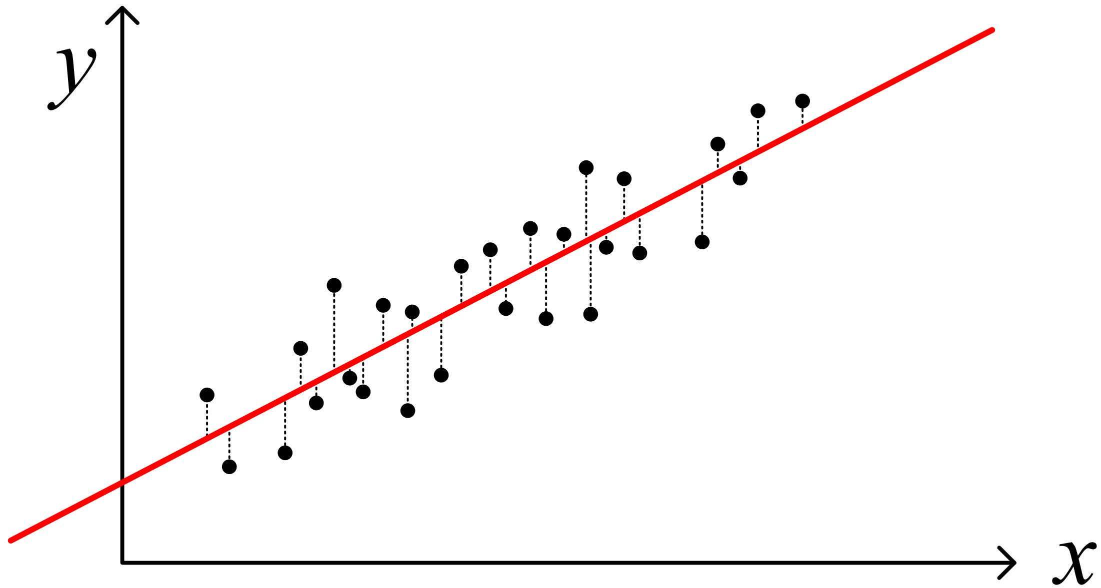
Regression 1

Industrial AI Lab.

Assumption: Linear Model

$$\hat{y}_i = f(x_i\,;\theta) \text{ in general}$$



- In many cases, a linear model to predict y_i is used

$$\hat{y}_i = \theta_1 x_i + \theta_2$$

Linear Regression

- Considering linear regression
  - Easy to extend to more complex predictions later on

$$\text{Given} \begin{cases} x_i & : \text{inputs} \\ y_i & : \text{outputs} \end{cases}, \text{Find } \theta_1 \text{ and } \theta_2$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \theta_1 x_i + \theta_2$$

  - $\hat{y}_i$: predicted output
  - $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$: model parameters

Re-cast Problem as Least Squares

- For convenience, we define a function that maps inputs to feature vectors, ϕ

$$
\begin{aligned}
\hat{y}_i &= \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \\
&= \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \\
&= \phi^T(x_i)\theta
\end{aligned}
$$

