# Numerical Calibration of the Lasso

ENS Advanced Math, Non parametrics

Franck Picard, Fall 2020

## Preliminaries

Groups of students are asked to send an `R markdown` report generated via `R studio` to franck.picard@ens-lyon.fr at the end of the tutorial. You will need `Rstudio`, LaTeX and packages for markdown:

```
library(knitr)
library(rmarkdown)
library(graphics) ##Necessary for plots
library(pracma)
library(glmnet)
library(rgl)
```

This report should answer the questions by commentaries and codes generating appropriate graphical outputs. A cheat sheet of the markdown syntax can be found here.

## 1 Regression model

This project aims at studying the empirical properties of the LASSO based on simulated data. The statistical framework is the linear regression model, such that

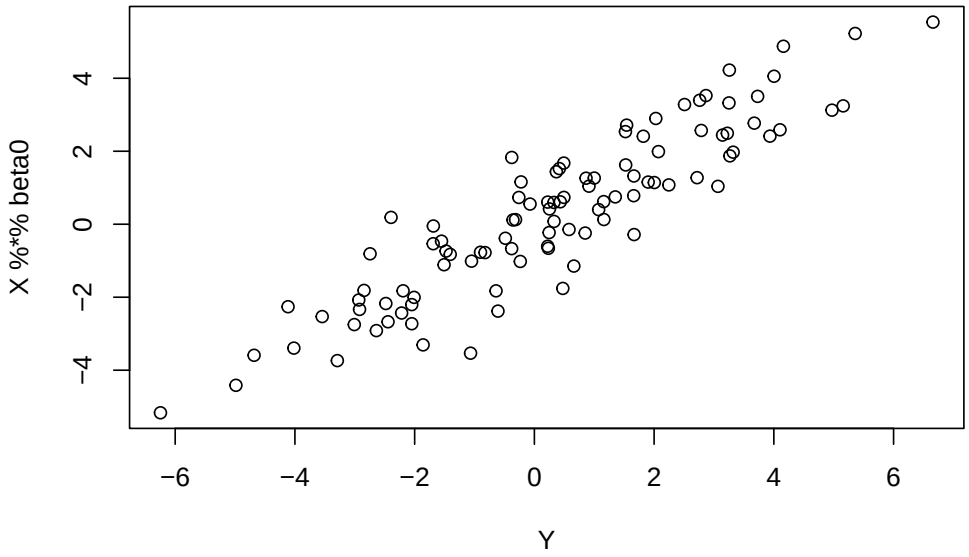
$$Y_i = x_i^T \beta^* + \varepsilon_i, \ \varepsilon_i \sim \mathcal{N}(0, \sigma^2),$$

with $Y$ a vector in $\mathbb{R}^n$, and $\beta^*$ a vector in $\mathbb{R}^p$ with $p_0$ non-null elements. In the following $J_0 = \{j \in \{1, ..., p\}, \beta_j^* \neq 0\}$. For simplification, we will consider that there is only one distinct non null value in $\beta^*$: $\beta^* = \beta_0^* \times (1, ..., 1, 0, ...0)$.

## 2 Simulation of observations

```
n       = 100
p       = 10
p0      = 5
sigma   = 1
sigmaX  = 1
b0      = 1
beta0   = c(rep(b0,p0),rep(0,p-p0))
```

```r
X       = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
Y       = X%*%beta0 + rnorm(n,0,sigma)

plot(Y,X%*%beta0)
```




## 3 Lasso and the `glmnet` R-package

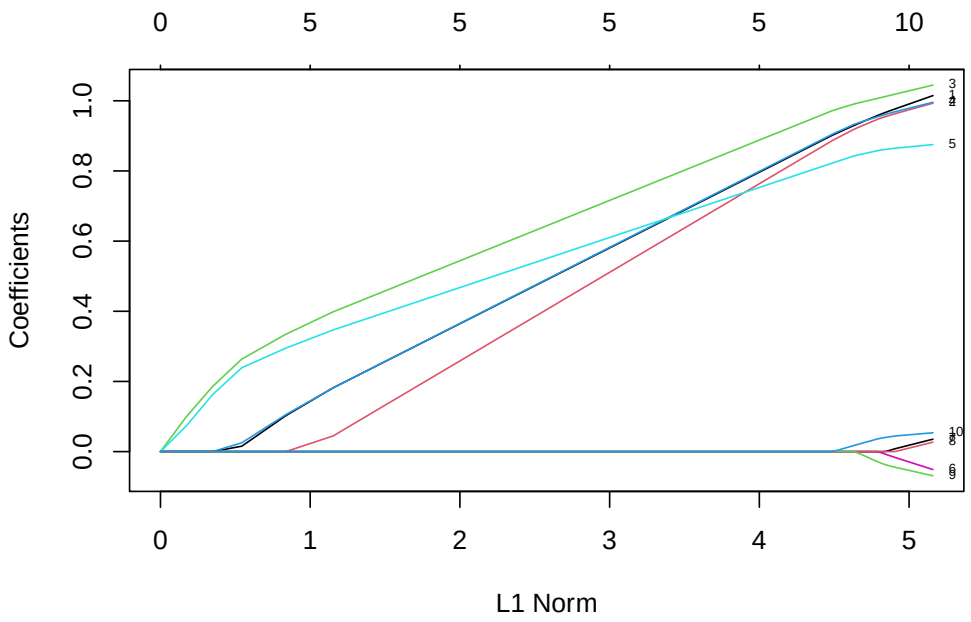
In practice, model parameters are estimated using the `glmnet` R-package to compute the lasso estimator

$$\widehat{\beta}_\lambda = \min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda \left[ (1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1 \right],$$

with $\alpha = 1$ for the Lasso, $\alpha = 0$ for Ridge Regression and $\alpha \in ]0,1[$ for Elastic Net. In a first step you are invited to check the online documentation of the package that is very complete. Then the purpose of calibration is to determine the value of the hyperparameter $\lambda$ based on the observations.

```r
library(glmnet)
n       = 100
p       = 10
p0      = 5
sigma   = 1
sigmaX  = 1
b0      = 1
beta0   = c(rep(b0,p0),rep(0,p-p0))
X       = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
```

```
Y        = X%*%beta0 + rnorm(n,0,sigma)

fit = glmnet(X, Y)
plot(fit,label=TRUE)
```




```
names(fit)
```

```
##  [1] "a0"        "beta"      "df"        "dim"       "lambda"    "dev.ratio"
##  [7] "nulldev"   "npasses"   "jerr"      "offset"    "call"      "nobs"
```

```
print(fit)
```

```
##
## Call:  glmnet(x = X, y = Y)
##
##    Df  %Dev  Lambda
## 1   0  0.00 1.36200
## 2   2  6.90 1.24100
## 3   2 13.28 1.13000
## 4   4 19.69 1.03000
## 5   4 28.44 0.93850
## 6   5 36.93 0.85510
## 7   5 45.14 0.77910
## 8   5 51.96 0.70990
## 9   5 57.62 0.64690
## 10  5 62.32 0.58940
## 11  5 66.22 0.53700
## 12  5 69.46 0.48930
```

```
## 13  5 72.15 0.44580
## 14  5 74.38 0.40620
## 15  5 76.24 0.37020
## 16  5 77.77 0.33730
## 17  5 79.05 0.30730
## 18  5 80.11 0.28000
## 19  5 80.99 0.25510
## 20  5 81.72 0.23250
## 21  5 82.33 0.21180
## 22  5 82.83 0.19300
## 23  5 83.25 0.17590
## 24  5 83.60 0.16020
## 25  5 83.89 0.14600
## 26  5 84.13 0.13300
## 27  5 84.33 0.12120
## 28  5 84.49 0.11040
## 29  5 84.63 0.10060
## 30  6 84.75 0.09169
## 31  6 84.85 0.08354
## 32  6 84.93 0.07612
## 33  6 85.00 0.06936
## 34  7 85.08 0.06320
## 35  7 85.15 0.05758
## 36  7 85.20 0.05247
## 37  7 85.25 0.04781
## 38  7 85.29 0.04356
## 39  8 85.33 0.03969
## 40  9 85.36 0.03616
## 41  9 85.39 0.03295
## 42 10 85.42 0.03002
## 43 10 85.44 0.02736
## 44 10 85.46 0.02493
## 45 10 85.48 0.02271
## 46 10 85.49 0.02069
## 47 10 85.50 0.01886
## 48 10 85.51 0.01718
## 49 10 85.52 0.01565
## 50 10 85.52 0.01426
## 51 10 85.53 0.01300
## 52 10 85.53 0.01184
## 53 10 85.54 0.01079
## 54 10 85.54 0.00983
## 55 10 85.54 0.00896
## 56 10 85.54 0.00816
## 57 10 85.54 0.00744
## 58 10 85.55 0.00678
## 59 10 85.55 0.00618
```

```
## 60 10 85.55 0.00563
## 61 10 85.55 0.00513
```

```
coef(fit)
```

```
## 11 x 61 sparse Matrix of class "dgCMatrix"
##
## (Intercept) 0.04946026 0.05674042 0.06549745 0.07119392 0.06847335 0.06358262
## V1          .          .          .          0.01543029 0.10235321 0.18166224
## V2          .          .          .          .          .          0.04447621
## V3          .          0.09952180 0.18546941 0.26364882 0.33441509 0.39830956
## V4          .          .          .          0.02508835 0.10518843 0.18142716
## V5          .          0.07336220 0.16241134 0.23898482 0.29496346 0.34668528
## V6          .          .          .          .          .          .
## V7          .          .          .          .          .          .
## V8          .          .          .          .          .          .
## V9          .          .          .          .          .          .
## V10         .          .          .          .          .          .
##
## (Intercept) 0.05673135 0.05048873 0.04480069 0.03961796 0.03489564 0.03059285
## V1          0.25404466 0.31999669 0.38008973 0.43484426 0.48473455 0.53019273
## V2          0.12913806 0.20627644 0.27656206 0.34060370 0.39895606 0.45212455
## V3          0.45596203 0.50849605 0.55636309 0.59997776 0.63971782 0.67592748
## V4          0.25413228 0.32037976 0.38074198 0.43574180 0.48585558 0.53151739
## V5          0.39451731 0.43809912 0.47780925 0.51399164 0.54695969 0.57699895
## V6          .          .          .          .          .          .
## V7          .          .          .          .          .          .
## V8          .          .          .          .          .          .
## V9          .          .          .          .          .          .
## V10         .          .          .          .          .          .
##
## (Intercept) 0.0266723 0.02310004 0.01984556 0.01687977 0.01417746 0.01171521
## V1          0.5716125 0.60935270 0.64373954 0.67507215 0.70362126 0.72963414
## V2          0.5005697 0.54471112 0.58493362 0.62158038 0.65497154 0.68539633
## V3          0.7089204 0.73898227 0.76636218 0.79132110 0.81406275 0.83478409
## V4          0.5731227 0.61103197 0.64556742 0.67704087 0.70571831 0.73184812
## V5          0.6043696 0.62930871 0.65203579 0.67274038 0.69160562 0.70879493
## V6          .         .          .          .          .          .
## V7          .         .          .          .          .          .
## V8          .         .          .          .          .          .
## V9          .         .          .          .          .          .
## V10         .         .          .          .          .          .
##
## (Intercept) 0.0094717 0.007427498 0.005564896 0.003867764 0.0023214
## V1          0.7533361 0.774932464 0.794610257 0.812539929 0.8288768
## V2          0.7131183 0.738377446 0.761392680 0.782363303 0.8014710
## V3          0.8536646 0.870867820 0.886542753 0.900825168 0.9138388
## V4          0.7556566 0.777350061 0.797116306 0.815126573 0.8315369
```

```
## V5          0.7244572 0.738728060 0.751731142 0.763579067 0.7743745
## V6          .         .           .           .           .
## V7          .         .           .           .           .
## V8          .         .           .           .           .
## V9          .         .           .           .           .
## V10         .         .           .           .           .
##
## (Intercept) 0.0009124101 -0.0003714086 -0.001541176 -0.002607025 -0.003578187
## V1          0.8437623067  0.8573254460  0.869683673  0.880944030  0.891204047
## V2          0.8188811338  0.8347446413  0.849198879  0.862369041  0.874369203
## V3          0.9256962853  0.9365004088  0.946344724  0.955314497  0.963487420
## V4          0.8464892968  0.8601134030  0.872527181  0.883838153  0.894144290
## V5          0.7842108093  0.7931733298  0.801339645  0.808780487  0.815560305
## V6          .             .             .            .            .
## V7          .             .             .            .            .
## V8          .             .             .            .            .
## V9          .             .             .            .            .
## V10         .             .             .            .            .
##
## (Intercept) -0.004463073 -0.004986676 -0.005378095 -0.005734919 -0.006060045
## V1           0.900552593  0.908794577  0.916219659  0.922985192  0.929149694
## V2           0.885303303  0.894799824  0.903312369  0.911068953  0.918136463
## V3           0.970934283  0.976752742  0.981751346  0.986309268  0.990462278
## V4           0.903534858  0.911822172  0.919286720  0.926089281  0.932287523
## V5           0.821737822  0.827586821  0.832985929  0.837904585  0.842386280
## V6           .            .            .            .            .
## V7           .            .            .            .            .
## V8           .            .            .            .            .
## V9           .            .            .            .            .
## V10          .            0.003962525  0.008765549  0.013140954  0.017127658
##
## (Intercept) -0.005977649 -0.00572889 -0.005501742 -0.005294772 -0.005106189
## V1           0.935811082  0.94236796  0.948339928  0.953781357  0.958739385
## V2           0.925059992  0.93159367  0.937545021  0.942967670  0.947908586
## V3           0.994404603  0.99807773  1.001420813  1.004466900  1.007242381
## V4           0.937758973  0.94266814  0.947139065  0.951212804  0.954924643
## V5           0.846239203  0.84964036  0.852740722  0.855565657  0.858139633
## V6           .            .           .            .            .
## V7           .            .           .            .            .
## V8           .            .           .            .            .
## V9          -0.005092399 -0.01208225 -0.018450798 -0.024253579 -0.029540856
## V10          0.021478885  0.02577240  0.029685820  0.033251586  0.036500580
##
## (Intercept) -0.00500413 -0.004970190 -0.005019018 -0.0050946040 -0.005384572
## V1           0.96377879  0.968538483  0.973023210  0.9771138357  0.981183848
## V2           0.95221755  0.956230940  0.960027541  0.9634909615  0.966666689
## V3           1.01027416  1.013303891  1.016325098  1.0190855059  1.021859552
```

```
## V4           0.95880071  0.962463056  0.965916597  0.9690640086  0.971930428
## V5           0.86019958  0.861990192  0.863553946  0.8649765027  0.866036264
## V6          -0.00453275 -0.008995486 -0.013210769 -0.0170557542 -0.020724281
## V7           .           0.001377378  0.005081461  0.0084477644  0.011317911
## V8           .           .            .            0.0004284353  0.003297292
## V9          -0.03457736 -0.038802345 -0.041982006 -0.0448625068 -0.047439638
## V10          0.03895438  0.041090746  0.042964207  0.0445948685  0.045556305
##
## (Intercept) -0.005642483 -0.005877468 -0.006091577 -0.006286665 -0.006464422
## V1           0.984855023  0.988199980  0.991247780  0.994024822  0.996555159
## V2           0.969560706  0.972197607  0.974600253  0.976789455  0.978784174
## V3           1.024365603  1.026648908  1.028729371  1.030625011  1.032352247
## V4           0.974546939  0.976930952  0.979103175  0.981082425  0.982885843
## V5           0.867027749  0.867931201  0.868754393  0.869504455  0.870187884
## V6          -0.024050552 -0.027081283 -0.029842773 -0.032358940 -0.034651577
## V7           0.013955635  0.016359064  0.018548978  0.020544347  0.022362453
## V8           0.005886637  0.008245925  0.010395620  0.012354342  0.014139057
## V9          -0.049782150 -0.051916540 -0.053861317 -0.055633325 -0.057247913
## V10          0.046445220  0.047255211  0.047993244  0.048665713  0.049278441
##
## (Intercept) -0.006626388 -0.006766658 -0.006901579 -0.007024704 -0.007136896
## V1           0.998860708  1.000945331  1.002860443  1.004605835  1.006196184
## V2           0.980601687  0.982251153  0.983760382  0.985135815  0.986389066
## V3           1.033926041  1.035311442  1.036621175  1.037815699  1.038904134
## V4           0.984529051  0.986002439  0.987368270  0.988613259  0.989747658
## V5           0.870810598  0.871393331  0.871909334  0.872379129  0.872807180
## V6          -0.036740543 -0.038628403 -0.040363624 -0.041945136 -0.043386161
## V7           0.024019043  0.025512870  0.026889410  0.028143838  0.029286826
## V8           0.015765222  0.017220475  0.018572125  0.019804451  0.020927318
## V9          -0.058719065 -0.060072308 -0.061292860 -0.062404676 -0.063417716
## V10          0.049836737  0.050367325  0.050829469  0.051249989  0.051633138
##
## (Intercept) -0.007239121 -0.007332264 -0.007417133 -0.007494463 -0.007564922
## V1           1.007645250  1.008965586  1.010168627  1.011264793  1.012263578
## V2           0.987530982  0.988571454  0.989519492  0.990383310  0.991170388
## V3           1.039895876  1.040799514  1.041622876  1.042373092  1.043056662
## V4           0.990781281  0.991723080  0.992581212  0.993363109  0.994075546
## V5           0.873197204  0.873552579  0.873876383  0.874171422  0.874440250
## V6          -0.044699171 -0.045895536 -0.046985619 -0.047978863 -0.048883869
## V7           0.030328274  0.031277202  0.032141831  0.032929648  0.033647478
## V8           0.021950433  0.022882657  0.023732065  0.024506014  0.025211208
## V9          -0.064340760 -0.065181804 -0.065948131 -0.066646380 -0.067282598
## V10          0.051982249  0.052300345  0.052590183  0.052854272  0.053094901
##
## (Intercept) -0.00762330 -0.007681816 -0.007735581
## V1           1.01315982  1.013990390  1.014747094
## V2           0.99189387  0.992547200  0.993142080
```

```
## V3           1.04363496  1.044202713  1.044723402
## V4           0.99469182  0.995283861  0.995825446
## V5           0.87470162  0.874924324  0.875126348
## V6          -0.04969915 -0.050450428 -0.051135781
## V7           0.03425587  0.034851253  0.035397890
## V8           0.02584077  0.026425475  0.026959920
## V9          -0.06787672 -0.068405825 -0.068885995
## V10          0.05332982  0.053529683  0.053710459
```

