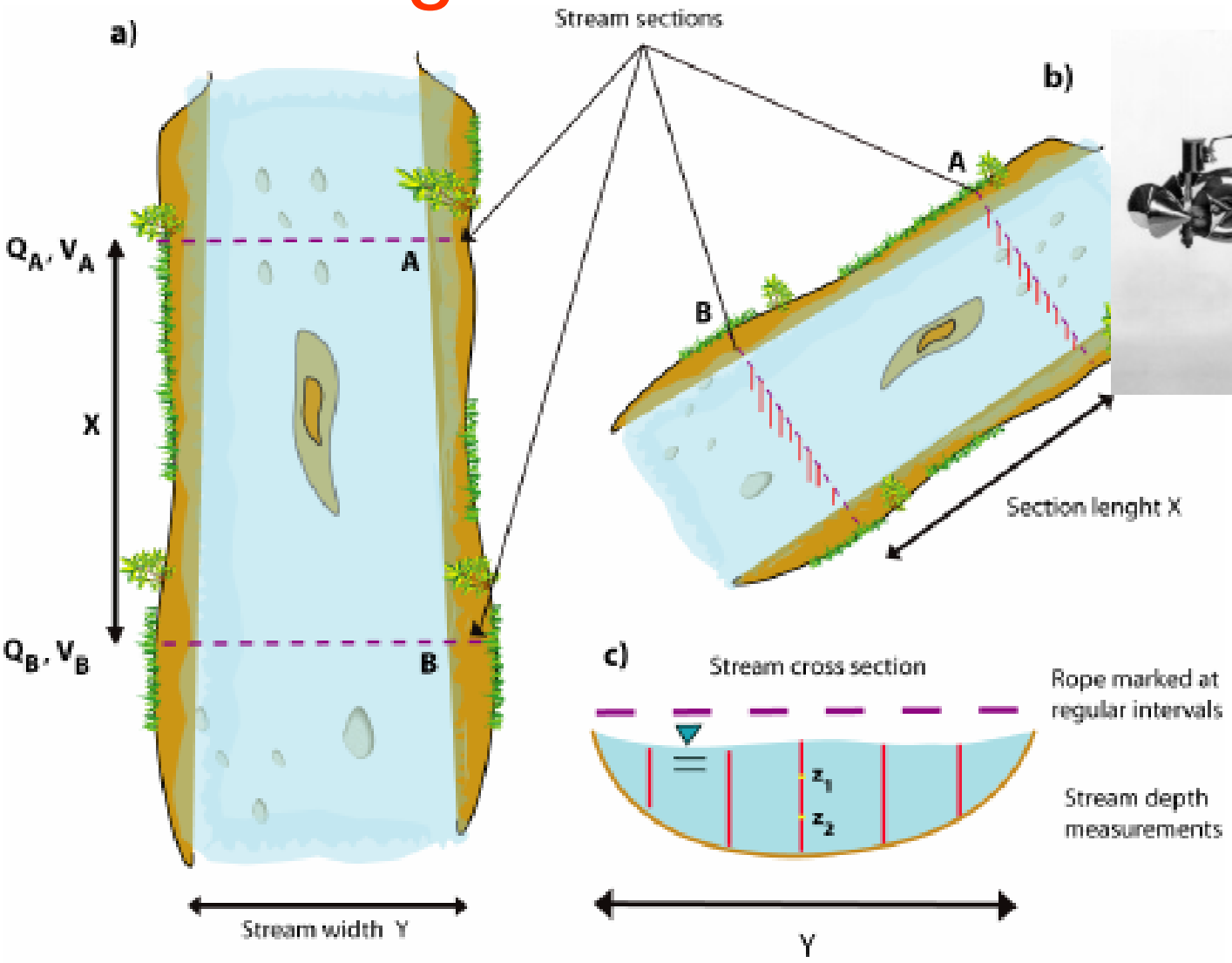
# Chapter 3

# Numerical Differentiation, Interpolation, and Integration

Instructor: Dr. Ming Ye

# Measuring Flow in Natural Channels



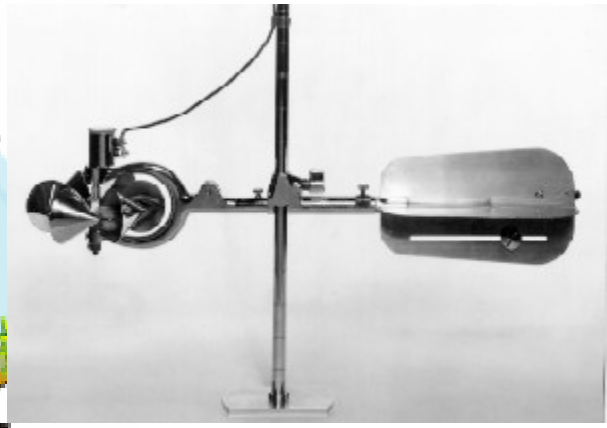
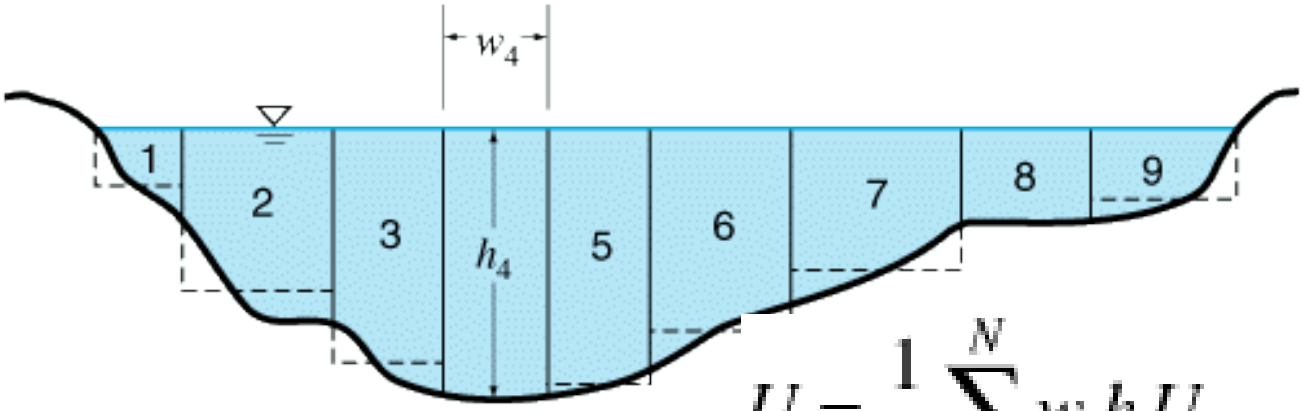



**Figure 3.** Schematic of the river reach, the two cross-sections and measurements obtained at each location. (a) Top view of the stream cross-section A, and B, showing the section length (X), stream width (Y), discharge ($Q_A$ and $Q_B$) and velocities ($V_A$ and $V_B$) at A and B. (b) 3d view of the stream section showing cross sections A and B and measurements of stream depth at different locations along the stream bed. (c) Stream cross sectional view of the stream depth measurements showing an example of the two velocity observation points ($z_1$ and $z_2$) to be taken at each cross section segment.

Mean-Section Method




$$U = \frac{1}{A}\int_0^w \int_0^h u(y,z)\,dz\,dy$$

$$U = \frac{1}{A}\sum_{i=1}^{N} w_i h_i U_i$$

$$Q = UA = \sum_{i=1}^{N} w_i h_i U_i$$

(1) Divide the stream into a number of rectangular elements
(2) Use current meter to measure speed of the flow in each rectangular. The velocity is approximately the average velocity for that rectangular.
(3) Multiply the average velocity by the area of the rectangular.
(4) Sum across the stream.
(5) Divide the sum by total area of the cross-section.

# Outline of This Chapter

- If the function is smooth, analytically known, and can be evaluated anywhere
  - Finite difference with Taylor series expansion
- If only certain function values of the function are known
  - Interpolation (if the values are sufficiently smooth) or curve fitting (if the values are noisy)
  - Numerical differentiation: a computer program consists of basic arithmetic operations and elementary functions, each of whose derivatives is easily computed.
- Numerical integration
- MATLAB functions

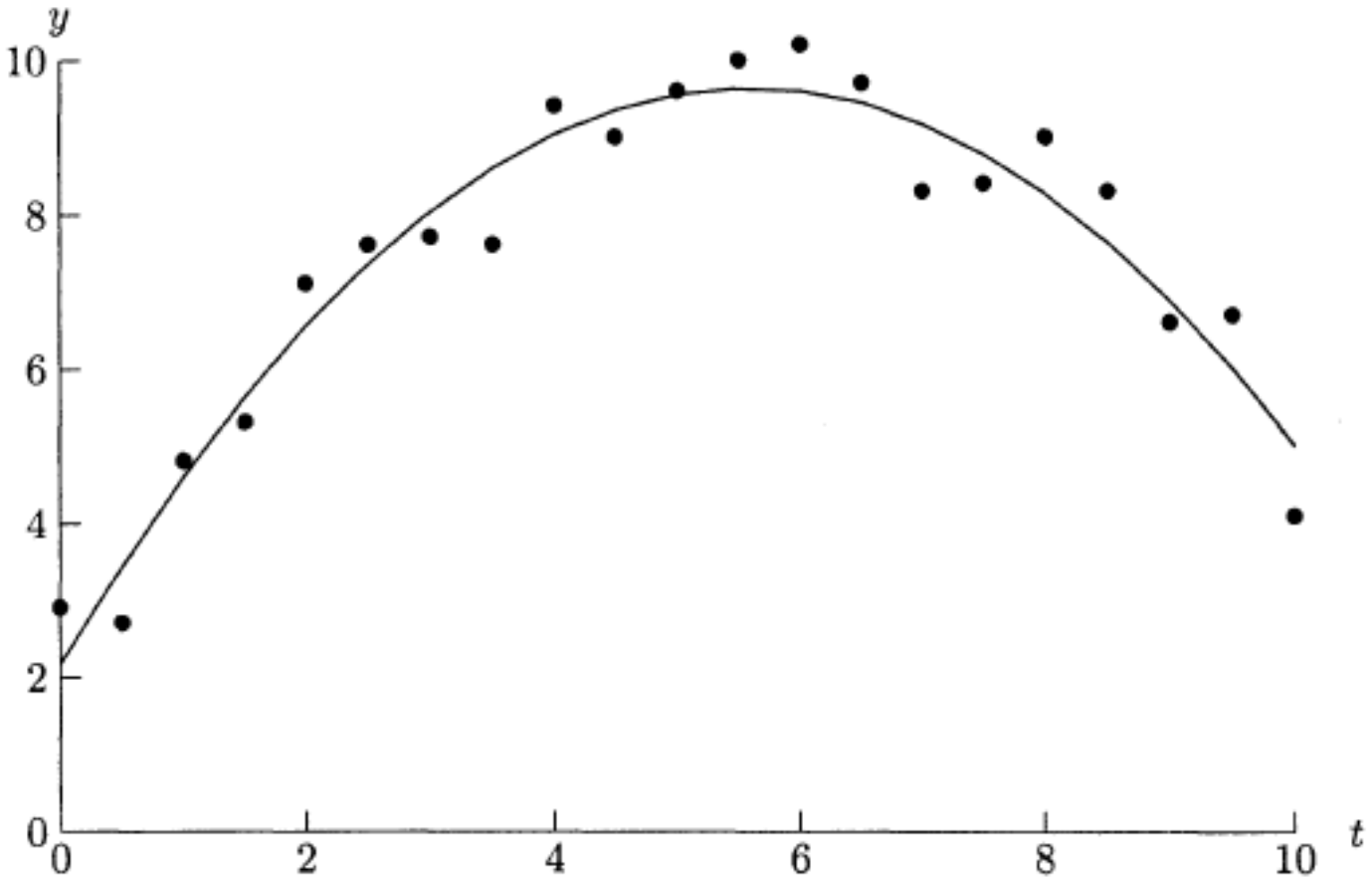



**Figure 3.1:** Least squares fit of quadratic polynomial to given data.

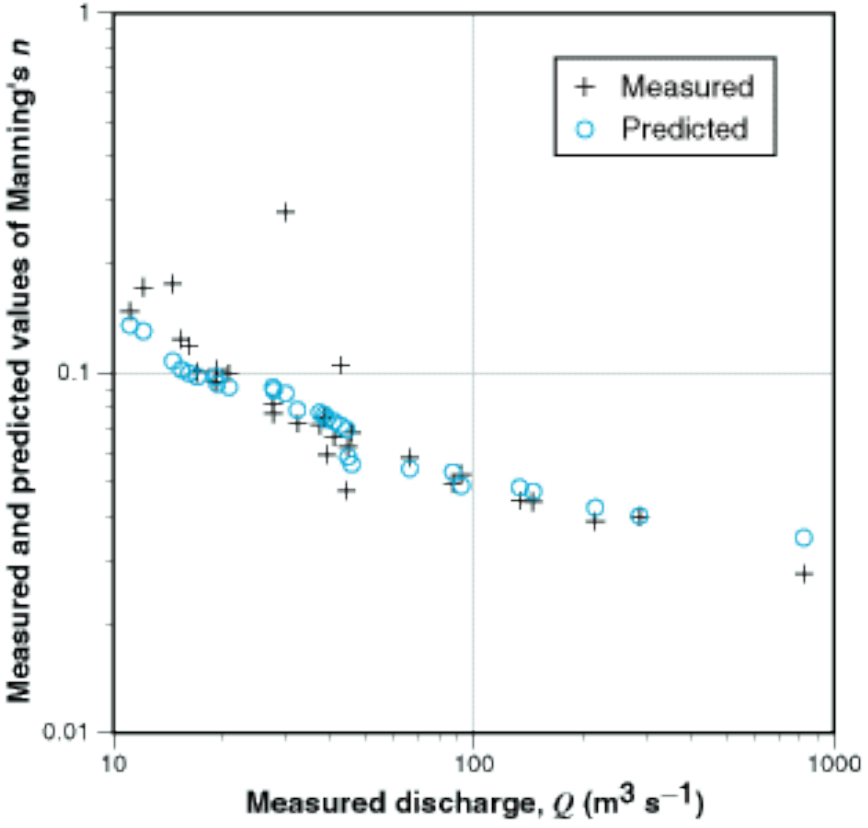
- If all the 21 data points were fit exactly by a polynomial of degree 20, then the derivative of that polynomial would be quite erratic, changing sign many times, and moreover the derivative would be highly sensitive to small change in the data.
- In sharp contrast, the derivative of the approximating quadratic polynomial is well behaved and relatively insensitive to change in the data.

