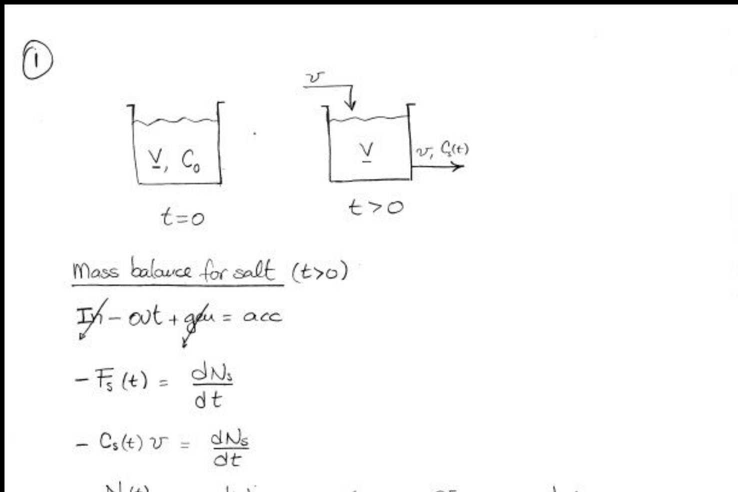
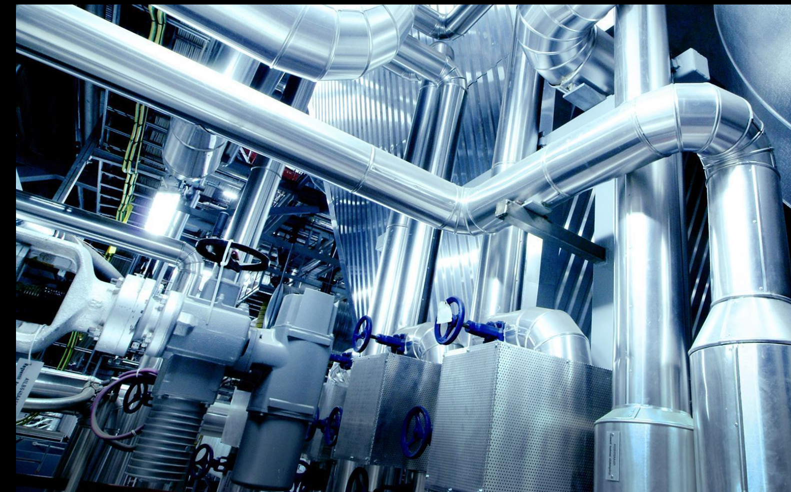


