Part 4
Chapter 17

Polynomial Interpolation

Polynomial Interpolation

- 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).
  - (1,1) (3,3) (5,5) → (2,?)

TABLE 15.1 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	0.833	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}

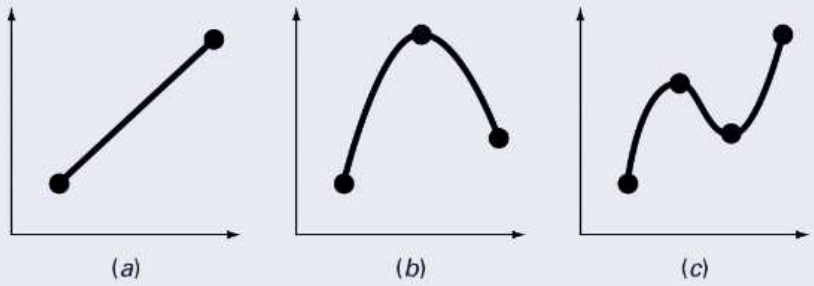
Polynomial Interpolation

- The most common method for this purpose is polynomial interpolation,
  - where an *(n-1)th* order polynomial is solved that passes through *n* data 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

1. Since polynomial interpolation provides as many basis functions as there are data points (*n*),
   - the polynomial coefficients can be found exactly using linear algebra.

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

```
>> p = polyfit(x,y,length(x)-1);
>> ft = polyval(p, t);
```

Polynomial Interpolation Problems

- One problem that can occur with solving for the coefficients of a polynomial is that the system to be inverted is in the form:

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

  - *Vandermonde matrices*
    - very ill-conditioned : solutions are very sensitive to round-off errors.
    - The issue can be minimized by scaling and shifting the data.

$$V = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \cdots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \cdots & \alpha_m^{n-1} \end{bmatrix}$$

named after Alexandre-Théophile Vandermonde

geometric progression in each row

Example

- x=[300 400 500] , y=[0.616 0.525 0.457]

- $f(x)=p_1x^2+p_2x+p_3$

```
b=y'
A=[x.^2' x' ones(3,1)]
p=A\b
```

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

Newton Interpolating Polynomials

- Interpolating between points at x_1 *and* x_2 .
  - First order : straight linear interpolation.
  - Second order : parabola interpolation.
  - *n-th order : fit (n+1) data point to n-th order* polynomial.

- Avoid *Vandermonde matrices.*

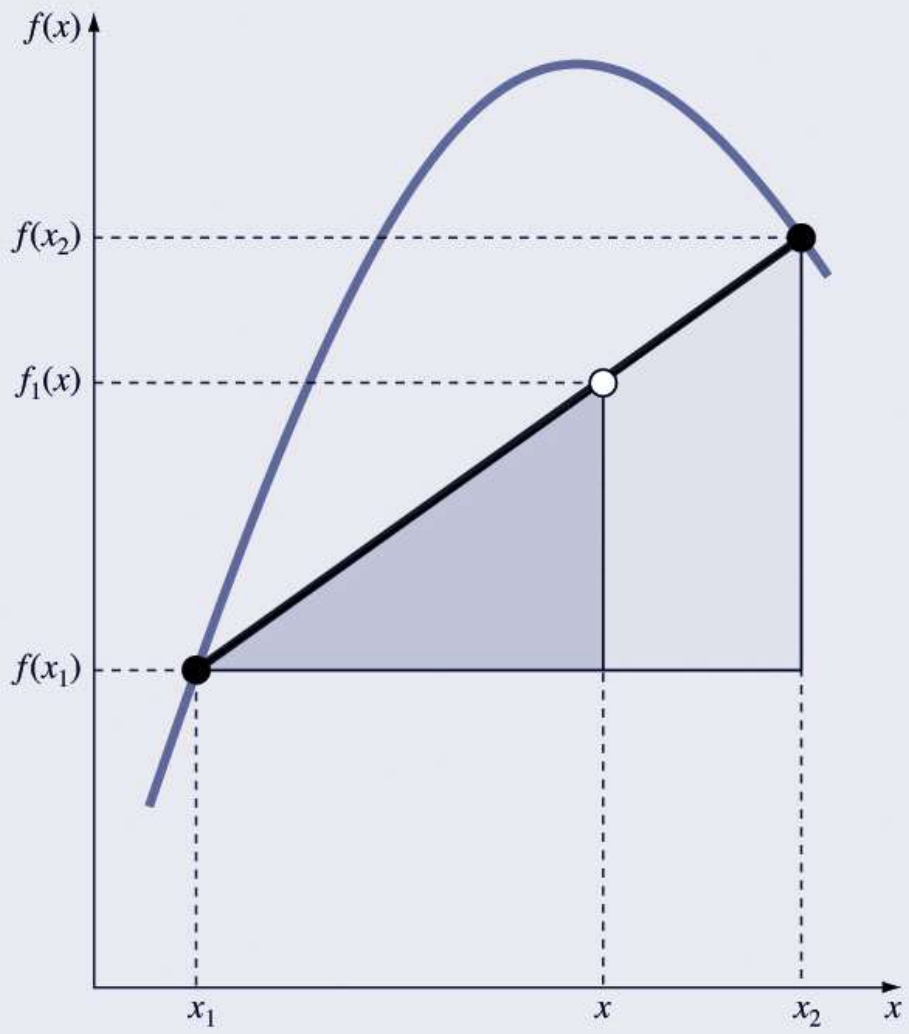
$$f_{n-1}(x) = b_1 + b_2(x - x_1) + \cdots + b_n(x - x_1)(x - x_2)\cdots(x - x_{n-1})$$

First-order Newton Interpolating Polynomials

- Linear interpolation of two point.
  - may be obtained from 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:

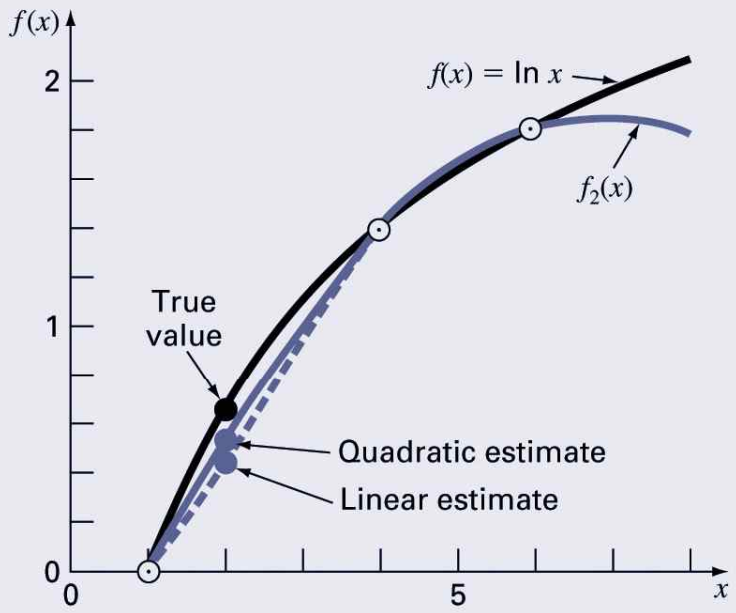
$$f_1(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1)$$



Second-order Newton Interpolating Polynomials

- Curvature to the line connecting the points, but still goes through the first two points.

- The resulting formula based on known points x_1, x_2, and x_3 and the values of the dependent function at those points is:

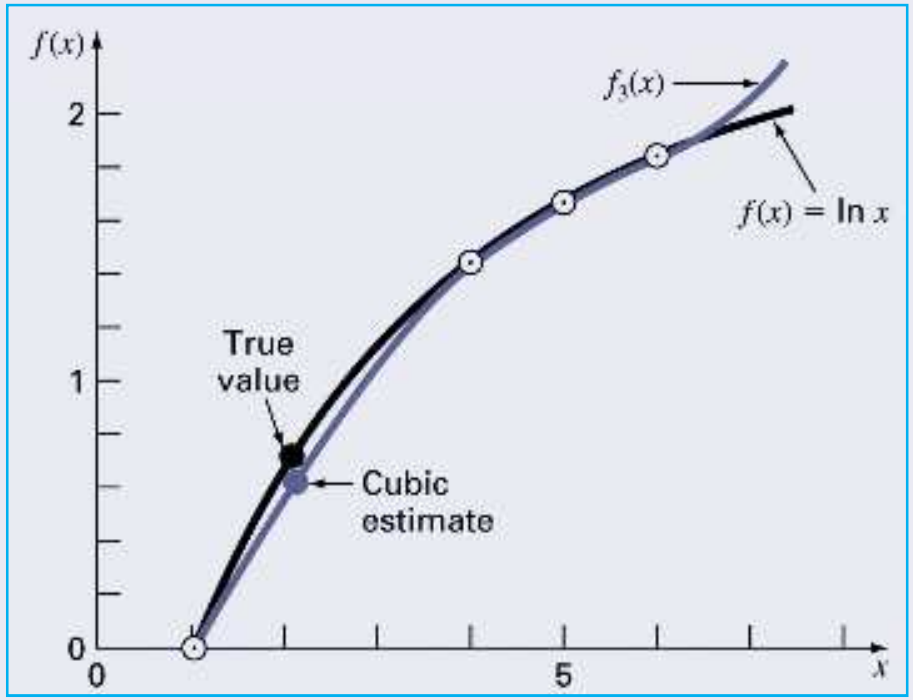


$$f_2(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) + \frac{\dfrac{f(x_3) - f(x_2)}{x_3 - x_2} - \dfrac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1}(x - x_1)(x - x_2)$$

Comparison

- Interpolate *ln x* function between [1, 4]

- Error at *ln 2*
  - First order: 48.3%
  - Second order: 18.4%
  - Third order: 9.3%.