# 4 Numerical Calibration in practice

Parameter $\lambda$ is chosen by cross validation using `cv.glmnet` such that:

```
library(glmnet)
n       = 100
p       = 10
p0      = 5
sigma   = 1
sigmaX  = 1
b0      = 1
beta0   = c(rep(b0,p0),rep(0,p-p0))
X       = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
Y       = X%*%beta0 + rnorm(n,0,sigma)

lambda.cv = cv.glmnet(X,Y, family = "gaussian",intercept=F)$lambda.1se  ##Careful, .1s
bh        = glmnet(X,Y,family = "gaussian",intercept=F,
                   lambda=lambda.cv)$beta
if ( sum(abs(bh))==0 ) {bh = rep(0,p)}
bh        = as.vector(bh)
```

The first part of your projet will be to re-implement the cross validation procedure, and to verify that your implementation is correct based on the `cv.glmnet` function that will be used to check your results. You will also implement the calibration of $\lambda$ based on the AIC and on the BIC.

# 5 Simulations setting

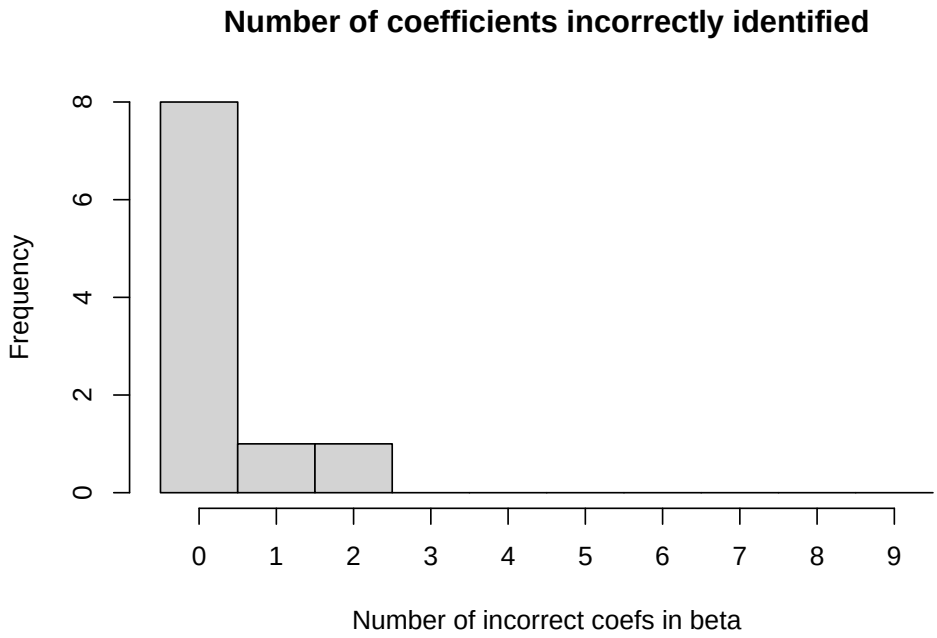
The performance of the lasso depends on different factors, and numerical simulations are used to study the impact of these factors on the capacity of the lasso to select the dimension of the model. Among those factors, we can identify $n$ (number of observations), $p$ (dimensionality), $p_0$ and $\beta^*$ (strength of the signal), $\sigma$ (strength of noise). Studying the impact of all factors would not be realistic, so we focus on:

- $n/p$ will be chosen so that we study the "low-dimensional regime" as well as the "high-dimensional regime".
- $p_0$ will be fixed at 5
- $\beta_0^* = 1$
- $\sigma$ will vary.

As an indicator of performance, we will study only the dimension of the selected model, ie $\hat{s} = \sum_{j=1}^{p} 1_{\hat{\beta}_j \neq 0}$. In order to conduct your simulations, you will start from the following example code:

```
n      = 100
p      = 10
p0     = 5  #Number of nonzero coefficients
sigma  = 1
sigmaX = 1
b0     = 1
beta0  = c(rep(b0,p0),rep(0,p-p0))
B      = 10 #Number of simulations
res    = matrix(NA,ncol=5,nrow=B) #Stores n,p, sigma, number of nonzero
                  #coefficients recovered, number of nz incorrectly identified

betah  = matrix(NA, ncol = p, nrow = B)

# fixed design setting
X = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})

for (b in 1:B){
  Y     = X%*%beta0 + rnorm(n,0,sigma)
  # estimate betah using the calibrated lasso
  # betah =
  lambda.cv = cv.glmnet(X,Y, family = "gaussian",intercept=F)$lambda.1se
  temp = glmnet(X,Y,family = "gaussian",intercept=F, lambda=lambda.cv)$beta
  ##the value returned by glmnet(...)$beta is a sparse matrix
  #In a sparse matrix : @i (non zero indices); @x non zero values

  indices_identified = c(rep(0, p)) #Fill indices_identified with 1 at
                                    #indices in temp@i, set others to 0
  for (index in temp@i) {
    indices_identified[index] = 1
  }

  betah[b,]  = c(temp@i, rep(NA, p-length(temp@i))) #We store in betah[b,]
    #the nonzero indices and we fill with NA to get a vector of the proper size
  res[b,] = c(n, p, sigma, length(temp@i), sum(abs(indices_identified - beta0)))
}
res           = as.data.frame(res)
colnames(res) = c("n","p","sigma","nz", "nzcorrect")


hist1<-hist(res$nz, breaks = 1:10, axes = FALSE,
            xlab="Number of non-zero coefs in beta",
            main = "Number of coefficients identified")
axis(side=1,at=hist1$mids,labels=1:9) #So that breakpoints are centered
axis(side=2)          #Because of previous trick, we had to hide both axis,
```


```
                    #so we display again the Y-axis


hist2 <- hist(res$nzcorrect, breaks = 0:10, axes = FALSE,
              xlab="Number of incorrect coefs in beta",
              main = "Number of coefficients incorrectly identified")
axis(side=1,at=hist2$mids,labels=0:9)
axis(side=2)
```




## 5.1 AIC

In this section, we implement the Akaike Information Criterion (AIC):

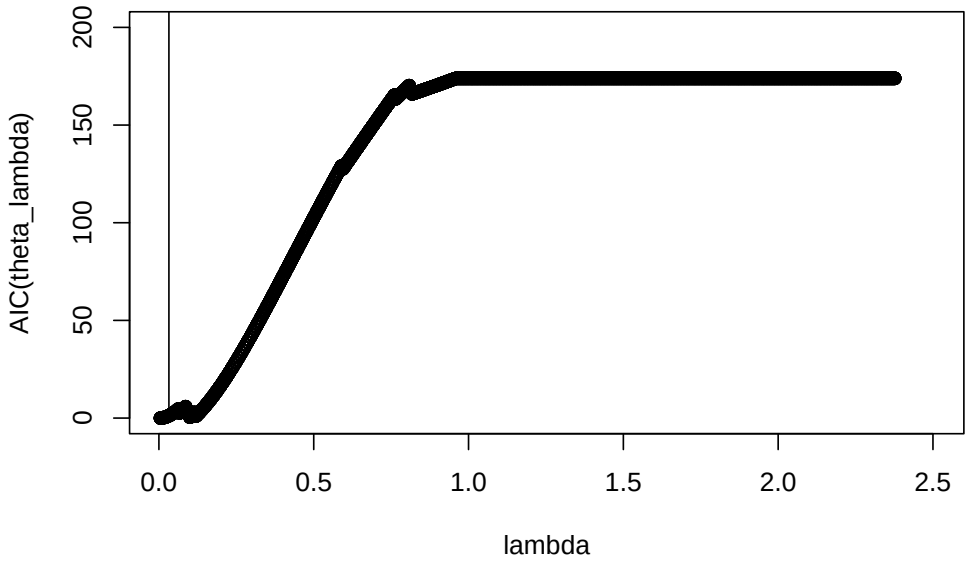
$$AIC = 2k - 2ln(\hat{L})$$

where $\hat{L}$ is the maximum value of the likelihood function for the model under consideration. The procedure consists of computing the Lasso with parameters $\lambda_1, \cdots, \lambda_N$.

