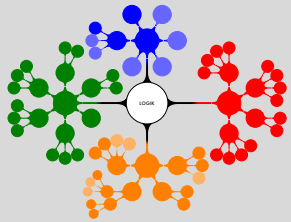
Lecture 13

Data Analysis: Interpolation and Curve Fitting II

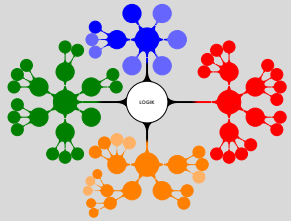
Millikan Oil-Drop Experiment

IKC-MH.55 *Scientific Computing with Python* at January 12, 2024

Dr. Cem Özdoğan
Engineering Sciences Department
İzmir Kâtip Çelebi University



Contents



Interpolation and Curve Fitting II

Divided Differences

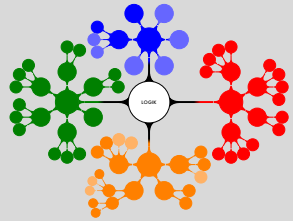
These provide a more efficient way to construct an interpolating polynomial, one that allows one to readily change the degree of the polynomiall.

Spline Curves

Using special polynomials, *splines*, one can fit polynomials to data more accurately than with an interpolating polynomial. At the expense of added computational effort, some important problems that one has with interpolating polynomials is overcome.

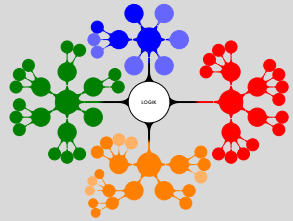
Least-Squares Approximations

Least-Squares Approximations are methods by which polynomials and other functions can be **fitted** to data that are subject to errors likely in experiments. These approximations are widely used **to analyze experimental observations**.



Divided Differences I

- There are two disadvantages to using the Lagrangian polynomial or Neville's method for interpolation.
  1. It involves more arithmetic operations than does the divided-difference method.
  2. More importantly, if we desire to add or subtract a point from the set used to construct the polynomial, we essentially have to start over in the computations.
- Both the Lagrangian polynomials and Neville's method also must repeat all of the arithmetic if we must interpolate at a new x-value.
- The divided-difference method avoids all of this computation.
- Actually, we will not get a polynomial different from that obtained by Lagrange's technique.



Divided Differences II

- Every n^{th}-degree polynomial that **passes through the same $n+1$ points** is identical.
- Only the way that the polynomial is expressed is different.

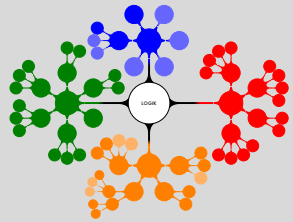
The function, $f(x)$, is known at several values for x:

x_0	f_0
x_1	f_1
x_2	f_2
x_3	f_3

- We do not assume that the x's are evenly spaced or even that the values are arranged in any particular order.
- Consider the n^{th}-degree polynomial written as:

$$P_n(x) = a_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2 + (x - x_0)(x - x_1)\ldots(x - x_{n-1})a_n$$

- If we chose the a_i's so that $P_n(x) = f(x)$ at the $n+1$ known points, then $P_n(x)$ is an interpolating polynomial.
- The a_i's are readily determined by using what are called the **divided differences of the tabulated values**.



Divided Differences III

- A special standard notation for divided differences is

$$\boxed{f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0}}$$

  called the *first divided difference* between x_0 and x_1.
- And, $f[x_0] = f_0 = f(x_0)$ (zero-order difference).

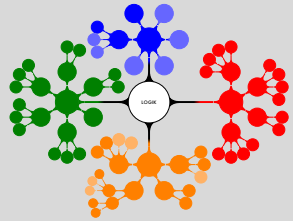
$$f[x_s] = f_s$$

- Second- and higher-order differences are defined in terms of lower-order differences.

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

- For n-terms,

$$\boxed{f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, f_n] - f[x_0, x_1, \ldots, f_{n-1}]}{x_n - x_0}}$$



Divided Differences IV

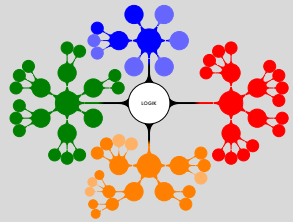
Using the standard notation, a divided-difference table is shown in symbolic form in Table 1.

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
x_0	f_0	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
x_1	f_1	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	$f[x_1, x_2, x_3, x_4]$
x_2	f_2	$f[x_2, x_3]$	$f[x_2, x_3, x_4]$	
x_3	f_3	$f[x_3, x_4]$		

Table: Divided-difference table in symbolic form.

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, \ldots, x_{i+3}]$	$f[x_i, \ldots, x_{i+4}]$
3.2	22.0	8.400	2.856	-0.528	0.256
2.7	17.8	2.118	2.012	0.0865	
1.0	14.2	6.342	2.263		
4.8	38.3	16.750			
5.6	51.7				

Table: Divided-difference table in numerical values.



Divided Differences V

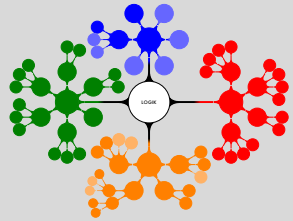
- Table 8 shows specific numerical values.

$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0} = \frac{17.8 - 22.0}{2.7 - 3.2} = 8.4$$

$$f[x_1, x_2] = \frac{f_2 - f_1}{x_2 - x_1} = \frac{14.2 - 17.8}{1.0 - 2.7} = 2.1176$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{2.1176 - 8.4}{1.0 - 3.2} = 2.8556$$

and the others..