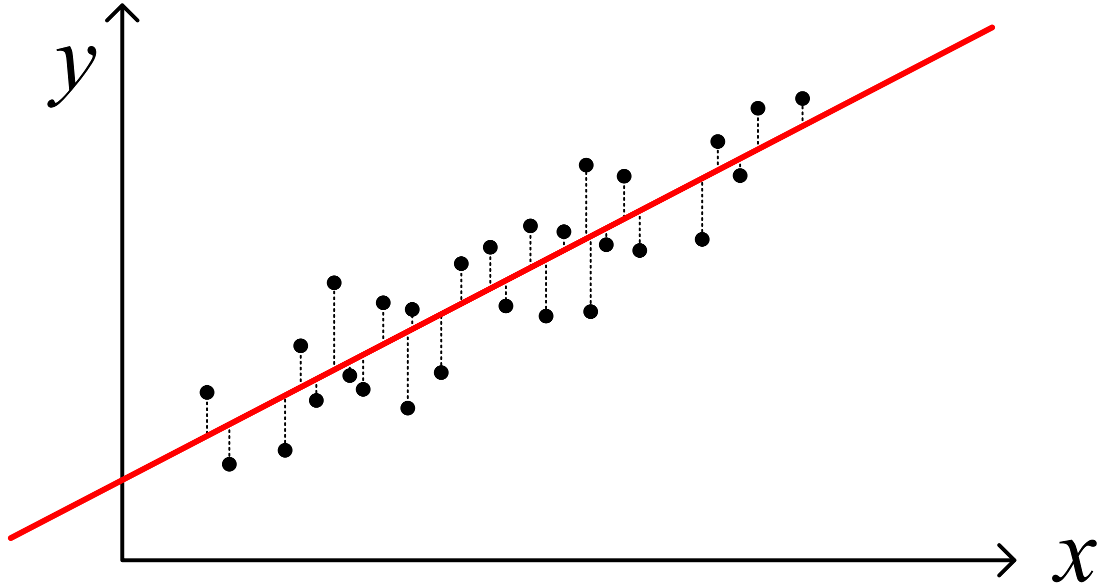
$$
\text{feature vector } \phi(x_i) = \begin{bmatrix} x_i \\ 1 \end{bmatrix}
$$

$$
\Phi = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_m & 1 \end{bmatrix} = \begin{bmatrix} \phi^T(x_1) \\ \phi^T(x_2) \\ \vdots \\ \phi^T(x_m) \end{bmatrix} \implies \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta
$$

Optimization

$$\min_{\theta_1,\theta_2} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \min_{\theta} \|\Phi\theta - y\|_2^2 \qquad (\text{same as } \min_{x} \|Ax - b\|_2^2)$$

$$\text{solution } \theta^* = (\Phi^T \Phi)^{-1} \Phi^T y$$



Optimization

- Note:

$$\begin{array}{ccccc} \text{input} & & \text{feature} & & \text{predicted output} \\ x_i & \rightarrow & \begin{bmatrix} x_i \\ 1 \end{bmatrix} & \rightarrow & \hat{y}_i \end{array}$$

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\begin{array}{cccc} \uparrow & \uparrow & \uparrow & \uparrow \\ \vec{a}_1 & \vec{a}_2 & \vec{x} & \vec{b} \end{array}$$

over-determined or projection

$$A(=\Phi) = [\vec{a}_1 \ \vec{a}_2]$$

Solving using Linear Algebra

- known as *least square*

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

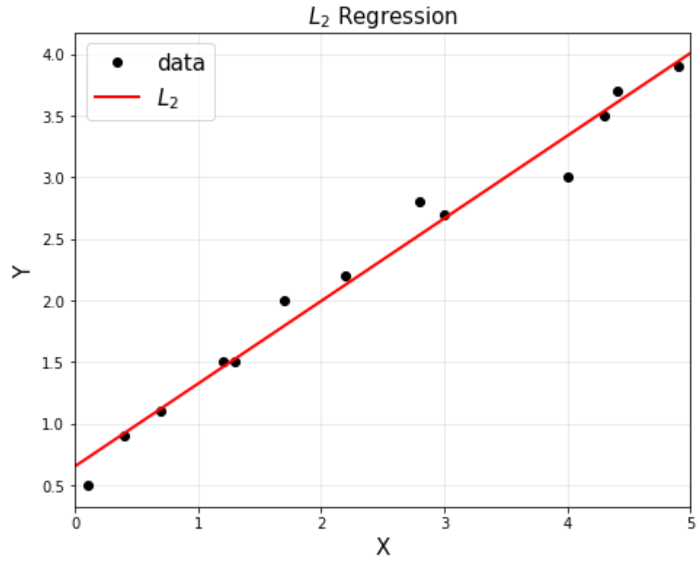
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# data points in column vector [input, output]
x = np.array([0.1, 0.4, 0.7, 1.2, 1.3, 1.7, 2.2, 2.8, 3.0, 4.0,
              4.3, 4.4, 4.9]).reshape(-1, 1)
y = np.array([0.5, 0.9, 1.1, 1.5, 1.5, 2.0, 2.2, 2.8, 2.7, 3.0,
              3.5, 3.7, 3.9]).reshape(-1, 1)

m = y.shape[0]
#A = np.hstack([x, np.ones([m, 1])])
A = np.hstack([x, x**0])
A = np.asmatrix(A)

theta = (A.T*A).I*A.T*y

print('theta:\n', theta)
```

```
theta:
 [[ 0.67129519]
 [ 0.65306531]]
```

Solving using Linear Algebra

```
# to plot
plt.figure(figsize=(10, 6))
plt.title('$L_2$ Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'ko', label="data")

