Regression Analysis HW8

Tuorui Peng, tuoruipeng2028@u.northwestern.edu

Question 4.1

(a)

Here note that H and H_k just differ by a rank-1 matrix

$$H_k = H - \frac{1}{n}\ell''(\hat{\varepsilon}_i)x_k x_k'$$

so using SMW formula we have

$$\begin{aligned} H_k^{-1} =& (H - \frac{1}{n}\ell''(\hat{\varepsilon}_k)x_k x_k')^{-1} \\ \overset{\text{SMW}}{=}& H^{-1} + \frac{\frac{1}{n}\ell''(\hat{\varepsilon}_k)H^{-1}x_k x_k' H^{-1}}{1 - \frac{1}{n}\ell''(\hat{\varepsilon}_k)x_k' H^{-1}x_k} \\ =& (I + \frac{\ell''(\hat{\varepsilon}_k)H^{-1}x_k x_k'}{n - \ell''(\hat{\varepsilon}_k)x_k' H^{-1}x_k})H^{-1} \\ \Rightarrow H_k^{-1}g_k =& (I + \frac{\ell''(\hat{\varepsilon}_k)H^{-1}x_k x_k'}{n - \ell''(\hat{\varepsilon}_k)x_k' H^{-1}x_k})H^{-1}g_k \end{aligned}$$

(b)

Using the above expreesion we have

$$\hat{\Delta}_k = -H_k^{-1}g_k = -(I + \frac{\ell''(\hat{\varepsilon}_k)H^{-1}x_k x_k'}{n - \ell''(\hat{\varepsilon}_k)x_k' H^{-1}x_k})H^{-1}g_k$$

then

$$\begin{aligned} \hat{y}_{-k} =& x_k'(\hat{\beta} + \hat{\Delta}_k) = x_k'\hat{\beta} - x_k' H_k^{-1}g_k \\ \hat{\varepsilon}_{-k} =& y_k - \hat{y}_{-k} = y_k - x_k'\hat{\beta} + x_k' H_k^{-1}g_k \\ =& \hat{\varepsilon}_k + x_k' H_k^{-1}g_k \end{aligned}$$

So to compute $\{\hat{y}_{-k}\}_{k=1}^n$, we can, for each k: first compute $x_k' H^{-1}x_k' := \xi_k$, which takes time $O(d^2)$, then compute

$$\begin{aligned} x_k' H_k^{-1}g_k =& x_k'(I + \frac{\ell''(\hat{\varepsilon}_k)H^{-1}x_k x_k'}{n - \ell''(\hat{\varepsilon}_k)x_k' H^{-1}x_k})H^{-1}\frac{1}{n}\ell'(\hat{\varepsilon}_k)x_k \\ =& \frac{1}{n}\ell'(\hat{\varepsilon}_k)[x_k' H^{-1}x_k + \frac{\ell''(\hat{\varepsilon}_k)x_k' H^{-1}x_k x_k' H^{-1}x_k}{n - \ell''(\hat{\varepsilon}_k)x_k' H^{-1}x_k}] \\ =& \frac{1}{n}\ell'(\hat{\varepsilon}_k)[\xi_k + \frac{\ell''(\hat{\varepsilon}_k)\xi_k^2}{n - \ell''(\hat{\varepsilon}_k)\xi_k}] \end{aligned}$$

which takes computation time $O(1)$ (all scalar computation involved). And computation of $\hat{y}_k = x_k'\hat{\beta}$ costs $O(n)$. So the total computation time is $O(nd^2 + n) = O(nd^2)$.

(c)

```
library(CVXR);
```

```
##
## 载入程辑包: 'CVXR'

## The following object is masked from 'package:stats':
##
##     power
```

```
library(tidyverse);
```