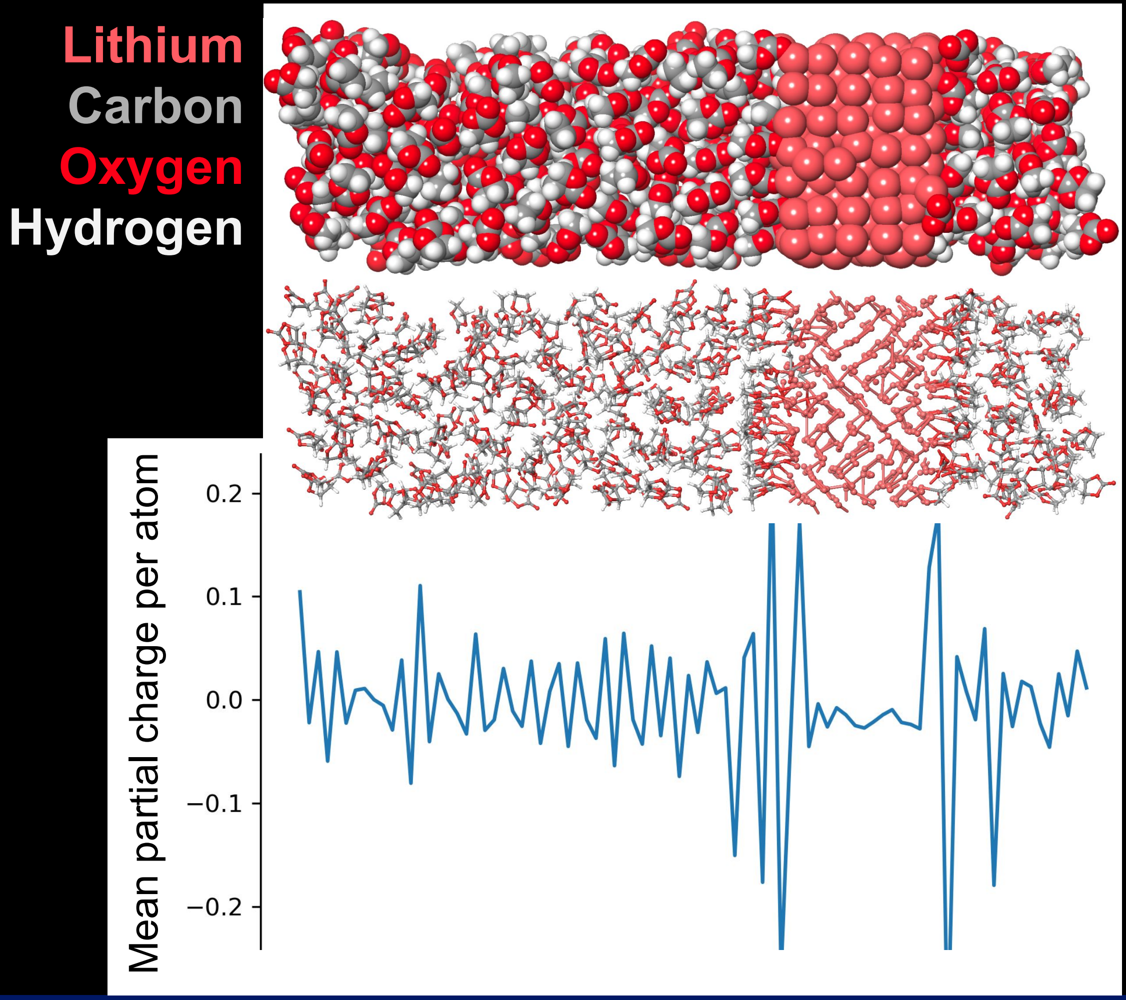


ChE352
Numerical Techniques for Chemical Engineers
Professor Stevenson

Lecture 7

Remote class Feb 21 & 22

I will be presenting at the Schrodinger Materials Science Summit in San Diego



Designing Without The Data

Suppose you need to scale up a reactor to make a new drug ASAP. In order to avoid thermal runaway, you need to know the thermal properties of your reactants and products.

You need heat capacities, but you don't have data at the right temperatures...

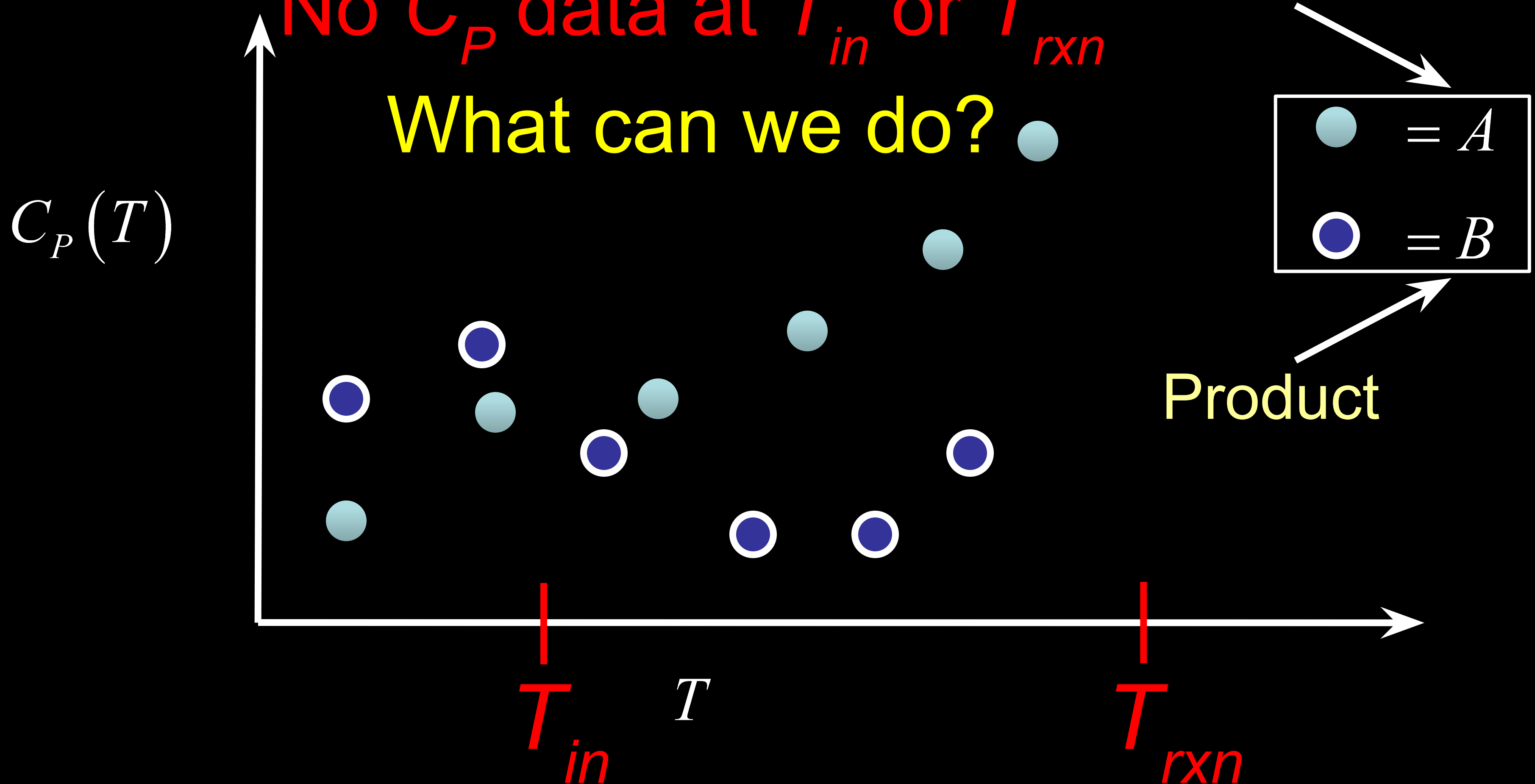
(Very few chemicals have sufficient data)

