计算物理

Lecture 7

傅子文

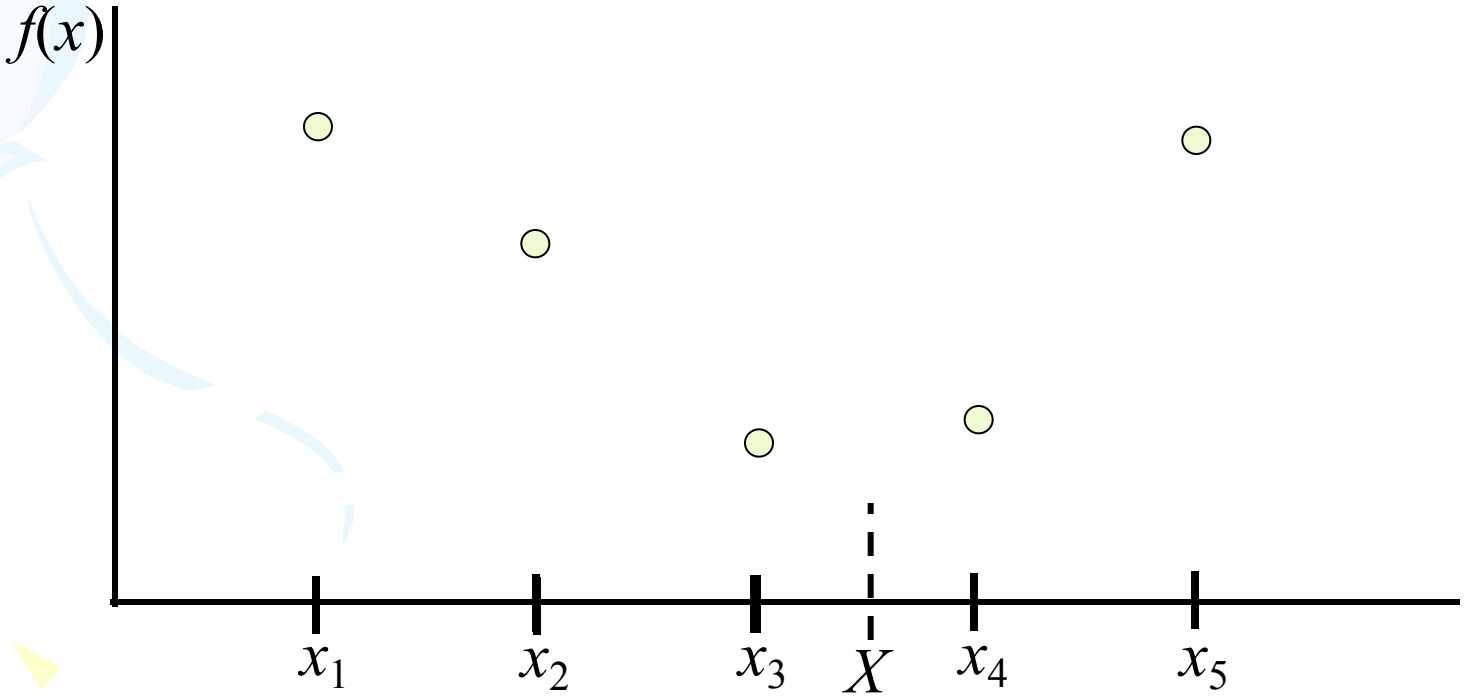
fuziwen@scu.edu.cn

Today's Lecture

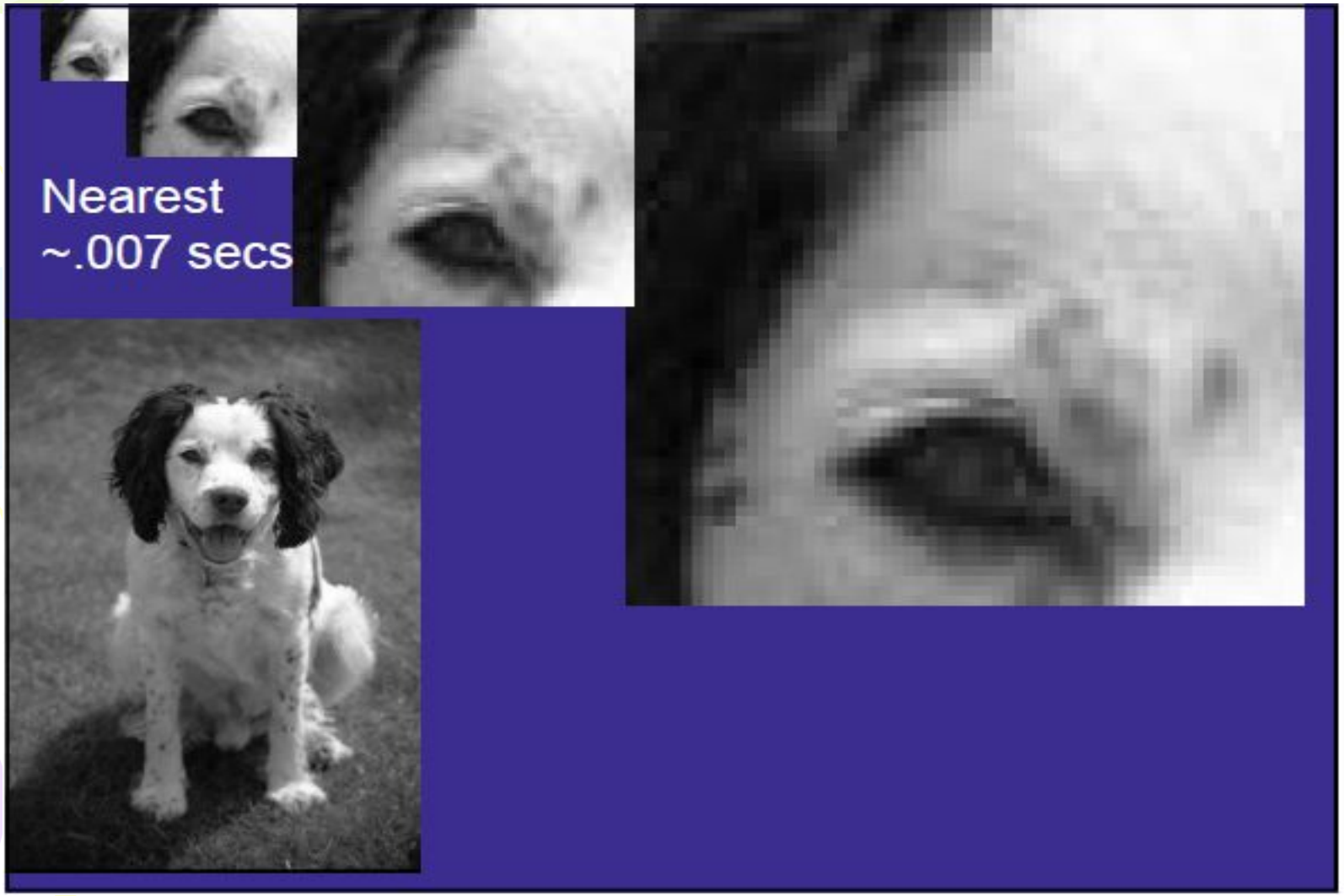
- Interpolation & Approximation I
  - Overview of ideas
    - Interpolation *vs* fitting
  - Polynomial interpolation
    - Lagrange interpolation
    - Hermite interpolation

General idea of interpolation

- Suppose we are given data at discrete points only, and we wish to estimate values between these known points
- Interpolation is the process of estimating unknown values that fall between known values.

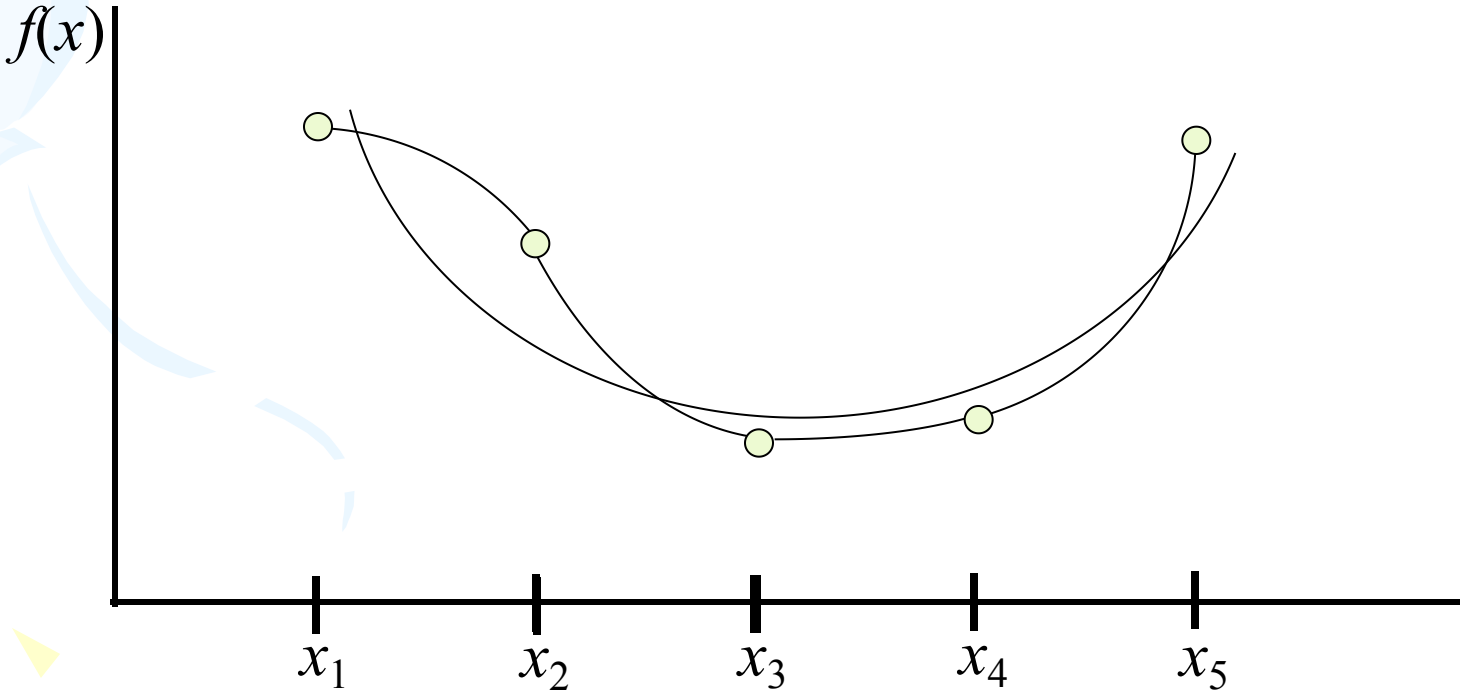


When we enlarge an image, we need values for the new pixels.