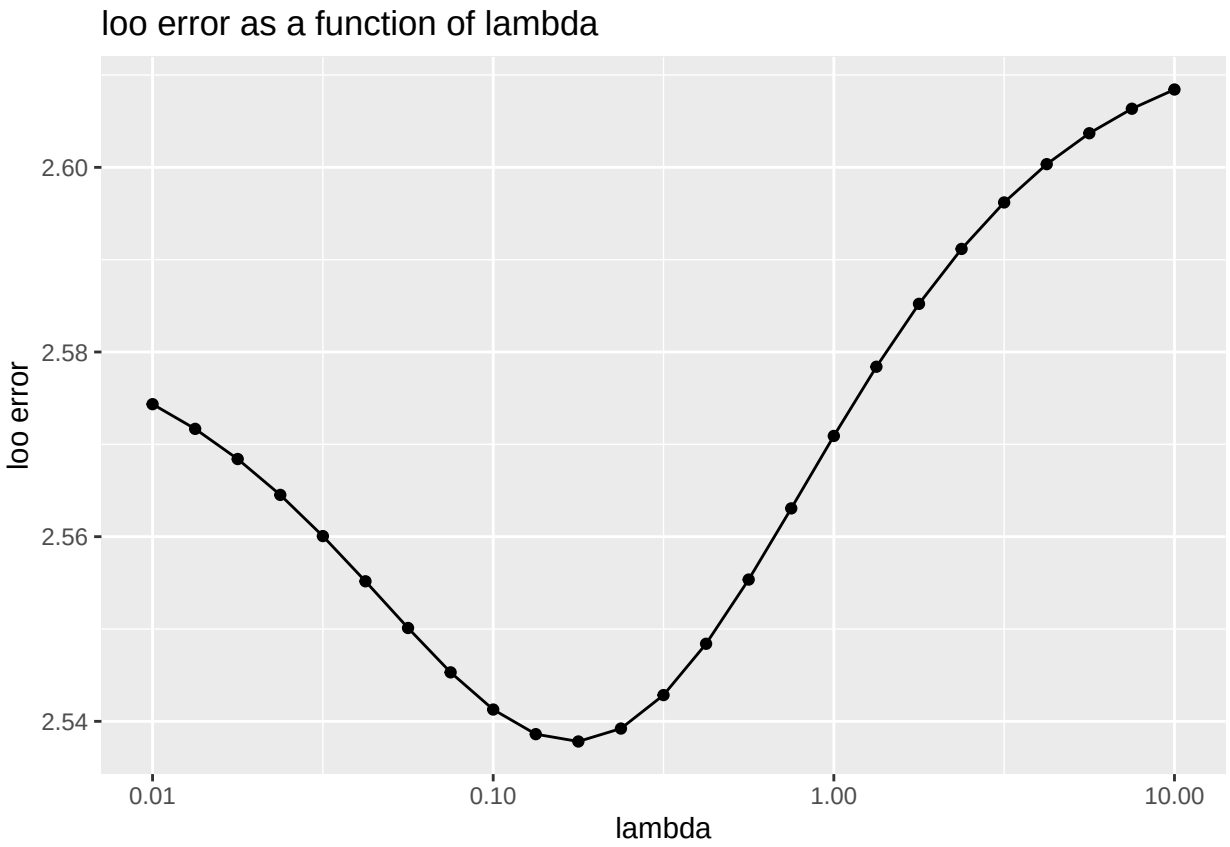
```
## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'
```

```
## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## Found more than one class "atomicVector" in cache; using the first, from namespace 'Matrix'

## Also defined by 'Rmpfr'

## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   1.0.1
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter()    masks stats::filter()
## x dplyr::id()        masks CVXR::id()
## x purrr::is_vector() masks CVXR::is_vector()
## x dplyr::lag()       masks stats::lag()
```

```r
## generate.data(n, d, num.outliers)
##
## Generates training data for the robust regression problem. The
## number of outliers num.outliers defaults to 10.
generate.data <- function(nn = 100, dd = 25, num.outliers = 10) {
    X.train = matrix(rnorm(nn * dd, 0, 1), nn, dd);
    X.test = matrix(rnorm(nn * dd, 0, 1), nn, dd);

    beta.star = rnorm(dd, 0, 1);
    beta.star = beta.star / sqrt(sum(beta.star^2));  # Makes X * beta ~ N(0, 1)

    train.noise = rnorm(nn, 0, 1);
    train.outliers = sample.int(nn, num.outliers);
    test.noise = rnorm(nn, 0, 1);
    test.outliers = sample.int(nn, num.outliers);

    ## Generate outlier measurements

    y.train = X.train %*% beta.star + train.noise;
    signs = sign(rnorm(num.outliers)) # Symmetric random outliers
    y.train[train.outliers] = 5 * signs * rnorm(num.outliers)^4
    y.test = X.test %*% beta.star + test.noise;
```

```
    signs = sign(rnorm(num.outliers)) # Symmetric noise
    y.test[test.outliers] = 5 * signs * rnorm(num.outliers)^4
    return(list("X.train" = X.train, "y.train" = y.train,
                "X.test" = X.test, "y.test" = y.test))
}

## Function to fit the best model to this data using the log(1 + exp)
## loss. To use this function, simply call
##
## minimize.robust.ridge(X.train, y.train, lambda),
##
## which will return a vector minimizing
##
##  (1/n) sum_{i = 1}^n l(y - x_i' * b) + (lambda/2) * ||b||^2
##
## where ||.|| denotes the l2 norm.

minimize.robust.ridge <- function(X, y, lambda) {
    nn = dim(X)[1]
    dd = dim(X)[2]
    beta <- Variable(dd)
    obj <- ((1/nn) * sum(logistic(X %*% beta - y))
        + (1/nn) * sum(logistic(y - X %*% beta))
        + (lambda/2) * sum_squares(beta))
    problem <- Problem(Minimize(obj))
    result <- solve(problem)
    return(result$getValue(beta))
}
# we use loss function ell(x)=log(1+exp(x))+log(1+exp(-x))
# first derivative
ell <- function(x, threshold = 10){ # for numerical stability put some threshold
    return(ifelse(x > threshold, x+2*log(1+exp(-x)), ifelse(x < -threshold, -x+2*log(1+exp(x)), log(1
}
ellprime <- function(x){
    return((exp(x)-1)/(exp(x)+1))
}
# second derivative
ellprimeprime <- function(x){
    return(2*exp(x)/(exp(x)+1)^2)
}
```

```
set.seed(2)
dat <- generate.data(100, 25, 10)
lambdas <- exp(seq(log(0.01), log(10), length.out = 25))
n <- nrow(dat$X.train)
d <- ncol(dat$X.train)
loo.errors <- rep(NA, length(lambdas))
for(lambda in lambdas){
    beta.hat <- minimize.robust.ridge(dat$X.train, dat$y.train, lambda)
    # compute residuals
    eps.hat <- dat$y.train - dat$X.train %*% beta.hat
    # as described, compute $H=1/n * \sum \ell''(\hat{\varepsilon }_i)x_ix_i' + \lambda I$
    H <- 1 / n * t(dat$X.train) %*% diag(as.vector(ellprimeprime(eps.hat))) %*% dat$X.train  + lambda
    Hinverse <- solve(H)
    # compute \xi
    xi <- rep(NA, n)
    for(k in 1:n){
        xi[k] <- t(dat$X.train[k,]) %*% Hinverse %*% dat$X.train[k,]
    }
    xHg <- 1/n*ellprime(eps.hat) * (xi + ellprimeprime(eps.hat) * xi^2 / (n - ellprimeprime(eps.hat)
    # compute \{\hat{\varepsilon}_{-k}\}
    loo.residuals <- eps.hat + xHg
    loo.error <- 1/n * sum(ell(loo.residuals))
    loo.errors[which(lambda == lambdas)] <- loo.error
}

# plotting, lambda on a log scale
plotdf <- data.frame(lambda = lambdas, loo.error = loo.errors)
plotdf %>% ggplot(aes(x = (lambda), y = loo.error)) + geom_point() + geom_line() + scale_x_log10() +
```



(d)

Using the result above, we choose $\lambda \approx 0.178$ for now, to do the following steps.

```
library(glmnet)
```

```
## 载入需要的程辑包: Matrix
```

