Focused and weigthed cross-validation

Martin Jullum & Kristoffer Hellton

September 21, 2016

Contour curves for tuning parameter

We aim to contruct a focused cross-validation scheme for finding tuning parameters in ridge regression. This will be extended to other methods later. Consider the data matrix X and outcome vector Y related by the linear regression model $Y = X\beta + \epsilon$. We use ridge regression

$$\hat{\beta}(\lambda) = (X^T X + \lambda I)^{-1} X^T Y,$$

to estimate β and wish to fine-tuning λ using cross-validation, but with a specific prediction at the covariates x_0 as focus. The focus parameter is given as $\hat{\mu} = x_0^T \hat{\beta}(\lambda)$. Consider the oracle prediction error of x_0, defined as

$$OPE(\lambda; x_0) = (x_0^T (X^T X + \lambda I)^{-1} X^T Y - x_0^T \beta)^2 = (x_0^T (\hat{\beta}(\lambda) - \beta))^2.$$

The minimizer of the oracle prediction error can be view as an optimal focused (unknown) tuning parameter:

$$\lambda_x 0 = \arg\min_{\lambda} OPE(\lambda; x_0).$$

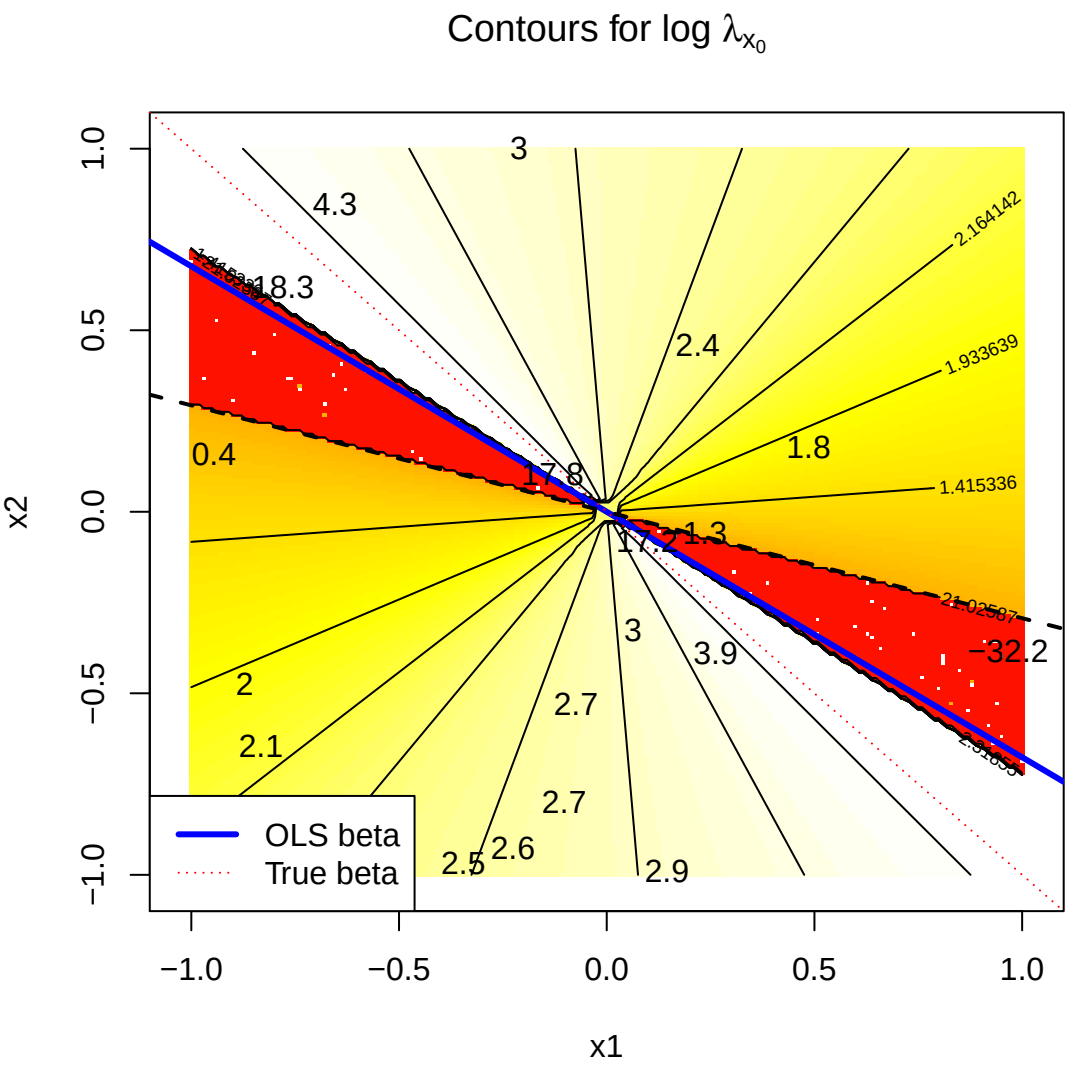
The distribution of $\lambda_x 0$ on a log in the data space ($p = 2$) for a random data matrix and outcome vector is as follows:

```
# Defining a true model: no intercept, require centering of outcome
p <- 2
sigma2 <- 2
beta <-rep(1,2)
n <- 50

#Drawing data matrix and focus from uniform distribution. Oracle, no distributed outcome?
set.seed(111)
X <- scale(matrix(runif(p*n,min=-1,max=1),ncol=p),center = TRUE, scale = FALSE)
set.seed(111)
Y <- X%*%beta + scale(rnorm(n,mean=0,sd=sqrt(sigma2)),center = TRUE, scale = FALSE)

PredictionErrorFocus <- function(x1,x2){
  x0 <- c(x1,x2)
  optim(par = 10,function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y
      - x0%*%beta)^2},lower=0,upper = Inf, method = "L-BFGS-B",control = list(factr=1e6))$par
}

betaOLS <- solve(crossprod(X,X))%*% t(X)%*%Y
x1 <- seq(-1,1,length.out = 200)
x2 <- seq(-1,1,length.out = 200)
VecFun <- Vectorize(PredictionErrorFocus)
z <- outer(x1,x2,VecFun)+10^-10
#hist(z)
#hist(log(z))

image(x1,x2,log(z),col=heat.colors(99),breaks=quantile(log(z),
```

```
  probs = exp(seq(log(0.01), log(0.9), length.out = 100))),
  main = expression(paste("Contours for log ", lambda[x[0]])),
  xlim = c(-1.1,1.1),ylim = c(-1.1,1.1),axes=TRUE)
```

```
## Warning in image.default(x1, x2, log(z), col = heat.colors(99), breaks =
## quantile(log(z), : unsorted 'breaks' will be sorted before use
```

```
contour(x1,x2,log(z),levels=quantile(log(z),probs = seq(0.2, 0.9, 0.1)),add=T,method='edge')
contour(x1,x2,log(z),levels=c(min(log(z)+2)),add=T,method='edge')
abline(a=0,b=-(betaOLS[1])/betaOLS[2],col='blue',lwd=3)
abline(a=0,b=-beta[1]/beta[2],col='red',lwd=1,lty=3)
text(X[1:20,],label=round(log(apply(X[1:20,],1,
                function(x){PredictionErrorFocus(x[1],x[2])})+10^-14),1))
legend('bottomleft',legend = c('OLS beta','True beta'),
       lty = c(1,3),lwd = c(3,1),col=c('blue','red'),bg='white' )
abline(a=0,b=-(betaOLS[1]-beta[1])/(betaOLS[2]-beta[2]),col='black',lwd=2,lty=2)
```



