L9: Polynomial Interpolation (Chapter 17)

BME 313L

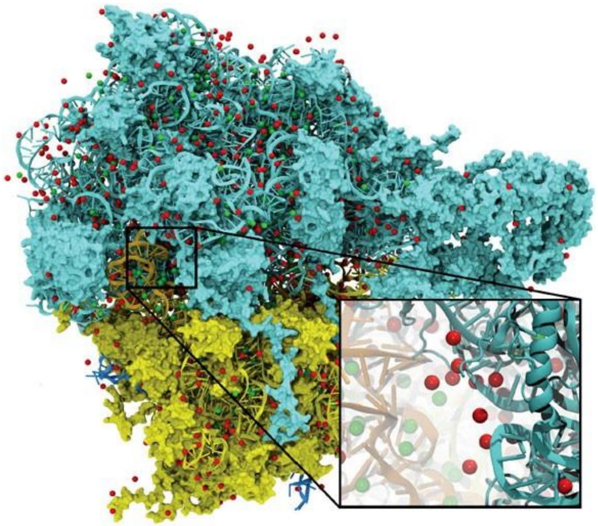
Introduction to Numerical Methods in BME

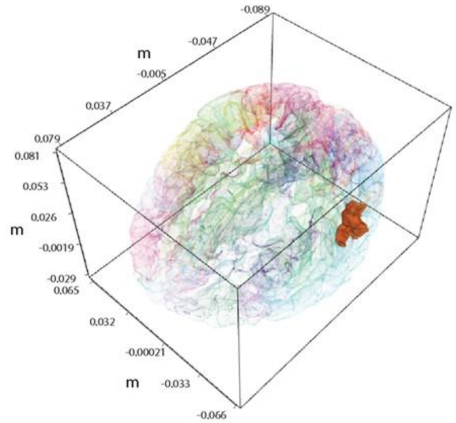
Tim Yeh

Department of Biomedical Engineering

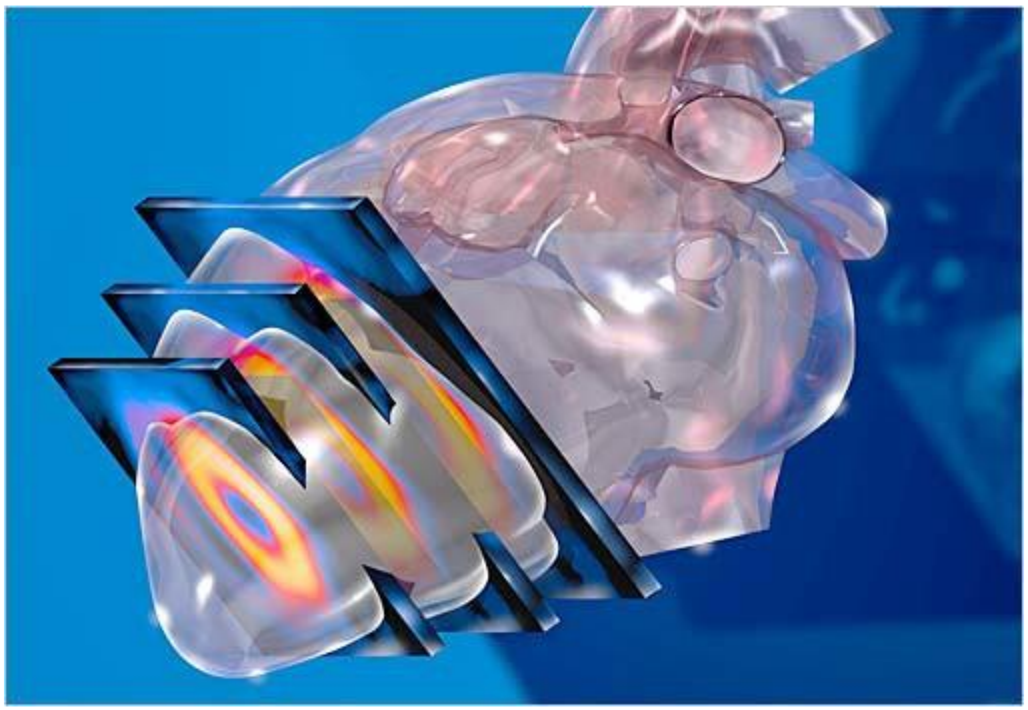
University of Texas at Austin

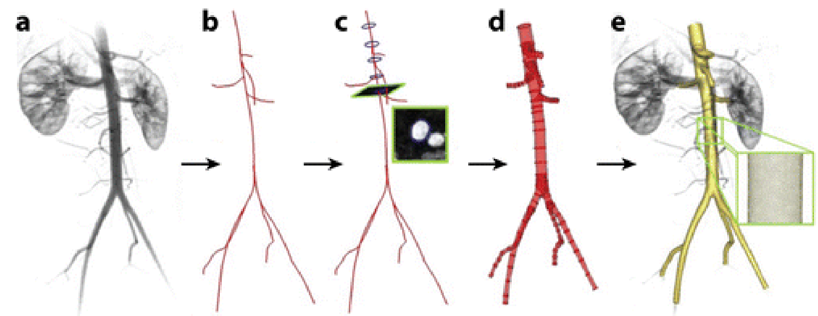
Tim.Yeh@austin.utexas.edu





Polynomial Interpolation and Extrapolation





- Polynomial interpolation
- Newton's and Lagrange's interpolation
- Inverse interpolation
- Extrapolation and oscillations
- Multi-dimensional interpolation

Interpolation (Extrapolation)

Learning Objectives

- Recognizing that evaluating **polynomial coefficients** with simultaneous equations is an **ill-conditioned problem**.
- Knowing how to evaluate polynomial coefficients and interpolate with MATLAB's ***polyfit*** and ***polyval*** functions.
- Knowing how to perform an interpolation with **Newton's and Lagrange's** polynomials.
- Knowing how to solve an **inverse interpolation problem** by recasting it as a **roots problem**.
- Appreciating the **dangers of extrapolation**.
- Recognizing that higher-order polynomials can manifest **large oscillations**.

Bungee Jumper Problem

- What has bee considered?

- Drag coefficient is complicated….

Important Properties Provided in Tabular Form

- You will frequently have occasions to **estimate intermediate values** between <u>precise</u> data points.

TABLE 17.1 Density (ρ), dynamic viscosity (μ), and kinematic viscosity (ν) as a function of temperature (T) at 1 atm as reported by White (1999).

