T04: Outliers and l_1 loss

MATH 4432 Statistical Machine Learning

WANG Zhiwei

MATH, HKUST

2022-09-27

Let's start by recalling linear regression!

Least squares

Recall the least squares (LS) problem for linear regression

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N}(y_i - x_i\beta)^2 = \underset{\beta}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i\beta)^2.$$

A toy example   Code   Minimize the squared loss

- Suppose we know the ground truth of $f(\cdot) : f(x) = x$

- Now given $\{x_i\}_{i=1}^{N}$, we have a set of observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ with

$$y_i = f(x_i) + \epsilon_i,$$

where $\epsilon_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 0.1^2)$ is random noise.

Least squares

Recall the least squares (LS) problem for linear regression

$$\hat{\beta} = \underset{\beta}{\text{argmin}} \sum_{i=1}^{N} (y_i - x_i\beta)^2 = \underset{\beta}{\text{argmin}} \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i\beta)^2.$$

A toy example   **Code**   Minimize the squared loss

```r
set.seed(123)
N <- 10 # Sample size
x <- runif(N, 0, 1)
y0 <- x # Ground truth
y_obs <- y0 + rnorm(N, mean = 0, sd = 0.1) # Add noise, observed data
ggplot(data = NULL, aes(x = x, y = y_obs)) +
  geom_point(aes(x = x, y = y_obs), size = 5) +
  geom_smooth(method = "lm", size = 1.5) +
  theme(
    text = element_text(size = 18),
    axis.text.y = element_text(size = 18),
    axis.text.x = element_text(size = 18)
    )
```
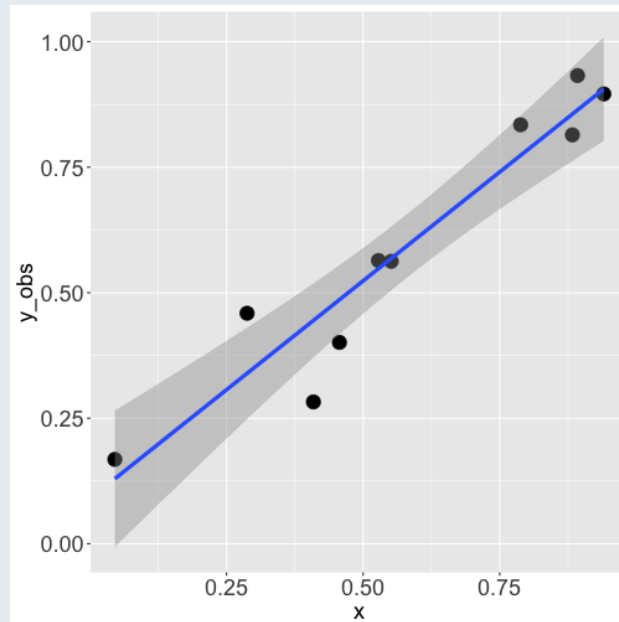
Least squares

Recall the least squares (LS) problem for linear regression

$$\hat{\beta} = \underset{\beta}{\mathrm{argmin}} \sum_{i=1}^{N}(y_i - x_i\beta)^2 = \underset{\beta}{\mathrm{argmin}} \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i\beta)^2.$$

A toy example   Code   **Minimize the squared loss**

What if there exists an outlier?

Add an outlier

- We follow the above problem setting.

- But add an outlier

$$(x, y, y_{\mathrm{obs}}) = (0.9, 0.9, -2),$$

of which the noise is extremely large. This situation is rare, but the probability is not zero!