(*n-1)th* Order Newton Interpolating Polynomials

- By linear equation,

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

$$\Rightarrow \begin{cases} P(x_0) = a_0 = f(x_0) \\ P_1(x_1) = a_0 + a_1(x_1 - x_0) = f(x_1) \\ \vdots \\ P(x_n) = a_0 + a_1(x_n - x_0) + a_2(x_n - x_0)(x_n - x_1) + \cdots + a_n(x_n - x_0)\cdots(x_n - x_{n-1}) \\ \qquad = f(x_n) \end{cases}$$

$$\Rightarrow \begin{bmatrix} 1, & 0, & 0, & \cdots\cdots, & 0 \\ 1, & (x_1 - x_0), & 0, & \cdots\cdots, & 0 \\ 1, & (x_2 - x_0), & (x_2 - x_0)(x_2 - x_1), & \cdots\cdots, & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 1, & (x_n - x_0), & (x_n - x_0)(x_n - x_1), & \cdots & (x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1}) \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}$$

(*n-1)th* Order Newton Interpolating Polynomials

- By *divided differences,*
  - *Difference*

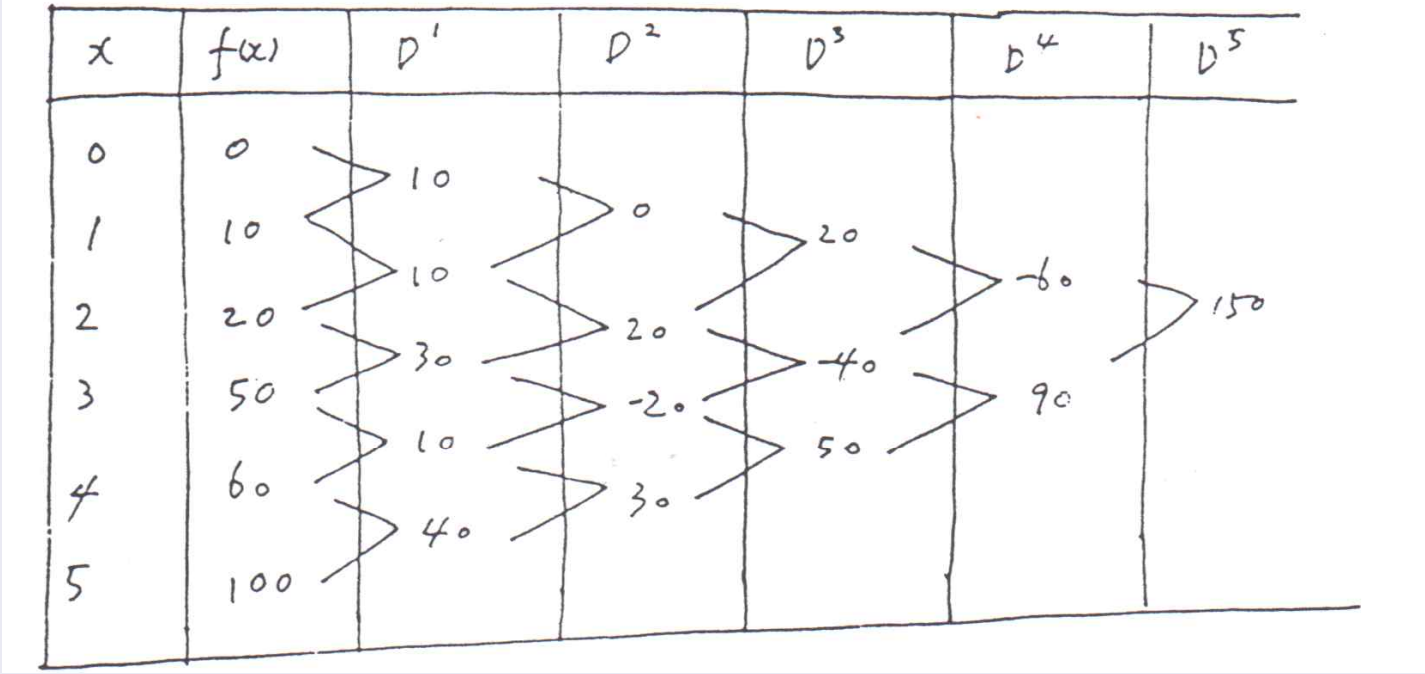
함수차분 (Difference) : $f(x_i) - f(x_j)$

전향차분 (Forward Difference) : $\triangle f(x_i) = f(x_{i+1}) - f(x_i)$

후향차분 (Backward Difference) : $\nabla f(x_i) = f(x_i) - f(x_{i-1})$

중심차분 (Central D.) : $\delta f(x_i) = \frac{1}{2}\{f(x_{i+1}) - f(x_{i-1})\}$

차분표 (Difference Table)



$x=0$ 에서 각 차분은

$\Delta f(0)=10$, $\Delta^2 f(0)=0$, $\Delta^3 f(0)=20$, $\Delta^4 f(0)=-60$, $\Delta^5 f(0)=150$

$x=3$ 에서 각 차분은

$\Delta f(3)=10$, $\Delta^2 f(3)=30$

Divided Differences

- Divided difference (차분상) are calculated as follows:

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

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

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

- Divided differences are calculated using divided difference of a smaller number of terms:

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	$f(x_4)$			

Newton 보간식에서

$$p(x_0) = a_0 = f(x_0) = f[x_0]$$

$$p(x_1) = a_0 + a_1(x_1 - x_0) = f(x_1)$$

$$\longrightarrow a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_0, x_1]$$

$$p(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) = f(x_2)$$

$$\longrightarrow a_2 = \frac{f(x_2) - f(x_0) - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}$$

$$= \frac{1}{x_2 - x_1}\left[\frac{f(x_2) - f(x_0)}{x_2 - x_0} - \frac{a_1(x_2 - x_0)}{x_2 - x_0}\right]$$

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

$$= f[x_0, x_1, x_2] \qquad \left(\because f[a,b,c] = f[b,c,a]\right)$$

$f[x_1, x_0, x_2]$

$$\vdots$$

$$a_n = f[x_0, x_1, x_2, \cdots, x_n]$$

$$\Rightarrow P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + f[x_0, x_1, \cdots, x_n](x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

(9x)

x	1	3	4
$f(x)$	2	10	17

이때 $x=2$ 에서의 보간값 $\left(f(x)=x^2+1\right)$

x_i	$f(x_i)$	$f[\,,\,]$	$f[\,,\,,\,]$
1	2	$(10-2)/(3-1)=4$	$(7-4)/(4-1)=1$
3	10	$(17-10)/(4-3)=7$	
4	17		

$$P_2(x)=2+4(x-1)+1\,(x-1)(x-3)$$

$$\therefore\ P_2(2)=2+4\cdot 1+1\cdot 1\cdot(-1)=5 \qquad \left(f(2)=5\right)$$

Newton 후향차분공식

$$P_n(x) = f[x_n] + f[x_{n-1}, x_n](x - x_n) + \cdots$$

$$+ f[x_0, x_1, \cdots, x_n](x - x_n)\cdots(x - x_1)$$

(예)

x	f(x)
1.0	0.765197
1.3	0.620086
1.6	0.455402
1.9	0.2818186
2.2	0.1103623

x	f(x)	first D.D.	second D.D.	third D.D	fourth D.D
1.0	0.765197	-0.483706	-0.108733	0.0658784	0.0018297
1.3	0.620086	-0.548946	-0.044433	0.0680740	
1.6	0.455402	-0.578620	0.0118233		
1.9	0.2818186	-0.571518			
2.2	0.1103632				

- $f(1.1) = ?$
- $f(2.0) = ?$

Lagrange Interpolating Polynomials

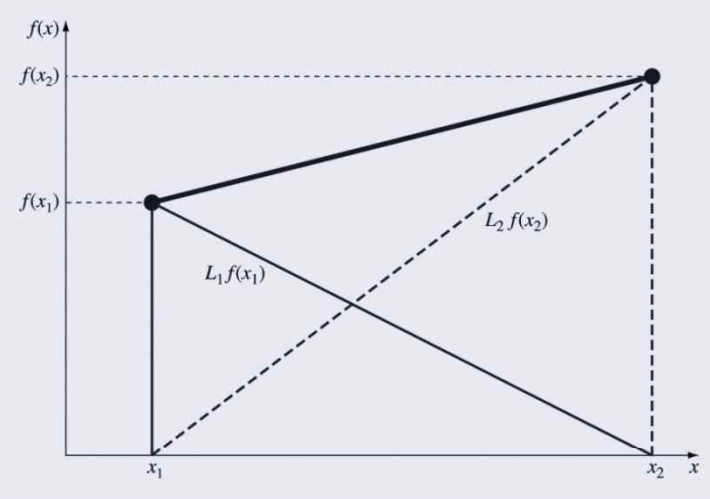
Order	Simple	Lagrange
$1st$	$f_1(x) = a_1 + a_2 x$	$f_1(x) = L_1 f(x_1) + L_2 f(x_2)$
$2nd$	$f_2(x) = a_1 + a_2 x + a_3 x^2$	$f_2(x) = L_1 f(x_1) + L_2 f(x_2) + L_3 f(x_3)$

where the L_i are weighting coefficients that are functions of x.