T, °C	ρ, kg/m³	μ, N · s/m²	ν, m²/s
−40	1.52	1.51×10^{-5}	0.99×10^{-5}
0	1.29	1.71×10^{-5}	1.33×10^{-5}
20	1.20	1.80×10^{-5}	1.50×10^{-5}
50	1.09	1.95×10^{-5}	1.79×10^{-5}
100	0.946	2.17×10^{-5}	2.30×10^{-5}
150	0.835	2.38×10^{-5}	2.85×10^{-5}
200	0.746	2.57×10^{-5}	3.45×10^{-5}
250	0.675	2.75×10^{-5}	4.08×10^{-5}
300	0.616	2.93×10^{-5}	4.75×10^{-5}
400	0.525	3.25×10^{-5}	6.20×10^{-5}
500	0.457	3.55×10^{-5}	7.77×10^{-5}

25°C ?

Another Example: Viscosity of Glycerol-Water Mixture

- You will frequently have occasions to **estimate intermediate values** between <u>precise</u> data points.

TABLE V. VISCOSITY OF AQUEOUS GLYCEROL SOLUTIONS

25°C

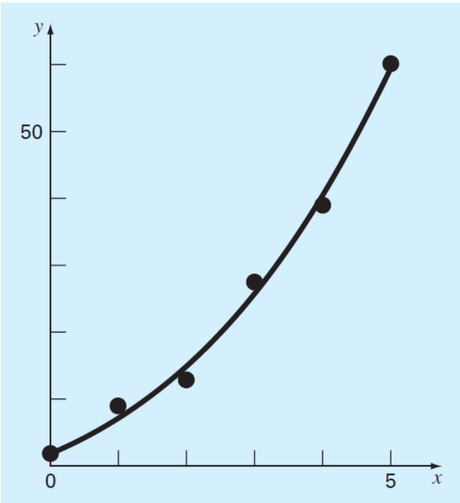
45%

Glycerol, % Wt.	0	10	20	30	40	50	60	70	80	90	100
	Temperature, ° C. / Viscosity, Centipoises										
0[a]	1.792	1.308	1.005	0.8007	0.6560	0.5494	0.4688	0.4061	0.3565	0.3165	0.2838
10	2.44	1.74	1.31	1.03	0.826	0.680	0.575	0.500	...	...	...
20	3.44	2.41	1.76	1.35	1.07	0.879	0.731	0.635	...	...	...
30	5.14	3.49	2.50	1.87	1.46	1.16	0.956	0.816	0.690	...	...
40	8.25	5.37	3.72	2.72	2.07	1.62	1.30	1.09	0.918	0.763	0.668
50	14.6	9.01	6.00	4.21	3.10	2.37	1.86	1.53	1.25	1.05	0.910
60	29.9	17.4	10.8	7.19	5.08	3.76	2.85	2.29	1.84	1.52	1.28
65	45.7	25.3	15.2	9.85	6.80	4.89	3.66	2.91	2.28	1.86	1.55
67	55.5	29.9	17.7	11.3	7.73	5.50	4.09	3.23	2.50	2.03	1.68
70	76.0	38.8	22.5	14.1	9.40	6.61	4.86	3.78	2.90	2.34	1.93
75	132	65.2	35.5	21.2	13.6	9.25	6.61	5.01	3.80	3.00	2.43
80	255	116	60.1	33.9	20.8	13.6	9.42	6.94	5.13	4.03	3.18
85	540	223	109	58.0	33.5	21.2	14.2	10.0	7.28	5.52	4.24
90	1310	498	219	109	60.0	35.5	22.5	15.5	11.0	7.93	6.00
91	1590	592	259	126	68.1	39.8	25.1	17.1	11.9	8.62	6.40
92	1950	729	310	147	78.3	44.8	28.0	19.0	13.1	9.46	6.82
93	2400	860	367	172	89.0	51.5	31.6	21.2	14.4	10.3	7.54
94	2930	1040	437	202	105	58.4	35.4	23.6	15.8	11.2	8.19
95	3690	1270	523	237	121	67.0	39.9	26.4	17.5	12.4	9.08
96	4600	1585	624	281	142	77.8	45.4	29.7	19.6	13.6	10.1
97	5770	1950	765	340	166	88.9	51.9	33.6	21.9	15.1	10.9
98	7370	2460	939	409	196	104	59.8	38.5	24.8	17.0	12.2
99	9420	3090	1150	500	235	122	69.1	43.6	27.8	19.0	13.2
100	12070	3900	1412	612	284	142	81.3	50.6	31.9	21.3	14.8

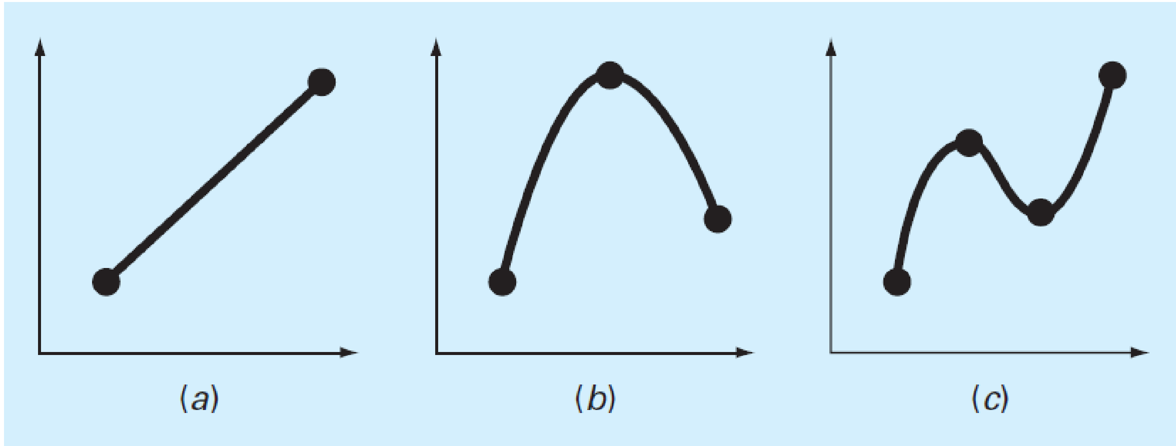
[a] Viscosity of water taken from Bingham and Jackson (4).

What is viscosity of pure water or pure glycerol?
What do you see here?

Difference Between Curve Fitting and Polynomial Interpolation



Fit with a parabola (Chapter 15, p.364)



Interpolating polynomials (Chapter 17, p.407)

- Polynomial interpolation consists of determining the unique **(n-1)th**-order polynomial that **passes through** n data points.
- However, the least-squares fit line does not necessarily pass through any of the points, but rather follows the **general trend of the data**.