Red color indicates a tuning parameter equal to 0, and white close or equal to infity. The exact optimal tuning parameter on log scale are given for the 20 first observation at their position.

Surprise number 1

It is clear that the contours of distribution are straight lines, but the figure shows that the oracle tuning parameter remains unchange along straight lines *going through the origin*, not parallell to zero prediction line marked in dashed red. What is *surprising* is that distribution is **NOT** even or smooth, but infact discontinuous along a line. The line seems to coincide with $x^T\beta_{OLS} = 0$ given by

$$x_2 = -\frac{\hat{\beta}_{OLS,1}}{\hat{\beta}_{OLS,2}}x_1,$$

but **NOT EXACTLY**. Is this due to an computing error or approximation somewhere, or can we find an exact expression for this line?

On one side of this line the optimal tuning parameter is 0, while one the other it is infinity. The sides swith at the origin. On either side of this discontinuity, the oracle predictions curves as functions of λ look as follows:

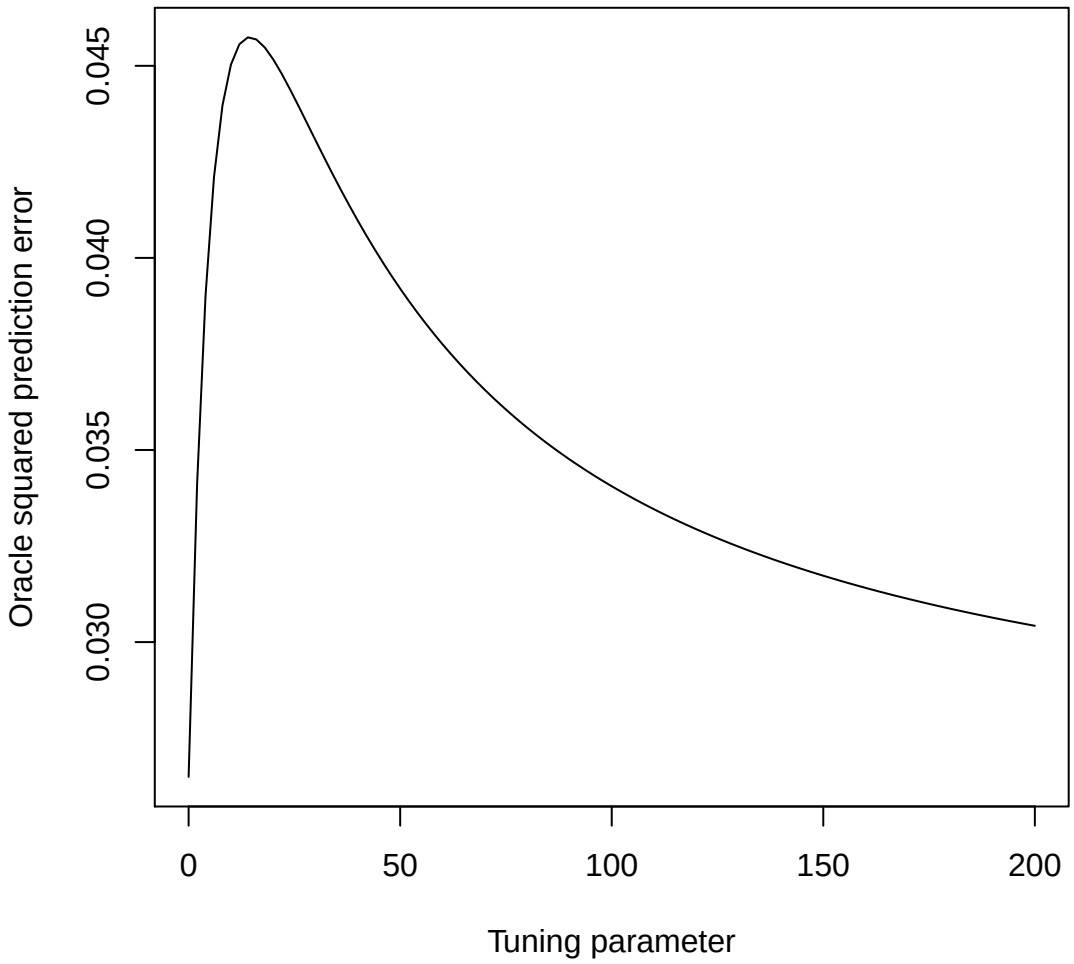
```
betaOLS
```

```
##          [,1]
## [1,] 1.247855
## [2,] 1.845468
```

```
## x0 in the zero area
x0 <- c(0.5,-betaOLS[1]/betaOLS[2]*0.5-0.001)
x0
```

```
## [1]  0.5000000 -0.3390862
```

```
curve(sapply(x,function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y- x0%*%beta)^2})
      ,0,200,ylab='Oracle squared prediction error',xlab = 'Tuning parameter')
```



