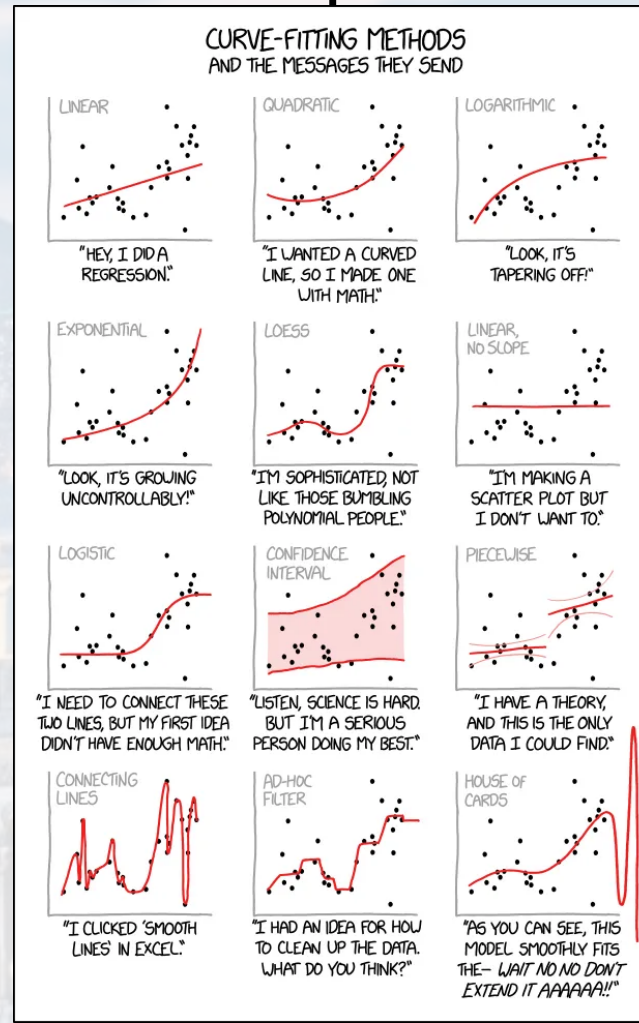


M. Hohle:

Physics 77: Introduction to Computational Techniques in Physics



syllabus:

- Introduction to Unix & Python (week 1 - 2)
- Functions, Loops, Lists and Arrays (week 3 - 4)
- Visualization (week 5)
- Parsing, Data Processing and File I/O (week 6)
- Statistics and Probability, Interpreting Measurements (week 7 - 8)
- Random Numbers, Simulation (week 9)
- Numerical Integration and Differentiation (week 10)
- **Root Finding, Interpolation (week 11)**
- Systems of Linear Equations (week 12)
- Ordinary Differential Equations (week 13)
- Fourier Transformation and Signal Processing (week 14)
- Capstone Project Presentations (week 15)

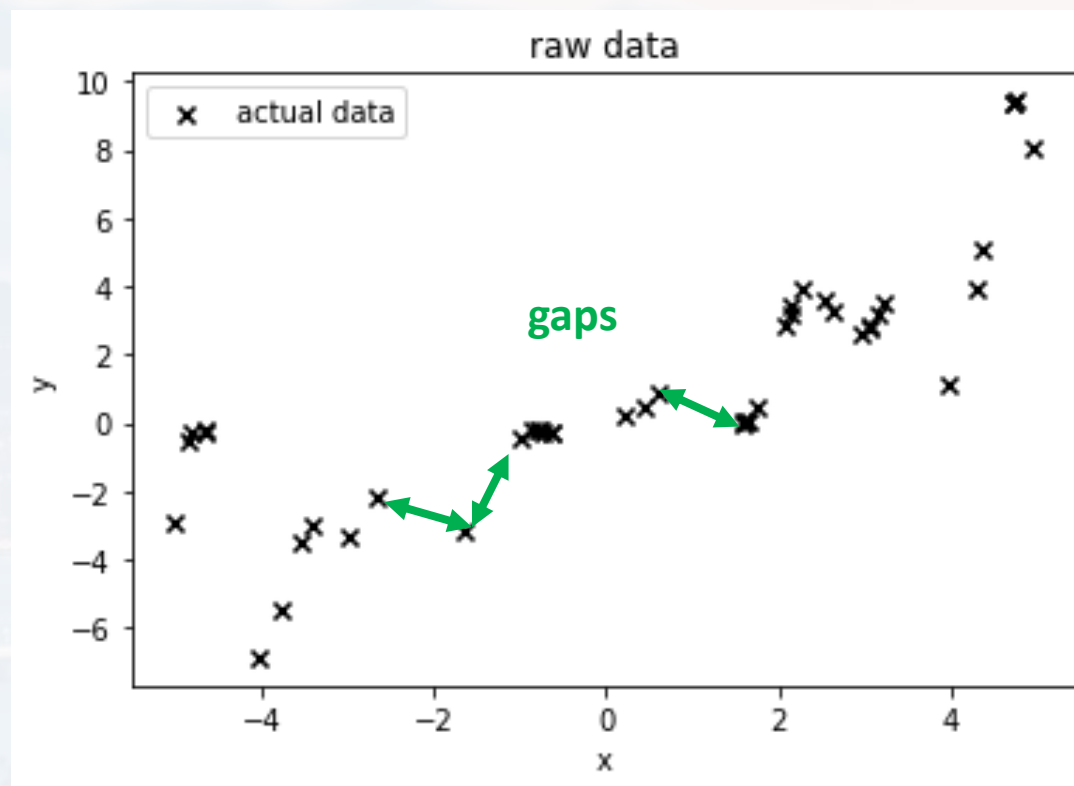


the problem:

Interpolation

Smoothing

Root Finding





the problem:

linear interpolation

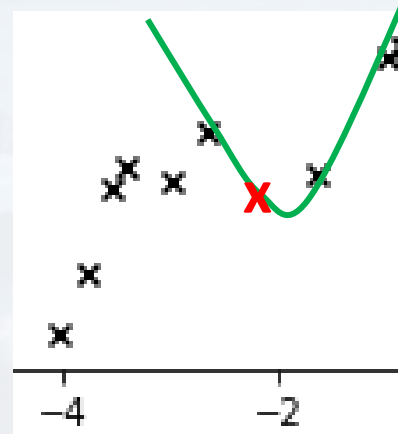
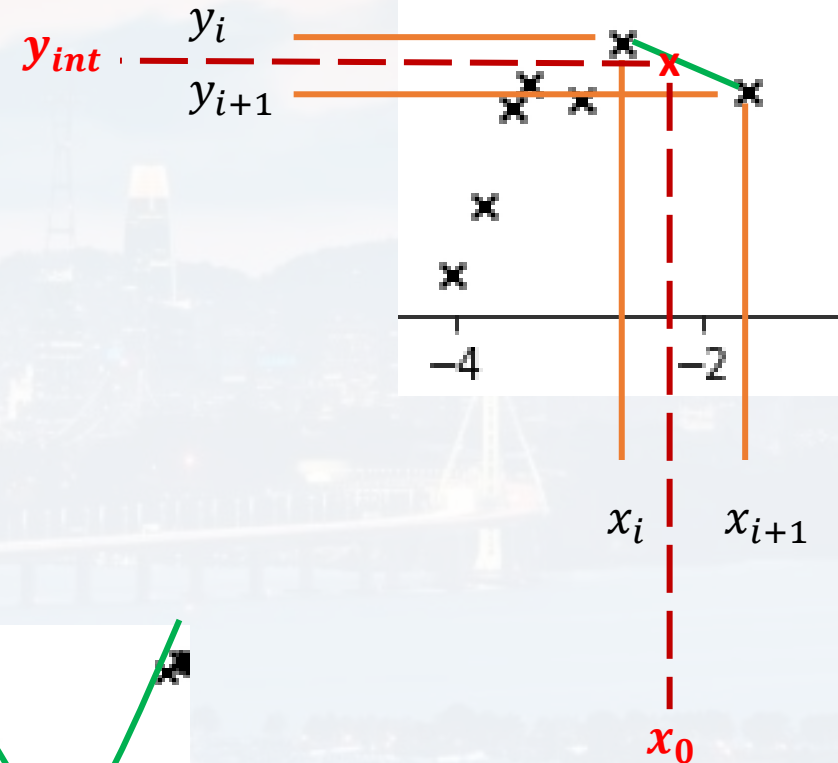
$$y_{int} = y_i + m (x_0 - x_i)$$