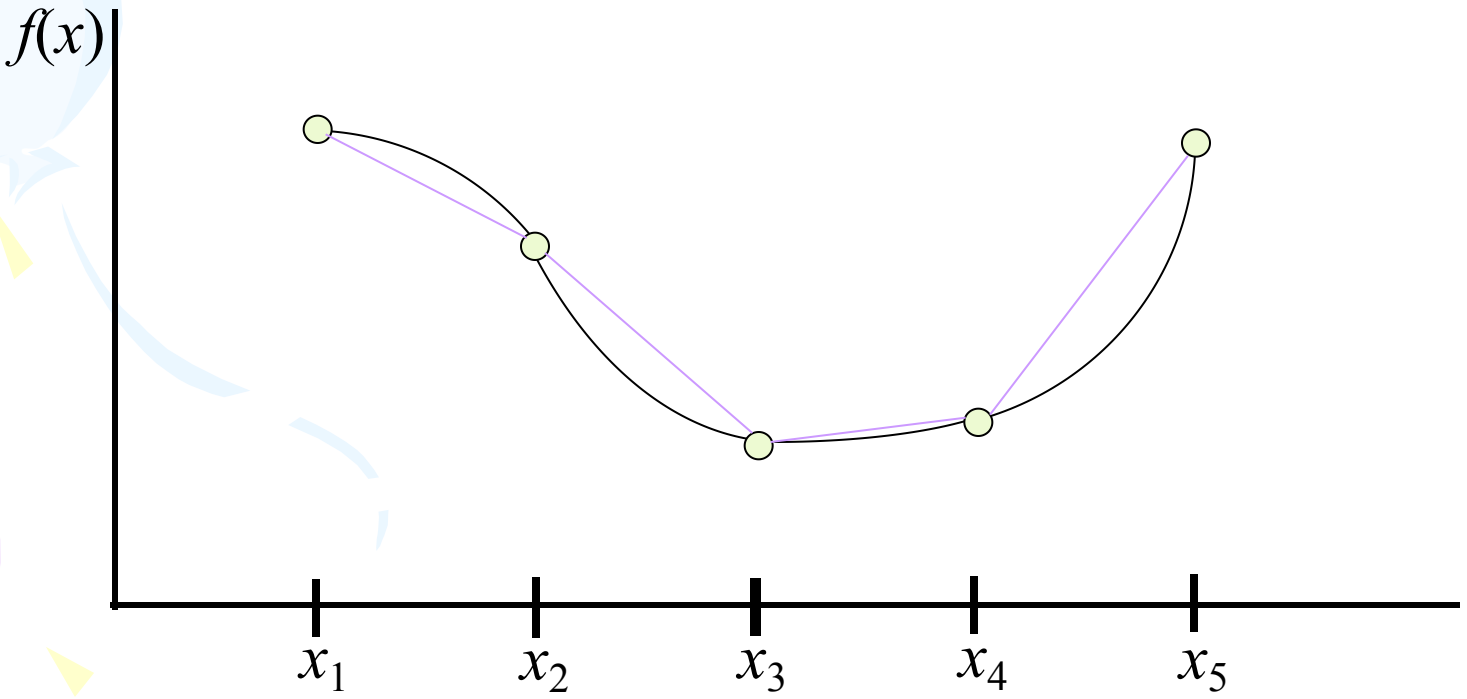
Difference between interpolation&fitting

- Interpolation passes through each datum
- Fitting seeks to produce a curve that approximates the data



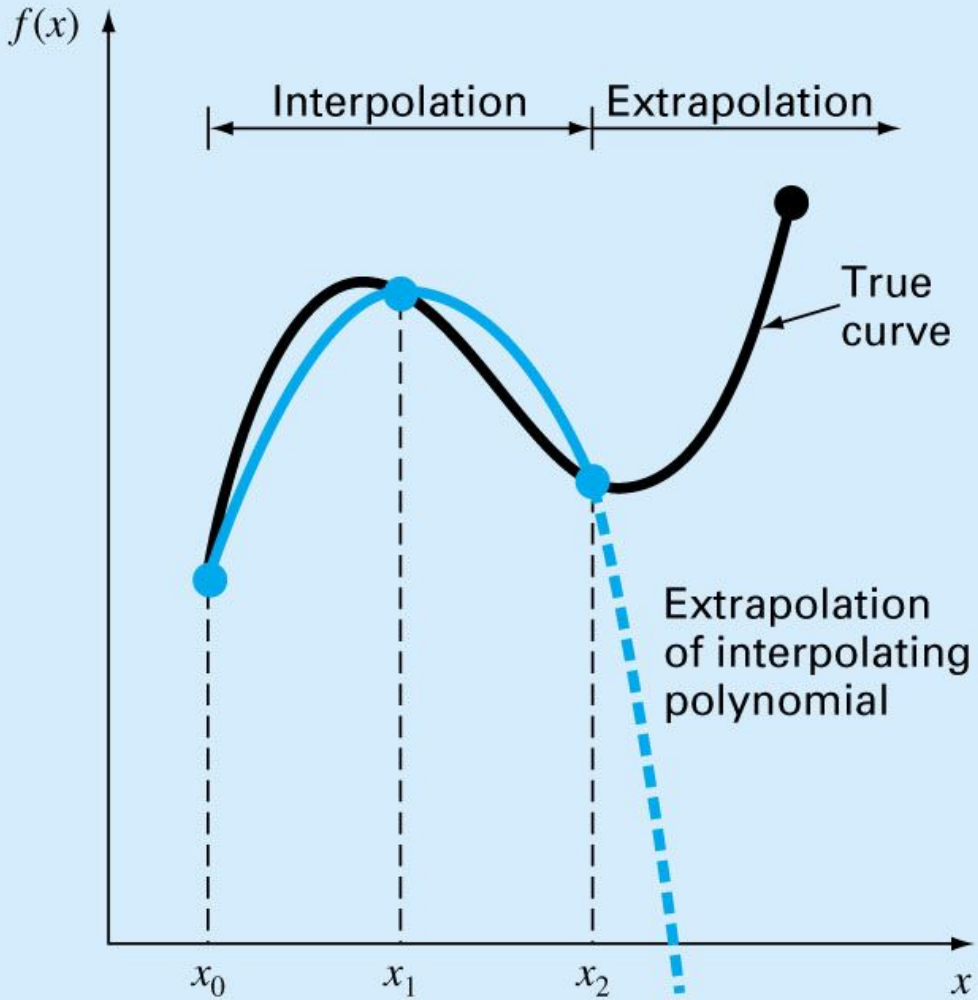
Global versus piecewise interpolation

- We can fit a single polynomial of degree n to n+1 data points
- Alternatively we can use a series of polynomials to link points together – frequently called *splines*
  - *Splines will usually be cubic polynomials – I've used linear here to emphasize the piecewise nature*



Extrapolation versus interpolation

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



Lagrange interpolation

- To fit a polynomial of degree n we need n+1 points
- Consider first interpolating between two points, $f(x_2), f(x_3)$ using a function $F(x)$
- The Taylor expansion about a point x gives

$$-f(x_2) = F(x) + (x_2 - x)F'(x) +$$
$$-f(x_3) = F(x) + (x_3 - x)F'(x) +$$

- If we use a polynomial of degree 1, $p(x)$, then the Taylor series truncates after $p'(x)$

$$-f(x_2) = p(x) + (x_2 - x)p'(x) \qquad \text{------(1)}$$
$$-f(x_3) = p(x) + (x_3 - x)p'(x) \qquad \text{------(2)}$$

Eliminate $p'(x)$

- After a short derivation (exercise) we find that eqns (1)&(2) give

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

- On varying the expansion point x, but keeping fixed end points x_2, x_3 this corresponds to a straightline – as expected

(check the following for yourself):

– If $x = x_3 \rightarrow p(x) = f(x_3)$

– If $x = x_2 \rightarrow p(x) = f(x_2)$

- Linear fits aren't that helpful though, and bringing in more points allows us to up the order of the interpolation

Quadratic fit using three points

- If we again Taylor expand put assume the function is a polymial, $p(x)$ of degree 2, then

$$f(x_1) = p(x) + (x_1 - x)p'(x) + \frac{1}{2}(x_1 - x)^2 p''(x) \qquad (3)$$

$$f(x_2) = p(x) + (x_2 - x)p'(x) + \frac{1}{2}(x_2 - x)^2 p''(x) \qquad (4)$$

$$f(x_3) = p(x) + (x_3 - x)p'(x) + \frac{1}{2}(x_3 - x)^2 p''(x) \qquad (5)$$

- With a bit more algebra (fairly lengthy) we can eliminate both using (3),(4),(5) to get

$$p'(x) \text{ and } p''(x)$$

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

You should see a pattern emerging here…

General (Lagrange) Polynomial fit

- If we have n points, a polynomial of degree n-1 can be constructed using the formula

$$p(x) = \sum_{j=1}^{n} l_{j,n}(x) f(x_j)$$

where

$$l_{j,n}(x) = \frac{(x-x_1)(x-x_2)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_1)(x_j-x_2)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}$$

- This is called the interpolation polynomial ***in Lagrange form,*** but it is frequently referred to as the Lagrange polynomial

Properties of the $l_{j,n}(x)$

- The $l_{j,n}(x)$ are clearly polynomials, and a called *basis polynomials* since they sum to find the interpolating polynomial