Divided Differences VI

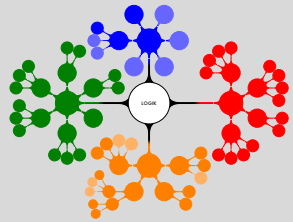
$$
\begin{aligned}
x = x_0: &\quad P_0(x_0) = a_0 \\
x = x_1: &\quad P_1(x_1) = a_0 + (x_1 - x_0)a_1 \\
x = x_2: &\quad P_2(x_2) = a_0 + (x_2 - x_0)a_1 + (x_2 - x_0)(x_2 - x_1)a_2 \\
\vdots &\quad \vdots \\
x = x_n: &\quad P_n(x_n) = a_0 + (x_n - x_0)a_1 + (x_n - x_0)(x_n - x_1)a_2 + \ldots \\
&\quad +(x_n - x_0)\ldots(x_n - x_{n-1})a_n
\end{aligned}
$$

- If $P_n(x)$ is to be an interpolating polynomial, it must match the table for all $n+1$ entries:

$$P_n(x_i) = f_i \text{ for } i = 0, 1, 2, \ldots, n.$$

- Each $P_n(x_i)$ will equal f_i, if $a_i = f[x_0, x_1, \ldots, x_i]$. We then can write:

$$
\begin{aligned}
P_n(x) = f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] \\
+(x - x_0)(x - x_1)(x - x_2)f[x_0, \ldots, x_3] \\
+(x - x_0)(x - x_1)\ldots(x - x_{n-1})f[x_0, \ldots, x_n]
\end{aligned}
$$



Divided Differences VII

- Write interpolating polynomial of degree-3 that fits the data of Table 8 at all points $x_0 = 3.2$ to $x_3 = 4.8$.

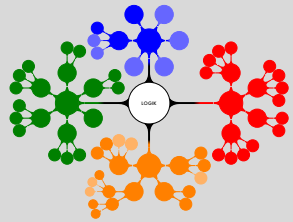
$$P_3(x) = 22.0 + 8.400(x - 3.2) + 2.856(x - 3.2)(x - 2.7)$$

$$-0.528(x - 3.2)(x - 2.7)(x - 1.0)$$

- What is the fourth-degree polynomial that fits at all five points?
- **We only have to add one more term to $P_3(x)$**

$$P_4(x) = P_3(x) + 0.2568(x - 3.2)(x - 2.7)(x - 1.0)(x - 4.8)$$

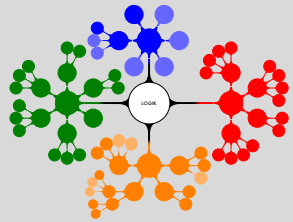
- If we compute the interpolated value at $x = 3.0$, we get the same result: $P_3(3.0) = 20.2120$.
- **This is not surprising, because all third-degree polynomials that pass through the same four points are identical.**
- **They may look different but they can all be reduced to the same form.**



Divided Differences VIII

Example py-file: Constructs a table of divided-difference coefficients. Diagonal entries are coefficients of the polynomial. mydivDiffTable_interpolation.py

```
Constructed MydivDiffTable:
[[22.          0.          0.          0.        ]
 [17.8         8.4         0.          0.        ]
 [14.2         2.11764706  2.85561497  0.        ]
 [38.3         6.34210526  2.01164676 -0.52748013]]
Diagonal entries are the coefficients of the polynomial:
[22.          8.4         2.85561497 -0.52748013]
MydivDiffTable - Estimated y-value for x=3.000000 : 20.211961
```

Divided Differences IX

- **Divided differences for a polynomial**
- It is of interest to look at the divided differences for $f(x) = P_n(x)$.
- Suppose that $f(x)$ is the cubic

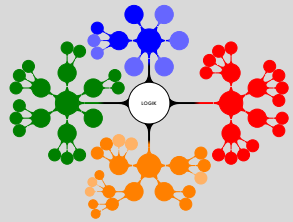
$$f(x) = 2x^3 - x^2 + x - 1.$$

- Here is its divided-difference table:

x_i	$f[x_i]$	$f[x_i, x_{i+1}$	$f[x_i, x_{i+1}$, $x_{i+2}]$	$f[x_i, \ldots$, $x_{i+3}]$	$f[x_i, \ldots$, $x_{i+4}]$	$f[x_i, \ldots$, $x_{i+5}]$
0.30	-0.736	2.480	3.000	2.000	0.000	0.000
1.00	1.000	3.680	3.600	2.000	0.000	
0.70	-0.104	2.240	5.400	2.000		
0.60	-0.328	8.720	8.200			
1.90	11.008	21.020				
2.10	15.212					

Table: Divided-difference table in numerical values for a polynomial.

- Observe that the third divided differences are all the same.
- It then follows that all higher divided differences will be zero.



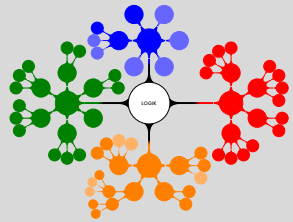
Divided Differences X

$$P_3(x) = f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2]$$

$$+(x - x_0)(x - x_1)(x - x_2)f[x_0, x_1, x_2, x_3]$$

```
1  import numpy as np
2  D=np.array([[ -0.736], [2.480], [3.000], [2.000]])
3  print(np.transpose(D))
4  # [[-0.736  2.48   3.     2.    ]]
5  import sympy as sym
6  x = sym.Symbol('x')
7  P3=D[0]+(x-0.3)*D[1]+(x-0.3)*(x-1)*D[2]+(x-0.3)*(x-1)*(x-0.7)*D[3]
8  print(P3)
9  # [2.48*x + 2.0*(x - 1)*(x - 0.7)*(x - 0.3) + 3.0*(x - 1)*(x -
       0.3) - 1.48]
10 print(sym.expand(2.48*x + 2.0*(x - 1)*(x - 0.7)*(x - 0.3) + 3.0*(x
       - 1)*(x - 0.3) - 1.48))
11 # 2.0*x**3 - 1.0*x**2 + 1.0*x - 1.0
```

which is same with the starting polynomial.



