Regression 3

Industrial AI Lab.

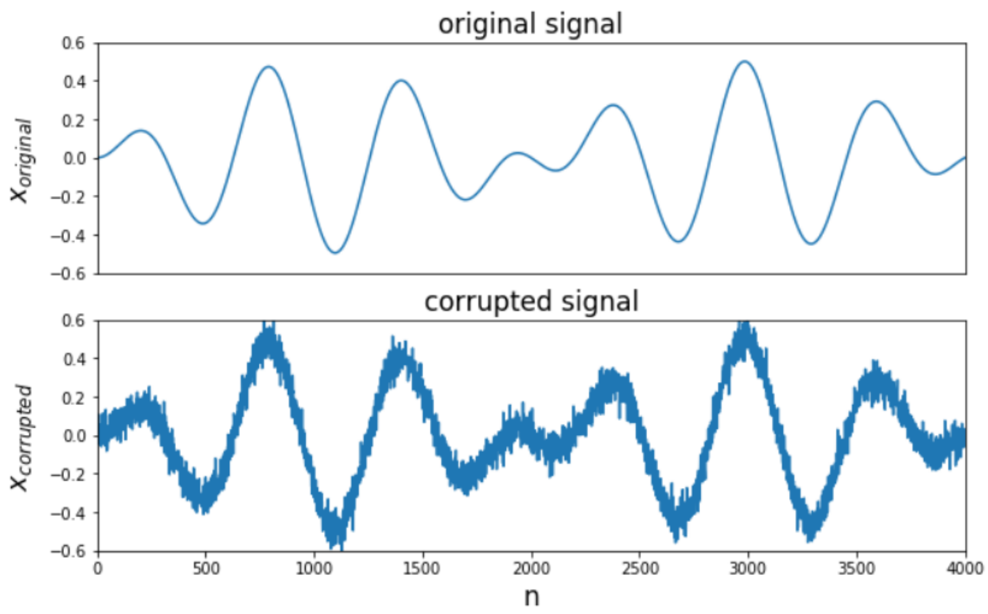
Linear Regression Examples

- 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.

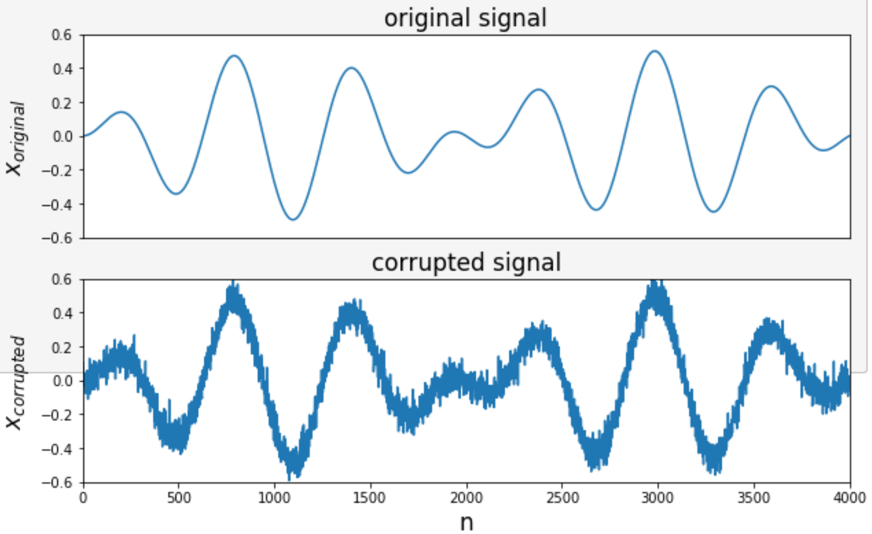
- Suppose x is corrupted by some small, rapidly varying noise v,
  - i.e. $x_{cor} = x + v$