A Typical Data Situation

Stirred-tank reactor

No C_P data at T_{in} or T_{rxn}

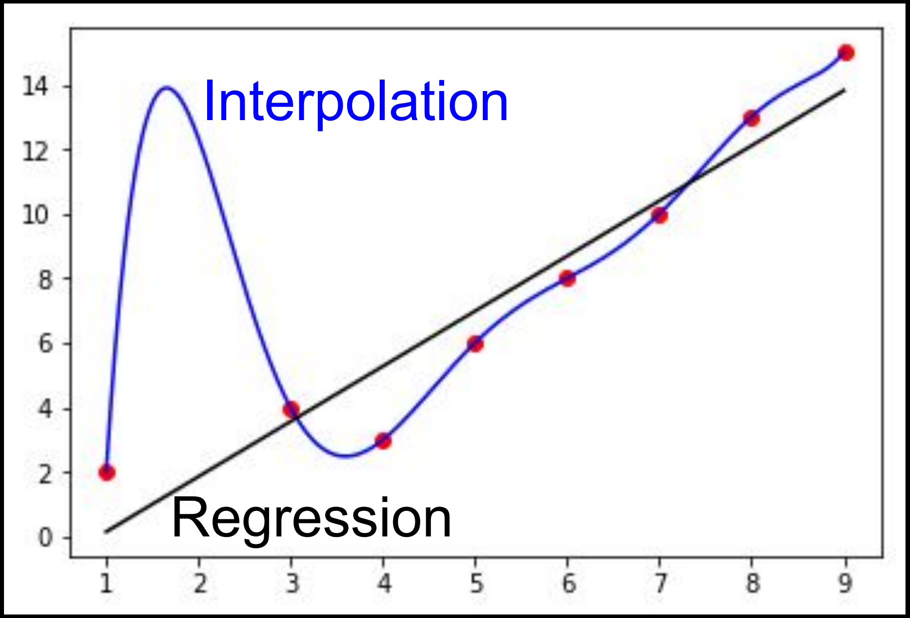
What can we do?



Interpolation & Regression

Interpolation matches data exactly at each known point, but maybe not in between.

Regression minimizes expected error overall - not an exact match at the known points, less risk of a big error.



Which is better?

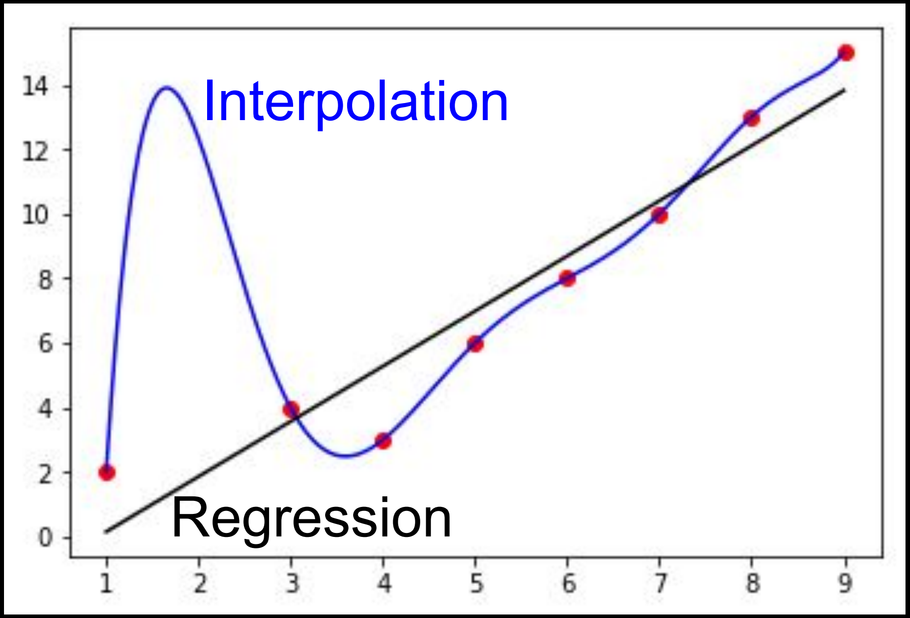
Depends on the engineering problem

Example engineering problems?

Think about the kinds of data that would make one method or another work better...

Interpolation?

Regression?



Which is better?