Pretty much every interpolation you want to do, you can find "calculator" on line

Calculate density and viscosity of glycerol/water mixtures

Enter temperature [C]: 0

Enter volume of water [litres]: 0

Enter volume of glycerol [litres]: 0

Click this button to do the sum: Calculate

Fraction of glycerol by volume is:

Fraction of glycerol by mass is:

Density of mixture is [kg/m^3]:

Dynamic viscosity of mixture is [Ns/m^2]:

Kinematic viscosity of mixture is [m^2/s]:

Based on the parameterisation in Cheng (2008) *Ind. Eng. Chem. Res.* **47** 3285-3288

- In old days, we relied on "**tables**" and "**graphs**"
- In modern days, there is always a "**calculator**" that you can find on internet

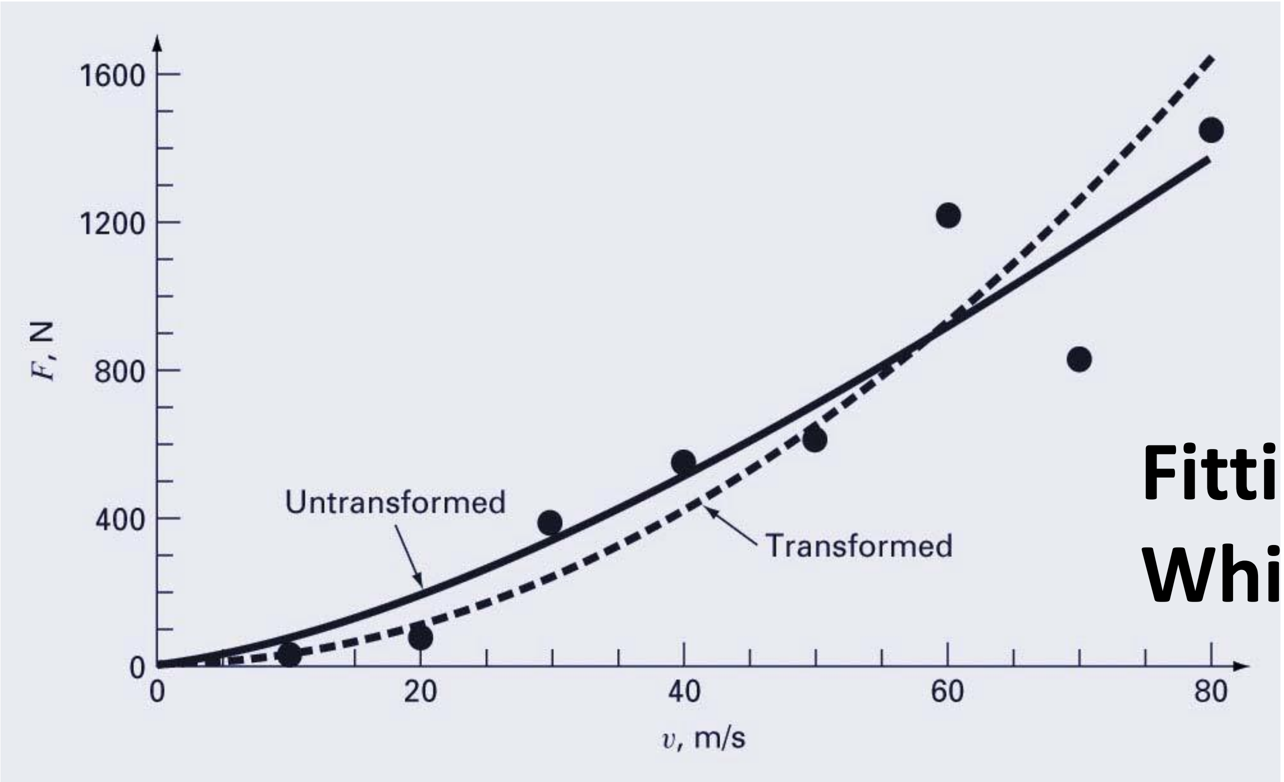
Polynomial Interpolation

- You will frequently have occasions to **estimate intermediate values** between precise data points.
- The function you use to interpolate must "**pass through" the actual data points -** this makes interpolation more **restrictive** than fitting.
- The most common method for this purpose is **polynomial interpolation**, where an **$(n\text{-}1)^{th}$** order polynomial passes through ***n*** data points:

"<u>one and only one polynomial</u> of order $(n-1)^{th}$"…..
Why?

Nonlinear regression result is different from "transformed linear regression" result (Chapter 15.5, p. 372)

- This is because the former minimizes the residuals of the original data whereas the latter minimizes the residuals of the transformed data



Fitting is less restrictive!
Which one is better?

Polynomial Interpolation

- To uniquely determine an **(n-1)th** order equation, how many data points are required?

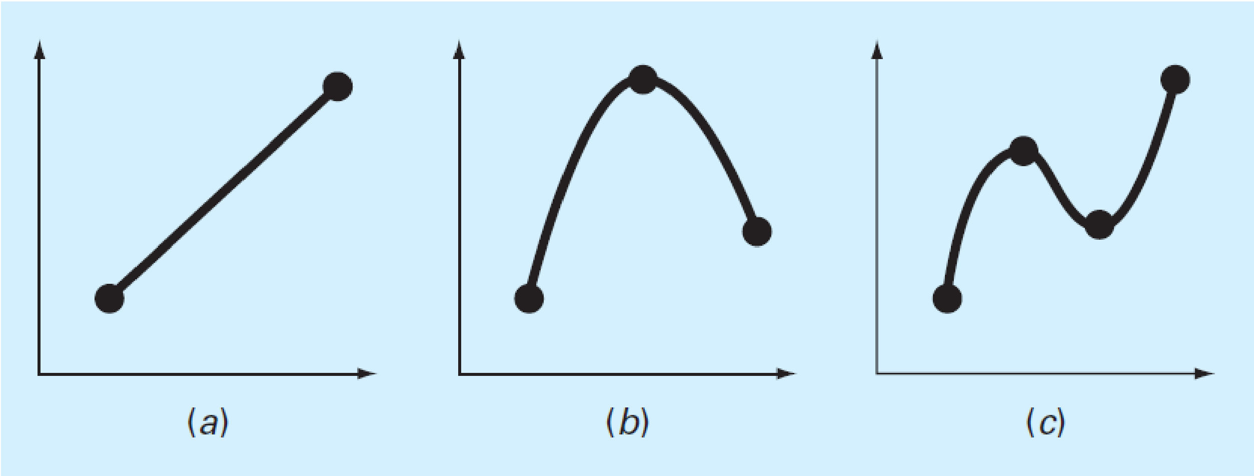


FIGURE 17.1
Examples of interpolating polynomials: (*a*) first-order (linear) connecting two points, (*b*) second-order (quadratic or parabolic) connecting three points, and (*c*) third-order (cubic) connecting four points.

$$f(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}$$

MATLAB version :

$$f(x) = p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x + p_n$$

Determining Coefficients

- Since polynomial interpolation provides as many **simultaneous equations (*n*)** as there are **data points (*n*)**, the polynomial coefficients (n) can be <u>found exactly</u> using linear algebra.

$$f(x) = p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x + p_n$$

- MATLAB's built in ***polyfit*** and ***polyval*** commands can also be used here - all that is required is making sure the order of the fit for ***n*** data points is ***n*-1**.