Corrupted-Signal Generation

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cvx

%matplotlib inline
```

```
n = 4000
t = np.arange(n).reshape(-1,1)
x = 0.5 * np.sin((2*np.pi/n)*t) * (np.sin(0.01*t))
x_cor = x + 0.05*np.random.randn(n,1)

plt.figure(figsize=(10, 6))
plt.subplot(2,1,1)
plt.plot(t,x,'-')
plt.axis([0, n, -0.6, 0.6])
plt.xticks([])
plt.title('original signal' , fontsize = 17)
plt.ylabel('$x_{original}$', fontsize = 17)

plt.subplot(2,1,2)
plt.plot(t, x_cor,'-')
plt.axis([0, n, -0.6, 0.6])
plt.title('corrupted signal', fontsize = 17)
plt.xlabel('n', fontsize = 17)
plt.ylabel('$x_{corrupted}$', fontsize = 17)
plt.show()
```



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 $x_i \approx x_{i+1}$

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

- μ:
  - 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

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

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

$$(x_2 - x_1) - 0 = [-1, \quad 1, \quad 0, \quad \cdots \quad 0] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - 0$$

$$(x_3 - x_2) - 0 = [0, \quad -1, \quad 1, \quad \cdots \quad 0] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - 0$$

$$\vdots$$

$$\Longrightarrow \left\| \underbrace{\begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{X} \underbrace{-}_{-} \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{0} \right\|_2^2$$

Least-Square Problems

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

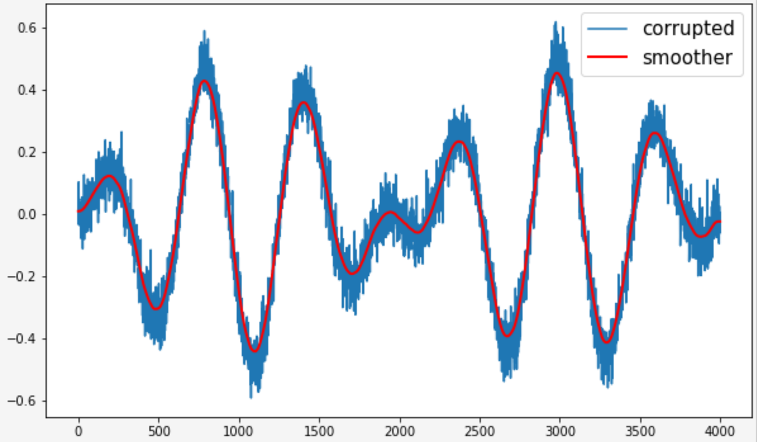
$$\begin{aligned} \left\| I_n X - X_{cor} \right\|_2^2 + \mu \left\| DX - 0 \right\|_2^2 &= \|Ax - b\|_2^2 \\ &= \left\| \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix} X - \begin{bmatrix} X_{cor} \\ 0 \end{bmatrix} \right\|_2^2 \\ \text{where } A = \begin{bmatrix} I_n \\ \sqrt{\mu} D \end{bmatrix},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

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

Coded in Python

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

```
n = 4000
mu = 1000

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)
```



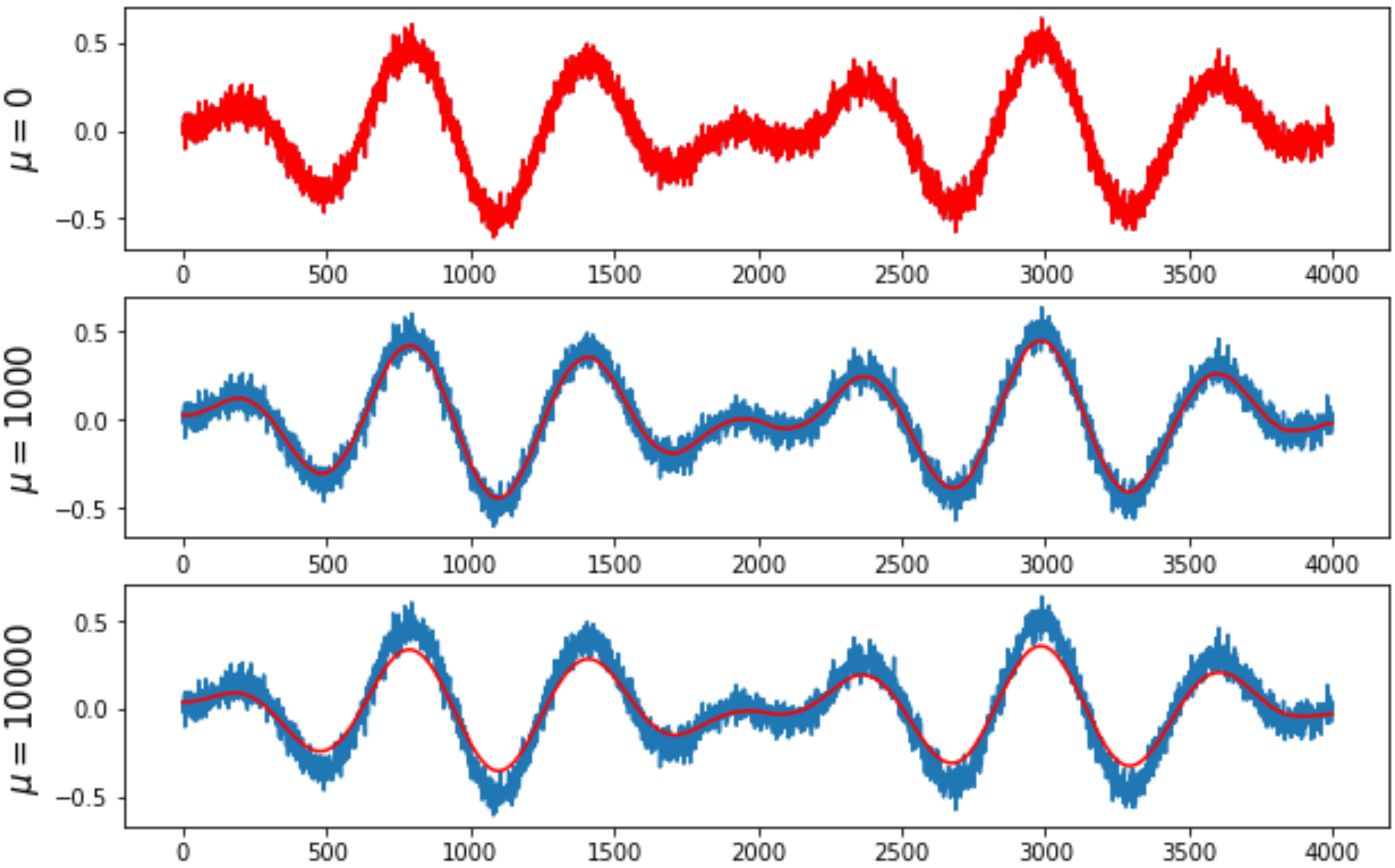