# Determine the Manning's Coefficient and Estimate Discharge

$$U = \frac{\kappa}{n} R_H^{2/3} S^{1/2}$$

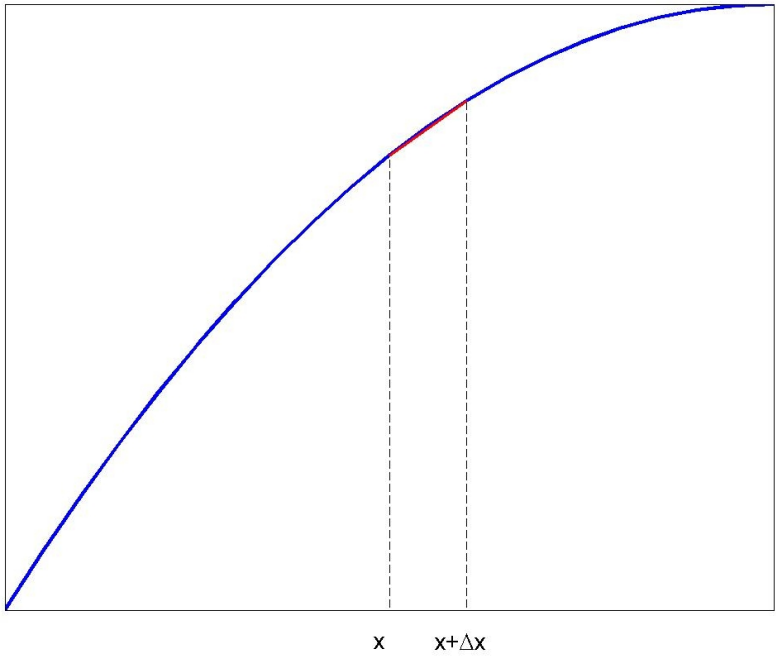
- Once we know mean velocity, slope, and depth (or hydraulic radius), we can determine the value of Manning's *n* using Manning's equation.

- Once we know the value of Manning's *n*, we can use the Manning's equation to estimate stream velocity and discharge, when direct hydraulic measurements are absent.



# Finite Difference

Mathematical definition of a derivative




$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

In numerical differentiation, instead of taking the limit as Δx approaches zero, Δx is allowed to have some **small** but finite value.

$$f'(x) \cong \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Pick a small value Δx; evaluate f(x+Δx); you probably have f(x) already evaluated, but if not, do it too; finally apply the equation.

# Approximation Error

- Thought of geometrically, this estimate of a derivative is the slope of a linear approximation to the function *f* over the interval $\Delta x$.
- Applied uncritically, the above procedure is almost guaranteed to produce inaccurate results.
- How well a linear approximation works depends on the shape of the function *f* and the size of the interval $\Delta x$.
- Truncation error and roundoff error

# Truncation Error: Taylor Series Approach

Taylor series expansion

$$f(x+\Delta x) = f(x) + f'(x)\Delta x + \frac{f''(x)}{2}(\Delta x)^2 + \ldots + \frac{f^{(n)}(x)}{n!}(\Delta x)^n + R_{n+1}$$

The remainder

$$R_{n+1} = \frac{f^{(n+1)}(\xi)}{(n+1)!}(\Delta x)^{n+1} \qquad \text{for} \quad x < \xi < x+\Delta x$$

The error produced by truncating the Taylor series after the $f^{(n)}$-th term is given by $R_{n+1}$ and is said to be of order $(\Delta x)^{n+1}$ or $O((\Delta x)^{n+1})$. Although in general we don't know the value of $f^{(n+1)}(\xi)$, it is evident that the smaller $\Delta x$, the smaller the error [but see Box 3.1 for additional information].

# Truncation Error of Approximating f’

If we truncate the Taylor series after the first-derivative term,

$$f(x \pm \Delta x) = f(x) \pm f'(x)\Delta x + O(\Delta x)^2$$

Forward difference

$$f'(x) = \left[ f(x + \Delta x) - f(x) \right] / \Delta x + O(\Delta x)$$

Backward difference

$$f'(x) = \left[ f(x) - f(x - \Delta x) \right] / \Delta x + O(\Delta x)$$

Both expressions have a truncation error of $O(\Delta x)$.

# Central Difference Approximation

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) + O((\Delta x)^3)$$

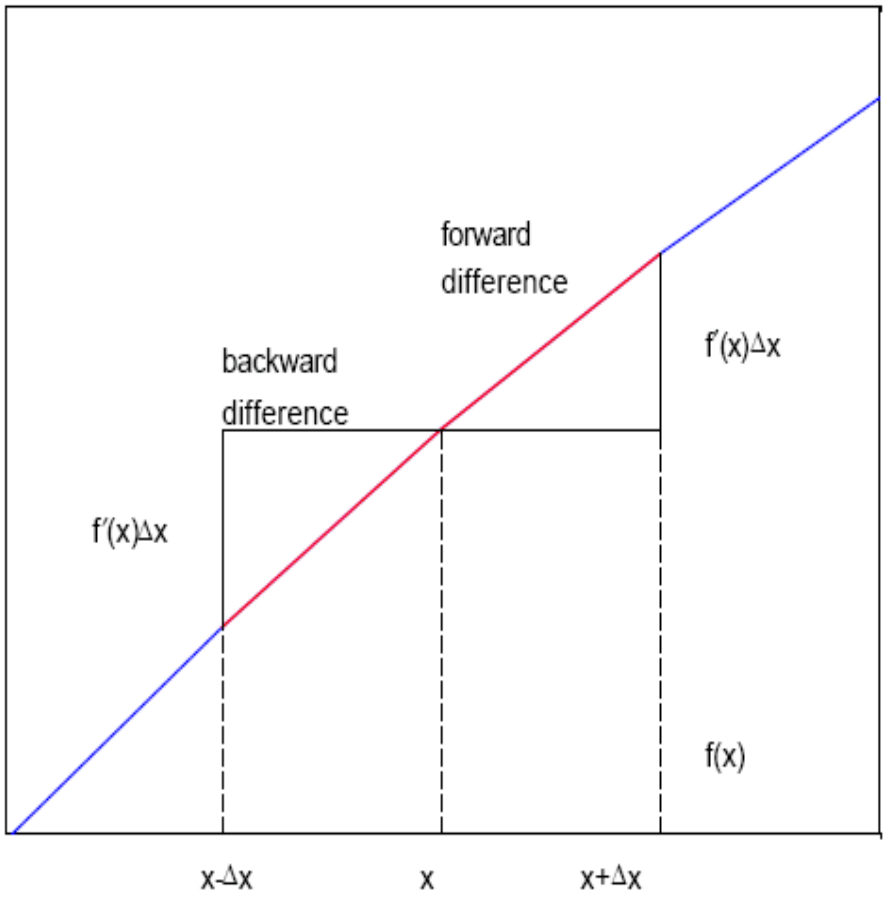
$$f(x-\Delta x) = f(x) - \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) + O((\Delta x)^3)$$

Subtracting these leaves

$$f(x+\Delta x) - f(x-\Delta x) = 2\Delta x f'(x) + O((\Delta x)^3)$$

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2(\Delta x)} + O((\Delta x)^2)$$

$$f'(x) = \left[f(x+\Delta x) - f(x)\right] / \Delta x + O(\Delta x)$$




The higher order approximation yield a more accurate estimate of the derivative, but

- Computational cost is higher.
- Not applicable at the domain boundaries where a forward or backward estimate is necessary.

# Higher-order Derivatives

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) + \frac{(\Delta x)^3}{6} f'''(x) + O((\Delta x)^4)$$

$$f(x-\Delta x) = f(x) - \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) - \frac{(\Delta x)^3}{6} f'''(x) + O((\Delta x)^4)$$

Add the two terms

$$f(x+\Delta x) + f(x-\Delta x) = 2f(x) + (\Delta x)^2 f''(x) + O((\Delta x)^4)$$

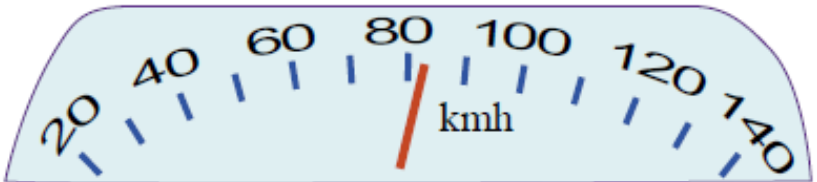
$$f''(x) = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{(\Delta x)^2} + O((\Delta x)^2)$$

$$f''(x) = \frac{f(x-\Delta x) - 2f(x) + f(x+\Delta x)}{(\Delta x)^2} + O((\Delta x)^2)$$

Can you get this from finite difference?

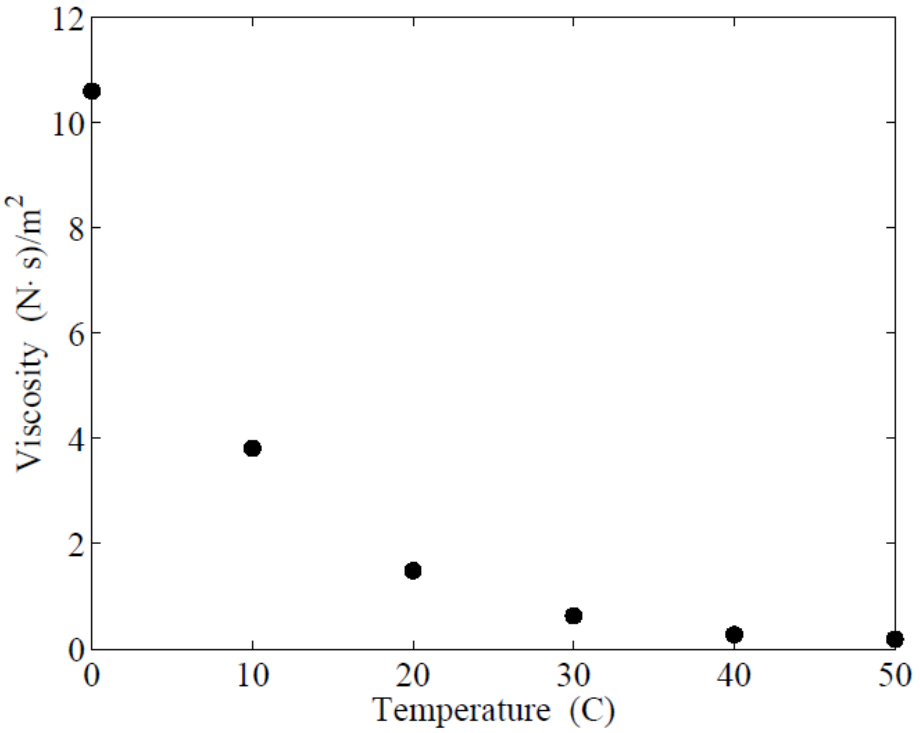
# Interpolation

- Using Taylor series expansions to derive finite difference formulas becomes increasingly cumbersome for approximations of increasingly high accuracy or higher-order derivatives.
- We next consider an alternative method based on polynomial interpolation that is not only more convenient but will also more readily permit generalization such as using unequally spaced points.
- Interpolation is an important part of many numerical methods.
- Interpolating polynomials are the building blocks of many other numerical methods such as numerical integration and finite element.




# Interpolation

Interpolation is concerned with the use of interpolation to approximate a function that is defined by a table of data.




Plot of viscosity of glycerin at different temperatures.

Viscosity at temperature of 22$^{o}$C?

- 1.319: Linear interpolation using the viscosity values of 20 and 30 $^{o}$C
- 1.203: Quadratic interpolation using more data
- The difference is about 10%.

# Polynomials

Polynomials are used to construct interpolating functions in two primary ways:

- The degree of the polynomial is increased to match greater number of points from the discrete data set.
- An interpolating function is created by assembling a set of low degree polynomials defined over subintervals of the domain.

# Interpolation

We will cover the following topics

- Basic ideas
- Interpolating polynomials of arbitrary degree
  - Monomial basis
  - Lagrange basis
  - Newton basis
- Piecewise polynomial interpolation
  - Linear
  - Hermite polynomials
  - Cubic splines
- MATLAB's built-in interpolation routines

# Basic Ideas

- In the one-dimensional case, a finite set of data $(x_i,y_i)$, i=1, …,n is given as discrete samples of some known or hard-to-evaluation function y=f(x).
- Interpolation involves constructing and then evaluating an interpolation function, or *interpolant*, y=F(x) at values of x that may or may not be in the $(x_i,y_i)$ data set.
- The interpolation function F(x) is determined by requiring that it pass through the known data $(x_i,y_i)$.
- When $x \neq x_i$, the F(x) should also be a good approximation to f(x), which created the tabular data in the first place.
- f(x) may be analytically unknown; the $(x_i,y_i)$ data represent the input and output values of n measurements.
- f(x) may be analytically known but difficult and tedious to evaluate, especially by manual calculations or computer memory is limited.