$$y_{int} = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i} (x_0 - x_i)$$

quadratic interpolation

$$y_{int} = y_i + m (x_0 - x_i) + a (x_0 - x_i)^2$$

this time we need **one more**
reference point for calculating ***a***



Interpolation

Smoothing

Root Finding



the problem:

quadratic interpolation

$$y_{int} = y_i + m(x_0 - x_i) + a(x_0 - x_i)^2$$

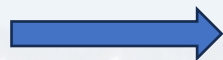
this time we need **one more**
reference point for calculating ***a***

all three reference points need to fit the same parabola

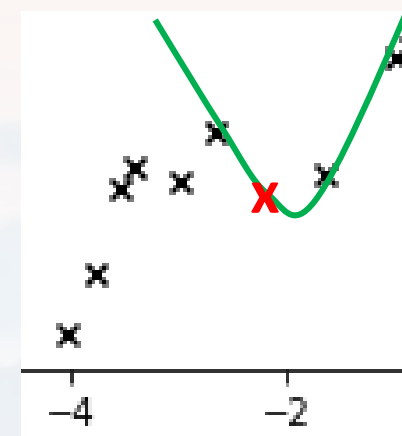
$$y_i = c + mx_i + ax_i^2$$

$$y_{i+1} = c + mx_{i+1} + ax_{i+1}^2$$

$$y_{i+2} = c + mx_{i+2} + ax_{i+2}^2$$



solving for ***c***, ***m*** and ***a***



Interpolation

Smoothing

Root Finding



check out

InterpolateExamples.py

Interpolation

Smoothing

Root Finding

```
from scipy import interpolate
```

```
I = interpolate.interp1d(x, y)
```

```
xint = np.arange(-4.99, 4.94, 0.1)
```

```
yint = I(xint)
```

```
plt.plot(xint, yint, c = 'r', linewidth = 3, alpha = 0.3,\n        label = 'interpolation')
```

```
plt.scatter(x, y, marker = 'x', c = 'k', label = 'actual data')
```

```
plt.xlabel('x')
```

```
plt.ylabel('y')
```

```
plt.legend()
```

```
plt.title('Linear interpolation')
```

```
plt.show()
```




check out

InterpolateExamples.py

Interpolation

Smoothing

Root Finding

```
from scipy import interpolate
```

```
I = interpolate.interp1d(x, y)
```

```
xint = np.arange(-4.99, 4.94, 0.1)
```

```
yint = I(xint)
```

```
plt.plot(xint, yint, c = 'r', linewidth =
```

```
plt.scatter(x, y, marker = 'x', c = 'k',
```

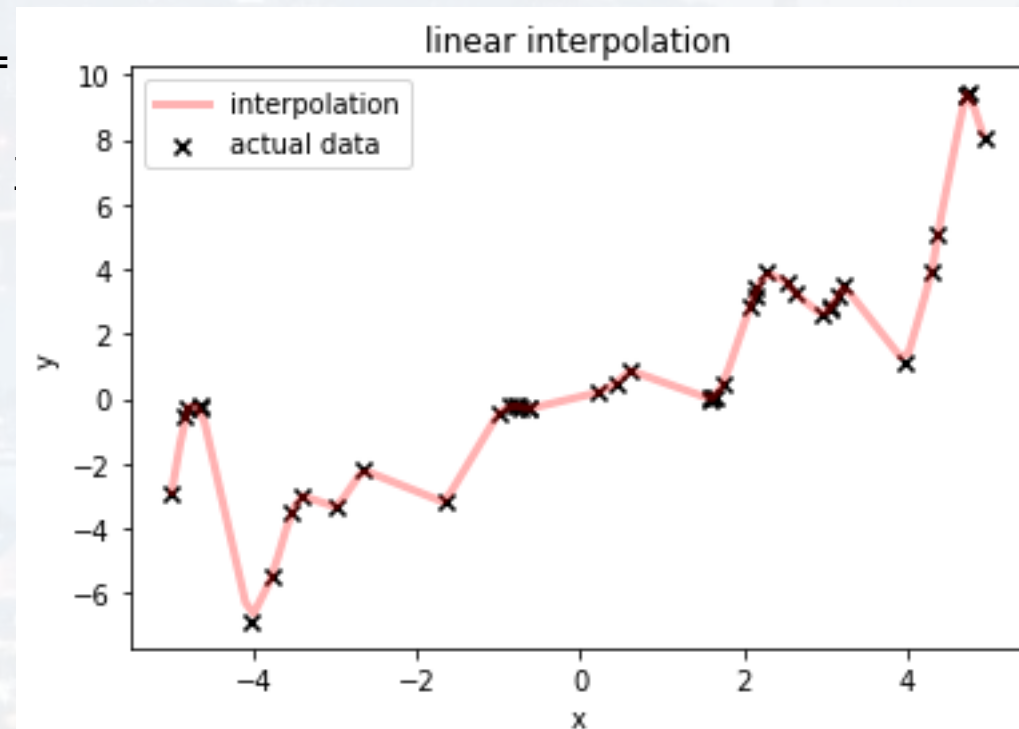
```
plt.xlabel('x')
```

```
plt.ylabel('y')
```

```
plt.legend()
```

```
plt.title('Linear interpolation')
```

```
plt.show()
```





check out

InterpolateExamples.py

Interpolation

Smoothing

Root Finding

```
from scipy import interpolate
```

```
I = interpolate.interp1d(x, y, kind = 2)
```

quadratic interpolation

```
xint = np.arange(-4.99, 4.94, 0.1)
```

```
yint = I(xint)
```

```
plt.plot(xint, yint, c = 'r', linewidth = 3, alpha = 0.3, \
        label = 'interpolation')
```

```
plt.scatter(x, y, marker = 'x', c = 'k', label = 'actual data')
```

```
plt.xlabel('x')
```

```
plt.ylabel('y')
```

```
plt.legend()
```

```
plt.title('Linear interpolation')
```

```
plt.show()
```




check out

InterpolateExamples.py

Interpolation

Smoothing

Root Finding

```
from scipy import interpolate
```

```
I = interpolate.interp1d(x, y, kind = 2)
```

quadratic interpolation

```
xint = np.arange(-4.99, 4.94, 0.1)
```

```
yint = I(xint)
```

```
plt.plot(xint, yint, c = 'r', linewidth =
```

```
plt.scatter(x, y, marker = 'x', c = 'k',
```