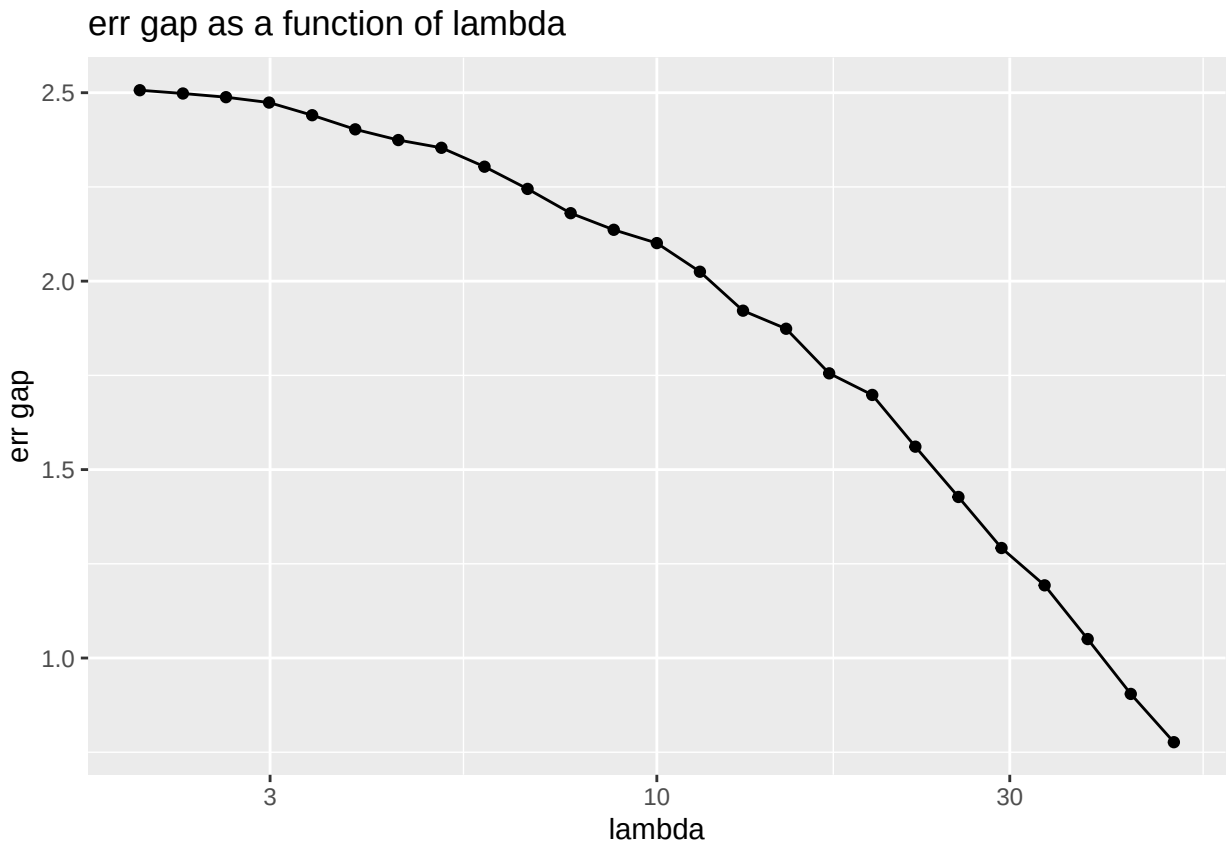
```
##
## 载入程辑包: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

```
# use the best lambda to fir the robust ridge estimator
best_lambda.robust <- lambdas[which.min(loo.errors)]
beta.hat.robust <- minimize.robust.ridge(dat$X.train, dat$y.train, best_lambda.robust)
err_median_robust <- median(abs(dat$X.test %*% beta.hat.robust - dat$y.test))

# get OLS ridge estimator with l2 loss
```

```r
lambdas.ols <- exp(seq(log(2), log(50), length.out = 25))
err_gap <- vector(length = length(lambdas.ols))
for(lambda in lambdas.ols){
    beta.hat.OLS <- solve(t(dat$X.train) %*% dat$X.train + lambda * diag(d)) %*% t(dat$X.train) %*% d
    err_median_OLS <- median(abs(dat$X.test %*% beta.hat.OLS - dat$y.test))
    err_gap[which(lambda == lambdas.ols)] <- err_median_OLS - err_median_robust
}

# plotting, lambda on a log scale
plotdf <- data.frame(lambda = lambdas.ols, err_gap = err_gap)
plotdf %>% ggplot(aes(x = (lambda), y = err_gap)) + geom_point() + geom_line() + scale_x_log10() + la
```



The absolute error gap is monotonely decreasing, which probably means that the influence of outlier is significant to OLS ridge estimator. Larger λ in OLS ridge can somehow fit this problem, but, we notice that the gap always has $\mathrm{ERR}_{\mathrm{test}}^{\mathrm{ls}(\lambda)} - \mathrm{ERR}_{\mathrm{test}} > 0$, which means that the robust ridge estimator always outperforms the OLS ridge estimator (in the sense minimizing expected risk, calculated as the median absolute prediction error on test set).

(e)

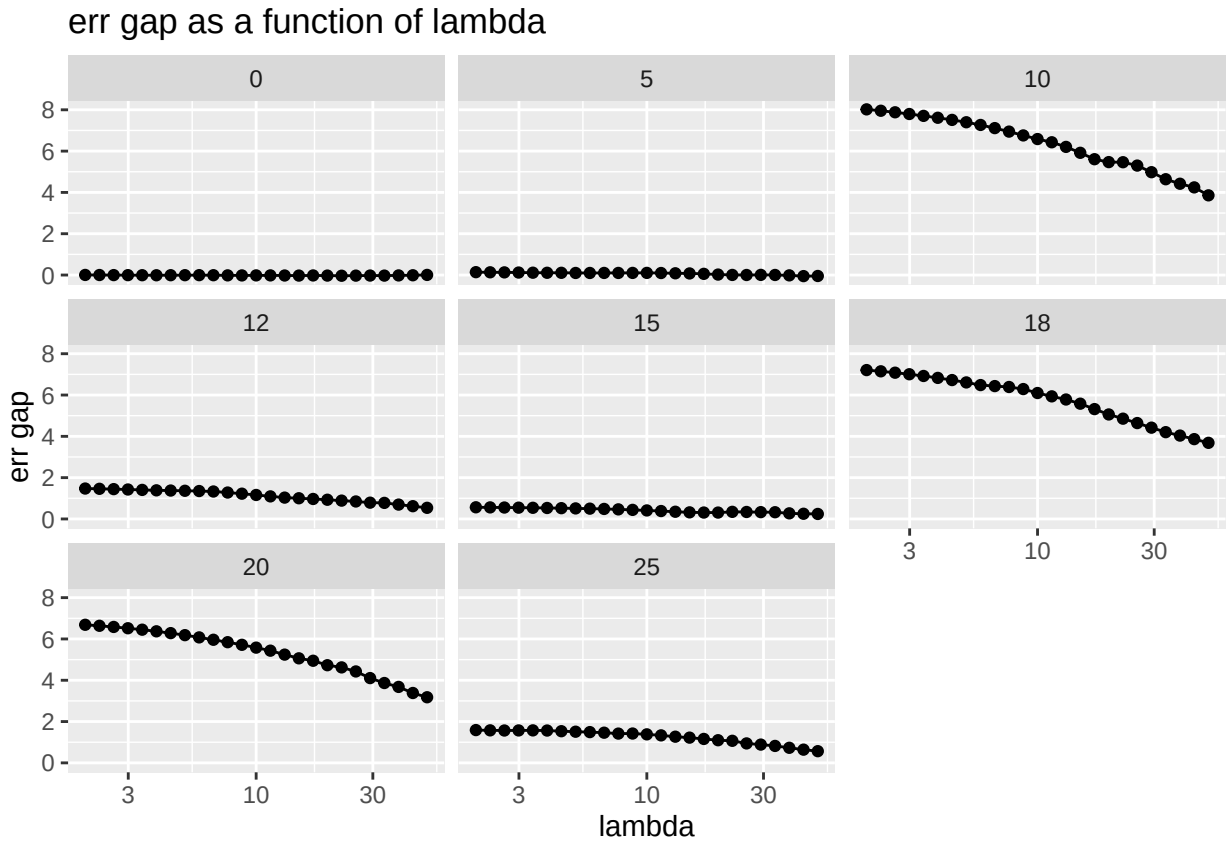
repeat for different number of outliers