# Basic Ideas

- In its most general form, interpolation involves determining the coefficients, $a_1$, $a_2$,… $a_n$ in the linear combination of *n* basis functions, $\Phi(x)$, that constitute the interpolant

  $F(x)= a_1\Phi_1(x)+a_2\Phi_2(x)+\ldots+a_n\Phi_n(x)$

  such that $F(x_i)=y_i$ for i=1,…,n.

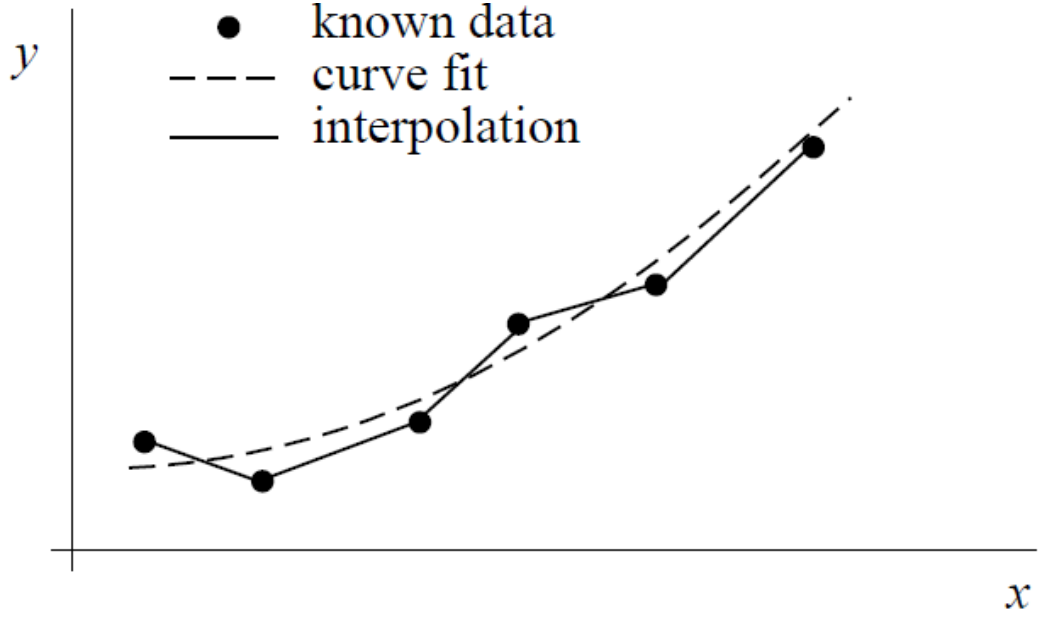
- The basis function may be polynomial

  $F(x)= a_1+a_2x+a_3x^2+\ldots+a_nx^{n-1}$

  or trigonometric

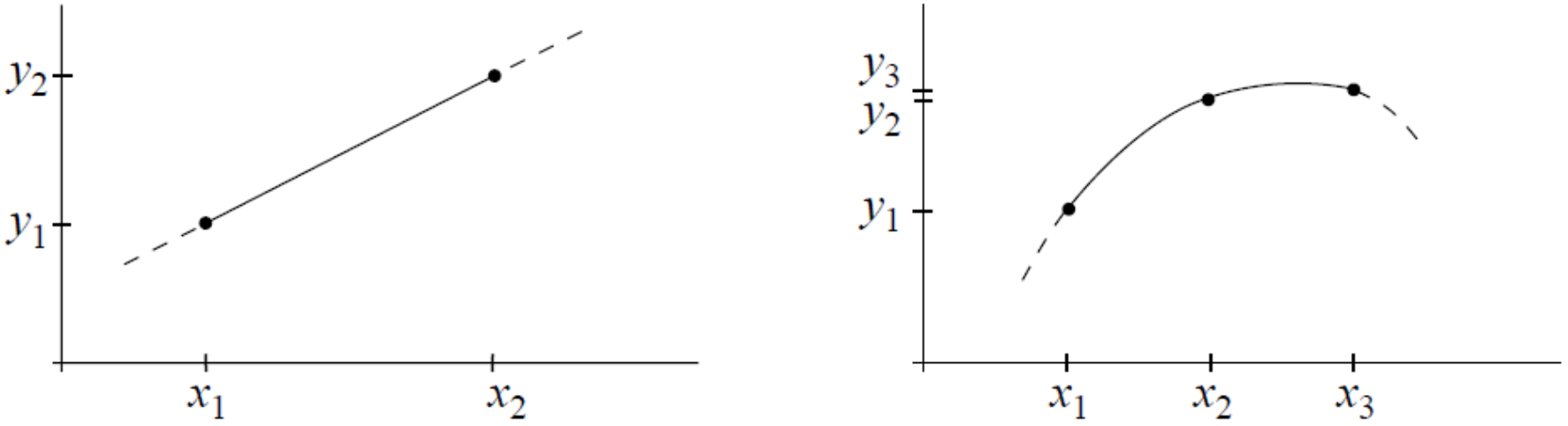
  $F(x)= a_1+a_2e^{ix}+a_3e^{2ix}+\ldots+a_ne^{(n-1)ix}$

# Interpolation versus Curve Fitting




- Interpolation
  - The interpolation function passes exactly through each of the known points.
  - It assume that the data have no uncertainty.
- Curve fitting
  - The approximating function passes near the data points, but (usually) not exactly through them.
  - Data uncertainty is recognized.

# Interpolation and Extrapolation

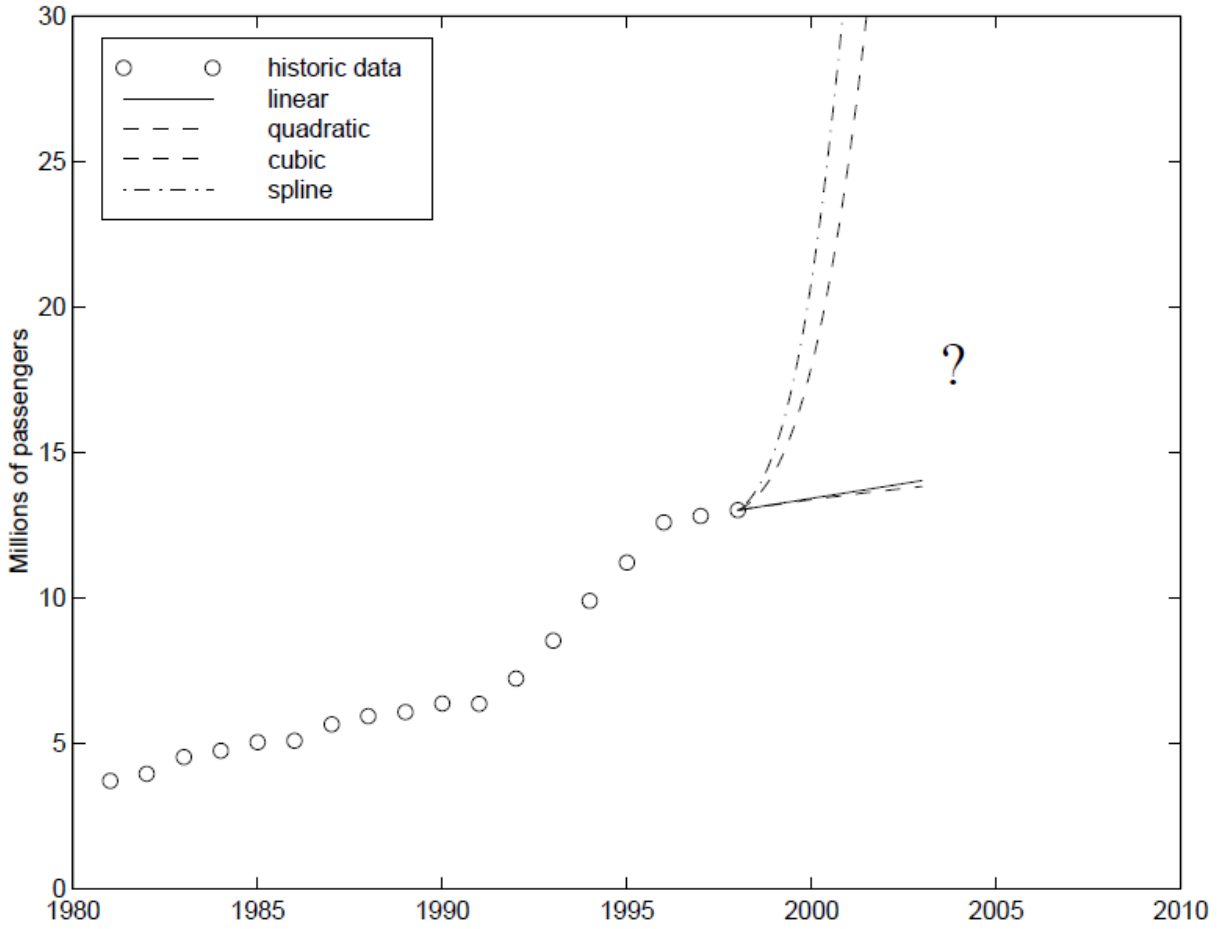



- Interpolation involves the construction and evaluation of an approximating function within the range of the independent variable of the given data set.
- Extrapolation is the evaluation of the interpolating function outside the range of the given independent variable.

# Extrapolation to Predict the Future

The extrapolated values at the first year after the known data are similar, but the values at the fifth year are dramatically different.

Four interpolating functions



# Interpolating Polynomials of Arbitrary Degrees

- A set of n data points may be interpolated with polynomials of up to degree n-1.
- Although the algorithms covered allow construction of polynomials with any *n*, in practice, only polynomials of low degree, say, less than 5, are useful.
- As *n* increases, the value of the interpolant between the data points takes on values that may deviate significantly from the nearby known data point. This is the problem of data wiggle.

# Monomial Basis

- Polynomial interpolation involves finding the equation $P_{n-1}(x)$, the unique polynomial of degree n-1 that passes through n known data pairs, or *supporting points*.

$$P_{n-1}(x) = a_1 + a_2 x + \ldots + a_n x^{n-1}$$

- The common representation of a polynomial of degree n-1 is defined in terms of a set of monomial basis functions, $x^0, x^1, \ldots, x^{n-1}$.
- Although the monomials uses a familiar and compact notations, the polynomials can be written in other formats.
- Consider the shifted polynomial with a known offset

$$P_{n-1}(x-\xi) = b_1 + b_2(x-\xi) + \ldots + b_n(x-\xi)^{n-1}$$

- With appropriate definitions of the coefficients, b, the two polynomials can be the same.

# Vandermonde Systems

- The built-in MATLAB routines for evaluating polynomials require that the polynomial be written in decreasing powers of x

$$P_{n-1}(x) = c_1 x^{n-1} + c_2 x^{n-2} + \ldots + c_{n-1} x + c_n$$

- Consider the construction of a quadratic interpolation function that passes through the (x,y) support points (-2,-2), (-1,1), and (2,-1).

$$P_2(x) = c_1 x^2 + c_2 x + c_3$$

$$-2 = c_1(-2)^2 + c_2(-2) + c_3$$

$$1 = c_1(-1)^2 + c_2(-1) + c_3$$

$$-1 = c_1(2)^2 + c_2(2) + c_3$$

$$\begin{bmatrix} 4 & -2 & 1 \\ 1 & -1 & 1 \\ 4 & 2 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ -1 \end{bmatrix}$$

# Vandermonde Systems

- For an arbitrary set of three points $(x_1,y_1)$, $(x_2,y_2)$, and $(x_3,y_3)$, the matrix equation is

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

- The coefficient matrix on the left-hand side is called a vandermonde matrix.
- Build the matrix in two ways:

```
>>x = [-2 -1 2]';
>>A = [x.^2 x ones(size(x))]; Manually
>>A = vander([-2 -1 2]'); MATLAB built-in function
```

# Exercise

Interpolate the fictitious gasoline price (cents) data below

| Year | 1986 | 1988 | 1990 | 1992 | 1994 | 1996 |
|---|---|---|---|---|---|---|
| Price | 133.5 | 132.2 | 138.7 | 141.5 | 137.6 | 144.2 |

- Plot the data and the interpolation function.
- What problem do you observe?
- Why does the problem occur?
- How to solve the problem?

# Lagrange Basis

In a monomial basis the linear interpolating polynomial through $(x_1,y_1)$ and $(x_2,y_2)$ is

$$P_1(x) = c_1 x + c_2$$

where the two coefficient are

How to obtain these coefficients?

$$c_1 = \frac{y_2 - y_1}{x_2 - x_1} \qquad c_2 = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}$$

Substituting the coefficients into $P_1$ and rearranging gives

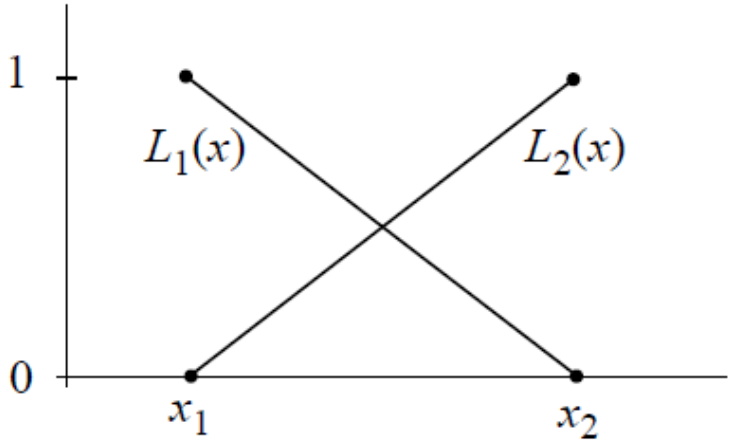
$$P_1(x) = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1} \longrightarrow P_1(x) = y_1 L_1(x) + y_2 L_2(x)$$

This expresses the linear interpolating polynomial in terms of a new pair of basis functions $L_1(x)$ and $L_2(x)$. They are the first-degree Lagrange interpolating polynomials.