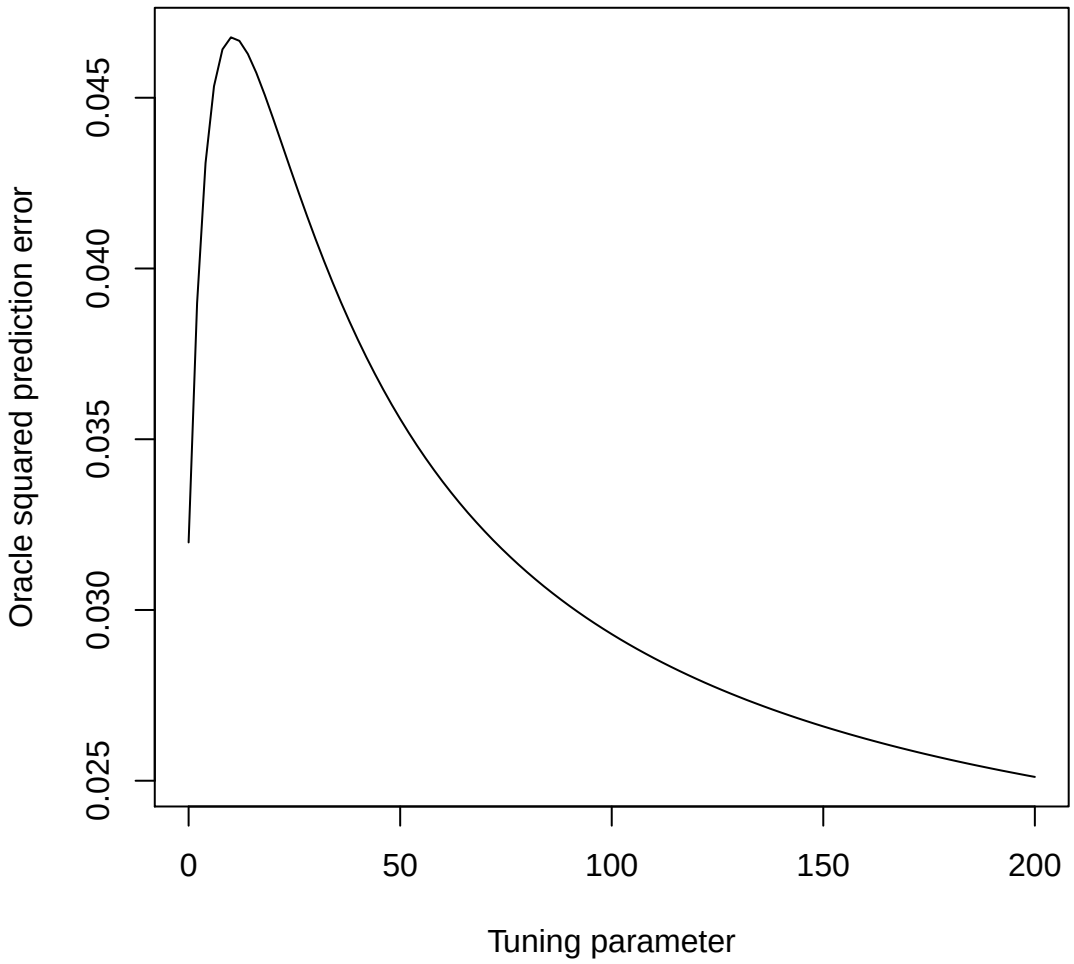

```
#Calculating the minimand
optim(par = 10,
      function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y- x0%*%beta)^2},
      lower=0,upper = Inf, method = "L-BFGS-B",control = list(factr=1e6))$par
```

```
## [1] 0
```

```
## x0 in the "infinity" area
x0 <- c(0.5,-betaOLS[1]/betaOLS[2]*0.5-0.02)
x0
```

```
## [1]  0.5000000 -0.3580862
```

```
curve(sapply(x,function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y- x0%*%beta)^2}),
      0,200,ylab='Oracle squared prediction error',xlab = 'Tuning parameter')
