BASIC TECHNIQUES FOR COMPUTER SIMULATIONS WPO – 4TH SESSION

INTERPOLATION AND EXTRAPOLATION

INTRODUCTION

- You are given the following data set for the values of (an unknown) function f(x) at points x_i:

x	0	20	50	100
f(x)	1	4	6	7

- What is the value of the function f at x = 30 ?

  → INTERPOLATION : estimate the desired value of f(x) that lies inside the range of the known base points $x_1, x_2, \dots, x_n$

- What is the value of the function f at x = 150 ?

  → EXTRAPOLATION : estimate the desired value of f(x) that lies outside the range of the known base points $x_1, x_2, \dots, x_n$

We can express function f in the form of a polynomial
and then use this polynomial to determine the desired values.

POLYNOMIAL INTERPOLATION

- Method used to create the unique (n-1)th order polynomial that fits n data points.

$$f(x) = P_{n-1}(x) = a_0\, x^0 + a_1 x^1 + a_2 x^2 + \ \dots + a_{n-1} x^{n-1}$$

x	0	20	50	100
f(x)	1	4	6	7

Step 1: Determine the order of the polynomial:

$$\boxed{f(x) = P_3(x) = a_0\, x^0 + a_1 x^1 + a_2 x^2 + a_3 x^3}$$

Step 2: The polynomial passes through each one of the n points of the dataset:

$$a_0 1 + a_1 0^1 + a_2 0^2 + a_3 0^3 = 1$$

$$a_0 1 + a_1 20^1 + a_2 20^2 + a_3 20^3 = 4$$

$$a_0 1 + a_1 50^1 + a_2 50^2 + a_3 50^3 = 6$$

$$a_0 1 + a_1 100^1 + a_2 100^2 + a_3 100^3 = 7$$

System of 4 linear algebraic equations with 4 unknowns: a_0, a_1, a_2, a_3

POLYNOMIAL INTERPOLATION

Step 3: Solve the system for the unknows a :

$$[x] = \begin{bmatrix} 1 & 0^1 & 0^2 & 0^3 \\ 1 & 20^1 & 20^2 & 20^3 \\ 1 & 50^1 & 50^2 & 50^3 \\ 1 & 100^1 & 100^2 & 100^3 \end{bmatrix} \quad [a] = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad [y] = \begin{bmatrix} 1 \\ 4 \\ 6 \\ 7 \end{bmatrix}$$

xa=y

(in the form of **Ax**=**b**)

Solve for **a**

PYTHON

def main(): Define matrix **x** and vectors **a** and **y**

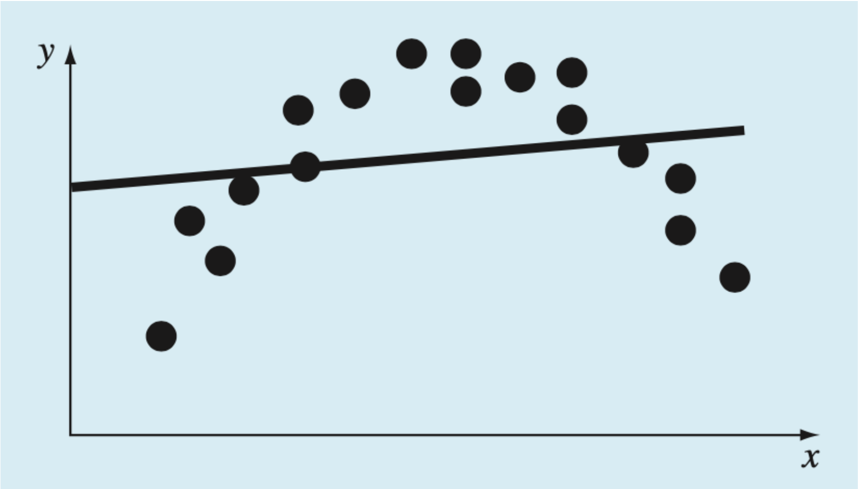
def polynomial_inter(..): 1) Solve for **a**

2) Create the polynomial $P_3(x)$ (can be done with numpy.poly1d) using the obtained elements of **a**

