Solution 1:

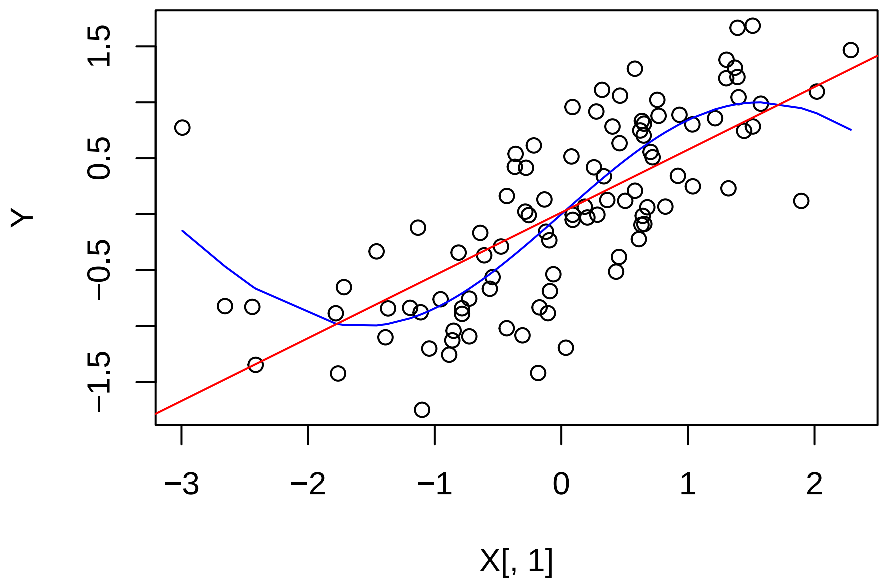
(a)
```r
set.seed(42)
n = 100
p_add = 100
# create matrix of features
X = matrix(rnorm(n * (p_add + 1)), ncol = p_add + 1)

Y = sin(X[,1]) + rnorm(n, sd = 0.5)
```

(b) Demonstration of

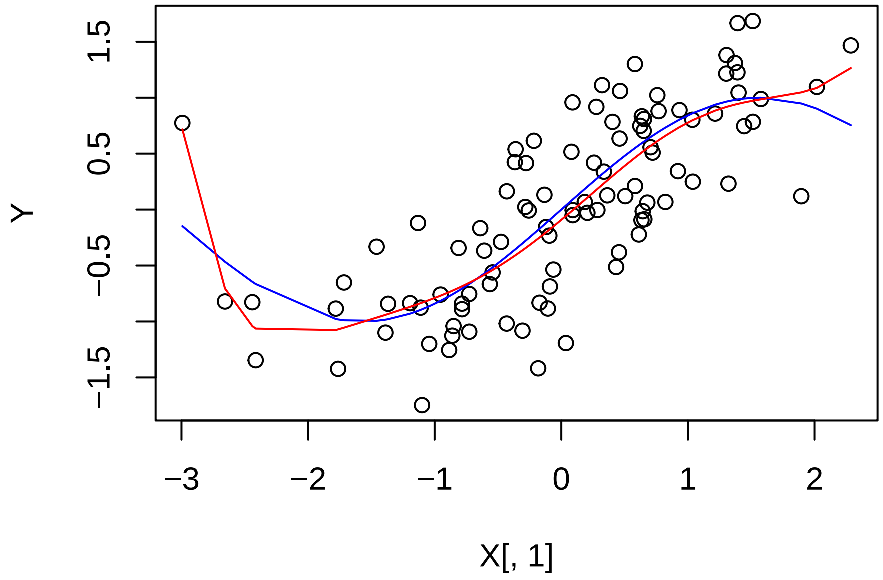
- underfitting:

```r
plot(X[,1], Y)
points(sort(X[,1]), sin(sort(X[,1])), type="l", col="blue")
abline(coef(lm(Y ~ X[,1])), col="red")
```