# to plot a straight line (fitted line)
xp = np.arange(0, 5, 0.01).reshape(-1, 1)
yp = theta[0,0]*xp + theta[1,0]

plt.plot(xp, yp, 'r', linewidth=2, label="$L_2$")
plt.legend(fontsize=15)
plt.axis('scaled')
plt.grid(alpha=0.3)
plt.xlim([0, 5])
plt.show()
```



Solving using CVXPY Optimization

```
import cvxpy as cvx

theta2 = cvx.Variable(2, 1)
obj = cvx.Minimize(cvx.norm(A*theta2-y, 2))
cvx.Problem(obj,[]).solve()

print('theta:\n', theta2.value)
```

```
theta:
 [[ 0.67129519]
 [ 0.65306531]]
```

$$\boxed{\min_{\theta} \; \|\hat{y} - y\|_2 = \min_{\theta} \; \|A\theta - y\|_2}$$

Solving using CVXPY Optimization

```
import cvxpy as cvx

theta2 = cvx.Variable(2, 1)
obj = cvx.Minimize(cvx.norm(A*theta2-y, 2))
cvx.Problem(obj,[]).solve()

print('theta:\n', theta2.value)
```

```
theta:
 [[ 0.67129519]
 [ 0.65306531]]
```

$$\boxed{\min_{\theta} \; \lVert \hat{y} - y \rVert_2 = \min_{\theta} \; \lVert A\theta - y \rVert_2}$$

- By the way, do we have to use only L_2 norm? No.
  - Let's use L_1 norm

```
theta1 = cvx.Variable(2, 1)
obj = cvx.Minimize(cvx.norm(A*theta1-y, 1))
cvx.Problem(obj).solve()

print('theta:\n', theta1.value)
```

```
theta:
 [[ 0.68531634]
 [ 0.62587346]]
```

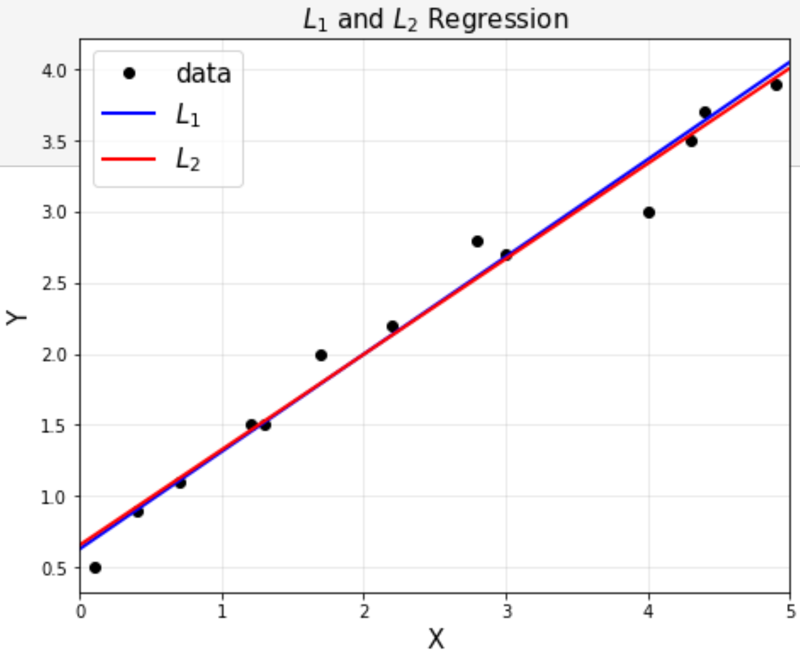
$$\boxed{\min_{\theta} \; \lVert \hat{y} - y \rVert_1 = \min_{\theta} \; \lVert A\theta - y \rVert_1}$$

L_2 Norm vs. L_1 Norm