```




```
#Calculating the minimand
optim(par = 100,
      function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y- x0%*%beta)^2},
      lower=0,upper = Inf, method = "L-BFGS-B",control = list(factr=1e6))$par
```

```
## [1] 26184201
```

Surprise number 2

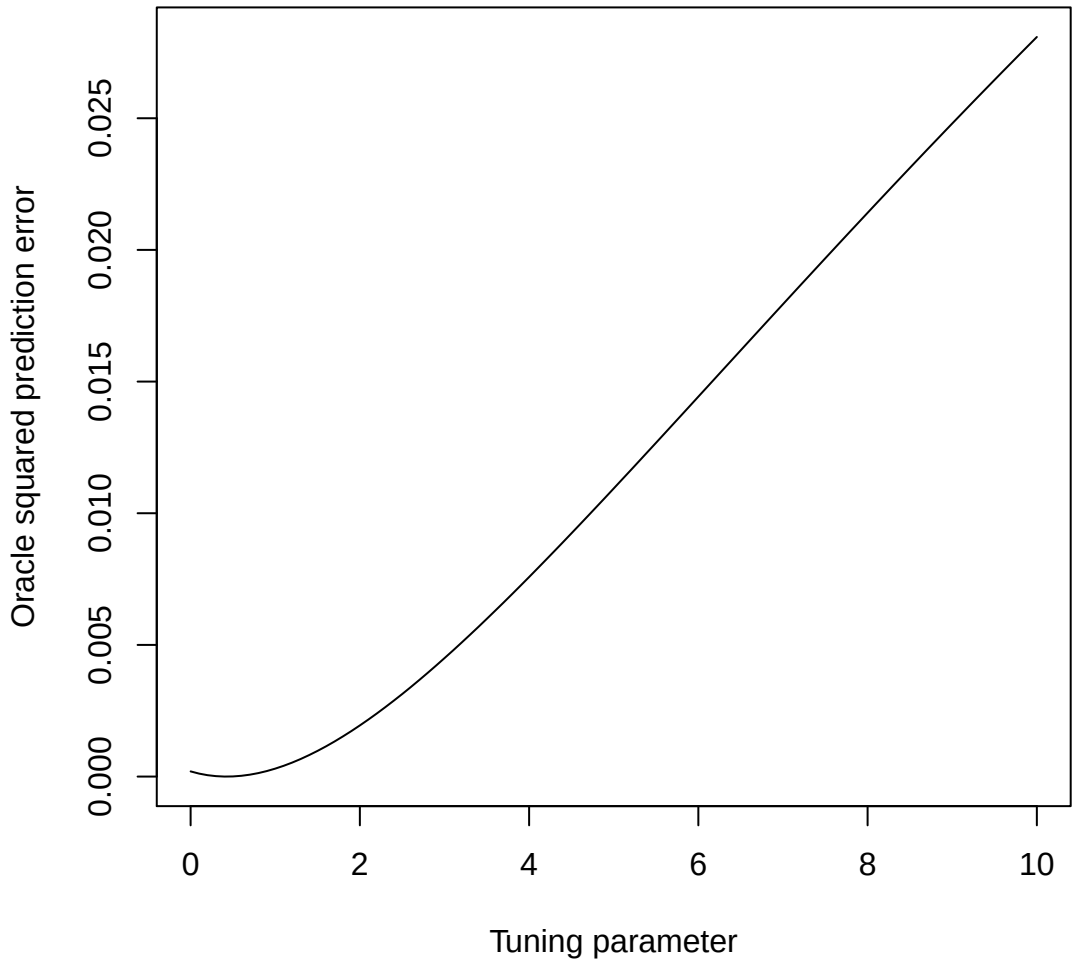
In addition(!), there exists a line, which together with the OLS line, defines a "zero" area, the red cone, where the optimal tuning parameter is exactly zero. This lines is not defined by the true β zero prediction line $x^T\beta = 0$, how is it defined???