Example: Fit 3 Points with a Quadratic Equation

$$f(x) = p_1 x^2 + p_2 x + p_3$$

$$x_1 = 300 \quad f(x_1) = 0.616$$
$$x_2 = 400 \quad f(x_2) = 0.525$$
$$x_3 = 500 \quad f(x_3) = 0.457$$

$$0.616 = p_1(300)^2 + p_2(300) + p_3$$
$$0.525 = p_1(400)^2 + p_2(400) + p_3$$
$$0.457 = p_1(500)^2 + p_2(500) + p_3$$

or in matrix form:

$$\begin{bmatrix} 90{,}000 & 300 & 1 \\ 160{,}000 & 400 & 1 \\ 250{,}000 & 500 & 1 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix} = \begin{Bmatrix} 0.616 \\ 0.525 \\ 0.457 \end{Bmatrix}$$

Do left division to find coefficients (p=A\b).

Alternatively

```
>> format long
>> T = [300 400 500];
>> density = [0.616 0.525 0.457];
>> p = polyfit(T,density,2)

p =
   0.00000115000000  -0.00171500000000   1.02700000000000
```

Fit with 2nd order eqn.

We can then use the `polyval` function to perform an interpolation as in

```
>> d = polyval(p,350)

d =
   0.56762500000000
```

- **Small resulting coefficients**
- **Ill-conditioned system (chapter 9, Gauss Elimination, p. 242-243)**
- **Extremely sensitive to "roundoff error"**

We need a better method for interpolation

Vandermonde Matrices

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix} = \begin{Bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{Bmatrix} \qquad \begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1}^{n-1} & x_{n-1}^{n-2} & \cdots & x_{n-1} & 1 \\ x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{n-1} \\ p_n \end{Bmatrix} = \begin{Bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{Bmatrix}$$

- Coefficient matrices of this form are referred to as ***Vandermonde matrices***, and they are very **ill-conditioned** - meaning their solutions are very sensitive to **roundoff errors**.
- The issue can be improved by **scaling and centering** (help *polyfit*) the data (see example 17.6, p422).
- Or by alternative approaches that do not manifest this shortcoming – **Newton** and **Lagrange polynomials**

Scaling and Centering (p. 422, also described in HELP *polyfit*)

The population in millions of the United States from 1920 to 1990

Date	1920	1930	1940	1950	1960	1970	1980	1990
Population	106.46	123.08	132.12	152.27	180.67	205.05	227.23	249.46

```
t = [1920:10:1990];
pop = [106.46 123.08 132.12 152.27 180.67 205.05 227.23
       249.46];
p = polyfit(t,pop,7)
Warning: Polynomial is badly conditioned. Remove repeated data
         points or try centering and scaling as described in HELP
         POLYFIT.
```

How to solve the issue?

No need to use the Common Era as year-number system

```
ts = (t - 1955)/35; p = polyfit(ts,pop,7);
```

Newton Interpolating Polynomials

- Another way to express a polynomial interpolation is to use ***Newton's interpolating polynomial***.

- The differences between a **simple polynomial** and **Newton interpolating polynomial** for first and second order interpolations are:

Order	Simple	Newton
$1st$	$f_1(x) = a_1 + a_2 x$	$f_1(x) = b_1 + b_2(x - x_1)$
$2nd$	$f_2(x) = a_1 + a_2 x + a_3 x^2$	$f_2(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2)$

What do we gain here?

1. **Number of coefficients reduced?**
2. **Formula simplified?**
3. **Then what is the benefit of Newton polynomial?**

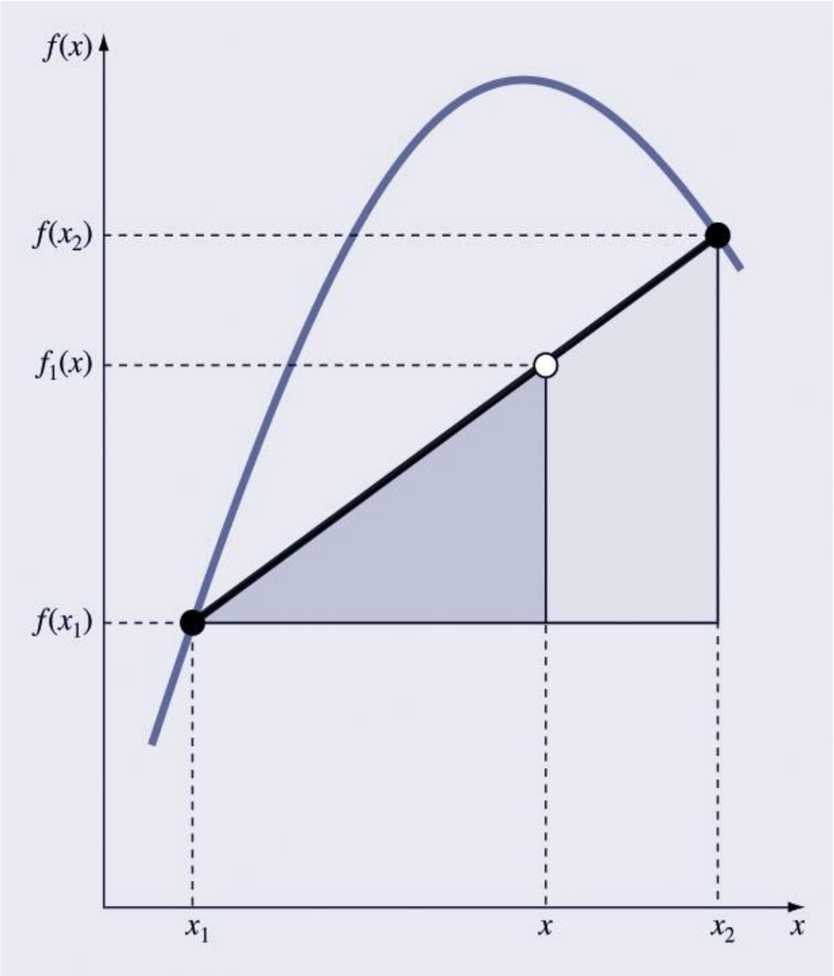
Newton Linear Interpolation

Order	Simple	Newton
1st	$f_1(x) = a_1 + a_2 x$	$f_1(x) = b_1 + b_2(x - x_1)$
2nd	$f_2(x) = a_1 + a_2 x + a_3 x^2$	$f_2(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2)$

- The **first-order Newton interpolating polynomial** may be obtained from **linear** interpolation and **similar triangles**, as shown.
- The resulting formula based on known points x_1 and x_2 and the values of the dependent function at those points is:

$$f_1(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1)$$

$$f_1(x) = b_1 + b_2(x - x_1)$$



Problem?