```
num_outliers <- c(0,5,10,12,15,18,20,25)
lambdas <- exp(seq(log(0.01), log(10), length.out = 25))
lambdas.ols <- exp(seq(log(2), log(50), length.out = 25))

err_gap_mat <- matrix(NA, nrow = length(lambdas.ols), ncol = length(num_outliers))
for(num_outlier in num_outliers){
    set.seed(num_outlier)
    dat <- generate.data(100, 25, num_outlier)
    n <- nrow(dat$X.train)
    d <- ncol(dat$X.train)
    loo.errors <- rep(NA, length(lambdas))
    for(lambda in lambdas){
        beta.hat <- minimize.robust.ridge(dat$X.train, dat$y.train, lambda)
        # compute residuals
        eps.hat <- dat$y.train - dat$X.train %*% beta.hat
        # as described, compute $H=1/n * \sum \ell''(\hat{\varepsilon }_i)x_ix_i' + \lambda I$
        H <- 1 / n * t(dat$X.train) %*% diag(as.vector(ellprimeprime(eps.hat))) %*% dat$X.train  + la
        Hinverse <- solve(H)
        # compute \xi
        xi <- rep(NA, n)
        for(k in 1:n){
            xi[k] <- t(dat$X.train[k,]) %*% Hinverse %*% dat$X.train[k,]
        }
        xHg <- 1/n*ellprime(eps.hat) * (xi + ellprimeprime(eps.hat) * xi^2 / (n - ellprimeprime(eps.h
        # compute \{\hat{\varepsilon}_{-k}\}
        loo.residuals <- eps.hat + xHg
        loo.error <- 1/n * sum(ell(loo.residuals))
        loo.errors[which(lambda == lambdas)] <- loo.error
    }

    best_lambda.robust <- lambdas[which.min(loo.errors)]
    beta.hat.robust <- minimize.robust.ridge(dat$X.train, dat$y.train, best_lambda.robust)
    err_median_robust <- median(abs(dat$X.test %*% beta.hat.robust - dat$y.test))
    print(cat('err_median_robust for num_outlier = ', num_outlier, ' is ', err_median_robust, sep = '

    # get OLS ridge estimator with l2 loss

    err_gap <- rep(NA, length(lambdas.ols))
    for(lambda in lambdas.ols){
        beta.hat.OLS <- solve(t(dat$X.train) %*% dat$X.train + lambda * diag(d)) %*% t(dat$X.train) %
        err_median_OLS <- median(abs(dat$X.test %*% beta.hat.OLS - dat$y.test))
```

```
        err_gap[which(lambda == lambdas.ols)] <- err_median_OLS - err_median_robust
    }

    err_gap_mat[, which(num_outlier == num_outliers)] <- err_gap
}
```

```
## err_median_robust for num_outlier = 0 is 0.819753NULL
## err_median_robust for num_outlier = 5 is 0.8252419NULL
## err_median_robust for num_outlier = 10 is 0.8777035NULL
## err_median_robust for num_outlier = 12 is 1.045174NULL
## err_median_robust for num_outlier = 15 is 0.8204534NULL
## err_median_robust for num_outlier = 18 is 1.048308NULL
## err_median_robust for num_outlier = 20 is 0.9295824NULL
## err_median_robust for num_outlier = 25 is 1.050889NULL
```