Add an outlier

```r
set.seed(123)

x_ol <- c(x, 0.9)
y0_ol <- c(y0, 0.9)
y_obs_ol <- c(y_obs, -2) # Add noise, observed data

ggplot(data = NULL) +
  geom_point(aes(x = x, y = y_obs), size = 5) +
  geom_point(aes(x = 0.9, y = -2), color = "red", size = 5) +
  geom_smooth(aes(x = x, y = y_obs), method = "lm", color = "blue", size
  geom_smooth(aes(x = x_ol, y = y_obs_ol), method = "lm", color = "red",
  theme(
    text = element_text(size = 18),
    axis.text.y = element_text(size = 18),
    axis.text.x = element_text(size = 18)
    )
```
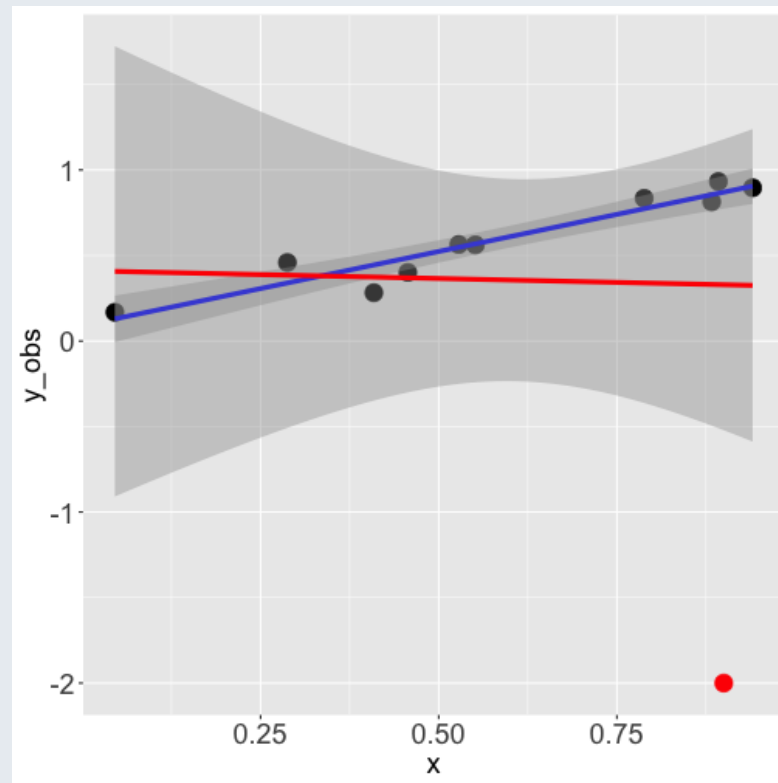
Add an outlier

Brief summary of squared loss

Pros

- Natural, intuitive (Euclidean distance)

- Closed form solution

Con

- Not robust to outliers, equal weights to all data (assumes Gaussian distributed residual)

Let's consider a more robust loss function!
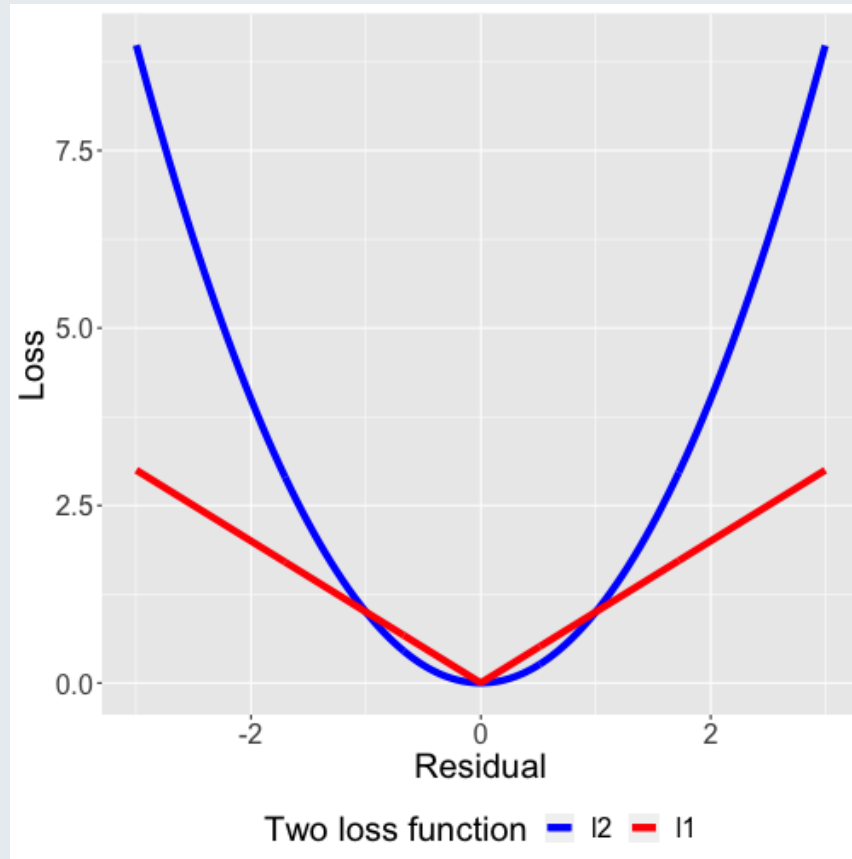
l_1 loss

- l_2 norm and l_1 norm

$$||x||_2 = \left(\sum_{j=1}^{p} x_j^2 \right)^{\frac{1}{2}}, \quad ||x||_1 = \sum_{j=1}^{p} |x_j|.$$

- l_2 loss and l_1 loss

$$\mathcal{L}_2 = \sum_{i=1}^{N} (y_i - x_i \beta)^2 = ||y - X\beta||_2^2,$$

$$\mathcal{L}_1 = \sum_{i=1}^{N} |y_i - x_i \beta| = ||y - X\beta||_1.$$

Why is l_1 more robust?