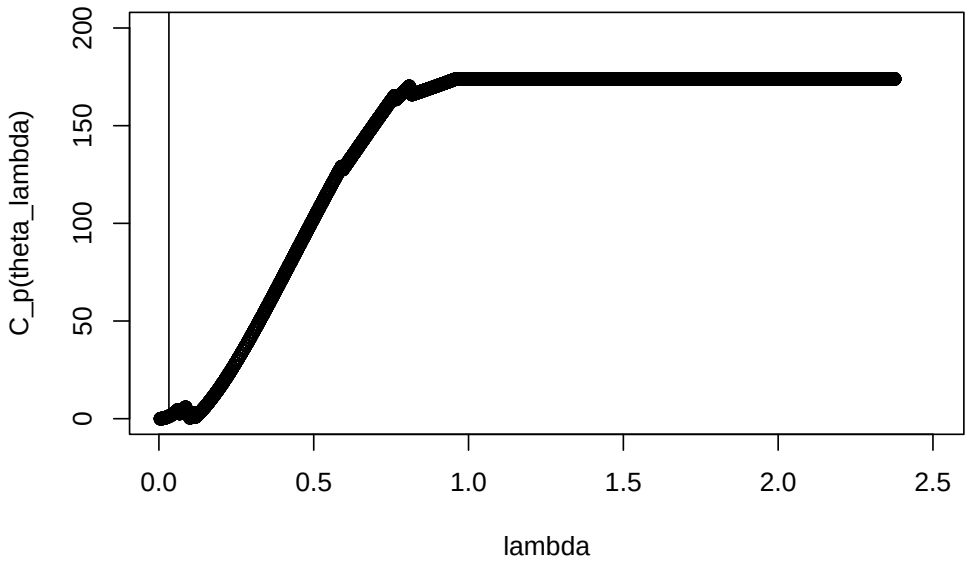
```
n      = 100
p      = 10
p0     = 5  #Number of nonzero coefficients
sigma  = 1
sigmaX = 1
b0     = 1
beta0  = c(rep(b0,p0),rep(0,p-p0))
betah  = matrix(NA, ncol = p, nrow = B)
X = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
Y     = X%*%beta0 + rnorm(n,0,sigma)



N <- 500 #Number of lambda tested
lambda_step <- 0.005 #Difference between two consecutive lambda
lambda_init <- 0.005 #First value used by lambda

lambda <- lambda_init
```

```r
results_frame <- data.frame(
  lambda = c(),
  betah = c(),
  k = c(),
  loglikelihood = c(),
  AIC = c(),
  Cp = c(),
  AICc = c(),
  BIC = c()
)

for (b in 1:N){#The increment is lambda
  temp = glmnet(X,Y,family = "gaussian",intercept=F, lambda=lambda)$beta
  ##the value returned by glmnet(...)$beta is a sparse matrix
  #In a sparse matrix : @i (non zero indices); @x non zero values

  betah <- c(rep(0,p))
  j<-0
  for(i in temp@i){
    betah[i+1] <- temp@x[j+1]
    j<-j+1
  }

  #llik = log(((1/sqrt(2 * pi))**(p/2))*exp(-t(Y-betah)%*%(Y-betah)/(2)))
  RSS = t(Y-X%*%betah)%*%(Y-X%*%betah) #Simpler expression for log likelihood in Gauss
  new_row_df <- data.frame(
    lambda = lambda,
    betah = betah,
    k = j,
    negloglikelihood = -log(RSS),
    AIC = 2*j + n*log(RSS/n),
    Cp = RSS + 2*j*sigma/n,
    AICc = 2*j + n*log(RSS/n) + 2*j*(j+1)/(n-j-1),
    BIC = j*log(n) + n*log(RSS/n)
  )
  results_frame <- rbind(results_frame, new_row_df)

  lambda <- lambda+lambda_step

}
results_frame$AIC <- results_frame$AIC - min(results_frame$AIC)
results_frame$Cp <- results_frame$Cp - min(results_frame$Cp)
results_frame$AICc <- results_frame$AICc - min(results_frame$AICc)
results_frame$BIC <- results_frame$BIC - min(results_frame$BIC)

lambda_cv = cv.glmnet(X,Y, family = "gaussian",intercept=F)$lambda.min  #We compute th
s <- c(rep(lambda_cv, 1000))  #And fill an axis with it
```

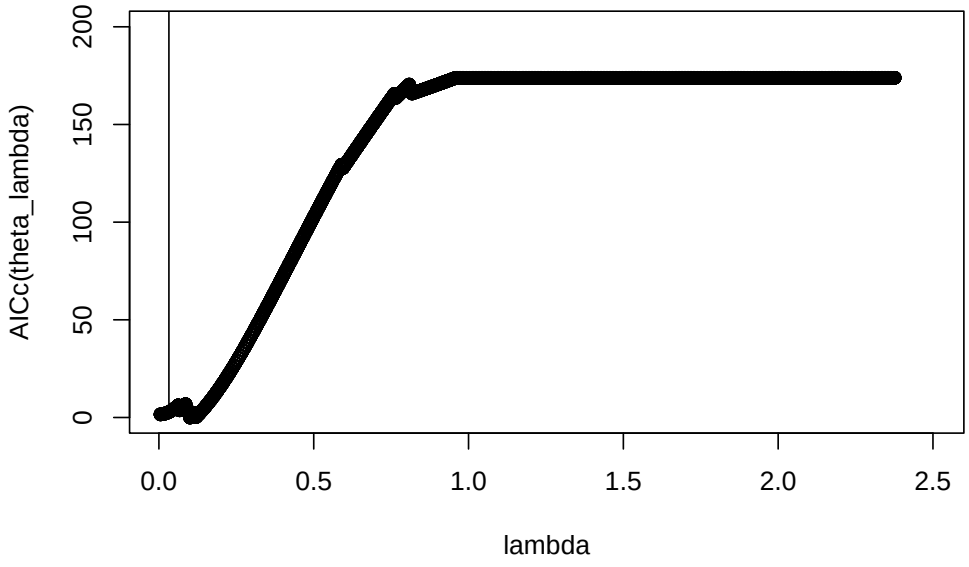
```
axis_lambda_cv <- xy.coords(x=s, y = 1:1000)  #that we will give to the plot

plot(x = results_frame$lambda, y = results_frame$AIC, type = "n", xlab = "lambda", yla
plot.xy(axis_lambda_cv, type = "l") #type = "l" to draw lines
plot.window(c(0.95*lambda_init, 1.05*lambda ), ylim = c(0.95*min(results_frame$AIC), 1
plot.xy( list(results_frame$lambda, results_frame$AIC, NULL, NULL), type="p")
```




```
plot(x = results_frame$lambda, y = results_frame$Cp, type = "n", xlab = "lambda", ylab
plot.xy(axis_lambda_cv, type = "l") #type = "l" to draw lines
plot.window(c(0.95*lambda_init, 1.05*lambda ), ylim = c(0.95*min(results_frame$AIC), 1
plot.xy( list(results_frame$lambda, results_frame$AIC, NULL, NULL), type="p")
```

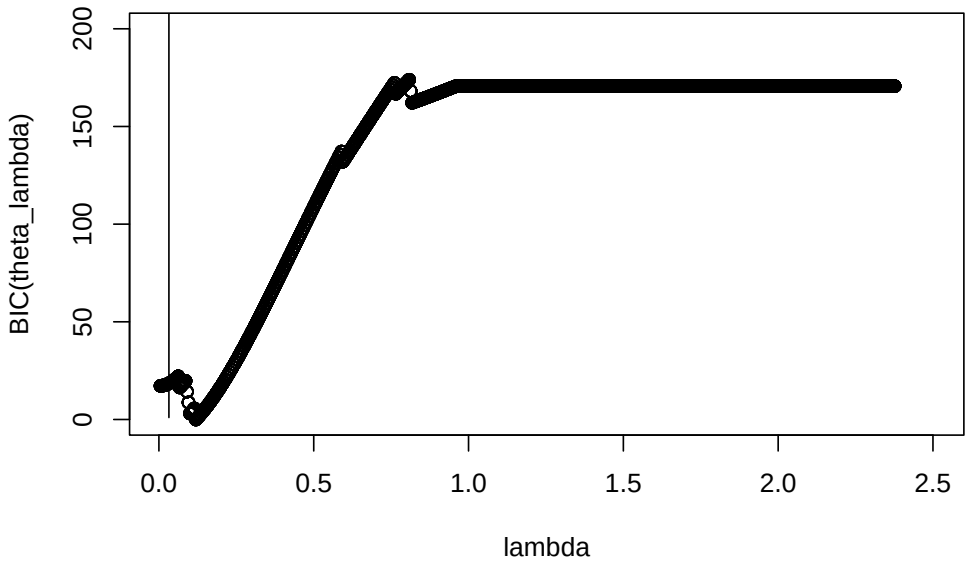



```
plot(x = results_frame$lambda, y = results_frame$AICc, type = "n", xlab = "lambda", yl
plot.xy(axis_lambda_cv, type = "l") #type = "l" to draw lines
plot.window(c(0.95*lambda_init, 1.05*lambda ), ylim = c(0.95*min(results_frame$AICc),
plot.xy( list(results_frame$lambda, results_frame$AICc, NULL, NULL), type="p")
```




```
plot(x = results_frame$lambda, y = results_frame$BIC, type = "n", xlab = "lambda", yla
plot.xy(axis_lambda_cv, type = "l") #type = "l" to draw lines
```

```
plot.window(c(0.95*lambda_init, 1.05*lambda ), ylim = c(0.95*min(results_frame$BIC), 1
plot.xy( list(results_frame$lambda, results_frame$BIC, NULL, NULL), type="p")
```




We then define the best value for $\lambda$ with respect to AIC as:

$$\lambda_{AIC} = argmin_{\lambda_i} AIC.$$

Hence, for each $i$ we need to compute the likelihood of the model with parameter $\beta_{\lambda_i}$ and $k$ as the number of nonzero coefficients of $\beta_{\lambda_i}$. The likelihood, with Gaussianity assumption and covariance matrix $\sigma I$, is given by

$$\hat{L_{\lambda_i}} = \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}(Y_i - \beta)^t(Y_i - \beta)$$

. In our setting, we compute $S = \sum_i (Y_i - \beta_\lambda)^2$ and then

$$L_Y(\lambda) = (2\pi\sigma^2)^{-\frac{n}{2}} \exp(-\frac{1}{2\sigma^2}S)$$

For each instance of $Y = X\theta^* + \epsilon$ for given parameters $p/n$(dimensionality)\$ of $k = s * (p/n)$ (sparsity), while the others are fixed; we obtain lambda_AIC, lambda_CV, lambda_BIC,.... the optimal calibration of lasso for a same problem and a given criterion.

We want to know given the dimensionality and perhaps the sparsity which criterion is optimal. For each instance of $Y = X\theta^*$ we may define

$$\lambda_{opt} = Argmin_\lambda ||X\theta^* - X\hat{\theta}_\lambda||_2$$

the "optimal" value of lambda, in the sense that given the solution $\theta^*$ (unknown in practice), we look for the lambda that would give the model with the smallest

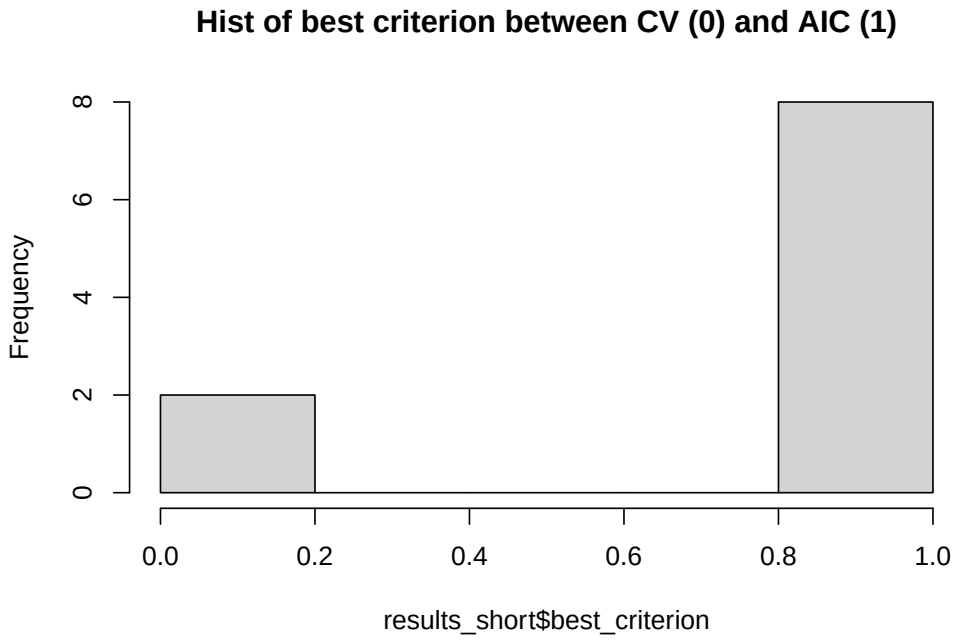
error (where the argmin could run through all lambdas or only on the one given by our criterions). Note that the choice of definition of $\lambda_{opt}$ is arbitrary, replacing the norm by an $\ell^1, \ell^\infty, \cdots$ can be justified and the argmin could ignore the dependence on $X$.