```
# to plot data
plt.figure(figsize=(10, 6))
plt.title('$L_1$ and $L_2$ Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'ko', label='data')

# to plot straight lines (fitted lines)
xp = np.arange(0, 5, 0.01).reshape(-1, 1)
yp1 = theta1.value[0,0]*xp + theta1.value[1,0]
yp2 = theta2.value[0,0]*xp + theta2.value[1,0]

plt.plot(xp, yp1, 'b', linewidth=2, label='$L_1$')
plt.plot(xp, yp2, 'r', linewidth=2, label='$L_2$')
plt.legend(fontsize=15)
plt.axis('scaled')
plt.xlim([0, 5])
plt.grid(alpha=0.3)
plt.show()
```

L_2 Norm vs. L_1 Norm

```
# to plot data
plt.figure(figsize=(10, 6))
plt.title('$L_1$ and $L_2$ Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'ko', label='data')

# to plot straight lines (fitted lines)
xp = np.arange(0, 5, 0.01).reshape(-1, 1)
yp1 = theta1.value[0,0]*xp + theta1.value[1,0]
yp2 = theta2.value[0,0]*xp + theta2.value[1,0]

plt.plot(xp, yp1, 'b', linewidth=2, label='$L_1$')
plt.plot(xp, yp2, 'r', linewidth=2, label='$L_2$')
plt.legend(fontsize=15)
plt.axis('scaled')
plt.xlim([0, 5])
plt.grid(alpha=0.3)
plt.show()
```



L_1 norm also provides a decent linear approximation.

L_1 Norm with Outliers

- L_1 norm also provides a decent linear approximation.

- **What if outliers exist?**
  - Fitting with the different norms
  - source:
    - Week 9 of Computational Methods for Data Analysis by Coursera of Univ. of Washington
    - Chapter 17, online book available

L_1 Norm with Outliers

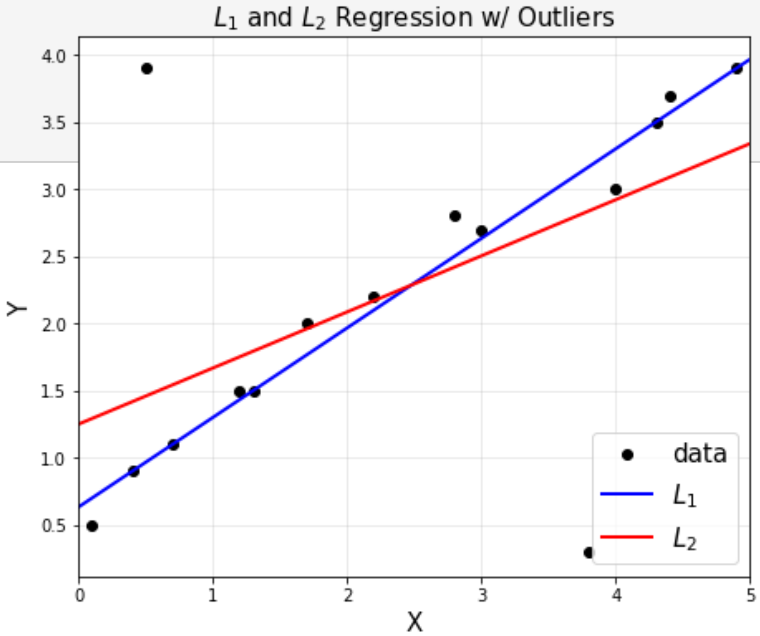
```
x = np.array([0.1, 0.4, 0.7, 1.2, 1.3, 1.7, 2.2, 2.8, 3.0, 4.0,
              4.3, 4.4, 4.9]).reshape(-1, 1)
y = np.array([0.5, 0.9, 1.1, 1.5, 1.5, 2.0, 2.2, 2.8, 2.7, 3.0,
              3.5, 3.7, 3.9]).reshape(-1, 1)

# add outliers
x = np.vstack([x, np.array([0.5, 3.8]).reshape(-1, 1)])
y = np.vstack([y, np.array([3.9, 0.3]).reshape(-1, 1)])