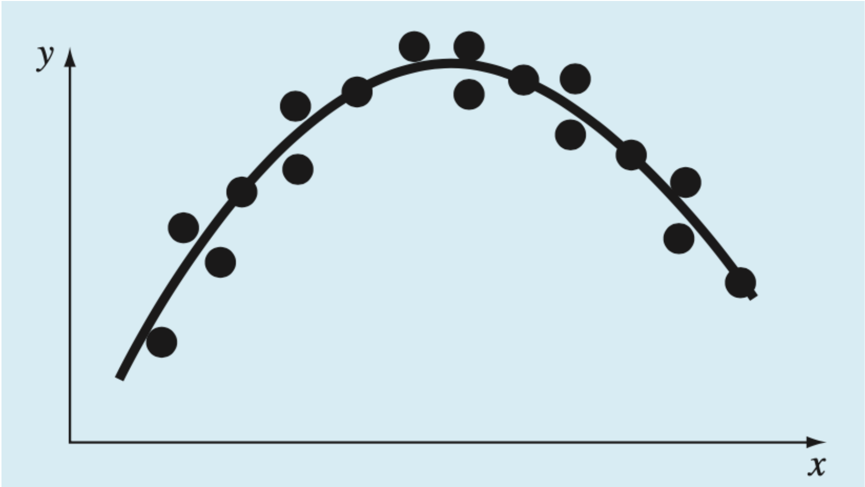
LEASE-SQUARES FIT

- Finding the best-fitting curve to a given set of points,
  by minimizing the sum of the square of the offsets of the points from the curve.

Linear



Nonlinear



source: Chapra SC. Applied numerical methods with MATLAB for engineers and scientistic, 2008.

PYTHON : **numpy.polyfit** to find the polynomial's coefficients
numpy.poly1d to create the polynomial

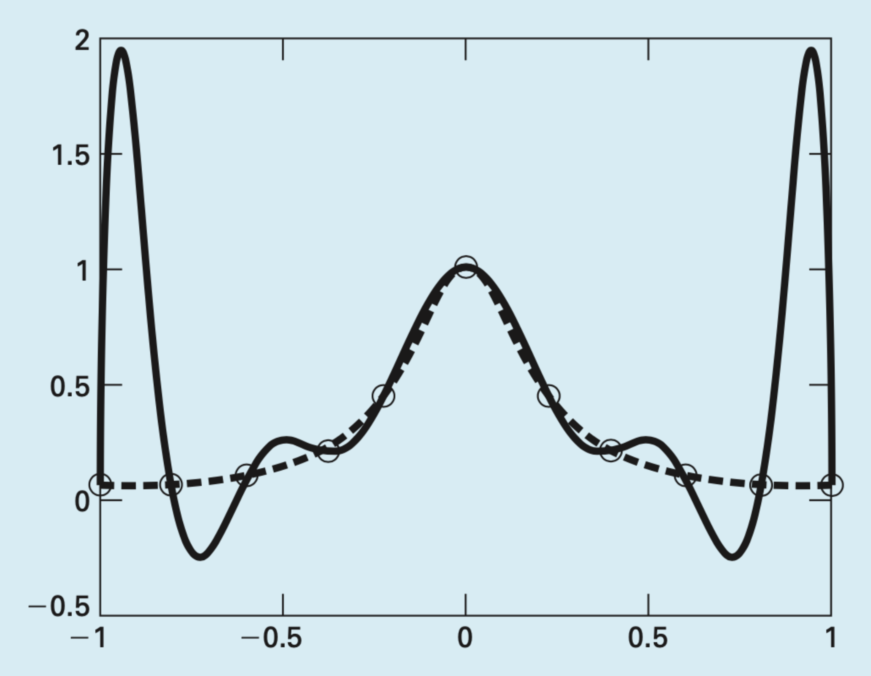
! Order of regression (degree of polynomial) has to be defined, when calling polyfit

DANGERS OF HIGHER ORDER REGRESSION

- Runge's function: $f(x) = \frac{1}{1+25x^2}$
- 11 equally spaced points in the range [-1, 1]: x = -1, -0.8, -0.6, …, 1

→ 10^{th} order polynomial $P_{10}(x) = a_0 + a_1x^1 + a_2x^2 + \dots + a_{10}x^{10}$

dashed line: f(x)
line: $P_{10}(x)$



bad interpolation

source: Chapra SC. Applied numerical methods with MATLAB for engineers and scientistic, 2008.