Spline Curves I

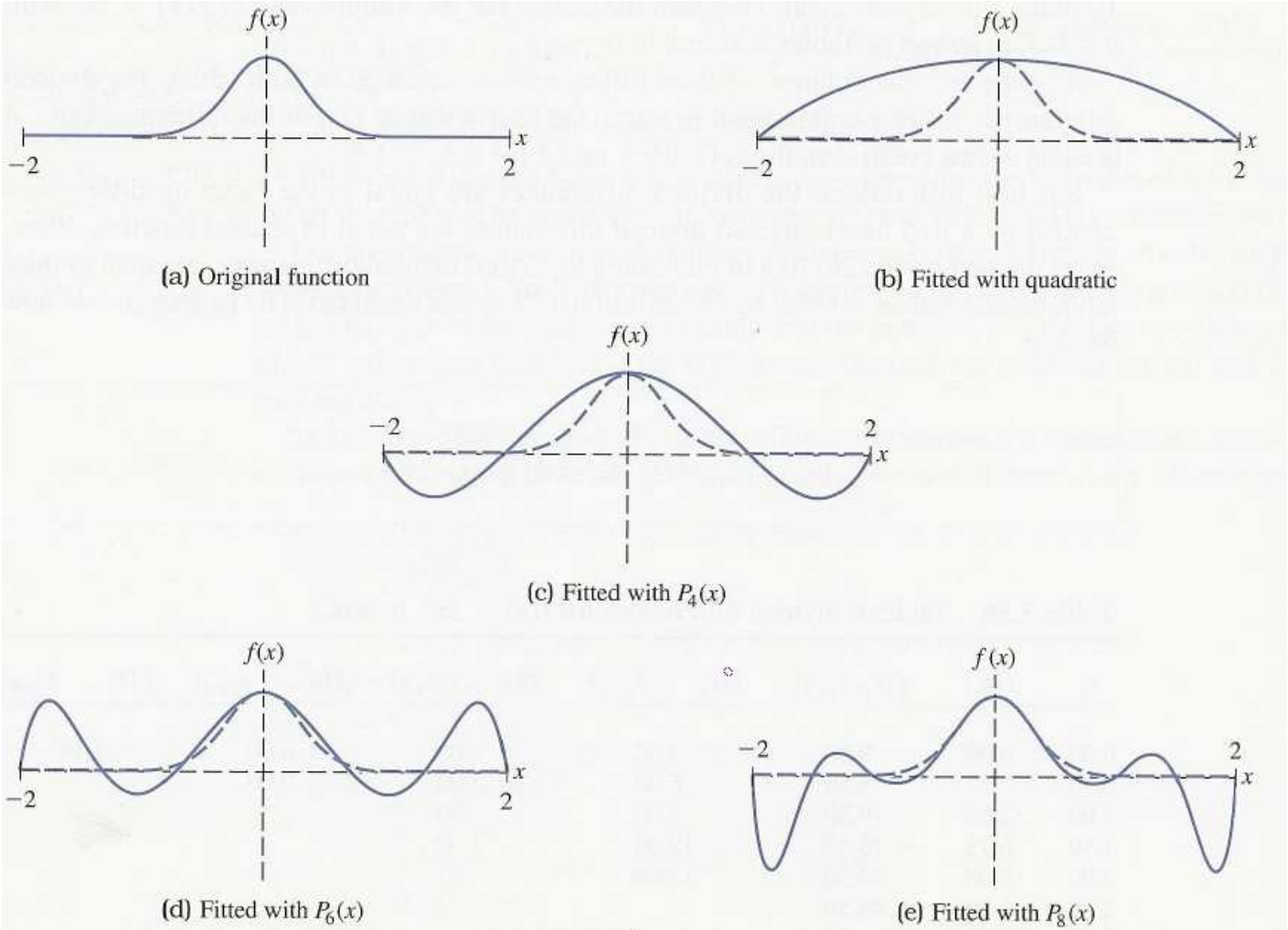
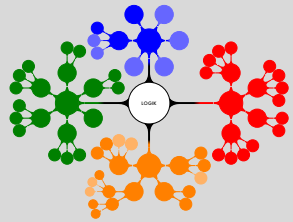
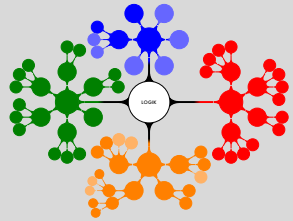


Figure: Fitting with different degrees of the polynomial.



Spline Curves II

- There are times when fitting an interpolating polynomial to data points is very difficult.
- Figure 1a is plot of $f(x) = cos^{10}(x)$ on the interval $[-2, 2]$.
- It is a nice, smooth curve but has a pronounced maximum at $x = 0$ and is near to the x-axis for $|x| > 1$.
- The curves of Figure 1b,c, d, and e are for polynomials of degrees $-2, -4, -6$, and -8 that match the function at evenly spaced points.
- **None of the polynomials is a good representation of the function**.



