(income.grid,c_preds$pred ,lwd =2, col ="red",lty=3)
matlines(income.grid ,cbind(c_preds$up,c_preds$lo) ,lwd =1, col =" blue",lty =3)

### ------------------------------------------------------------------------------------
### 5.
### ------------------------------------------------------------------------------------
library(mgcv)
```




```
model_gam      = gam(prestige ~ s(education,bs='cr') + s(income,bs='cr'))
education.grid = seq(range(education)[1],range(education)[2],length.out = 100)
income.grid    = seq(range(income)[1],range(income)[2],length.out = 100)
grid           = expand.grid(education.grid,income.grid)
names(grid)    = c('education','income')
pred           = predict(model_gam,newdata=grid)

library(rgl)
options(rgl.useNULL = TRUE)
persp3d(education.grid,income.grid,pred,col='yellow')
points3d(education,income,prestige,col='black',size=5)

train_gam = function(x,y,out=NULL){
  colnames(x) = c('var1','var2')
  train_data  = data.frame(y,x)
  model_gam   = gam(y ~ s(var1,bs='cr') + s(var2,bs='cr'),data=train_data)
}

predict_gam = function(obj, new_x){
  new_x           = data.frame(new_x)
  colnames(new_x) = c('var1','var2')
  predict.gam(obj,new_x)
}

c_preds = conformal.pred(cbind(education,income),prestige,c(median(education),median(income)),
                         alpha=0.05,verbose=T,train.fun = train_gam ,
                         predict.fun = predict_gam,num.grid.pts = 200)
```

```
c_preds
```

```
## $pred
##          [,1]
## [1,] -2.34859
##
## $lo
##           [,1]
## [1,] -16.48137
##
## $up
##          [,1]
## [1,] 11.91729
##
## $fit
##               [,1]
##   [1,]  19.2294887
##   [2,]  23.2945943
##   [3,]  14.9380828
##   [4,]   ..
##  [99,] -15.1073362
## [100,] -12.6521835
## [101,]  -3.6092008
## [102,] -15.8490737
```

```
c_preds = conformal.pred.split(
  cbind(education,income),prestige,c(median(education),median(income)),
  alpha=0.05,verbose=T,train.fun = train_gam ,predict.fun = predict_gam)
```

```
c_preds
```

```
## $pred
##            [,1]
## [1,] -0.2195247
##
## $lo
##           [,1]
## [1,] -16.63503
##
## $up
##          [,1]
## [1,] 16.19598
##
## $fit
##                [,1]
##   [1,]  16.413759884
##   [2,]  18.587928551
##   [3,]  13.213091128
##   [4,]   ..
##  [99,] -12.925732037
## [100,] -10.063741285
## [101,]  -1.708097470
## [102,] -13.970819156
##
## $split
##  [1]  21  81  66  61  15  29   3  69  30  94  84  82  47  63  59  92   5  90  50
## [20]  11  74  79  39  53  42   9  51  13   6  85  58  17  91  86  60  36  10  45
## [39]  52  72  70  40  14  77  35   7  34  49  20  78 102
```

```
## $pred
##          [,1]
## [1,] -2.34859
##
## $lo
##           [,1]
## [1,] -16.48137
##
## $up
##          [,1]
## [1,] 11.91729
##
## $fit
##               [,1]
##   [1,]  19.2294887
##   [2,]  23.2945943
##   [3,]  14.9380828
##         ..
##  [99,] -15.1073362
## [100,] -12.6521835
## [101,]  -3.6092008
## [102,] -15.8490737
```

```
c_preds = conformal.pred.split(
  cbind(education,income),prestige,c(median(education),median(income)),
  alpha=0.05,verbose=T,train.fun = train_gam ,predict.fun = predict_gam)
```

```
c_preds
```

```
## $pred
##           [,1]
## [1,] -2.113719
##
## $lo
##           [,1]
## [1,] -17.66071
##
## $up
##          [,1]
## [1,] 13.43327
##
## $fit
##               [,1]
##   [1,]  16.1176900
##   [2,]  23.6753947
##   [3,]  13.1021437
##         ..
##  [99,] -15.9941387
## [100,] -13.3872816
## [101,]  -3.8187425
## [102,] -17.5079451
##
## $split
##  [1]  24  23  50  69 100  54  26  97  55  17  96 101  76  85  80  20  81  60  31
## [20]  84  40  43  64  42  38  25  65  89  19  34  22  66   4  93  53  68   2   7
## [39]   5  13  47  78  51  52  49  67  74   6   9  32  41
```