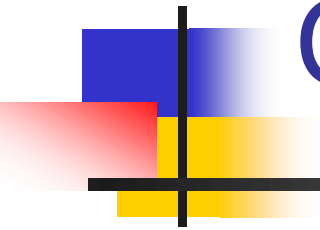
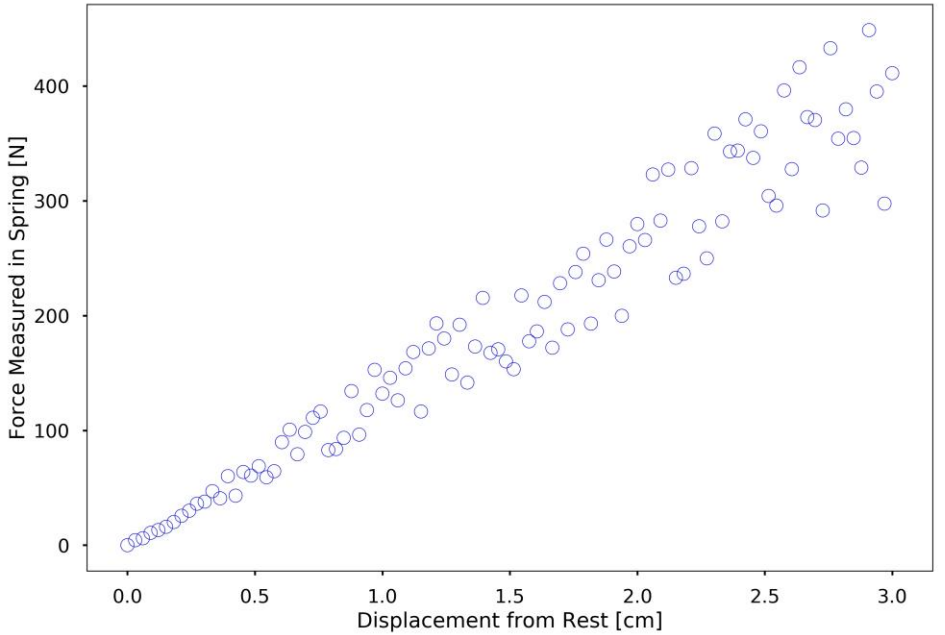


Computer Methods (MAE 3403)

Least square regression

Motivation

- Given parameters of a system, determine its state.
  - F = kx
- However, parameters of the system (e.g., k) may not be known a priori.
- We have data points, e.g., (x_i, F_i), (most likely noisy) of the system state from the past to determine the system parameters.



How to estimate model parameters (e.g., stiffness) given a set of data?

Formulation

- Given a set of independent data points x_i and dependent data points y_i, i=1,…,m, we would like to find an **estimation function**, $\hat{y}(x)$ that describes the data as well as possible.
- In least squares regression,

$$\hat{y}(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \cdots + \alpha_n f_n(x)$$

  - α_i: parameters, $f_i(x)$: basis function
- For $F = kx, \alpha_1 = k, f_1(x) = x$.

Least squares regression

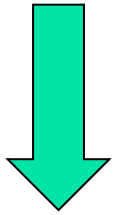
- Find/Estimate the parameters that minimize the total *squared* error $E = (\hat{y}(x_1) - y_1)^2 + (\hat{y}(x_2) - y_2)^2 + \cdots + (\hat{y}(x_m) - y_m)^2$

- Individual error $\mathrm{e_i} = (\hat{y}(x_i) - y_i)$.

- $e = [e_1, e_2, \ldots, e_m]$. Then $E = ||e||_2^2$ (L_2 norm)

How to find the parameters

$$\hat{y}(x_1) = \alpha_1 f_1(x_1) + \alpha_2 f_2(x_1) + \cdots + \alpha_n f_n(x_1),$$
$$\hat{y}(x_2) = \alpha_1 f_1(x_2) + \alpha_2 f_2(x_2) + \cdots + \alpha_n f_n(x_2),$$
$$\cdots$$
$$\hat{y}(x_m) = \alpha_1 f_1(x_m) + \alpha_2 f_2(x_m) + \cdots + \alpha_n f_n(x_m).$$



$$\hat{Y} = A\beta$$

How?

