# Regression 3

**Industrial AI Lab.**

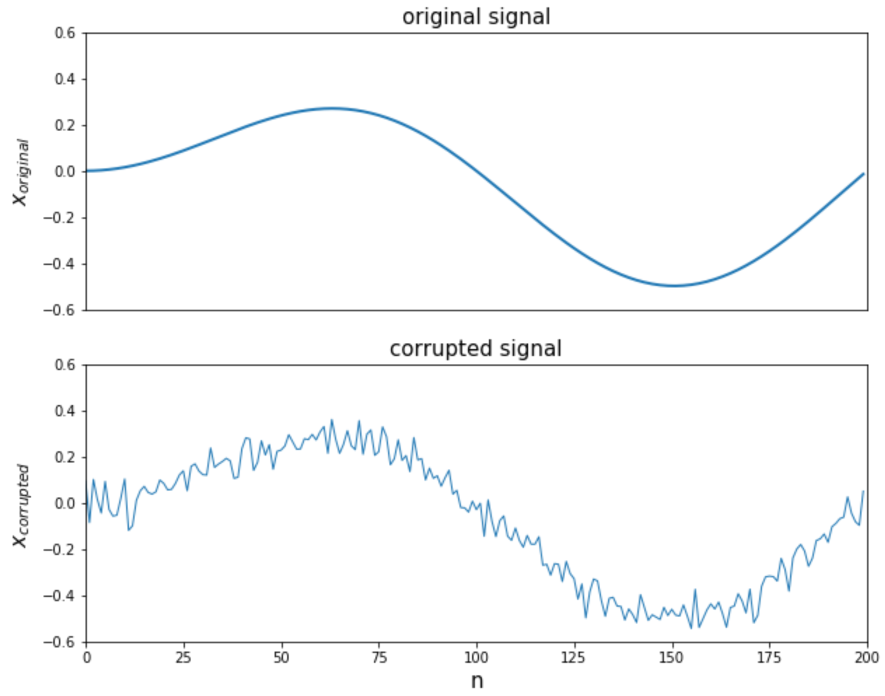
**Prof. Seungchul Lee**

# Linear Regression Examples

- De-noising
- Total Variation

# De-noising Signal

- We start with a signal represented by a vector $x \in \mathbb{R}^n$
  - $x_i$ corresponds to the value of some function of time, evaluated (or sampled) at evenly spaced points.
- Suppose $x$ is corrupted by some small, rapidly varying noise ε,
  - i.e. $x_{cor} = x + \varepsilon$



# Transform it to an Optimization Problem

- Transform de-noising in time into an optimization problem
- It is usually assumed that the signal does not vary too rapidly, which means that usually, we have $x_i \approx x_{i+1}$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \min_X \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

- $\mu$
  - to adjust the relative weight of the first and second terms
  - to controls the "smoothness" of $\hat{x}$

Source:
- Boyd & Vandenberghe's book "Convex Optimization"
- http://cvxr.com/cvx/examples/ (Figures 6.8-6.10: Quadratic smoothing)
- Week 4 of Linear and Integer Programming by Coursera of Univ. of Colorado

# Transform it to an Optimization Problem

$$\min_X \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

1) $X - X_{cor} = I_n X - X_{cor}$

2) $\sum (x_{k+1} - x_k)^2$ $\Longrightarrow$

$$\left\{ \begin{array}{l} (x_2 - x_1) - 0 = \begin{bmatrix} -1, & 1, & 0, & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - 0 \\ (x_3 - x_2) - 0 = \begin{bmatrix} 0, & -1, & 1, & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - 0 \\ \qquad\qquad \vdots \\ \Longrightarrow \left\| \underbrace{\begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{X} \underbrace{-}_{-} \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{0} \right\|_2^2 \end{array} \right.$$

# Least-Square Problems

$$\min_X \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

$$\|I_n X - X_{cor}\|_2^2 + \mu \|DX - 0\|_2^2 = \|Ax - b\|_2^2$$

$$= \left\| \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix} X - \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix} \right\|_2^2 \qquad \text{where } A = \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix}, \quad b = \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix}$$

- Then, plug $A, b$ into Python to numerically solve
- Note: de-noising is generally conducted by a low pass filter in the frequency domain