- If we let $x=x_i$ in the formula (so that we are picking one of the interpolation points) then

$$l_{j,n}(x_i) = \frac{(x_i - x_1)(x_i - x_1)...(x_i - x_{j-1})(x_i - x_{j+1})...(x_i - x_n)}{(x_j - x_1)(x_j - x_1)...(x_j - x_{j-1})(x_j - x_{j+1})...(x_j - x_n)}$$

$= 1$ if $i = j$ (since the numerator & denominator are equal)

$= 0$ if $i \neq j$ (since one of numerator terms must be zero)

Thus

$l_{j,n}(x_i) = \delta_{ij}$ where δ_{ij} is the Kronecker delta, and hence

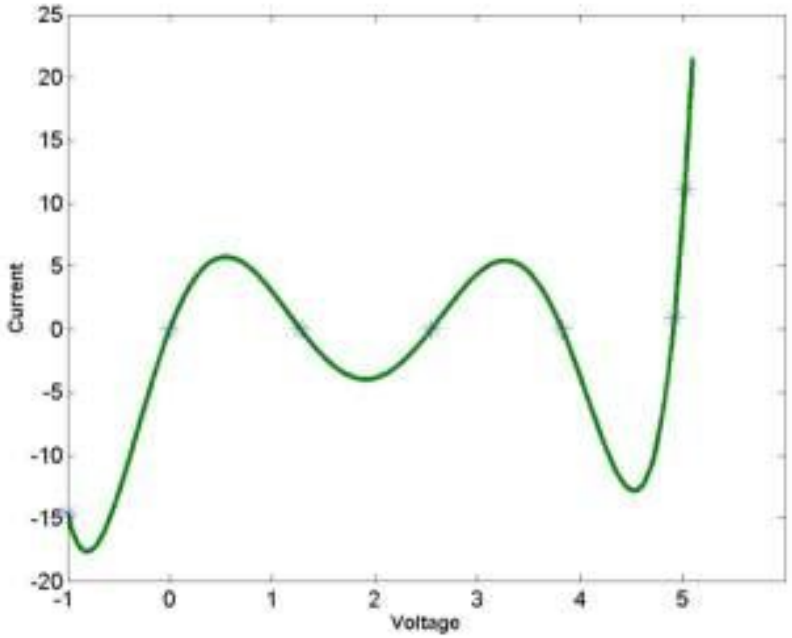
$p(x_i) = \sum_{j=1}^{n} \delta_{ij} f(x_j) = f(x_i)$ which is what we would expect.

Problems with polynomial interpolation

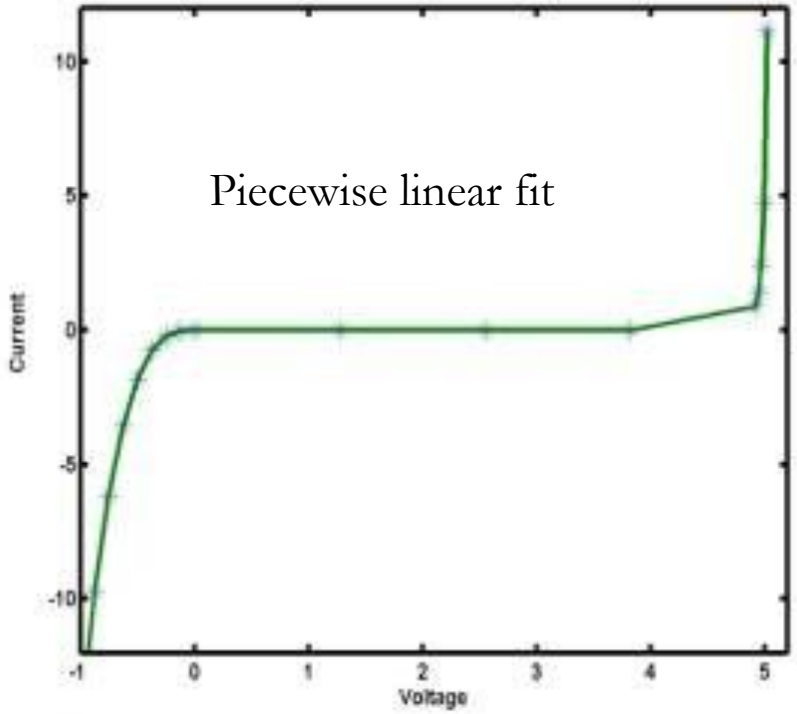
- As the number of points increases so does the degree of the polynomial
  - This implies many turns and a lot of "structure" between interpolation points in high order polynomials
  - High order polynomials will have a lot of "wiggles" and if points do not change smoothly they can "overshoot" in between datums
- Cubic fits tend to be about optimal
  - 4 data points can surround x symmetrically (2 on each side)
- Increasing the number of points adds more data from further away from a given interpolation point
  - This doesn't necessarily improve the accuracy of the fit locally

Example of "wiggles"

Voltage	*-1.00*	*0.00*	*1.27*	*2.55*	*3.82*	*4.92*	*5.02*
Current	*-14.58*	*0.00*	*0.00*	*0.00*	*0.00*	*0.88*	*11.17*



6th order polynomial fit



From http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes/Lecture3_3.htm

Segue - numerically implementing the general formula

- Implementing the general formula is not exactly trivial
- It helps to notice that the $l_{j,n}(x)$ can be written as a product

$$l_{j,n}(x) = \prod_{i=1, i\neq j}^{n} \frac{x - x_i}{x_j - x_i} = \frac{(x-x_1)(x-x_2)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_1)(x_j-x_2)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}$$

- One can then think about using a loop to do the multiplication of the terms in the product
- However, the best way to calculate large interpolation polynomials is to use *Neville's Algorithm*

Segue - Neville's Algorithm

- We won't go into the details, however you can take a look (if you are interested) at *Numerical Recipes section 3.1*

- Neville's algorithm is better because it builds up the interpolation polynomial in a recursive fashion
  - It uses a similar table idea to building up coefficients in a binomial expansion

- The employed recursion cuts down the total number of operations required and helps reduce concerns about rounding error

Advantages of Neville's Algorithm

1. *Numerically Stable*
   - –Uses the Given Data Directly
   - –No Need to Represent the Polynomial in the Basis 1 ,t,t^2 ,…
2. *Fast*
   - –Dynamic Programming
   - –$O(n^2)$ vs. $O(2^n)$
3. *Simple Structure*
   - –Parallel Property
   - –Strip off First and Last Indices
4. *Easy to Update*
   - –Add a Computation of $O(n)$ Instead of Redoing Work of $O(n^2)$.

A concrete example

When it is expensive or difficult to evaluate a function $f(x)$ at an arbitrary value of x, we might consider, instead, interpolating from a table of values. Consider the exponential integral,

$$Ei(x) = \int_{-\infty}^{x} \frac{e^{-t}}{t} dt$$

tabulated below for small x .

x	Ei(x)
0.1	-1.6228
0.2	-0.8218
0.3	-0.3027
0.4	0.1048
0.5	0.4542

- This function diverges as $\log(x)$ at $x=0$.

$$Ei(x) = \int_{-\infty}^{x} \frac{e^{-t}}{t} dt$$

- To evaluate it requires doing the integral, or summing a series, so it is somewhat expensive.

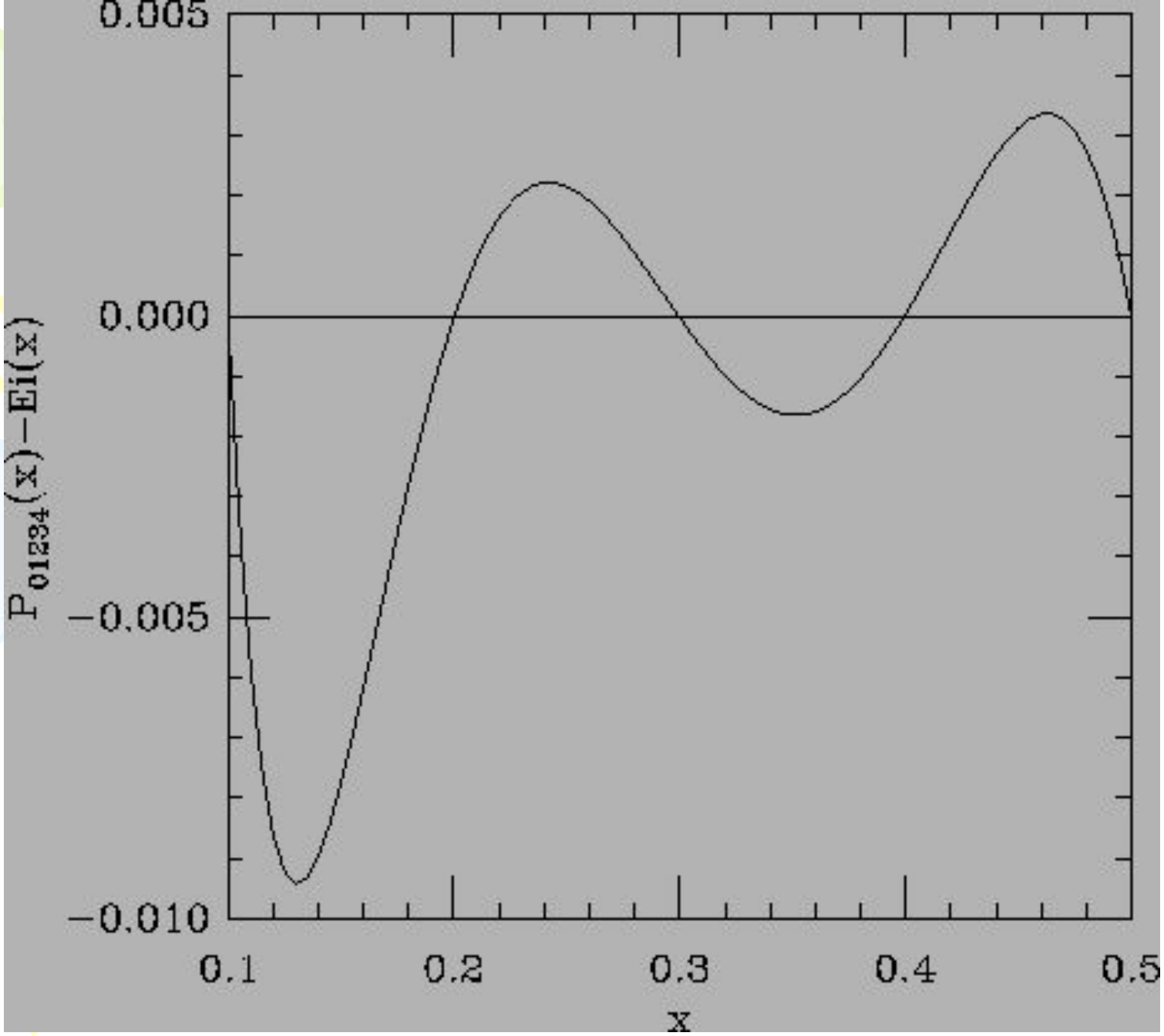
- Suppose we want the value of Ei(0.15) .
  - A very crude approximation chooses either of the two nearby values Ei(0.1)=-1.6228 or Ei(0.2)=-0.8218 .
  - A common and somewhat better approach makes a linear interpolation. Since 0.15 is midway between 0.1 and 0.2 the linear interpolation just averages the two values, giving -1.2223 .
  - No numerical result has meaning without an estimate of the error. The exponential integral function we are attempting to interpolate is not a polynomial, so a polynomial representation is bound to be inexact. If we construct the quartic polynomial interpolating all five points in the table above, and evaluate it at 0.15, we get

    $P_{01234}(0.15) = -1.1719$

    whereas the correct value to six decimal digits is Ei(x)=-1.16409. The quartic polynomial is low by about 1%.