```
## plotting, lambda on a log scale, facet with different number of outliers
plotdf <- data.frame(lambda = lambdas.ols, err_gap = err_gap_mat)
names(plotdf) <- c("lambda", num_outliers)
plotdf %>% gather(key = "num_outliers", value = "err_gap", -lambda) %>% mutate(num_outliers = as.nume
```



We can see similar result for different numbers of outlier $\in \{0, 5, 10, 12, 15, 18, 20, 25\}$. Here I tried many times and observe strange result:

- For small number of outlier, two models (using ℓ_2 loss v.s. robust loss) are almost the same. Because intuitively the effect of outlier is not quite severe, so robust fit and OLS fit are similar.
- For number of outlier ≈ 10 we have a significant outperfom of robust regression over OLS regression. Maybe in this case the robust can really fix the problem of outlier
- For number of outlier $12-15$ we have just slight outperform of robust regression over OLS regression. Maybe in this case the robust can fix the problem of outlier, but not quite well.
- Finally for really large number of outlier, the robust regression is again outperforming OLS regression. Maybe because intuitively the effect of outlier is quite severe, so robust fit and OLS fit are both bad, but just OLS fit is influenced so much.

(Well anyway I can't explain this, perhaps I got something wrong)

Question 4.2

(a)

Using the expression of $L_n(b) \coloneqq \frac{1}{n}\sum_{i=1}^n \ell(y_i - x_i'b)$, we have

$$\nabla^2 L_n(\hat{\beta}+\Delta) - \nabla^2 L_n(\hat{\beta}) = \frac{1}{n}\sum_{i=1}^n x_i x_i' \left(\ell''(\hat{\varepsilon}_i - x_i'\Delta) - \ell''(\hat{\varepsilon})\right)$$

thus $\forall \xi \in \mathbb{R}^d$ we have

$$\begin{aligned}
\xi'\left[\nabla^2 L_n(\hat{\beta}+\Delta) - \nabla^2 L_n(\hat{\beta}) + D_x\|\Delta\|_2 \frac{1}{n}X'X\right]\xi =& \xi'\left[\frac{1}{n}\sum_{i=1}^n x_i x_i' \left(\ell''(\hat{\varepsilon}_i - x_i'\Delta) - \ell''(\hat{\varepsilon}) + D_x\|\Delta\|_2\right)\right]\xi \\
=& \frac{1}{n}\sum_{i=1}^n (\xi' x_i)^2 \left(\ell''(\hat{\varepsilon}_i - x_i'\Delta) - \ell''(\hat{\varepsilon}) + D_x\|\Delta\|_2\right) \\
\geq& \frac{1}{n}\sum_{i=1}^n (\xi' x_i)^2 \left(D_x\|\Delta\|_2 + \min \ell''(\hat{\varepsilon}_i - x_i'\Delta) - \ell''(\hat{\varepsilon})\right) \\
=& 0
\end{aligned}$$

which proves the positive semi-definiteness of $\nabla^2 L_n(\hat{\beta}+\Delta) - \nabla^2 L_n(\hat{\beta}) + D_x\|\Delta\|_2 \frac{1}{n}X'X$, and gives $\nabla^2 L_n(\hat{\beta}+\Delta) \succeq \nabla^2 L_n(\hat{\beta}) - D_x\|\Delta\|_2 \frac{1}{n}X'X$.

Further if $H \coloneqq \nabla^2 L_n(\hat{\beta}) \succeq 2\lambda I$ then we have

$$\begin{aligned}
\nabla^2 L_n(\hat{\beta}+\Delta) \succeq& \nabla^2 L_n(\hat{\beta}) - D_x\|\Delta\|_2 \frac{1}{n}X'X \\
\succeq& 2\lambda I - D_x\|\Delta\|_2 \frac{1}{n}X'X
\end{aligned}$$

then we have, if $\|\Delta\|_2 \leq \lambda/(2D_x \left\|n^{-1}X'X\right\|_{\text{op}})$, then noticing that I has the same eigenvector as $X'X$, we

have

$$\begin{aligned}\lambda_i(2\lambda I - D_x\|\Delta\|_2\frac{1}{n}X'X) =& 2\lambda - \lambda_i(D_x\|\Delta\|_2\frac{1}{n}X'X)\\ \geq& 2\lambda - \lambda_i(D_x\|\Delta\|_2\frac{1}{n}\|X'X\|_{\text{op}})\\ \geq& \frac{3}{2}\lambda, \quad \forall i\end{aligned}$$

which means we have

$$\lambda_{\min}(\nabla^2 L_n(\hat{\beta}+\Delta)) \succeq \frac{3}{2}\lambda$$

(b)

Consider $\nabla^2 L_{-k}(b)$, we have

$$\begin{aligned}\nabla^2 L_{-k}(b) =& \nabla^2\frac{1}{n}\sum_{i\neq k} x_i x_i'\ell''(y_i - x_i'b)\\ =& \nabla^2 L_n(b) - \frac{1}{n}x_k x_k'\ell''(y_k - x_k'b)\\ \Rightarrow \nabla^2 L_{-k}(\hat{\beta}+\Delta) =& \nabla^2 L_n(\hat{\beta}+\Delta) - \frac{1}{n}x_k x_k'\ell''(\hat{\varepsilon}_k - x_k'\Delta)\end{aligned}$$

Then utilizing the result from previous question, when $\|\Delta\|_2 \leq \lambda/(2D_x\|n^{-1}X'X\|_{\text{op}})$, and when $n \geq 2D_x^2/\lambda$, we have

$$\begin{aligned}\lambda_i(\nabla^2 L_{-k}(\hat{\beta}+\Delta)) \geq& \frac{3}{2}\lambda - \frac{1}{n}\lambda_i(x_k x_k'\ell''(\hat{\varepsilon}_k - x_k'\Delta))\\ \overset{(i)}{\geq}& \frac{3}{2}\lambda - \frac{1}{n}D_x^2\\ \geq& \lambda, \quad \forall i\end{aligned}$$

in which (i) uses $\sup|\ell''(t)| \leq 1$ and $\|x_k\|_2 \leq D_x$. Thus we prove

$$\lambda_i(\nabla^2 L_{-k}(\hat{\beta}+\Delta)) \succeq \lambda I$$

(c)

Here definitely our LOO loss function $L_{-k}(b)$ is convex, and we've proven that $\nabla^2 L_{-k}(\hat{\beta}+\Delta) \succeq \lambda I$, thus we can use the lemma introduced to get: $\forall\|\Delta\| \leq c \equiv \lambda/(2D_x\|n^{-1}X'X\|_{\text{op}})$, we have

$$\begin{aligned}L_{-k}(\hat{\beta}+\Delta) \geq& L_{-k}(\hat{\beta}) + \nabla L_{-k}(\hat{\beta})'\Delta + \frac{\lambda}{2}\min\{c, \|\Delta\|_2\}\|\Delta_2\|_2\\ =& L_{-k}(\hat{\beta}) + \frac{1}{n}\ell'(\hat{\varepsilon}_k)x_k'\Delta + \frac{\lambda}{2}\min\{c, \|\Delta\|_2\}\|\Delta\|_2\\ \geq& L_{-k}(\hat{\beta}) - \frac{1}{n}D_x\|\Delta\|_2 + \frac{\lambda}{2}\|\Delta\|_2^2\end{aligned}$$