# Lagrange Basis

$$P_1(x) = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1} \longrightarrow P_1(x) = y_1 L_1(x) + y_2 L_2(x)$$

$$L_1(x) = \frac{x - x_2}{x_1 - x_2} \qquad L_2(x) = \frac{x - x_1}{x_2 - x_1}$$




If $x=x_i$ is a supporting point, then

$$L_j(x_i) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

where $\delta_{ij}$ is the kronecker delta.

# Quadratic Interpolating Polynomial

Quadratic interpolating polynomial using a Lagrange basis and passing through three points, $(x_1,y_1)$, $(x_2,y_2)$, and $(x_3,y_3)$, is

$$P_2(x) = y_1 L_1(x) + y_2 L_2(x) + y_3 L_3(x)$$

$$L_1(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} \qquad L_2(x) = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}$$

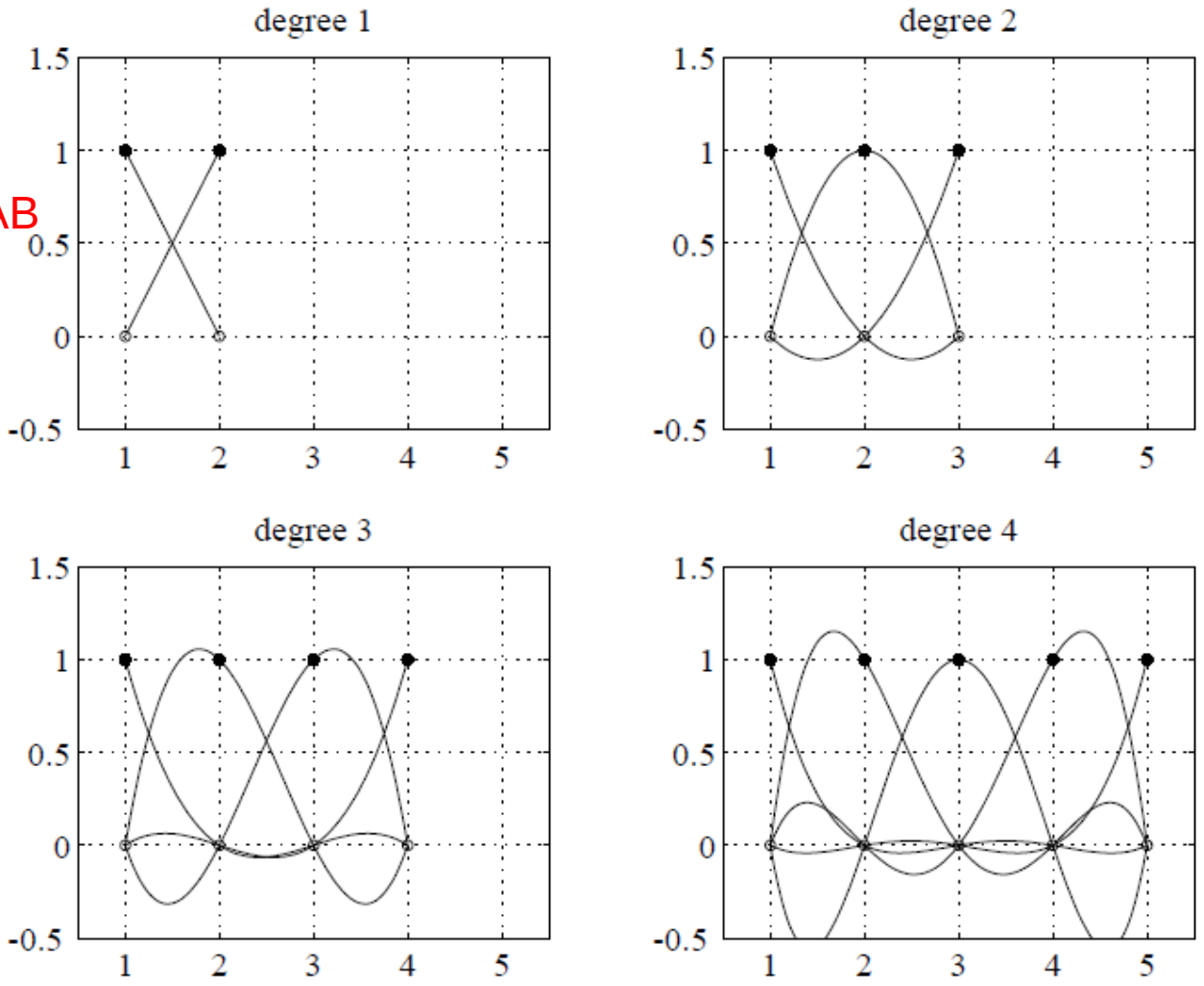
$$L_3(x) = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

# The polynomial of degree n-1 in a Lagrange basis is

$$P_{n-1}(x) = y_1 L_1(x) + y_2 L_2(x) + \mathrm{K} + y_n L_n(x) = \sum_{j=1}^{n} y_j L_j(x)$$

$$L_j(x) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}$$

Create a figure using MATLAB



# Lagrange vs. Monomial Polynomials

Lagrange polynomials have two important advantages over interpolating polynomials expressed in a monomial basis:

- The coefficients of the Lagrange basis are known without solving a system of equations.
- The evaluation of the Lagrange polynomials is much less susceptible to roundoff error.

# Interpolating Polynomials Are Unique

- There is only one polynomial of degree *n-1* passing through *n* support points.
- The uniqueness of polynomials means that the polynomial of degree n-1 passing the n points must be the same whether it is expressed in a monomial basis or a Lagrange basis or any other polynomial basis.
- The coefficients of the individual basis functions will be different, but the values produced by the two polynomial expressions will be identical in exact arithmetic. However, the results in floating-point arithmetic will be different.

# Differentiation using Interpolating Polynomial

- If $P_n(x)$ is a good approximation to f(x) over some range of x, then $P_n'(x)$ should approximate f'(x) in that range.
- For i=1,…,n-1, the polynomial of degree one interpolating the two points, $(x_i,y_i)$ and $(x_{i+1},y_{i+1})$ ($h=x_{i+1}-x_i$), is given by the Lagrange interpolant

$$P_1(x) = y_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i} = y_i \frac{x - x_{i+1}}{-h} + y_{i+1} \frac{x - x_i}{h}$$

- Differentiating this polynomial with respect to x gives

$$P_1'(x) = \frac{y_{i+1} - y_i}{h}$$

After taking $x=x_i$, it is the same as the first-order, forward difference formula for the first derivative that we derived earlier using Taylor series.

# Exercise

Extend the derivation in the previous slide for

- the central different of the first-order derivative
- the second order derivative

using the three data points: $(x_{i-1},y_{i-1})$, $(x_i,y_i)$, and $(x_{i+1},y_{i+1})$ and assuming that the discretization space is equal, i.e.,

$h=x_{i+1}-x_i=x_i-x_{i-1}$.

# Unequally Spaced Data

$$P_2(x) = y_{i-1}L_{i-1}(x) + y_i L_i(x) + y_{i+1}L_{i+1}(x)$$

$$= y_{i-1}\frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} + y_i\frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} + y_{i+1}\frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}$$

Take the derivative with respect to x

$$P_2'(x)$$

$$= y_{i-1}\frac{(x-x_i)+(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} + y_i\frac{(x-x_{i-1})+(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} + y_{i+1}\frac{(x-x_{i-1})+(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}$$

Evaluate the derivative at $x_i$

$$P_2'(x_i)$$

$$= y_{i-1}\frac{(x_i-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} + y_i\frac{(x_i-x_{i-1})+(x_i-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} + y_{i+1}\frac{(x_i-x_{i-1})}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}$$

# Implementation

$$P_{n-1}(x) = y_1 L_1(x) + y_2 L_2(x) + \mathrm{K} + y_n L_n(x) = \sum_{j=1}^{n} y_j L_j(x)$$

$$L_j(x) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} = \left[ \prod_{k=1}^{j-1} \frac{x - x_k}{x_j - x_k} \right] \left[ \prod_{k=j+1}^{n} \frac{x - x_k}{x_j - x_k} \right]$$

Given a value of *x*, the numerator of each $L_j$ contains

$$(x - x_1)(x - x_2)\mathrm{L}\ (x - x_{j-1})(x - x_{j+1})\mathrm{L}\ (x - x_n)$$

Because these terms are repeated in each of the $L_j$, it makes sense to compute the difference between x and $x_k$ once and store them.

```
>>xi = …;  %equivalent to x
>>x = …;  %equivalent to xk
>>dxi = xi –x; %equivalent to x-xk
>>num = prod(dxi(1:j-1))*prod(dxi(j+1:n)); %prod is a built-in       function
```

$$L_j(x) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} = \left[ \prod_{k=1}^{j-1} \frac{x - x_k}{x_j - x_k} \right] \left[ \prod_{k=j+1}^{n} \frac{x - x_k}{x_j - x_k} \right]$$

The denominator can be constructed the same way, but wait

$$\begin{bmatrix} & (x_1 - x_2) & (x_1 - x_3) & (x_1 - x_4) & \cdots & (x_1 - x_n) \\ (x_2 - x_1) & & (x_2 - x_3) & (x_2 - x_4) & \cdots & (x_2 - x_n) \\ (x_3 - x_1) & (x_3 - x_2) & & (x_3 - x_4) & \cdots & (x_3 - x_n) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (x_{n-1} - x_1) & (x_{n-1} - x_2) & (x_{n-1} - x_3) & \cdots & & (x_{n-1} - x_n) \\ (x_n - x_1) & (x_n - x_2) & (x_n - x_3) & \cdots & (x_n - x_{n-1}) & \end{bmatrix}$$

- We need to store a matrix (at least a triangle matrix) instead of a matrix. The memory cost may be high.
- The cost of computing each term (not storing them) is just a factor of two of the minimum cost.

```
>>den = prod(x(j)-x(1:j-1))*prod(x(j)-x(j+1:n));
```

# Exercise

- Write a MATLAB code to interpret the gas price of the previous exercise.

# Polynomial Interpolation with a Newton Basis

- The Newton form of an interpolating polynomial of degree n is

$$P_n(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) + \mathrm{K} + c_{n+1}(x - x_1)(x - x_2)\mathrm{K}\ (x - x_n)$$

- The Newton bases are 1, $(x-x_1)$, $(x-x_1)(x-x_2)$, $(x-x_1)(x-x_2)(x-x_3)$, …
- The coefficients are found by requiring $P_n(x_i)=y_i$.
- Although these basis functions may initially seem cumbersome, it turns out that the Newton form is more computationally efficient than interpolating polynomials written in monomial or Lagrange basis.
- The Newton form has good numerical properties, and it is very useful for theoretical analysis of interpolation schemes and numerical integration methods.

# Divided-Difference Notation

- The Newton form is more computationally efficient than interpolating polynomials written in monomial or Lagrange bases.
- Determining the coefficients, $c_i$, is made easier with the introduction of divided-difference notation.
- Start with the quadratic interpolating polynomial, and then extend it to the cubic one. The general form is given at last.

# Quadratic Polynomial in Newton Form

Consider a quadratic polynomial passing three points expressed in terms of the Newton basis function

$$P_2(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2)$$

Applying the three constraints gives

$$P_2(x_1) = c_1 = y_1$$

$$P_2(x_2) = c_1 + c_2(x_2 - x_1) = y_2$$

$$P_2(x_3) = c_1 + c_2(x_3 - x_1) + c_3(x_3 - x_1)(x_3 - x_2) = y_3$$

or in the matrix form

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & (x_2 - x_1) & 0 \\ 1 & (x_3 - x_1) & (x_3 - x_1)(x_3 - x_2) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

You can solve it using the forward substitution.