Newton Quadratic Interpolation

Order	Simple	Newton
1st	$f_1(x) = a_1 + a_2 x$	$f_1(x) = b_1 + b_2(x - x_1)$
2nd	$f_2(x) = a_1 + a_2 x + a_3 x^2$	$f_2(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2)$

$$f(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2)$$

$$b_1 = f(x_1) \qquad b_2 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$b_3 = \frac{\dfrac{f(x_3) - f(x_2)}{x_3 - x_2} - \dfrac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1}$$

Three data points:

x_1, $f(x_1)$
x_2, $f(x_2)$
x_3, $f(x_3)$

b_3 looks like "**finite difference approximation**" of the 2nd derivative! This is called **finite divided difference** on p. 413

- Looks like we can solve **Newton Interpolating Polynomial** with **iterative approaches**, without the need to solve **linear algebraic equations**

Newton Cubic Interpolation (not in textbook)

Order	Simple	Newton
1st	$f_1(x) = a_1 + a_2 x$	$f_1(x) = b_1 + b_2(x - x_1)$
2nd	$f_2(x) = a_1 + a_2 x + a_3 x^2$	$f_2(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2)$

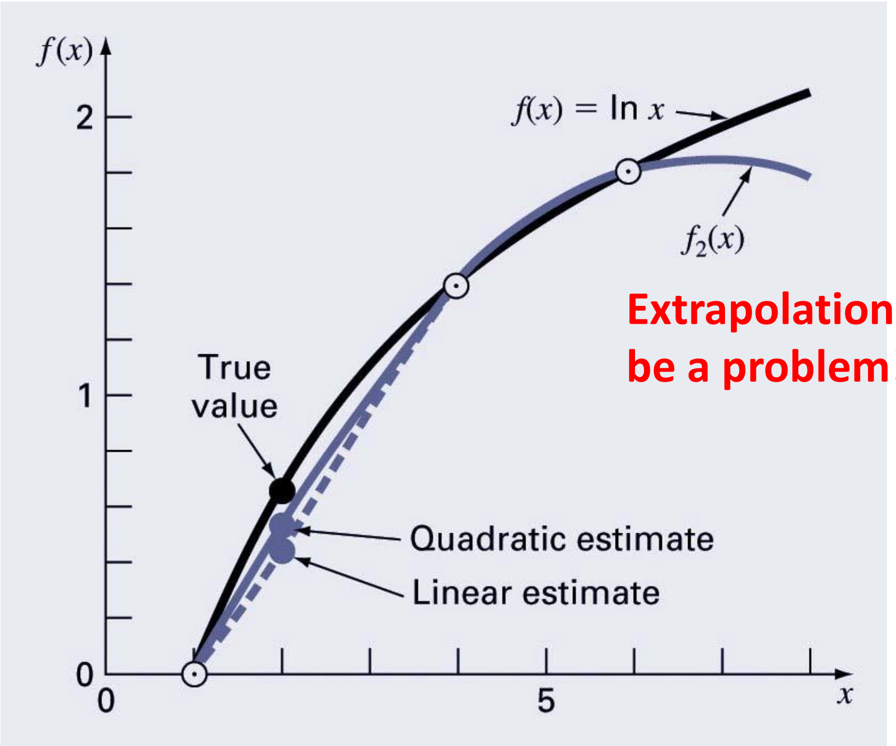
$$f(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2)$$

$$b_1 = f(x_1) \qquad b_2 = \frac{f(x_2) - f(x_1)}{x_2 - x_1} \qquad b_3 = \frac{\dfrac{f(x_3) - f(x_2)}{x_3 - x_2} - \dfrac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1}$$

What about *b_4*?

Linear vs. Quadratic Interpolation

- We know ln(1) = 0, ln(4) = 1.386294, and ln(6) = 1.791759. Please estimate **ln(2)**.
- Yes, use polynomial interpolation.
- Which one gives a better estimate? Linear interpolation or Quadratic interpolation?
- Why?
- The **second-order** interpolating polynomial introduces some **curvature** to the line connecting the points, but still goes through the first two points.



Extrapolation can be a problem!

General Form of Newton's Interpolating Polynomial

- The general formula for (n-1)th-order polynomial ($f_{n-1}(x)$) is:

$$f_{n-1}(x) = b_1 + b_2(x - x_1) + \cdots + b_n(x - x_1)(x - x_2)\cdots(x - x_{n-1})$$

where

$$b_1 = f(x_1)$$
$$b_2 = f[x_2, x_1]$$
$$b_3 = f[x_3, x_2, x_1]$$
$$\vdots$$
$$b_n = f[x_n, x_{n-1}, \cdots, x_2, x_1]$$

$$\boxed{b_n = f[x_n, x_{n-1}, \cdots, x_2, x_1] = \frac{f[x_n, x_{n-1}, \ldots, x_2] - f[x_{n-1}, x_{n-2}, \ldots, x_1]}{x_n - x_1}}$$

b_n is based on **2** previously calculated terms!

and the *f* [...] represent ***finite divided differences*** (p. 413)

Divided Difference Table (p. 414)

- Divided difference are calculated as follows:

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$$

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k}$$

$$f[x_n, x_{n-1}, \cdots, x_2, x_1] = \frac{f[x_n, x_{n-1}, \cdots, x_2] - f[x_{n-1}, x_{n-2}, \cdots, x_1]}{x_n - x_1}$$

$$\boxed{b_n = f[x_n, x_{n-1}, \cdots, x_2, x_1] = \frac{f[x_n, x_{n-1}, \ldots, x_2] - f[x_{n-1}, x_{n-2}, \ldots, x_1]}{x_n - x_1}}$$

- Graphical depiction of the **recursive nature** of divided differences.

x_i	$f(x_i)$	First	Second	Third
x_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	$f[x_4, x_3, x_2, x_1]$
x_2	$f(x_2)$	$f[x_3, x_2]$	$f[x_4, x_3, x_2]$	
x_3	$f(x_3)$	$f[x_4, x_3]$		
x_4	$f(x_4)$			

This is my solution!

Where is my solution (b_1, b_2, b_3.....b_n)?