Spline Curves III

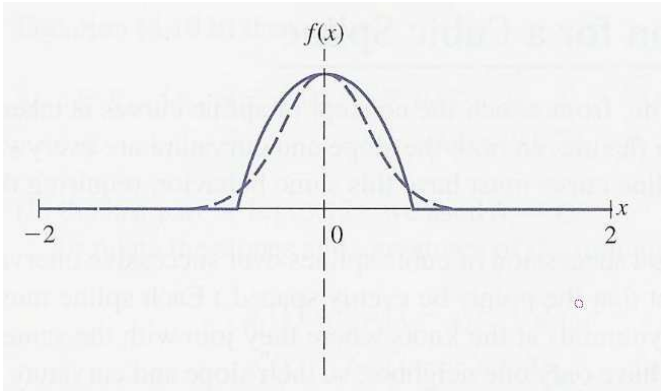
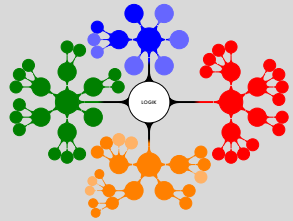


Figure: Fitting with quadratic in subinterval.

- One might think that a solution to the problem would be to break up the interval $[-2, 2]$ into subintervals
- and **fit separate polynomials** to the function in these smaller intervals.
- Figure 2 shows a much better fit if we use a quadratic between $x = -0.65$ and $x = 0.65$ and with $P(x) = 0$ outside that interval.
- That is better but there are discontinuities in the slope where the separate polynomials join.
- **This solution is known as spline curves**.



Spline Curves IV

- Suppose that we have a set of $n+1$ points (which do not have to be evenly spaced):

$$(x_i, y_i), \ \textit{with } i = 0, 1, 2, \ldots, n.$$

- A spline fits a set of n^{th}-degree polynomials, $g_i(x)$, between each pair of points, from x_i to x_{i+1}.
- The points at which the splines join are called knots.

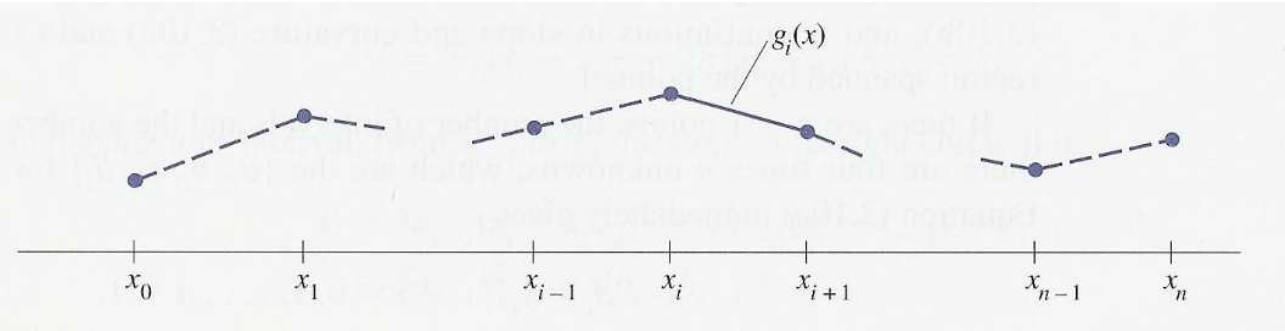
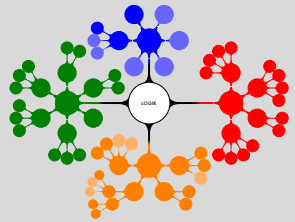


Figure: Linear spline.

- If the polynomials are all of degree 1, we have a *linear spline* and the curve would appear as in the Fig. 3.
- **The slopes are discontinuous where the segments join.**



The Equation for a Cubic Spline I

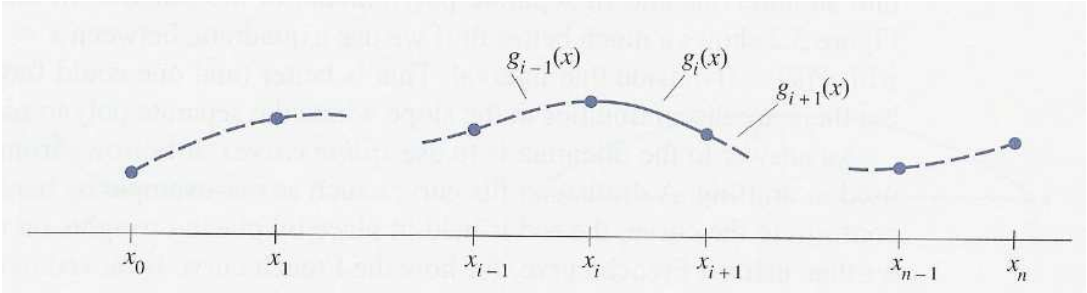
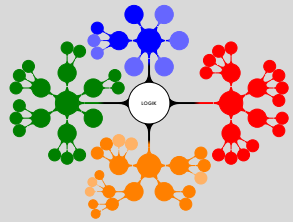


Figure: Cubic spline.

- We will create a succession of cubic splines over successive intervals of the data (See Fig. 4).
- Each spline must join with its neighbouring cubic polynomials at the knots where they join with the **same slope and curvature**.
- **We write the equation for a cubic polynomial, $g_i(x)$, in the i^{th} interval, between points** $(x_i, y_i), (x_{i+1}, y_{i+1})$ (*solid line*).
- It has this equation:

$$g_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

- The *dashed curves* are other cubic spline polynomials.



The Equation for a Cubic Spline II

- Thus, the cubic spline function is of the form

$$g(x) = g_i(x) \text{ on the interval} [x_i, x_{i+1}], \text{ for } i = 0, 1, \ldots, n-1$$

- and meets these conditions:

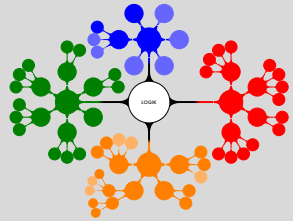
$$g_i(x_i) = y_i, \ i = 0, 1, \ldots, n-1 \text{ and } g_{n-1}(x_n) = y_n \quad (1)$$

$$g_i(x_{i+1}) = g_{i+1}(x_{i+1}), \ i = 0, 1, \ldots, n-2 \quad (2)$$

$$g_i^{'}(x_{i+1}) = g_{i+1}^{'}(x_{i+1}), \ i = 0, 1, \ldots, n-2 \quad (3)$$

$$g_i^{''}(x_{i+1}) = g_{i+1}^{''}(x_{i+1}), \ i = 0, 1, \ldots, n-2 \quad (4)$$

- Equations say that the cubic spline fits to each of the points Eq. 1, is continuous Eq. 2, and is continuous in slope and curvature Eq. 3 and Eq. 4, throughout the region spanned by the points.