As an application of this result, let's write the polynomial interpolating the values at x=0.2,0.3,0.4 in our table.
With three points it must be a quadratic.
The Lagrange construction gives

$$P_{1,2,3} = -0.8218\frac{(x-0.3)(x-0.4)}{(0.2-0.3)(0.2-0.4)} - 0.3027\frac{(x-0.2)(x-0.4)}{(0.3-0.2)(0.3-0.4)}$$
$$+0.1048\frac{(x-0.2)(x-0.3)}{(0.4-0.2)(0.4-0.3)}$$


0.005
0.000
−0.005
−0.010
0.1
0.2
0.3
0.4
0.5
x
P01234(x)−Ei(x)

Constructing the interpolating polynomial is somewhat tedious. If our goal is merely to get the interpolated value, and we don't care to know the coefficients of the polynomial, we may use the Neville algorithm. This algorithm starts from the requested interpolation point x and generates a table of the form

x_0	$P_0(x)$				
		$P_{01}(x)$			
x_1	$P_1(x)$		$P_{012}(x)$		
		$P_{12}(x)$		$P_{0123}(x)$	
x_2	$P_2(x)$		$P_{123}(x)$		$P_{01234}(x)$
		$P_{23}(x)$		$P_{1234}(x)$	
x_3	$P_3(x)$		$P_{234}(x)$		
		$P_{34}(x)$			
x_4	$P_4(x)$				

where $P_{i,\ldots,k}(x)$ is the polynomial interpolating the points $x_i, x_{i+1} \ldots, x_j$, evaluated at the point x. That is to say, the second column comes from interpolating only one point each, meaning that it reproduces trivially the original values: $P_i(x)=y_i$. The third column comes from interpolating pairs of adjacent points. Notice that $P_{0,1}$ is the polynomial we constructed above. The fourth column comes from interpolating three contiguous points. We illustrated P_{123} above. The last column has only one entry and comes from interpolating all the points.

The identity

$$P_{i,\ldots,l}(x) = \frac{(x-x_j)P_{i,\ldots,j-1,j+1,\ldots l} - (x-x_k)P_{i,\ldots,k-1,k+1,\ldots l}}{x_k - x_j}$$

is used to calculate the values in any column from pairs of values in one column to the left. For example the rule gives

$$P_{1,2,3}(x) = \frac{(x-x_3)P_{12}(x) - (x-x_1)P_{23}(x)}{x_1 - x_3}$$

The student should verify that $P_{123}(x_i) = y_i$ for i=1,2,3, as it should.

Here we apply the Neville scheme to the example of the exponential integral, interpolated at x=0.15.

0.1	-1.6228				
		-1.22230			
0.2	-0.8218		-1.18706		
		-1.08135		-1.17642	
0.3	-0.3027		-1.12320		-1.17186
		-0.91395		-1.13992	
0.4	0.1048		-1.02289		
		-0.76870			
0.5	0.4542				

We carry an extra decimal digit in the intermediate calculations to reduce round-off errors.

Compare the two tables. As promised,

- The second column is a copy of the y_i values.
- The third column gives the result of doing a linear interpolation between the two neighboring points and evaluating the resulting function at x=0.15 .
- Notice that only the first value, -1.2223, is really an interpolation.
- The others in the third column are actually linear extrapolations back to 0.15 from interpolations on the other intervals.
- The fourth column gives the result of a quadratic interpolation of the central point on that row and its two neighbors.
- For x=0.15 only the first value in that column is really an interpolation. The others are extrapolations.

There can be two interpolating values in a column. For example, if we had constructed a similar table for interpolation at x=0.25, the first two values in the fourth column would be interpolations, since the polynomials $P_{012}(x)$ and $P_{123}(x)$ both cover the 12 interval. Which value gives the best answer? Polynomial extrapolations can be risky, so clearly we should use only the interpolating values. For the example above, the interpolating values can be read from the first number in each column, starting with the third column: -1.2223,-1.1871,-1.1764,-1.1719. Since we expect the estimate to improve as we increase the degree of the polynomial, at least as long as the degree isn't too high, our best result should be -1.1719, coming from the quartic polynomial interpolating all the points.

It is not always necessary to carry out the Neville iteration to the bitter end. To avoid disasters with high degree polynomials on evenly spaced points near the edges of a table, it is best to do interpolations with roughly the same number of tabulated values on either side of the interpolation point, which may make stopping advisable, before exhausting all the points in a table.

Finally, we need an estimate of the interpolation error in our best result without peeking at the exact value. (If we know it, why interpolate?) A crude estimate of the error is given by the magnitude of the change between the best value and the next best. In our case it is |-1.1719+1.1764|, or about 0.005 . How far off are we, really? The actual value to six figures is -1.16409 , so the error in the last value is really 0.008, nearly twice our estimated error.
Well, knowing what we do about the logarithmic singularity in this function at the origin and the hazards of high-degree polynomial interpolations, we might have expected trouble. The moral of the story: make your error estimates with a good measure of humility.

As we add one more point to the list of interpolated points, which tells us how to compute the value of the new interpolating polynomial. The matrix structure of the interpolating table suggests introducing a matrix $Q_{i,j}$ to hold the i-th row and j-th column in the format

x_0	$Q_{0,0}$				
		$Q_{1,1}$			
x_1	$Q_{1,0}$		$Q_{2,2}$		
		$Q_{2,1}$		$Q_{3,3}$	
x_2	$Q_{2,0}$		$Q_{3,2}$		$Q_{4,4}$
		$Q_{3,1}$		$Q_{4,3}$	
x_3	$Q_{3,0}$		$Q_{4,2}$		
		$Q_{4,1}$			
x_4	$Q_{4.0}$				

so that $Q_{i,j} = P_{i-j,\mathrm{i}-j+1,\ \ .i}$. Converting Eq (1) to the Q's gives

$$Q_{i,j}(x) = \frac{(x - x_{i-j})Q_{i,j-1}(x) - (x - x_i)Q_{i-1,j-1}(x)}{x_i - x_{i-j}}$$

for $i=j,\ldots n$-1 for each j=1,2,… and with the initial values $Q_{i,0}=y_i$. We leave the construction of the pseudocode and the computer code as an exercise. With care the algorithm can be designed so it keeps only one column of the matrix, saving computer memory. The trick is to run the inner loop over i backwards so the fresh values for the j-th column do not overwrite values still needed from the j-1-st column

Runge's phenomenon

$$f(x) = \frac{1}{1+25x^2}$$

Table : Six equidistantly spaced points in [-1, 1]

x	$y = \frac{1}{1+25x^2}$
-1.0	0.038461
-0.6	0.1
-0.2	0.5
0.2	0.5
0.6	0.1
1.0	0.038461