- The coefficients, $c_i$, are given by compact formulas called divided differences.
- $c_1=y_1$. To solve for $c_2$ and $c_3$, subtracting the first row from the second and third rows and then the new second row from the new third row leads to

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & (x_2-x_1) & 0 \\ 0 & (x_3-x_2) & (x_3-x_1)(x_3-x_2) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2-y_1 \\ y_3-y_2 \end{bmatrix}$$

- Normalizing the second column by dividing the second row by $x_2$-$x_1$ and dividing the third row by $x_3$-$x_2$ gives

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & (x_3-x_1) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ \dfrac{y_2-y_1}{x_2-x_1} \\ \dfrac{y_3-y_2}{x_3-x_2} \end{bmatrix}$$

# First-order divided differences

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & (x_3 - x_1) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ \dfrac{y_2 - y_1}{x_2 - x_1} \\ \dfrac{y_3 - y_2}{x_3 - x_2} \end{bmatrix} = \begin{bmatrix} y_1 \\ f[x_1, x_2] \\ f[x_2, x_3] \end{bmatrix}$$

$$f[x_1, x_2] = \frac{y_2 - y_1}{x_2 - x_1}$$

$$f[x_2, x_3] = \frac{y_3 - y_2}{x_3 - x_2}$$

In general, the first-order divided difference involving the ordered pairs $(x_i,y_i)$ and $(x_j,y_j)$ is

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

# Second-order divided differences

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & (x_3 - x_1) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ f[x_1, x_2] \\ f[x_2, x_3] \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & (x_3 - x_1) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ f[x_1, x_2] \\ f[x_2, x_3] - f[x_1, x_2] \end{bmatrix}$$

Subtract the second row from the third row to get

Divide the third row by $x_3$-$x_1$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ f[x_1, x_2] \\ \dfrac{f[x_2, x_3] - f[x_1, x_2]}{(x_3 - x_1)} \end{bmatrix} = \begin{bmatrix} y_1 \\ f[x_1, x_2] \\ f[x_1, x_2, x_3] \end{bmatrix}$$

$f[x_1,x_2,x_3]$ is the second-order divided difference involving the three points $(x_1,y_1)$, $(x_2,y_2)$, and $(x_3,y_3)$.

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ f[x_1, x_2] \\ f[x_1, x_2, x_3] \end{bmatrix}$$

The convention is to define the zeroth-order divided difference as $f[x_i]=y_i$ so that $y_1=f[x_1]$.

The Newton form of the quadratic polynomial can be written as

$$P_2(x) = f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2)$$

The coefficients of the Newton polynomial are divided differences.

What about the cubic polynomial in Newton Form?

$$P_3(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) + c_4(x - x_1)(x - x_2)(x - x_3)$$

$$P_3(x) = f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2) + f[x_1, x_2, x_3, x_4](x - x_1)(x - x_2)(x - x_3)$$

$$f[x_1, x_2, x_3, x_4] = \frac{f[x_2, x_3, x_4] - f[x_1, x_2, x_3]}{x_4 - x_1}$$

$$f[x_2, x_3, x_4] = \frac{f[x_3, x_4] - f[x_2, x_3]}{x_4 - x_2}$$

Homework to derive these expressions.

$$P_3(x) = P_2(x) + f[x_1, x_2, x_3, x_4](x - x_1)(x - x_2)(x - x_3)$$

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

# Properties of Divided Differences

- The divided difference $f[x_1,x_2,\ldots x_k]$ is the coefficient of $x^{k-1}$ in the polynomial that interpolate $(x_1,f_1)$, $(x_2,f_2)$, …, $(x_k,f_k)$,

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

- The interpolating polynomial of degree *n* can be obtained by adding a single term to the polynomial of degree *n*-1 expressed in the Newton form

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

- If f(x) is a polynomial of degree k, then the divided differences of order k+2, k+3, …, are identically zero.

# Properties of Divided Differences

- A sequence of divided differences may be constructed recursively from the formula

$$f[x_1,\ldots,x_k]=\frac{f[x_2,\ldots,x_k]-f[x_1,\ldots,x_{k-1}]}{x_k-x_1}$$

and the zeroth-order divided difference is defined by

$$f[x_i]=y_i$$

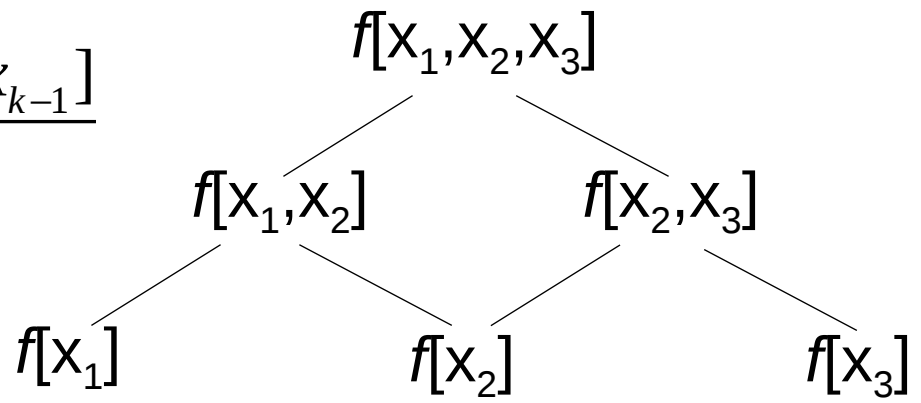
- The divided difference $f[x_1,\ldots,x_k]$ is invariant under all permutations of its arguments $x_1,\ldots,x_k$.

# Divided Difference Tables

- The equation of divided differences is recursive:
  - A higher order coefficient is defined as the difference of two lower order coefficients, which in turn, defined as a differences of the next lower order coefficients.
  - The recursion terminates when only zeroth-order difference remains.
- The divided differences can be manually evaluated with the aid of divided-difference table.

$$f[x_1,...,x_k] = \frac{f[x_2,...,x_k] - f[x_1,...,x_{k-1}]}{x_k - x_1}$$

$$f[x_i] = y_i$$



$$f[x_1,\ldots,x_k]=\frac{f[x_2,\ldots,x_k]-f[x_1,\ldots,x_{k-1}]}{x_k-x_1}$$

| $x_i$ | $f[\,]$ | $f[,]$ | $f[,,]$ | $f[,,,]$ |
|---|---|---|---|---|
| $x_1$ | $f[x_1]$ | | | |
| $x_2$ | $f[x_2]$ | $f[x_1,x_2]$ | | |
| $x_3$ | $f[x_3]$ | $f[x_2,x_3]$ | $f[x_1,x_2,x_3]$ | |
| $x_4$ | $f[x_4]$ | $f[x_3,x_4]$ | $f[x_2,x_3,x_4]$ | $f[x_1,x_2,x_3,x_4]$ |

- The table organizes the calculation in a way that shows the dependence of higher order coefficients on lower order coefficients.
- The first two columns are the known ($x_i$,$y_i$) data
- The third column contains first-order divided differences, which depend on the data in the preceding columns.
- Higher order differences are obtained from lower order differences in the columns to the left.

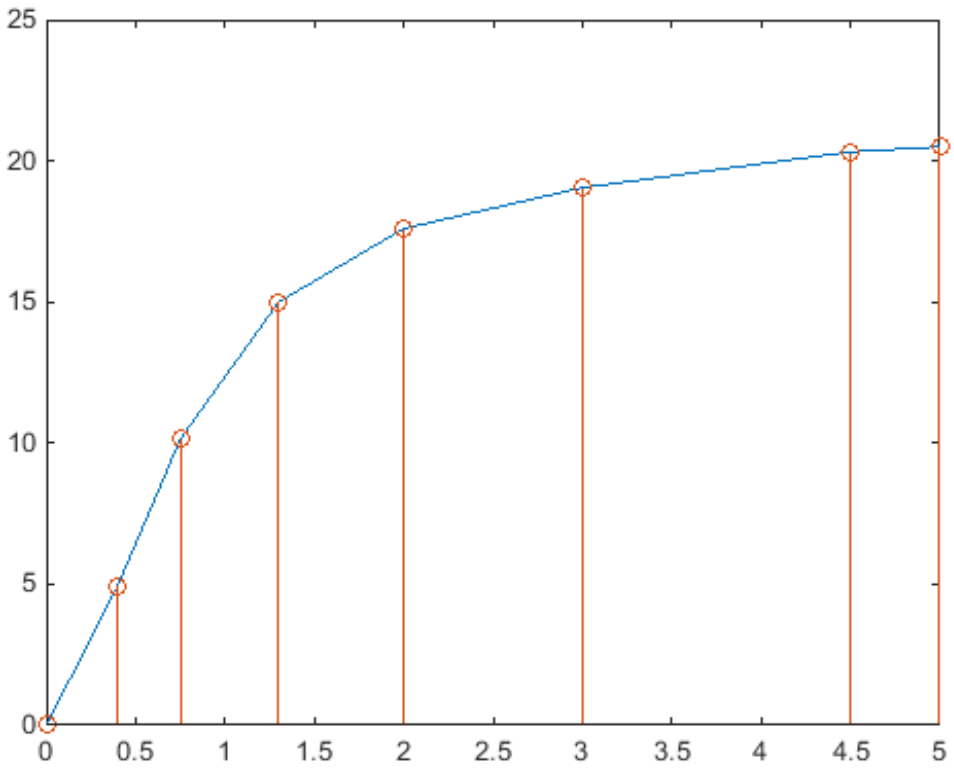
# Exercise

| V | 0 | 0.4 | 0.75 | 1.3 | 2 | 3 | 4.5 | 5 |
|---|---|---|---|---|---|---|---|---|
| I | 0 | 4.95 | 10.14 | 15.0 | 17.6 | 19.05 | 20.32 | 20.5 |

Use cubic interpolation to find the value of I for V=1.5, with the four data pairs of v = 0.4, 0.75, 1.3, and 2.0.

Why choosing these four points?

Think about how to automatically calculate the divided-differences and the interpolation for V=1.5.




Use the code divDiffTable_exercise.m as the starting point.

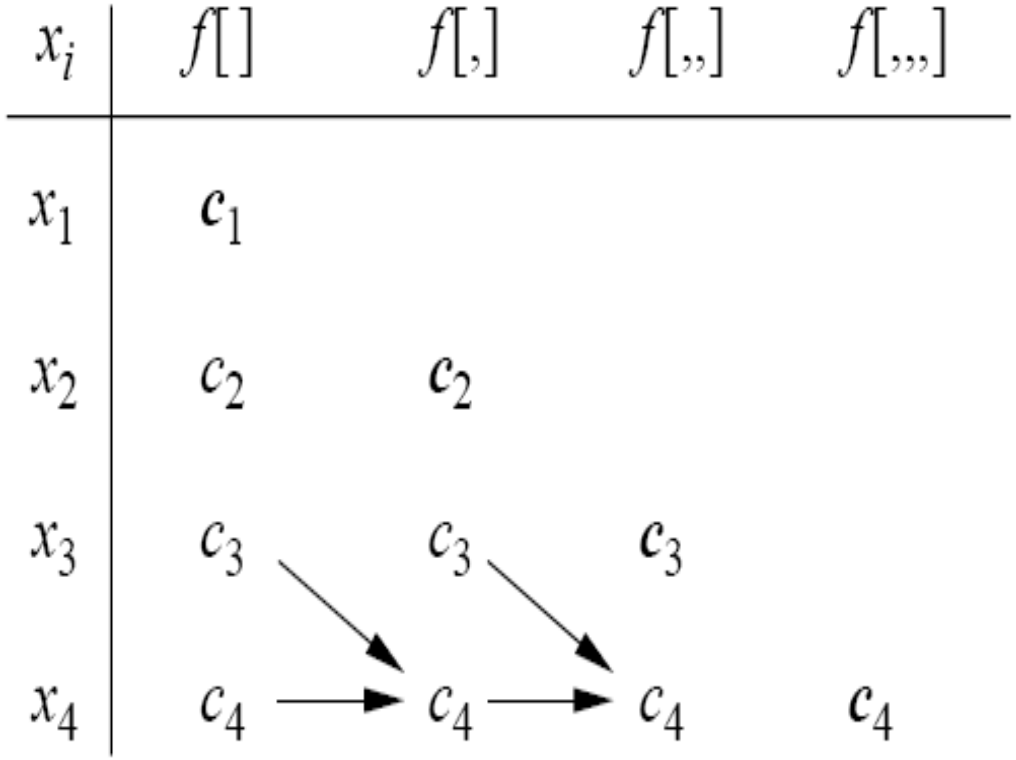
# Implementation

- Only the diagonal entries in the divided-difference table are used to evaluate the Newton polynomial.
- However, it is impossible to compute only the diagonal entries of the table, because higher order divided differences are defined by lower order divided differences.
- The diagonal entries must be built up from the lower order differences.
- It is not necessary to store all the intermediate results.

- Consider the creation of a cubic polynomial in a Newton basis. For convenience, the polynomial is written as

$$P_3(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) + c_4(x - x_1)(x - x_2)(x - x_3)$$

- The immediate goal is to devise an algorithm to obtain the coefficients without first constructing the entire matrix and then extracting the diagonals for the interpolation.
- This is achieved by storing the intermediate (lower order) divided differences in vector *c* and overwriting appropriate elements of *c* as needed.




- The boldface symbols on the diagonal of the table are the final values of the $c_i$. The $c_i$ values below the diagonal are overwritten during the execution of the algorithm.
- Since the values of $c_i$ are overwritten, it is necessary to sweep from the bottom of the table up to the diagonal.

# Notations

- In the equations above, the degree-n polynomial is defined over the set of support points $(x_i,y_i)$, i=1,…,n+1.
- In most of books on numerical mathematics, the range of indices of the n+1 support points is i=0, …,n.
- The i=0 index enables notational convenience, but it is inconsistent with MATLAB vectors and matrices, which have a starting index of 1.
- We will use the conventional notations (starting from i=0) in the following slides so that you are familiar with both systems of notations.

# Differentiation Using Interpolating Polynomials

- The expression of function f is unknown, but its n+1 values are known, $(x_0,f_0)$, $(x_1,f_1)$, $(x_2,f_2)$, … $(x_n,f_n)$.
- You want to build a n-th degree polynomial with the Newton basis

$$P_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_n(x - x_1)(x - x_2)\ldots(x - x_{n-1})$$

- If we choose the coefficient *c* so that $P_n(x)=f(x)$ at the n+1 known points, $(x_i,f_i)$, i=0,1,…,n, then $P_n(x)$ is an interpolating polynomial.
- Derivative of this polynomial then provides an approximation to the derivative of the function f(x).