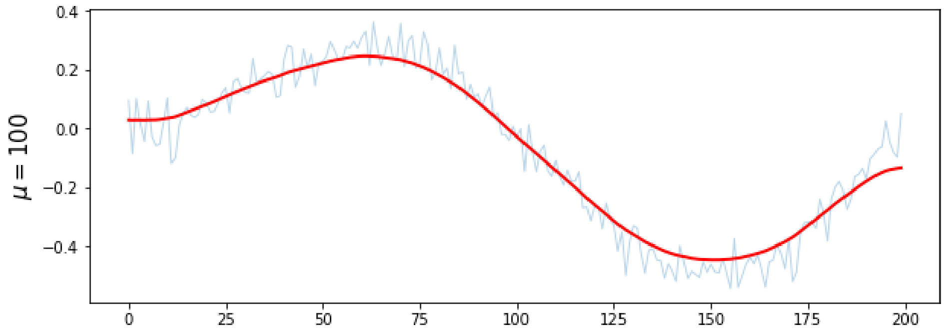
# Coded in Python

$$\left\| \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix} X - \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix} \right\|_2^2$$

where $A = \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix}, \quad b = \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix}$

$$\theta = (A^T A)^{-1} A^T y$$

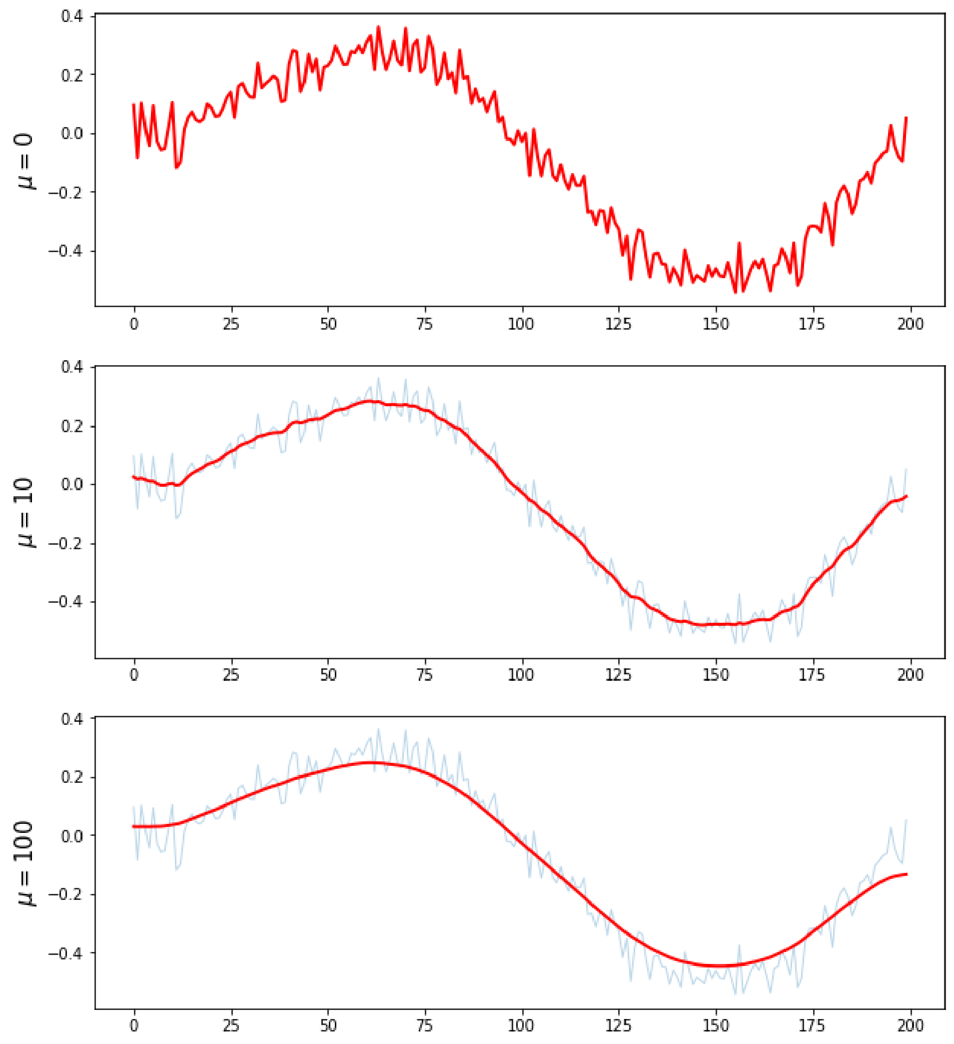
```
mu = 100

D = np.zeros([n-1, n])
D[:,0:n-1] -= np.eye(n-1)
D[:,1:n] += np.eye(n-1)
A = np.vstack([np.eye(n), np.sqrt(mu)*D])

b = np.vstack([x_cor, np.zeros([n-1,1])])

A = np.asmatrix(A)
b = np.asmatrix(b)

x_reconst = (A.T*A).I*A.T*b
```