Least-Squares Approximations I

- Until now, we have assumed that the data are accurate,
- but when these values are derived **from an experiment**, there is **some error in the measurements**.

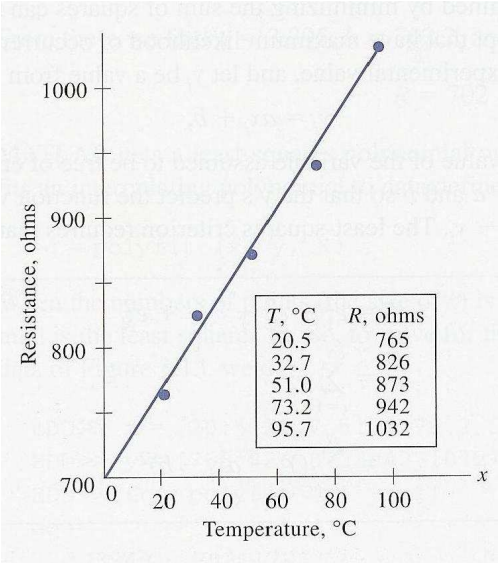
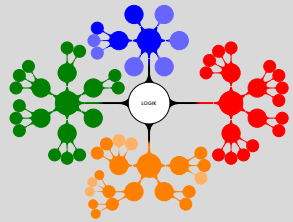


Figure: Resistance vs Temperature graph for the Least-Squares Approximation.

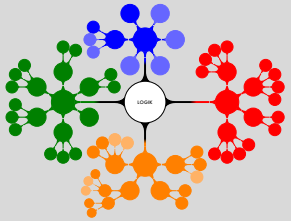
- Some students are assigned to find the effect of temperature on the resistance of a metal wire.
- They have recorded the temperature and resistance values in a table and have plotted their findings, as seen in Fig. 5.
- **The graph suggest a linear relationship**.

$$R = aT + b$$



Least-Squares Approximations II

- Values for the parameters, *a* & *b*, can be obtained from the plot.
- If someone else were given same data and asked to draw the line,
- it is not likely that they would draw exactly the same line and they would get different values for *a* & *b*.
- A way of fitting a line to experimental data that is to **minimize the deviations** of the points from the line.
- The usual method for doing this is called the **least-squares method**.



Least-Squares Approximations III

- The deviations are determined by the **distances between the points and the line**.

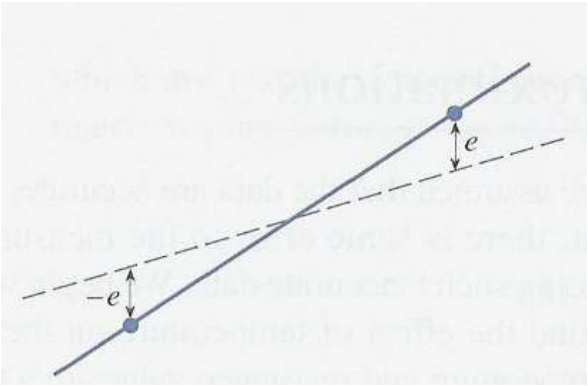
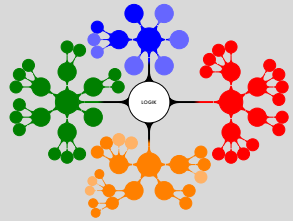


Figure: Minimizing the deviations by making the sum a minimum.

- Consider the case of only two points (See Fig. 6).
- Obviously, the best line passes through each point,
- but any line that passes through the midpoint of the segment connecting them has a *sum of errors equal to zero*.

- We might first suppose we could minimize the deviations by making their sum a minimum, but this is **not an adequate criterion**.
- We might accept the criterion that we make the magnitude of the maximum error a minimum (the so-called *minimax* criterion).
- The usual criterion is to minimize the sum of the *squares* of the errors, the *least-squares* principle.



Least-Squares Approximations IV

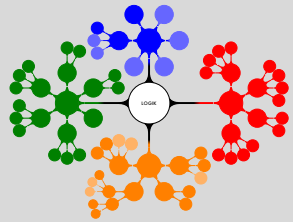
- Let Y_i represent an experimental value, and let y_i be a value from the equation

$$y_i = ax_i + b$$

  where x_i is a particular value of the variable assumed to be free of error.
- We wish to determine the best values for a & b so that the y's predict the function values that correspond to x-values.
- Let errors defined by $e_i = Y_i - y_i = Y_i - (ax_i + b)$
- The least-squares criterion requires that S be a minimum.

$$\begin{aligned} S &= e_1^2 + e_2^2 + \ldots + e_n^2 = \sum_{i=1}^{N} e_i^2 \\ &= \sum_{i=1}^{N} (Y_i - ax_i - b)^2 \end{aligned}$$

  N is the number of (x, Y)-pairs.
- We reach the minimum by proper choice of the parameters a & b, so they are the *variables* of the problem.