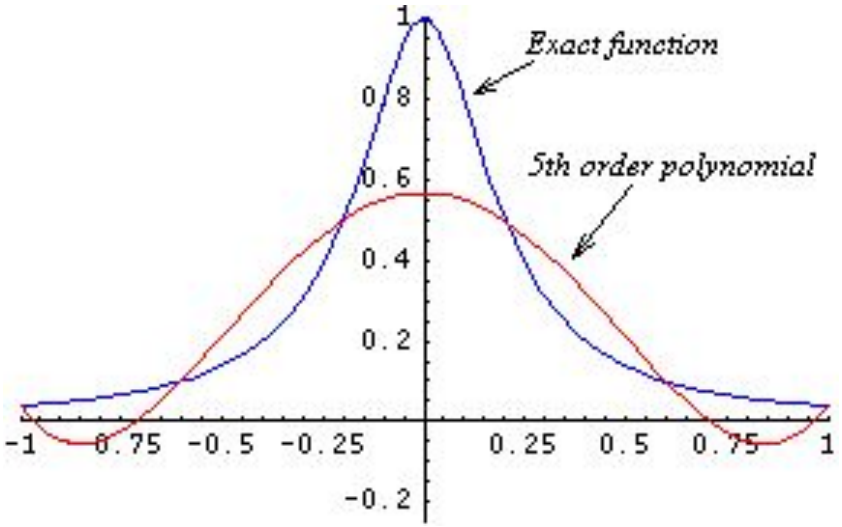


Figure : 5th order polynomial vs. exact function

$$f(x) = \frac{1}{1+25x^2}$$

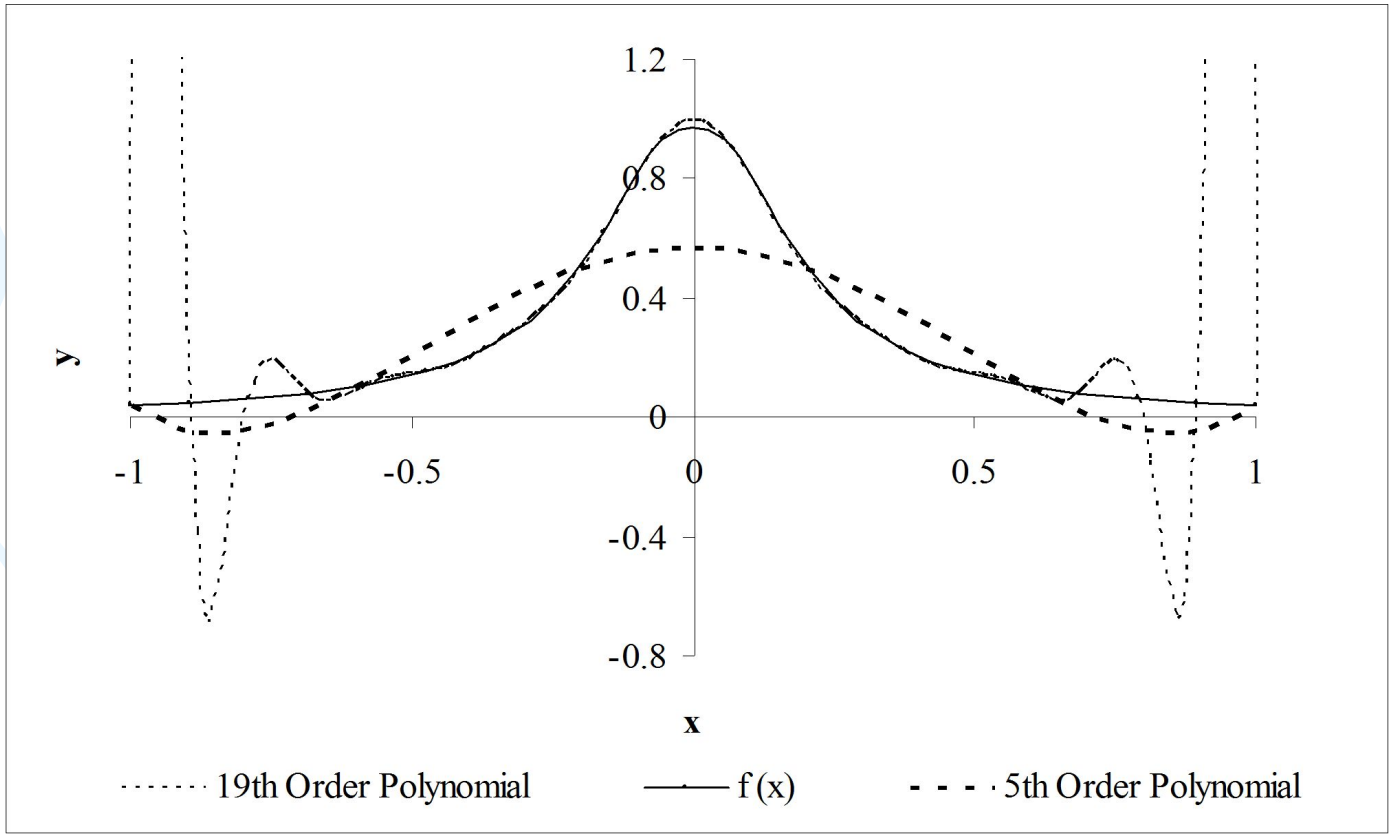
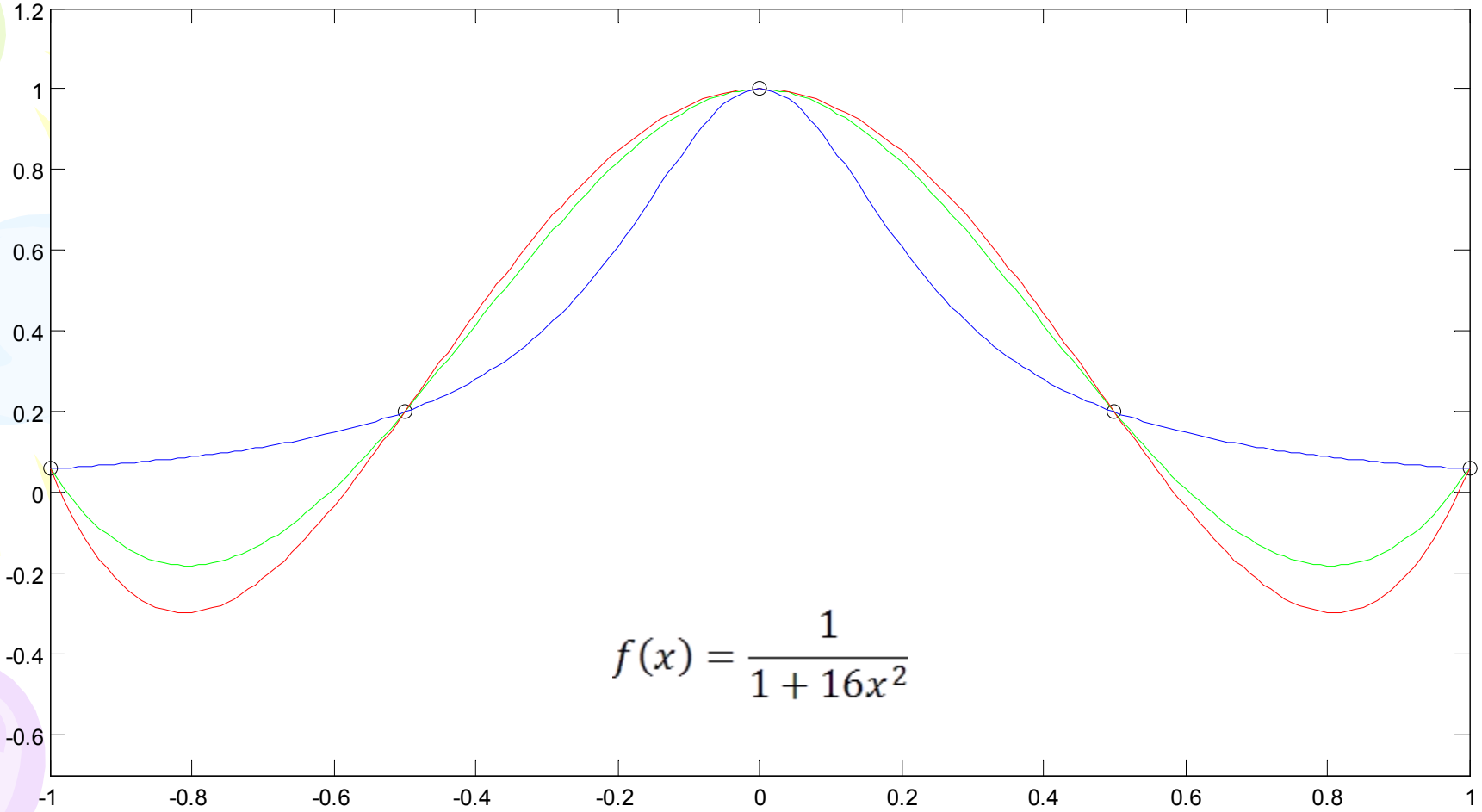


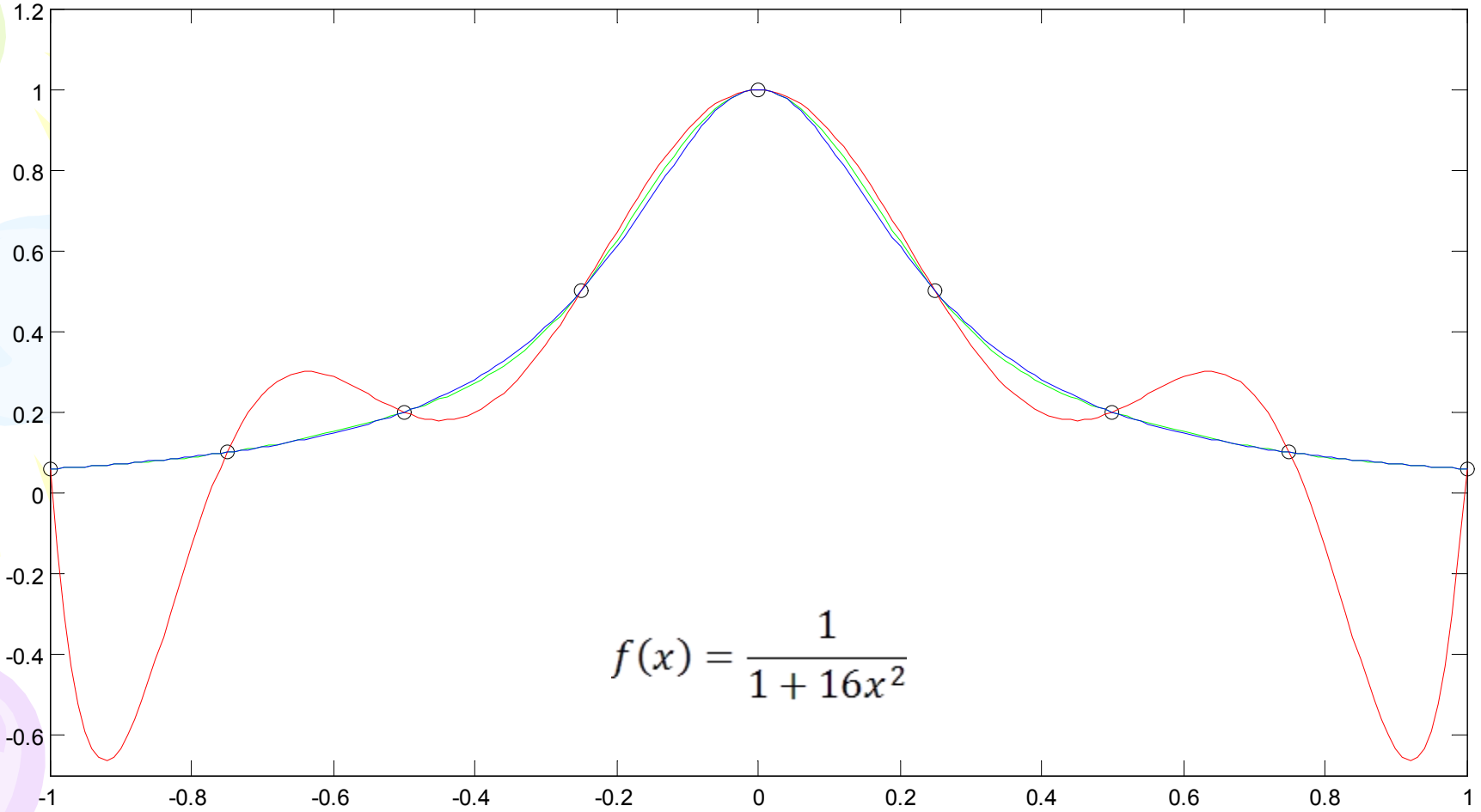
Figure : Higher order polynomial interpolation is a bad idea