On the edge of this area, along the unknown line the oracle prediction are continuous and the squared prediction error curves as functions of λ look as follows:

```
## x0 with a minimum away from zero
x0 <- c(0.5,-0.13)
x0
```

```
## [1]  0.50 -0.13
```

```
curve(sapply(x,function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y- x0%*%beta)^2}),
      0,10,ylab='Oracle squared prediction error',xlab = 'Tuning parameter')
```



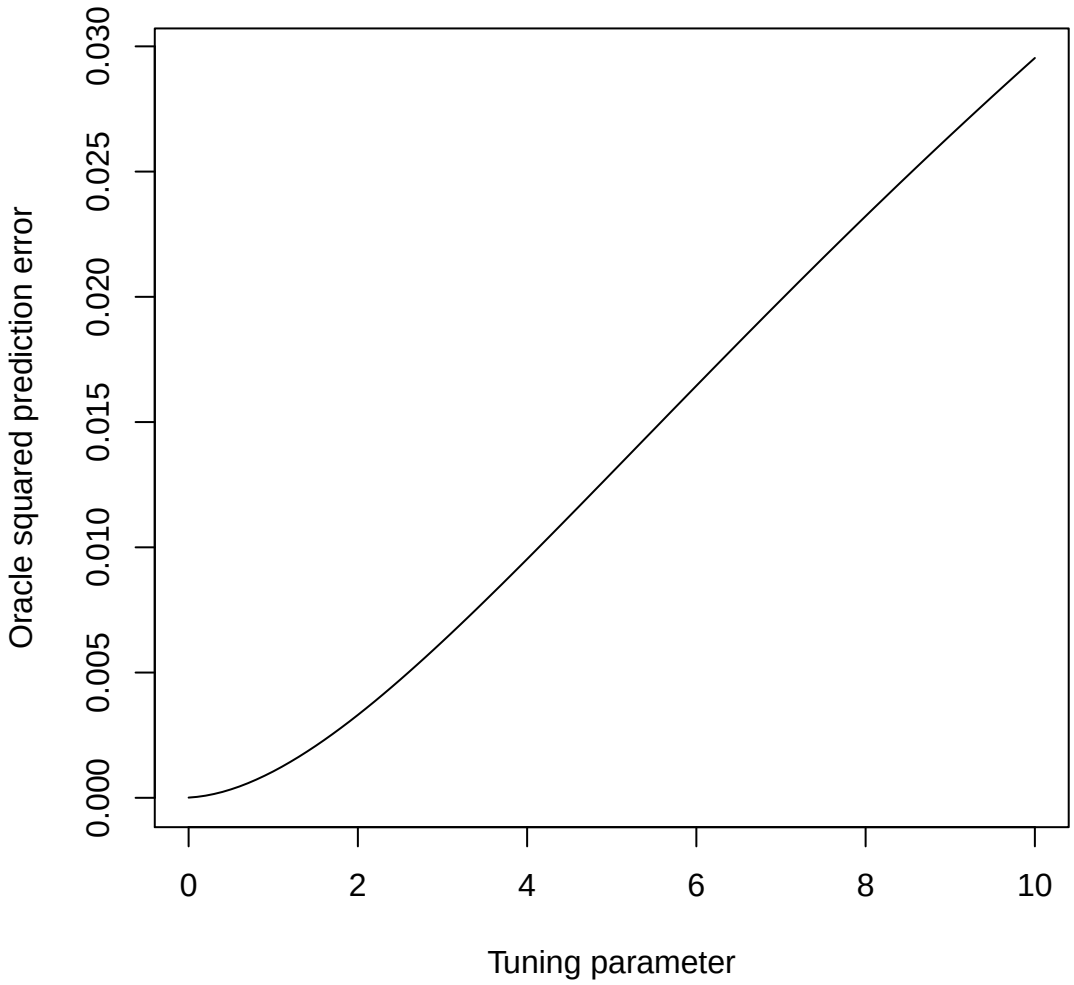

```
#Calculating the minimand
optim(par = 100,
      function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y- x0%*%beta)^2},
      lower=0,upper = Inf, method = "L-BFGS-B",control = list(factr=1e6))$par
```

```
## [1] 0.4259488
```

```
## x0 with minimum at zero
x0 <- c(0.5,-0.15)
x0
```

```
## [1]  0.50 -0.15
```

```
curve(sapply(x,function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y- x0%*%beta)^2})
      ,0,10,ylab='Oracle squared prediction error',xlab = 'Tuning parameter')
```




```
#Calculating the minimand
optim(par = 10,
      function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y- x0%*%beta)^2},
      lower=0,upper = Inf, method = "L-BFGS-B",control = list(factr=1e6))$par
```

```
## [1] 0
```