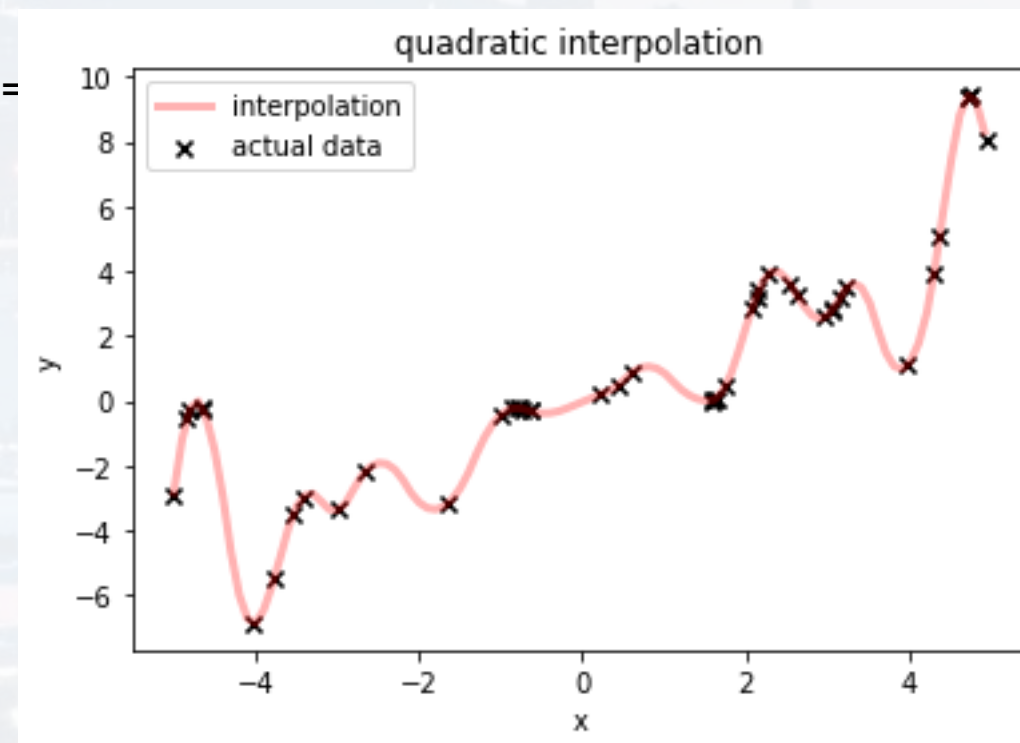
```
plt.xlabel('x')
```

```
plt.ylabel('y')
```

```
plt.legend()
```

```
plt.title('Linear interpolation')
```

```
plt.show()
```



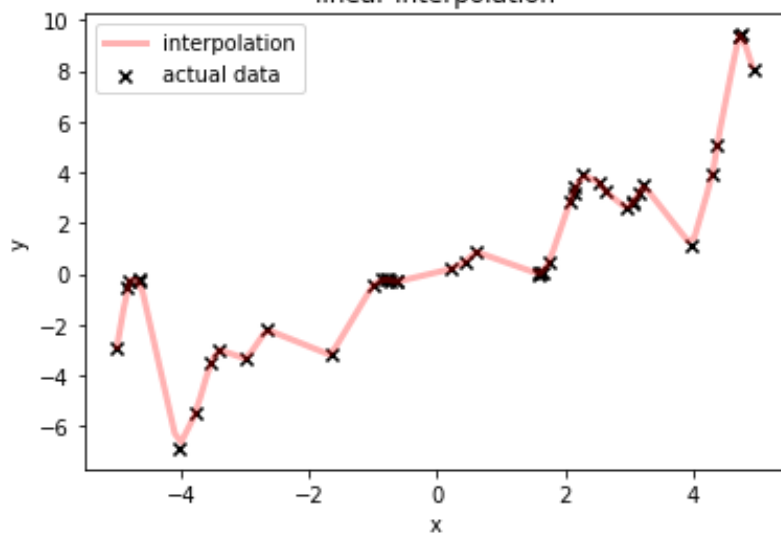


Interpolation

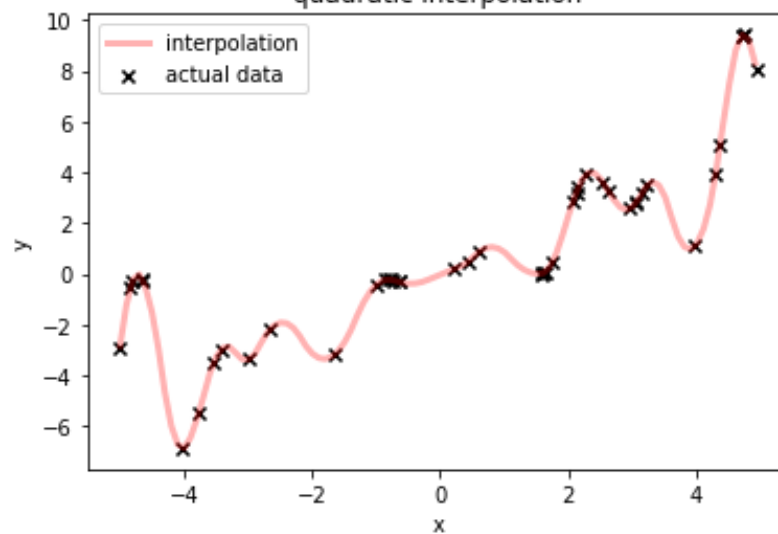
Smoothing

Root Finding

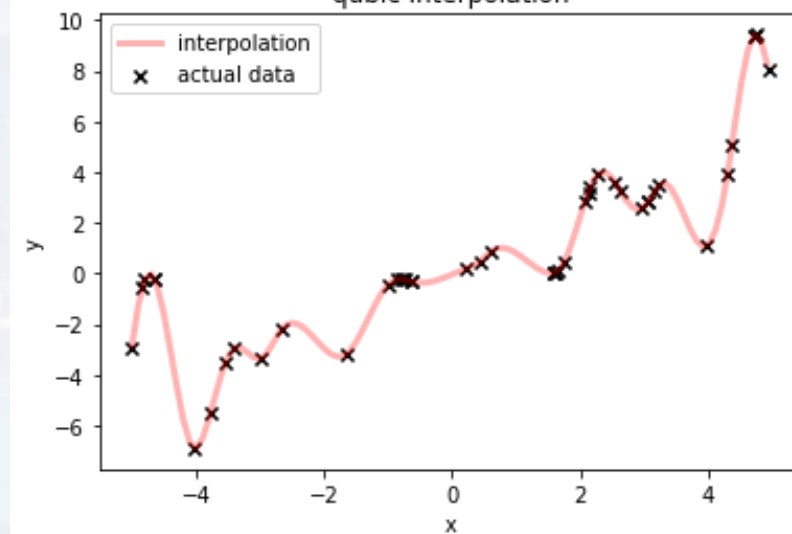
linear interpolation



quadratic interpolation

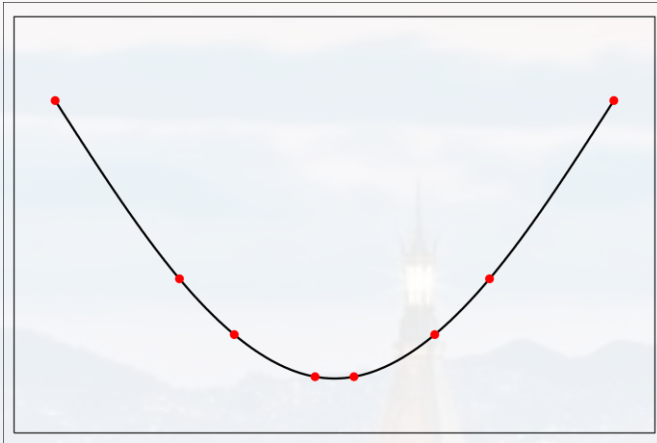


cubic interpolation





spline interpolation



A shape that minimizes the curvature κ under the constraint of passing through all reference points

$$\kappa = \frac{\frac{d^2y}{dx^2}}{\left(1 + \left[\frac{dy}{dx}\right]^2\right)^{3/2}}$$