However, the bad news is that l_1 loss function is not differentiable :(

Let's relax it!

MM algorithm *

The "MM" stands for "Majorization-Minimization" or "Minorization-Maximization". In the following "MM" refers to "Majorization-Minimization".

- Consider the following presumably difficult optimization problem

$$\operatorname*{minimize}_{\mathbf{x}} f(\mathbf{x})$$

$$\text{subject to } \mathbf{x} \in \mathcal{X},$$

  with $\mathcal{X}$ being the feasible set and $f(\mathbf{x})$ being continuous.

- Idea: successively minimize a more managable surrogate function $u\left(\mathbf{x}, \mathbf{x}^k\right)$

$$\mathbf{x}^{k+1} = \arg \min_{\mathbf{x} \in \mathcal{X}} u\left(\mathbf{x}, \mathbf{x}^k\right),$$

  hoping the sequence of minimizers $\left\{\mathbf{x}^k\right\}$ will converge to optimal $\mathbf{x}^\star$.

[*] Not required in this course. Materials are form ELEC 5470 / IEDA 6100A Convex Optimization, Prof. Daniel P. Palomar, ECE, HKUST.

Iterative algorithm

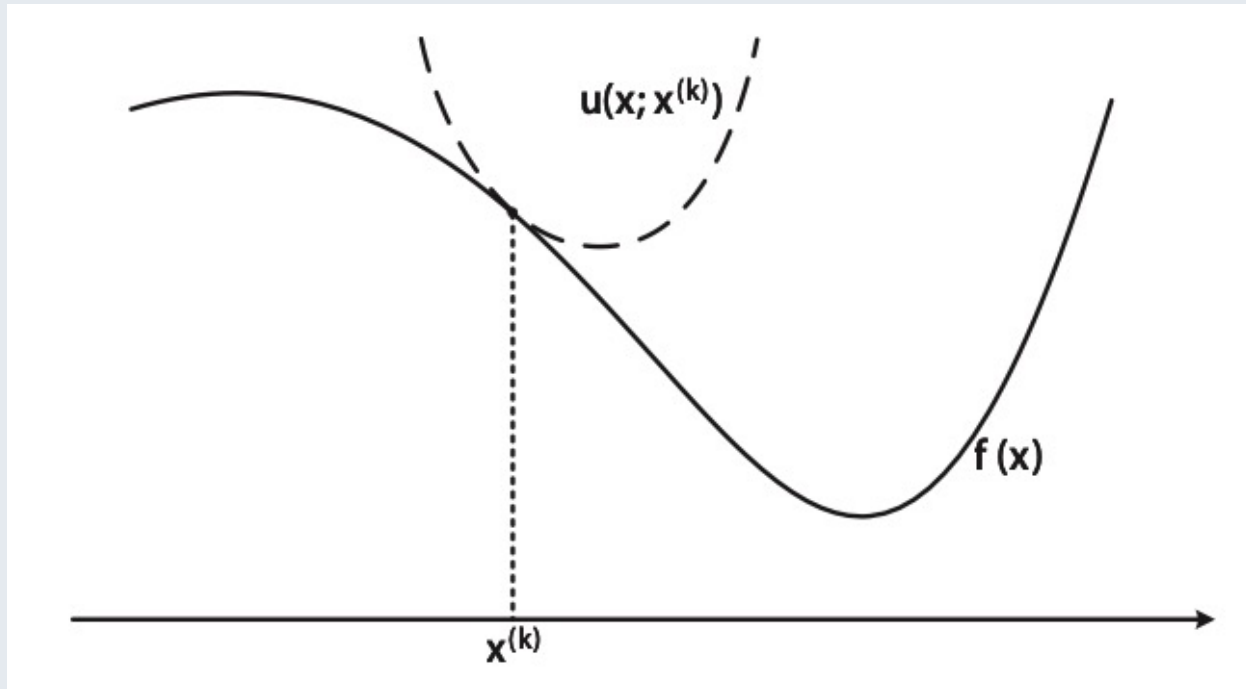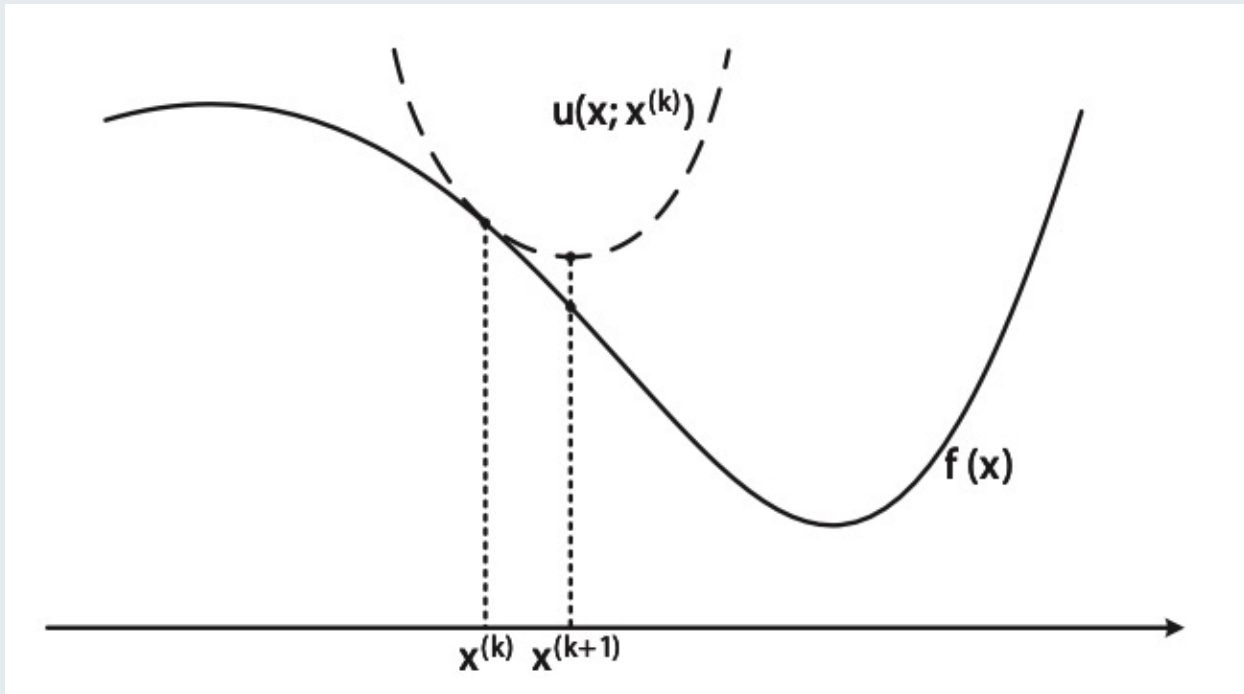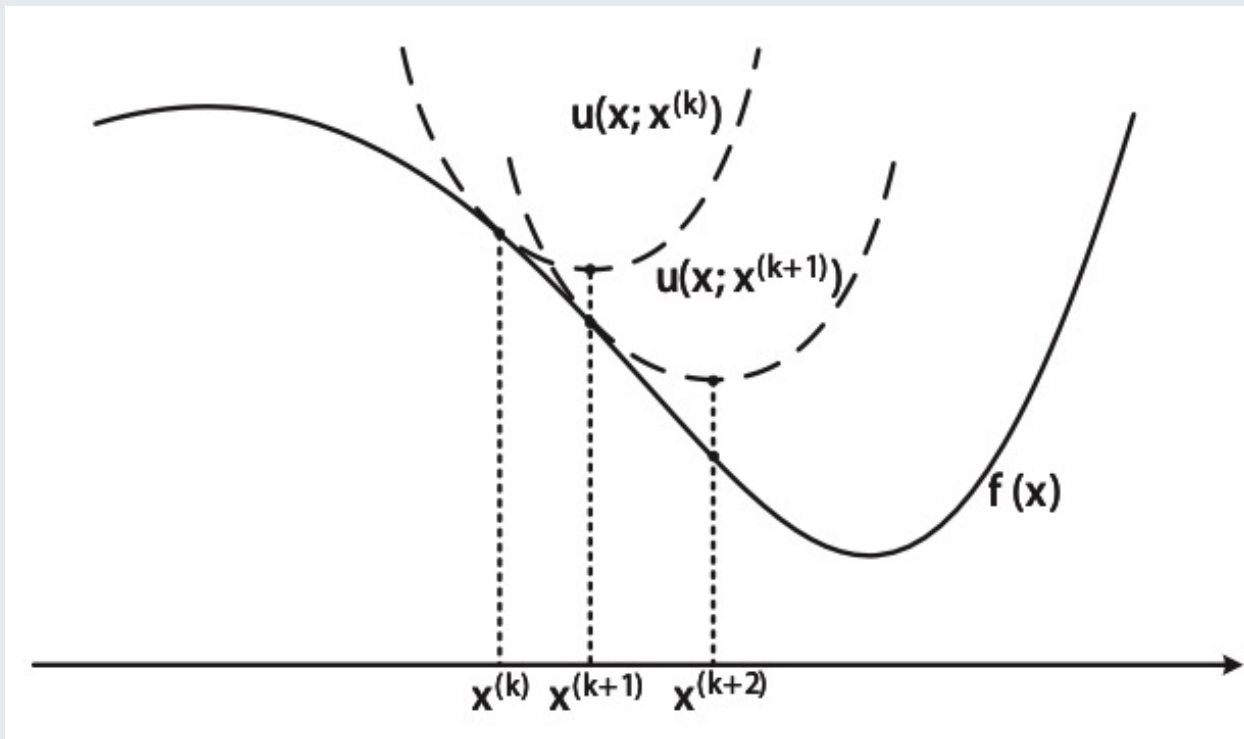Suppose x_0 is the initial point, in k-th step, we want $x_{k-1} \to x_k$.