# See How $\mu$ Affects Smoothing Results

$$\min_{X} \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

```
mu = [0, 10, 100];

for i in range(len(mu)):
    A = np.vstack([np.eye(n), np.sqrt(mu[i])*D])
    b = np.vstack([x_cor, np.zeros([n-1,1])])

    A = np.asmatrix(A)
    b = np.asmatrix(b)

    x_reconst = (A.T*A).I*A.T*b

    plt.subplot(3,1,i+1)
    plt.plot(t, x_cor, '-', linewidth = 1, alpha = 0.3)
    plt.plot(t, x_reconst, 'r', linewidth = 2)
    plt.ylabel('$\mu = {}$'.format(mu[i]), fontsize = 15)

plt.show()
```

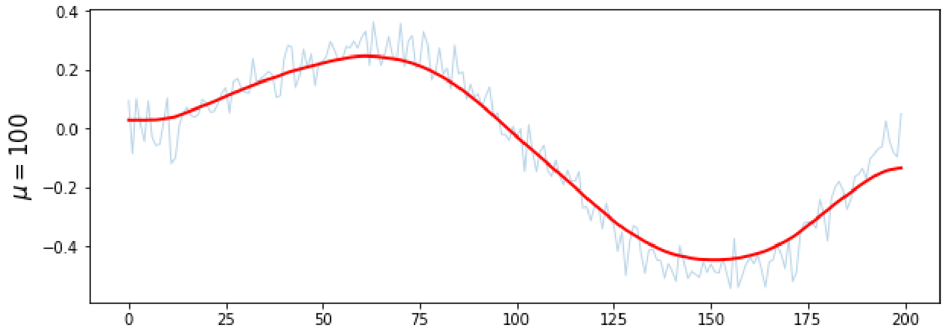


# CVXPY Implementation

$$\min \ \left\{ \|x - x_{cor}\|_2^2 + \mu \|Dx\|_2^2 \right\}$$

$$\min_X \left\{ \underbrace{\|(X - X_{cor})\|_2^2}_{\text{how much } x \text{ deviates from } x_{cor}} + \mu \underbrace{\sum_{k=1}^{n-1} (x_{k+1} - x_k)^2}_{\text{penalize rapid changes of } X} \right\}$$

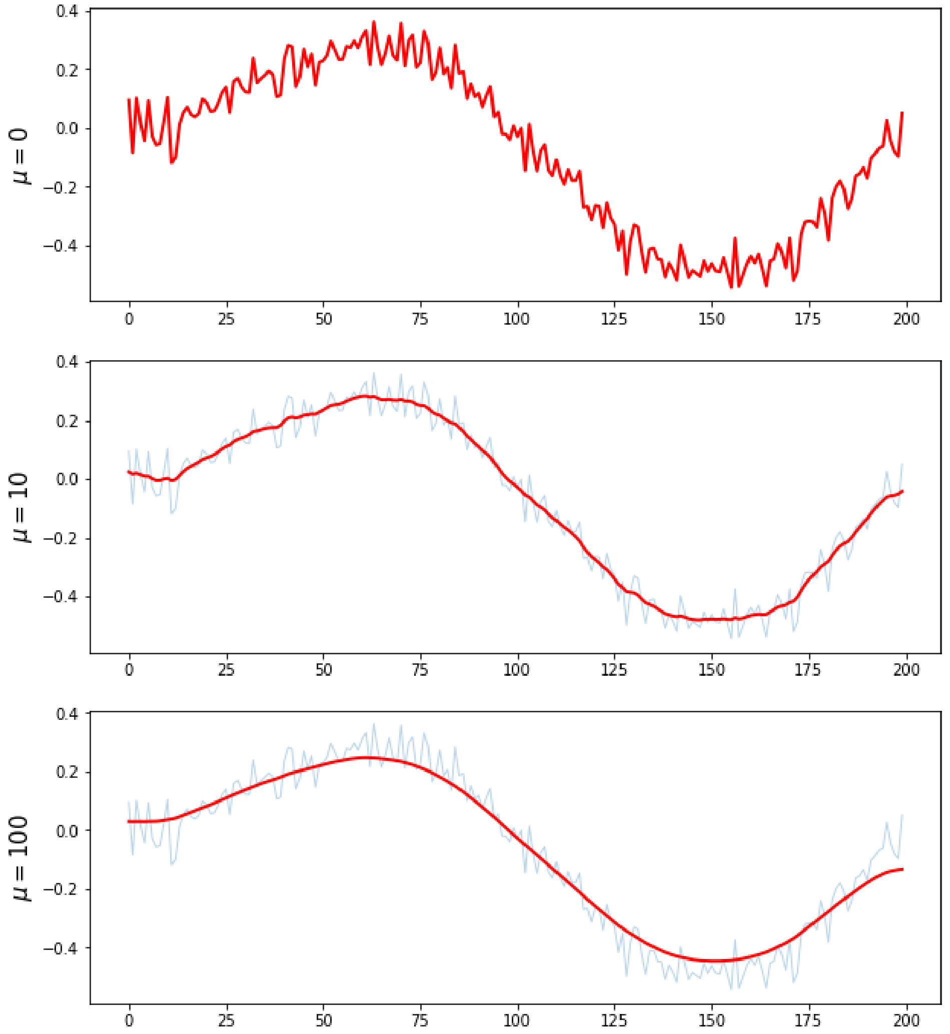
```
mu = 100

x_reconst = cvx.Variable([n,1])
#obj = cvx.Minimize(cvx.sum_squares(x_reconst-x_cor) + mu*cvx.sum_squares(x_reconst[1:n]-x_reconst[0:n-1]))
obj = cvx.Minimize(cvx.sum_squares(x_reconst-x_cor) + mu*cvx.sum_squares(D*x_reconst))

prob = cvx.Problem(obj).solve()
```