Idea : modelling $\lambda_{opt}$ as a function of the dimensionality, the sparsity, the different criterions Naive idea: Define the norm in the argmin as some sup norm so that $hatlambda_{opt}$ is the criterion that gives the lambda of the criterion which is closest from sparsity estimate. Improved: A nearest-neighbour Improved (bis): Model $\hat{\lambda}_{opt}$ as some linear combination of $p/n, s, \lambda_{AIC}, \lambda_{BIC}, \ldots$. Improved (ter): Model as a polynomial/log linear of $p/n$ and $s$ (and $\lambda_{AIC}, \lambda_{Bic}, \cdots$)

```
##We fix p, n and k
#We run M instances of Y = X\theta + eps
#Each time we obtain $\lambda_{AIC, CV, opt}$ and we create a list which contains for
#We plot the histogram to get an idea if AIC or CV is best in our setup
#TODO : same thing, but instead pick an index whenever it is the one that gives the $\
n      = 100
p      = 10
p0     = 5  #Number of nonzero coefficients
sigma  = 1
sigmaX = 1
b0     = 1
beta0  = c(rep(b0,p0),rep(0,p-p0))
betah  = matrix(NA, ncol = p, nrow = B)


B <- 20 #Number of lambda tested
lambda_step <- 0.1 #Difference between two consecutive lambda
lambda_init <- 0.0 #First value used by lambda


results_short <- data.frame(
  lambda_AIC =c(),
  lambda_AICc = c(),
  lambda_CV = c(),
  lambda_opt = c(),
  best_criterion = c()
)

N<- 10 ## Number of instances of solving Y=X\theta + ups
for(t in 1:N){

  X = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
  Y = X%*%beta0 + rnorm(n,0,sigma)

  lambda <- lambda_init
  results_frame <- data.frame(
```

```
  lambda = c(),
  betah = c(),
  k = c(),
  loglikelihood = c(),
  AIC = c(),
  BIC = c()
)

err_opt <- NA
lambda_opt <- 1

for (b in 1:B){#The increment is lambda
  temp = glmnet(X,Y,family = "gaussian",intercept=F, lambda=lambda)$beta
  ##the value returned by glmnet(...)$beta is a sparse matrix
  #In a sparse matrix : @i (non zero indices); @x non zero values

  betah <- c(rep(0,p))
  j<-0
  for(i in temp@i){
    betah[i+1] <- temp@x[j+1]
    j<-j+1
  }

  #llik = log(((1/sqrt(2 * pi))**(p/2))*exp(-t(Y-betah)%*%(Y-betah)/(2)))
  RSS = t(Y-X%*%betah)%*%(Y-X%*%betah) #Simpler expression for log likelihood in Ga
  new_row_df <- data.frame(
    lambda = lambda,
    betah = betah,
    k = j,
    negloglikelihood = -log(RSS),
    AIC = 2*j + n*log(RSS/n),
    AICc = 2*j + n*log(RSS/n) + 2*j*(j+1)/(n-j-1),
    BIC = j*log(n) + n*log(RSS/n)
  )
  results_frame <- rbind(results_frame, new_row_df)
  if(err_opt >= Norm(betah - beta0, p=2) || is.na(err_opt)){
    lambda_opt <- lambda   #Obtaining the lambda which gives the solution closest fr
    err_opt <- Norm(betah - beta0, p = 2)
  }
  lambda <- lambda+lambda_step
}
esc <-1
lambda_min_AIC <- results_frame$AIC[0]
min_lambda_AIC <- min(results_frame$AIC)
i<-1
while (esc != 0 || i < B) {
  if( min_lambda_AIC >= results_frame$AIC[i]){
```

```
      esc = 0
      lambda_min_AIC <-results_frame$lambda[i]  #Obtaining the optimal value of lambda
    }
    i<- i + 1
  }
  esc <-1
  lambda_min_AICc <- results_frame$AICc[0]
  min_lambda_AICc <- min(results_frame$AICc)
  i<-1
  while (esc != 0 || i < B) {
    if( min_lambda_AICc >= results_frame$AICc[i]){
      esc = 0
      lambda_min_AICc <-results_frame$lambda[i]  #Obtaining the optimal value of lambda
    }
    i<- i + 1
  }

  new_row_short_df <- data.frame(
    lambda_AIC = lambda_min_AIC,
    lambda_AICc = lambda_min_AICc,
    lambda_CV = cv.glmnet(X,Y, family = "gaussian",intercept=F)$lambda.min,
    lambda_opt = lambda_opt,
    best_criterion = NA
  )
  if(abs(new_row_short_df$lambda_AIC - new_row_short_df$lambda_opt) > abs(new_row_shor
    new_row_short_df$best_criterion = 0 ##0 : best criterion is CV
  }
  else{
    new_row_short_df$best_criterion = 1  ##1: best criterion is AIC
  }
  results_short <- rbind(results_short, new_row_short_df)
  disp(t, "/", N)
}

hist(results_short$best_criterion, main = "Hist of best criterion between CV (0) and A
```




```
###We wrap the previous code in a function
#Then we compute for varying $n/p$ whether best is AIC, BIC or CV

GetLambdaComparison <- function(n = 100,
                                p      = 10,
                                p0     = 5,  #Number of nonzero coefficients
                                sigma  = 1,
                                sigmaX = 1,
                                b0     = 1,
                                B = 20, #Number of lambda tested
                                N = 10, #Number of instances of solving Y = X\theta +
                                lambda_step = 0.1, #Difference between two consecutive
                                lambda_init = 0.0 #First value used by lambda
                                ){

  if(p < p0 ){##We treat this (impossible) case so that we can obtain dataframes of eq
    results_short <- data.frame(
      lambda_AIC =NA,
      lambda_AICc = NA,
      lambda_BIC = NA,
      lambda_CV = NA,
      lambda_opt = NA,
      best_criterion = NA
    )
    # results_short_row <- data.frame(
    #   lambda_AIC =NA,
    #   lambda_BIC = NA,
```

```
  #   lambda_CV = NA,
  #   lambda_opt = NA,
  #   best_criterion = NA
  # )
  # results_short <- rbind(results_short, results_short_row)

  return(results_short)
}

beta0  = c(rep(b0,p0),rep(0,p-p0))
betah  = matrix(NA, ncol = p, nrow = B)

results_short <- data.frame(
  lambda_AIC =c(),
  lambda_AICc = c(),
  lambda_BIC = c(),
  lambda_CV = c(),
  lambda_opt = c(),
  best_criterion = c(),
  res_AIC = c(),
  res_AICc = c(),
  res_BIC = c(),
  res_CV = c()
)

if(n<30){
  print("Warning, if n < 30, cv will start throwing warnings")
}

for(t in 1:N){ ##We do the simulation N times

  X = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
  Y = X%*%beta0 + rnorm(n,0,sigma)

  lambda <- lambda_init
  results_frame <- data.frame(
    lambda = c(),
    betah = c(),
    k = c(),
    loglikelihood = c(),
    AIC = c(),
    AICc = c(),
    BIC = c()
  )

  err_opt <- NA
  lambda_opt <- 1
```

```
min_lambda_AIC <- NA
lambda_AIC <- NA
min_lambda_AICc <- NA
lambda_AICc <- NA
min_lambda_BIC <- NA
lambda_BIC <- NA
for (b in 1:B){#The increment is lambda
  temp = glmnet(X,Y,family = "gaussian",intercept=F, lambda=lambda)$beta
  ##the value returned by glmnet(...)$beta is a sparse matrix
  #In a sparse matrix : @i (non zero indices); @x non zero values

  betah <- c(rep(0,p))
  j<-1
  for(i in temp@i){
    betah[i+1] <- temp@x[j]
    j<-j+1
  }

  #llik = log(((1/sqrt(2 * pi))**(p/2))*exp(-t(Y-betah)%*%(Y-betah)/(2)))
  RSS = t(Y-X%*%betah)%*%(Y-X%*%betah) #Simpler expression for log likelihood in G
  new_row_df <- data.frame(
    lambda = lambda,
    betah = betah,
    k = j,
    negloglikelihood = -log(RSS),
    AIC = 2*j + n*log(RSS),
    AICc = 2*j + n*log(RSS/n) + 2*j*(j+1)/(n-j-1),
    BIC = j*log(n) + n*log(RSS/n)
  )
  results_frame <- rbind(results_frame, new_row_df)

  ##Before restarting to loop we check for the best value for lambda_opt
  if(err_opt >= Norm(betah - beta0, p=2) || is.na(err_opt)){
    lambda_opt <- lambda   #Obtaining the lambda which gives the solution closest
    err_opt <- Norm(betah - beta0, p = 2)
  }

  #And we also search for the best value for lambda under AIC
  if(min_lambda_AIC > new_row_df$AIC || is.na(min_lambda_AIC)){
    min_lambda_AIC <- new_row_df$AIC
    lambda_AIC <- lambda
  }
  #And we also search for the best value for lambda under AICc
  if(min_lambda_AICc > new_row_df$AICc || is.na(min_lambda_AICc)){
    min_lambda_AICc <- new_row_df$AICc
    lambda_AICc <- lambda
  }
```

```
      #And we also search for the best value for lambda under BIC
      if(min_lambda_BIC > new_row_df$BIC || is.na(min_lambda_BIC)){
        min_lambda_BIC <- new_row_df$BIC
        lambda_BIC <- lambda
      }
      lambda <- lambda+lambda_step
    }##End of for loop for finding best lambda (depending on criterion)


    ##We create a frame in which we store the best values obtained
    new_row_short_df <- data.frame(
      lambda_AIC = lambda_AIC,
      lambda_AICc = lambda_AICc,
      lambda_BIC = lambda_BIC,
      lambda_CV = cv.glmnet(X,Y, family = "gaussian",intercept=F)$lambda.min,
      lambda_opt = lambda_opt,
      best_criterion = NA,
      res_AIC = norm(glmnet(X,Y, family = "gaussian", lambda = lambda_AIC, intercept =
      res_AICc = norm(glmnet(X,Y, family = "gaussian", lambda = lambda_AICc, intercept
      res_BIC = norm(glmnet(X,Y, family = "gaussian", lambda = lambda_BIC, intercept =
      res_CV = NA
    )
    new_row_short_df$res_CV = norm(glmnet(X,Y, family = "gaussian", lambda = new_row_s

    ##We check which criterion gives the closest lambda wrt lambda_opt
    if(min(abs(new_row_short_df$lambda_AIC - new_row_short_df$lambda_opt), abs(new_row
      new_row_short_df$best_criterion = 0 ##0 : best criterion is CV
    }
    else if(    abs(new_row_short_df$lambda_AIC - new_row_short_df$lambda_opt)
                  > abs(new_row_short_df$lambda_BIC - new_row_short_df$lambda_opt)
            ){
      new_row_short_df$best_criterion = 2  ##1: best criterion is BIC
    }
    else{
        new_row_short_df$best_criterion = 1  ##1: best criterion is AIC
    }


    results_short <- rbind(results_short, new_row_short_df)
  }##End of for loop for computing on different instance of Y = Xbeta + eps


  return(results_short)
}
```

```
n <- 100
prange<- (0:40)*5
```

```r
ratio_best_AIC = c(rep(0,length(prange)))
ratio_best_BIC = c(rep(0,length(prange)))
ratio_best_CV = c(rep(0,length(prange)))

result_var_np = data.frame(
  p = c(),
  num_AIC_best = c(),   ##Number of times when AIC is the best
  num_CV_best = c(),   ##Number of times when CV is best
  num_BIC_best = c(),
  err_AIC = c(),
  err_BIC = c(),
  err_CV = c()
)
for(s in 1:length(prange)){
  results_lambda <- GetLambdaComparison(n=n, p = prange[s], B = 20, N= 50)
  new_row_result_var_np = data.frame(
    p = prange[s],
    num_AIC_best = sum(results_lambda$best_criterion == 1),
    num_CV_best = sum(results_lambda$best_criterion == 0),
    num_BIC_best = sum(results_lambda$best_criterion == 2),
    err_AIC = mean(results_lambda$res_AIC, na.rm = TRUE),
    err_AICc = mean(results_lambda$res_AICc, na.rm = TRUE),
    err_BIC = mean(results_lambda$res_BIC, na.rm = TRUE),
    err_CV = mean(results_lambda$res_CV, na.rm = TRUE)
  )

    ratio_best_AIC[s] <- new_row_result_var_np$num_AIC_best / (new_row_result_var_np$n
  ratio_best_BIC[s] <- new_row_result_var_np$num_BIC_best / (new_row_result_var_np$num
  ratio_best_CV[s] <- new_row_result_var_np$num_CV_best / (new_row_result_var_np$num_C

  disp(s, "/", length(prange))

  result_var_np <- rbind(result_var_np, new_row_result_var_np)
}
```

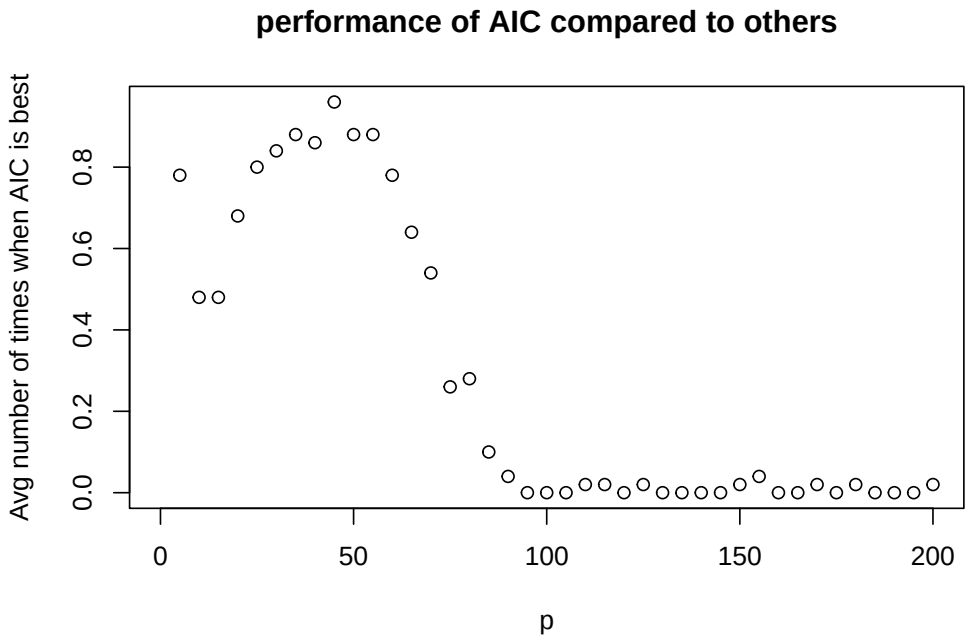
```
## Warning in mean.default(results_lambda$res_AIC, na.rm = TRUE): l'argument n'est
## ni numérique, ni logique : renvoi de NA
```

```
## Warning in mean.default(results_lambda$res_AICc, na.rm = TRUE): l'argument n'est
## ni numérique, ni logique : renvoi de NA
```

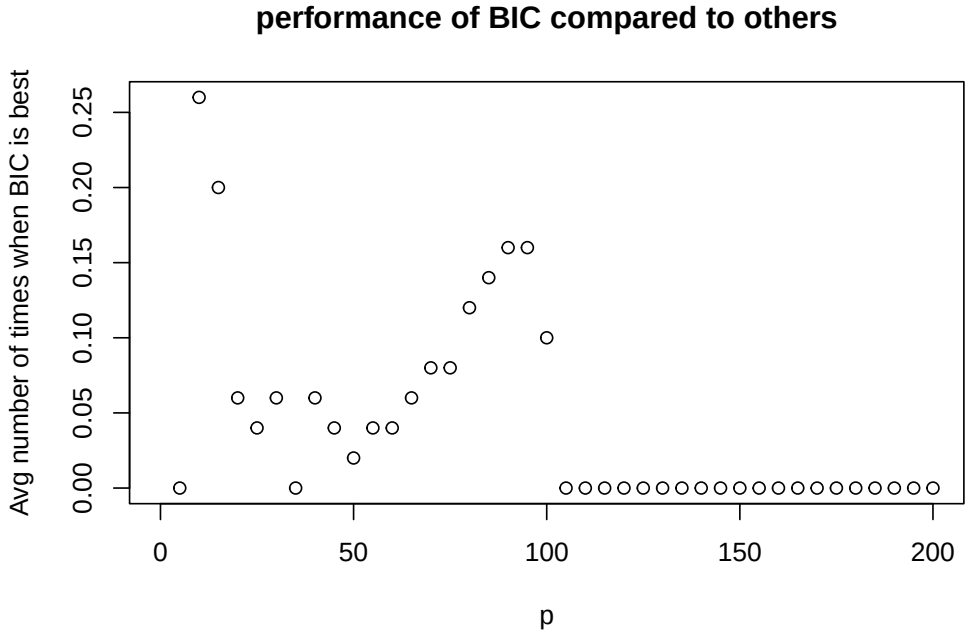
```
## Warning in mean.default(results_lambda$res_BIC, na.rm = TRUE): l'argument n'est
## ni numérique, ni logique : renvoi de NA
```

```
## Warning in mean.default(results_lambda$res_CV, na.rm = TRUE): l'argument n'est
## ni numérique, ni logique : renvoi de NA
```