```
x_reconst = (A.T*A).I*A.T*b

plt.figure(figsize=(10, 6))
plt.plot(t, x_cor, '-', label='corrupted');
plt.plot(t, x_reconst, 'r', label='smoother', linewidth=2)
plt.legend(fontsize=15)
plt.show()
```

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

See How μ Affects Smoothing Results

```
plt.figure(figsize=(10, 9))
mu = [0, 1000, 1e4];

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(4,1,i+1)
    plt.plot(t, x_cor, '-')
    plt.plot(t, x_reconst, 'r')
    plt.ylabel('$\mu = {}$'.format(mu[i]), fontsize=15)

plt.show()
```

De-noised Signals

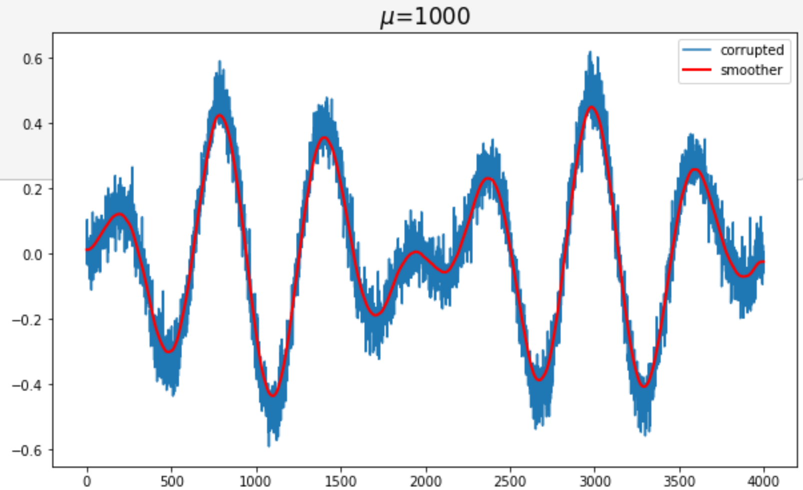


Use CVXPY

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