PIECEWISE INTERPOLATION : CUBIC SPLINES

- Given a set of n data ”base” points $(x_i, f(x_i))$,
  cubic spline $S(x)$ derives a 3rd order polynomial $Ci(x)$ for each interval between points $x_0, x_1, \ldots, x_n$

$$C_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$$

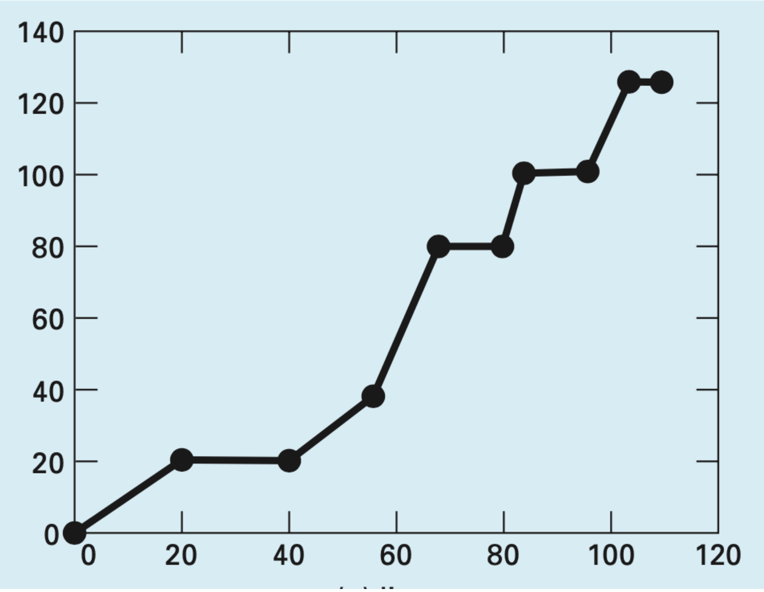
- $$S(x) = \begin{cases} C_1(x) & , x_0 \leq x \leq x_1 \\ \ldots & \\ C_i(x) & , x_{i-1} \leq x \leq x_1 \\ \ldots & \\ C_n(x) & , x_{n-1} \leq x \leq x_n \end{cases}$$

- In order to determine $S(x)$, we need to determine a_i, b_i, c_i, d_i for every polynomial $C_i(x)$
  (4n unknowns in total)

OTHER PIECEWISE INTERPOLATION METHODS

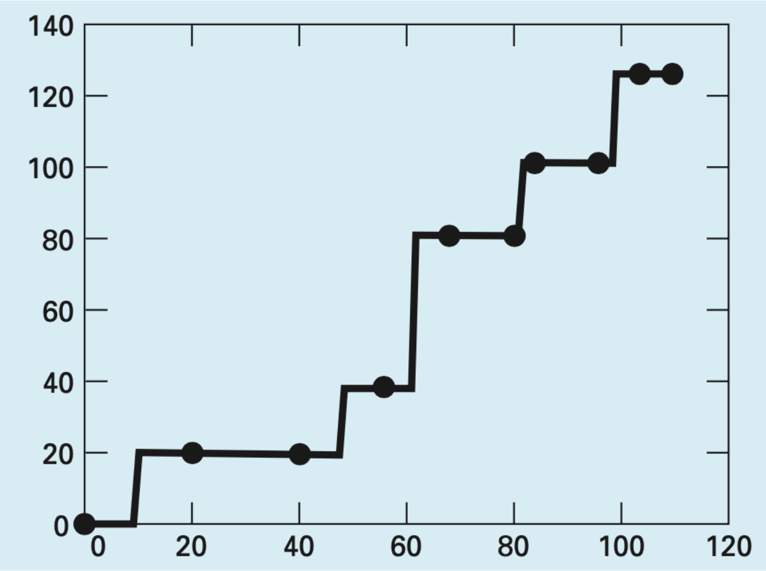
- Linear: Straight lines (first-order polynomials) connect points $x_1, x_2, \ldots, x_n$

$$C_i(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i)$$



source: Chapra SC. Applied numerical methods with MATLAB for engineers and scientistic, 2008.

- Nearest-neighbor: Sets the value of an interpolated point to the value of the nearest existing data point.
  The interpolation looks like a series of plateaus, which can be thought as zero-order polynomials.