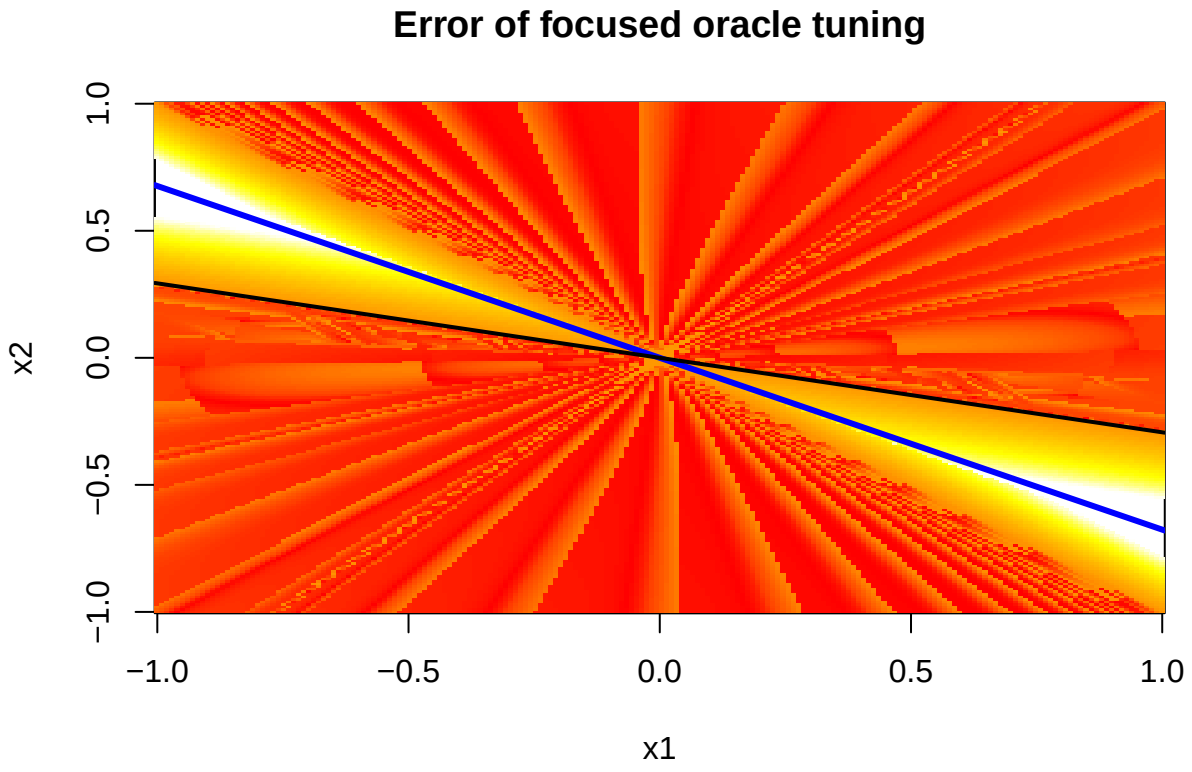
Quantification of the benefit of focussing

To be able to quantify how much better a focused cross-validation can preform than ordinary cross-validation, we start by quantifying how much better one can prediction if one knew the oracle.