Least squares regression solution

- Estimate β: β is n-dimensional, A is a m by n matrix
- If m = n, we just take the inverse of A to get

$$\beta = A^{-1}\hat{Y}$$

- What if m > n? More data than the number of unknowns

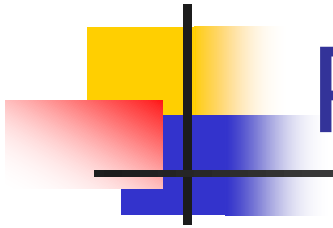
$$\hat{Y} = A\beta \Rightarrow A^T\hat{Y} = \underbrace{(A^TA)}_{n\times n}\beta$$

$$\boxed{\beta = (A^TA)^{-1}A^T\hat{Y}}$$

Properties

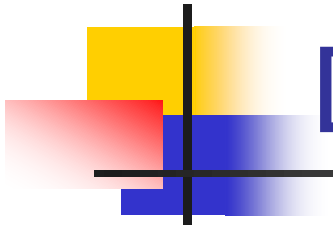
$$\beta = (A^T A)^{-1} A^T \hat{Y}$$

- Minimizes the squared error E
- One can solve it also via optimization and calculus
- Pseudo-inverse of A (when m > n): $(A^T A)^{-1} A^T$
  - Exists when m > n and A is full column rank
  - $A^T A$ is invertible except in some pathological cases



Python implementation

- Direct inverse
- Pseudo inverse
- numpy.linalg.lstsq
- optimize.curve_fit



Direct inverse

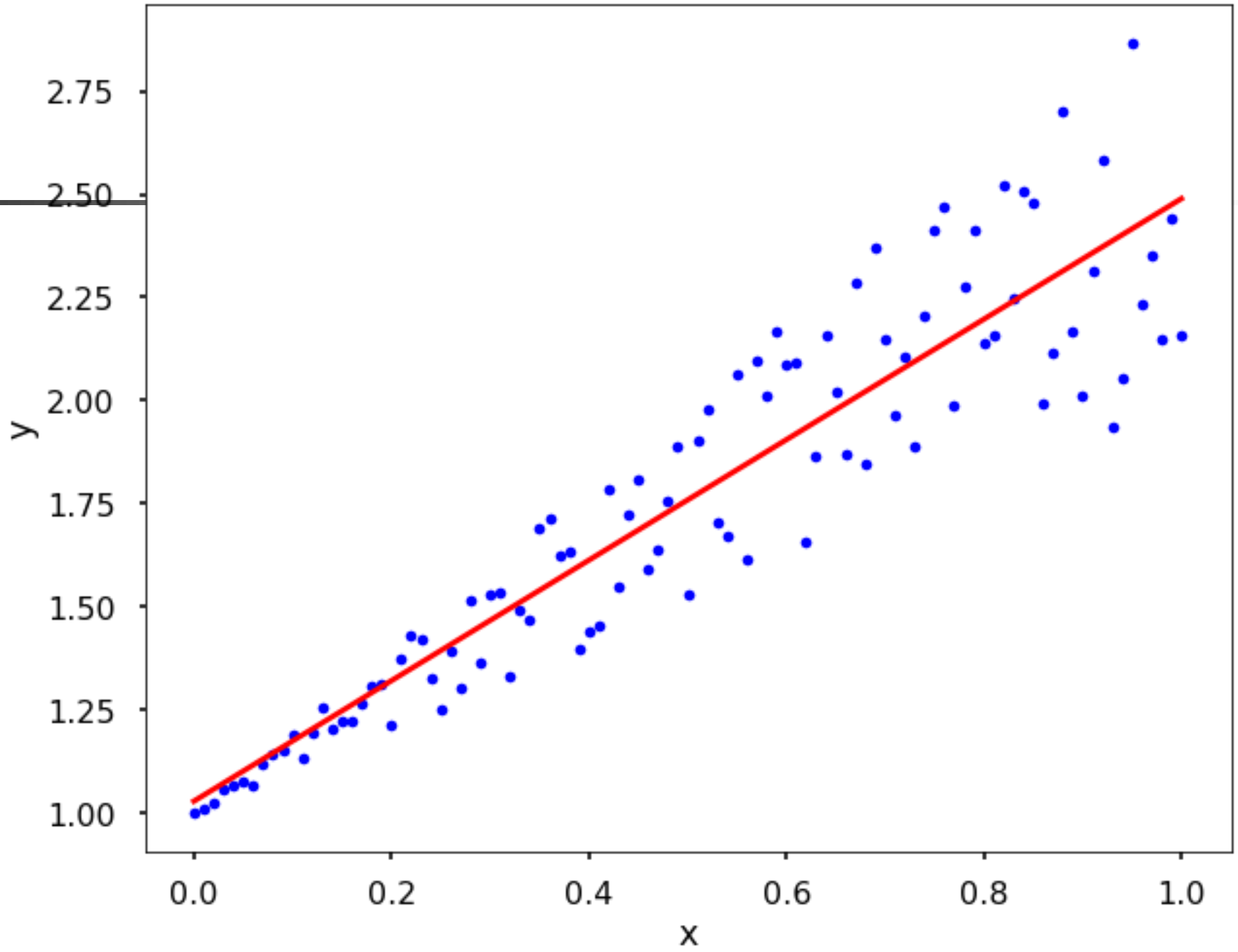
```
import numpy as np
# Formulate the A and y
...
# Direct least square regression
alpha =np.dot((np.dot(np.linalg.inv(np.dot(A.T,A)),A.T)),y)
print(alpha)
```