# Determine Coefficient using Divided Differences

Special standard notation used for divided differences.

First divided difference between $x_0$ and $x_1$

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f_1 - f_0}{x_1 - x_0} = (f_0^{[1]})$$ a shorter version

First divided difference between $x_1$ and $x_2$

$$f[x_1, x_2] = \frac{f_2 - f_1}{x_2 - x_1} = (f_1^{[1]})$$

$$f[x_s, x_t] = \frac{f_t - f_s}{x_t - x_s}$$ General Expression

Second divided difference between $x_0$, $x_1$ and $x_2$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = (f_0^{[2]})$$

$$f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}]}{x_n - x_0} = (f_0^{[n]})$$ n-th order

# Differentiation Based on Newton Basis

$$P_n(x) = f_0^{[0]} + f_0^{[1]}(x - x_0) + f_0^{[2]}(x - x_0)(x - x_1)$$
$$+ f_0^{[3]}(x - x_0)(x - x_1)(x - x_2) + \ldots$$
$$+ f_0^{[n]}(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

Taking the derivative of $P_n(x)$ with respect to x gives

$$P_n'(x) = f_0^{[1]} + f_0^{[2]}[(x - x_0) + (x - x_1)]$$
$$+ f_0^{[3]}[(x - x_0)(x - x_1) + (x - x_1)(x - x_2) + (x - x_0)(x - x_2)]$$
$$+ \ldots + f_0^{[n]} \sum_{i=0}^{n-1} \frac{(x - x_0)(x - x_1)\ldots(x - x_{n-1})}{(x - x_i)}$$

Because $f(x)=P_n(x)+\text{Error}$, $f'(x)= P'_n(x)+\text{Error}'$.

# Error of Interpolation

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

Define the error function

$$E(x) = f(x) - P_n(x) = (x - x_0)(x - x_1)\cdots(x - x_n)g(x)$$

which is zero for $x=x_0, x_1, \ldots, x_n$. Therefore,

$$f(x) - P_n(x) - E(x) =$$

$$f(x) - P_n(x) - (x - x_0)(x - x_1)\cdots(x - x_n)g(x) = 0$$

To determine g(x), we define an auxiliary function, W(t), as

$$W(t) = f(t) - P_n(t) - (t - x_0)(t - x_1)\cdots(t - x_n)g(x) = 0$$

which is zero for $t=x$ and $x=x_0, x_1, \ldots, x_n$.

$$W(t) = f(t) - P_n(t) - (t - x_0)(t - x_1)\cdots(t - x_n)g(x) = 0$$

Take the n+1 order of derivative of W(t) with respect to t. There must be a ξ in the interval that has $x_0$, $x_n$, or x as endpoint, i.e., ξ between ($x_0$, $x_n$, x). Take t=ξ, we have

$$W^{(n+1)}(\xi) = f^{(n+1)}(\xi) - 0 - (n+1)!\,g(x) = 0$$

Therefore,

$$g(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}$$

According to the error definition,

$$E(x) = f(x) - P_n(x) = (x - x_0)(x - x_1)\cdots(x - x_n)g(x)$$

we have

$$E(x) = (x - x_0)(x - x_1)\cdots(x - x_n)\frac{f^{(n+1)}(\xi)}{(n+1)!} \quad \xi \text{ between } (x_0, x_n, x)$$

# Error of Interpolation

- The expression for error is not always useful, because the function, *f*, is unknown. We thus cannot evaluate its (n+1)-st derivative.
- However, we can conclude that, if the function is "smooth" (the smaller the higher derivative of a function, the smoother the function is), the error is small. In line with this, a lower-degree polynomial should work satisfactorily.
- On the other hand, a "rough" function can be expected to have larger errors when interpolated.
- We can also conclude that extrapolation will have larger errors than for extrapolation, because of the $(x-x_i)$ terms.

# Error of Derivative

$$P_n(x) = f_0^{[0]} + (x - x_0) f_0^{[1]} + (x - x_0)(x - x_1) f_0^{[2]} + (x - x_0)(x - x_1)(x - x_2) f_0^{[3]} + \ldots + (x - x_0)(x - x_1)\ldots(x - x_{n-1}) f_0^{[n]}$$

$$P_n'(x) = f_0^{[1]} + f_0^{[2]}[(x - x_0) + (x - x_1)] + f_0^{[3]}[(x - x_0)(x - x_1) + (x - x_1)(x - x_2) + (x - x_0)(x - x_2)] + \ldots + f_0^{[n]} \sum_{i=0}^{n-1} \frac{(x - x_0)(x - x_1)\ldots(x - x_{n-1})}{(x - x_i)}$$

Error of the approximation to f'(x), when $x=x_i$, is

$$\left[ \prod_{\substack{j=0 \\ j \neq i}}^{n} (x_i - x_j) \right] \frac{f^{(n+1)}(\xi)}{(n+1)!} \quad \xi \text{ between } (x_0, x_n, x)$$

# Equally Spaced Data

Even though divided difference can handle any table, it is instructive to see how ordinary differences can estimate the derivative when a table is evenly spaced.

Denote $s=(x-x_i)/h$, $\Delta f_i=f_{i+1}-f_i$, $\Delta^2 f_i=\Delta(\Delta f_i)$, and $\Delta^n f_i=\Delta(\Delta^{n-1} f_i)$, we can write the general form

$$P_n(x) = f_0^{[0]} + f_0^{[1]}(x-x_0) + f_0^{[2]}(x-x_0)(x-x_1) + f_0^{[3]}(x-x_0)(x-x_1)(x-x_2)+\ldots + f_0^{[n]}(x-x_0)(x-x_1)\ldots(x-x_{n-1})$$

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

as

$$P_n(s) = f_i + s\Delta f_i + \frac{s(s-1)}{2!}\Delta^2 f_i + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_i + \ldots + \prod_{j=0}^{n-1}(s-j)\frac{\Delta^n f_i}{n!}$$

$$E(s) = \left[\prod_{j=0}^{n}(s-j)\right] h^{n+1}\frac{f^{(n+1)}(\xi)}{(n+1)!}$$

$$P_n(s) = f_i + s\Delta f_i + \frac{s(s-1)}{2!}\Delta^2 f_i + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_i$$

$$+ \ldots + \prod_{j=0}^{n-1}(s-j)\frac{\Delta^n f_i}{n!}$$

$$E(s) = \left[\prod_{j=0}^{n}(s-j)\right] h^{n+1}\frac{f^{(n+1)}(\xi)}{(n+1)!}$$

$$\frac{d}{dx}P_n(s) = \frac{d}{ds}P_n(s)\frac{ds}{dx} = \frac{1}{h}\frac{d}{ds}P_n(s)$$

$$= \frac{1}{h}\left[\Delta f_i + \sum_{j=1}^{n}\left(\sum_{k=0}^{j-1}\prod_{\substack{l=0 \\ l\neq k}}^{j-1}(s-l)\frac{\Delta^j f_i}{j!}\right)\right]$$

When $x=x_i$, $s=0$. Then the error is

$$\text{Error} = \frac{(-1)^n h^n}{(n+1)} f^{(n+1)}(\xi) = O(h^n)$$

# Simpler Formulas

The equation of $P_n'(s)$ will becomes substantially simpler, if we evaluate it at $x=x_i$, i.e., $s=0$.

$$P_n(s) = f_i + s\Delta f_i + \frac{s(s-1)}{2!}\Delta^2 f_i + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_i$$

$$+ \ldots + \prod_{j=0}^{n-1}(s-j)\frac{\Delta^n f_i}{n!}$$

$$f'(x_i) = \frac{1}{\Delta x}\left[\Delta f_i - \frac{\Delta^2 f_i}{2} + \frac{\Delta^3 f_i}{3} - \ldots \frac{\Delta^n f_i}{n}\right]_{x=x_i}$$

The error is $O(h^n)$, depending on the number of data used for the differentiation, i.e., the n value.

# An Example of 2nd-order polynomial

- Three points $(x_i,f_i)$, $(x_{i+1},f_{i+1})$, $(x_{i+2},f_{i+2})$
- 2nd order polynomial

$$P_2(x) = a_i + (x - x_i)a_{i+1} + (x - x_i)(x - x_{i+1})a_{i+2}$$

- Three unknowns: $a_i$, $a_{i+1}$, and $a_{i+2}$

$$P_2(x_i) = f_i = a_i$$

$$P_2(x_{i+1}) = f_{i+1} = a_i + (x_{i+1} - x_i)a_{i+1} \rightarrow a_{i+1} = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} = f[x_i, x_{i+1}]$$

$$P_2(x_{i+2}) = f_{i+2} = a_i + (x_{i+2} - x_i)a_{i+1} + (x_{i+2} - x_i)(x_{i+2} - x_{i+1})a_{i+2}$$

$$a_{i+2} = f[x_i, x_{i+1}, x_{i+2}] \quad f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} = \frac{1}{x_{i+2} - x_i}\left(\frac{f_{i+2} - f_{i+1}}{x_{i+2} - x_{i+1}} - \frac{f_{i+1} - f_i}{x_{i+1} - x_i}\right)$$

# Evenly Spaced Data

If the three x points are evenly spaced, the expression of $P_2(x)$ can be simplified.

let $\Delta x = x_{i+1} - x_i$ and denote $f_{i+1} - f_i$ as $\Delta f_i$.

$$P_2(x) = f_i + \frac{(x - x_i)}{\Delta x}\Delta f_i + \frac{(x - x_i)(x - x_{i+1})}{2(\Delta x)^2}\Delta^2 f_i$$

where $\Delta^2 f_i = \Delta(\Delta f_i) = \Delta(f_{i+1} - f_i) = \Delta f_{i+1} - \Delta f_i = f_{i+2} - 2f_{i+1} + f_i$. Letting $s = (x - x_i)/\Delta x$, the equation takes a still simpler form,

$$P_2(x) = f_i + s\,\Delta f_i + \frac{s(s-1)}{2}\Delta^2 f_i \qquad df/dx = df/ds\,(ds/dx) = (1/\Delta x)\,df/ds$$

$P_2'(x) =$ $$f'(x) = \frac{1}{\Delta x}\left(\Delta f_i + (2s-1)\frac{\Delta^2 f_i}{2}\right) + error\ of\ O((\Delta x)^2)$$

# Derivative of Interpolating Polynomial

If we restrict ourselves to evaluating the derivative at one of the given points, say $x=x_{i+1}$, then $s=1$ and the derivative reduces to

$$f'(x_{i+1}) = \frac{1}{\Delta x}\left( \Delta f_i + \frac{\Delta^2 f_i}{2} \right) + O((\Delta x)^2)$$

$$= \frac{1}{\Delta x}\left[ (f_{i+1} - f_i) + \frac{f_{i+2} - 2f_{i+1} + f_i}{2} \right] + O((\Delta x)^2)$$

$$= \frac{f_{i+2} - f_i}{2\Delta x} + O((\Delta x)^2)$$

This is the same result we obtained using the Taylor series.

Exercise:

Derive the forward difference at $x_i$

- $f'(x_i)=(f_{i+1}-f_i)/\Delta x+O(\Delta x)$
- $f'(x_i)=(-f_{i+2}+4f_{i+1}-3f_i)/(2\Delta x)+O(\Delta x^2)$

## *Table 3.1. Formulas for numerical differentiation*

| | | |
|---|---|---|
| ***First derivatives*** | $f'(x_0) = \frac{f_1 - f_0}{\Delta x} + O(\Delta x)$ | 1st order, forward* difference |
| | $f'(x_0) = \frac{f_1 - f_{-1}}{2\Delta x} + O((\Delta x)^2)$ | 2nd order, central difference |
| | $f'(x_0) = \frac{-f_2 + 4f_1 - 3f_0}{2\Delta x} + O((\Delta x)^2)$ | 2nd order, forward* difference |
| ***Second derivatives*** | $f''(x_0) = \frac{f_1 - 2f_0 + f_{-1}}{(\Delta x)^2} + O((\Delta x)^2)$ | 2nd order, central difference |
| | $f''(x_0) = \frac{-f_3 + 4f_2 - 5f_1 + 2f_0}{(\Delta x)^2} + O((\Delta x)^2)$ | 2nd order, forward§ difference |
| ***Third derivatives*** | $f'''(x_0) = \frac{f_2 - 2f_1 + 2f_{-1} - f_{-2}}{2(\Delta x)^3} + O((\Delta x)^2)$ | 2nd order, central difference |
| | $f'''(x_0) = \frac{-3f_4 + 14f_3 - 24f_2 + 12f_1 - 5f_0}{(\Delta x)^3} + O((\Delta x)^2)$ | 2nd order, forward* difference |

* To obtain the backward difference approximation for odd-order derivatives, multiply the forward difference equation by $-1$ and make all non-zero subscripts negative; e.g., the 1st-order backward difference approximation to the first derivative is $f'(x_0) = (f_0 - f_{-1}) / \Delta x$

§ To obtain the backward difference approximation for even-order derivatives, make all non-zero subscripts negative.