```
n = 4000
mu = 1000

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('SCS')
```

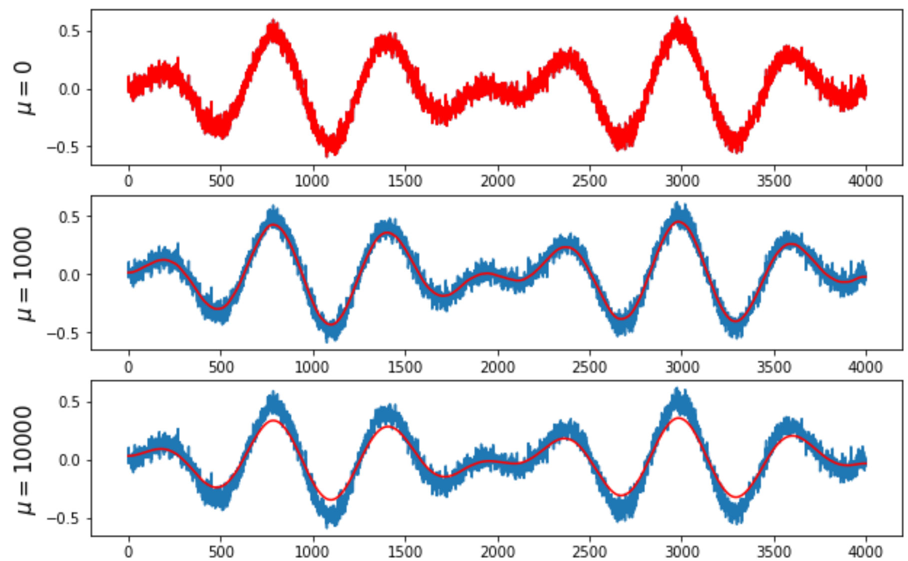
```
plt.figure(figsize=(10, 6))
plt.plot(t, x_cor)
plt.plot(t, x_reconst.value, 'r', linewidth = 2)
title = '=%s' % str(mu)
plt.title('$\mu$' + title, fontsize = 17)
plt.legend(['corrupted', 'smoother'], loc = 1)
plt.show()
```



Use CVXPY

```
plt.figure(figsize=(10, 9))

mu = [0, 1000, 1e4]

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('SCS')

    plt.subplot(4,1,i+1)
    plt.plot(t,x_cor,'-')
    plt.plot(t,x_reconst.value,'r')
    plt.ylabel('$\mu = {}$'.format(int(mu[i])), fontsize=15)

plt.show()
```

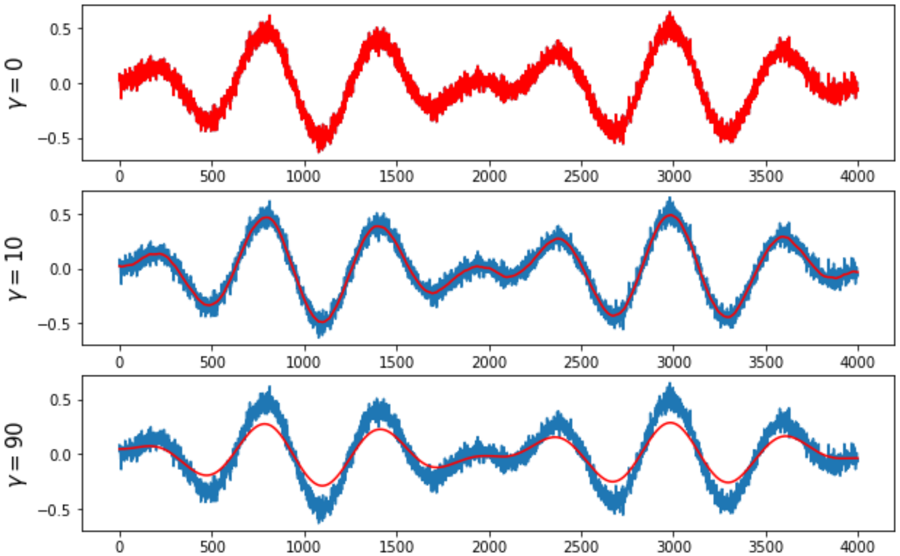
$$\boxed{\min \; \left\{ \|x - x_{cor}\|_2^2 + \mu \|Dx\|_2^2 \right\}}$$



L_2 Norm

```
plt.figure(figsize=(10, 9))

gammas = [0, 10, 90]