Iterative algorithm

Suppose x_0 is the initial point, in k-th step, we want $x_{k-1} \rightarrow x_k$.

After k-th step | **(k+1)-th step** | (k+2)-th step

Iterative algorithm

Suppose x_0 is the initial point, in k-th step, we want $x_{k-1} \to x_k$.

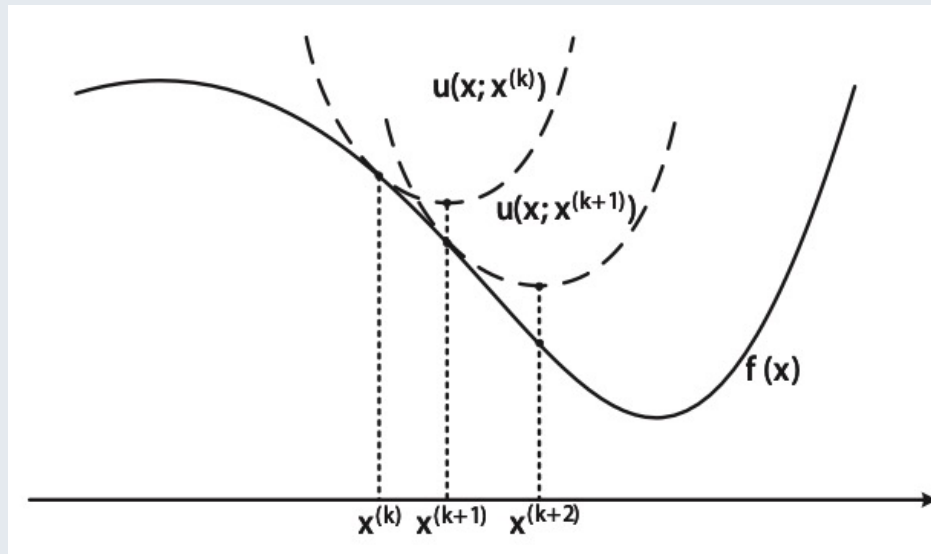| After k-th step	(k+1)-th step	(k+2)-th step

Construction rule of the surrogate / majorizer function *

$$u(\mathbf{y}, \mathbf{y}) = f(\mathbf{y}), \forall \mathbf{y} \in \mathcal{X}$$

$$u(\mathbf{x}, \mathbf{y}) \geq f(\mathbf{x}), \forall \mathbf{x}, \mathbf{y} \in \mathcal{X}$$

$$u'(\mathbf{x}, \mathbf{y}; \mathbf{d})\big|_{\mathbf{x}=\mathbf{y}} = f'(\mathbf{y}; \mathbf{d}), \forall \mathbf{d} \text{ with } \mathbf{y} + \mathbf{d} \in \mathcal{X}$$

$$u(\mathbf{x}, \mathbf{y}) \text{ is continuous in } \mathbf{x} \text{ and } \mathbf{y}$$

Question: how to construct $u\left(\mathbf{x}, \mathbf{x}^k\right)$?

Answer: that's more like an art :)

Luckily, the MM algorithm for l_1-norm minimization has been well established!