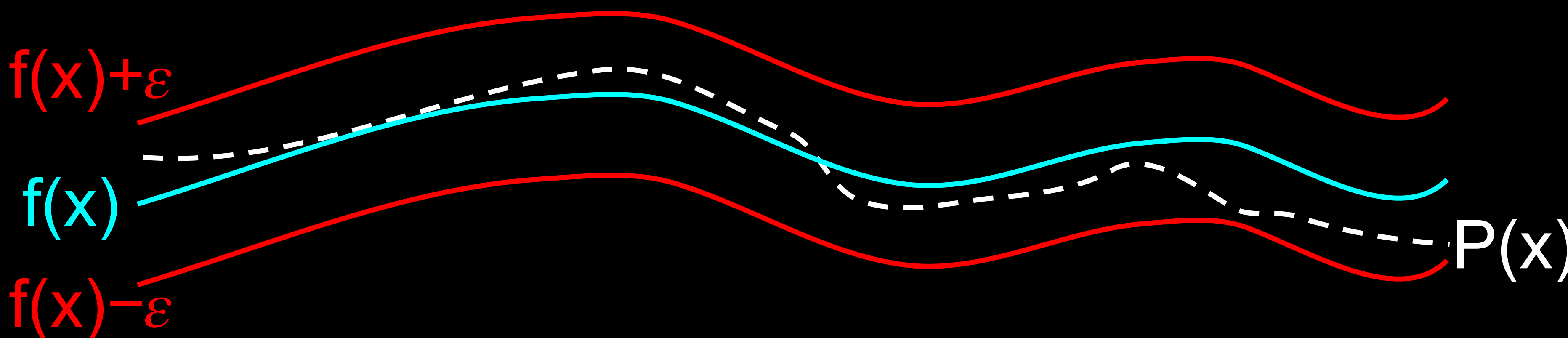
Depends on the engineering problem

Weierstrass theorem

- For any continuous function f, there exists a polynomial P(x) with an error bound ε:

$$\forall f \in C[a,b], \quad \forall \varepsilon > 0$$

$$\exists P(x) : |f(x) - P(x)| < \varepsilon \quad \forall x \in [a,b]$$



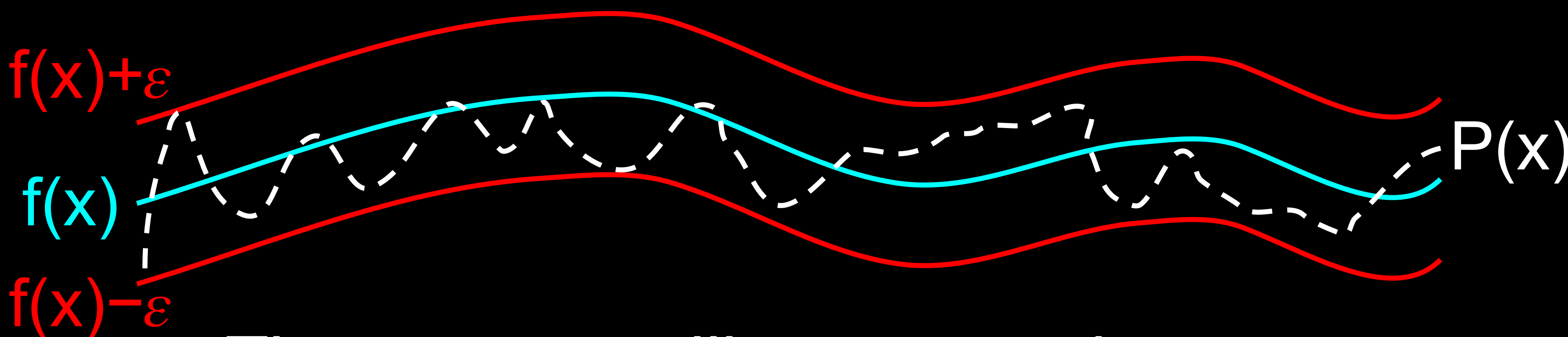
What does this *not* guarantee?

Weierstrass theorem traps

- No bounds on the *derivative* or *curvature*
- No guarantee that the *optima* are the same

$$\forall f \in C[a,b], \quad \forall \varepsilon > 0$$

$$\exists P(x) : |f(x) - P(x)| < \varepsilon \quad \forall x \in [a,b]$$



Theorems are like contract lawyers

We want a well-behaved polynomial

Lagrange Polynomials

- Say we want our polynomial to be exact at our data points, and have lowest possible degree
- This is the definition of <u>Lagrange Polynomials</u>

$$L_{n,j}(x)=\frac{(x-x_0)(x-x_1)\dots(x-x_{j-1})(x-x_{j+1})\dots(x-x_n)}{(x_j-x_0)(x_j-x_1)\dots(x_j-x_{j-1})(x_j-x_{j+1})\dots(x_j-x_n)}$$