Runge Phenomenon: 5 nodes



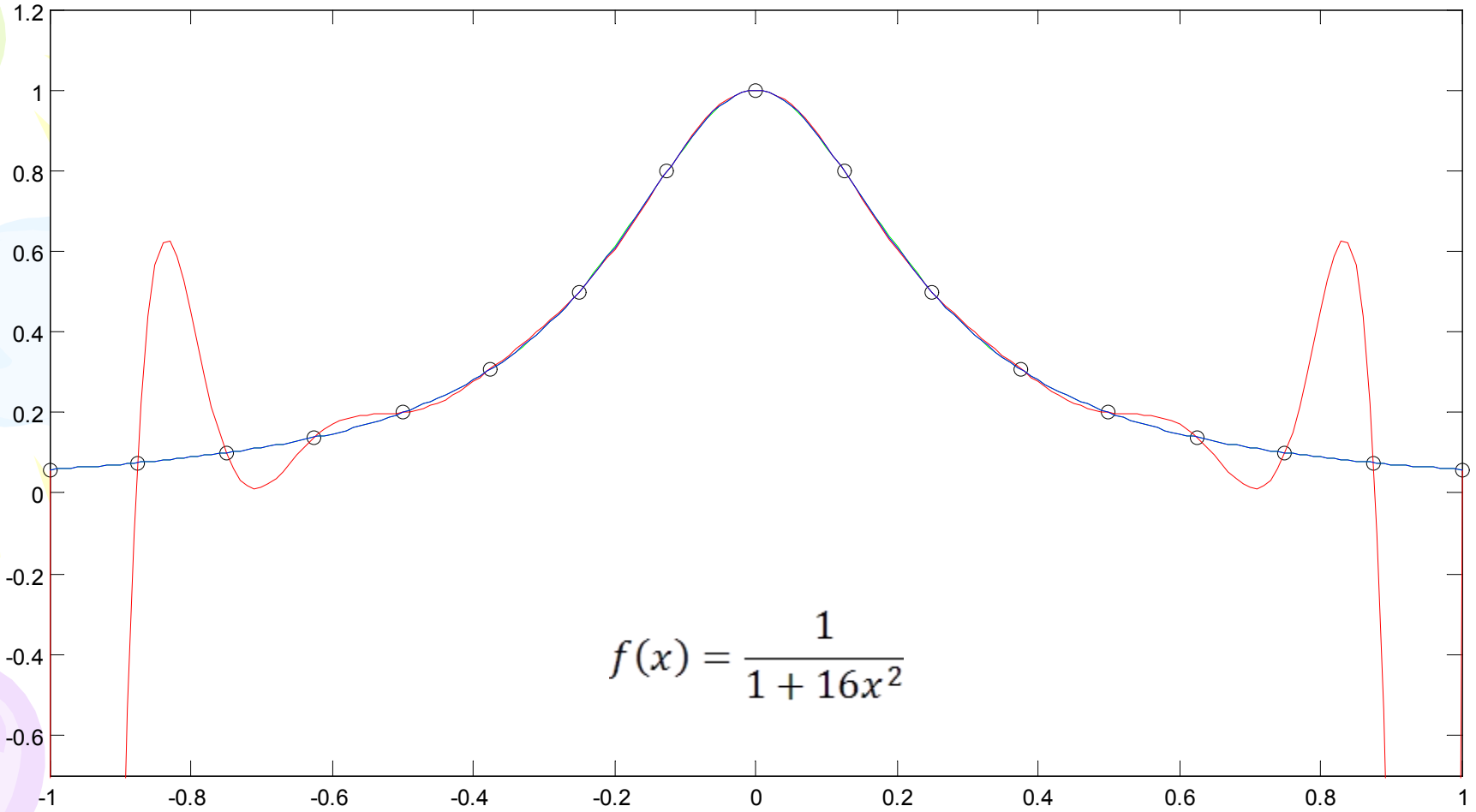
$$f(x) = \frac{1}{1 + 16x^2}$$

Runge Phenomenon: 9 nodes



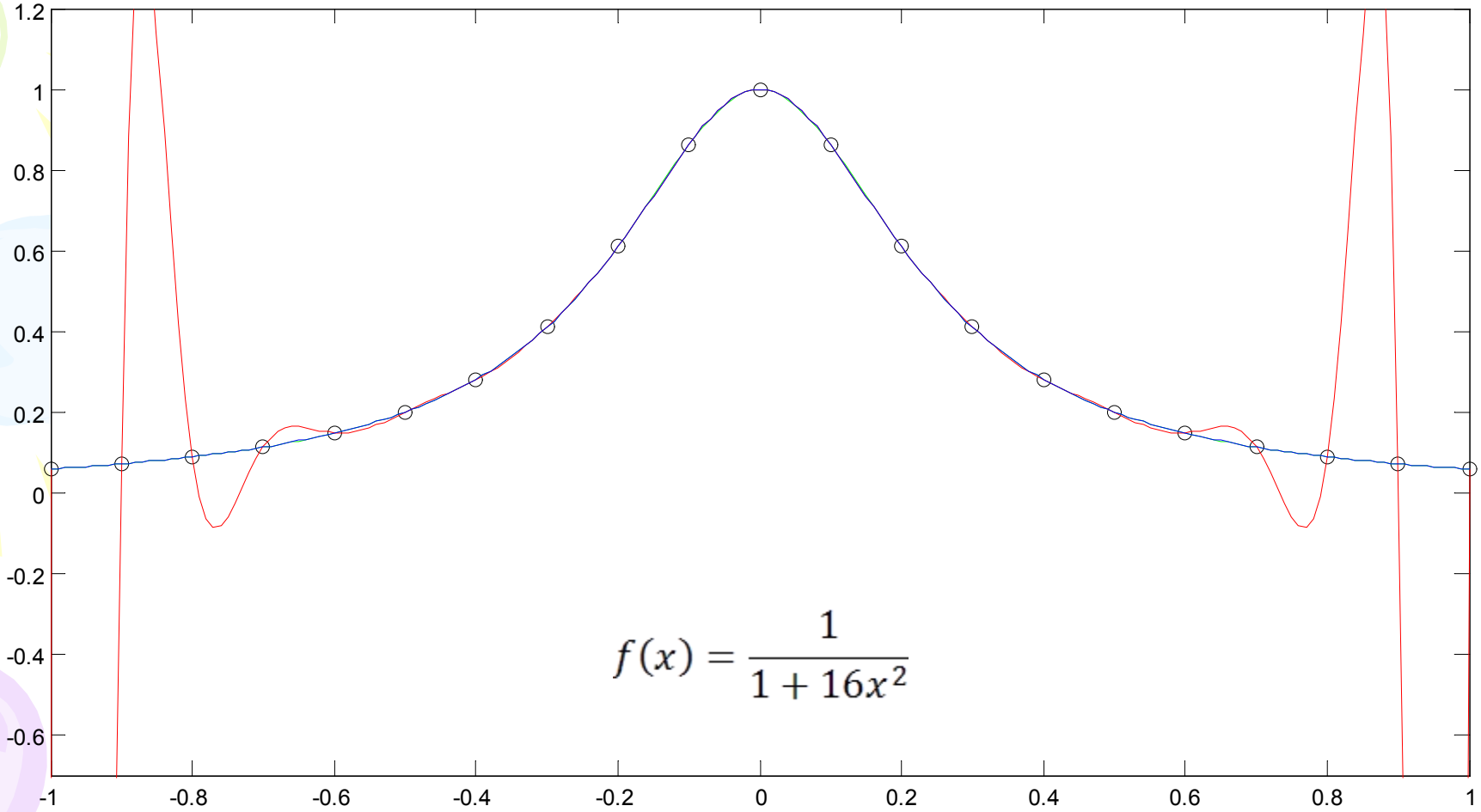
$$f(x) = \frac{1}{1 + 16x^2}$$

Runge Phenomenon: 17 nodes



$$f(x) = \frac{1}{1+16x^2}$$

Runge Phenomenon: 21 nodes

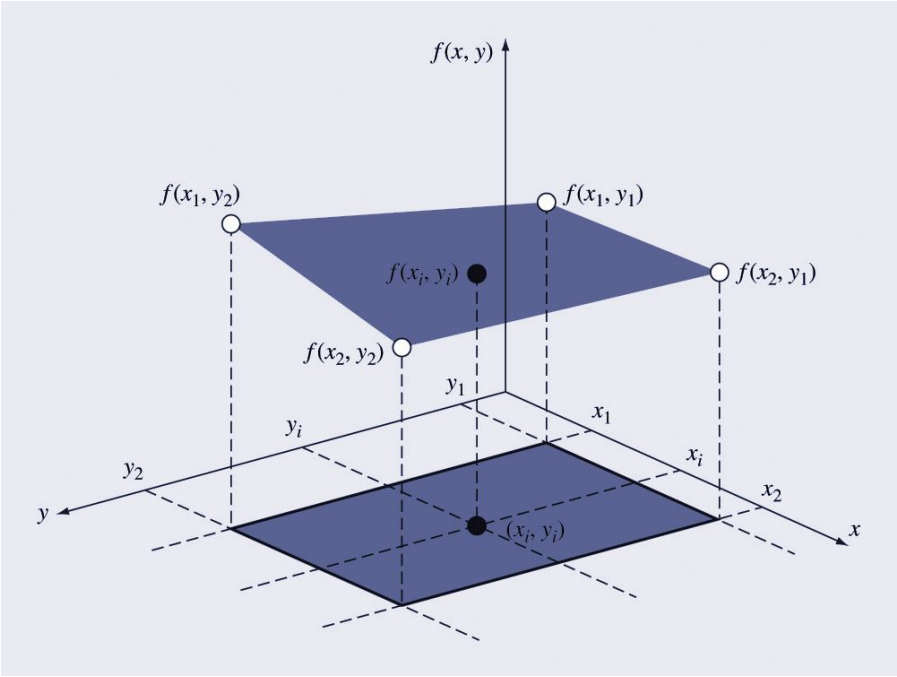


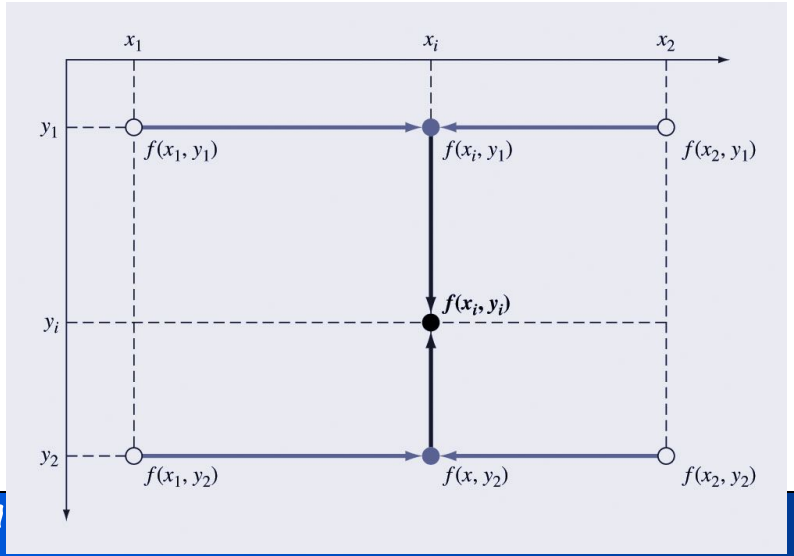
$$f(x) = \frac{1}{1+16x^2}$$

Multidimensional Interpolation

- The interpolation methods for one-dimensional problems can be extended to multidimensional interpolation.
- Example - bilinear interpolation using Lagrange-form equations:

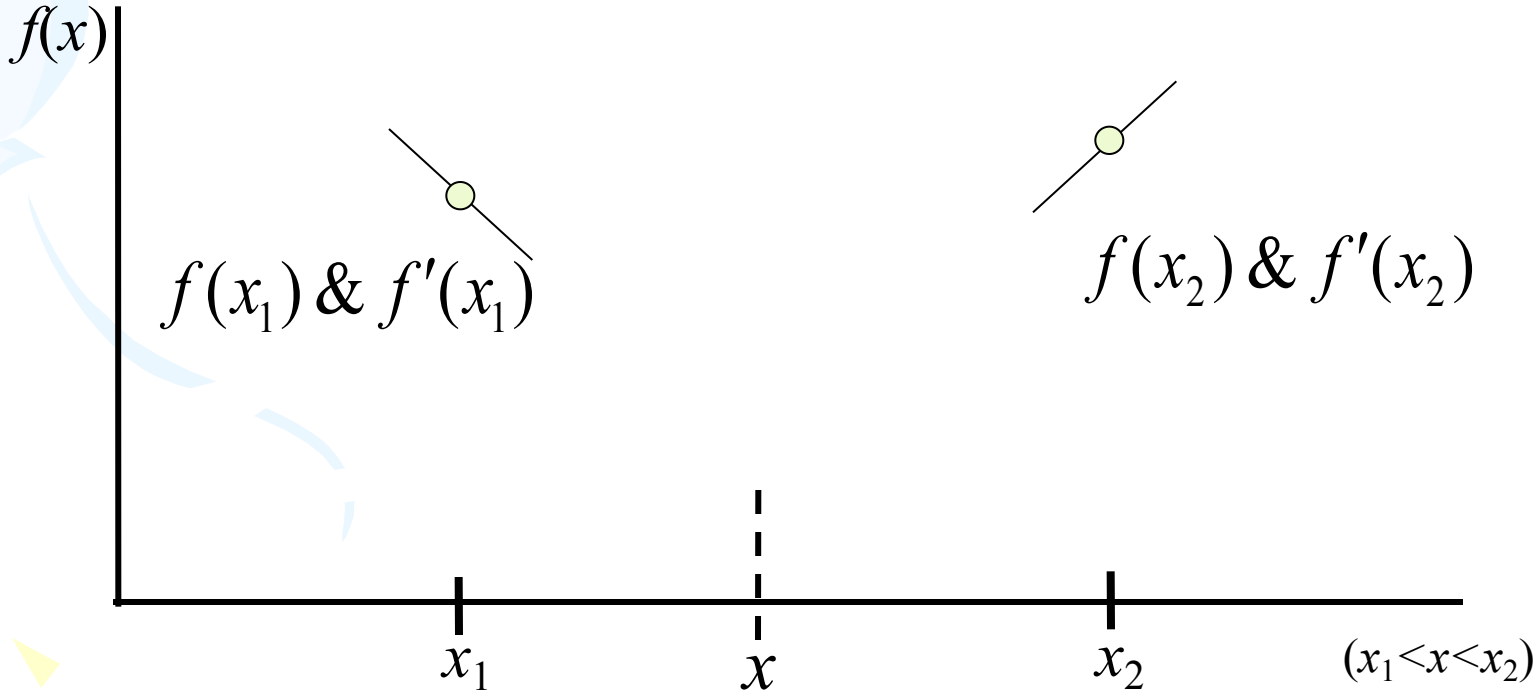
$$f(x_i,y_i)=\frac{x_i-x_2}{x_1-x_2}\frac{y_i-y_2}{y_1-y_2}f(x_1,y_1)+\mathrm{L}$$
$$\frac{x_i-x_1}{x_2-x_1}\frac{y_i-y_2}{y_1-y_2}f(x_2,y_1)+\mathrm{L}$$
$$\frac{x_i-x_2}{x_1-x_2}\frac{y_i-y_1}{y_2-y_1}f(x_1,y_2)+\mathrm{L}$$
$$\frac{x_i-x_1}{x_2-x_1}\frac{y_i-y_1}{y_2-y_1}f(x_2,y_2)$$





Hermite Interpolation