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

    plt.subplot(4,1,i+1)
    plt.plot(t,x_cor,'-')
    plt.plot(t,x_reconst.value,'r')
    plt.ylabel('$ \gamma = {}$'.format(int(gammas[i])), fontsize=15)

plt.show()
```

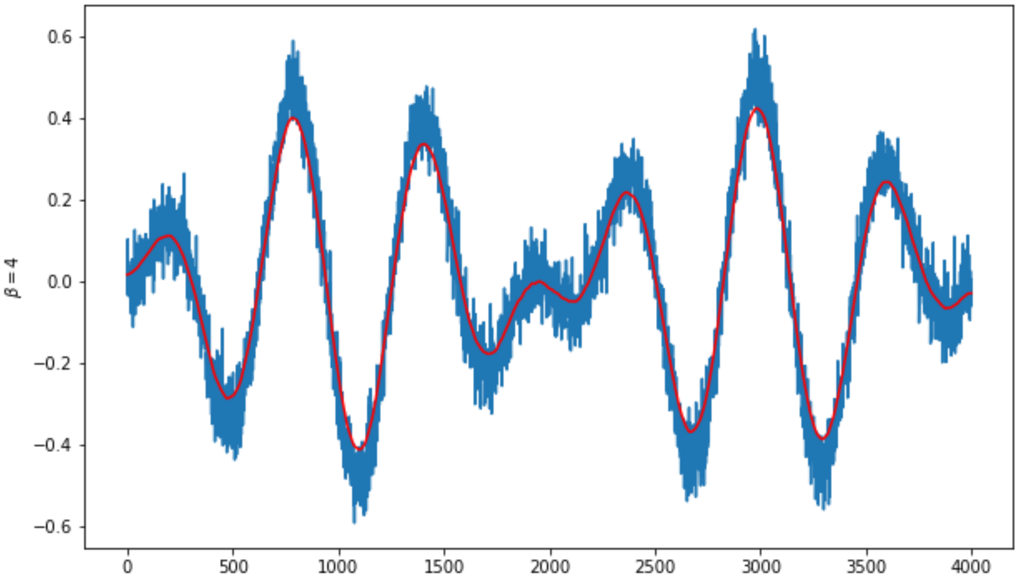
$$\min \ \{\|x - x_{cor}\|_2 + \gamma\|Dx\|_2\}$$



L_2 Norm with a Constraint

```
plt.figure(figsize=(10, 6))

beta = 4

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

plt.plot(t,x_cor,'-')
plt.plot(t,x_reconst.value,'r')
plt.ylabel(r'$\beta = {}$'.format(beta))
plt.show()
```

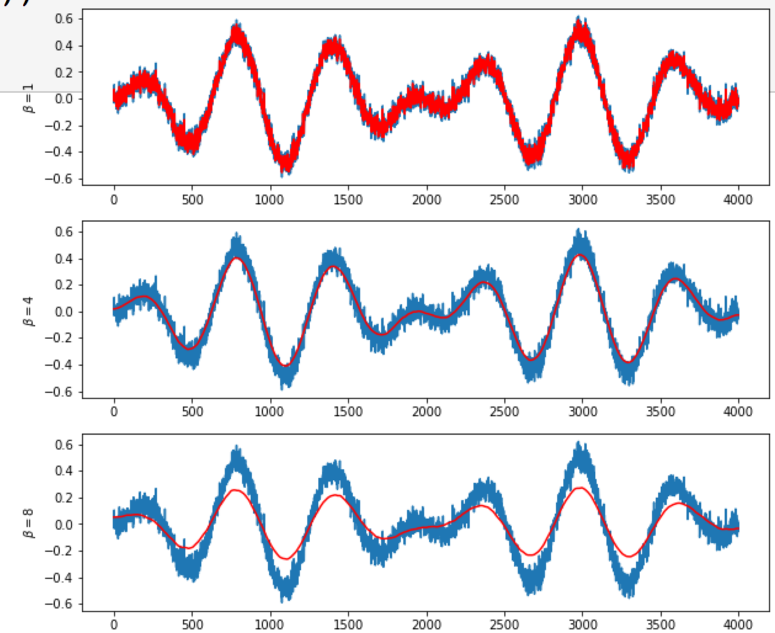
$$\begin{aligned} &\min \; \lVert Dx \rVert_2 \\ s.t. \quad &\lVert x - x_{cor} \rVert_2 < \beta \end{aligned}$$



L_2 Norm with a Constraint