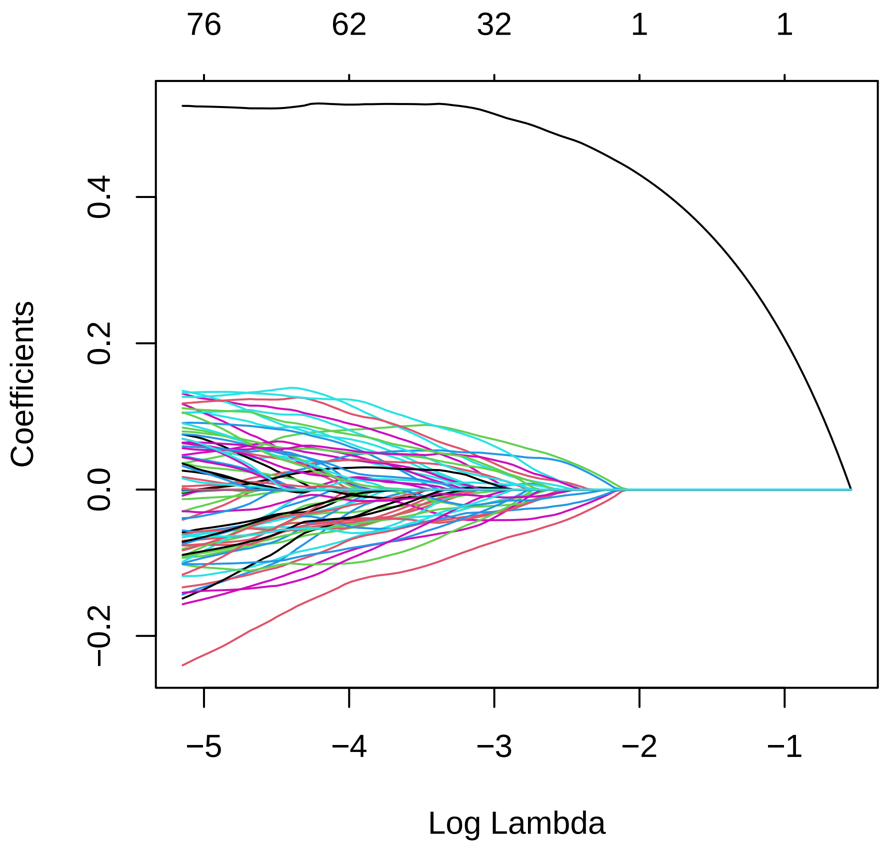
- overfitting:

```r
plot(X[,1], Y)
sX1 <- sort(X[,1])
points(sX1, sin(sX1), type="l", col="blue")
points(sX1, fitted(lm(Y ~ X[,1] + I(X[,1]^2) + I(X[,1]^3) +
                        I(X[,1]^4) + I(X[,1]^5) + I(X[,1]^6) +
                        I(X[,1]^7)))[order(X[,1])],
       type="l", col="red")
```