- If we have both the function value, *and the derivative* we can *fully constrain* a higher order polynomial through two points



Interpolating a cubic to two points

- Given the general form for a cubic

$p(x) = ax^3 + bx^2 + cx + d$ the derivative is
$p'(x) = 3ax^2 + 2bx + c$

fitting the two points gives
$f(x_1) = ax_1^3 + bx_1^2 + cx_1 + d$
$f(x_2) = ax_2^3 + bx_2^2 + cx_2 + d$

fitting the derivatives gives
$f'(x_1) = 3ax_1^2 + 2bx_1 + c$
$f'(x_2) = 3ax_2^2 + 2bx_2 + c$

So we have 4 equations with 4 unknowns to solve.

Solve for $p(x)$

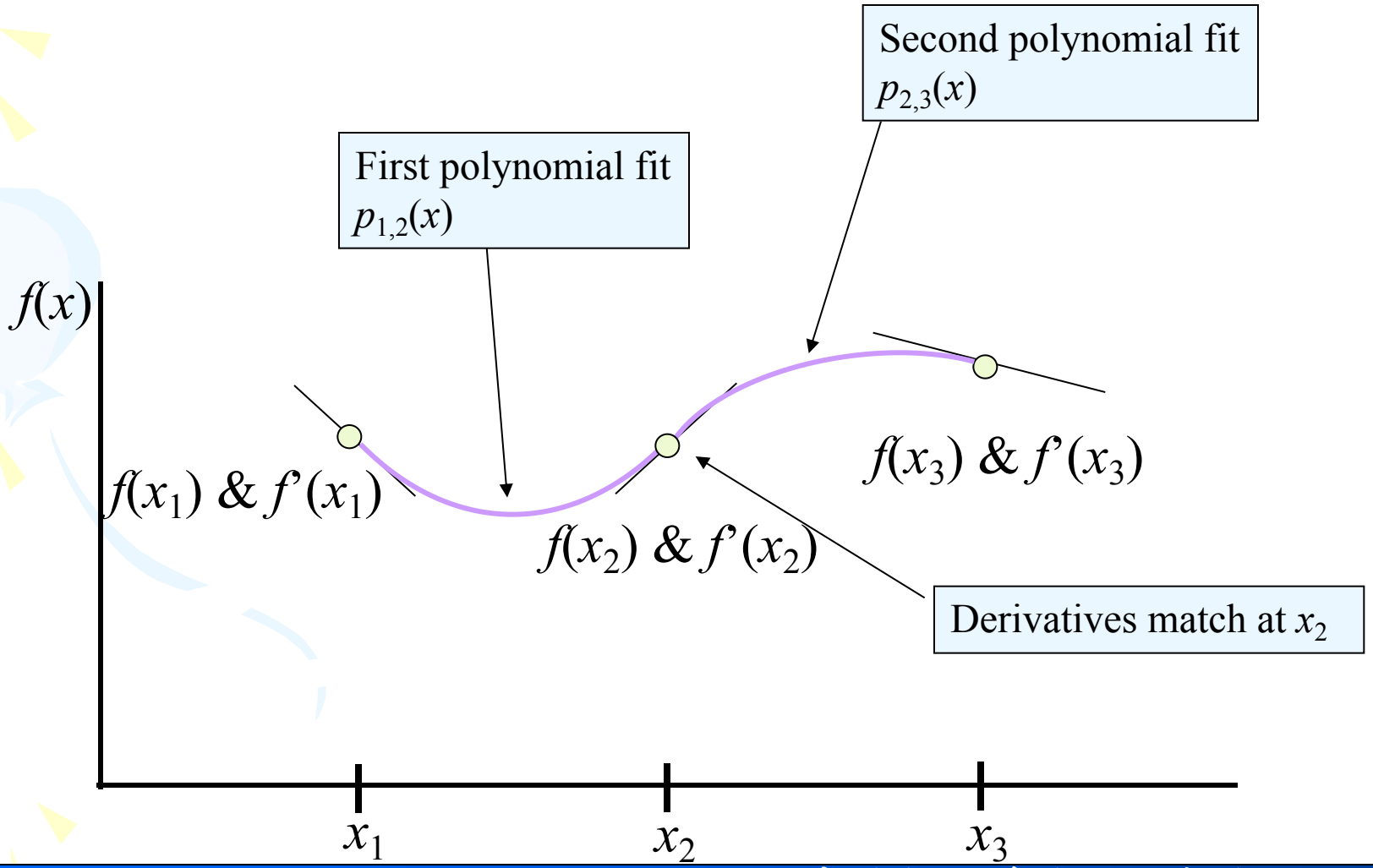
- While again somewhat lengthy, we can solve for $p(x)$ in terms of the known values (subscript denotes for x_1, x_2)

$$p_{1,2}(x) = \frac{(3x_1 - 2x - x_2)(x - x_2)^2}{(x_1 - x_2)^3} f(x_1) + \frac{(3x_2 - 2x - x_1)(x - x_1)^2}{(x_2 - x_1)^3} f(x_2)$$
$$+ \frac{(x - x_1)(x - x_2)^2}{(x_1 - x_2)^2} f'(x_1) + \frac{(x - x_2)(x - x_1)^2}{(x_2 - x_1)^2} f'(x_2)$$