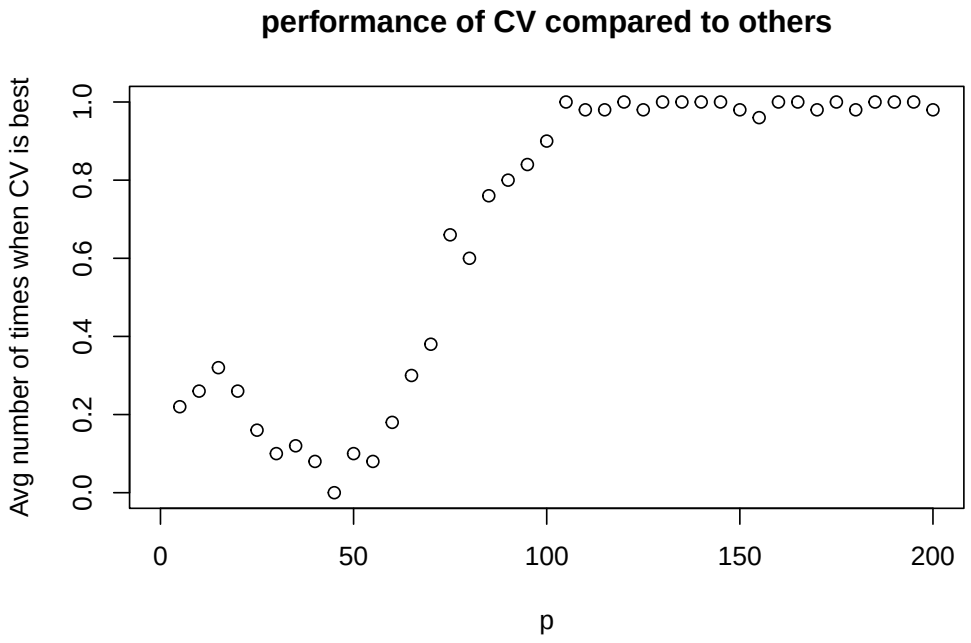
```r
plot(ratio_best_AIC, x=prange, main = "performance of AIC compared to others", ylab =
```



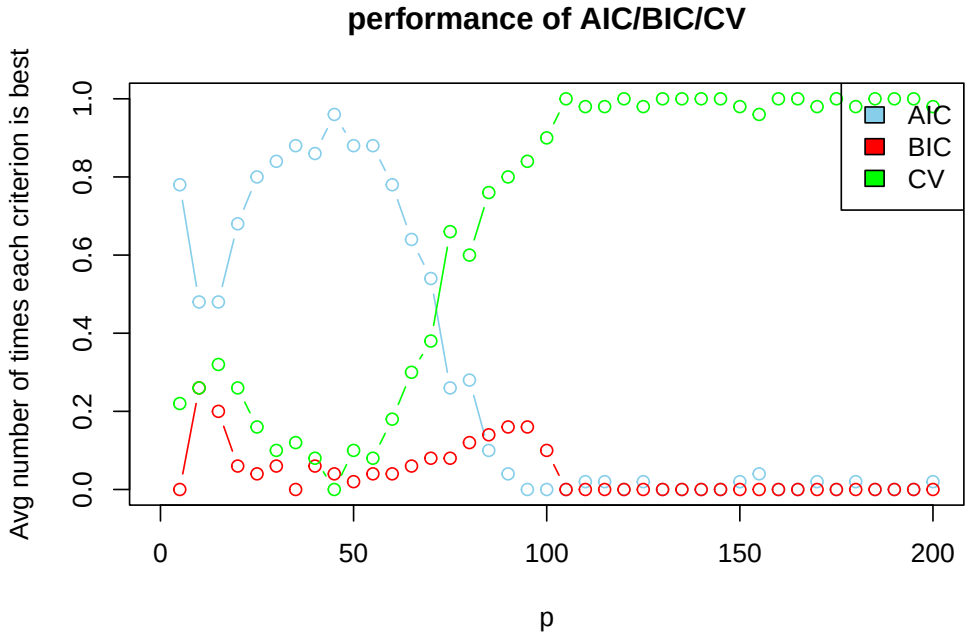

```
plot(ratio_best_BIC, x = prange, main = "performance of BIC compared to others", ylab
```




```
plot(ratio_best_CV, x = prange, main = "performance of CV compared to others", ylab =
```

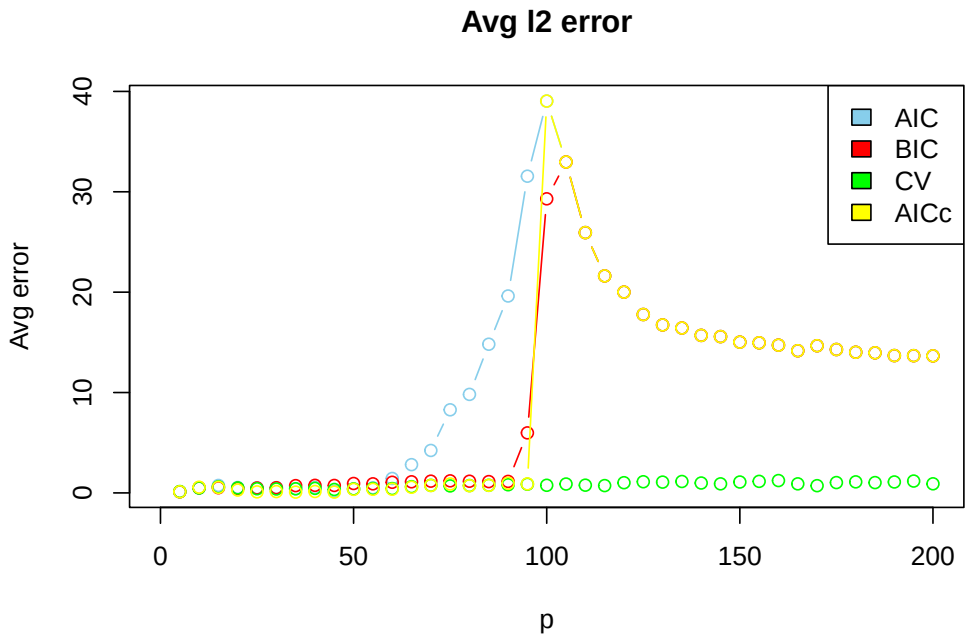



```
plot(ratio_best_AIC, x=prange, main = "performance of AIC/BIC/CV", ylab = "Avg number
points(ratio_best_BIC, x = prange, type = "b", col = "red")
points(ratio_best_CV, x = prange, type = "b", col = "green")
legend(x="topright", legend=c("AIC", "BIC", "CV"), fill=c("skyblue", "red", "green"))
```




```
plot(result_var_np$err_AIC, x = prange, main = "Avg l2 error", ylab = "Avg error", xla
points(result_var_np$err_BIC, x = prange, type = "b", col ="red")
```

```
points(result_var_np$err_CV, x = prange, type = "b", col = "green")
points(y=result_var_np$err_AICc, x = prange, type = "b", col="yellow")

legend(x="topright", legend=c("AIC", "BIC", "CV", "AICc"), fill=c("skyblue", "red", "g
```




We now compute, depending on n/p and k (the number of nonzero coefficient) the performance of AIC, BIC and CV.

```
##We wrap the previous code in a function to get, for fixed n and a fixed sparsity lev
#Then we will vary k between some values and then display the results
CompareCriterionVarNP <- function(
                                  n = 100,
                                  prange      = 5:200, #List of p tested, all p should b
                                  p0     = 5,  #Number of nonzero coefficients
                                  sigma  = 1,
                                  sigmaX = 1,
                                  b0     = 1,
                                  B = 30, #Number of lambda tested
                                  N = 10, #Number of instances of solving Y = X\theta +
                                  lambda_step = 0.05, #Difference between two consecutiv
                                  lambda_init = 0.0 #First value used by lambda

                                    ){
  # result_var_np = data.frame(
  #   p = c(rep(NA, max(p0 - prange[1], 0))),
  #   num_AIC_best = c(rep(NA, max(p0 -prange[1], 0))),   ##Number of times when AIC t
  #   num_CV_best = c(rep(NA, max(p0 -prange[1], 0))),   ##Number of times when CV is
```

```
  #   num_BIC_best = c(rep(NA, max(p0 -prange[1], 0)))
  #
  #   ## prange[1]  prange[2]
  #   #k1  k2  k3
  #   # k < prange[1] -> _best[k, ..., prange[1]] = NA
  # )
  result_var_np = data.frame(
    p = c(),
    num_AIC_best = c(),
    num_CV_best = c(),
    num_BIC_best = c(),
    err_AIC = c(),
    err_AICc = c(),
    err_BIC = c(),
    err_CV = c()
  )
  for(p in prange ){
    results_lambda <- GetLambdaComparison(n=n, p = p, B = B, N= N, p0 = p0,sigma = sig
    new_row_result_var_np = data.frame(
      p = p,
      num_AIC_best = sum(results_lambda$best_criterion == 1),
      num_CV_best = sum(results_lambda$best_criterion == 0),
      num_BIC_best = sum(results_lambda$best_criterion == 2),
      err_AIC = mean(results_lambda$res_AIC),
      err_AICc = mean(results_lambda$res_AICc),
      err_BIC = mean(results_lambda$res_BIC),
      err_CV = mean(results_lambda$res_CV)
    )
    result_var_np <- rbind(result_var_np, new_row_result_var_np)

  }
  return(result_var_np)
}
```

```
n<-100
results_glob = data.frame(
  k = c(),
  comp_criterion = c()
)
krange = c(5, 10, 20, 30, 40, 50)
prange = c(5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 1
N = 20
for(k in krange){

  new_row = data.frame(
    k = k,
    comp_criterion = CompareCriterionVarNP(n = n, prange = prange, p0 = k,N=N)
```

```
  )

  results_glob <- rbind(results_glob, new_row)
  disp(k, "/", max(krange))
}
```

```
##Here we do the plotting
print(length(results_glob$comp_criterion.num_AIC_best))
x = prange
y = krange
print(length(x)*length(y))
z = matrix(data = results_glob$comp_criterion.num_AIC_best, nrow = length(x), ncol = l
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p", col="skyblue", typ
z = matrix(data = results_glob$comp_criterion.num_BIC_best, nrow = length(x), ncol = l
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p", col="red", type =
z = matrix(data = results_glob$comp_criterion.num_CV_best, nrow = length(x), ncol = le
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p",  col="green", add
rgl.snapshot("comp_all_crits.png")
```

```
z = matrix(data = results_glob$comp_criterion.err_AIC, nrow = length(x), ncol = length
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p", col="skyblue", typ
z = matrix(data = results_glob$comp_criterion.err_AICc, nrow = length(x), ncol = lengt
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p", col="yellow", type
z = matrix(data = results_glob$comp_criterion.err_BIC, nrow = length(x), ncol = length
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p",  col="red", add =
z = matrix(data = results_glob$comp_criterion.err_CV, nrow = length(x), ncol = length(
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p",  col="green", add
rgl.snapshot("comp_all_crits_error.png")
```

Now, instead of comparing among criterions which one gives the closest lambda to the best (knowing the true solution) lambda, we want to compare for each criterion which ones give the right model dimension.

```
GetDimComparison <- function(n = 100,
                             p      = 10,
                             p0     = 5,  #Number of nonzero coefficients
                             sigma  = 1,
                             sigmaX = 1,
                             b0     = 1,
                             B = 100, #Number of lambda tested
                             N = 10, #Number of instances of solving Y = X\theta +
                             lambda_step = 0.1, #Difference between two consecutive
                             lambda_init = 0.0 #First value used by lambda
                             ){

  if(p < p0 ){##We treat this (impossible) case so that we can obtain dataframes of eq
    results_short <- data.frame(
      AIC_dim =NA,
```

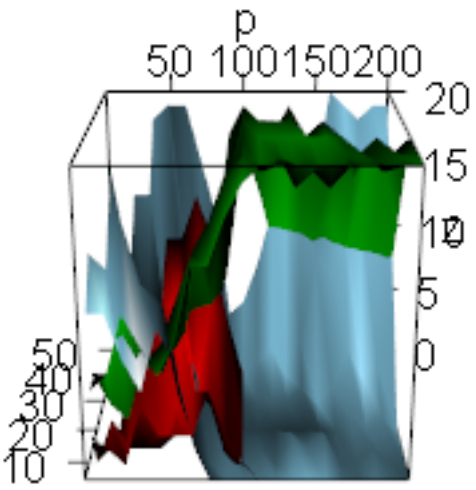



Figure 1: Comparison of which criterion gives the lambda closest to best lambda for varying dimensionality and sparsity (higher is better)