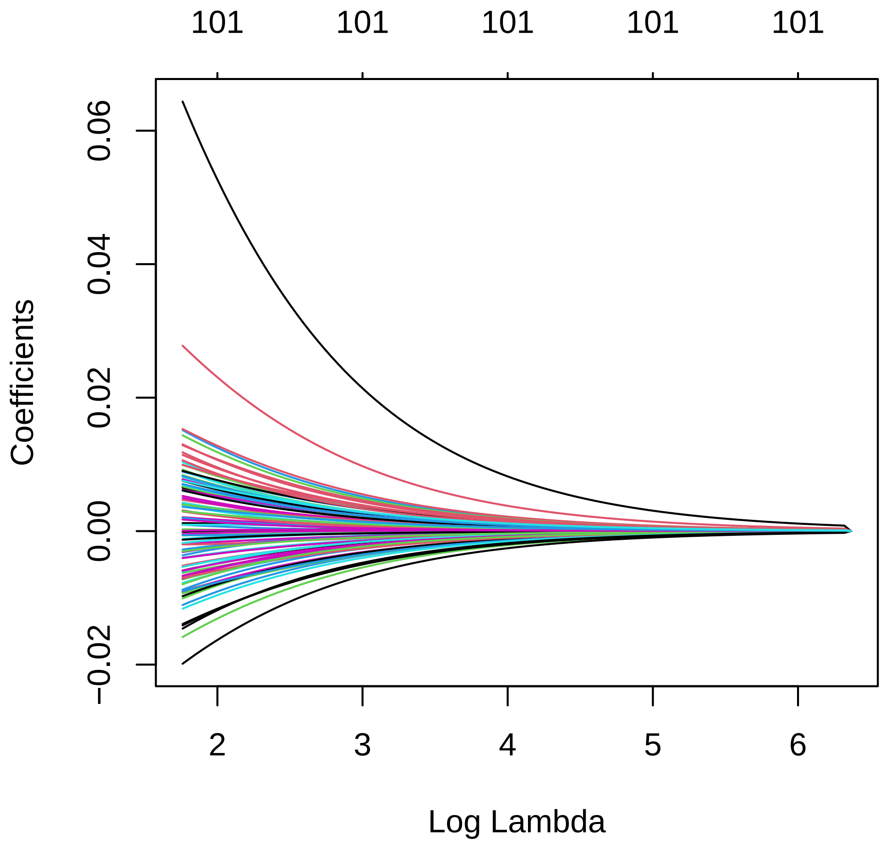
- $L1$ penalty:

```
library(glmnet)
plot(glmnet(X, Y), xvar = "lambda")
```



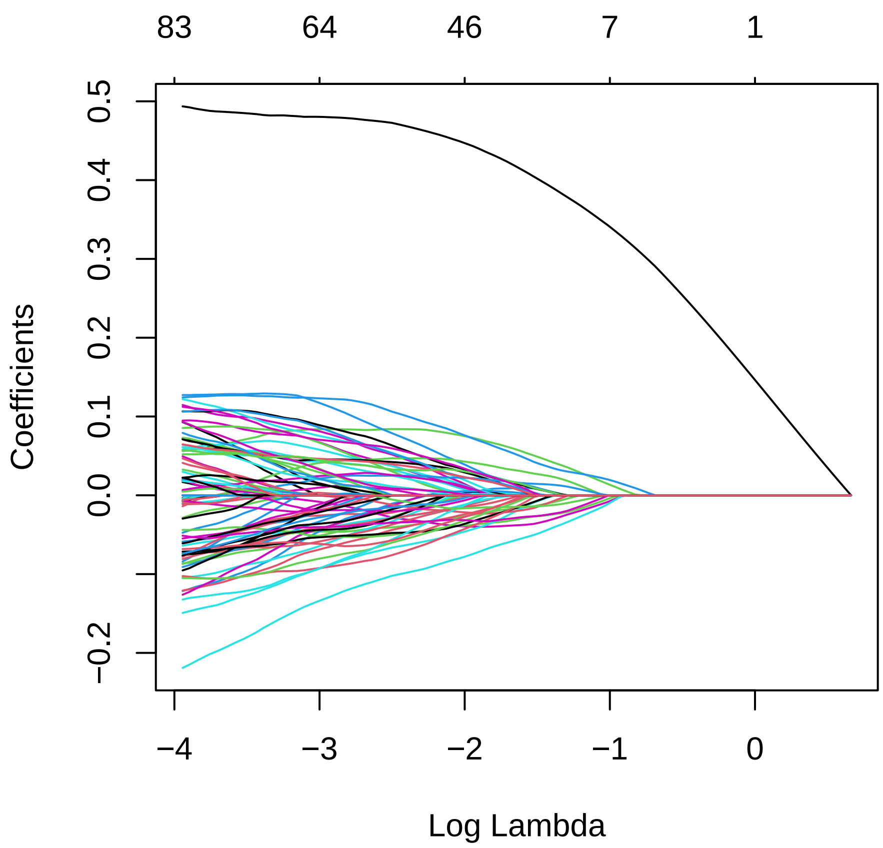
- $L2$ penalty

```
plot(glmnet(X, Y, alpha = 0), xvar = "lambda")
```