By Stamcose - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=12802780>

Interpolation

Smoothing

Root Finding

x needs to be sorted in ascending order
* stands for unpacking zipped objects

```
sorted_pairs = sorted(zip(x, y))  
x_sorted, y_sorted = zip(*sorted_pairs)
```

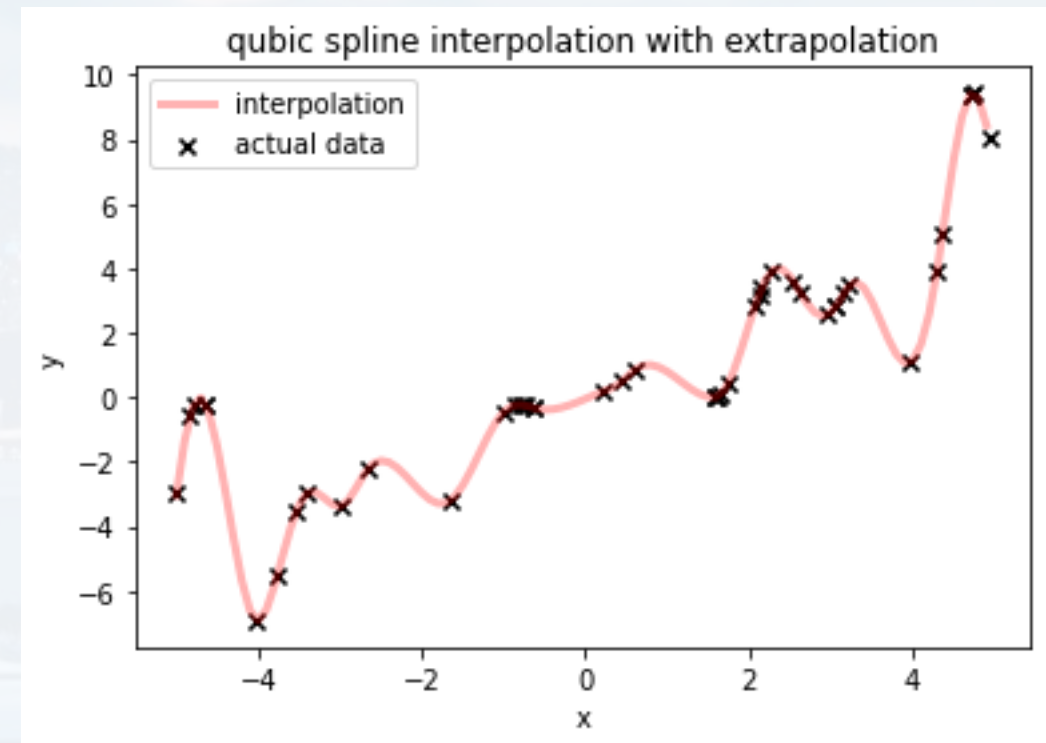
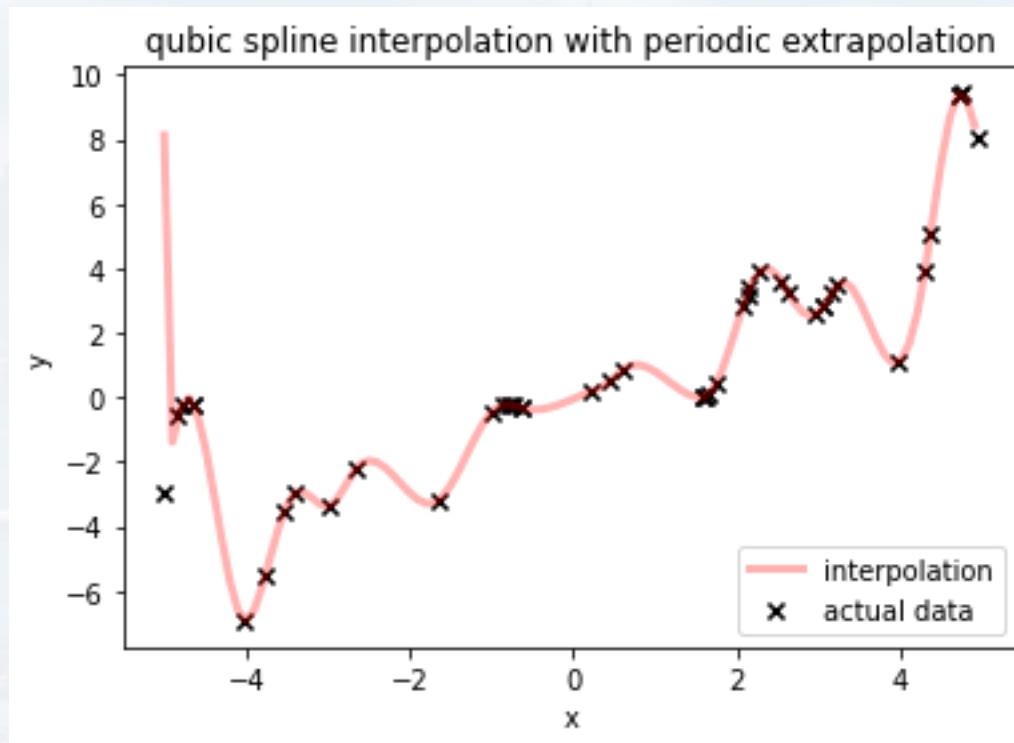
```
I = interpolate.CubicSpline(x_sorted, y_sorted,\n                             extrapolate = 'periodic')
```




Interpolation

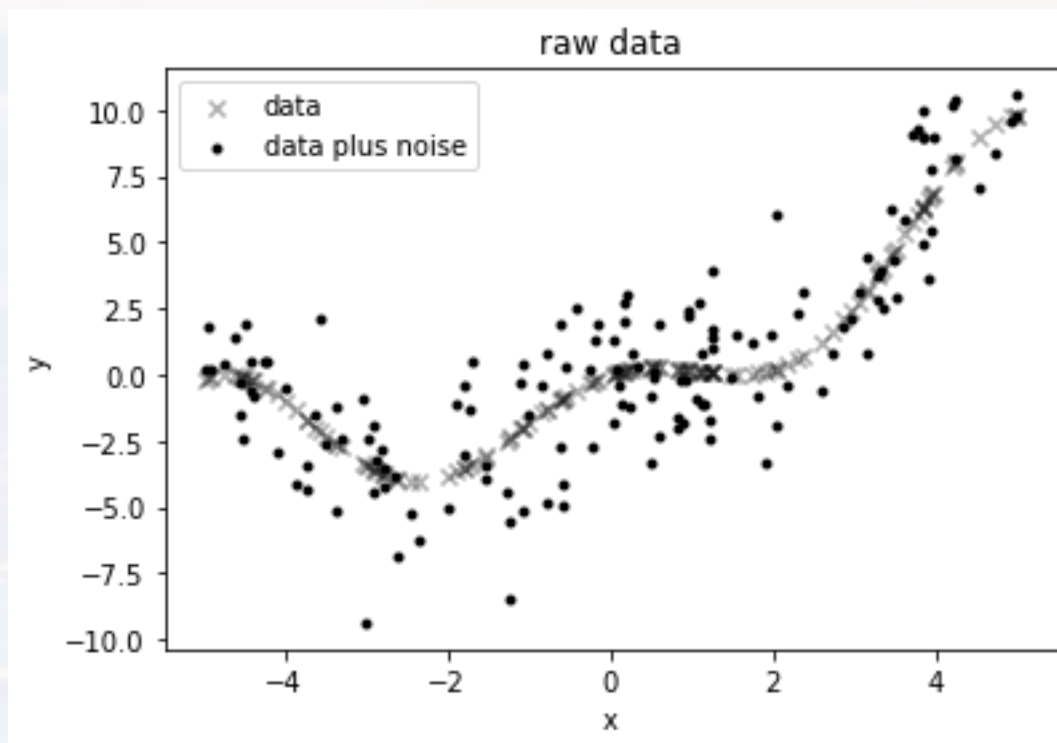
Smoothing

Root Finding





the problem:



when interpolating

→ you don't want to interpolate noise

many noise filter are **low pass** filter

Interpolation
Smoothing
Root Finding



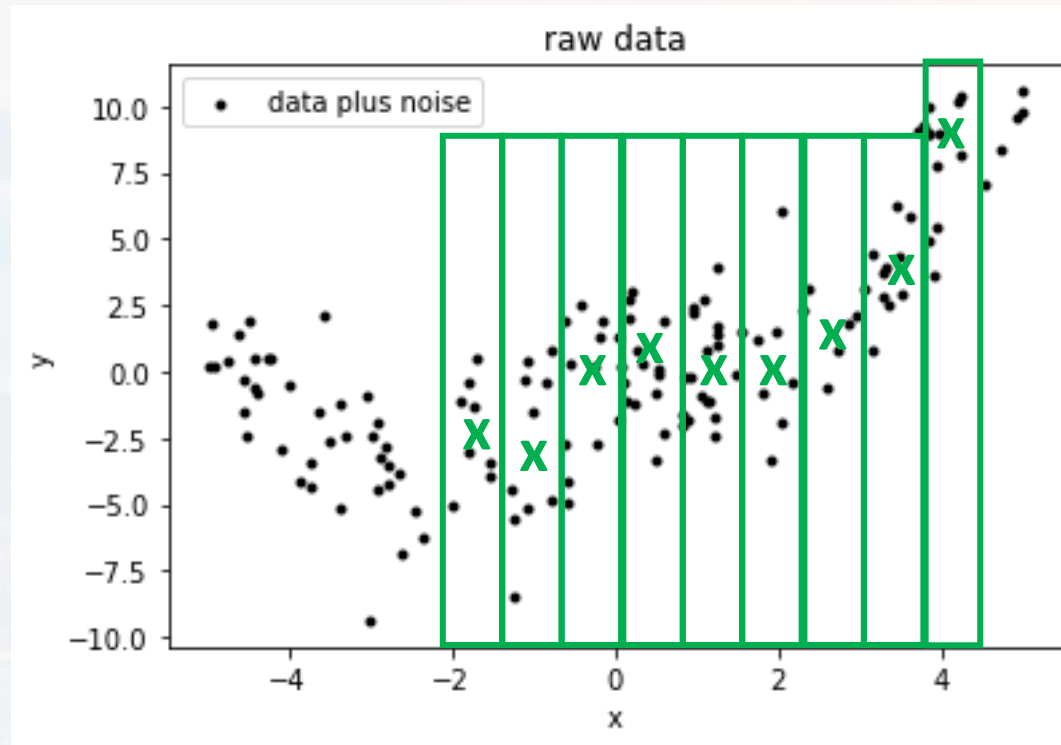
smoothing filter:

Algorithm	Overview and uses	Pros	Cons
Additive smoothing	used to smooth categorical data .		
Butterworth filter	Slower roll-off than a Chebyshev Type I/Type II filter or an elliptic filter	<ul style="list-style-type: none">• More linear phase response in the passband than Chebyshev Type I/ Type II and elliptic filters can achieve.• Designed to have a frequency response as flat as possible in the passband.	<ul style="list-style-type: none">• requires a higher order to implement a particular stopband specification
Chebyshev filter	Has a steeper roll-off and more passband ripple (type I) or stopband ripple (type II) than Butterworth filters .	<ul style="list-style-type: none">• Minimizes the error between the idealized and the actual filter characteristic over the range of the filter	<ul style="list-style-type: none">• Contains ripples in the passband.
Digital filter	Used on a sampled, discrete-time signal to reduce or enhance certain aspects of that signal		
Elliptic filter			
Exponential smoothing	<ul style="list-style-type: none">• Used to reduce irregularities (random fluctuations) in time series data, thus providing a clearer view of the true underlying behaviour of the series.• Also, provides an effective means of predicting future values of the time series (forecasting).^[3]		