- One important property of Hermite interpolation is that if we fit a different polynomial $p_{2,3}(x)$ between x_2 & x_3 then it will have the same derivative at x_2 as $p_{1,2}(x_2)$ *i.e.* $p'_{1,2}(x_2) = p'_{2,3}(x_2)$
  - Interpolation between successive pairs of points is continuous
  - The *derivatives* are also *continuous*

- Hermite interpolation is thus considered to be *smooth*

Diagram of two Hermite interpolation polynomials



Advantages of Hermite interpolation over Lagrange interpolation

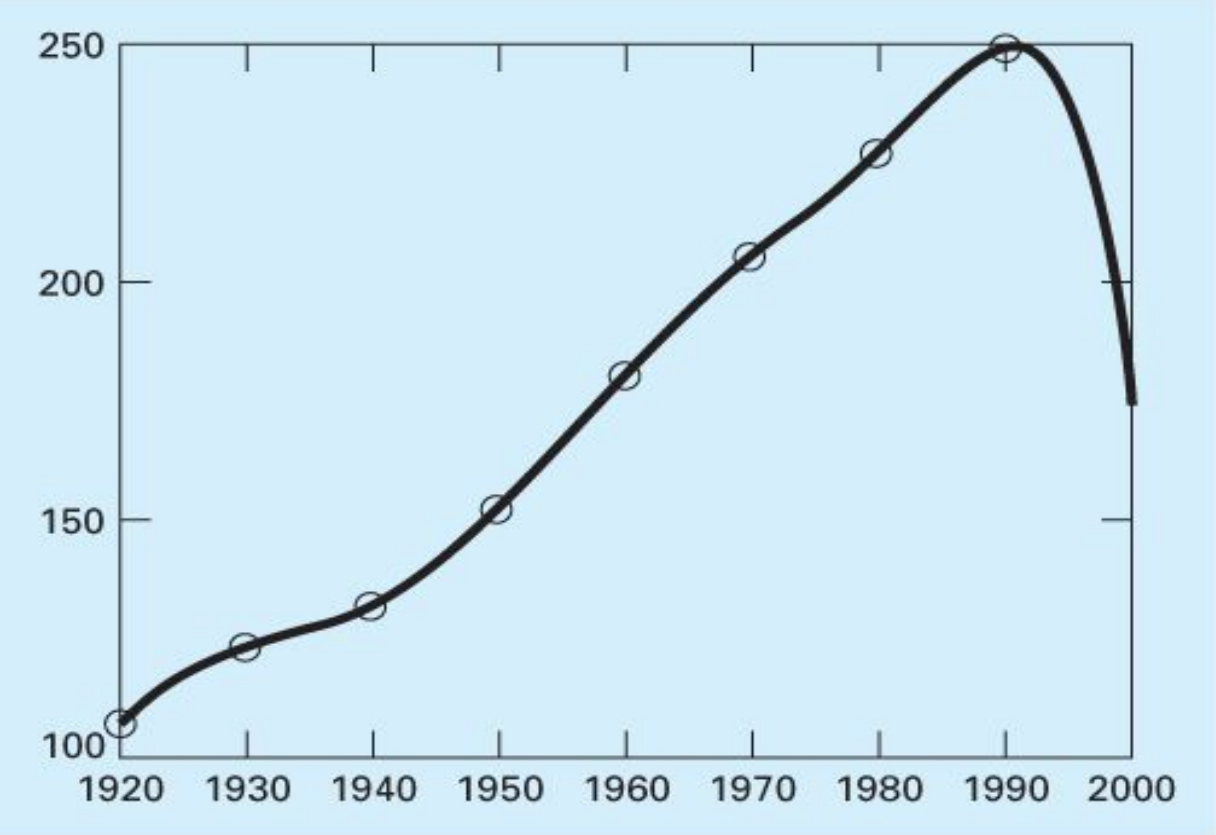
- A series of H.I. polynomials will be continuous at the datums, as will the derivatives (*i.e. smooth*)
- While a series of L.I. polynomials is continuous at the datums the derivatives will *not* be continuous
- H.I. can fit a cubic to `local' data (*i.e.* $x_1<x<x_2$)
- L.I. requires four points to fit a cubic ($x_1<x_2<x<x_3<x_4$)

However, even if we don't have derivatives it would be nice to fit a series of polynomials with continuous slopes at each datum

Extrapolation Hazards

The following shows the results of extrapolating a seventh-order polynomial which is derived from the first 8 points (1920 to 1990) of the USA 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



Summary

- Interpolation shouldn't be confused with fitting, although the terms are frequently used interchangeably
- Lagrange interpolation will fit an order n polynomial to n+1 data points
- Hermite interpolation uses derivatives to solve for a cubic fit with only 2 datums

Homework 7： 04/05/2017

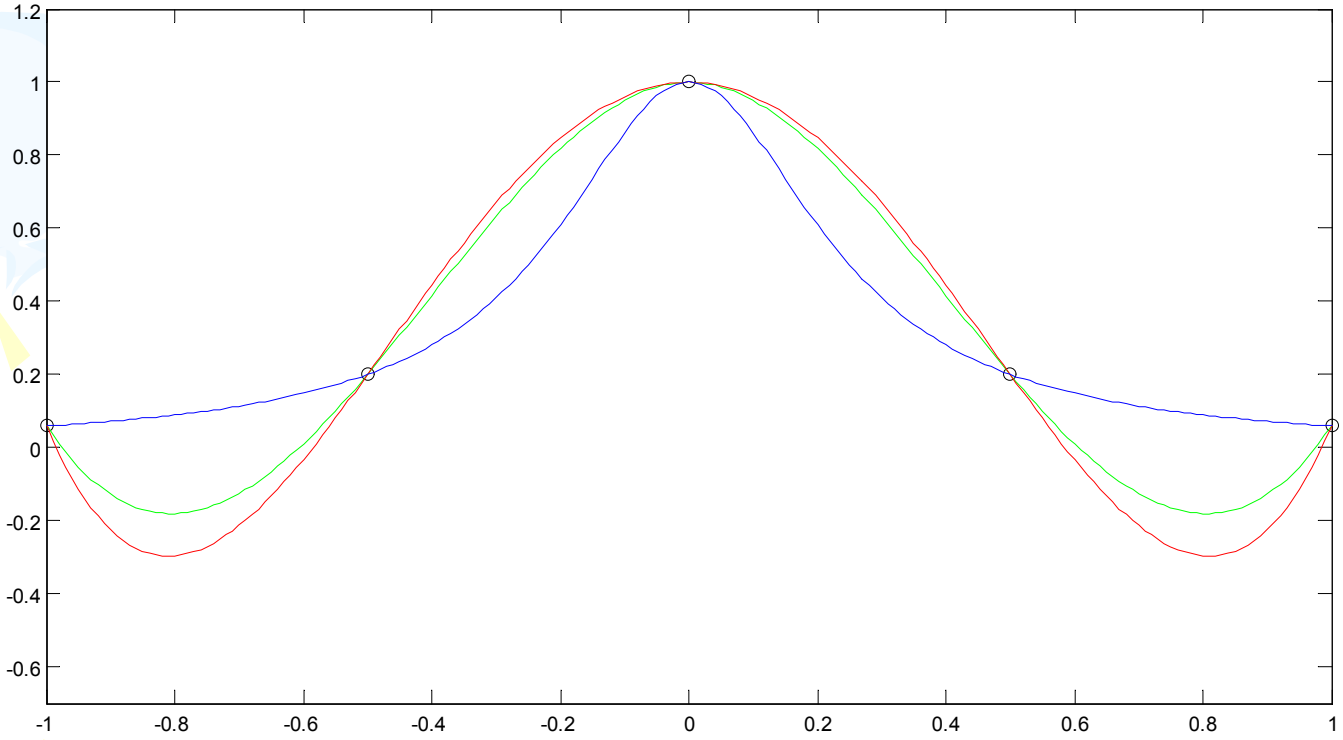
Problem 1: Given a set of seven data points for a voltage-current characteristic of a zener diode

Voltage	*-1.00*	*0.00*	*1.27*	*2.55*	*3.82*	*4.92*	*5.02*
Current	*-14.58*	*0.00*	*0.00*	*0.00*	*0.00*	*0.88*	*11.17*

If the seven points are connected by a polynomial $y = P_6(x)$ of order **6**, the continuous polynomial interpolation is not good because of large swings of the interpolating polynomial between the data points, Instead, the piecewise interpolation made of polynomials of lower order is used to represent a sufficiently smooth curve

Problem 2: For **Runge's function** $f(x) = \dfrac{1}{1+16x^2}$

Please produce five equidistantly spaced points in [-1,1], use 5th Order polynomial fit. On same picture, plot this fit and exact function. Redo for 7,9,17,19,21 nodes



Problem 3: For $f(x) = x\sin(2\pi x + 1)$

Please produce seven equidistantly spaced points in [-1,1], use 7th Order polynomial fit. On same picture, plot this fit and exact function. Redo for 7,9,17,19,21 nodes