- elastic net regularization:

```
plot(glmnet(X, Y, alpha = 0.3), xvar = "lambda")
```



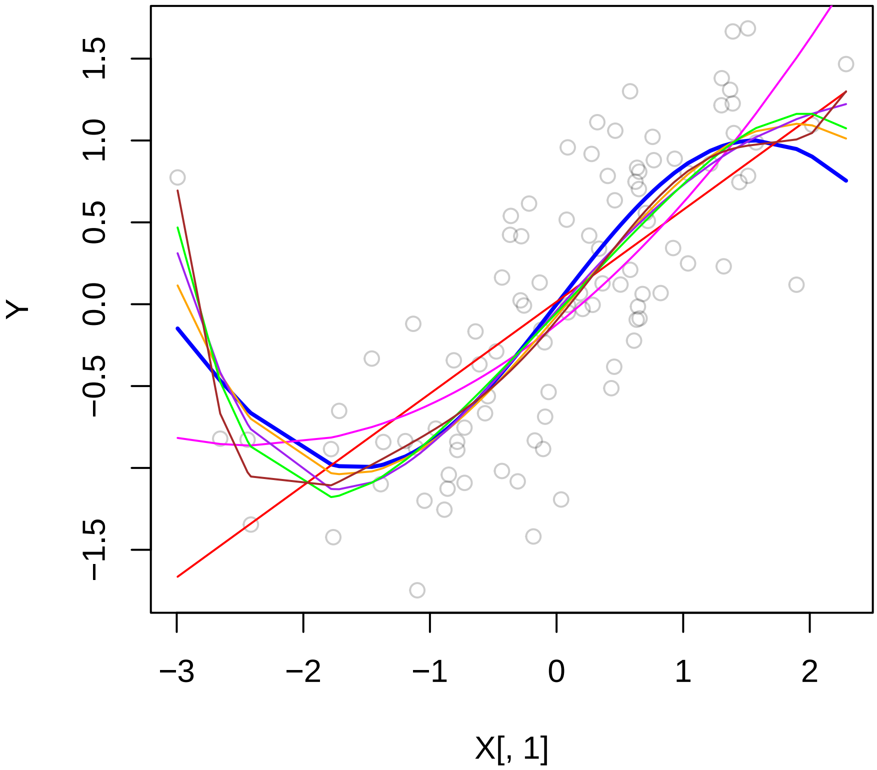
- the underdetermined problem:

```
try(ls_estimator <- solve(crossprod(X), crossprod(X,Y)))

## Error in solve.default(crossprod(X), crossprod(X, Y)) :
##   system is computationally singular: reciprocal condition number = 5.84511e-18
```

- the bias-variance trade-off:

```
plot(X[,1], Y, col=rgb(0,0,0,0.2))
sX1 <- sort(X[,1])
points(sX1, sin(sX1), type="l", col="blue", lwd=2)
points(sX1, fitted(lm(Y ~ X[,1]))[order(X[,1])],
       type="l", col="red")
points(sX1, fitted(lm(Y ~ X[,1] + I(X[,1]^2)))[order(X[,1])],
       type="l", col="magenta")
points(sX1, fitted(lm(Y ~ X[,1] + I(X[,1]^2) + I(X[,1]^3)))[order(X[,1])],
       type="l", col="orange")
points(sX1, fitted(lm(Y ~ X[,1] + I(X[,1]^2) + I(X[,1]^3) +
                        I(X[,1]^4)))[order(X[,1])],
       type="l", col="purple")
points(sX1, fitted(lm(Y ~ X[,1] + I(X[,1]^2) + I(X[,1]^3) +
                        I(X[,1]^4) + I(X[,1]^5)))[order(X[,1])],
       type="l", col="green")
points(sX1, fitted(lm(Y ~ X[,1] + I(X[,1]^2) + I(X[,1]^3) +
                        I(X[,1]^4) + I(X[,1]^5) + I(X[,1]^6)))[order(X[,1])],
       type="l", col="brown")
```