Interpolation
Smoothing
Root Finding



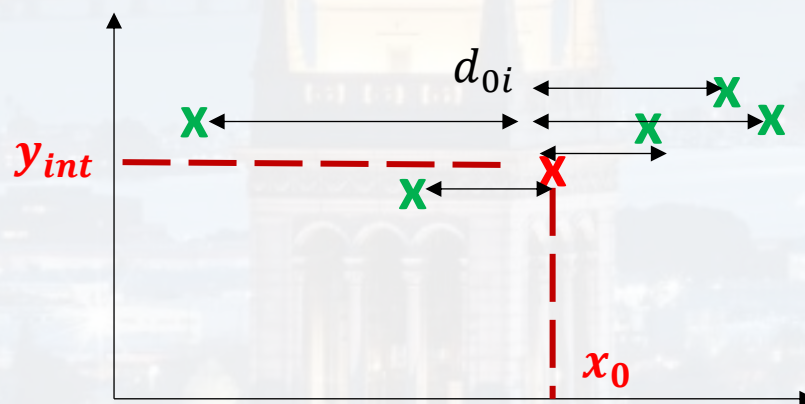
Interpolation
Smoothing
Root Finding



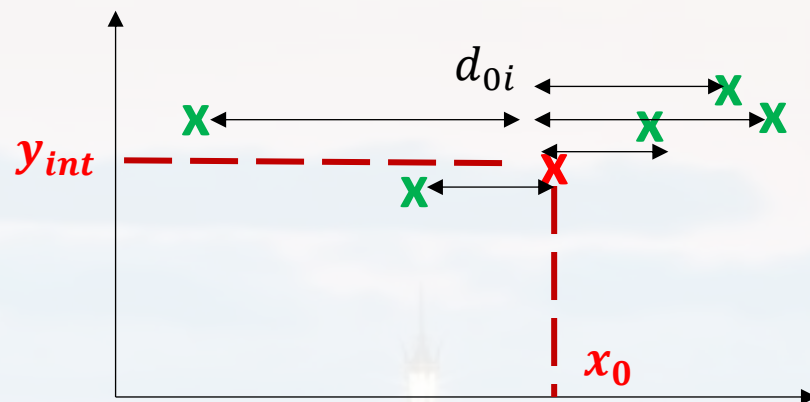
moving averages

better: → weighted average

→ data points **further away** from reference point have **lower weights w**



$$y_{int} \sim \sum_{i=1}^I w_i y_i \quad w_i \sim \frac{1}{d_{0i}}$$



data points **further away** from reference point have **lower weights** w

$$y_{int} \sim \sum_{i=1}^I w_i y_i$$

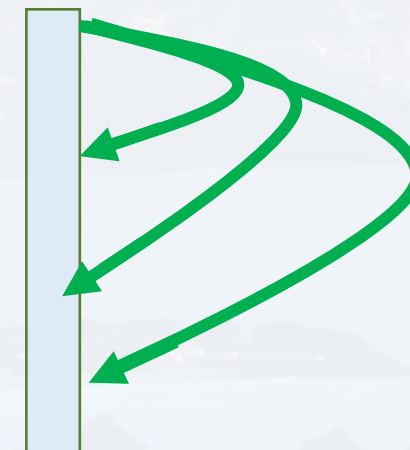
$$w_i \sim \frac{1}{d_{oi}}$$

Interpolation
Smoothing
Root Finding

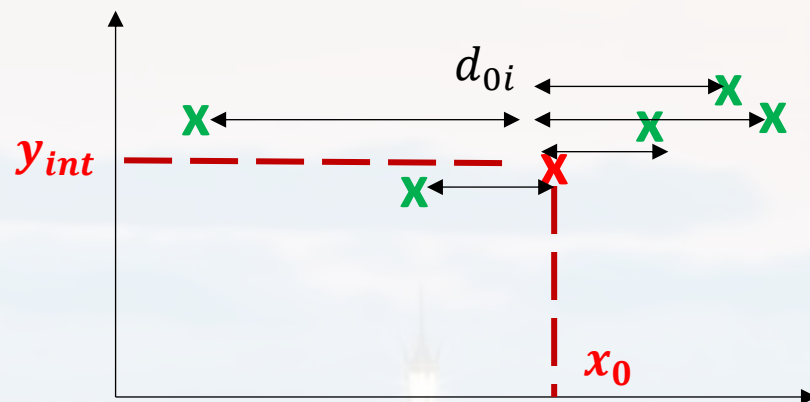
```
L = np.random.uniform(0,1,(100,1))
```

```
D = np.zeros((len(L),len(L)))
```

```
for ii, i in enumerate(L):  
    for jj, j in enumerate(L):  
        D[ii,jj] = i - j
```



But that is very inefficient!



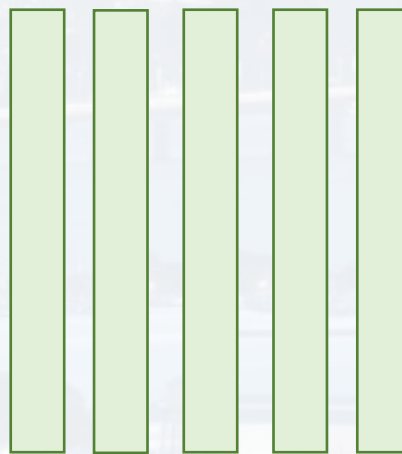
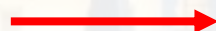
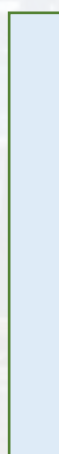
data points **further away** from
reference point have **lower weights** w

$$y_{int} \sim \sum_{i=1}^I w_i y_i$$

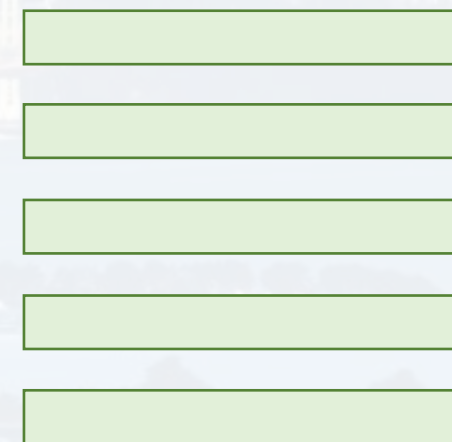
$$w_i \sim \frac{1}{d_{0i}}$$

Interpolation
Smoothing
Root Finding

better:



-





Interpolation
Smoothing
Root Finding



```
L = np.random.uniform(0,1,(100,1))
```

```
D = np.tile(L, (1, len(L))) - np.tile(L.transpose(), (len(L), 1))
```

check out:

SmoothGaussKernel.py
SmoothExamples.py



```
import numpy as np
```

```
def SmoothGaussKernel(x, xint, y, sigma):
```

```
    Dx = np.tile(x.transpose(), (len(xint), 1))
```

```
    Dxint = np.tile(xint.transpose(), (len(x), 1))
```

```
    D = Dx.transpose() - Dxint
```

```
    W = np.exp(-(D**2)/(sigma))
```

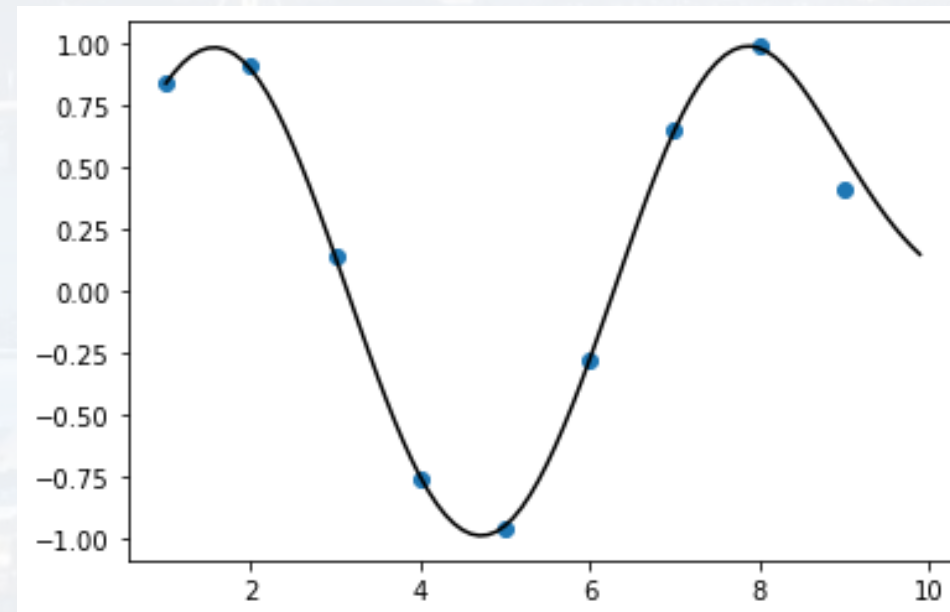
```
    W = W/np.sum(W)
```

```
    yint = np.dot(W.transpose(), y)
```

```
    Scale = np.max(y)/np.max(yint)
```

```
    return yint*Scale
```