First-Order Lagrange Interpolating Polynomials

- a weighted combination of two linear interpolations

- The resulting formula based on known points x_1 and x_2 and the values of the dependent function at those points is:



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

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

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

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:

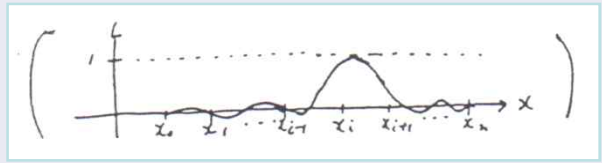
$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

❑ Polynomial passing thru $(x_1, y_1), \ldots, (x_n, y_n)$

$$p(x) = L_1 y_1 + L_2 y_2 + \ldots + L_n y_n$$

$$L_k(x) = \frac{(x - x_1)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)}{(x_k - x_1)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)}$$

$$L_k(x_i) = \begin{cases} 1 & if\ i = k \\ 0 & if\ i \neq k \end{cases}$$

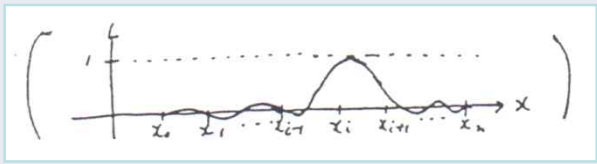


Lagrange 보간다항식

$$P_n(x) = \sum_{i=0}^{n} a_i L_i(x) = L_0(x)a_0 + L_1(x)a_1 + \cdots + L_n(x)a_n$$

$$P_n(x_i) = f(x_i)$$

$$\begin{bmatrix} L_0(x_0) & & & \\ & L_1(x_1) & & \\ & & L_2(x_2) & \\ & & & \ddots \\ & & & & L_n(x_n) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}$$



$$L_i(x_i) = 1$$

$$\therefore \; a_i = f(x_i)$$

$$\Rightarrow P_n(x) = L_0(x)f(x_0) + L_1(x)f(x_1) + \cdots + L_n(x)f(x_n)$$

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

- Lagrange 보간식 : x를 순서로 나열할 필요 없음
  자료구간이 등간격일 필요 없음.

(ex) 완전타원적분 (Elliptic Integrals of the 1st Kind)

$$K(k)=\int_0^{\frac{\pi}{2}} \frac{1}{[1-(\sin k)^2\sin^2 x]^{\frac{1}{2}}}\,dx$$

$K(1)=1.5708$, $K(4)=1.5127$, $K(6)=1.5751$ 일 때 $K(3.5)$ 를 라그랑지 보간식으로 추정하자.

$x_0=1,\quad x_1=4,\quad x_2=6$

$$L_0(x)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}=\frac{(x-4)(x-6)}{(1-4)(1-6)} \qquad L_0(3.5)=\frac{(3.5-4)(3.5-6)}{15}=0.08333$$

$$L_1(x)=\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}=\frac{(x-1)(x-6)}{(4-1)(4-6)} \qquad L_1(3.5)=\frac{(3.5-1)(3.5-6)}{-6}=1.04167$$

$$L_2(x)=\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}=\frac{(x-1)(x-4)}{(6-1)(6-4)} \qquad L_2(3.5)=\frac{(3.5-1)(3.5-4)}{10}=-0.12500$$

$$\therefore K(3.5)\cong L_0(3.5)K(1)+L_1(3.5)K(4)+L_2(3.5)K(6)$$
$$=(0.08333)(1.5708)+(1.04167)(1.5127)+(-0.14)(1.5751)$$
$$=1.57225$$

Inverse Interpolation

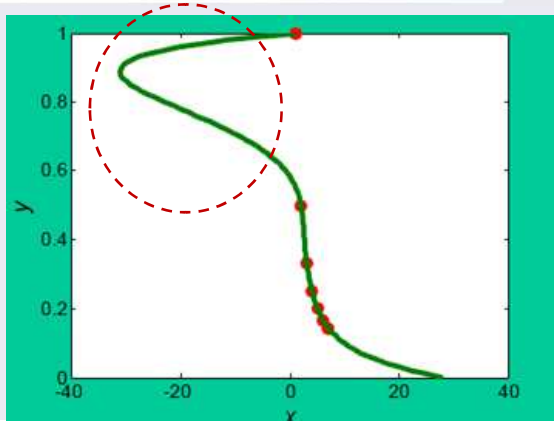
- Interpolation : find *f(x) for a given x*
- Inverse interpolation : find the *x* for a given *f*(*x*)
  - Reverse the role of *f(x) and x.*
    - *Let y=f(x), then x=f $^{-1}$(y).*
    - Not always yield good result if *y is not distributed* evenly.
  - First, interpolate *f(x) as a function of x, then*
    - *S*olve the corresponding roots problem:
    - *f(x)-f* $_{desired}$*=0*

- **(1,1) (3,3) (5,5) → (?,2)**
- **Polynomial interpolation**
  - **Determine (n-1)-th order polynomial** $p(x)$
  - **solve** $p(t) = 2$

Example of Bad Reverse Interpolation

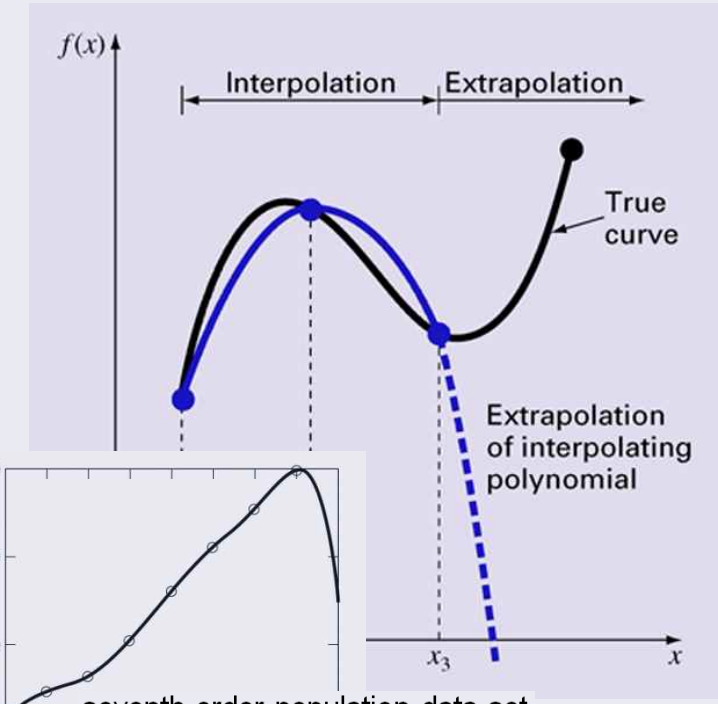
- *f(x)=1/x*. Interpolate *x with f(x).*

f(x)	0.1429	0.1667	0.2	0.25	0.3333	0.5	1
x	7	6	5	4	3	2	1



Extrapolation

- Estimating a value of $f(x)$ that lies outside the range of the known base points $x_1, x_2, \ldots, x_n$.
  - Extreme care should be exercised when extrapolating
  - Can lead to great error.



다항식 보간법의 오차 $E(\hat{x})$

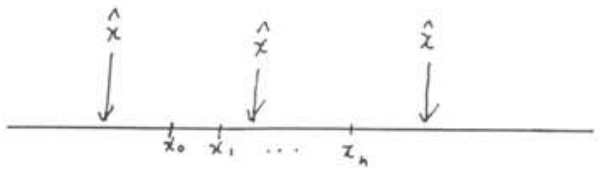
$$E(\hat{x}) = \frac{f^{(n+1)}(c)}{(n+1)!}(\hat{x}-x_0)(\hat{x}-x_1)\cdots\cdots(\hat{x}-x_n)$$

$$|E(\hat{x})| <= \left|\frac{f^{(n+1)}(c)}{(n+1)!}\right| \cdot |(\hat{x}-x_0)| \cdot |(\hat{x}-x_1)| \cdots\cdots |(\hat{x}-x_n)|$$

$\Rightarrow |\hat{x}-x_i|$가 작아야 $|E(x)|$ 도 작음

$\Rightarrow \hat{x}$가 $(x_0, \ldots, x_n)$의 중간 부분에 위치할 때 $|E(\hat{x})|$ 가 최소

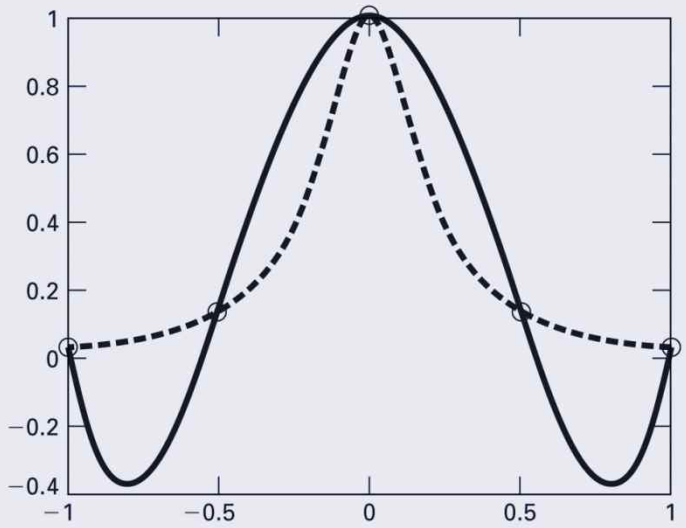
$\Rightarrow \hat{x}$가 $(x_0, \ldots, x_n)$의 바깥에 있을 때 가장 커짐 (Extrapolation)

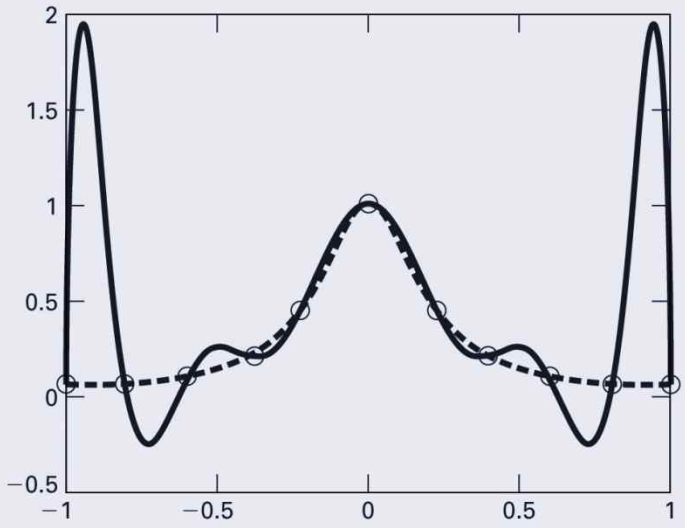


Oscillations

- Higher-order polynomials not always better than lower order:
  - more round-off errors due to ill-conditioning.
  - Oscillations.