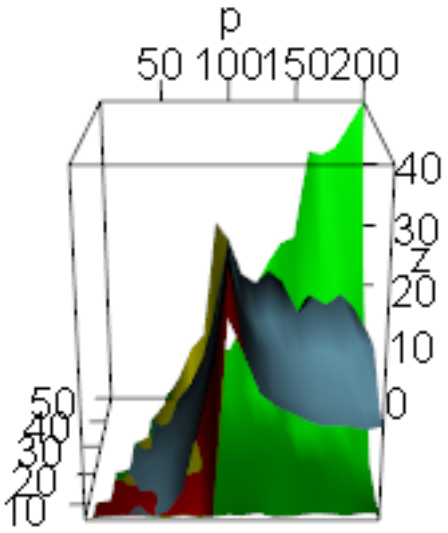


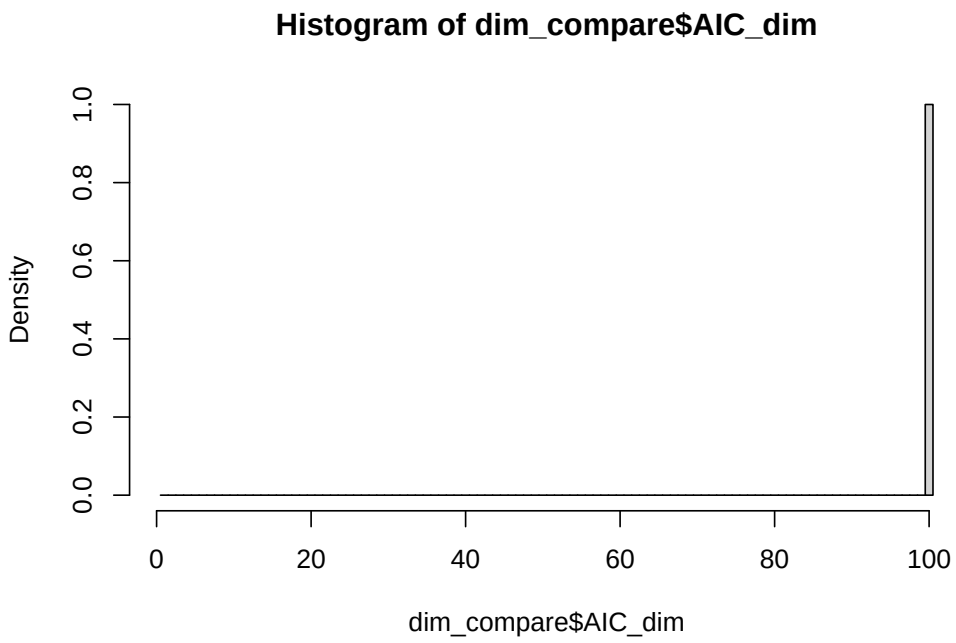

Figure 2: Comparison of prediction error for varying dimensionality and sparsity (lower is better)

```
    BIC_dim = NA,
    CV_dim = NA,
    true_dim = NA
  )
  return(results_short)
}

beta0  = c(rep(b0,p0),rep(0,p-p0))
betah  = matrix(NA, ncol = p, nrow = B)

results_dim <- data.frame(
  AIC_dim =c(),
  BIC_dim = c(),
  CV_dim = c(),
  true_dim = c()
)



for(t in 1:N){

  X = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
  Y = X%*%beta0 + rnorm(n,0,sigma)

  lambda <- lambda_init
  results_frame <- data.frame(
    lambda = c(),
    betah = c(),
    k = c(),
    loglikelihood = c(),
    AIC = c(),
    BIC = c()
  )

  min_lambda_AIC <- NA
  lambda_AIC <- NA
  min_lambda_BIC <- NA
  lambda_BIC <- NA
  for (b in 1:B){#The increment is lambda
    temp = glmnet(X,Y,family = "gaussian",intercept=F, lambda=lambda)$beta
    ##the value returned by glmnet(...)$beta is a sparse matrix
    #In a sparse matrix : @i (non zero indices); @x non zero values

    betah <- c(rep(0,p))
    j<-1
    for(i in temp@i){
      betah[i+1] <- temp@x[j]
```

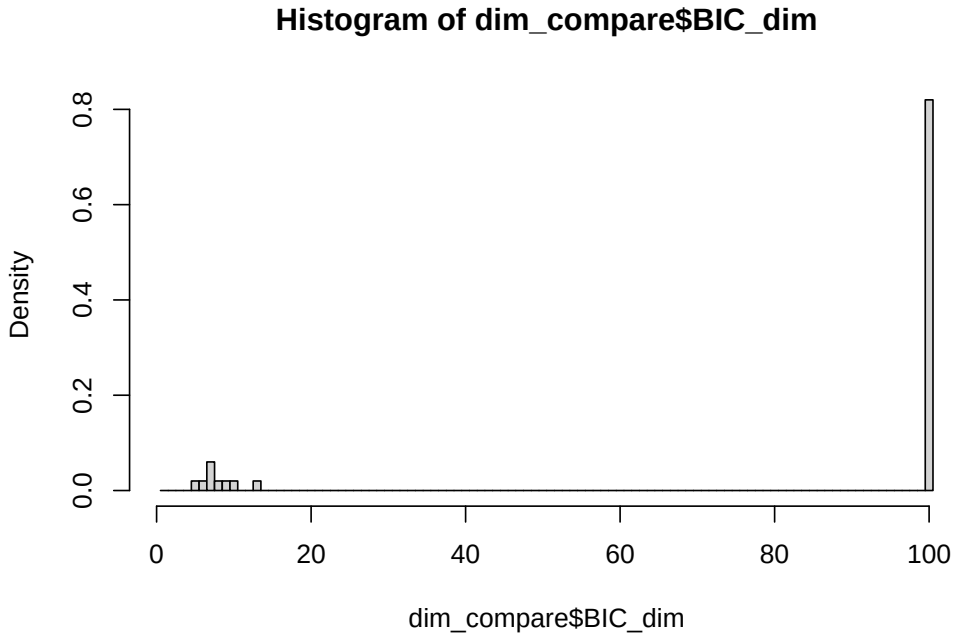
```
      j<-j+1
    }

    #llik = log(((1/sqrt(2 * pi))**(p/2))*exp(-t(Y-betah)%*%(Y-betah)/(2)))
    RSS = t(Y-X%*%betah)%*%(Y-X%*%betah) #Simpler expression for log likelihood in G
    new_row_df <- data.frame(
      lambda = lambda,
      betah = betah,
      k = j,
      negloglikelihood = -log(RSS),
      AIC = 2*j + n*log(RSS),
      BIC = j*log(n) + n*log(RSS/n)
    )
    results_frame <- rbind(results_frame, new_row_df)


    #We search for the best value for lambda under AIC
    if(min_lambda_AIC > new_row_df$AIC || is.na(min_lambda_AIC)){
      min_lambda_AIC <- new_row_df$AIC
      lambda_AIC <- lambda
    }
    #And we also search for the best value for lambda under BIC
    if(min_lambda_BIC > new_row_df$BIC || is.na(min_lambda_BIC)){
      min_lambda_BIC <- new_row_df$BIC
      lambda_BIC <- lambda
    }
    lambda <- lambda+lambda_step
  }
  lambda_CV = cv.glmnet(X,Y, family = "gaussian",intercept=F)$lambda.min

  #We compute for each optimal lambda the corresponding solution beta
  Beta_AIC = glmnet(X, Y, family = "gaussian", intercept=F, lambda = lambda_AIC)$bet
  Beta_BIC = glmnet(X, Y, family = "gaussian", intercept=F, lambda = lambda_BIC)$bet
  Beta_CV = glmnet(X, Y, family = "gaussian", intercept=F, lambda = lambda_CV)$beta

  #For each beta we compute its dimension
  new_row_results_dim <- data.frame(
    AIC_dim = length(Beta_AIC@i),
    BIC_dim = length(Beta_BIC@i),
    CV_dim = length(Beta_CV@i),
    true_dim = p0
  )
  results_dim <- rbind(results_dim, new_row_results_dim)
 }
 return(results_dim)
}
```

```
dim_compare = GetDimComparison(N = 50, n = 100, p = 100, p0 = 5)
```

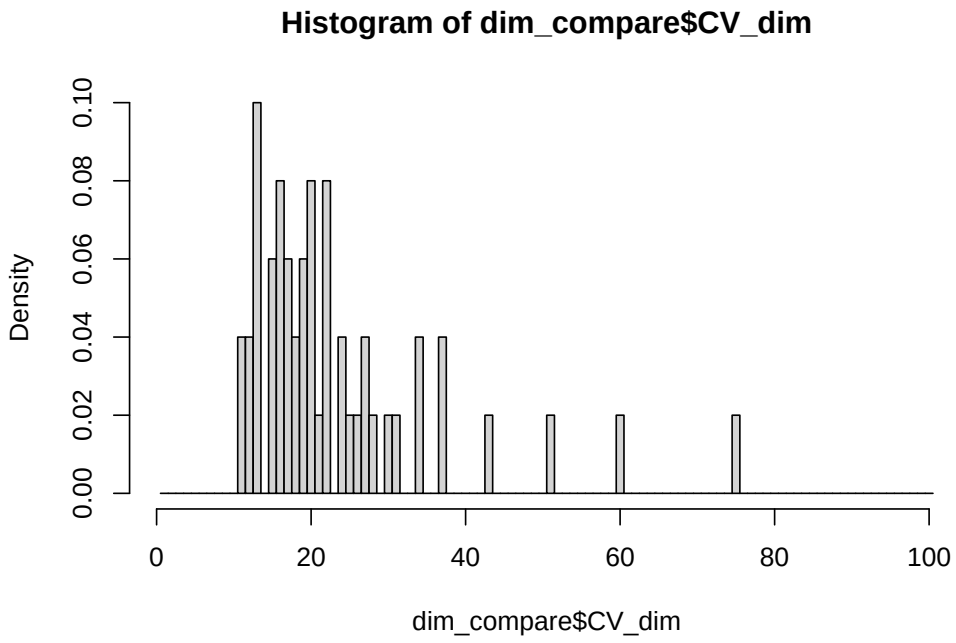
```
hist(dim_compare$AIC_dim, breaks = 0:100 +0.5, freq = FALSE)
```




```
hist(dim_compare$BIC_dim, breaks =0:100+0.5, freq = FALSE)
```



```
hist(dim_compare$CV_dim, breaks =0:100 +0.5, freq = FALSE)
```




We obtain from previous plots (with e.g. N=100, p=10 or p =100) that CV is the best criterion to obtain the right model dimension. Also BIC performs better than AIC for dimension identification, however in a high dimensional setting, the performance of BIC is quite low. In order to back this claim, we do as before and display for varying $n/p$ the dimension of the models identified by the different criterions.

We now plot, depending on n/p and k (the number of nonzero coefficient) the performance (avg model dimension) of AIC, BIC and CV.

```
##We wrap the previous code in a function to get, for fixed n and a fixed sparsity lev
#Then we will vary k between some values and then display the results
CompareDimCriterionVarNP <- function(
                                n = 100,
                                prange      = 5:200, #List of p tested, all p should b
                                p0     = 5,  #Number of nonzero coefficients
                                sigma  = 1,
                                sigmaX = 1,
                                b0     = 1,
                                B = 30, #Number of lambda tested
                                N = 10, #Number of instances of solving Y = X\theta +
                                lambda_step = 0.05, #Difference between two consecutiv
                                lambda_init = 0.0 #First value used by lambda
                                  ){
  # result_dim_var_np = data.frame(
  #   p = c(rep(prange[1], max(prange[1]-1, 0))),
```

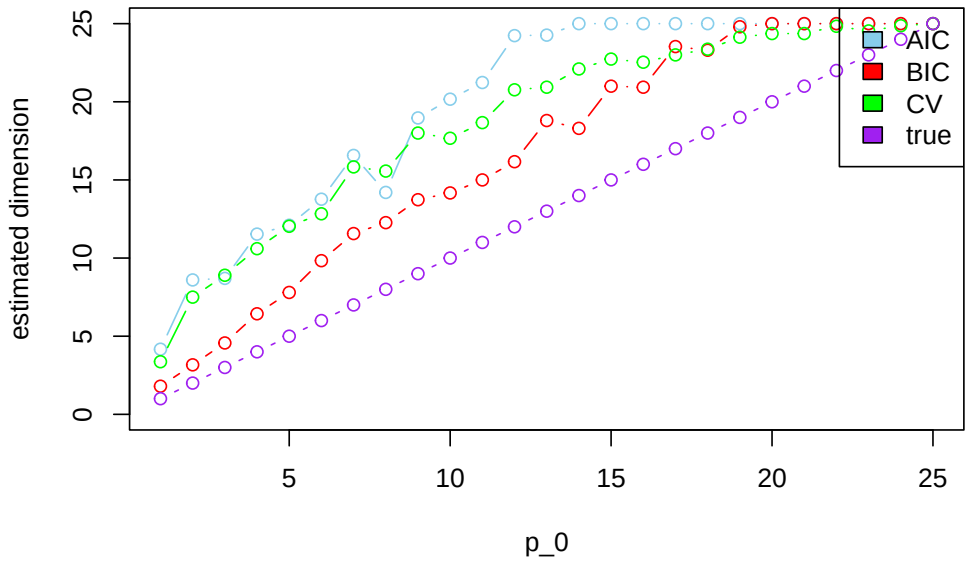
```
  #   dim_AIC = c(rep(NA, prange[1]-1)),   ##Average dimension of AIC
  #   dim_BIC = c(rep(NA, prange[1]-1)),   ##of BIC
  #   dim_CV = c(rep(NA, prange[1]-1)),    ##of CV
  #   dim_true = c(rep(NA, prange[1] - 1))
  # )
  result_dim_var_np = data.frame(
    p = c(),
    dim_AIC = c(),
    dim_BIC = c(),
    dim_CV = c(),
    dim_true = c()
  )
  for(s in 1:length(prange)){
    results_dim <- GetDimComparison(n=n, p = prange[s], B = B, N= N, p0 = p0, sigma =
    new_row_result_dim_var_np = data.frame(
      p = prange[s],
      dim_AIC = mean(results_dim$AIC_dim, na.rm=TRUE),
      dim_BIC = mean(results_dim$BIC_dim, na.rm=TRUE),
      dim_CV = mean(results_dim$CV_dim, na.rm=TRUE),
      dim_true = p0
    )
    result_dim_var_np <- rbind(result_dim_var_np, new_row_result_dim_var_np)

    #disp(s, "/", length(prange))
  }
  return(result_dim_var_np)
}
```

We compute estimated dimension with fixed n,p,sigma^2 for varying p_0

```
krange = 1:25
dim_AIC_k = c(rep(0,length(krange)))
dim_BIC_k = c(rep(0,length(krange)))
dim_CV_k = c(rep(0,length(krange)))
dim_true_k = krange

for(i in 1:length(krange)){
  new_row_var_spars <- GetDimComparison(n=100, p = 25, B = 50, N= 30, p0 = krange[i])
  dim_AIC_k[i] = mean(new_row_var_spars$AIC_dim, na.rm=TRUE)
  dim_BIC_k[i] = mean(new_row_var_spars$BIC_dim, na.rm=TRUE)
  dim_CV_k[i] = mean(new_row_var_spars$CV_dim, na.rm=TRUE)
  disp(i, "/", max(krange))
}
```

```
plot(y=dim_AIC_k, x=krange, xlab="p_0", ylab="estimated dimension", ylim = c(0,25), co
points(y=dim_BIC_k, x=krange, xlab="p_0", ylab="estimated dimension", col = "red", typ
points(y=dim_CV_k, x=krange, xlab="p_0", ylab="estimated dimension", col = "green", ty
points(y=dim_true_k, x=krange, xlab="p_0", ylab="estimated dimension", col = "purple",
```

```
legend(x="topright", legend=c("AIC", "BIC", "CV", "true"), fill=c("skyblue", "red", "g
```