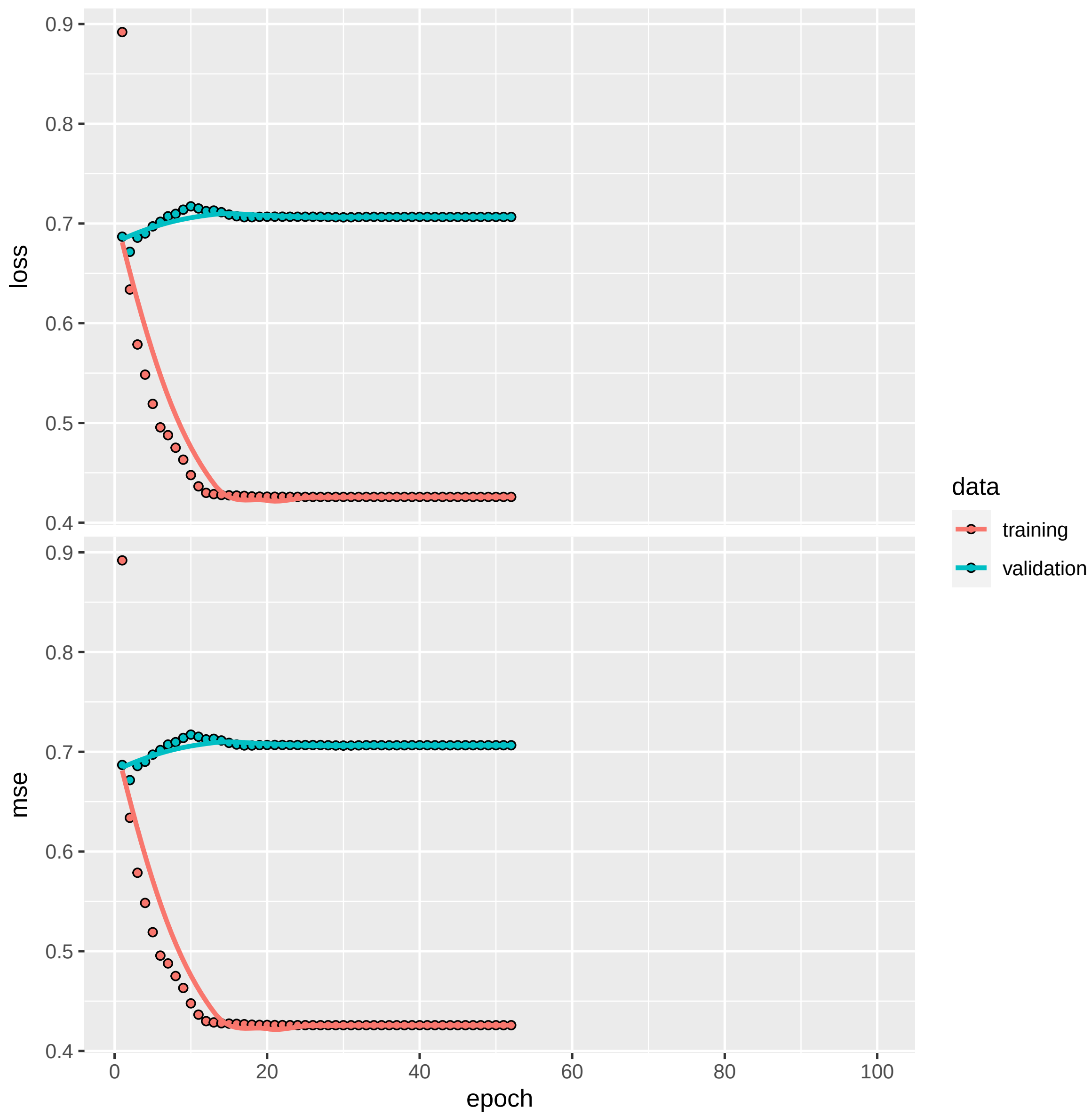
- early stopping (use a simple neural network as in Exercise 2):

```
library(dplyr)
library(keras)

neural_network <- keras_model_sequential()

neural_network %>%
  layer_dense(units = 50, activation = "relu") %>%
  layer_dense(units = 50, activation = "relu") %>%
  layer_dense(units = 1, activation = "relu") %>%
  compile(
    optimizer = "adam",
    loss      = "mse",
    metric = "mse"
  )

history_minibatches <- fit(
  object           = neural_network,
  x                = X,
  y                = Y,
  batch_size       = 24,
  epochs           = 100,
  validation_split = 0.2,
  callbacks = list(callback_early_stopping(patience = 50)),
  verbose = FALSE, # set this to TRUE to get console output
  view_metrics = FALSE # set this to TRUE to get a dynamic graphic output in RStudio
)
plot(history_minibatches)
```