Least-Squares Approximations V

- At a minimum for S, the two partial derivatives will be zero.

$$\partial S/\partial a \qquad \& \qquad \partial S/\partial b$$

- Remembering that the x_i and Y_i are data points unaffected by our choice our values for a and b, we have

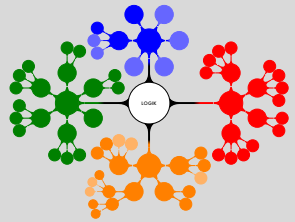
$$\frac{\partial S}{\partial a} = 0 = \sum_{i=1}^{N} 2(Y_i - ax_i - b)(-x_i)$$
$$\frac{\partial S}{\partial b} = 0 = \sum_{i=1}^{N} 2(Y_i - ax_i - b)(-1)$$

- Dividing each of these equations by -2 and expanding the summation, we get the so-called **normal equations**

$$\begin{aligned} a\sum x_i^2 + b\sum x_i &= \sum x_i Y_i \\ a\sum x_i + bN &= \sum Y_i \end{aligned}$$

From $i = 1$ to $i = N$.

- Solving these equations simultaneously gives the values for slope and intercept a & b.



Least-Squares Approximations VI

- For the data in Fig. 5 we find that

$$N = 5, \quad \sum T_i = 273.1, \quad \sum T_i^2 = 18607.27,$$

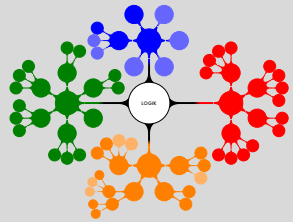
$$\sum R_i = 4438, \quad \sum T_i R_i = 254932.5$$

- Our **normal equations** are then

$$\begin{aligned} 18607.27a + 273.1b &= 254932.5 \\ 273.1a + 5b &= 4438 \end{aligned}$$

- From these we find $a = 3.395$, $b = 702.2$, and

$$R = 702.2 + 3.395T$$



Least-Squares Approximations VII

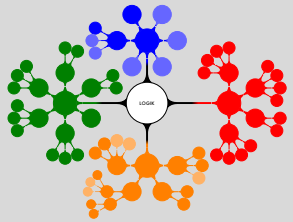
- In many cases, data from experimental tests are *not linear*, so we need to fit to them some function other than a first-degree polynomial.
- Popular forms are the exponential form

$$y = ax^b \quad or \quad y = ae^{bx}$$

- The exponential forms are usually linearized by taking logarithms before determining the parameters, for the case $y = ax^b$:

$$lny = lna + blnx \quad or \quad lny = lna + bx$$

- We now fit the new variable $z = lny$ as a linear function of lnx or x as described earlier.
- Here we do not minimize the sum of squares of the deviations of Y from the curve, but rather the deviations of lnY.
  - In effect, this amounts to minimizing the squares of the percentage errors, which itself may be a desirable feature.



Least-Squares Approximations VIII

- Because polynomials can be readily manipulated, fitting such functions to data that do not plot linearly is common.
- We assume the functional relationship

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \tag{5}$$

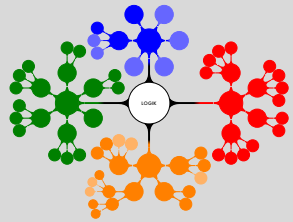
with errors defined by

$$e_i = Y_i - y_i = Y_i - (a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n)$$

- We again use Y_i to represent the observed or experimental value corresponding to x_i, with x_i free of error.
- We minimize the sum of squares;

$$S = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (Y_i - a_0 - a_1 x - a_2 x^2 - \ldots - a_n x^n)^2$$

At the minimum, all the partial derivatives $\partial S/\partial a_0, \partial S/\partial a_n$ vanish.



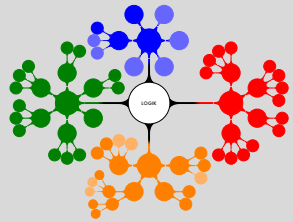
Least-Squares Approximations IX

- Writing the equations for these gives $n+1$ equations:

$$\begin{aligned}
\frac{\partial S}{\partial a_0} &= 0 = \sum_{i=1}^{N} 2(Y_i - a_0 - a_1 x_i - a_2 x_i^2 - \ldots - a_i x_i^n)(-1) \\
\frac{\partial S}{\partial a_1} &= 0 = \sum_{i=1}^{N} 2(Y_i - a_0 - a_1 x_i - a_2 x_i^2 - \ldots - a_i x_i^n)(-x_i) \\
&\vdots \\
\frac{\partial S}{\partial a_n} &= 0 = \sum_{i=1}^{N} 2(Y_i - a_0 - a_1 x_i - a_2 x_i^2 - \ldots - a_i x_i^n)(-x_i^n)
\end{aligned}$$

- Dividing each by -2 and rearranging gives the $n+1$ **normal equations** to be solved simultaneously:

$$\begin{aligned}
a_0 N + a_1 \sum x_i + a_2 \sum x_i^2 + \ldots + a_n \sum x_i^n &= \sum Y_i \\
a_0 \sum x_i + a_1 \sum x_i^2 + a_2 \sum x_i^3 + \ldots + a_n \sum x_i^{n+1} &= \sum x_i Y_i \\
a_0 \sum x_i^2 + a_1 \sum x_i^3 + a_2 \sum x_i^4 + \ldots + a_n \sum x_i^{n+2} &= \sum x_i^2 Y_i \\
&\vdots \\
a_0 \sum x_i^n + a_1 \sum x_i^{n+1} + a_2 \sum x_i^{n+2} + \ldots + a_n \sum x_i^{2n} &= \sum x_i^n Y_i
\end{aligned} \tag{6}$$