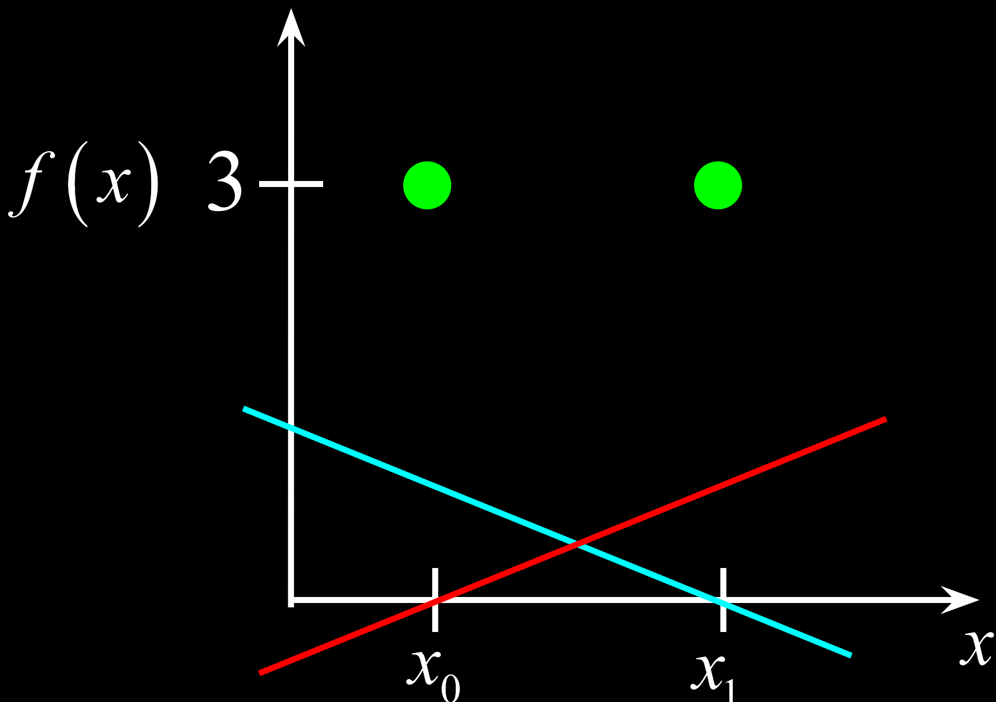
Same

$$\boxed{\underbrace{L_{n,j}(x)}_{\text{Basis}}\equiv\prod_{\substack{i=0\\ i\neq j}}^{n}\frac{(x-x_i)}{(x_j-x_i)}},\qquad \boxed{\underbrace{P_n(x)}_{\text{Lagrange}}\equiv\sum_{j=0}^{n}\underbrace{f(x_j)}_{\text{Data}}\underbrace{L_{n,j}(x)}_{\text{Basis}}}$$

Lagrange polynomial for 2 points

Find the Lagrange polynomial for this data:

$$x_0, y_0 = (1, 3), x_1, y_1 = (2, 3)$$



$$L_0(x) = \frac{x - x_1}{x_0 - x_1}$$

$$L_1(x) = \frac{x - x_0}{x_1 - x_0}$$

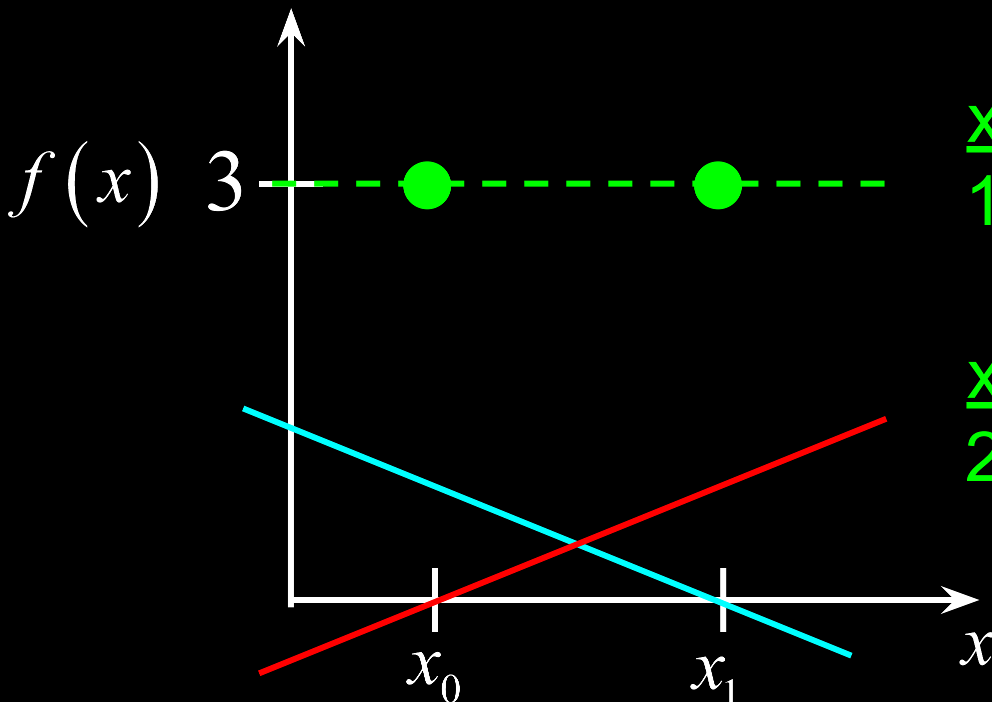
With a weighted sum of these two lines, you can make any line!