loss
mse
epoch
data
training
validation
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100

Solution 2:

(a) The Taylor approximation of first order of a function $f(x)$ at point x_0 is

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0).$$

On the other hand, a differentiable function f is said to be convex on an interval $\mathcal{I}$ if and only if

$$f(x) \geq f(x_0) + f'(x_0)(x - x_0)$$

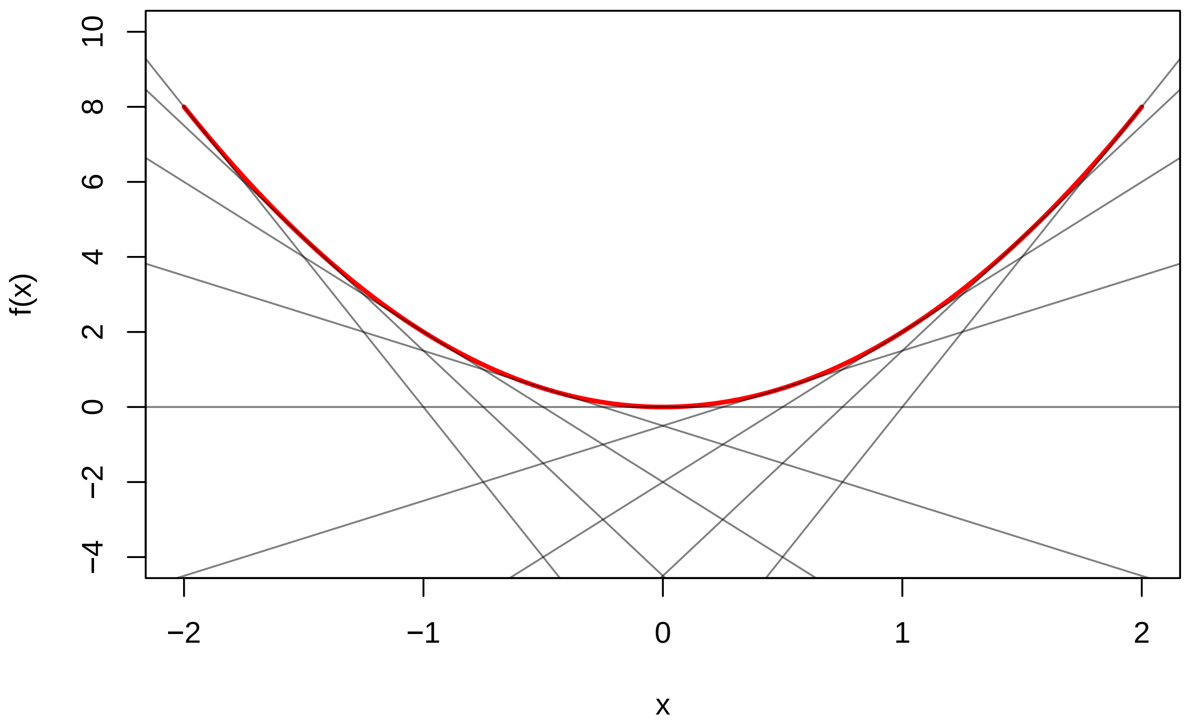
for all points $x, x_0 \in \mathcal{I}$.

(i) If we approximate a convex function with a Taylor approximation of first order, we will always get a lower bound at the given point as the second equation states.