# CVXPY: See How $\mu$ Affects Smoothing Results

$$\min \; \left\{ \|x - x_{cor}\|_2^2 + \mu \|Dx\|_2^2 \right\}$$

```python
mu = [0, 10, 100]

for i in range(len(mu)):
    x_reconst = cvx.Variable([n,1])
    obj = cvx.Minimize(cvx.sum_squares(x_reconst - x_cor) +
                       mu[i]*cvx.sum_squares(D*x_reconst))
    prob = cvx.Problem(obj).solve()

    plt.subplot(3,1,i+1)
    plt.plot(t,x_cor,'-', linewidth = 1, alpha = 0.3)
    plt.plot(t,x_reconst.value, 'r', linewidth = 2)
    plt.ylabel('$\mu = {}$'.format(int(mu[i])), fontsize = 15)

plt.show()
```

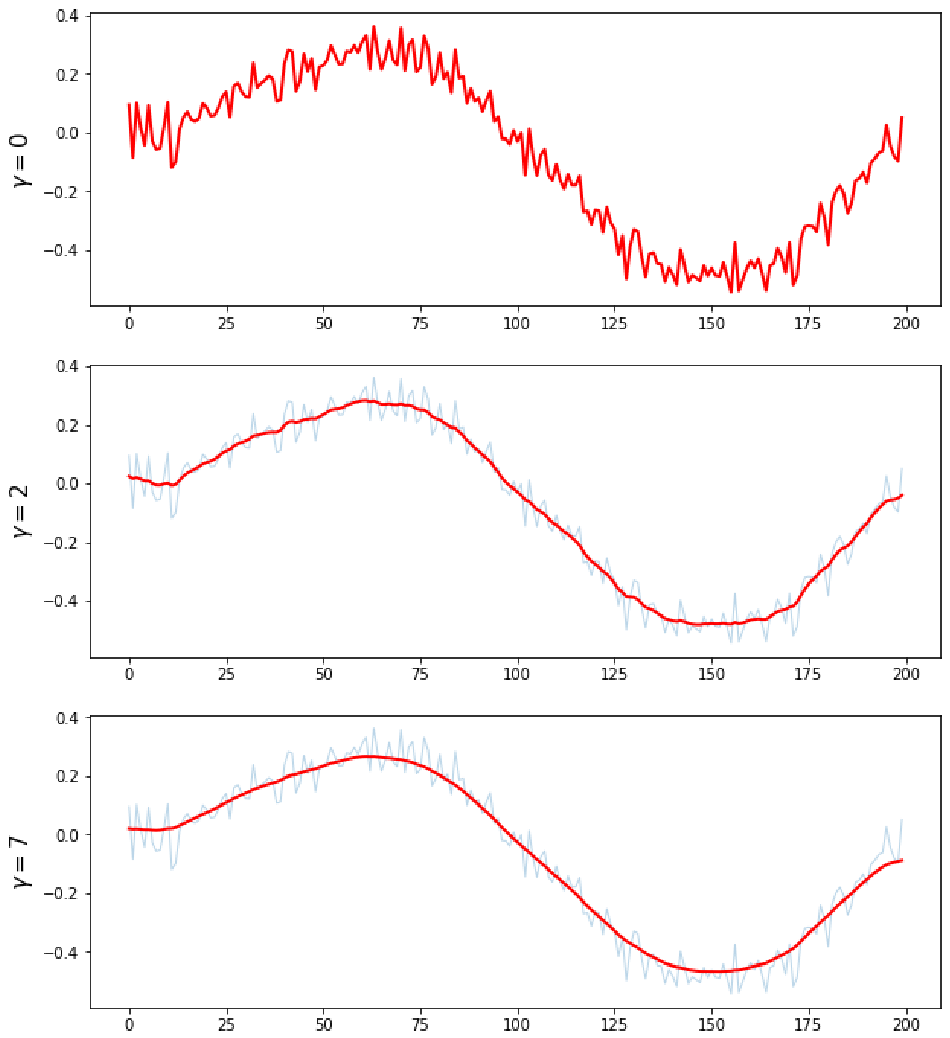


# $L_2$ Norm

$$\min \ \{\|x - x_{cor}\|_2^2 + \mu\|Dx\|_2^2\}$$

$$\min \ \{\|x - x_{cor}\|_2 + \gamma\|Dx\|_2\}$$

```
plt.figure(figsize=(10, 12))

gammas = [0, 2, 7]

for i in range(len(gammas)):
    x_reconst = cvx.Variable([n,1])
    obj = cvx.Minimize(cvx.norm(x_reconst-x_cor, 2) + gammas[i]*(cvx.norm(D*x_reconst, 2)))