MATLAB Implementation (p. 416)

```
function yint = Newtint(x,y,xx)
% Newtint: Newton interpolating polynomial
% yint = Newtint(x,y,xx): Uses an (n - 1)-order Newton
%   interpolating polynomial based on n data points (x, y)
%   to determine a value of the dependent variable (yint)
%   at a given value of the independent variable, xx.
% input:
%   x = independent variable
%   y = dependent variable
%   xx = value of independent variable at which
%        interpolation is calculated
% output:
%   yint = interpolated value of dependent variable

% compute the finite divided differences in the form of a
% difference table
n = length(x);
if length(y)~=n, error('x and y must be same length'); end
b = zeros(n,n);
% assign dependent variables to the first column of b.
b(:,1) = y(:);  % the (:) ensures that y is a column vector.
for j = 2:n
  for i = 1:n-j+1
    b(i,j) = (b(i+1,j-1)-b(i,j-1))/(x(i+j-1)-x(i));
  end
end
% use the finite divided differences to interpolate
xt = 1;
yint = b(1,1);
for j = 1:n-1
  xt = xt*(xx-x(j));
  yint = yint+b(1,j+1)*xt;
end
```

Use $b_1..b_n$ to predict interpolating values!

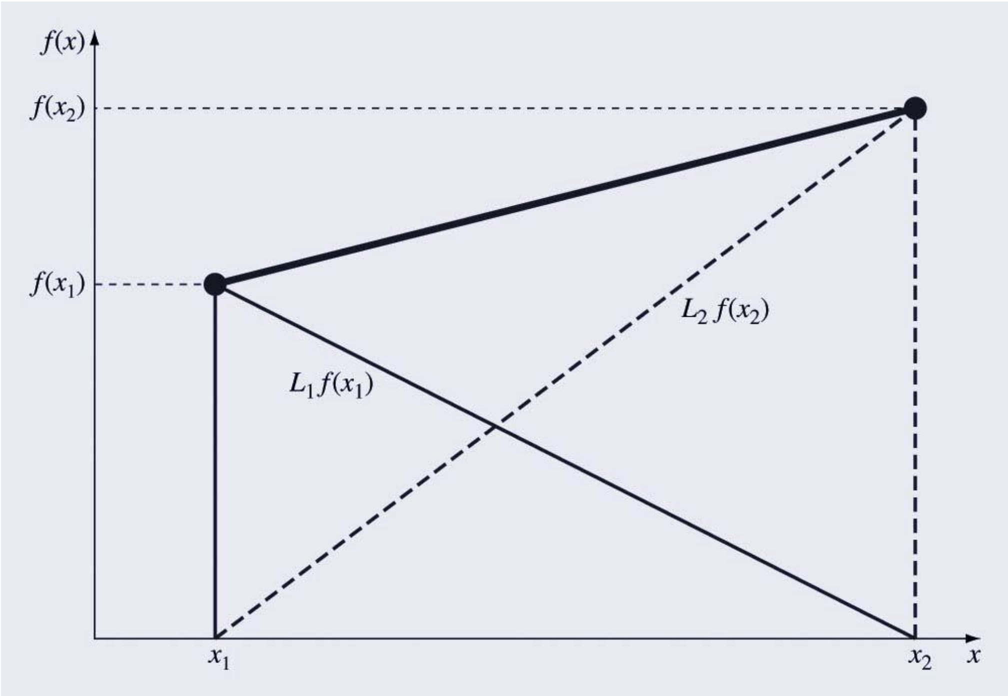
j

i

x_i	$f(x_i)$	First	Second	Third
x_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	$f[x_4, x_3, x_2, x_1]$
x_2	$f(x_2)$	$f[x_3, x_2]$	$f[x_4, x_3, x_2]$	
x_3	$f(x_3)$	$f[x_4, x_3]$		
x_4	$f(x_4)$			

Lagrange Interpolating Polynomials

- The first-order Lagrange interpolating polynomial may be obtained from **a weighted combination** of **two linear interpolations**, as shown.
- The resulting formula based on known points x_1 and x_2 and the values of the dependent function at those points is:



$$f_1(x) = L_1 f(x_1) + L_2 f(x_2)$$

$$L_1 = \frac{x - x_2}{x_1 - x_2}, L_2 = \frac{x - x_1}{x_2 - x_1}$$

$$\boxed{f_1(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2)}$$

- **No need of unknown coefficients b_n or a_n**
- **Not a simple form, but relatively easy for using computer to calculate**

2nd-order Lagrange Interpolating Polynomials

$$f_2(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} f(x_1) + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} f(x_2) + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} f(x_3)$$

Practice at home!

1

When x = x_1

$$f_2(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} f(x_1) + \cancel{\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}} f(x_2) + \cancel{\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}} f(x_3)$$

Therefore ***$f_2(x) = f(x_1)$***

$\Rightarrow$ (x1, f(x1)) indeed is a point on this polynomial!

Similarly, when $x = x_2$ and $x = x_3$, $f_2(x) = f(x_2)$ and $f(x_3)$, respectively. The resulting Lagrange polynomial indeed **will pass through all 3 data points**!

Can you write a 3rd-order Lagrange polynomial?

Lagrange Interpolating Polynomials

- In general, the Lagrange polynomial interpolation for *n* points is:

$$f_{n-1}(x_i)=\sum_{i=1}^{n} L_i(x)f(x_i)$$

where L_i is given by:

L_i are the weighting coefficients.

$$L_i(x)=\prod_{\substack{j=1\\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}$$

π is capital pi notation, the product of….

j* cannot be equal to *i

MATLAB Implementation (p. 419)

```
function yint = Lagrange(x,y,xx)
% Lagrange: Lagrange interpolating polynomial
%   yint = Lagrange(x,y,xx): Uses an (n - 1)-order
%     Lagrange interpolating polynomial based on n data points
%     to determine a value of the dependent variable (yint) at
%     a given value of the independent variable, xx.
% input:
%   x = independent variable
%   y = dependent variable
%   xx = value of independent variable at which the
%        interpolation is calculated
% output:
%   yint = interpolated value of dependent variable

n = length(x);
if length(y)~=n, error('x and y must be same length'); end
s = 0;
for i = 1:n
  product = y(i);
  for j = 1:n
    if i ~= j
      product = product*(xx-x(j))/(x(i)-x(j));
    end
  end
  s = s+product;
end
yint = s;
```

Capital π calculation

Where are coefficients?

No we are not solving for coefficients. We know the polynomial already.