A = np.hstack([x, x**0])
A = np.asmatrix(A)

theta1 = cvx.Variable(2, 1)
obj1 = cvx.Minimize(cvx.norm(A*theta1-y, 1))
cvx.Problem(obj1).solve()

theta2 = cvx.Variable(2, 1)
obj2 = cvx.Minimize(cvx.norm(A*theta2-y, 2))
prob2 = cvx.Problem(obj2).solve()
```

$$\boxed{\min_{\theta} \; \|A\theta - y\|_1}$$

$$\boxed{\min_{\theta} \; \|A\theta - y\|_2}$$

Think About What Makes Different

```
# to plot data
plt.figure(figsize=(10, 6))
plt.plot(x, y, 'ko', label='data')
plt.title('$L_1$ and $L_2$ Regression w/ Outliers', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)

# to plot straight lines (fitted lines)
xp = np.arange(0, 5, 0.01).reshape(-1,1)
yp1 = theta1.value[0,0]*xp + theta1.value[1,0]
yp2 = theta2.value[0,0]*xp + theta2.value[1,0]

plt.plot(xp, yp1, 'b', linewidth=2, label='$L_1$')
plt.plot(xp, yp2, 'r', linewidth=2, label='$L_2$')
plt.axis('scaled')
plt.xlim([0, 5])
plt.legend(fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```



It is important to understand what makes them different

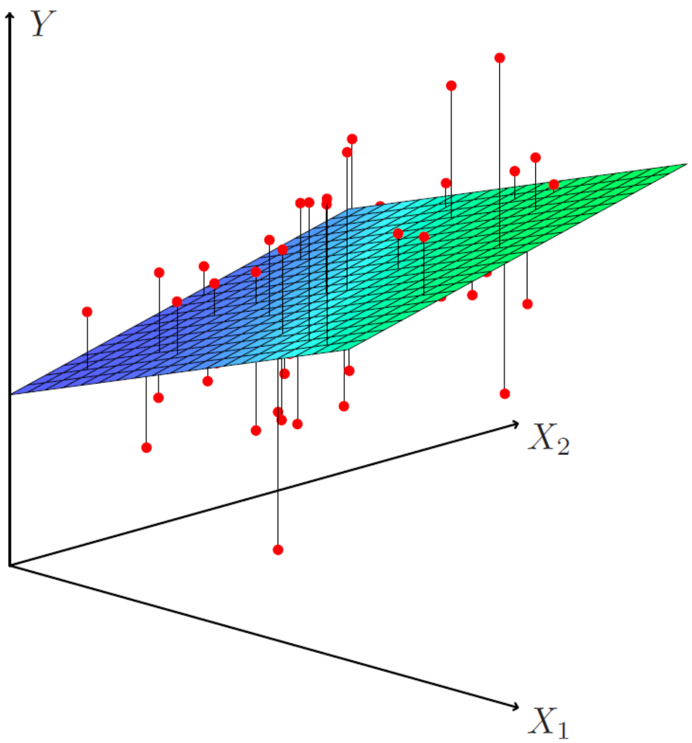
Multivariate Linear Regression

- Linear regression for multivariate data

$$\hat{y} = \theta_1 x_1 + \theta_2 x_2 + \theta_3$$

$$\phi\left(x^{(i)}\right) = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ 1 \end{bmatrix}$$

$$\Phi = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & 1 \\ x_1^{(2)} & x_2^{(2)} & 1 \\ \vdots & & \\ x_1^{(m)} & x_2^{(m)} & 1 \end{bmatrix} \implies \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = \Phi\theta$$