    prob = cvx.Problem(obj).solve()

    plt.subplot(3,1,i+1)
    plt.plot(t,x_cor,'-', linewidth = 1, alpha = 0.3)
    plt.plot(t,x_reconst.value, 'r', linewidth = 2)
    plt.ylabel('$ \gamma = {}$'.format(gammas[i]), fontsize = 15)

plt.show()
```



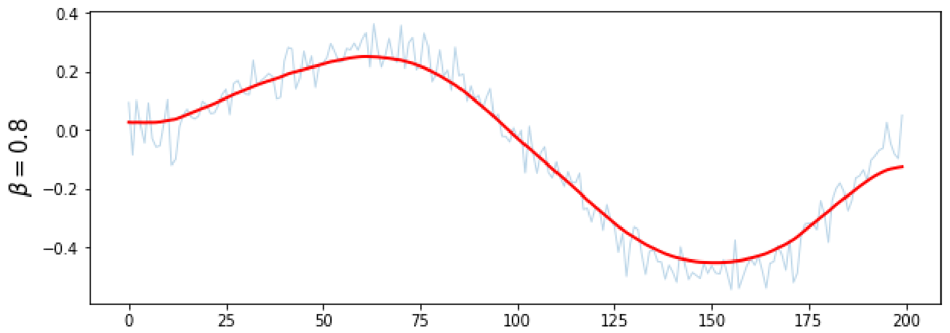
# $L_2$ Norm with a Constraint

$$\min \; \{\|x - x_{cor}\|_2^2 + \mu \|Dx\|_2^2\}$$

$$\min \; \{\|x - x_{cor}\|_2 + \gamma \|Dx\|_2\}$$

$$\begin{aligned} &\min \|Dx\|_2 \\ s.t. \quad &\|x - x_{cor}\|_2 < \beta \end{aligned}$$

```
beta = 0.8

x_reconst = cvx.Variable([n,1])
obj = cvx.Minimize(cvx.norm(D*x_reconst, 2))
const = [cvx.norm(x_reconst-x_cor, 2) <= beta]
prob = cvx.Problem(obj, const).solve()

plt.figure(figsize=(10, 4))
plt.plot(t,x_cor,'-', linewidth = 1, alpha = 0.3)
plt.plot(t,x_reconst.value, 'r', linewidth = 2)
plt.ylabel(r'$\beta = {}$'.format(beta), fontsize = 15)
plt.show()
```