$$P(x) = L_0(x) f(x_0) + L_1(x) f(x_1) \approx f(x) \; on \; [x_0, x_1]$$

Lagrange polynomial for 2 points

Find the Lagrange polynomial for this data:

$$x_0, y_0 = (1, 3),\ x_1, y_1 = (2, 3)$$



$$\frac{x-2}{1-2} \qquad \boxed{L_0(x) = \frac{x - x_1}{x_0 - x_1}}$$

$$\frac{x-1}{2-1} \qquad \boxed{L_1(x) = \frac{x - x_0}{x_1 - x_0}}$$

In this case, a constant: 3

$$(2 - x) \;*\; 3 \quad + \quad (x - 1) \;*\; 3 \quad = \quad \mathbf{3}$$

$$\boxed{P(x) = L_0(x) f(x_0) + L_1(x) f(x_1) \approx f(x) \; on \; [x_0, x_1]}$$

Lagrange math & code

$$\boxed{L_{n,j}(x) \equiv \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}},$$

$$\boxed{P_n(x) \equiv \sum_{j=0}^{n} f(x_j) L_{n,j}(x)}$$

```python
def lagrange_Lj(x, j, x_data):
    '''
    Returns the j-th Lagrange function L,
        as defined by the points x_data,
        evaluated at point x.
    '''
    L = 1.0
    for i in range(len(x_data)):
        if i==j:
            continue
        dx = x - x_data[i]
        x_interval = x_data[j] - x_data[i]
        L *= dx / x_interval
    return L
```

```python
def lagrange_P(x, x_data, y_data):
    '''
    Returns the value of the Lagrange
        polynomial P defined by x_data,
        y_data, evaluated at point x.
    '''
    assert len(x_data) == len(y_data)
    P = 0.0
    for j in range(len(x_data)):
        Lj = lagrange_Lj(x, j, x_data)
        P += y_data[j] * Lj
    return P
```

Lagrange Polynomials for n > 1

- What if we have 3, 5, 10, etc. data points?
- Same procedure: values of $f(x_k)$ are the weights for the basis functions $L_{n,k}(x)$

$$L_{n,j}(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)(x_j-x_1)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}$$

Same

$$\boxed{\underbrace{L_{n,j}(x)}_{\text{Basis}} \equiv \prod_{\substack{i=0 \\ i\neq j}}^{n} \frac{(x-x_i)}{(x_j-x_i)}},$$

Lagrange

$$\boxed{P_n(x) \equiv \sum_{j=0}^{n} \underbrace{f(x_j)}_{\text{Data}} \underbrace{L_{n,j}(x)}_{\text{Basis}}}$$

$P_n(x)$ has Bounded Error, but. . .

We can bound the error in using $P_n(x)$ to approximate f(x) (Look familiar?):

True function

Error term

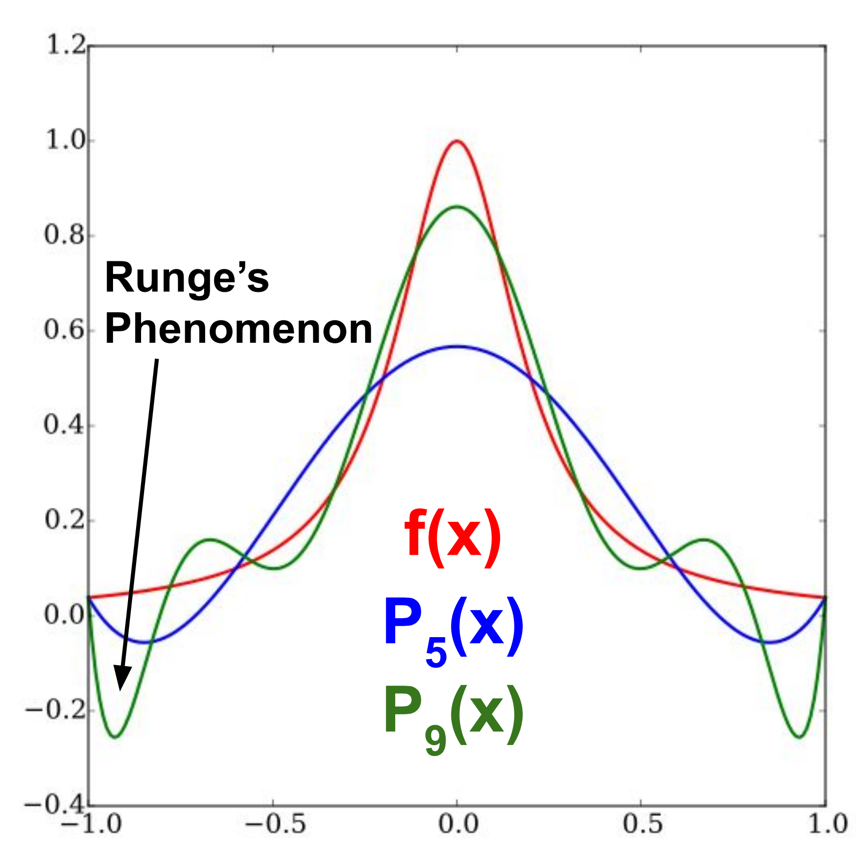
$$f(x) = P_n(x) + \frac{f^{(n+1)}(\zeta(x))}{(n+1)!}(x-x_0)(x-x_1)\ldots(x-x_n)$$

Lagrange

Not always helpful: the true f(x) is not known when we only have data, let alone a bound on its $(n+1)^{th}$ derivative.