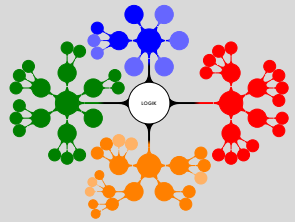


Least-Squares Approximations X

- Putting these equations in matrix form shows the coefficient matrix;

$$\begin{bmatrix} N & \sum x_i & \sum x_i^2 & \sum x_i^3 & \cdots & \sum x_i^n \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \cdots & \sum x_i^{n+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \cdots & \sum x_i^{n+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sum x_i^n & \sum x_i^{n+1} & \sum x_i^{n+2} & \sum x_i^{n+3} & \cdots & \sum x_i^{2n} \end{bmatrix} [a] = \begin{bmatrix} \sum Y_i \\ \sum x_i Y_i \\ \sum x_i^2 Y_i \\ \vdots \\ \sum x_i^n Y_i \end{bmatrix} \quad (7)$$

  All the summatins in Eqs. 6 and 7 run from 1 to N. We will let B stand for the coefficient matrix.
- Equation 7 represents a linear system.
- Degrees higher than 4 are used very infrequently. It is often better to fit a series of lower-degree polynomials to subsets of the data.
- Matrix B of Eq. 7 is called the *normal matrix* for the least-squares problem.



Least-Squares Approximations XI

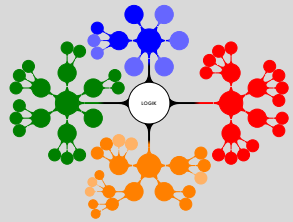
- There is another matrix that corresponds to this, called the *design matrix*. It is of the form;

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & \dots & x_N \\ x_1^2 & x_2^2 & x_3^2 & \dots & x_N^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^n & x_2^n & x_3^n & \dots & x_N^n \end{bmatrix}$$

- AA^T is just the coefficient matrix of Eq. 7. It is easy to see that Ay, where y is the column vector of Y-values, gives the right-hand side of Eq. 7. We can rewrite Eq. 7 in matrix form, as

$$AA^T a = Ba = Ay$$

  so it is to find the solution.
- It is illustrated the use of Eq. 6 to fit a quadratic to the data of Table 4. Figure 7 shows a plot of the data.
- The data are actually a perturbation of the relation $y = 1 - x + 0.2x^2$.



Least-Squares Approximations XII

To set up the **normal equations**, we need the sums tabulated in Table 4.

x_i	0.05	0.11	0.15	0.31	0.46	0.52	0.70	0.74	0.82	0.98	1.171
Y_i	0.956	0.890	0.832	0.717	0.571	0.539	0.378	0.370	0.306	0.242	0.104

$\Sigma x_i = 6.01$ $\quad N = 11$

$\Sigma x_i^2 = 4.6545$ $\quad \Sigma Y_i = 5.905$

$\Sigma x_i^3 = 4.1150$ $\quad \Sigma x_i Y_i = 2.1839$

$\Sigma x_i^4 = 3.9161$ $\quad \Sigma x_i^2 Y_i = 1.3357$

Table: Data to illustrate curve fitting.

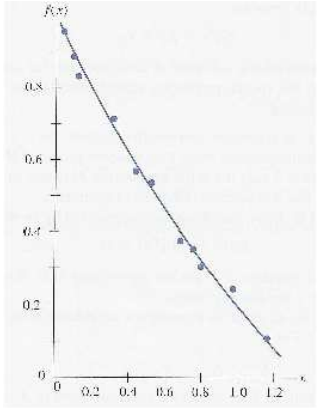


Figure: Figure for the data to illustrate curve fitting.

- The equations to be solved are:

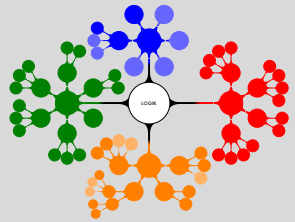
$$\begin{aligned} 11a_0 + 6.01a_1 + 4.6545a_2 &= 5.905 \\ 6.01a_0 + 4.6545a_1 + 4.1150a_2 &= 2.1839 \\ 4.6545a_0 + 4.1150a_1 + 3.9161a_2 &= 1.3357 \end{aligned}$$

The result is $a_0 = 0.998$, $a_2 = -1.018$, $a_3 = 0.225$.

- So the least- squares method gives

$$y = 0.998 - 1.018x + 0.225x^2$$

which we compare to $y = 1 - x + 0.2x^2$. Errors in the data cause the equations to differ.



Least-Squares Approximations XIV

Example py-file: Fitting an 4^{th} order polynomial to $y = cosx$ function in [0,π] by Least-Square Approximation. Gaussian elimination & back substitution. Pivoting. mylsa.py

```
MyGEPivShow - Solution Vector :  [[ 9.88494865e-01  8.91986965e-02 -6.82520463e-01  1.42129999e-01
   7.82504827e-04]]
NumPy Solve - Solution Vector : [[ 9.88494865e-01  8.91986965e-02 -6.82520463e-01  1.42129999e-01
   7.82504827e-04]]
```

Step x	Exact cos x	Least-Square Approx. p(x)	Error
0.32	0.9492354180824408	0.9518138732787098	-0.0025784551962690
0.64	0.8020957578842927	0.8034114580487246	-0.0013157001644319
0.96	0.5735199860724567	0.5715268997183653	0.0019930863540913
1.28	0.2867152096319555	0.2845964026657791	0.0021188069661764
1.60	-0.0291995223012888	-0.0287469049433018	-0.0004526173579870
1.92	-0.3421496511508982	-0.3396729711555569	-0.0024766799953413
2.24	-0.6203616120126798	-0.6191548202300811	-0.0012067917825987
2.56	-0.8355887771314077	-0.8379685526383827	0.0023797755069750
2.88	-0.9659793123979747	-0.9666933450643868	0.0007140326664120
3.20	-0.9982947757947531	-0.9757114504044269	-0.0225833253903263