(ii) Visualization of such an approximation for $2x^2$ on $\mathcal{I} = [-2, 2]$ (we will only later see how to calculate a derivative(-like) measure for the non-differentiable functions). The approximation in this case is $f(x) \approx 2x_0^2 + 4x_0(x - x_0) = -2x_0^2 + 4x_0 x$ and for $x = 0$ exactly the function $2x^2$ mirrored at the $f(x) = 0$ axis. We can plot this for several values of x:

```r
xx <- seq(-2, 2, by = 0.01)
yy <- 2*xx^2
# this will give us the approximation function for x=0
# and what happens if we vary x (its slope)
# for given x0
approx_fun <- function(x0) c(-2*x0^2, 4*x0)

plot(xx, yy, type = "l", xlab = "x", ylab = "f(x)", ylim=c(-4,10), col ="red", lwd=2.5)
for(x0 in seq(-2,2,by=0.5))
  abline(approx_fun(x0), col = rgb(0,0,0,0.5))
```



(b) A subdifferential of f is a set of values $\breve{\nabla}_x f$ defined as

$$\breve{\nabla}_x f = \{g : f(x) \geq f(x_0) + g \cdot (x - x_0)\, \forall x \in \mathcal{I}\}.$$

Every scalar value $g \in \breve{\nabla}_x$ is said to be a subgradient of f. A subdifferential thus generalizes the idea of a lower approximation from before by replacing $f'(x_0)$ with any constant g for which the approximation is still strictly below the objective function f.

(c) We can make use of subdifferentials for convex but non-differentiable loss functions like the one induced by the Lasso, because we are now not restricted to cases, where we can compute $f'(x_0)$. It holds that:

A point x_0 is the global minimum of a convex function $f \Leftrightarrow 0$ is contained in the subdifferential $\breve{\nabla}_x f$.

We can define a subdifferential at point x_0 also as a non empty interval $[x_l, x_u]$ where the lower and upper limit is defined by

$$x_l = \lim_{x \to x_0^-} \frac{f(x) - f(x_0)}{x - x_0}, \quad x_u = \lim_{x \to x_0^+} \frac{f(x) - f(x_0)}{x - x_0}.$$

These resemble the limits of the derivative $\partial f / \partial x$ evaluated at a point very close to x_0 when coming from the left or right side, respectively.

(i) In the case for $f(x) = |x|$, $\lim_{x \to 0^\pm} |x|/x = \pm 1$ and thus $\breve{\nabla}_x f = [-1, 1]$ at $x_0 = 0$.

(ii) x_0 is a global minimum as $0 \in \breve{\nabla}_x f$

(iii) The l_1-penalty has no derivative at $\theta_k = 0$ for all θ_k with $k \in \{1, \dots, p\}$. Thus we are particularly interested in the subdifferential at this point, which is

$$\breve{\nabla}_{\theta_k} \lambda \sum_{j=1}^{p} |\theta_j| = \sum_{j=1}^{p} \breve{\nabla}_{\theta_k} \lambda |\theta_j| = \breve{\nabla}_{\theta_k} \lambda |\theta_k| = [-\lambda, \lambda],$$

where in the second equation we use that the subdifferential of a constant function is zero. For a (sub-)gradient at any other differentiable point, we get the conventional gradient using the given hint, which is $-\lambda$ for $\theta_k < 0$ and λ for $\theta_k > 0$.

(d) The subdifferential for the Lasso w.r.t. θ_2 is then simply the combination of the standard gradient for the unregularized risk $\nabla_{emp} := n^{-1} \sum_{i=1}^{n} -2x_2^{(i)}(y^{(i)} - x_1^{(i)}\theta_1 - x_2^{(i)}\theta_2)$ plus the subdifferential for the penalty:

$$\breve{\nabla}_{\theta_2} \mathcal{R}_{reg} = \begin{cases} \nabla_{emp} - \lambda & \text{if } \theta_2 < 0 \\ [\nabla_{emp} - \lambda, \nabla_{emp} + \lambda] & \text{if } \theta_2 = 0 \\ \nabla_{emp} + \lambda & \text{if } \theta_2 > 0. \end{cases}$$