Problem: *Runge's Phenomenon*

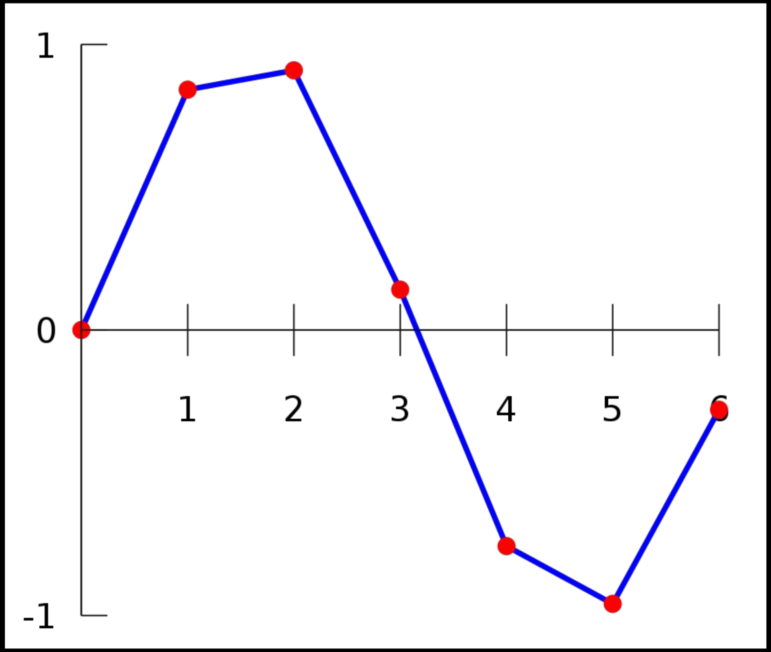
- Lagrange gives higher-n polynomials for higher-n datasets
- High-n polynomials have large derivatives (Why?)
- Between datapoints, P(x) will oscillate ("ring")
- This can get really bad



How can we fit more points without higher-order polynomials?

Piecewise Methods

- Piecewise linear interpolation – draw/calculate a line between adjacent data points to estimate an intermediate value – How many lines for n+1 points?

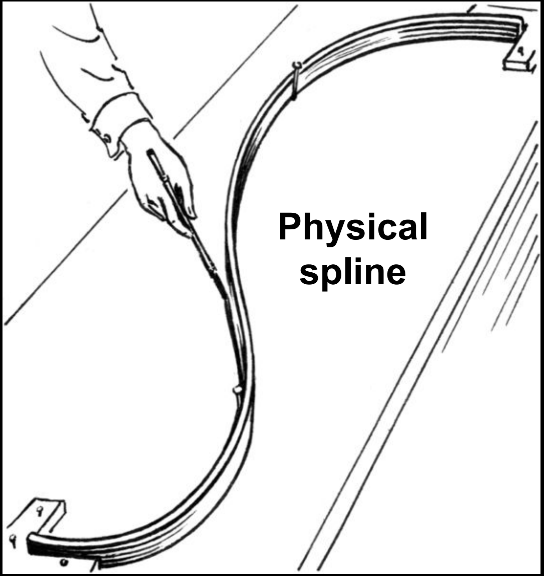


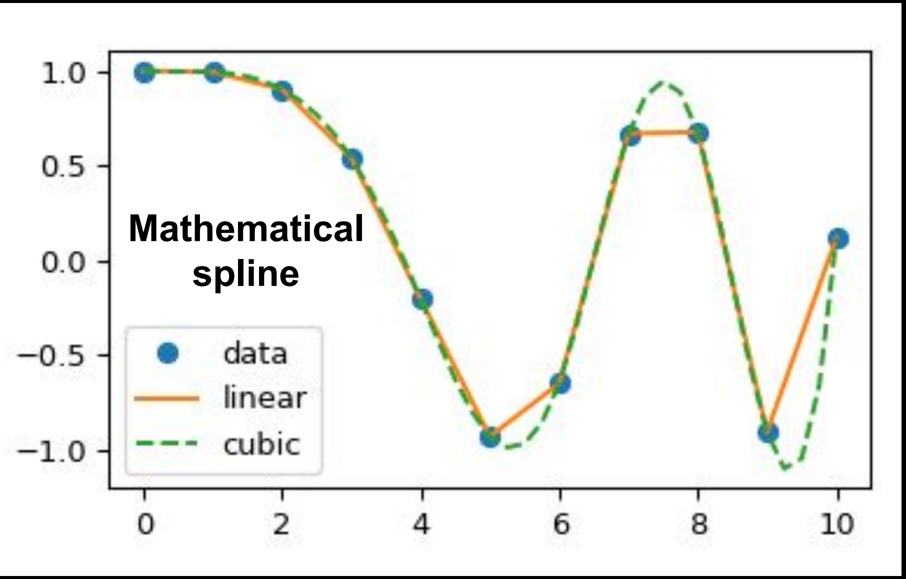
- Simple
- Robust
- Any dataset size
- No oscillation

Any problems?