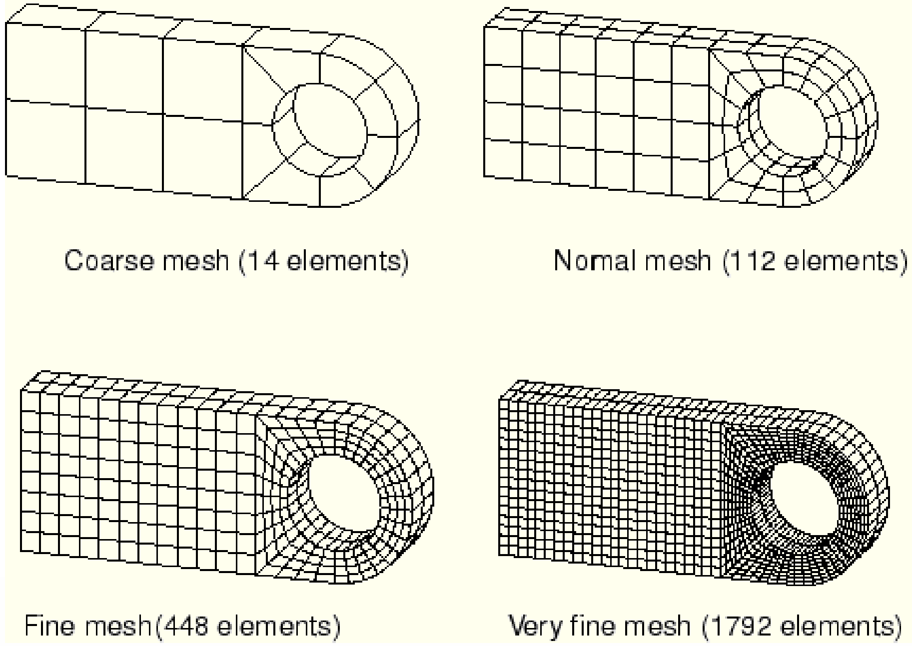


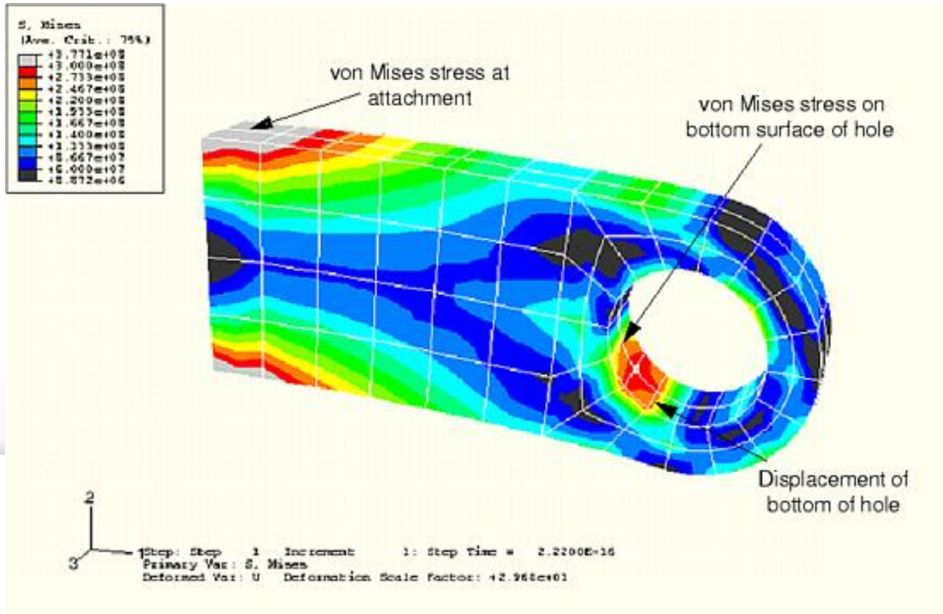
PYTHON : scipy.interpolate.**interp1d** (see theory .pdf p. 37)

kind = ‘linear’, ‘nearest’ or ‘cubic’ (cubic splines)

! For **extrapolation**: scipy.interpolate.interp1d(…, **fill_value=‘extrapolate’**) !

INTERPOLATION IN SIMULATIONS





source: https://classes.engineering.wustl.edu

<u>Meshing</u>: the computational domain is divided into discrete cells.

<u>Solution</u>: properties are calculated at discrete points, located at the corners or centres of the mesh cells.

<u>Post-processing</u>: Interpolation is used to estimate the properties (pressure, temperature, stress etc.) at points within these cells.