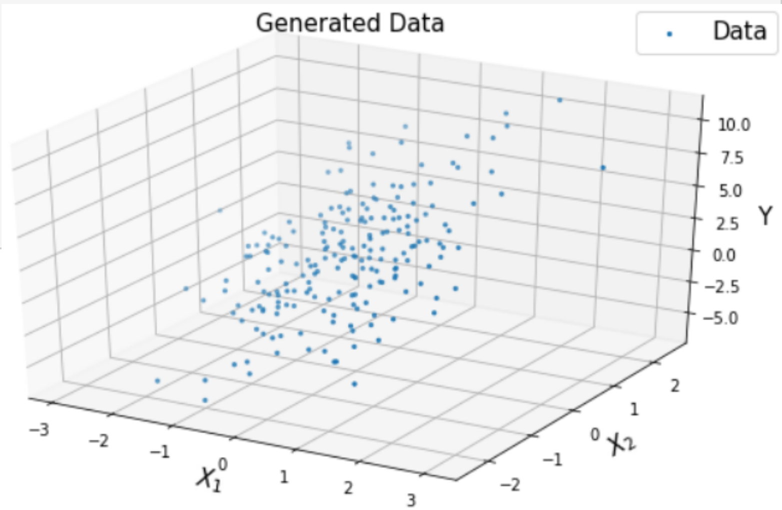
$$\implies \theta^* = (\Phi^T\Phi)^{-1}\Phi^T y$$

- Same in matrix representation

Multivariate Linear Regression

```
# for 3D plot
from mpl_toolkits.mplot3d import Axes3D
```

```
# y = theta1*x1 + theta2*x2 + theta3 + noise

n = 200
x1 = np.random.randn(n, 1)
x2 = np.random.randn(n, 1)
noise = 0.5*np.random.randn(n, 1);

y = 1*x1 + 3*x2 + 2 + noise

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1, 1, 1, projection='3d')
ax.set_title('Generated Data', fontsize=15)
ax.set_xlabel('$X_1$', fontsize=15)
ax.set_ylabel('$X_2$', fontsize=15)
ax.set_zlabel('Y', fontsize=15)
ax.scatter(x1, x2, y, marker='.', label='Data')
#ax.view_init(30,30)
plt.legend(fontsize=15)
plt.show()
```



Multivariate Linear Regression

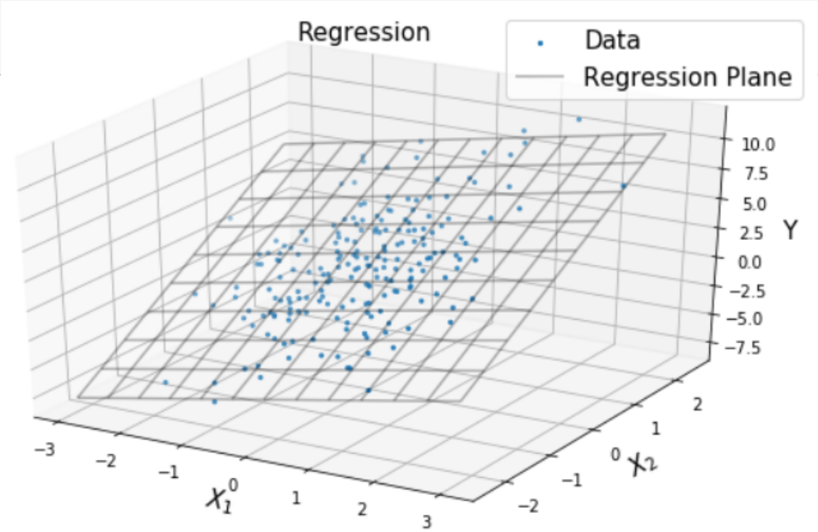
$$\theta = (A^T A)^{-1} A^T y$$

```
#% matplotlib qt5

A = np.hstack([x1, x2, np.ones((n, 1))])
A = np.asmatrix(A)
theta = (A.T*A).I*A.T*y

X1, X2 = np.meshgrid(np.arange(np.min(x1), np.max(x1), 0.5),
                     np.arange(np.min(x2), np.max(x2), 0.5))
YP = theta[0,0]*X1 + theta[1,0]*X2 + theta[2,0]

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1, 1, 1, projection='3d')
ax.set_title('Regression', fontsize=15)
ax.set_xlabel('$X_1$', fontsize=15)
ax.set_ylabel('$X_2$', fontsize=15)
ax.set_zlabel('Y', fontsize=15)
ax.scatter(x1, x2, y, marker='.', label='Data')
ax.plot_wireframe(X1, X2, YP, color='k', alpha=0.3, label='Regression Plane')
#ax.view_init(30,30)
plt.legend(fontsize=15)
plt.show()
```