- Example: fit $f(x)=1/(1+25x^2)$ *with 4-th and 10-th order.*
  - The dashed line represents an function (the circles represent samples of the function), and
  - the solid line represents the results of a polynomial interpolation:





THE END

Homework : MATLAB 예제
Report : 17-1, 17-5, 15-6

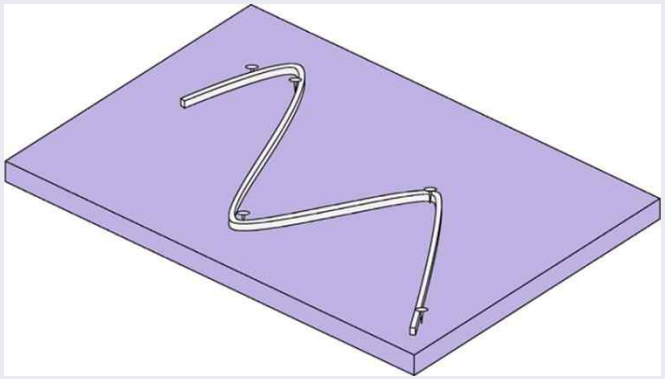
Part 4
Chapter 18

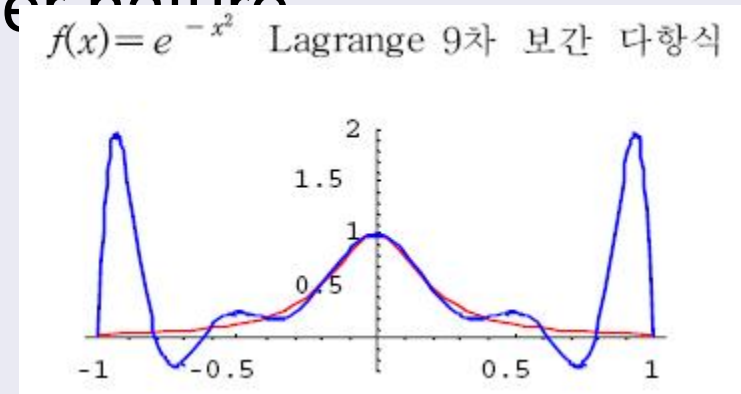
Splines and Piecewise Interpolation

Introduction to Splines

- An alternative approach to using a single *(n-1)th* order polynomial to interpolate between *n* points is
  - to apply lower-order polynomials in a piecewise fashion to subsets of data points.
  - These connecting polynomials are called spline functions.

- Splines minimize oscillations and reduce round-off error due to their lower-order nature



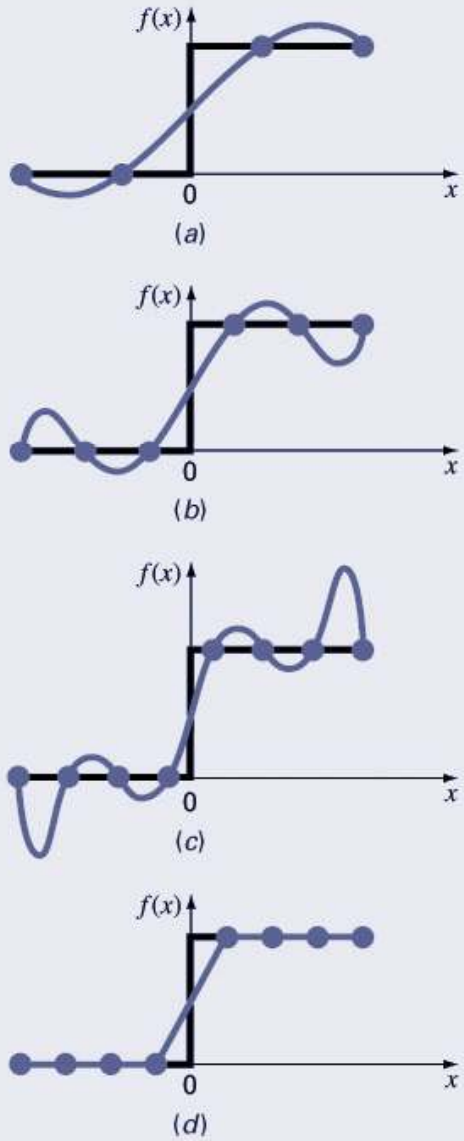


Runge's phenomenon

Higher Order vs. Splines

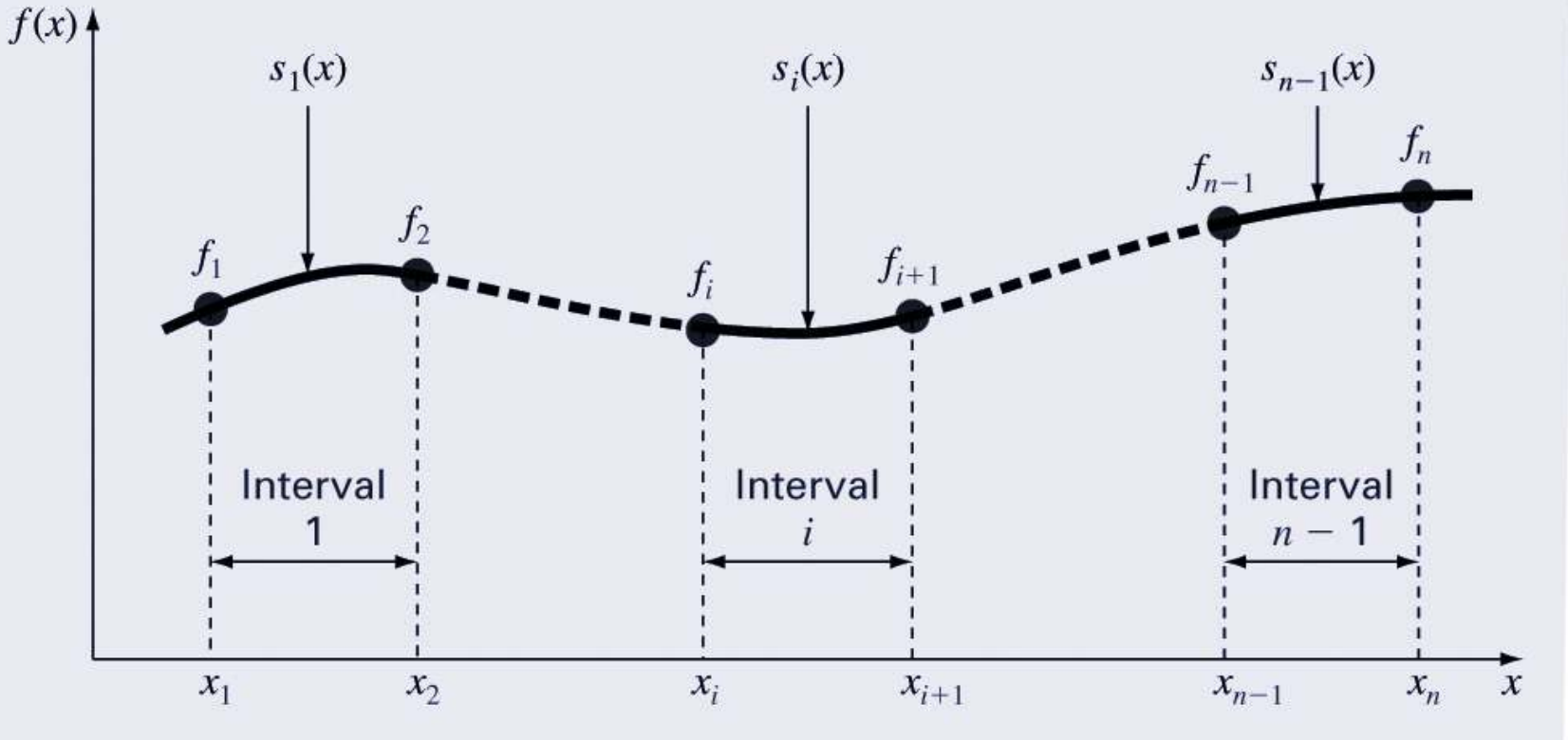
- Splines eliminate oscillations by
  - using small subsets of points for each interval rather than every point.

- This is especially useful when there are jumps in the data:
  a. 3rd order polynomial
  b. 5th order polynomial
  c. 7th order polynomial
  d. Linear spline
     - seven *1st* order polynomials generated by using pairs of points at a time
     - direct connection of the data points.
     - no oscillation.
     - not smooth.



Spline Development

- Spline function ($s_i(x)$) coefficients are calculated for each interval of a data set.
- The number of data points (f_i) used for each spline function depends on the order of the spline function.