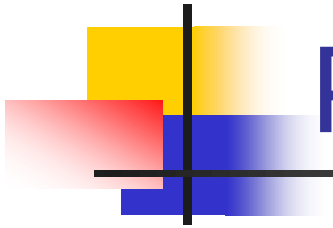
Example

```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8')
# generate x and y
x = np.linspace(0, 1, 101)
y = 1 + x + x *np.random.random(len(x))
# assemble matrix A
A = np.vstack([x, np.ones(len(x))]).T
# turn y into a column vector
y = y[:, np.newaxis] # len(x)X1
```

- Then use the code from previous page
- Plotting

```
plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.')
plt.plot(x, alpha[0]*x +alpha[1], 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```


2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
y
0.0
0.2
0.4
0.6
0.8
1.0
x

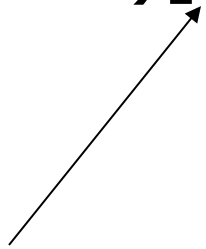


Pseudo-inverse

```
pinv = np.linalg.pinv(A)
alpha = pinv.dot(y)
print(alpha)
```

numpy.linalg.lstsq

```
alpha = np.linalg.lstsq(A, y, rcond=None)[0]
print(alpha)
```



return more things than the solution itself

Use Least squares regression for nonlinear problems

- Some nonlinear problems can be converted to linear forms
  - By taking logarithms
- Use curve_fit from scipy to directly estimate parameters

Examples

- Exponential functions (illustrate in Python)

$$\hat{y}(x) = \alpha e^{\beta x} \Rightarrow \underbrace{\log \hat{y}(x)}_{\tilde{y}(x)} = \underbrace{\log \alpha}_{\tilde{\alpha}} + \beta x$$
$$\tilde{y}(x) = \tilde{\alpha} + \beta x$$

- Power functions

$$\hat{y}(x) = bx^m \Rightarrow \underbrace{\log \hat{y}(x)}_{\tilde{y}(x)} = \underbrace{\log b}_{\tilde{\alpha}} + m \log(x)$$

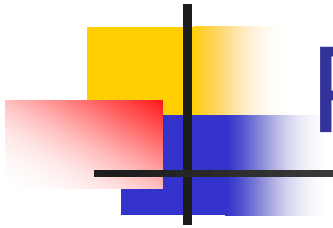
Polynomial regression

- We can also use polynomial and least squares.

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x^1 + a_0$$

  - a_i: coefficients, n: order of the polynomial