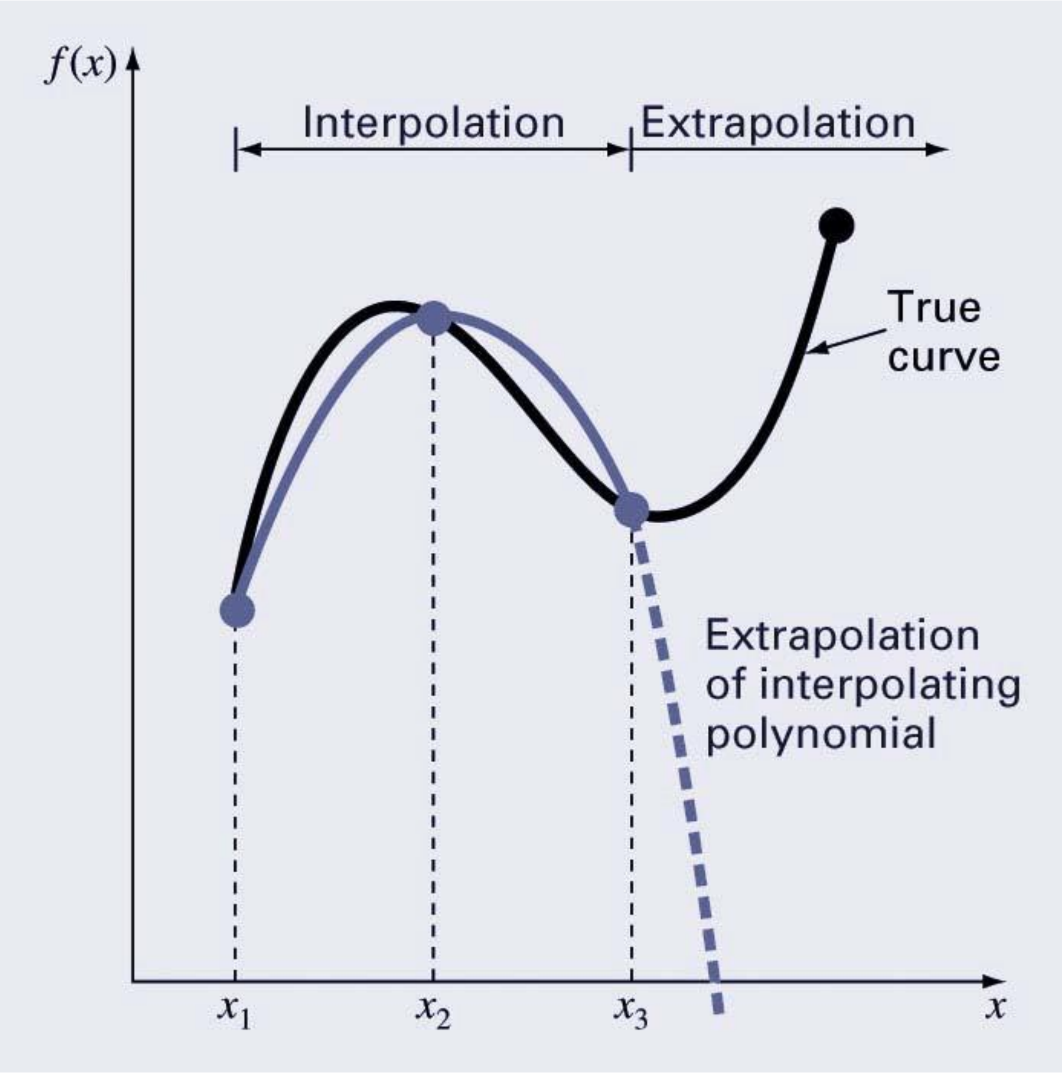
We just directly plug in x to find the associated f(x).

Inverse Interpolation

- Interpolation means finding some value ***f*(*x*) for some *x*** that is between given independent data points.
- Sometimes, it will be useful to **find the *x*** for which ***f*(*x*) is a certain value** - this is ***inverse interpolation***.
- Rather than finding an interpolation of x as a function of $f(x)$, it may be useful to find an equation for $f(x)$ as a function of x using interpolation and then solve the **corresponding roots problem**:
  $f(x)-f_{\text{desired}}=0$ for x.

Extrapolation

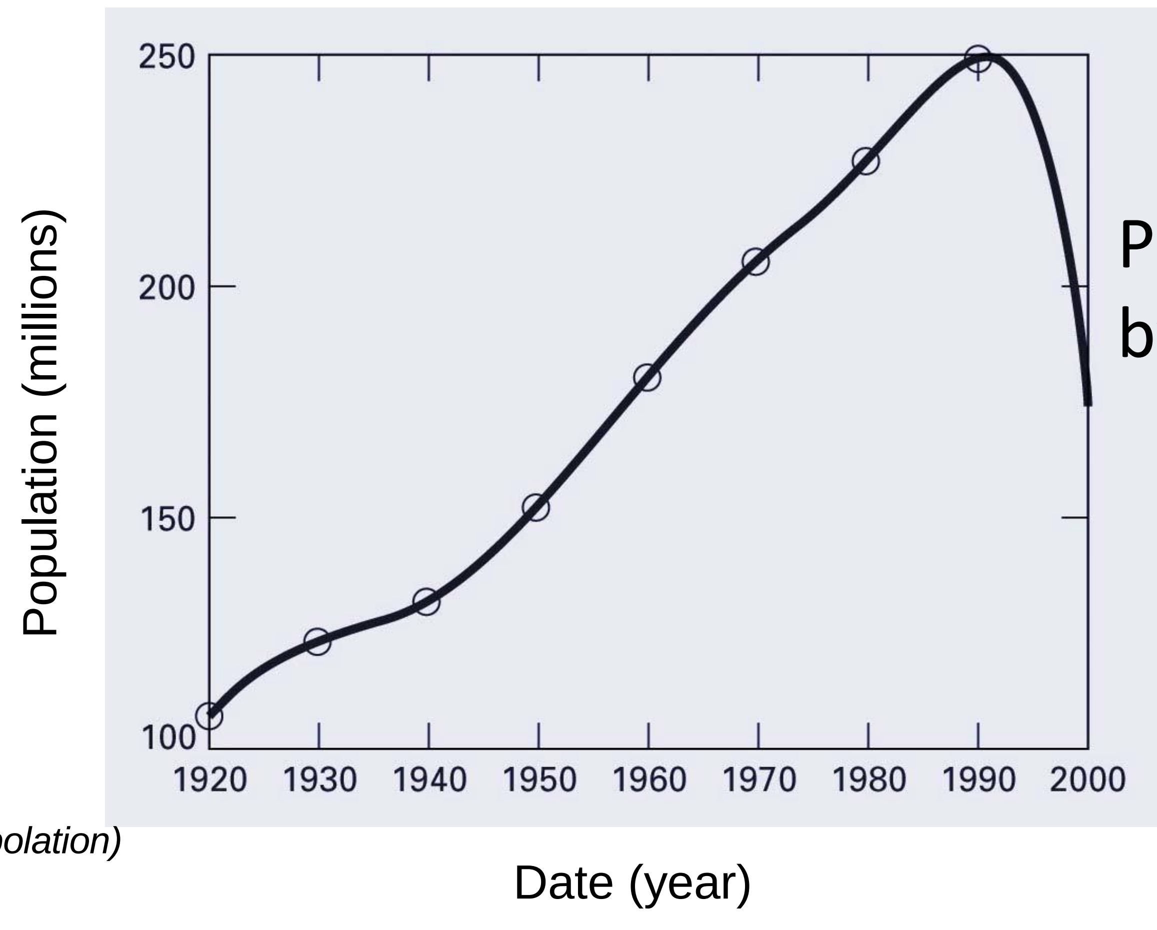
- *Extrapolation* is the process of estimating a value of $f(x)$ that **lies outside the range** of the known data points x_1, x_2, …, x_n.
- Extrapolation represents a **step into the unknown**, and extreme care should be exercised when extrapolating!