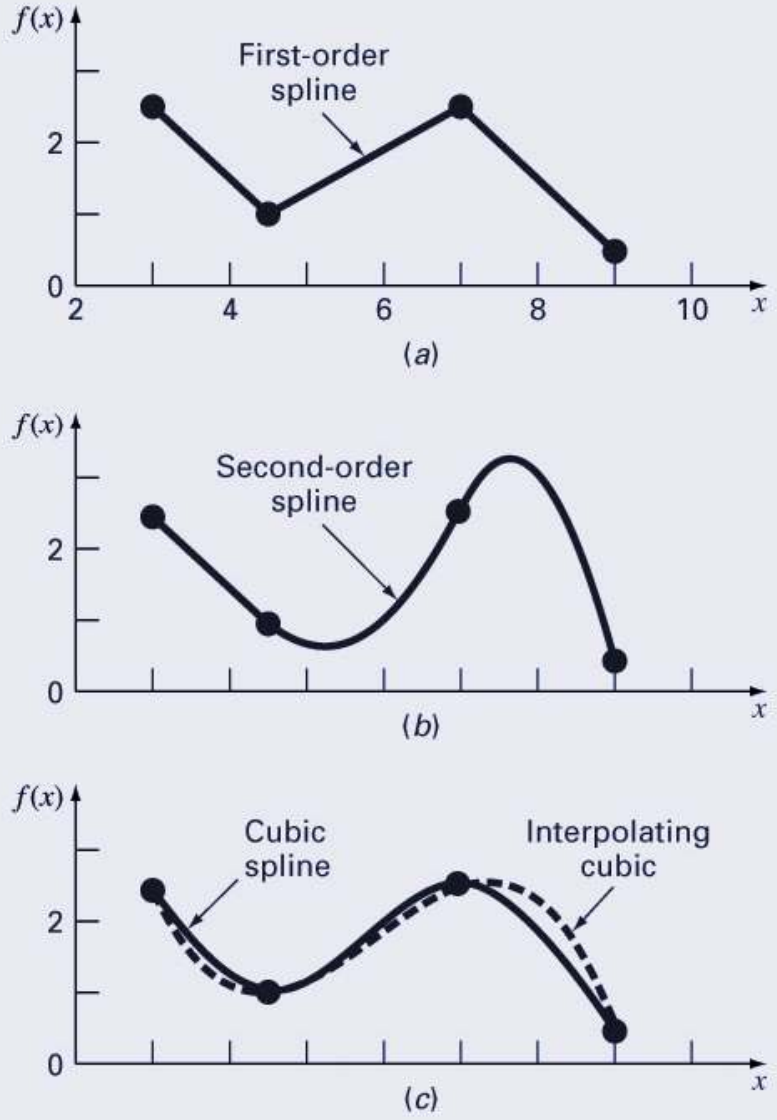
Spline Development

a) First-order splines find straight-line equations between each pair of points that
   - Go through the points

b) Second-order splines find quadratic equations between each pair of points that
   - Go through the points
   - Match first derivatives at the interior points

c) Third-order splines find cubic equations between each pair of points that
   - Go through the points
   - Match first and second derivatives at the interior points

   Note that the results of cubic spline interpolation are different from the results of an interpolating cubic.



Cubic Splines

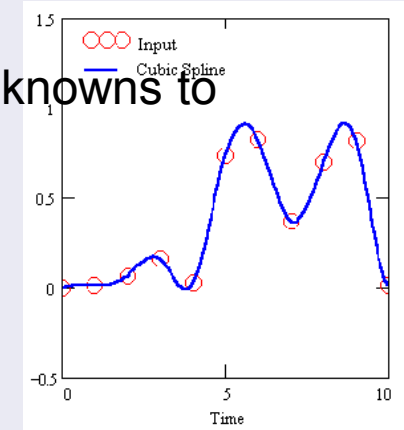
- Merits : simple and smooth
  - Linear (*1st order*) splines have discontinuous first derivatives
  - Quadratic (*2nd order*) splines have discontinuous second derivatives

- Quartic or higher-order splines tend to exhibit the instabilities inherent in higher order polynomials
  - ill-conditioning or oscillations
  - Linear → Quadratic → Cubic → Quartic → Quintic

- In general, the *i* th spline function for a cubic spline can be written as:

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

  - For *n* data points, there are *(n-1)* intervals and thus *4(n-1)* unknowns to evaluate to solve all the spline function coefficients.



Solving Spline Coefficients

- One condition requires that the spline function goes through the first and last point of the interval, yielding *2(n-1)* equations of the form:

$$s_i(x_i) = f_i \Rightarrow a_i = f_i$$
$$s_i(x_{i+1}) = f_{i+1} \Rightarrow s_i(x_{i+1}) = a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3 = f_{i+1}$$

- Another requires that the first derivative is continuous at each interior point, yielding *(n-2)* equations of the form:

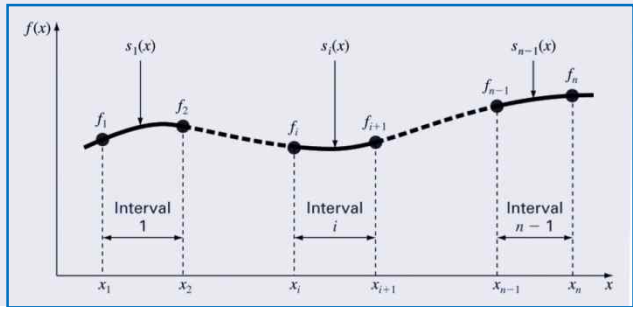
$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \Rightarrow b_i + 2c_i(x_{i+1} - x_i) + 3d_i(x_{i+1} - x_i)^2 = b_{i+1}$$

- A third requires that the *second* derivative is continuous at each interior point, yielding *(n-2)* equations of the form:

$$s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \Rightarrow 2c_i + 6d_i(x_{i+1} - x_i) = 2c_{i+1}$$

- These give *(4n-6)* equations, *(4n-4)* unknowns

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



$$s(x)\begin{cases} s_1(x) = a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3, & x_1 \le x \le x_2 \\ \vdots \\ s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, & x_i \le x \le x_{i+1} \\ \vdots \\ s_{n-1}(x) = a_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3, & x_{n-1} \le x \le x_n \end{cases}$$

$$\begin{cases} s_i(x_i) = f_i \rightarrow a_i = f_i \\ \\ s_i(x_{i+1}) = f_{i+1} \rightarrow f_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = f_{i+1} \end{cases}$$

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \rightarrow b_i + 2c_i h_i + 3d_i h_i^2 = b_{i+1}$$

$$s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \rightarrow c_i + 3d_i h_i = c_{i+1}$$

$$s_1''(x_1) = s_{n-1}''(x_n) = 0 \rightarrow 2c_1 = 2c_{n-1} + 6d_{n-1} h_{n-1} = 0$$

$$\begin{cases} c_i + 3d_i h_i = c_{i+1} \rightarrow d_i = (c_{i+1} - c_i)/3h_i \\ f_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = f_i + b_i h_i + \frac{h_i^2}{3}(2c_i + c_{i+1}) = f_{i+1} \rightarrow b_i = \frac{f_{i+1} - f_i}{h_i} - \frac{h_i}{3}(2c_i + c_{i+1}) \\ b_i + 2c_i h_i + 3d_i h_i^2 = b_i + h_i(c_i + c_{i+1}) = b_{i+1} \end{cases}$$

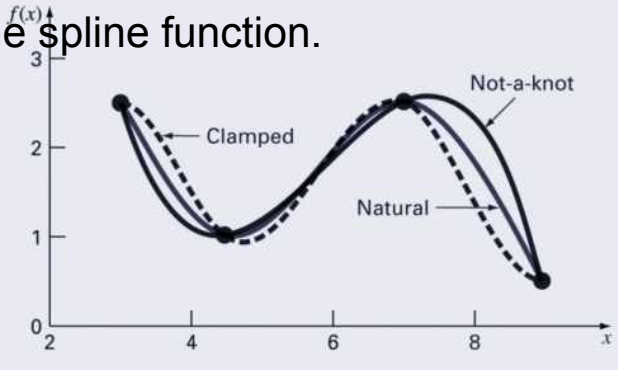
$$\begin{cases} 2c_1 = 0, \\ h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = 3\frac{f_{i+1} - f_i}{h_i} - 3\frac{f_i - f_{i-1}}{h_{i-1}} \\ 2c_n = 0 \end{cases}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ h_1 & 2(h_1 + h_2) & h_2 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 3\frac{f_3 - f_2}{x_3 - x_2} - 3\frac{f_2 - f_1}{x_2 - x_1} \\ \vdots \\ \vdots \\ 3\frac{f_n - f_{n-1}}{x_n - x_{n-1}} - 3\frac{f_{n-1} - f_{n-2}}{x_{n-1} - x_{n-2}} \\ 0 \end{bmatrix}$$

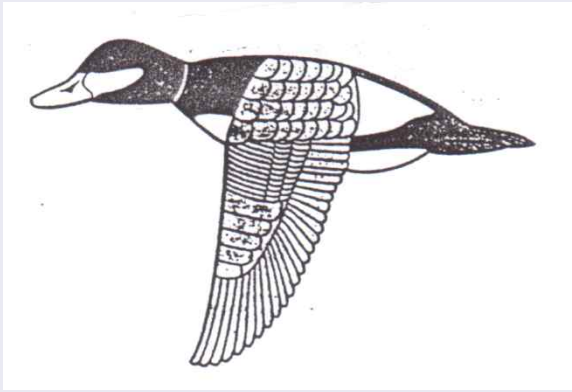
$c_i \rightarrow d_i$, b_i ($a_i = f_i$)

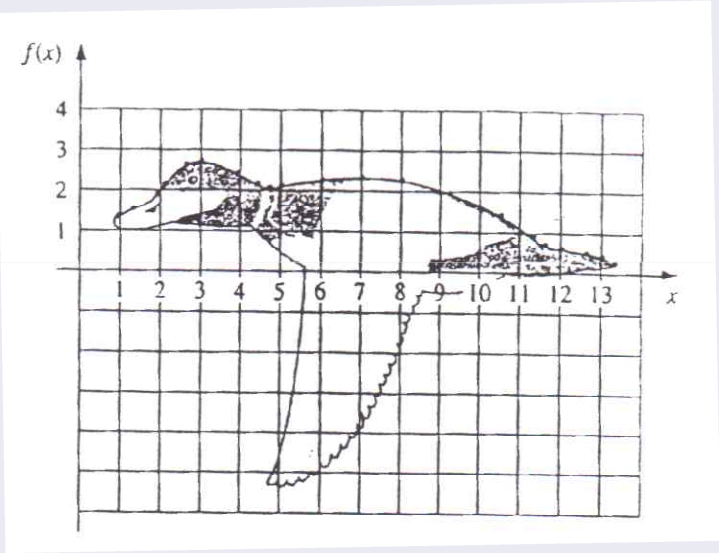
Two Additional Equations

- There are several options for the final two equations:
  - Natural end conditions (free boundary)
    - set the second derivative at the two end points zero
    - ($s''(x_0) = s''(x_n) = 0$) (중간이 구부러지는 막대기)
  - Clamped end conditions (clamp boundary)
    - set the first derivatives at the two end points are known values
    - Hermite(에르미트) interpolation : when we know the $f(x)$ and $f'(x)$ for all i
    - ($s'(x_0) = f'(x_0)$, $s'(x_n) = f'(x_n)$)
  - "Not-a-knot" end conditions : (knot : (x, f(x))
    - the first two and the last two intervals share the same spline function.
    - $s_1(x) = s_2(x)$, $s_{n-2}(x) = s_{n-1}(x)$
    - MATLAB default