Multivariate Linear Regression

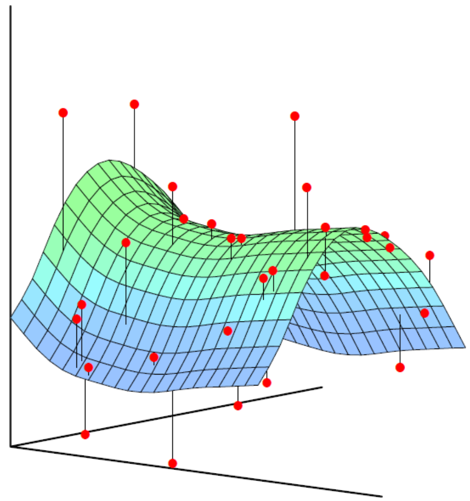
```
#% matplotlib qt5

A = np.hstack([x1, x2, np.ones((n, 1))])
A = np.asmatrix(A)
theta = (A.T*A).I*A.T*y

X1, X2 = np.meshgrid(np.arange(np.min(x1), np.max(x1), 0.5),
                     np.arange(np.min(x2), np.max(x2), 0.5))
YP = theta[0,0]*X1 + theta[1,0]*X2 + theta[2,0]

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1, 1, 1, projection='3d')
ax.set_title('Regression', fontsize=15)
ax.set_xlabel('$X_1$', fontsize=15)
ax.set_ylabel('$X_2$', fontsize=15)
ax.set_zlabel('Y', fontsize=15)
ax.scatter(x1, x2, y, marker='.', label='Data')
ax.plot_wireframe(X1, X2, YP, color='k', alpha=0.3, label='Regression Plane')
#ax.view_init(30,30)
plt.legend(fontsize=15)
plt.show()
```

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



Nonlinear Regression

- Linear regression for non-linear data



- Same as linear regression, just with non-linear features

- Method 1: constructing explicit feature vectors
  - polynomial features
  - Radial basis function (RBF) features

- Method 2: implicit feature vectors, kernel trick (*optional*)

Nonlinear Regression

- Polynomial (here, quad is used as an example)

$$y = \theta_1 + \theta_2 x + \theta_3 x^2 + \text{noise}$$

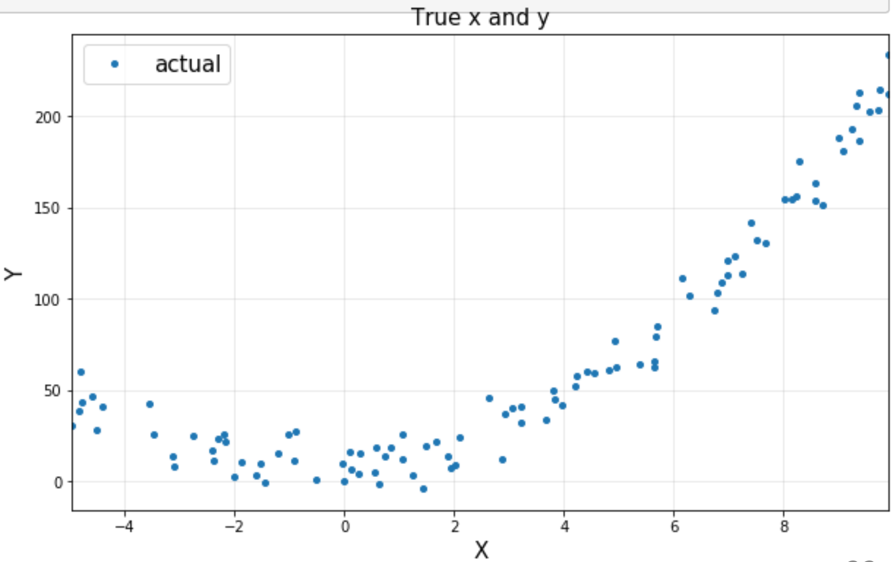
$$\phi(x_i) = A = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix}$$