which is a quadratic function of $\|\Delta\|_2$, we can see that the minimum is achieved at $\|\Delta\|_2 = \dfrac{D_x}{n\lambda}$. Consider the minimizer of $L_{-k}(\hat{\beta}+\Delta)$, we can see that the minimizer Δ_k must have $|\|\Delta_k\|_2 - \dfrac{D_x}{n\lambda}| \leq \dfrac{D_x}{\lambda n}$, otherwise

we have $L_{-k}(\hat{\beta}+\Delta_k) \geq L_{-k}(\hat{\beta}) - \frac{1}{n}D_x\|\Delta_k\|_2 + \frac{\lambda}{2}\|\Delta_k\|_2^2 \geq L_{-k}(\hat{\beta})$, which contradicts the fact that Δ_k is the minimizer. Thus we have

$$|\|\Delta_k\|_2 - \frac{D_x}{n\lambda}| \leq \frac{D_x}{\lambda n} \Rightarrow \|\Delta_k\|_2 \leq \frac{2D_x}{\lambda n}$$

in which we notice that $n \geq \frac{4D_x^2 |\|n^{-1}X'X\|\|_{\text{op}}}{\lambda^2}$ automatically guarentees that

$$\|\Delta_k\|_2 \leq \frac{2D_x}{\lambda n} \leq \frac{\lambda}{2D_x \left|\|n^{-1}X'X\|\right|_{\text{op}}}$$

which means our $\hat{\beta}+\Delta_k$ is still in the neighborhood of $\hat{\beta}$, i.e. $\|\Delta\|_2 \leq c$ such that the quadric bound lemma still holds.

(d)

(I did't find a complete proof of the problem, but I think the following is a good try)

If we express $\Delta_k := (H_k+E)^{-1}(-g_k)$ where E is some 'noise matrix', then we have

$$\begin{aligned}
\|\Delta_k - \hat{\Delta}_k\|_2 =& \|((H_k+E)^{-1} - H_k^{-1})(-g_k)\| \\
\leq& \left|\|(H_k+E)^{-1} - H_k^{-1}\|\right|_{\text{op}} \|g_k\|_2 \\
\leq& \frac{|\|E\||_{\text{op}}}{\lambda_{\min}(H_k) - |\|E\||_{\text{op}}} \frac{D_x}{n} \\
\leq& \frac{|\|E\||_{\text{op}}}{\lambda - |\|E\||_{\text{op}}} \frac{D_x}{\lambda}
\end{aligned}$$

In this way, **IF** we can prove that $|\|E\||_{\text{op}} \leq \frac{2D_x^4}{\lambda n}$, then we can prove

$$\begin{aligned}
\|\Delta_k - \hat{\Delta}_k\|_2 \leq& \frac{|\|E\||_{\text{op}}}{\lambda - |\|E\||_{\text{op}}} \frac{D_x}{\lambda} \\
\leq& \frac{2D_x^4/\lambda n}{\lambda - 2D_x^4/\lambda n} \frac{D_x}{\lambda} \\
=& \frac{2D_x^5}{\lambda^2 - 2D_x^4} \frac{1}{n^2} = O(\frac{1}{n^2})
\end{aligned}$$

Question 4.3

(a)

- **"⇒"**: if $Z_{n+1} \leq Z_{(k,n)}$, then we have

  $$\{Z_{n+1}, \{Z_{(\kappa,n)}\}_{\kappa=1}^{k-1}\} = \{Z_{(\kappa,n+1)}\}_{\kappa=1}^{k}$$

  which gives

  $$Z_{n+1} \leq \max\{Z_{(\kappa,n+1)}\}_{\kappa=1}^{k} = Z_{(k,n+1)}$$

- **"⇐"**: if $Z_{n+1} \leq Z_{(k,n+1)}$, note that in this case $Z_{n+1} \in \{Z_{(\kappa,n+1)}\}_{\kappa=1}^{n+1}$, we have

$$\{Z_{(\kappa,n+1)}\}_{\kappa=1}^{k} = \{Z_{(\kappa,n)}\}_{\kappa=1}^{k-1} \uplus \{Z_{n+1}\}$$

thus get

$$Z_{n+1} \leq Z_{(k,n+1)} \leq Z_{(k,n)}$$

(b)

Note that for empirical quantile we have

$$\mathbb{P}\left(Z_n \leq \hat{Q}_n(\alpha)\right) \geq \alpha$$

in which $\hat{Q}_n(\alpha) = Z_{(\lceil n\alpha \rceil, n)}$. So now it suffices to show that

$$\begin{aligned}
&\mathbb{P}\left(Z_{n+1} \leq \hat{Q}_n((1+\frac{1}{n})\alpha)\right) \geq \alpha \\
\Leftrightarrow &\mathbb{P}\left(Z_{n+1} \leq Z_{(\lceil n(1+\frac{1}{n})\alpha \rceil, n)}\right) \geq \alpha \\
\Leftrightarrow &\mathbb{P}\left(Z_{n+1} \leq Z_{(\lceil (n+1)\alpha \rceil, n)}\right) \geq \alpha \\
\overset{(i)}{\Leftarrow} &\mathbb{P}\left(Z_{n+1} \leq Z_{(\lceil (n+1)\alpha \rceil, n+1)}\right) \geq \alpha \\
\Leftrightarrow &\mathbb{P}\left(Z_{n+1} \leq \hat{Q}_{n+1}(\alpha)\right) \geq \alpha
\end{aligned}$$

in which (i) uses the fact that $Z_{(k,n)} \geq Z_{(k,n+1)}$, $\forall k$, so

$$\mathbb{P}\left(Z_{n+1} \leq Z_{(\lceil (n+1)\alpha \rceil, n)}\right) \geq \mathbb{P}\left(Z_{n+1} \leq Z_{(\lceil (n+1)\alpha \rceil, n+1)}\right) \geq \alpha$$

thus proves the result.

(c)

Replace $\alpha \mapsto (1-\alpha)$ in result off previous question, we have

$$\mathbb{P}\left(Z_{n+1} > \hat{\tau}_n\right) = \mathbb{P}\left(Z_{n+1} > \hat{Q}_n((1+\frac{1}{n})(1-\alpha))\right) < \alpha$$

(d)

Using definition $\hat{C}_{\hat{\tau}_n}(x) = [\hat{f}(x) - \hat{\tau}_n, \hat{f}(x)\hat{\tau}_n]$, we have

$$\begin{aligned}
\mathbb{P}\left(Y_{n+1} \in \hat{C}_{\hat{\tau}_n}(X_{n+1})\right) &= \mathbb{P}\left(\left|Y_{n+1} - \hat{f}(X_{n+1})\right| \leq \hat{\tau}_n\right) \\
&= \mathbb{P}\left(Z_{n+1} \leq \hat{Q}_n((1+\frac{1}{n})(1-\alpha))\right) \geq 1-\alpha
\end{aligned}$$

(e)

With the definition mentioned, we have

$$\begin{aligned}\mathbb{P}\left(Y_{n+1} \in \hat{C}_\delta(X_{n+1})\right) =& \mathbb{P}\left(\max\left\{\hat{q}_{\delta_{\text{low}}} - Y_{n+1}, Y_{n+1} - \hat{q}_{\delta_{\text{high}}}\right\} \leq \hat{\tau}_n\right) \\ =& \mathbb{P}\left(Z_{n+1} \leq \hat{Q}_n(\frac{1+n}{n}(1-\alpha))\right) \geq 1-\alpha\end{aligned}$$

(f)

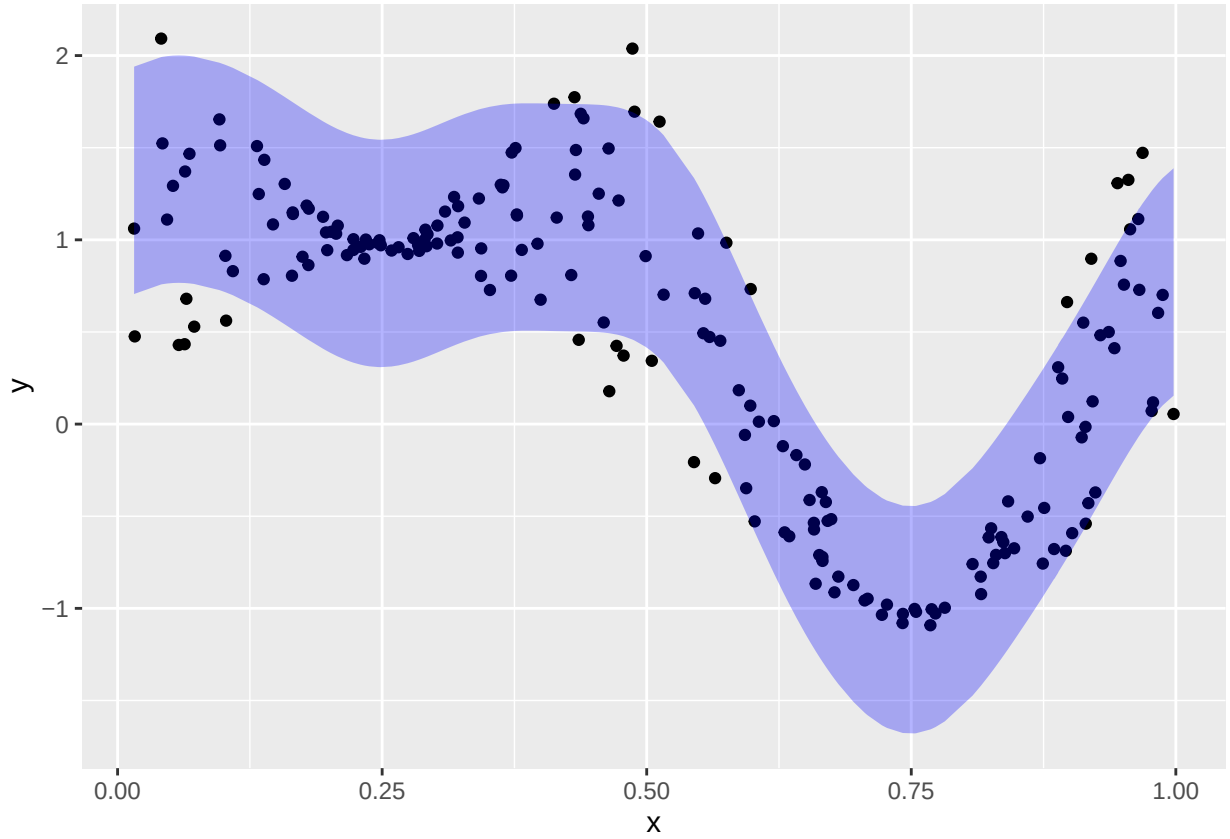
```r
## function provided in lecture file
n.sample = 200