(ex) 오리 윗부분 윤곽선 (Profile) 근사



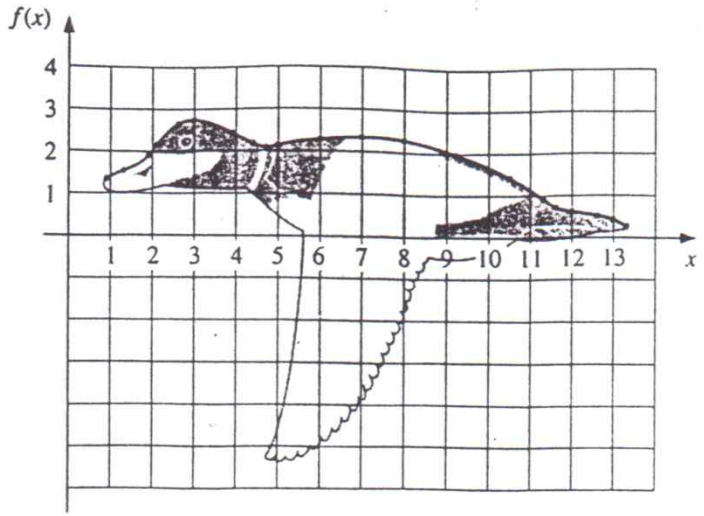


x	0.9	1.3	1.9	2.1	2.6	3.0	3.9	4.4	4.7	5.0	6.0	7.0	8.0	9.2	10.5	11.3	11.6	12.0	12.6	13.0	13.3
$f(x)$	1.3	1.5	1.85	2.1	2.6	2.7	2.4	2.15	2.05	2.1	2.25	2.3	2.25	1.95	1.4	0.9	0.7	0.6	0.5	0.4	0.25

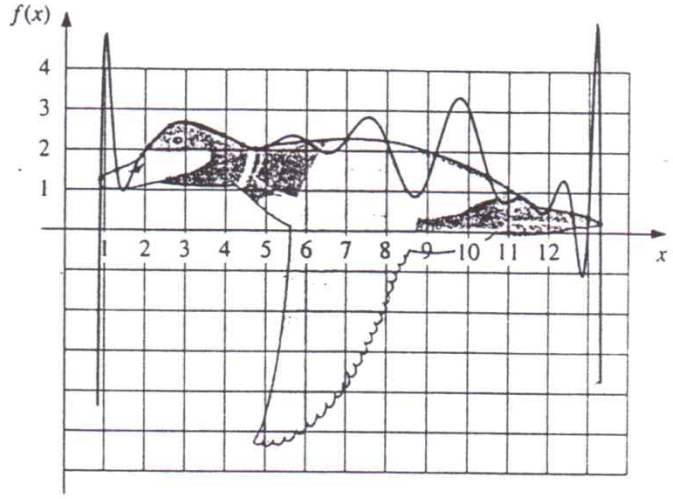
• 계수 산출.

j	x_j	a_j	b_j	c_j	d_j
0	0.9	1.3	5.40	0.00	−0.25
1	1.3	1.5	0.42	−0.30	0.95
2	1.9	1.85	1.09	1.41	−2.96
3	2.1	2.1	1.29	−0.37	−0.45
4	2.6	2.6	0.59	−1.04	0.45
5	3.0	2.7	−0.02	0.50	0.17
6	3.9	2.4	−0.50	−0.03	0.08
7	4.4	2.15	−0.48	0.08	1.31
8	4.7	2.05	−0.07	1.27	−1.58
9	5.0	2.1	0.26	−0.16	0.04
10	6.0	2.25	0.08	−0.03	0.00
11	7.0	2.3	0.01	−0.04	−0.02
12	8.0	2.25	−0.14	−0.11	0.02
13	9.2	1.95	−0.34	−0.05	−0.01
14	10.5	1.4	−0.53	−0.10	−0.02
15	11.3	0.9	−0.73	−0.15	1.21
16	11.6	0.7	−0.49	0.94	−0.84
17	12.0	0.6	−0.14	−0.06	0.04
18	12.6	0.5	−0.18	0.00	−0.45
19	13.0	0.4	−0.39	−0.54	0.60
20	13.3	0.25			

→



cf. Lagrange 보간식 : 차수 20 → 지옥.



Piecewise Interpolation in MATLAB

- MATLAB has several built-in functions to implement piecewise interpolation.

- The first is spline:

```
yy=spline(x, y, xx)
```

  - This performs cubic spline interpolation, generally using not-a-knot conditions.
  - If y contains two more values than x has entries, then
    - clamped : the first and last value in y are used as the derivatives at the end points

MATLAB Example

- Generate data:

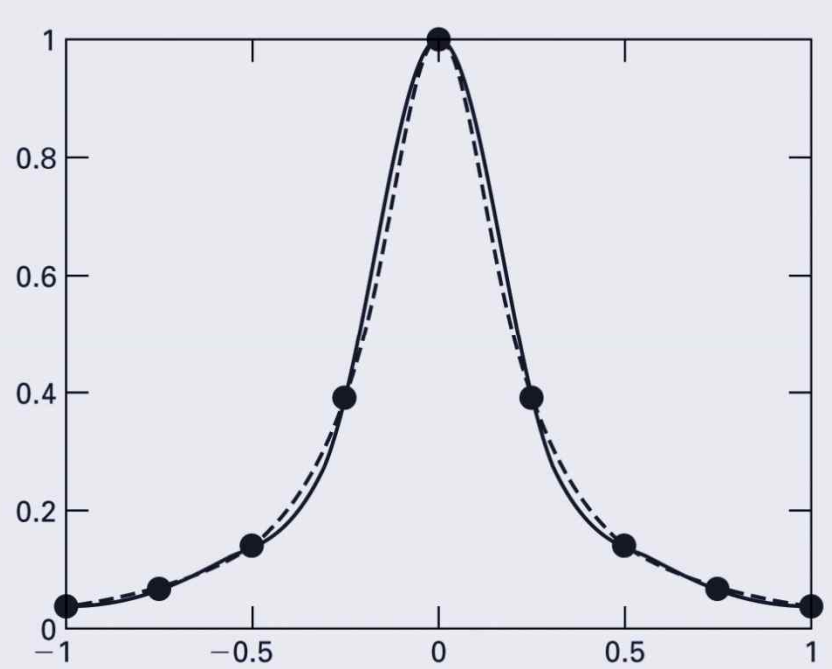
```
x = linspace(-1, 1, 9);
y = 1./(1+25*x.^2);
```

- Calculate 100 model points and determine not-a-knot interpolation

```
xx = linspace(-1, 1);
yy = spline(x, y, xx);
```

- Calculate actual function values at model points and data points, the 9-point not-a-knot interpolation (solid), and the actual function (dashed),

```
yr = 1./(1+25*xx.^2)
plot(x, y, 'o', xx, yy, '-', xx, yr, '--')
```

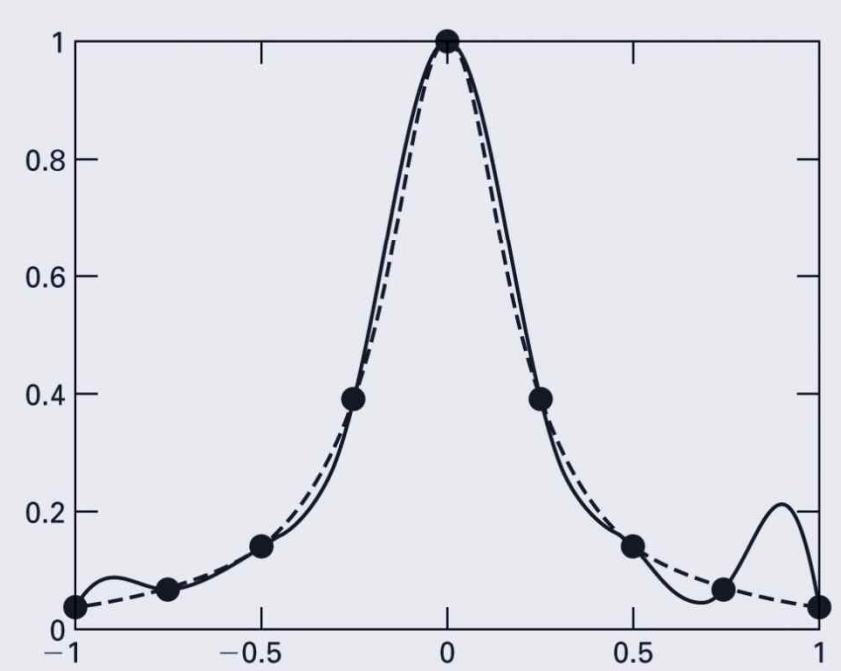


Clamped Example

- Generate data w/ first derivative information:
  x = linspace(-1, 1, 9);
  y = 1./(1+25*x.^2);
  yc = [1 y -4]

- Calculate 100 model points and determine clamped interpolation
  xx = linspace(-1, 1);
  yyc = spline(x, **yc**, xx);

- Calculate actual function values at model points and data points, the 9-point clamped interpolation (solid), and the actual function (dashed),
  yr = 1./(1+25*xx.^2)
  plot(x, y, 'o', xx, **yyc**, '-', xx, yr, '--')