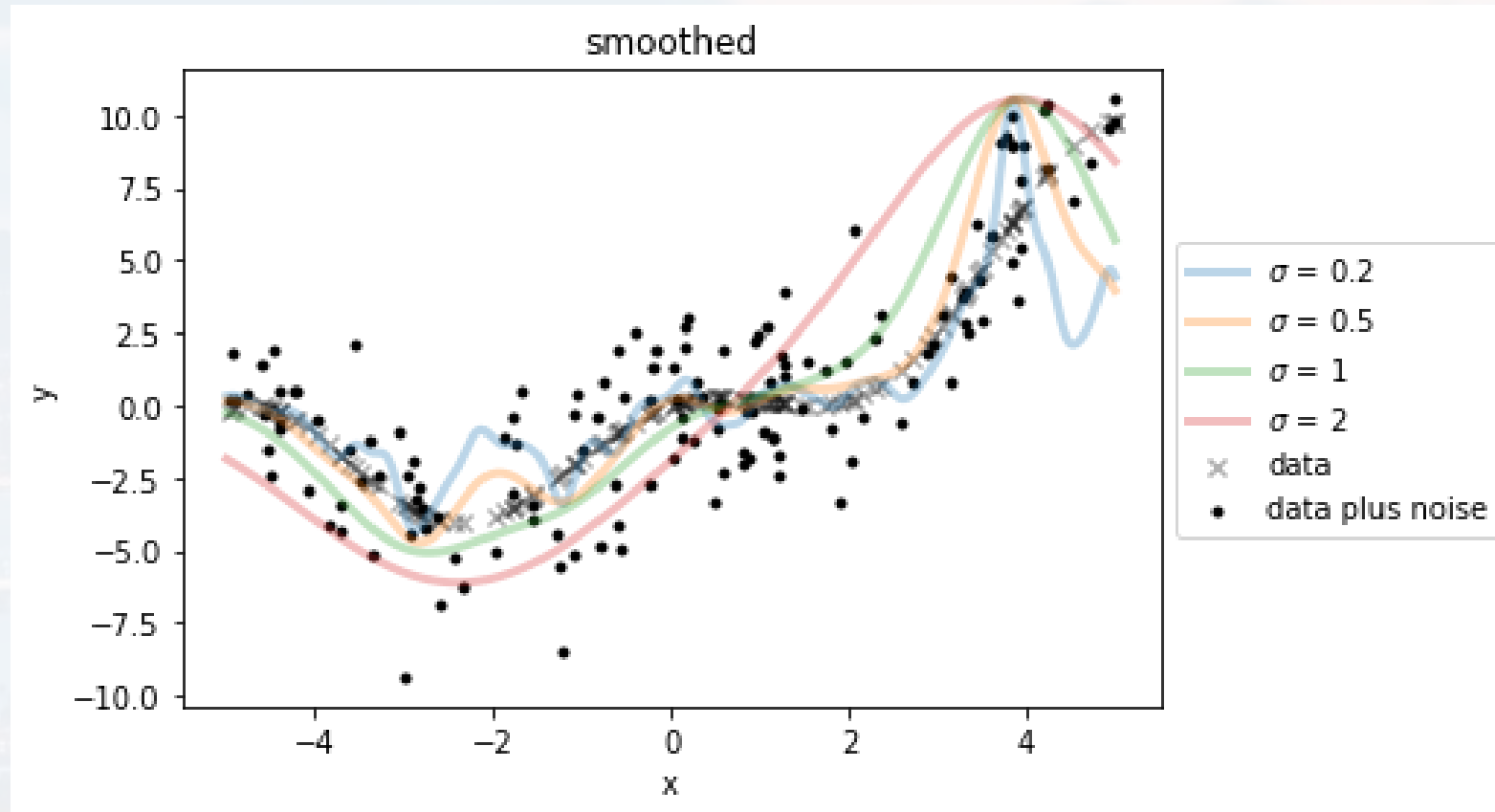
determining how distances are been weighted. Here: normal distribution aka **kernel**





SmoothExamples.py

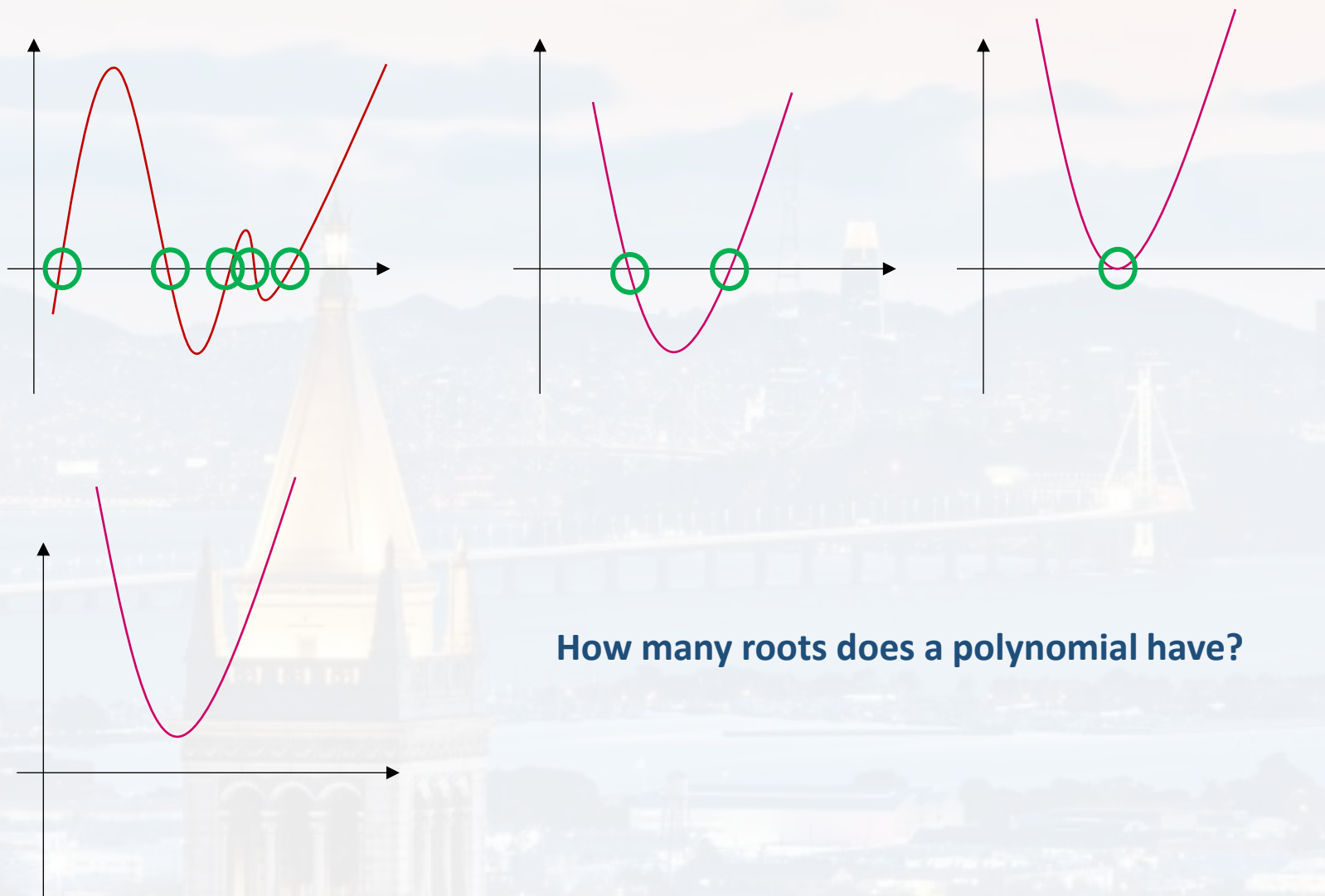
Interpolation
Smoothing
Root Finding





root finding: finding the zeros of a polynomial

Interpolation
Smoothing
Root Finding



How many roots does a polynomial have?