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_m & x_m^2 \end{bmatrix} \quad \Longrightarrow \quad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta$$

$$\Longrightarrow \quad \theta^* = (\Phi^T \Phi)^{-1} \Phi^T y$$

Polynomial Regression

```
# y = theta1 + theta2*x + theta3*x^2 + noise

n = 100
x = -5 + 15*np.random.rand(n, 1)
noise = 10*np.random.randn(n, 1)

y = 10 + 1*x + 2*x**2 + noise

plt.figure(figsize=(10, 6))
plt.title('True x and y', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'o', markersize=4, label='actual')
plt.xlim([np.min(x), np.max(x)])
plt.grid(alpha=0.3)
plt.legend(fontsize=15)
plt.show()
```



Polynomial Regression

```
A = np.hstack([x**0, x, x**2])
A = np.asmatrix(A)

theta = (A.T*A).I*A.T*y
print('theta:\n', theta)
```

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

```
theta:
 [[ 10.08455652]
 [  1.28294638]
 [  1.96288127]]
```

```
xp = np.linspace(np.min(x), np.max(x))
yp = theta[0,0] + theta[1,0]*xp + theta[2,0]*xp**2

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o', markersize=4, label='actual')
plt.plot(xp, yp, 'r', linewidth=2, label='estimated')

plt.title('Nonlinear regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.xlim([np.min(x), np.max(x)])
plt.grid(alpha=0.3)
plt.legend(fontsize=15)
plt.show()
```

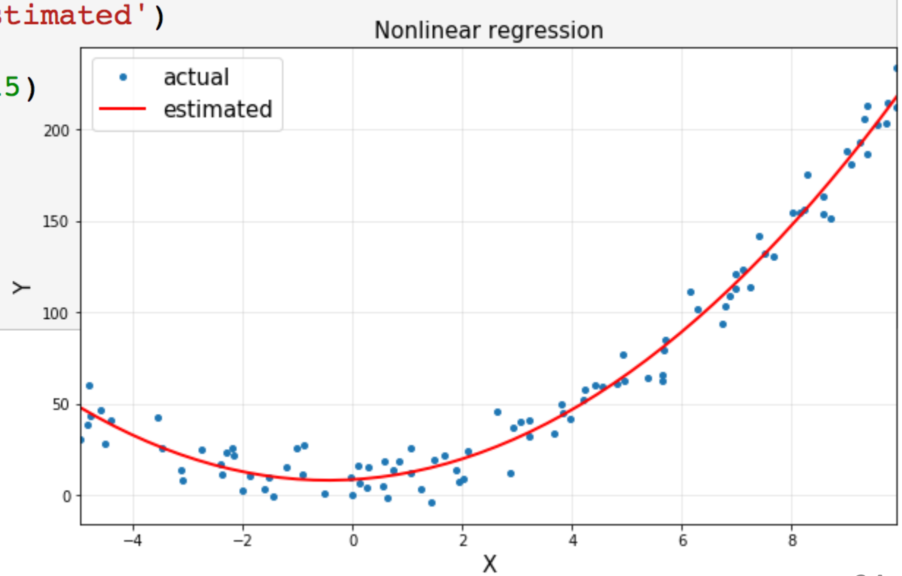
Polynomial Regression

```
A = np.hstack([x**0, x, x**2])
A = np.asmatrix(A)

theta = (A.T*A).I*A.T*y
print('theta:\n', theta)
```

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

```
theta:
 [[ 10.08455652]
 [  1.28294638]
 [  1.96288127]]
```

```
xp = np.linspace(np.min(x), np.max(x))
yp = theta[0,0] + theta[1,0]*xp + theta[2,0]*xp**2

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o', markersize=4, label='actual')
plt.plot(xp, yp, 'r', linewidth=2, label='estimated')

plt.title('Nonlinear regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.xlim([np.min(x), np.max(x)])
plt.grid(alpha=0.3)
plt.legend(fontsize=15)
plt.show()
```



Summary: Linear Regression

- Though linear regression may seem limited, it is very powerful, since the input features can themselves include non-linear features of data

- Linear regression on non-linear features of data

- For least-squares loss, optimal parameters still

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