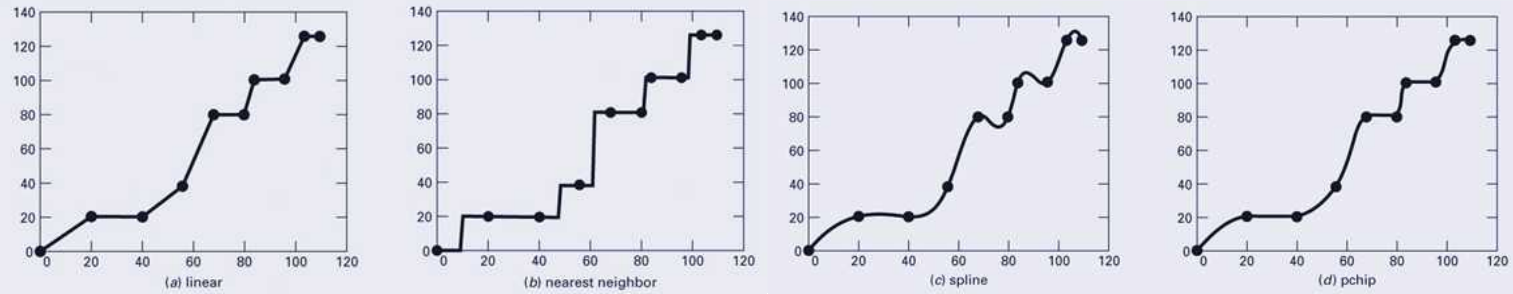
yc = [1 y -4] % the first and last value in y are used as the derivatives at the end points

MATLAB's interp1 Function

- While spline can only perform cubic splines, MATLAB's interp1 function can perform several different kinds of interpolation:

  yi = interp1(x, y, xi, 'method')

  - x , y : the original data
  - xi : the points at which to interpolate
  - 'method' : the desired method:
    - 'linear' : connects the points with straight lines
    - 'nearest' : nearest neighbor interpolation
    - 'spline' : not-a-knot cubic spline interpolation
    - 'pchip' or 'cubic' : piecewise cubic Hermite interpolation (PCHIP)

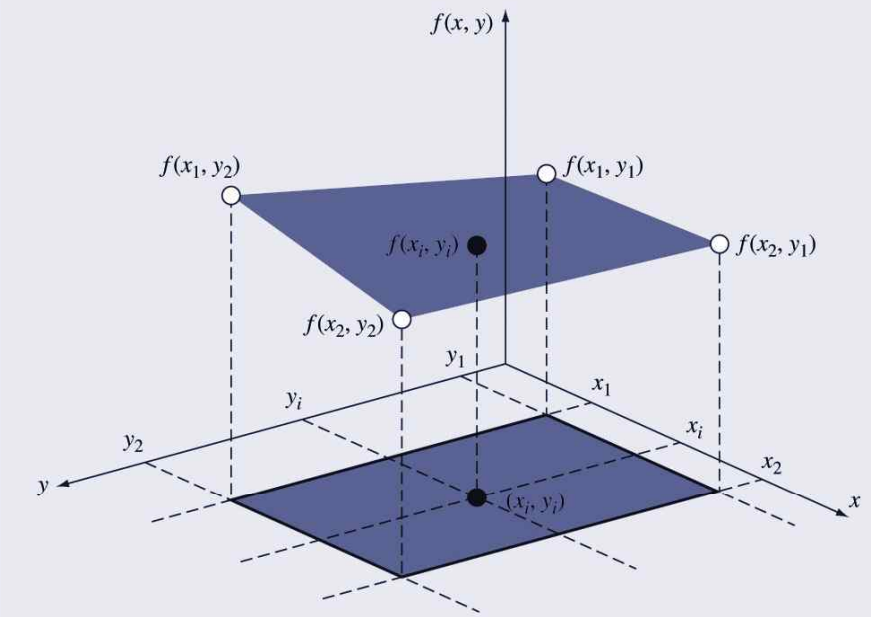


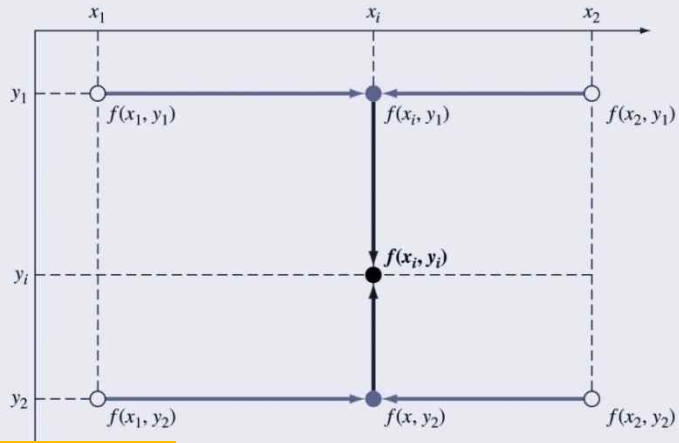
Multidimensional Interpolation

- The interpolation methods for one-dimensional problems can be extended to multidimensional interpolation.

- Example : bilinear interpolation using Lagrange-form equations:

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





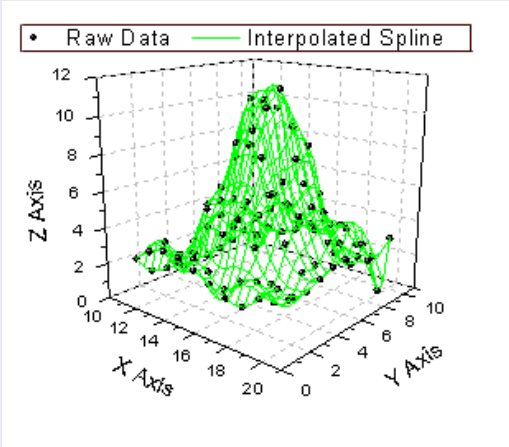
$(x-x_1)(x-x_2)(y-y_1)(y-y_2)$

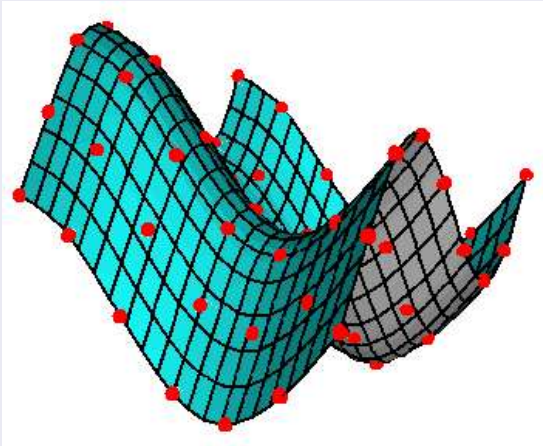
Multidimensional Interpolation in MATLAB

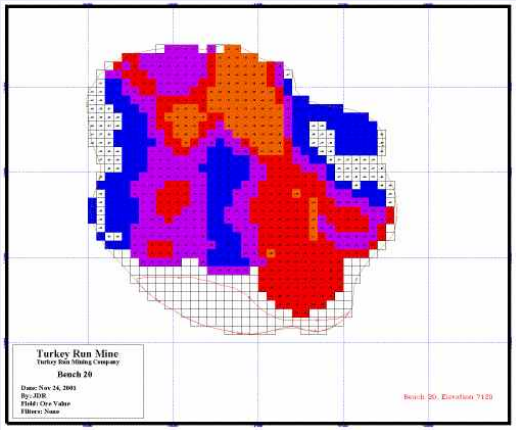
- For two- and three-dimensional piecewise interpolation:

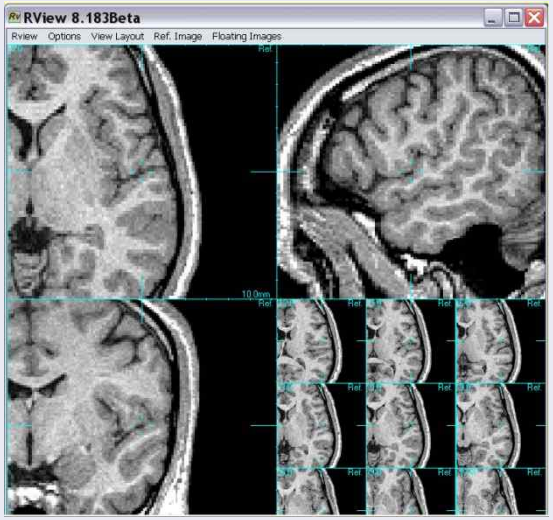
  zi = interp2(x, y, z, xi, yi, ‘method’)
  vi = interp3(x, y, z, v, xi, yi, zi, ‘method’)