# MATLAB Implementation and Textbook Example

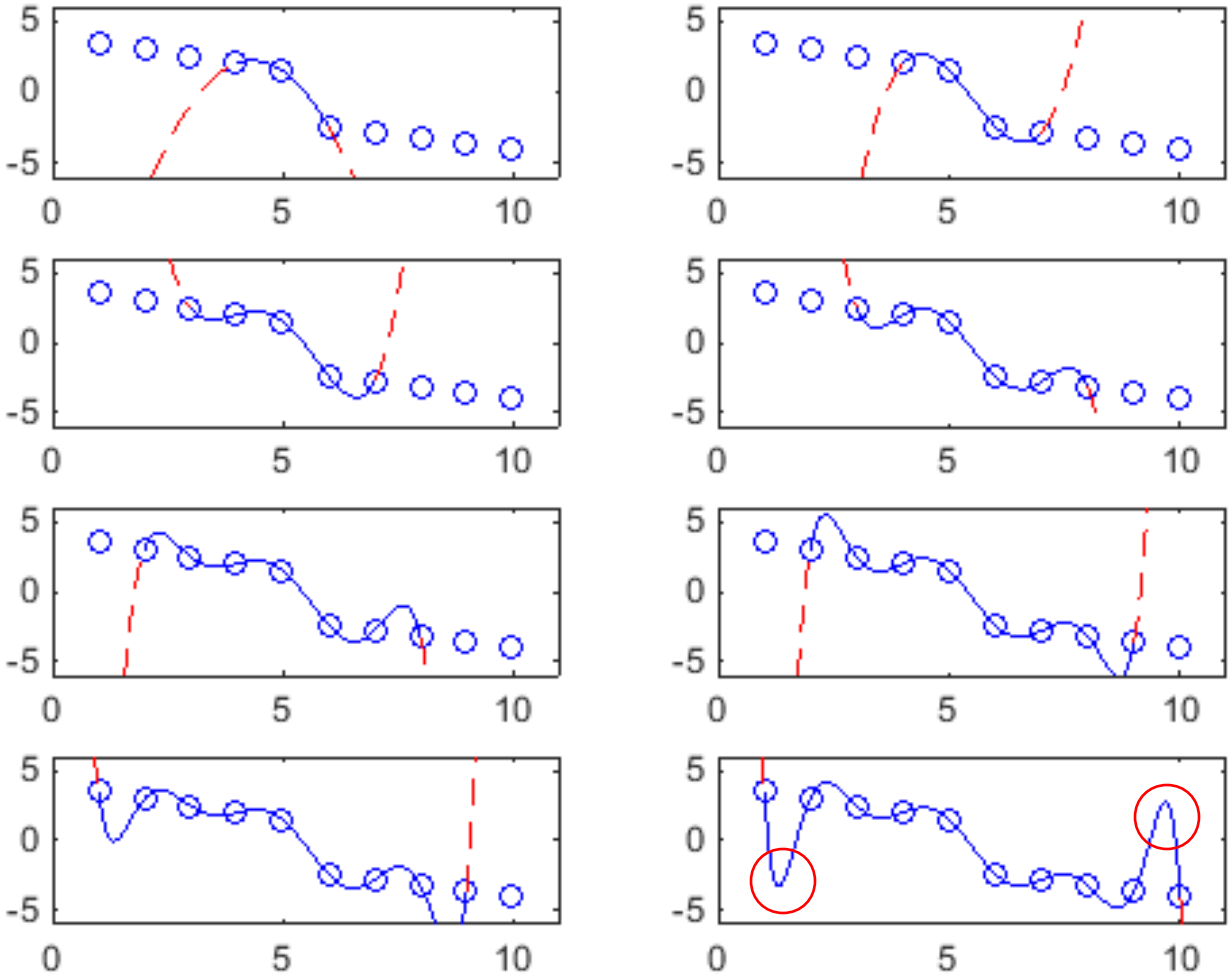
- Diff(f) is equivalent to Δf.
- A simple forward-difference estimate of the derivative is given by diff(f)./diff(x), where *f* are the function values at the *n* points *x*.
- Note that if *x* has length *n*, diff returns a vector of length *n*-l.
- "diff" can be nested, so that $\Delta^2 f$ =diff(diff(f)).

```
x=x(:)'; %makes x a row vector to begin
xd=[x-h;x+h];
yd=f(xd); %define a function f or substitute the function for f
dfdx=diff(yd)./diff(xd);
```

- How does the truncation error changes when more and more terms are included?
- Does the truncation error matter here?

# Polynomial Wiggle

- Increasing the degree of a polynomial interpolant does not necessarily increases the accuracy of the interpolation.
- By definition, the interpolant, F(x) matches the true function at the support points. However, one cannot guarantee that “between” the support points, F(x) will be a good approximation to the true f(x) that generated the support points.
- If, for example, f(x) is a known analytical function and the size n of the set of support points used to define F(x) is allowed to increase (and hence increase the degree of the polynomial F(x)), the interpolant will likely tend to oscillate between the support points.
- This polynomial wiggle can occur even when the true f(x) is smooth.



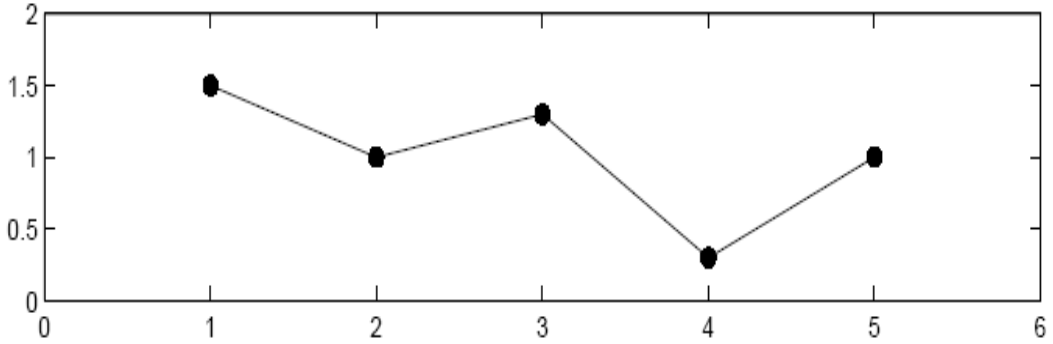

- Solid line:
- Interpolation
- Dashed line:
- Extrapolation

- The variation between data is OK when polynomials are constructed for up to six data.
- Interpolation error however increases afterward, and becomes so huge in the end that the interpolating polynomial has nothing in common compared to the characteristic revealed by data.
- The polynomial wiggle make appearance of higher-order polynomials unacceptable for data interpolation.

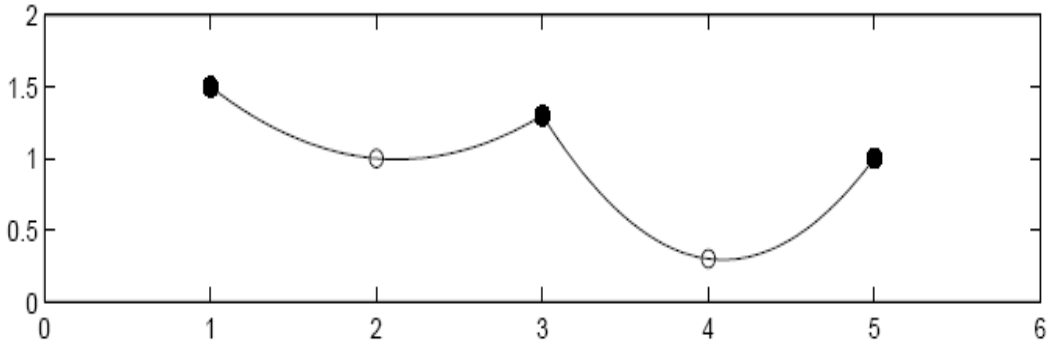
# Piecewise Polynomial Interpolation

- Interpolation with piecewise polynomials provides a practical solution to the shortcoming of high-degree polynomial interpolation.
- Instead of approximating the function by passing a single interpolant through a large number of support points, piecewise polynomial interpolation uses a set of lower degree interpolants, each of which is defined on a subinterval of the whole domain.
- The joints between adjacent piecewise interpolants are called the breakpoints, or knots.
- The use of piecewise functions introduces new features to the interpolant.
- The relation between adjacent piecewise functions is of fundamental importance.

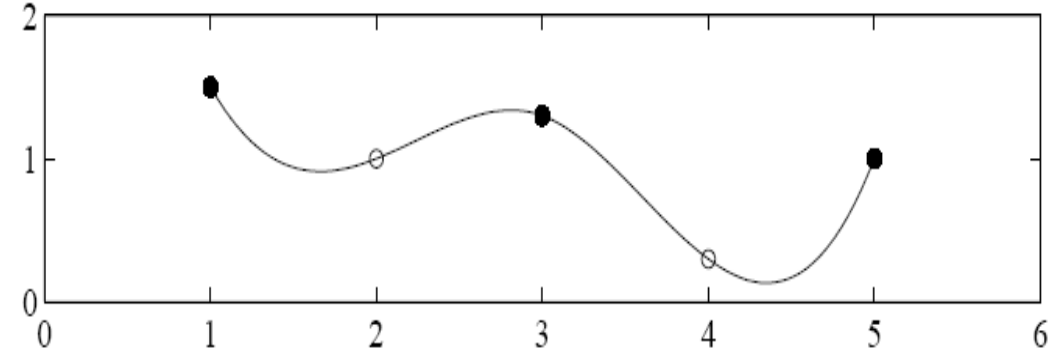
The shape of adjacent interpolants is affected by constraints on the continuity of the interpolants and its derivatives at the breakpoints.




Four piece-wise linear functions; functions continuous but derivatives discontinuous at knots




Two adjacent quadratic interpolants; functions continuous but derivatives discontinuous at knots




Two adjacent quadratic interpolants; functions and derivatives continuous at knots (cubic spline)

- Change of notations

For the piecewise polynomial interpolation, $P_i(x)$ will designate a polynomial defined n the i-th segment of a piecewise continuous curve. The i-th segment is delimited by $x_i \leq x \leq x_{i+1}$.

- Computational tasks
  - The overall computing is still to first construct and then evaluate the interpolant.
  - Since the piecewise interpolant is not global (i.e., not a single function for all support points), the appropriate subinterval must be located before the interpolation function is evaluate.
  - The tasks required to perform a piecewise interpolation at a point x are:
    - Location x in the set of support points $(x_i,y_i)$, i=1,…,n
    - Compute the coefficients of the local interpolating polynomial defined for the appropriate support points
    - Evaluate the local interpolating polynomial
  - If the piecewise interpolants have continuity of first and higher derivatives, then adjacent interpolants may be coupled. Thus, although the individual interpolants are local, by virtual of being connected to each other at the breakpoints, they exhibits some global behavior.

# Piecewise-Linear Interpolation

- The simplest piecewise interpolation scheme: use linear interpolants between each successive pair of breakpoints.
- Exercise: Consider interpolation to find the value of y at x=0.75 for the dataset below:

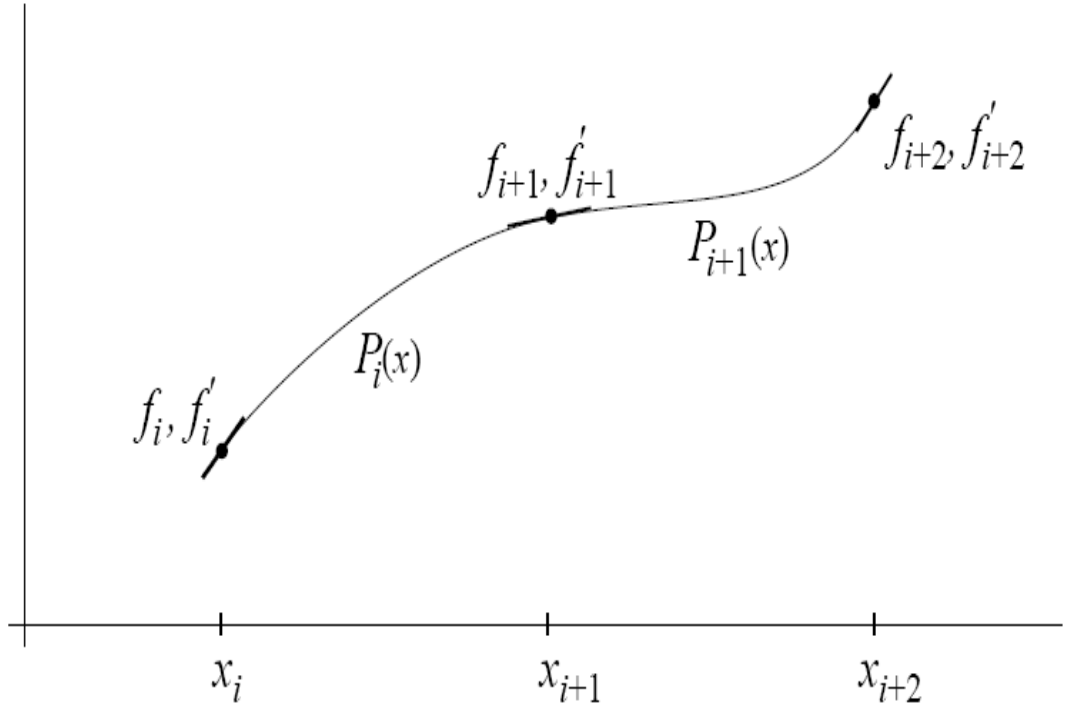
| x | 0.2 | 0.6 | 1.0 | 1.4 |
|---|---|---|---|---|
| y | 0.5535 | 1.0173 | 1.0389 | 0.8911 |

- The only complication in implementing piecewise-linear interpolation is in determining the appropriate pair of support points for use in constructing the interpolant.
- The search of support points can be done by using *incremental search* or *binary search* methods.