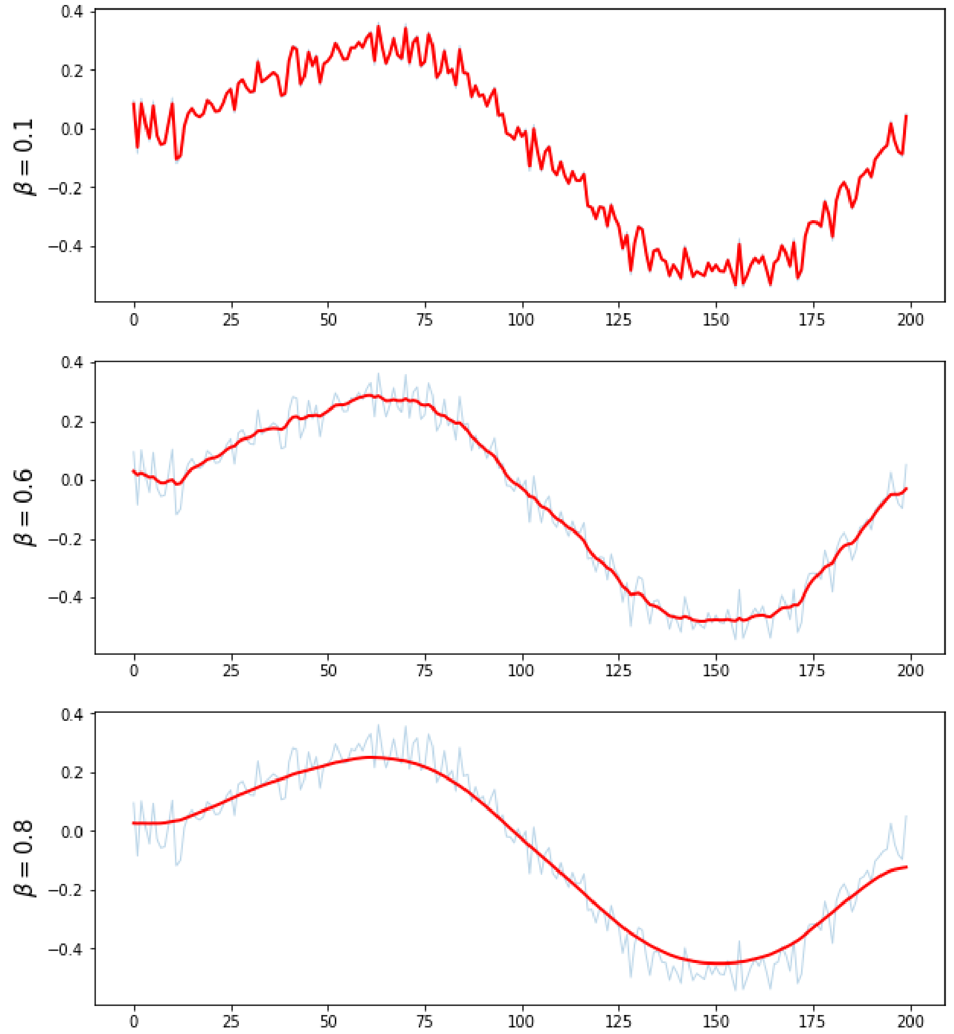


# $L_2$ Norm with a Constraint

$$\min \; \{\|x - x_{cor}\|_2^2 + \mu \|Dx\|_2^2\}$$

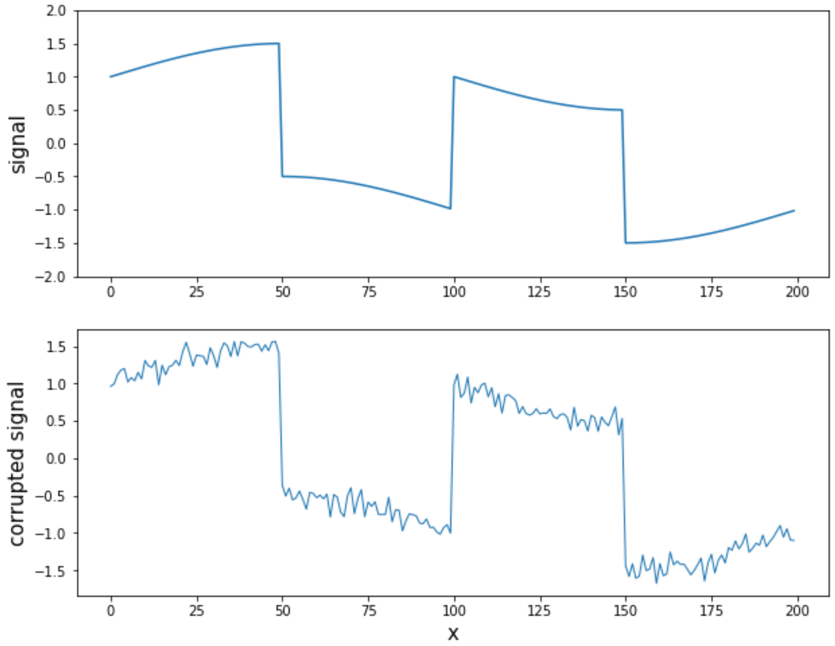
$$\min \; \{\|x - x_{cor}\|_2 + \gamma \|Dx\|_2\}$$

$$\begin{aligned} &\min \; \|Dx\|_2 \\ s.t. \quad &\|x - x_{cor}\|_2 < \beta \end{aligned}$$



# Signal with Sharp Transition + Noise

- Suppose we have a signal $x$, which is mostly smooth, but has several rapid variations (or jumps).