```
plt.figure(figsize=(10, 9))

beta = [1, 4, 8]

for i in range(len(beta)):

    x_reconst = cvx.Variable(n)
    obj = cvx.Minimize(cvx.norm(x_reconst[1:n] - x_reconst[0:n-1], 2))
    const = [cvx.norm(x_reconst-x_cor, 2) <= beta[i]]
    prob = cvx.Problem(obj, const).solve(solver='SCS')

    plt.subplot(len(beta),1,i+1)
    plt.plot(t,x_cor,'-')
    plt.plot(t,x_reconst.value,'r')
    plt.ylabel(r'$\beta = {}$'.format(int(beta[i])))

plt.show()
```

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



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*

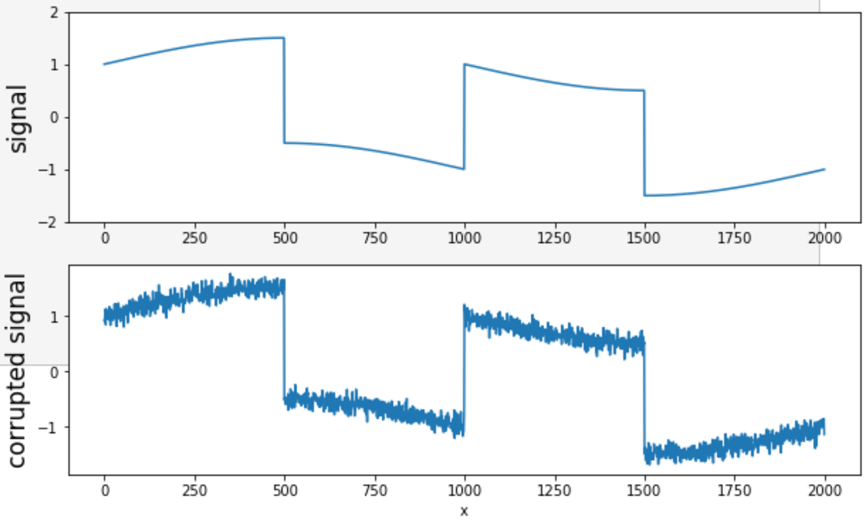
  – Source:

    - Chapter 6.3 from Boyd & Vandenberghe's book "Convex Optimization"
    - m files of total variation reconstruction

Signal with Sharp Transition + Noise

```
n = 2000
t = np.arange(n).reshape(-1,1)

exact = np.vstack([np.ones([500,1]), -np.ones([500,1]), \
                   np.ones([500,1]), -np.ones([500,1])])

x = exact + 0.5*np.sin((2*np.pi/n)*t)
exact_variation = np.sum(np.abs(exact[1:n] - exact[0:n-1]))

noise = 0.1*np.random.randn(n,1)
x_cor = x + noise
noise_variation = np.sum(np.abs(x_cor[1:n] - x_cor[0:n-1]))
```

```
plt.figure(figsize=(10, 6))

plt.subplot(2,1,1)
plt.plot(t, x)
plt.ylim([-2.0,2.0])
plt.ylabel('signal', fontsize = 17)