Majorizer for l_1-norm

- Consider the following quadratic majorizer of $f(t) = |t|$ for $t \neq 0$ (for simplicity we ignore this case)

$$u\left(t, t^k\right) = \frac{1}{2\left|t^k\right|}\left(t^2 + \left(t^k\right)^2\right).$$
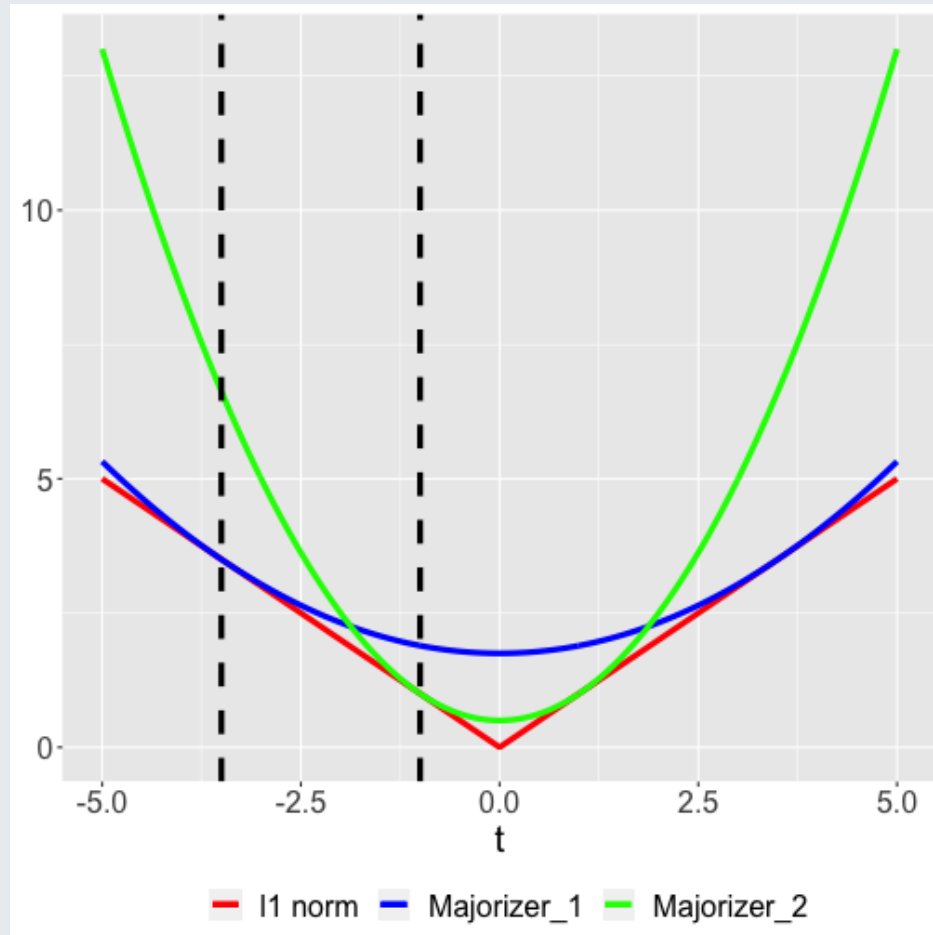
- It is a valid majorizer since it is continuous, and

$$u\left(t, t^k\right) \geq f(t),$$

$$u\left(t^k, t^k\right) = f(t),$$

$$\frac{d}{dt}u\left(t^k, t^k\right) = \frac{d}{dt}f\left(t^k\right).$$

Majorizer for l_1-norm

Reweighted LS for l_1-norm minimization

- Now we can apply it to the ℓ_1-norm: a quadratic majorizer of $f(\beta) = \|X\beta - y\|_1$ is

$$u\left(\beta, \beta^k\right) = \sum_{i=1}^{N} \frac{1}{2\left|[X\beta^k - y]_i\right|}\left([X\beta - y]_i^2 + \left(\left[X\beta^k - y\right]_i\right)^2\right)$$

- Now that we have the majorizer, we can write the MM iterative algorithm for $k = 0, 1, \ldots$ as

$$\underset{\beta}{\text{minimize}} \, \|(X\beta - y) \odot w^k\|_2^2$$

where $w_i^k = \sqrt{\frac{1}{2|[X\beta^k - y]_i|}}$.