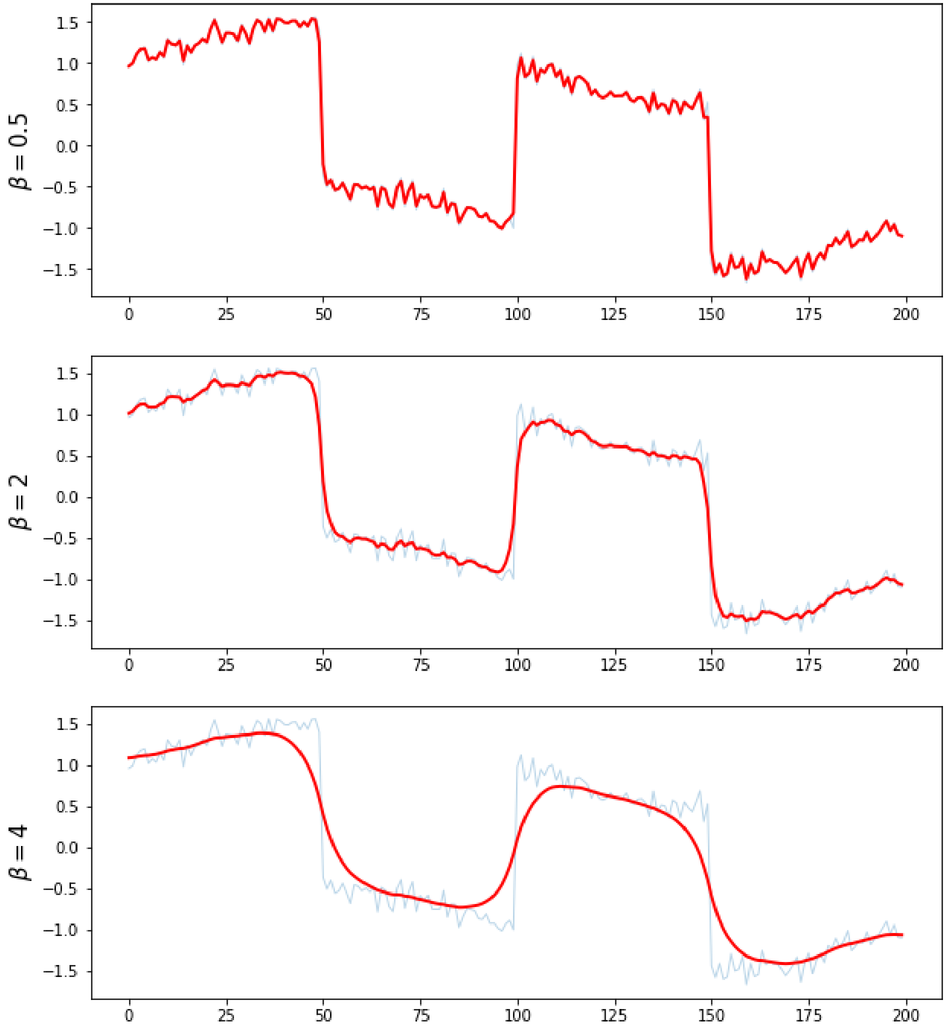

- First, apply the same method that we used for smoothing signals before
- known as a *total variation problem*



Chapter 6.3 from Boyd & Vandenberghe's book "Convex Optimization"

# Quadratic Smoothing ($L_2$ Norm)

$$
\begin{aligned}
&\min \; \|Dx\|_2 \\
s.t. \quad &\|x - x_{cor}\|_2 < \beta
\end{aligned}
$$




- Quadratic smoothing smooths out both *noise and sharp transitions* in signal, but this is not what we want
- We will not be able to preserve the signal's sharp transitions.

- Any ideas ?