X.train = sort(runif(n.sample, min=0, max = 1))
X.valid = sort(runif(n.sample, min=0, max = 1))
X.test = sort(runif(n.sample, min=0, max = 1))

y.train = sin(2 * pi * X.train) - .1 +
    (1.1 + cos(4 * pi * X.train)) * runif(n.sample, min = 0, max = 1)

y.valid = sin(2 * pi * X.valid) - .1 +
    (1.1 + cos(4 * pi * X.valid)) * runif(n.sample, min = 0, max = 1)

y.test = sin(2 * pi * X.test) - .1 +
    (1.1 + cos(4 * pi * X.test)) * runif(n.sample, min = 0, max = 1)

library(CVXR)
library(tidyverse)

predictKRR <- function(X, Z, beta, tau = .1, offset = 0) {
    if ((is.matrix(X) && ncol(X) != 1) ||
        (is.matrix(Z) && ncol(Z) != 1)) {
        stop("Data should be 1-dimensional")
    }
    nn = length(X)
    mm = length(Z)
    ip.matrix = X %*% t(Z)  # n-by-m matrix with entries x_i' * z_j
    squared.Xs = replicate(mm, X * X)  # n-by-m matrix whose ith
                                       # row is all x_i' * x_i
    squared.Zs = replicate(nn, Z * Z)  # m-by-n matrix whose ith row
                                       # is all z_i' * z_i
    dist.squared.matrix = squared.Xs - 2 * ip.matrix + t(squared.Zs)
```

```
    kernel.matrix = t(exp(-dist.squared.matrix / (2 * tau^2)))
    predictions = offset + kernel.matrix %*% beta;
    return(predictions)
}
fitQuantileKernel <- function(X, y, quantile.level = .5,
                              lambda = .01, tau = .1) {
    nn = length(X)
    if (is.matrix(X) && ncol(X) != 1) {
        stop("Data X should be 1-dimensional")
    }
    ## Construct the Gram matrix
    ip.matrix = X %*% t(X)  # n-by-n matrix with entries x_i' * x_j
    squared.Xs = (X * X) %*% t(rep(1, nn))  # turn it into an n-by-n matrix
    dist.squared = squared.Xs + t(squared.Xs) - 2 * ip.matrix
    G = exp(-dist.squared / (2 * tau^2))
    ## We formulate the problem as solving
    ##
    ##  minimize    sum_i l_alpha(z_i) + (lambda/2) * beta' * G * beta
    ##  subject to  z = y - G * beta
    ##
    ## but we don't actually introduce the new variable z...
    beta = Variable(nn)
    obj = ((1/nn) * sum(quantile.level * pos(y - G %*% beta) +
                        (1 - quantile.level) * neg(y - G %*% beta)) +
           (lambda/2) * quad_form(beta, G))
    problem = Problem(Minimize(obj))
    result = solve(problem)
    return(result$getValue(beta))
}
fitKernelRidge <- function(X, y, lambda = .01, tau = .1) {
    nn = length(X)
    if (is.matrix(X) && ncol(X) != 1) {
        stop("Data X should be 1-dimensional")
    }
    ## Construct the Gram matrix
    ip.matrix = X %*% t(X)  # n-by-n matrix with entries x_i' * x_j
    squared.Xs = (X * X) %*% t(rep(1, nn))  # turn it into an n-by-n matrix
    dist.squared = squared.Xs + t(squared.Xs) - 2 * ip.matrix
    G = exp(-dist.squared / (2 * tau^2))
    diag(G) = 1 + lambda;
    beta = solve(G, y)
```