Algorithm

- Set $k = 0$ and initialize with a feasible point β^0

- **repeat**

  - $w_i^k \leftarrow \sqrt{\dfrac{1}{2\left|[X\beta^k - y]_i\right|}}$

  - Update $\beta^{k+1} \leftarrow \underset{\beta}{\operatorname{argmin}} \left\| (X\beta - y) \odot w^k \right\|_2^2$

  - $k \leftarrow k + 1$

- **until** convergence

- **return** β^k

Implementation (first 4 steps)

Initialization   Call lm()   Call geom_smooth()   ggplot

```r
df <- data.frame(y = y_obs_ol, x = x_ol, weight = rep(0, length(x_ol)))

#beta_0 <- rnorm(1) # Initialize intercept
#beta_1 <- rnorm(1) # Initialize slope
beta_0 <- -3 # To demonstrate, we fit beta_0 = -3
beta_1 <- 1 # To demonstrate, we fit beta_1 = 1
```

Implementation (first 4 steps)

Initialization   **Call lm()**   Call geom_smooth()   ggplot

```r
opar <- par()
par(mfrow = c(2, 2))

for (k in 0:3) { # Here we only repeat 4 times
  fit_l1 <- lm(y ~ x, data = df, weights = sqrt(1 / 2 / abs(beta_0 + x

  beta_0 <- coef(fit_l1)[1] # Update intercept
  beta_1 <- coef(fit_l1)[2] # Update slope

  # Visualize
  plot(x_ol, y_obs_ol, pch = 16,
       xlab = "x", ylab = "y",
       cex = 1.5, cex.axis = 1.5, cex.lab = 1.5)
  abline(fit_l1, col = "blue", lwd = 3,
         cex = 1.5, cex.axis = 1.5, cex.lab = 1.5)
}
```

Implementation (first 4 steps)

Initialization   Call lm()   **Call geom_smooth()**   ggplot

```r
p_list <- list() # Figure list
for (k in c(1:4)) { # Here we only repeat 4 times
  df$weight <- sqrt(1 / 2 / abs(beta_0 + x_ol * beta_1 - y_obs_ol))
  fit_l1 <- lm(y ~ x, data = df, weights = weight) # Reweighted LS for
  beta_0 <- coef(fit_l1)[1] # Update intercept
  beta_1 <- coef(fit_l1)[2] # Update slope
  p <- ggplot(df, aes(x = x, y = y, size = weight / 3)) +
    geom_point(shape = 16) +
    geom_smooth(method = "lm", aes(weight = weight), size = 1.5, show.le
    theme(
      text = element_text(size = 18),
      legend.title = element_text(size = 12),
      legend.text = element_text(size = 11),
      legend.position = "bottom"
    )
  p_list <- c(p_list, list(p))
}
```
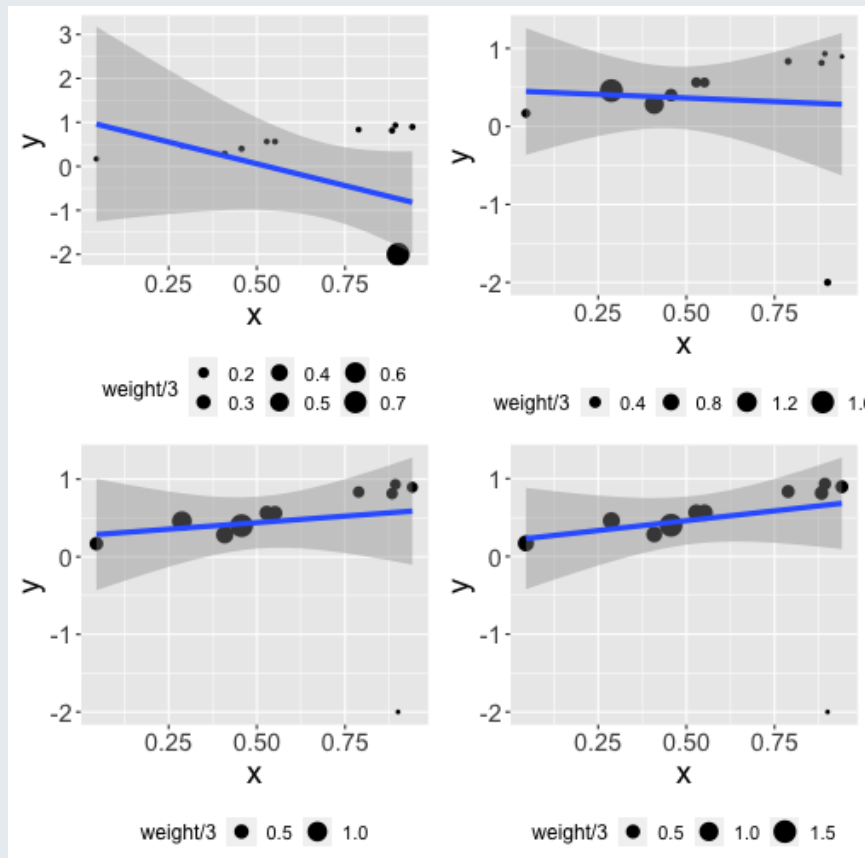
Implementation (first 4 steps)