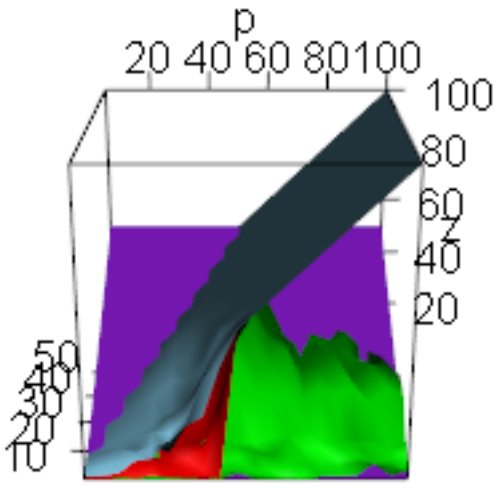



```
results_dim_glob = data.frame(
  k = c(),
  comp_criterion = c()
)


n = 50
krange = c(1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50)
prange = c(5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95,

for(k in krange){
  new_row_dim = data.frame(
    k = k,
    comp_criterion = CompareDimCriterionVarNP(n = n, prange = prange, p0 = k)
  )
  results_dim_glob <- rbind(results_dim_glob, new_row_dim)
  disp(k,"/", max(krange))
}
```

And we plot the results

```
##Here we do the plotting
print(length(results_dim_glob$comp_criterion.dim_CV))
y = krange
x = prange
```

```
print(length(x)*length(y))
z = matrix(data = results_dim_glob$comp_criterion.dim_AIC, nrow = length(x), ncol = le
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p", col="skyblue", typ
z = matrix(data = results_dim_glob$comp_criterion.dim_BIC, nrow = length(x), ncol = le
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p", col="red", type =
z = matrix(data = results_dim_glob$comp_criterion.dim_CV, nrow = length(x), ncol = len
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p",  col="green", add
z = matrix(data = results_dim_glob$comp_criterion.dim_true, nrow = length(x), ncol = l
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "p",  col="purple", add
rgl.snapshot("comp_all_crits_dim.png")
```




We can observe that when $p = p0$, then all criterions perform equally. This probably comes from (TODO : check and add details) the fact when the sparsity of the input vector is constant and $p = p0$, hence AIC and BIC can be proved equivalent. Also, in the Gaussian case this certainly amounts to solving a least square to find the CV solution which is the same as for AIC or BIC.

We can also take a closer look at what happens in the sparse low dimensional area:

```
results_dim_glob = data.frame(
  k = c(),
  comp_criterion = c()
)


n = 50
krange = 1:25
prange = 2*(2:25)
```

```
for(k in krange){
  new_row_dim = data.frame(
    k = k,
    comp_criterion = CompareDimCriterionVarNP(n = n, prange = prange, p0 = k)
  )
  results_dim_glob <- rbind(results_dim_glob, new_row_dim)
  disp(k, "/", max(krange))
}
```

```
##Here we do the plotting
print(length(results_dim_glob$comp_criterion.dim_CV))
x = krange
y = prange

print(length(x)*length(y))
z = matrix(data = results_dim_glob$comp_criterion.dim_AIC, nrow = length(y), ncol = le
persp3d(x=y, y=x,z=z, ylab = "number of nonzero coefs", xlab = "p", col="skyblue", typ
z = matrix(data = results_dim_glob$comp_criterion.dim_BIC, nrow = length(y), ncol = le
persp3d(x=y, y=x,z=z, ylab = "number of nonzero coefs", xlab = "p", col="red", type =
z = matrix(data = results_dim_glob$comp_criterion.dim_CV, nrow = length(y), ncol = len
persp3d(x=y, y=x,z=z, ylab = "number of nonzero coefs", xlab = "p",  col="green", add
z = matrix(data = results_dim_glob$comp_criterion.dim_true, nrow = length(y), ncol = l
persp3d(x=y, y=x,z=z, ylab = "number of nonzero coefs", xlab = "p",  col="purple", add
rgl.snapshot("comp_all_crits_low_dim_sparse.png")
```

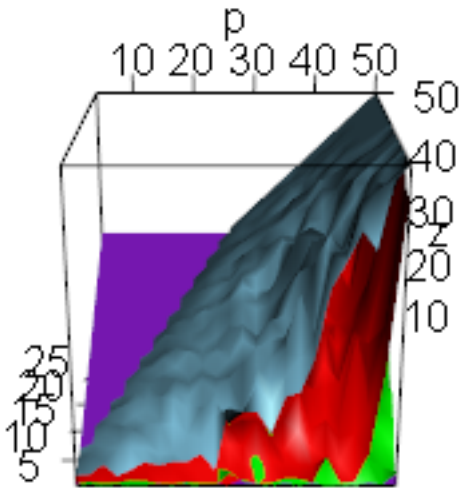



Figure 3: Comparison of the dimension identified depending on dimensionality and sparsity for sparse low dimension

#Influence of noise

We are now left with having to measure the influence of the noise. In order to do that we will plot graphs of dimensionality identified (or distance w.r.t true $\lambda$) where the parameters are $(\sigma, n/p)$ or $(\sigma, k)$. #Dimensionality We compute dimension of the estimated model w.r.t to criterions for varying *sigma* and $k$

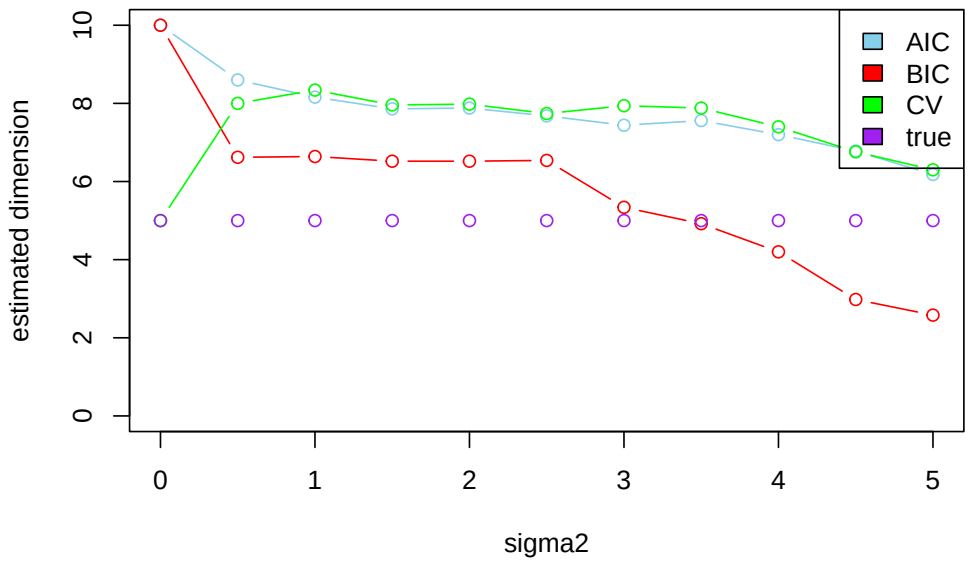
```
CompareDimCriterionVarSigma <- function(
                                  n = 100,
                                  p = 10,
                                  sigmarange      = (0:20)/10, #List of sigma tested
                                  p0      = 5,  #Number of nonzero coefficients
                                  sigma  = 1,
                                  sigmaX = 1,
                                  b0     = 1,
                                  B = 30, #Number of lambda tested
                                  N = 10, #Number of instances of solving Y = X\theta +
                                  lambda_step = 0.05, #Difference between two consecutiv
                                  lambda_init = 0.0 #First value used by lambda
                                    ){
  result_dim_var_sigma = data.frame(
    sigma = c(),#c(rep(prange[1], max(prange[1]-1, 0))),
    dim_AIC = c(), #c(rep(NA, prange[1]-1)),   ##Average dimesnsion of AIC
    dim_BIC = c(), #c(rep(NA, prange[1]-1)),   ##of BIC
    dim_CV = c(),#c(rep(NA, prange[1]-1)),    ##of CV
    dim_true = c()#c(rep(NA, prange[1] - 1))
  )
  for(s in sigmarange){
    results_dim_sigma <- GetDimComparison(n=n, p = p, sigma = s, B = B, N= N, p0 = p0,
    new_row_result_dim_var_sigma = data.frame(
      sigma = s,
      dim_AIC = mean(results_dim_sigma$AIC_dim, na.rm=TRUE),
      dim_BIC = mean(results_dim_sigma$BIC_dim, na.rm=TRUE),
      dim_CV = mean(results_dim_sigma$CV_dim, na.rm=TRUE),
      dim_true = p0
    )
    result_dim_var_sigma <- rbind(result_dim_var_sigma, new_row_result_dim_var_sigma)

    #print(s)
  }
  return(result_dim_var_sigma)
}
```

We compute for varying $\sigma^2$, fixed $p_0 = 5$ the dimension estimated

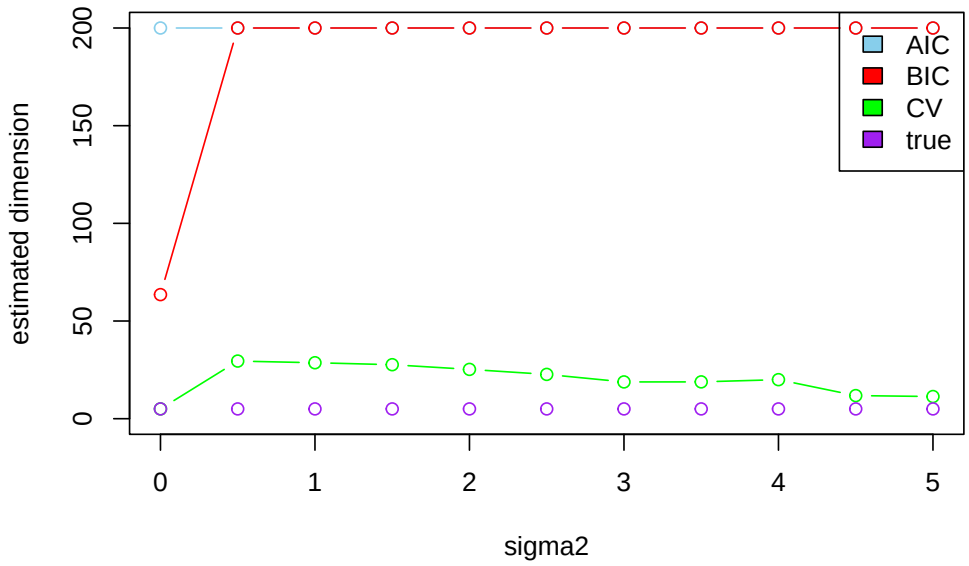
```
sigmarange = c(0.0, 0.5, 1.0, 1.50, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0)
varSigmaLowDim= CompareDimCriterionVarSigma(n = 100, p = 10, p0 = 5, N = 50, B = 50, s
varSigmaHighDim= CompareDimCriterionVarSigma(n = 100, p = 200, p0 = 5, N = 50, B = 50,
```

And we plot

```
plot(y=varSigmaLowDim$dim_AIC, x=sigmarange, xlab="sigma2", ylab="estimated dimension"
points(y=varSigmaLowDim$dim_BIC, x=sigmarange, col = "red", type="b")
points(y=varSigmaLowDim$dim_CV, x=sigmarange, col = "green", type="b")
points(y=varSigmaLowDim$dim_true, x=sigmarange, col = "purple", type="p")
legend(x="topright", legend=c("AIC", "BIC", "CV", "true"), fill=c("skyblue", "red", "g
```




```
plot(y=varSigmaHighDim$dim_AIC, x=sigmarange, xlab="sigma2", ylab="estimated dimension
points(y=varSigmaHighDim$dim_BIC, x=sigmarange, col = "red", type="b")
points(y=varSigmaHighDim$dim_CV, x=sigmarange, col = "green", type="b")
points(y=varSigmaHighDim$dim_true, x=sigmarange, col = "purple", type="p")
legend(x="topright", legend=c("AIC", "BIC", "CV", "true"), fill=c("skyblue", "red", "g
```

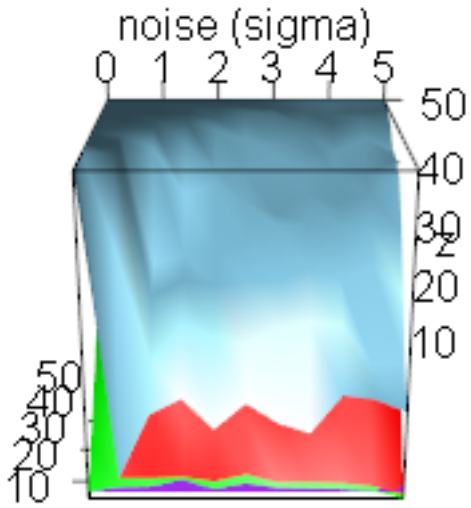



```
n<-50
p<-200
results_dim_glob_sigma = data.frame(
  k = c(),
  comp_criterion = c()
)
sigmarange = c(0.0, 0.5, 1.0, 1.50, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0)
krange = c(5, 10, 15, 20, 25, 30, 35, 40, 45, 50)

for(k in krange){
  new_row_dim_sigma = data.frame(
    k = k,
    comp_criterion = CompareDimCriterionVarSigma(n = n, p = p, p0 = k, sigmarange = si
  )
  results_dim_glob_sigma <- rbind(results_dim_glob_sigma, new_row_dim_sigma)
  disp(k, "/", max(krange))
}
```