```
    return(beta)
}
plotUpperAndLower <- function(X, y, y.low, y.high, filename) {
    dat = tibble(x = X, y = y,
                 low = y.low, high = y.high)
    p = ggplot(dat, aes(x = x)) +
        geom_point(aes(y = y)) +
        geom_ribbon(aes(ymin = low, ymax = high),
                    fill = "blue", alpha = .3)
    ggsave(filename, p, device="pdf")
    return(p)
}

delta.low <- 0.1
delta.high <- 0.9
tau <- 0.1
lambda <- 0.01
alpha <- 0.1
n <- length(X.valid)

## Fit squared loss KRR
beta.krr <- fitKernelRidge(X.train, y.train, lambda = lambda, tau = tau)
## Fit quantile KRR
beta.qkrr.low <- fitQuantileKernel(X.train, y.train, quantile.level = delta.low, lambda = lambda, tau
beta.qkrr.high <- fitQuantileKernel(X.train, y.train, quantile.level = delta.high, lambda = lambda, t

# For ridge regression estimate, find its conformal tau
yhat.krr.val <- predictKRR(X.train, X.valid, beta.krr, tau = tau)
tau.hat.krr <- quantile(abs(yhat.krr.val - y.valid), (1+1/n)*(1 - alpha))
# For quantile kernel ridge regression estimate, find its conformal tau
yhat.qkrr.low.val <- predictKRR(X.train, X.valid, beta.qkrr.low, tau = tau)
yhat.qkrr.high.val <- predictKRR(X.train, X.valid, beta.qkrr.high, tau = tau)
Z.qkrr.val <- pmax(yhat.qkrr.low.val - y.valid, y.valid - yhat.qkrr.high.val)
tau.hat.qkrr <- quantile(Z.qkrr.val, (1+1/n)*(1 - alpha))

# Plot conformal band on test set
yhat.krr.test <- predictKRR(X.train, X.test, beta.krr, tau = tau)
yhat.qkrr.low.test <- predictKRR(X.train, X.test, beta.qkrr.low, tau = tau)
yhat.qkrr.high.test <- predictKRR(X.train, X.test, beta.qkrr.high, tau = tau)

plot.krr <- plotUpperAndLower(X.test, y.test, yhat.krr.test - tau.hat.krr, yhat.krr.test + tau.hat.kr
```

```
## Saving 6.5 x 4.5 in image
```

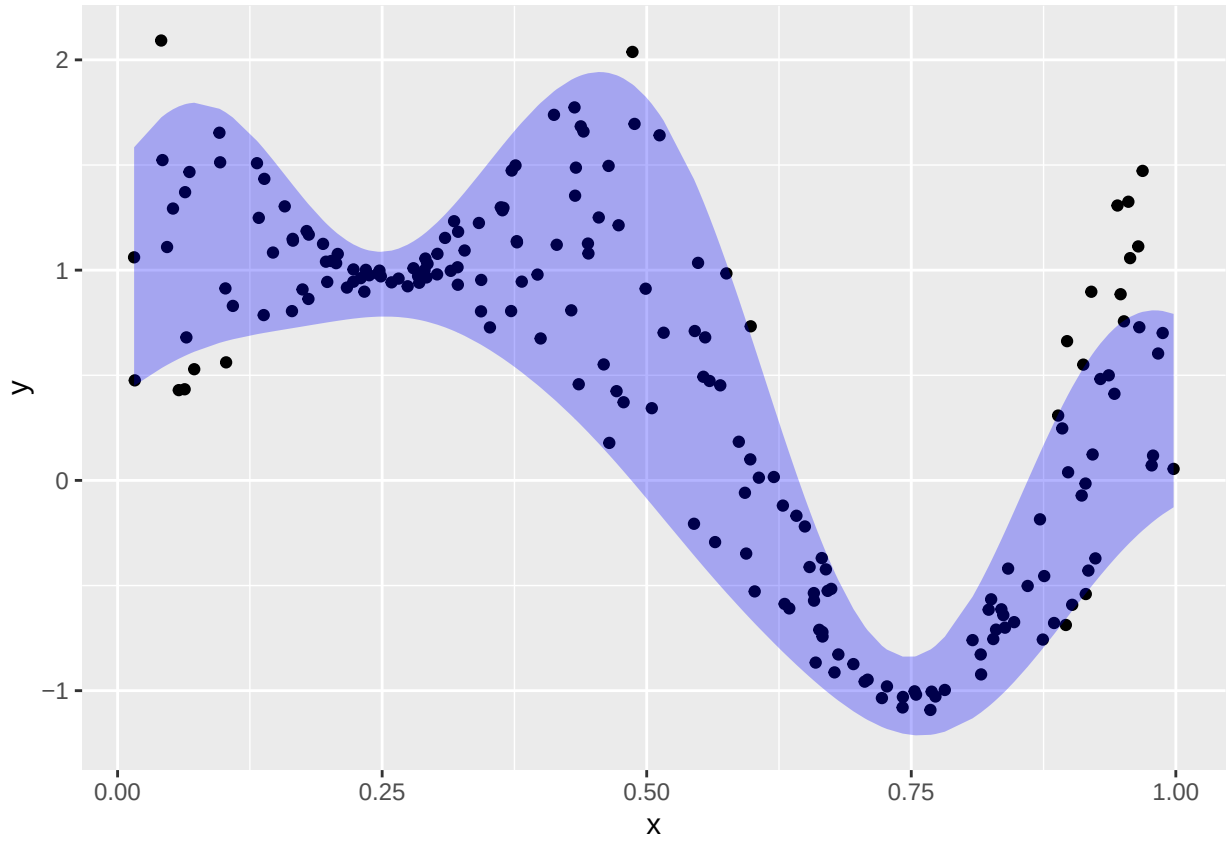
```
plot.krr
```




```
plot.qkrr <- plotUpperAndLower(X.test, y.test, yhat.qkrr.low.test - tau.hat.qkrr, yhat.qkrr.high.test
```

```
## Saving 6.5 x 4.5 in image
```

```
plot.qkrr
```




```
## Compute covering rate
covering.rate.krr <- mean(abs(y.test - yhat.krr.test) <= tau.hat.krr)
covering.rate.qkrr <- mean( pmax(yhat.qkrr.low.test - y.test, y.test - yhat.qkrr.high.test) <= tau.ha

cat("Covering rate for KRR is ", covering.rate.krr, "\n", sep = '', labels = '.')
```

```
## Covering rate for KRR is 0.875
```

```
cat("Covering rate for quantile KRR is ", covering.rate.qkrr, "\n", sep = '', labels = '.')
```

```
## Covering rate for quantile KRR is 0.91
```