# $L_1$ Norm

- We can instead apply total variation reconstruction on the signal by solving

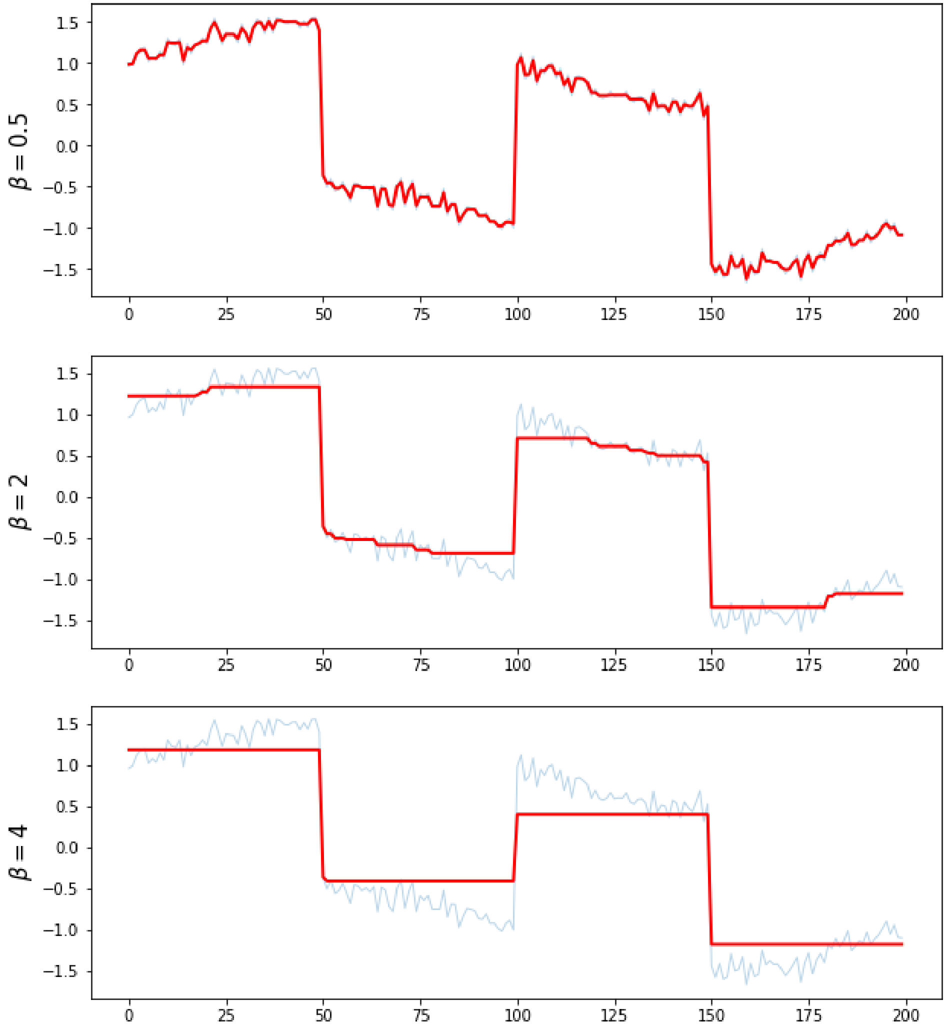
$$\min \|x - x_{cor}\|_2 + \lambda \sum_{i=1}^{n-1} |x_{i+1} - x_i|$$

where the parameter $\lambda$ controls the “smoothness” of $x$

$$\begin{aligned} & \min \|Dx\|_2 \\ s.t. \quad & \|x - x_{cor}\|_2 < \beta \end{aligned} \quad \Longrightarrow \quad \begin{aligned} & \min \|Dx\|_1 \\ s.t. \quad & \|x - x_{cor}\|_2 < \beta \end{aligned}$$

# $L_1$ Norm

$$\min \|Dx\|_1$$
$$s.t. \quad \|x - x_{cor}\|_2 < \beta$$




- Total Variation (TV) smoothing preserves sharp transitions in signal, and this is not bad

- Note that how TV reconstruction does a better job of preserving the sharp transitions in the signal while removing the noise.

# Total Variation Image

- Q: Apply $L_1$ norm to the image, and guess what kind of an image will be produced ?

```
n = row*col
imbws = resized_imbw.reshape(-1, 1)
```

Idea comes from http://www2.compute.dtu.dk/~pcha/mxTV/

# Total Variation Image

$$\min \ \|Dx\|_1$$
$$s.t. \quad \|x - x_{cor}\|_2 < \beta$$

```
n = row*col
imbws = resized_imbw.reshape(-1, 1)

beta = 1500

x = cvx.Variable([n,1])
obj = cvx.Minimize(cvx.norm(x[1:n] - x[0:n-1],1))
const = [cvx.norm(x - imbws,2) <= beta]
prob = cvx.Problem(obj, const).solve()

imbwr = x.value.reshape(row, col)

plt.figure(figsize = (8,8))
plt.imshow(imbwr,'gray')
plt.axis('off')
plt.show()
```

Idea comes from http://www2.compute.dtu.dk/~pcha/mxTV/