- Given data (x_i, y_i), we can use polynomials of different orders to fit it.
  - The coefficients can be estimated via least squares regression.

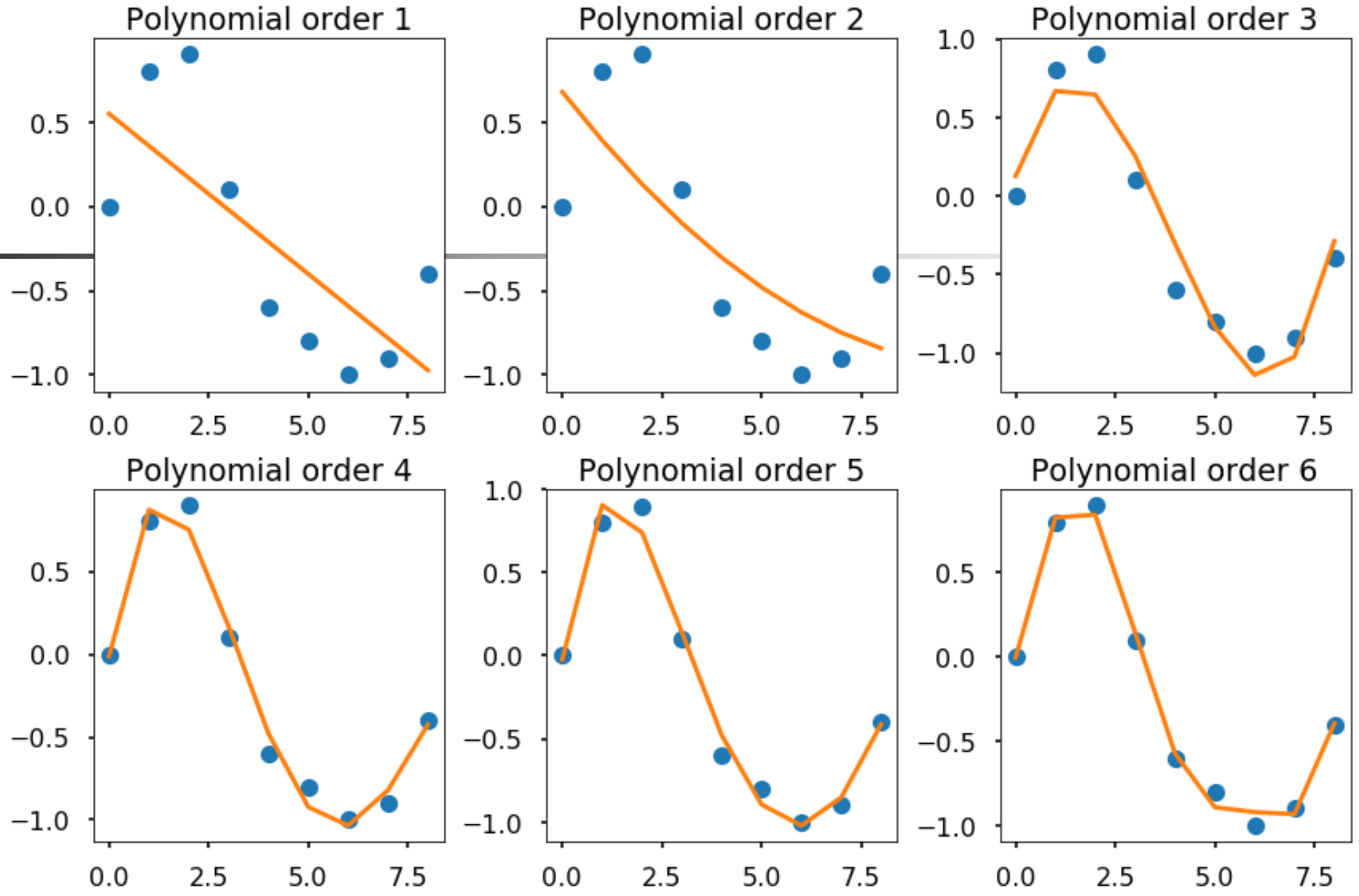


Python implementation

- numpy.polyfit: obtain coefficients of different order polynomials with least squares
- numpy.polyval: evaluate a polynomial with given coefficients

Example

```
x_d = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
y_d = np.array([0, 0.8, 0.9, 0.1, -0.6, -0.8, -1, -0.9, -0.4])
plt.figure(figsize = (12, 8))
for i in range(1, 7):
    # get the polynomial coefficients
    y_est = np.polyfit(x_d, y_d,i)
    plt.subplot(2,3,i)
    plt.plot(x_d, y_d, 'o')
    # evaluate the values for a polynomial
    plt.plot(x_d, np.polyval(y_est, x_d))
    plt.title(f'Polynomial order {i}')
    plt.tight_layout() plt.show()
```



- Higher order, better fitting, but could be overfitting

Curve fit from Scipy

- Use function optimize.curve_fit from scipy
  - Don't forget **from scipy import** optimize

```
# generate x and y
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
```

```
def func(x, a, b):
    y = a*x + b
    return y
```

```
alpha = optimize.curve_fit(func, xdata = x, ydata = y)[0]
```

- Can also be used for nonlinear functions

optimize.curve_fit from scipy

- Use optimize.curve_fit to fit any form function and estimate the parameters in it.
- Example: given data points (x,y), fit an exponential function without the logarithm trick

```
# let's define the function form
def func(x, a, b):
      y = a*np.exp(b*x)
       return y
alpha, beta = optimize.curve_fit(func, xdata = x, ydata = y)[0]
print(f'alpha={alpha}, beta={beta}')
```