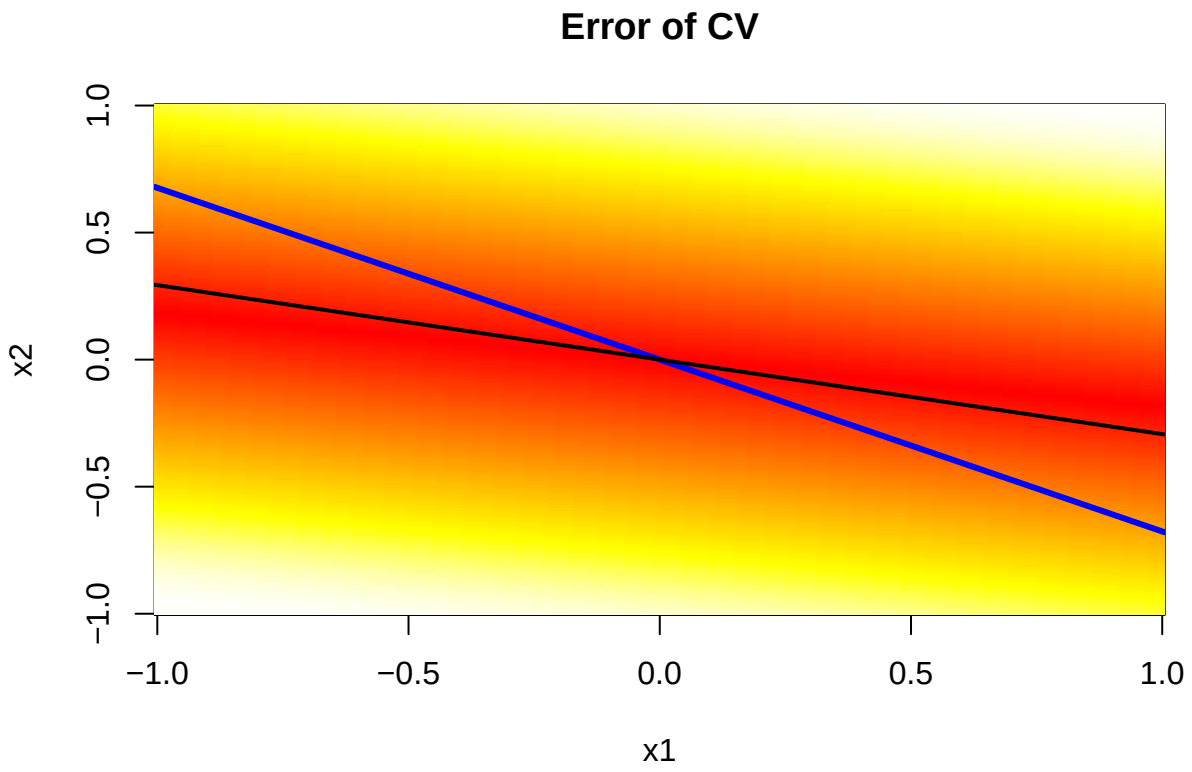
To quantify the overall behavoir of a tuning based on the focused approach, we test these measures: sample extensively from a normal and uniform focus distribution for a fixed data matrix (this should amount to weighting the error difference for each x_0 with the density of the distribution). For the uniform distribution, this is equivalent to taking the average. A third option; only look at the difference at the observations in the data matrix.

```
## Take the mean first, equivalent to sampling from a uniform distribution
CV <- function(lambda){
  H <- X %*% solve(crossprod(X,X)+lambda*diag(p))%*% t(X)
```