Splines

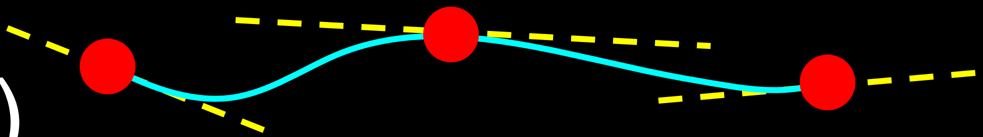
- Piecewise low-n polynomials fitted to match each datapoint <u>and be smooth</u>
- Based on how an elastic piece of metal (a "spline") physically curves between points





Cubic splines

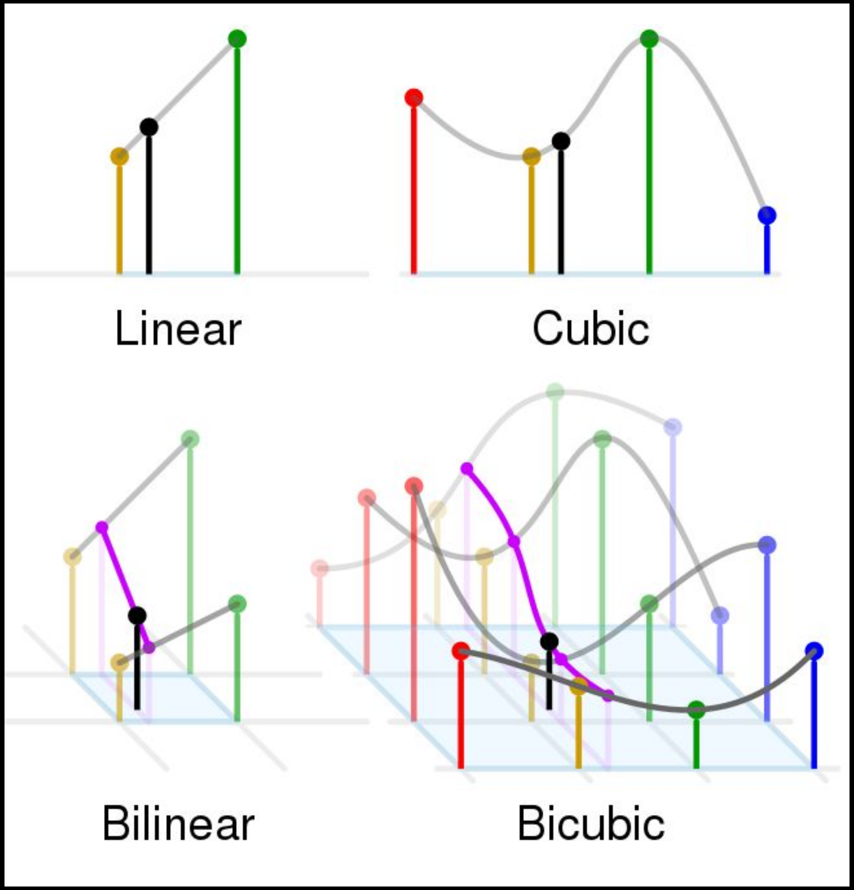
- **Smooth** means that for each datapoint x_i:
  - $P'_i(x_i) == P'_{i+1}(x_i)$
  - $P''_i(x_i) == P''_{i+1}(x_i)$



- Every cubic has 4 coefficients
- Every point has x, f(x), P'(x), P''(x)
  - Get P', P'' from $f(x_{i+1}) - f(x_i) / (x_{i+1} - x_i)$
  - Need to choose P', P'' at boundaries
- Solve a linear system of equations to determine coefficients (Ax = b)

Higher dimension interpolation

- Example: heat capacity as a function of T and P (more dimensions: mixture ratio...)



- Can use similar methods
- Getting enough data is even harder
- Regression helps
- Difficult to visualize after 2-D

Tools for interpolation

- Spreadsheet – trendlines on graphs are interpolation if polynomial is order <= n for n+1 points, regression otherwise (Ax = b)
- Python – numpy.polyfit, numpy.polyval (evaluates polynomial at point), scipy.interpolate.lagrange, scipy.interpolate.CubicSpline
- Calculator – never a bad idea to try a simple linear interpolation as a check
- Brain - nearest-neighbor is very fast!

Interpolation Summary

- You can always find a single polynomial of order n which fits any n+1 data points...
  - ...But if $n > 4$, you should use a piecewise method
- Lagrange (and similar methods) use one polynomial for entire data set
- You can use interpolation methods to extrapolate, but expect problems (Why?)
  - Safer to use linear, not higher order (Why?)
- We'll see Lagrange's $P_n(x)$ again soon...

Summary and Problems

- Open Python Numerical Methods, go to Chapter 17.6: Summary and Problems

https://pythonnumericalmethods.berkeley.edu/notebooks/chapter17.06-Summary-and-Problems.html

- Do problem 1:

```
def my_lin_interp(x, y, X):
    # returns an array Y containing linear
    # interpolation values of data x,y
    # for the array of desired inputs X
```