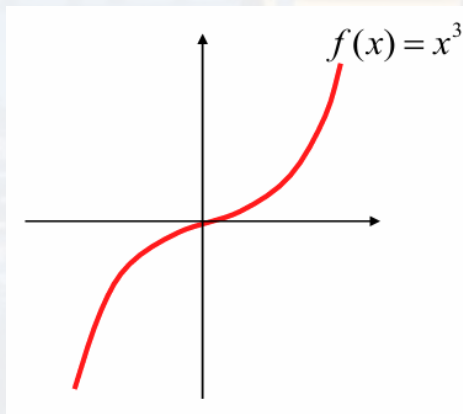
How many roots does a polynomial have?

$$f_N(x) = \sum_{i=0}^N a_i x^i = \alpha \prod_{i=1}^N (x - x_i) \quad \text{factored form}$$

x_i : zeros

- a polynomial of **Nth order** has **N roots** (real & complex)
- for $N \geq 5$: no analytical solutions
- for N is odd: at least one real zero

$$f(x) = x^3 = (x - x_1)(x - x_2)(x - x_3)$$



zeros: $x_1 = x_2 = x_3 = 0$

one zero with multiplicity $m = 3$



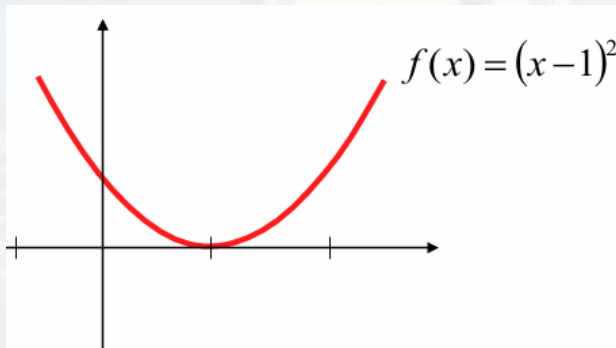
How many roots does a polynomial have?

$$f_N(x) = \sum_{i=0}^N a_i x^i = \alpha \prod_{i=1}^N (x - x_i) \quad \text{factored form}$$

x_i : zeros

- a polynomial of **Nth order** has **N roots** (real & complex)
- for $N \geq 5$: no analytical solutions
- for N is odd: at least one real zero

$$f(x) = x^2 - 2x + 1 = (x - x_1)(x - x_2)$$



zeros: $x_1 = x_2 = 1$

one zero with multiplicity $m = 2$



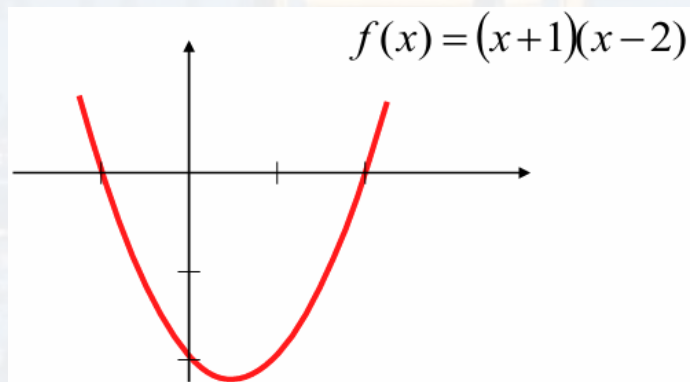
How many roots does a polynomial have?

$$f_N(x) = \sum_{i=0}^N a_i x^i = \alpha \prod_{i=1}^N (x - x_i) \quad \text{factored form}$$

x_i : zeros

- a polynomial of **Nth order** has **N roots** (real & complex)
- for $N \geq 5$: no analytical solutions
- for N is odd: at least one real zero

$$f(x) = x^2 - x - 2 = (x - x_1)(x - x_2)$$



$$f(x) = (x+1)(x-2)$$

zeros: $x_1 = 2$, $x_2 = -1$

two zeros with multiplicity $m = 1$ each



methods:

Interpolation
Smoothing

Root Finding

Root finding [[edit](#)]

Main article: [Root-finding algorithm](#)

- [Bisection method](#)
- [False position method](#): and Illinois method: 2-point, bracketing
- [Halley's method](#): uses first and second derivatives
- [ITP method](#): minmax optimal and superlinear convergence simultaneously
- [Muller's method](#): 3-point, quadratic interpolation
- [Newton's method](#): finds zeros of functions with [calculus](#)
- [Ridder's method](#): 3-point, exponential scaling
- [Secant method](#): 2-point, 1-sided

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

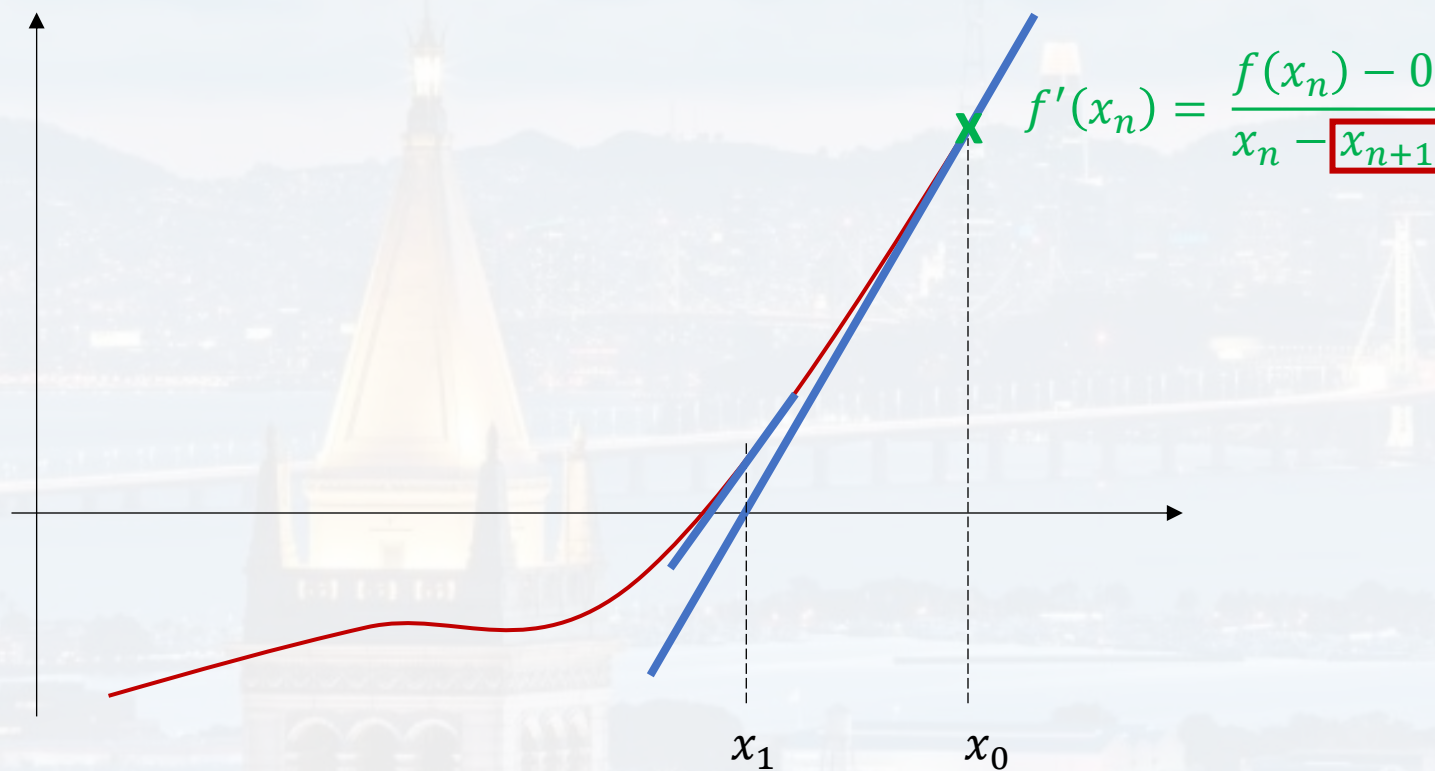


methods:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Interpolation
Smoothing

Root Finding



M. Hohle:

Thank you for your attention!