And we plot the results

```
print(length(results_dim_glob_sigma$comp_criterion.dim_CV))
y = krange
x = sigmarange
print(length(x)*length(y))
z = matrix(data = results_dim_glob_sigma$comp_criterion.dim_AIC, nrow = length(y), ncc
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "noise (sigma)", col="s
z = matrix(data = results_dim_glob_sigma$comp_criterion.dim_BIC, nrow = length(y), ncc
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "noise (sigma)", col="r
```

```
z = matrix(data = results_dim_glob_sigma$comp_criterion.dim_CV, nrow = length(y), ncol
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "noise (sigma)",  col="
z = matrix(data = results_dim_glob_sigma$comp_criterion.dim_true, nrow = length(y), nc
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "noise (sigma)",  col="
rgl.snapshot("comp_all_crits_dim_var_sigma_high_dim.png")
```




We can make several observations from the plot in a low-dimension setting (for n = 50, p = 100, sigmarange =

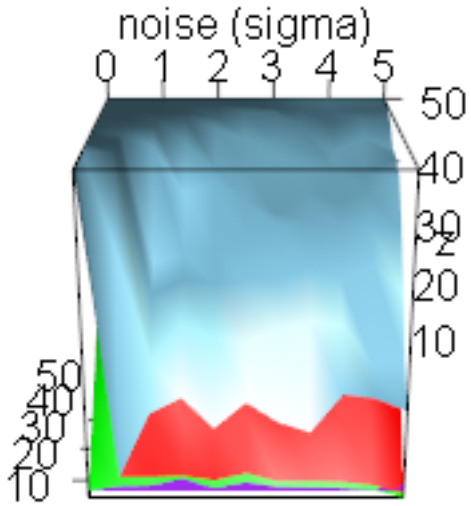
$$0,5$$

) : 1) All criterions overfit (include too many parameters) the data. CV is the best among three as it consistently identifies a dimension between 2 and 3 times the true dimension. 2) There is a cliff given by some $\sigma_i = \phi_i(p_0)$ AIC and BIC perform nearly no dimension reduction (BIC makes dimension reduction soon) which is for small noise (!) 3) In a situation without noise CV is the best, when noise is quite large BIC performs equivalently to CV (but is faster)

```
n<-400
p<-50
results_dim_glob_sigma = data.frame(
  k = c(),
  comp_criterion = c()
)
sigmarange = c(0.0, 0.5, 1.0, 1.50, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0)
krange = c(5, 10, 15, 20, 25, 30, 35, 40, 45, 50)

for(k in krange){
  new_row_dim_sigma = data.frame(
    k = k,
```

```
    comp_criterion = CompareDimCriterionVarSigma(n = n, p = p, p0 = k, sigmarange = si
  )
  results_dim_glob_sigma <- rbind(results_dim_glob_sigma, new_row_dim_sigma)
  disp(k, "/", max(krange))
}
```

```
print(length(results_dim_glob_sigma$comp_criterion.dim_CV))
y = krange
x = sigmarange
print(length(x)*length(y))
z = matrix(data = results_dim_glob_sigma$comp_criterion.dim_AIC, nrow = length(y), ncc
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "noise (sigma)", col="s
z = matrix(data = results_dim_glob_sigma$comp_criterion.dim_BIC, nrow = length(y), ncc
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "noise (sigma)", col="r
z = matrix(data = results_dim_glob_sigma$comp_criterion.dim_CV, nrow = length(y), ncol
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "noise (sigma)",  col="
z = matrix(data = results_dim_glob_sigma$comp_criterion.dim_true, nrow = length(y), nc
persp3d(x=x, y=y,z=z, ylab = "number of nonzero coefs", xlab = "noise (sigma)",  col="
rgl.snapshot("comp_all_crits_dim_var_sigma_low_dim.png")
```



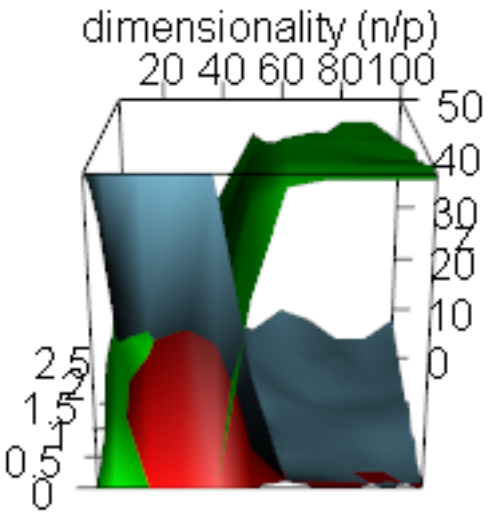

We can make several observations from the plot (for n = 50, p = 100, sigmarange = $0, 5$ ) :

#Closeness to $lambda_{opt}$

```
n<-50
results_glob_sigma_np = data.frame(
  sigma = c(),
```

```
  comp_criterion = c()
)
sigmarange = c(0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.50, 1.75, 2.0, 2.5)
prange = c(5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
for(sigma in sigmarange){
  new_row_sigma_np = data.frame(
    sigma = sigma,
    comp_criterion = CompareCriterionVarNP(n = n, prange = prange, p0 = 5, sigma = sig
  )
  results_glob_sigma_np <- rbind(results_glob_sigma_np, new_row_sigma_np)
  disp(sigma, "/", max(sigmarange))
}
```

```
print(length(results_glob_sigma_np$comp_criterion.num_CV_best))
x = prange
y = sigmarange
print(length(x)*length(y))
z = matrix(data = results_glob_sigma_np$comp_criterion.num_AIC_best, nrow = length(y),
persp3d(x=x, y=y,z=z, xlab = "dimensionality (n/p)", ylab = "noise (sigma)", col="skyb
z = matrix(data = results_glob_sigma_np$comp_criterion.num_BIC_best, nrow = length(y),
persp3d(x=x, y=y,z=z, xlab = "dimensionality (n/p)", ylab = "noise (sigma)", col="red"
z = matrix(data = results_glob_sigma_np$comp_criterion.num_CV_best, nrow = length(y),
persp3d(x=x, y=y,z=z, xlab = "dimensionality (p)", ylab = "noise (sigma)",  col="green
rgl.snapshot("comp_all_crits_var_sigma_p.png")
```



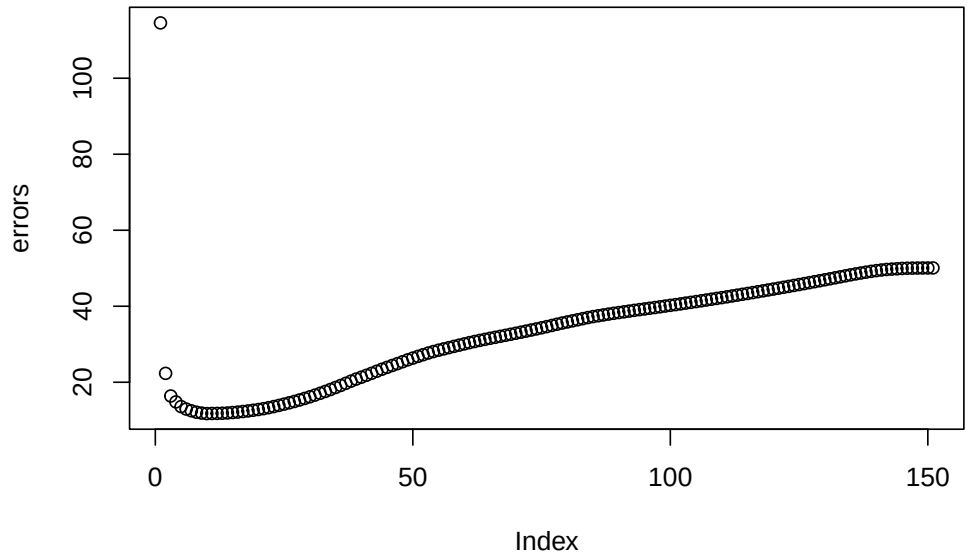

At the moment, this graph makes not much sense. It might be better to plot something like $\ell^2$ distance between $\beta_{\lambda_{crit}}$ and $\beta^*$.

#Cross validation

```
cross_validation <- function(Y, X, nfolds = 10, lambdarange = (0:20)/10){
  err_fold = c(rep(0, length(lambdarange)))
  train_indices = cross_validation_get_indices(1:length(Y), nfolds = nfolds)
  min_err = NA
  min_index = NA
  for(lambda_index in 1:length(lambdarange)){
    for(cur_fold in 1:nfolds){

        #print(X[train_indices[[cur_fold]],])
        bh = glmnet(x= X[train_indices[[cur_fold]],], y = Y[train_indices[[cur_fold]]]
        #print(bh)
        test_indices = !train_indices[[cur_fold]]
        #print( X[test_indices,]%*%bh)
        err_fold[lambda_index] = err_fold[lambda_index] + t(Y[test_indices] - X[test_i
    }

    err_fold[lambda_index] = err_fold[lambda_index]/nfolds

    if(is.na(min_err) || err_fold[lambda_index] <= min_err ){
      min_err = err_fold[lambda_index]
      min_index = lambda_index
      min_lambda = lambdarange[lambda_index]
    }
  }


  return(list("errors" = err_fold, "min_err" = min_err, "min_index" = min_index, "min_
}

cross_validation_get_indices <- function(index_range, nfolds = 10){
  len <- length(index_range)
  all_indices = list(rep(TRUE, length(Y)))
  index_list = rep(all_indices, nfolds)
  offset = len/nfolds
  for( k in 1:nfolds){
    for(i in ((k-1)*offset+1):(k*offset)){
      index_list[[k]][i] = FALSE
    }
  }
  return(index_list)
}
```

```
n       = 100
p       = 100
p0      = 5
sigma   = 1
sigmaX  = 1
```

```
b0      = 1
beta0   = c(rep(b0,p0),rep(0,p-p0))
X       = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
Y       = X%*%beta0 + rnorm(n,0,sigma)

ret <- cross_validation(Y=Y, X=X, nfolds = 10, lambdarange = (0:150)/100)
errors<-ret$errors
plot(errors)
```




```
disp("Value of lambda_cv of our own implementation:",ret$min_lambda)
```

```
## Value of lambda_cv of our own implementation: 0.09
```

```
disp("Value of lambda of cv.glmnet:", cv.glmnet(y=Y, x=X, type.measure = "mse", nfolds
```

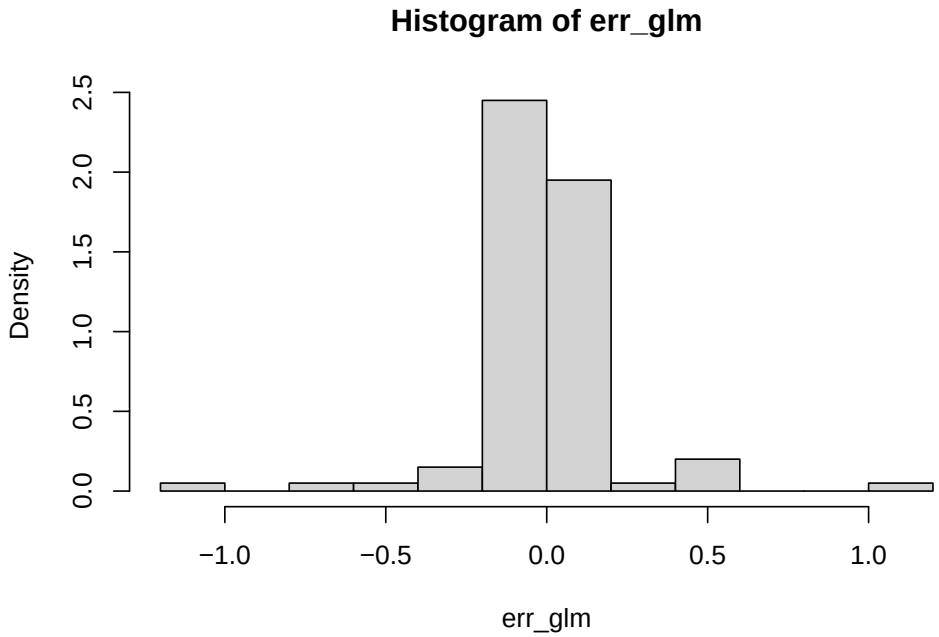
```
## Value of lambda of cv.glmnet: 0.13
```

We check that on average (varying $n, p$ and $\sigma^2$) we indeed have the same results as cv.glmnet for our own implementation of cross validation. We also compute running times of both implementations.

```
N<-100
err_glm = c(rep(0,N))
time_own = 0.0
time_glm = 0.0
for(i in 1:N){
  n       = as.integer(sample(c(30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,1
  p       = as.integer(sample(10:n, size = 1))
  p0      = 5
```

```
  sigma    = sample(sample(0:5),size = 1)
  sigmaX   = 1
  b0       = 1
  beta0    = c(rep(b0,p0),rep(0,p-p0))
  X        = sapply(1:p, FUN=function(x){rnorm(n,0,sigmaX)})
  Y        = X%*%beta0 + rnorm(n,0,sigma)

  step_time = Sys.time()
  ret_own = cross_validation(Y=Y, X=X, nfolds = 10, lambdarange = (0:200)/100)
  step_time = Sys.time() - step_time
  time_own = time_own + step_time

  step_time = Sys.time()
  ret_glm = cv.glmnet(y=Y, x=X, nfolds = 10, family = "gaussian", lambda = (0:150)/100
  step_time = Sys.time() - step_time
  time_glm = time_glm + step_time

  err_glm[i] = ret_own$min_lambda - ret_glm$lambda.min

  disp(i, "/", N)
}
```

```
hist(err_glm, freq = FALSE)
```




```
disp("mean error between our implemention and cv.glmnet:", mean(err_glm))
```

```
## mean error between our implemention and cv.glmnet: 0.0068
```

```
disp("time spent in our own implementation:", time_own)
```

```
## time spent in our own implementation: 182.3179
```

```
disp("time spent in cv.glmnet implementation:", time_glm)
```

```
## time spent in cv.glmnet implementation: 8.145465
```

```
disp("ratio of times own/glm:",as.double(time_own)/as.double(time_glm))
```

```
## ratio of times own/glm: 22.38275
```

# 6 Overview of the project

You will first implement the cross validation and the BIC to calibrate the Lasso numerically in the regression setting. Then you will study the statistical properties of the lasso in terms of model selection in different simulation setting (high signal / high noise, low dimensional / high dimensional). You will compete CV, the AIC, and the BIC in order to determine if one procedure is more accurate than the other.

# 7 Reference

Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition, Springer, 2009