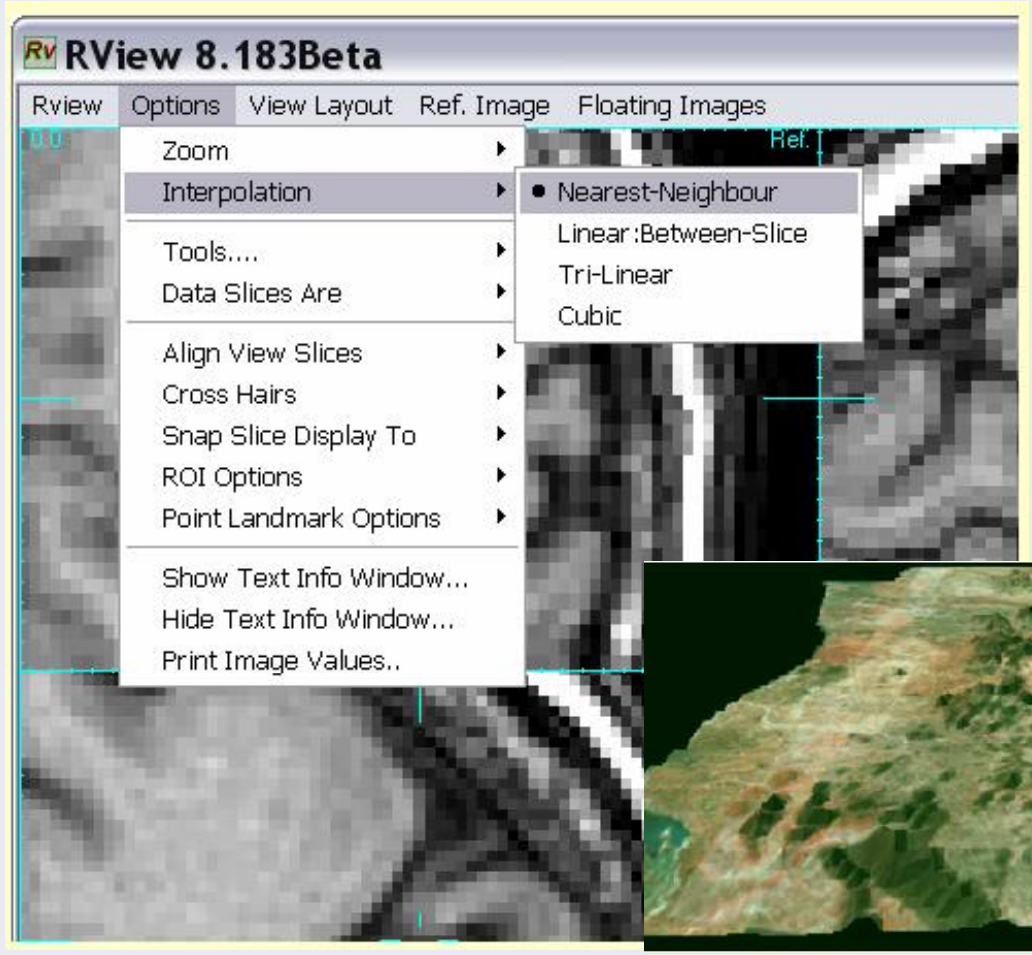
  – ‘method’ : ‘nearest’, ‘linear’, ‘spline’, ‘pchip’, or ‘cubic’



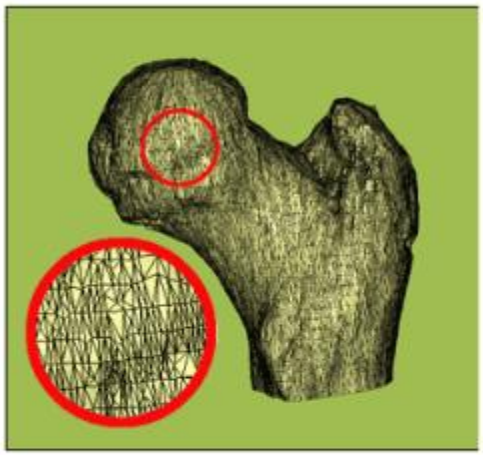








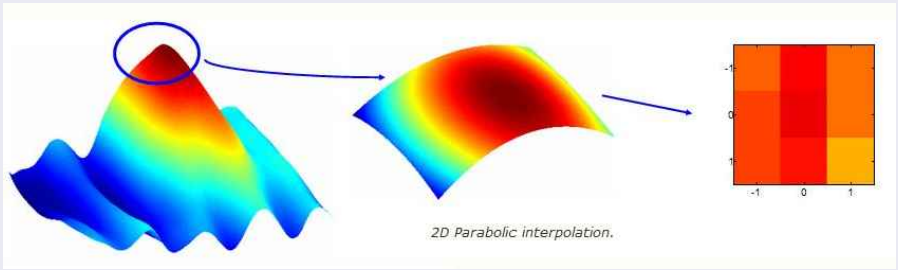
The last frame of an uplift sequence generated by 3D interpolation



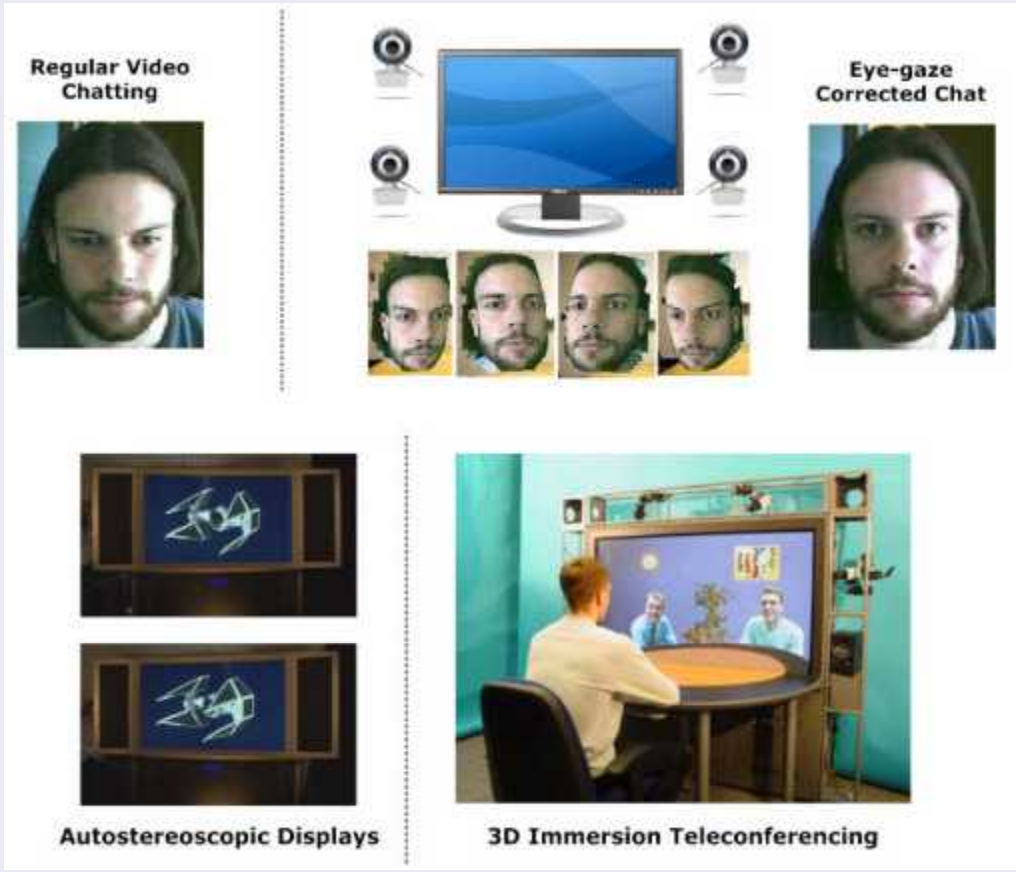
Contour interpolation

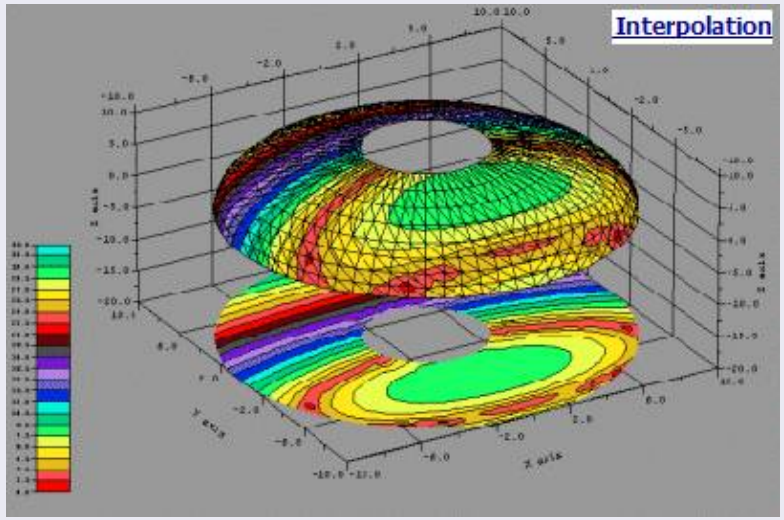
2D interpolation in the plane of the im
is smoothly expanded in the third dir

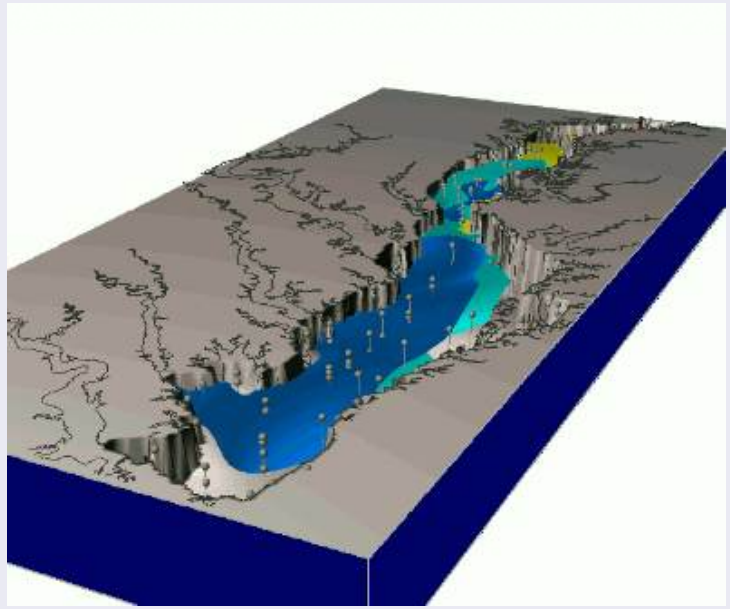
... techniques and **2D interpolation ...**



2D Parabolic interpolation.


Regular Video Chatting
Eye-gaze Corrected Chat
Autostereoscopic Displays
3D Immersion Teleconferencing


Interpolation



THE END

Homework : MATLAB 예제
Report : 18-1,18-11