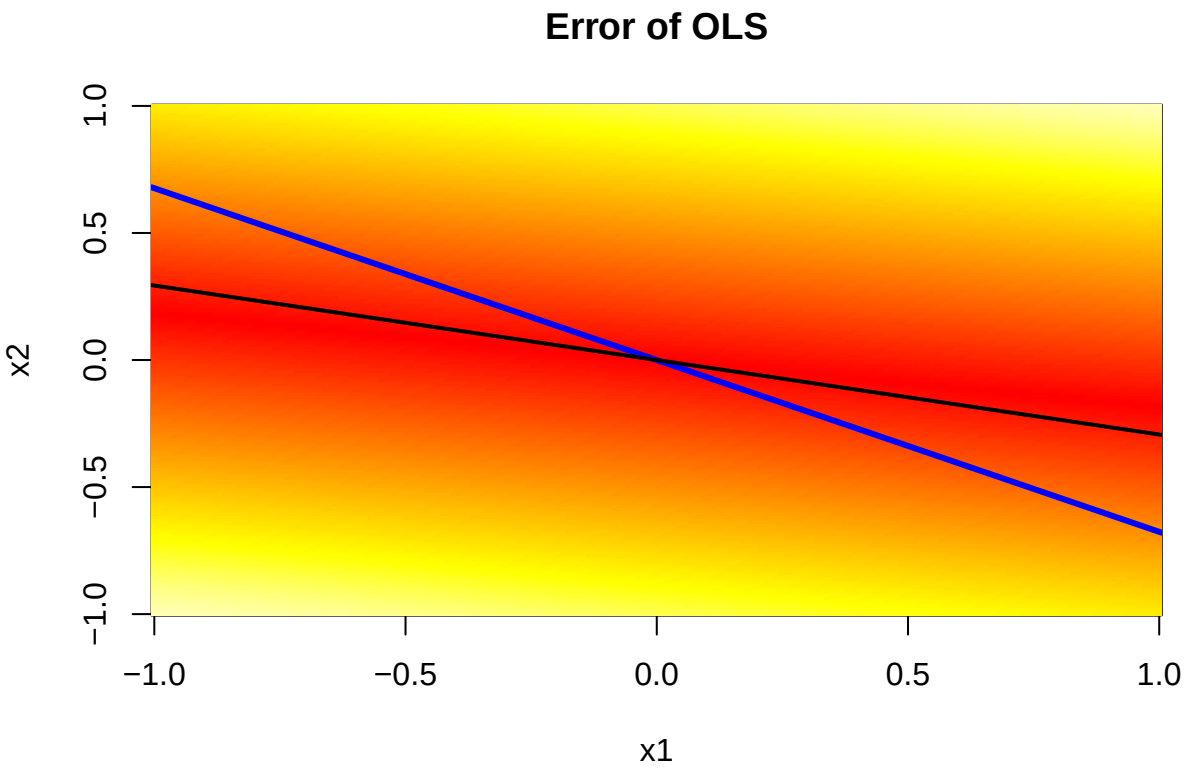
```
  e <- (diag(n) - H) %*% Y
  mean((e/(1-diag(H)))^2)
}

tuning.cv <-optim(par = 100,CV,
      lower=0,upper = Inf, method = "L-BFGS-B",control = list(factr=1e6))$par

ErrorFocus  <- function(x1,x2){
  x0 <- c(x1,x2)
  optim(par = 10,function(lambda){(x0%*%solve(crossprod(X,X)+lambda*diag(p))%*% t(X)%*%Y
      - x0%*%beta)^2},lower=0,upper = Inf, method = "L-BFGS-B",control = list(factr=1e6))$value
}

ErrorCV  <- function(x1,x2){
  x0 <- c(x1,x2)
 (x0 %*% solve(crossprod(X,X)+tuning.cv*diag(p))%*% t(X)%*%Y - x0%*%beta)^2
}

ErrorOLS <- function(x1,x2){
  x0 <- c(x1,x2)
  (x0%*%solve(crossprod(X,X)) %*% t(X)%*%Y- x0%*%beta)^2
}

betaOLS <- solve(crossprod(X,X))%*% t(X)%*%Y
x1 <- seq(-1,1,length.out = 200)
x2 <- seq(-1,1,length.out = 200)

error.focus <- outer(x1, x2, Vectorize(ErrorFocus))
image(x1,x2,error.focus ,col=heat.colors(99),breaks=quantile(error.focus ,probs = 1 - exp(-seq(log(1), l
abline(a=0,b=-(betaOLS[1])/betaOLS[2],col='blue',lwd=3)
abline(a=0,b=-(betaOLS[1]-beta[1])/(betaOLS[2]-beta[2]),col='black',lwd=2)
```




```
error.cv <- outer(x1, x2, Vectorize(ErrorCV))
image(x1,x2,error.cv,col=heat.colors(99),breaks=quantile(error.cv ,probs=seq(0,1, length.out = 100)),ma
abline(a=0,b=-(betaOLS[1])/betaOLS[2],col='blue',lwd=3)
abline(a=0,b=-(betaOLS[1]-beta[1])/(betaOLS[2]-beta[2]),col='black',lwd=2)
```



```
error.ols <- outer(x1, x2, Vectorize(ErrorOLS))
image(x1,x2,error.cv,col=heat.colors(99),breaks=quantile(error.ols ,probs=seq(0,1, length.out = 100)),ma
abline(a=0,b=-(betaOLS[1])/betaOLS[2],col='blue',lwd=3)
abline(a=0,b=-(betaOLS[1]-beta[1])/(betaOLS[2]-beta[2]),col='black',lwd=2)
```




```
#Subtract the squared error of CV from OLS: if positive, the OLS error
#is largest, and average
mean(error.ols-error.cv)
```

```
## [1] 0.0786804
```

```
#OLS has larger error than ridge-cv

#If cv error is larger, the difference is positive
mean(error.cv-error.focus)
```

```
## [1] 0.1793835
```

```
#The percentage of improvement with focused app
mean(error.cv-error.focus)/mean(error.cv)
```

```
## [1] 0.9820116
```