plt.subplot(2,1,2)
plt.plot(t, x_cor)
plt.ylabel('corrupted signal' , fontsize = 17)
plt.xlabel('x')
plt.show()
```

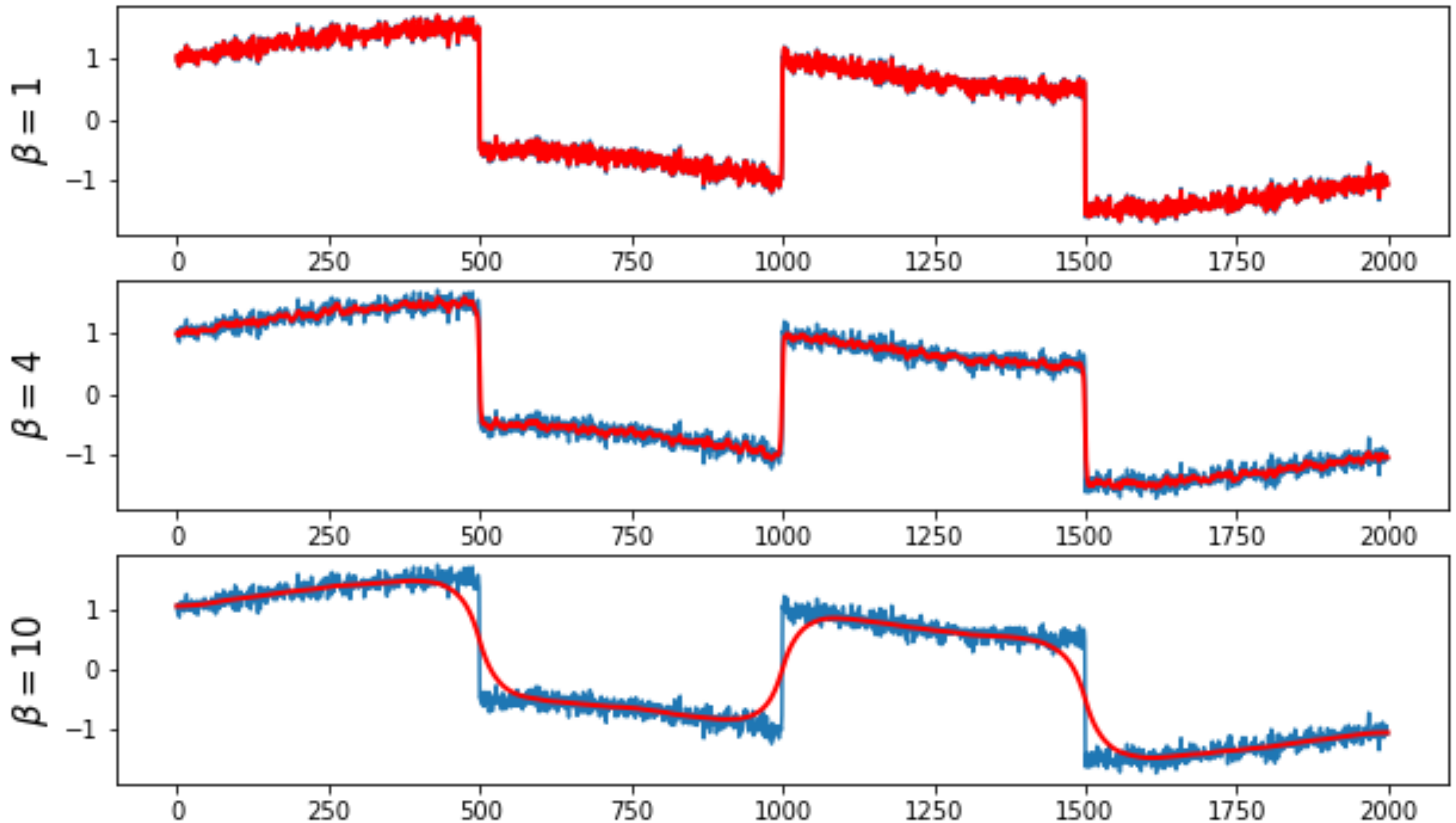


Quadratic Smoothing (L_2 Norm)

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

```
plt.figure(figsize=(10, 6))

beta = [1, 4, 10]

for i in range(len(beta)):
    x_reconst = cvx.Variable(n)
    obj = cvx.Minimize(cvx.norm(x_reconst[1:n] - x_reconst[0:n-1], 2))
    const = [cvx.norm(x_reconst-x_cor, 2) <= beta[i]]
    prob = cvx.Problem(obj, const).solve('SCS')

    plt.subplot(len(beta), 1, i+1)
    plt.plot(t, x_cor)
    plt.plot(t, x_reconst.value, 'r', linewidth=2)
    plt.ylabel(r'$\beta = {}$'.format(beta[i]), fontsize=15)