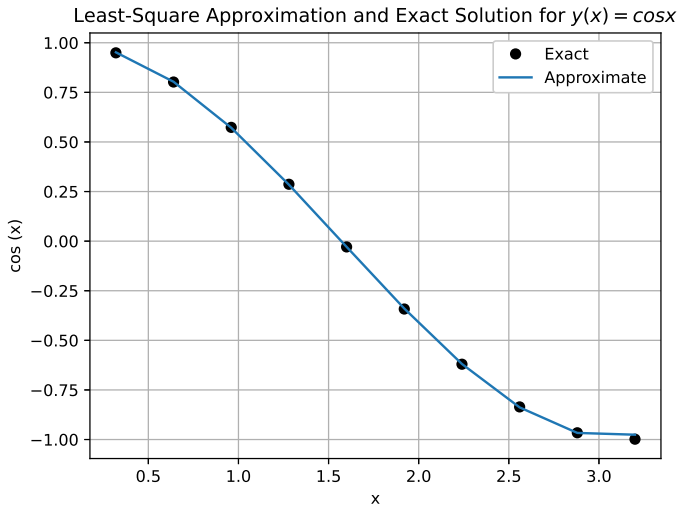
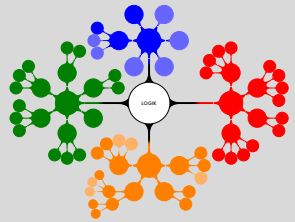
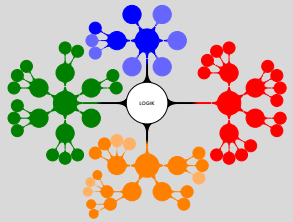


Figure: Polynomial Least-Square Approximation.



Millikan oil-drop experiment I

- In 1910, Millikan succeeded in measuring the electron charge for the first time with a very sensitive experiment.
- In this experiment, electrically charged oil droplets remained suspended in the air under the influence of a force that balanced the electric field applied between the plates of a capacitor and the gravitational force.
- From the equation $qV/d = mg$, the electric charge of each droplet could be calculated by measuring the potential difference and masses.
- When Millikan listed these electric charges from smallest to largest, he showed that they were multiples of a basic unit of charge, and from there he determined the electric charge as $e = 1.65 \times 10^{-9}\ C$ (Today's value is $e = 1.602 \times 10^{-9}\ C$).



Millikan oil-drop experiment II

Example py-file: Millikan oil-drop experiment by Least-Square Approximation. Gaussian elimination & back substitution. Pivoting. mylsa_millikan.py

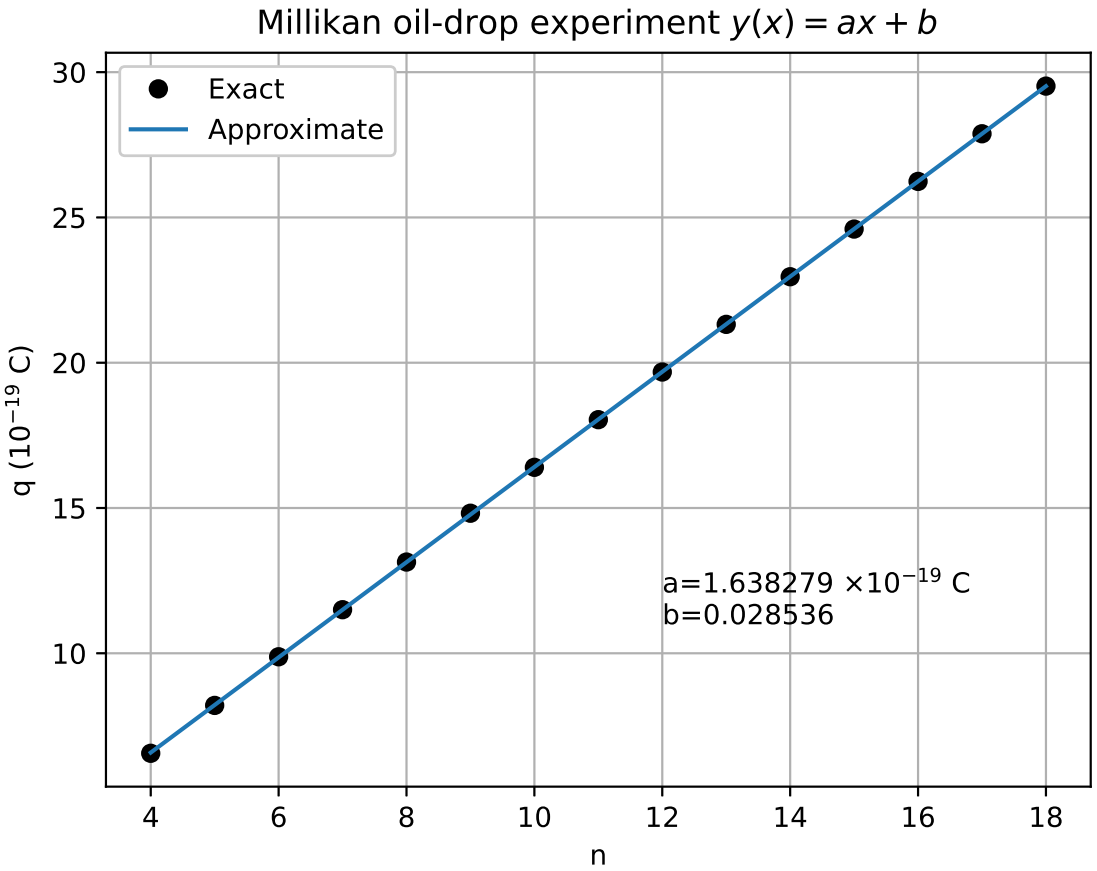


Figure: Millikan oil-drop experiment.