Initialization   Call lm()   Call geom_smooth()   **ggplot**

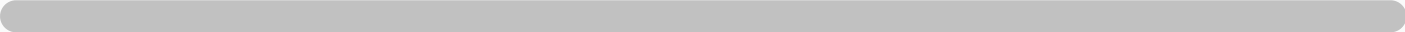# Full implementation

Initialization   l1 norm minimization   Visualization

```r
df <- data.frame(y = y_obs_ol, x = x_ol, weight = rep(0, length(x_ol)))

#beta_0 <- rnorm(1) # Initialize intercept
#beta_1 <- rnorm(1) # Initialize slope
beta_0 <- -3 # To demonstrate, we fit beta_0 = -3
beta_1 <- 1 # To demonstrate, we fit beta_1 = 1

tol <- 1e-6 # Convergence tolerance / criterion
iter_max <- 5000 # Maximum number of iterations
```

Full implementation

```r
l1_loss <- sum(abs(beta_0 + x_ol * beta_1 - y_obs_ol)) # l1 loss at init
for (k in 1:iter_max) {
  df$weight <- sqrt(1 / 2 / abs(beta_0 + x_ol * beta_1 - y_obs_ol))
  fit_l1 <- lm(y ~ x, data = df, weights = weight) # Reweighted LS for
  beta_0 <- coef(fit_l1)[1] # Update intercept
  beta_1 <- coef(fit_l1)[2] # Update slope

  l1_loss_current <- sum(abs(beta_0 + x_ol * beta_1 - y_obs_ol)) # l1 lo
  l1_loss <- c(l1_loss, l1_loss_current)

  # Whether converges
  if(abs((l1_loss[k + 1] - l1_loss[k]) / l1_loss[k]) < tol){
    break
  }
}
```
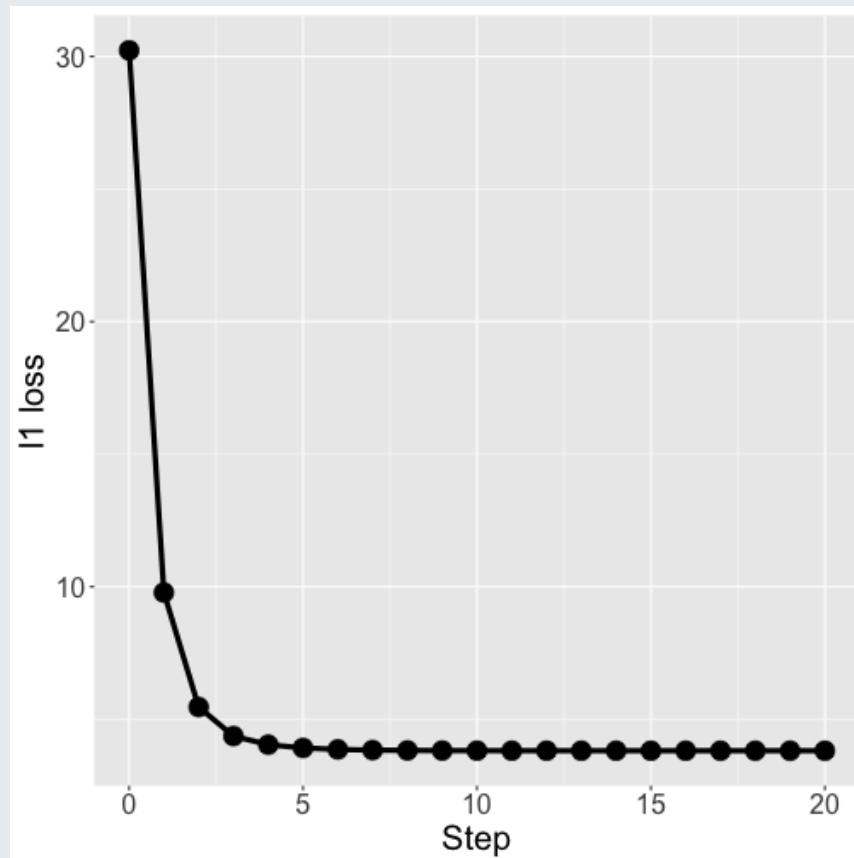
Full implementation

Initialization   l1 norm minimization   **Visualization**

Good night!

Slides created via Yihui Xie's R package **xaringan**.

Theme customized via Garrick Aden-Buie's R package **xaringanthemer**.

Tabbed panels created via Garrick Aden-Buie's R package **xaringanExtra**.

The chakra comes from **remark.js**, **knitr**, and **R Markdown**.