plt.show()
```

De-noised Signal



Quadratic Smoothing (L_2 Norm)

- 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 λ controls the "smoothness" of x

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

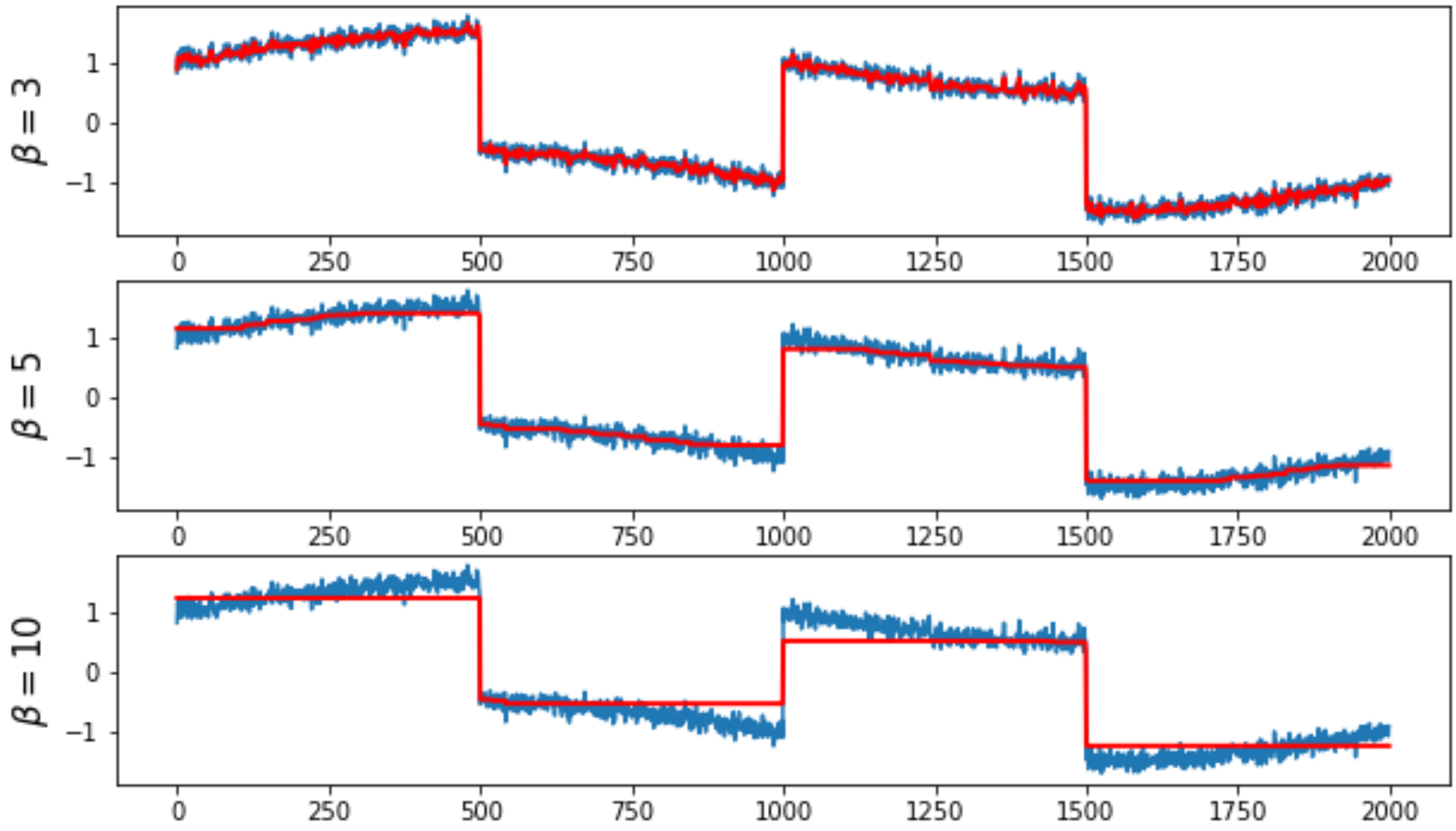
L_1 Norm

$$\min \|Dx\|_1$$

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

```
plt.figure(figsize=(10, 6))

beta = [3, 5, 10]

for i in range(len(beta)):
    x_reconst = cvx.Variable(n)
    obj = cvx.Minimize(cvx.norm(x_reconst[1:n] - x_reconst[0:n-1], 1))
    const = [cvx.norm(x_reconst-x_cor, 2) <= beta[i]]
    prob = cvx.Problem(obj, const).solve()

    plt.subplot(len(beta), 1, i+1)
    plt.plot(t, x_cor)
    plt.plot(t, x_reconst.value, 'r', linewidth=2)
    plt.ylabel(r'$\beta = {}$'.format(beta[i]), fontsize=15)

plt.show()
```

De-noised Signal



Total Variation

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

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

Total Variation Image Reconstruction

```
import scipy as sc

imbw = sc.misc.imread('./image_files/dog.jpg','L')
plt.imshow(imbw,'gray')
plt.xticks([])
plt.yticks([])
plt.show()
```

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

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

Total Variation Image Reconstruction

```
row, col = imbw.shape
n = row * col

imbws = imbw.reshape(-1, 1)

beta = 2400

x = cvx.Variable(n)
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('SCS')

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

plt.imshow(imbwr,'gray')
plt.xticks([])
plt.yticks([])
plt.show()
```

$$\min \ \|Dx\|_1$$

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

Total Variation Image Reconstruction

```
row, col = imbw.shape
n = row * col

imbws = imbw.reshape(-1, 1)

beta = 2400

x = cvx.Variable(n)
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('SCS')

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

plt.imshow(imbwr,'gray')
plt.xticks([])
plt.yticks([])
plt.show()
```

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

- Cartoonish effect