Extrapolation Hazards (p. 422)

- The following shows the results of extrapolating a 7th order population data set:

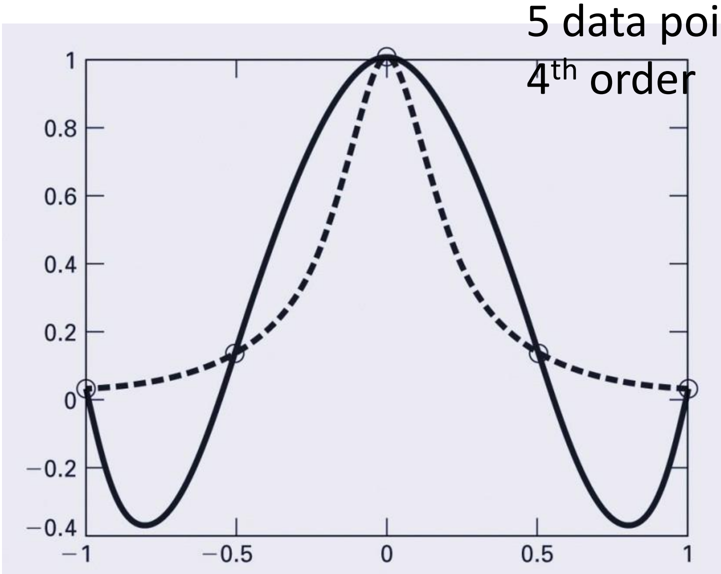
Date	1920	1930	1940	1950	1960	1970	1980	1990	2000
Population	106.46	123.08	132.12	152.27	180.67	205.05	227.23	249.46	281.42

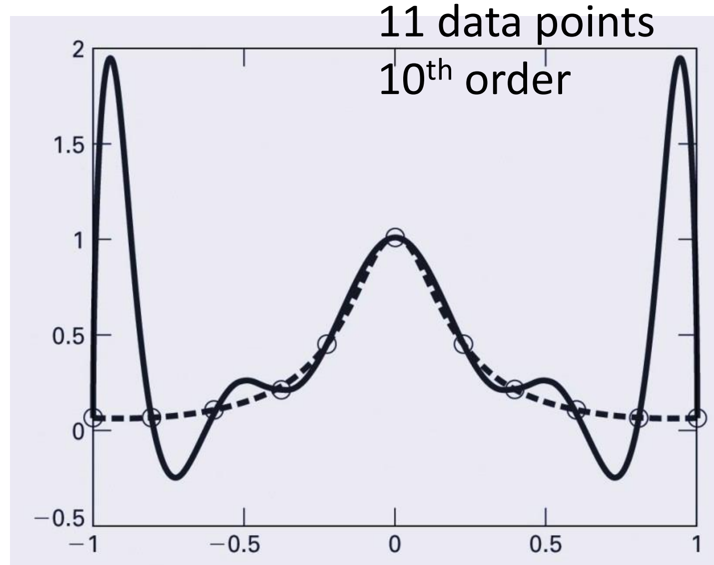


Polynomial found by ***polyfit***

Oscillations

- Higher-order polynomials can not only lead to **round-off errors** due to **ill-conditioning**, but can also introduce **oscillations** to an interpolation.
- In the figures below, the dashed line represents a function (this case called **Runge's function**), the circles represent samples of the function, and the **solid line represents the results of a polynomial interpolation**





Is "more" always better?

Example

During the night and early morning, the **body temperature of a hospital patient** rose dramatically, for an as-yet-unknown reason, until the nurse detected this variation and administered medication. You need to interpret this change in temperature. As the first step, you must determine exactly when the patient's body reached its **maximum temperature** and what was the value of this temperature. The body temperature was recorded by a computer at one-hour intervals (see table below).

Time (a.m.)	Temperature (ºF)	Time (a.m.)	Temperature (ºF)
1	98.9	7	104.0
2	99.5	8	104.1
3	99.9	9	102.5
4	101.3	10	101.2
5	101.6	11	100.5
6	102.5	12	100.2

Example

```
% Interpolation of the time-temperature data using
% polynomial and
% Gregory-Newton forward interpolation formula
% to find the maximum temperature and
% the time this maximum occurred.

clc; clear all; close all;

% Input data

time = 1:12;
temp = [98.9 99.5 99.9 101.3 101.6 102.5 104.0 104.1
102.5 101.2 100.5 100.2];
plot(time, temp, 'o')
    title('Patient''s Temperature Profile')
    xlabel('Time (a.m.)');
    ylabel('Temperature (deg F)')
pause
```

Example

```
% Polynomial fit
p=polyfit(time,temp,2)     % 2-nd order polynomial
temp_p=polyval(p,time);
figure(1); plot(time, temp, 'o', time, temp_p);
title('Patient''s Temp… Profile: 2-nd order polyfit')
    xlabel('Time (a.m.)'); ylabel('Temperature (deg F)')
pause

for i=3:max(size(time)-1)+5
      p=polyfit(time,temp,i)
      temp_p=polyval(p,time);
figure(1); plot(time, temp, 'o', time, temp_p);
title(['Patient''s Temp… : polyfit n=', num2str(i)])
    xlabel('Time (a.m.)'); ylabel('Temperature (deg F)')
    pause
end
```

Example

```
% Vector of time for interpolation
ti = linspace(min(time),max(time));
redo = 1;
while redo
   disp(' '); n = input(' Order of interpolation = ');
   te = gregory_newton(time,temp,ti,n); % Interpolation
   [max_temp,k] = max(te);
   max_time = ti(k);
% Show the results
   fprintf('\n Maximum temperature of %4.1f F reached at
%4.2f.\n',max_temp,max_time)
% Show the results graphically
   figure(2); plot(time,temp,'o',ti,te)
   title('Patient''s Temperature Profile')
   xlabel('Time (a.m.)'); ylabel('Temperature (deg F)')
   axis([1 12 98 105])
   disp(' ')
   redo = input(' Repeat the calculation (1/0) : ');
end
```