# Piecewise-Cubic Hermite Interpolation

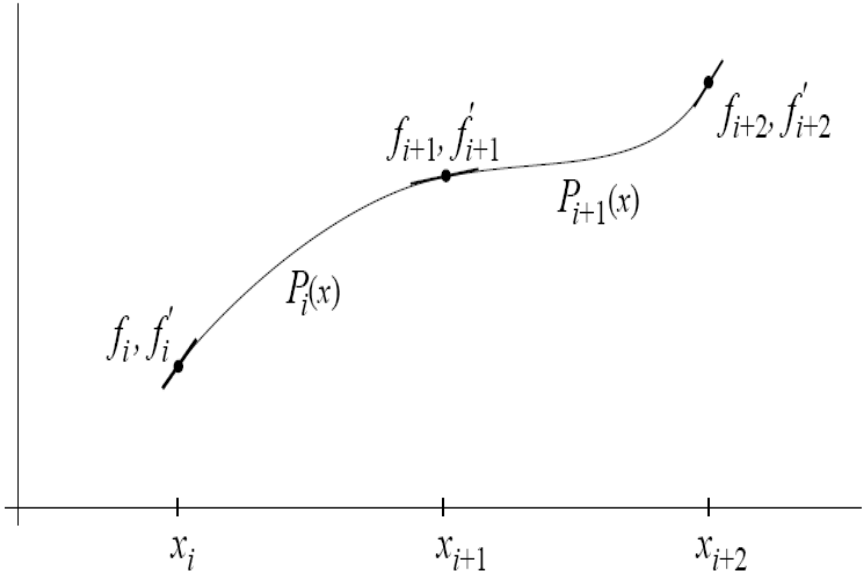
- It would be logical to follow the presentation of piecewise-linear interpolation with a piecewise-quadratic interpolation.
- However, in practice, he advantage of piecewise-quadratic functions over piecewise-linear functions are not compelling.
- For the purpose of interpolating known data, piecewise-cubic functions are much more useful.
- There are two different types of piecewise-cubic interpolating functions: *Hermite polynomials* and *cubic splines*

# Piecewise-Cubic Hermite Interpolation




Hermite polynomials are required to agree with the functions and its first derivative at each of the support points.

Two piecewise polynomials:
$P_i(x)$ on $[x_i, x_{i+1}]$ and $P_{i+1}(x)$ on $[x_{i+1}, x_{i+2}]$




$$a_i = f(x_i)$$

$$b_i = f'(x_i)$$

$$c_i = \frac{3f[x_i, x_{i+1}] - 2f'(x_i) - f'(x_{i+1})}{(x_{i+1} - x_i)}$$

$$d_i = \frac{f'(x_i) - 2f[x_i, x_{i+1}] + f'(x_{i+1})}{(x_{i+1} - x_i)^2}$$

If the cubic form of $P_i(x)$ is written as

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

The four unknown coefficients are defined by requiring that

$$P_i(x_i) = f(x_i),\ P_i'(x_i) = f'(x_i)$$

$$P_i(x_{i+1}) = f(x_{i+1}),\ P_i'(x_{i+1}) = f'(x_{i+1})$$

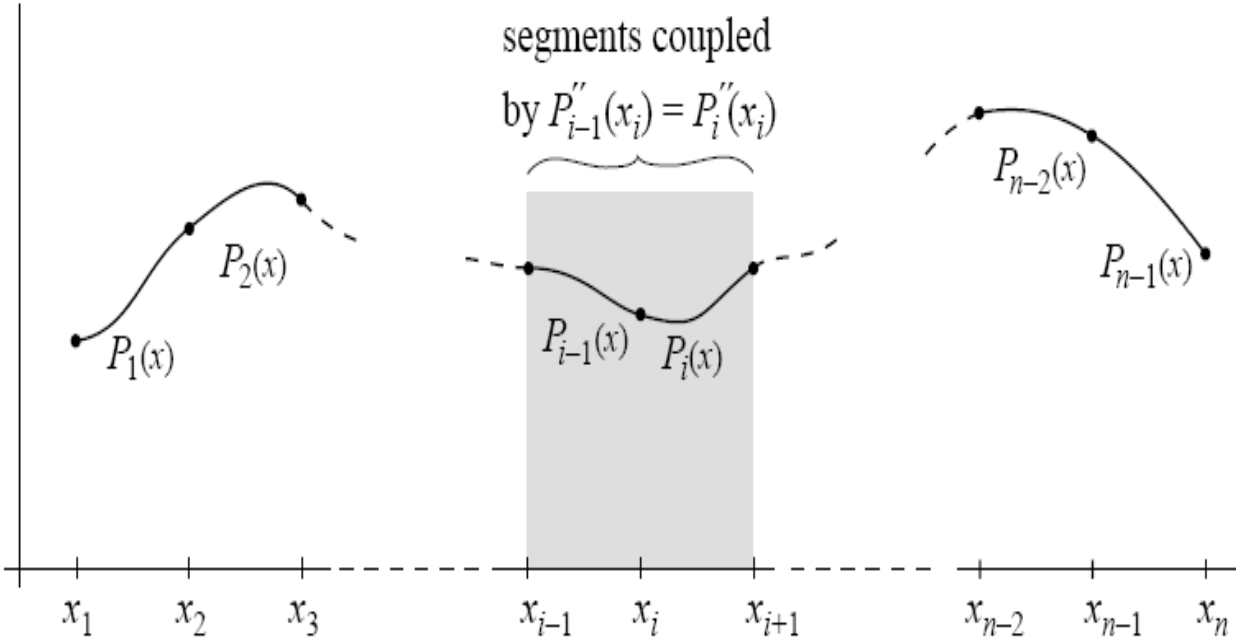
# Piecewise-Cubic Hermite Interpolation

- The Hermite polynomials have the following two desirable properties:
  - The segments of the piecewise Hermite polynomials have continuous first derivative at the support points.
  - The shape of the function being interpolated is better matched, because the tangent of this function and the tangent of the Hermite polynomial agree at the support points.
- Because interpolating Hermite polynomials requires values of both y=f(x) and f'(x), they are better suited to interpolation of analytical functions that discrete data, say, from an experiment.
- For the interpolation of discrete data wit unknown f'(x), splines are recommended.
- Questions: if the function is known, why do we need to interpolate the function?

# Purpose of Interpolation

- Plotting a smooth curve through discrete data points
- Reading the lines of a table
- Differentiating or integrating tabular data
- Evaluating a mathematical function quickly and easily
- Replacing a complicated function by a simple one

# Cubic Spline Interpolation




- A spline is a piecewise polynomial of degree *k* that is continuously differentiable *k-1* time.
- Each $P_i(x)$ is a cubic polynomial, and at each breakpoint, the $P_i(x)$, $P_i'(x)$, and $P_i''(x)$ are continuous.
- By requiring continuous $P_i''(x)$, the cubic spline avoids the need to specify the value of *f*'(x).
- This however requires solving a system of linear equation.

# Knows and Unknows

- For n data pairs, there are n-1 piecewise-cubic polynomials. The number of unknown is 4(n-1), because each polynomial has four unknowns.
- Since each of the n-1 polynomials must match the 2 given y values at the breakpoints of its segment, we have 2(n-1) constraints.
- Since it is required that the first derivative is continuous for the interior points, we have (n-2) constraints.
- Since it is required that the second derivative is continuous for the interior points, we have (n-2) constraints.
- We therefore have 4n-6 constraints, and need two more!

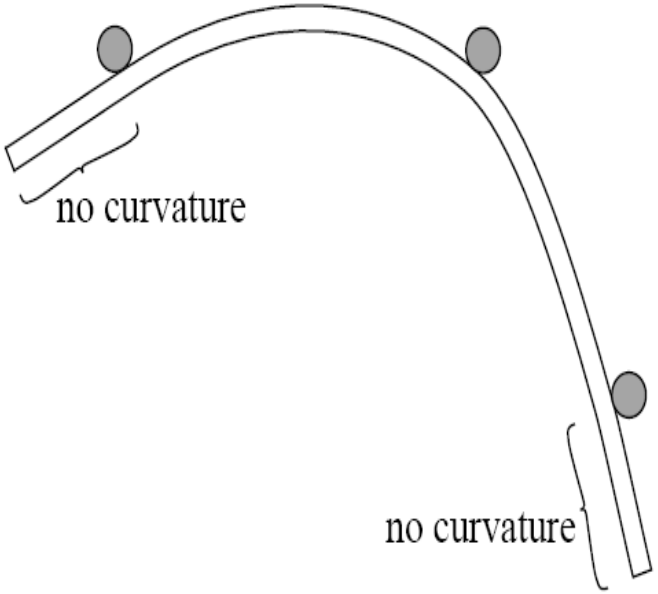
# A Simple Example

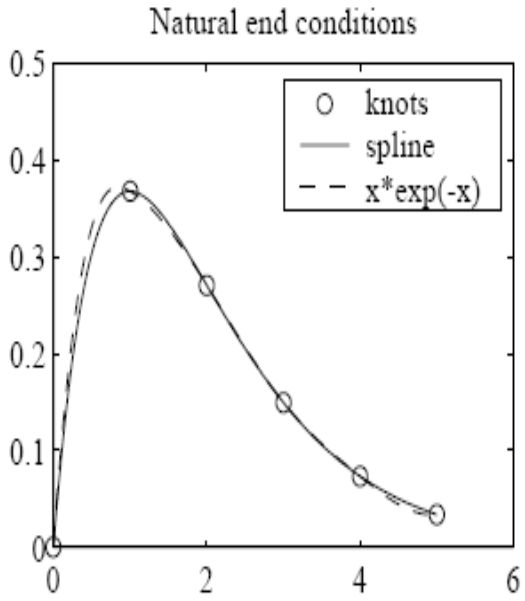
- We have three data pairs $(t_1,y_1)$, $(t_2,y_2)$, and $(t_3,y_3)$.
- We want to build two cubic splines in the intervals of $[t_1,x_2]$ and $[t_2,x_3]$.
- For simplicity, let's use the monomial basis

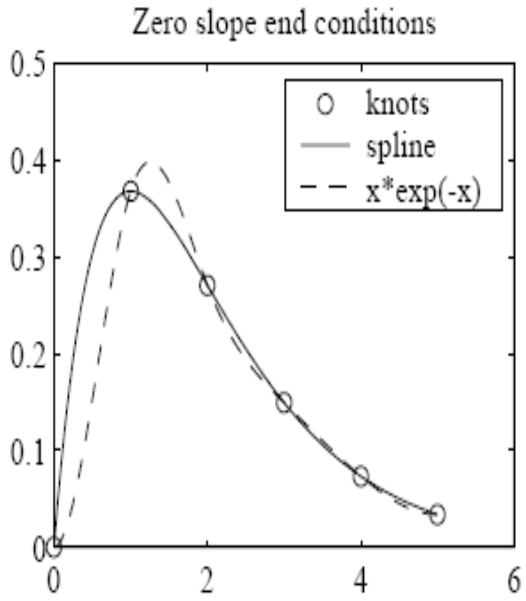
$$p_1(t) = \alpha_1 + \alpha_2 t + \alpha_3 t^2 + \alpha_4 t^3, \qquad p_2(t) = \beta_1 + \beta_2 t + \beta_3 t^2 + \beta_4 t^3$$

# The Two Constraints

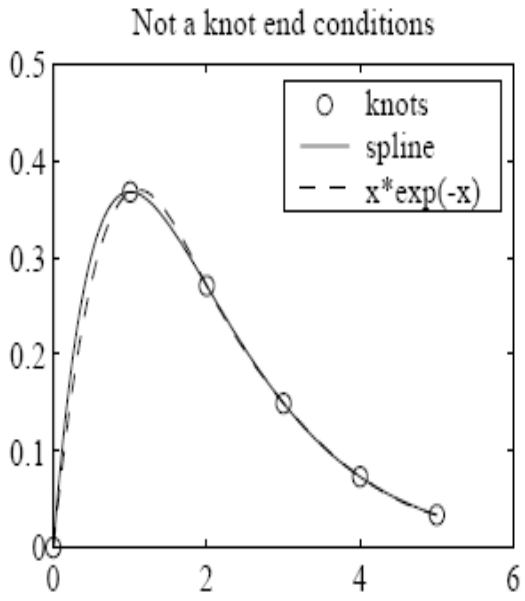
- Fixed-slope end condition: Specify the first derivative at the two end points, based either on desired boundary conditions (e.g., f’(x)=0) or on estimates of the derivative from the data.
- “Natural” end condition: force the second derivative to be zero at the endpoints,
- Not-a-knot condition: require continuity of P’’’(x) at the first interval knots at $x_2$ and $x_{n-1}$.

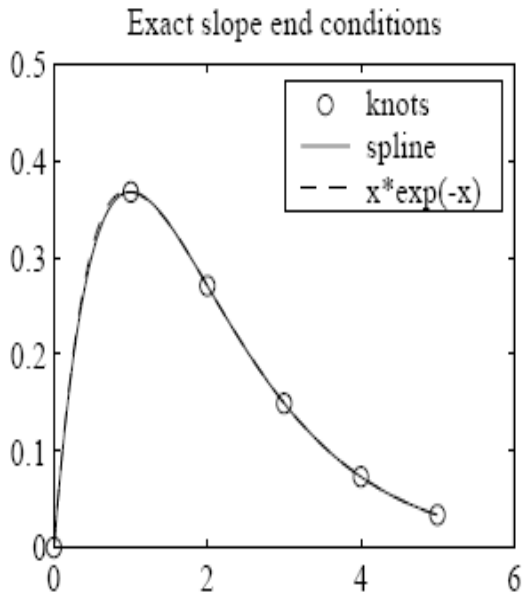



The not-a-knot end conditions are the most accurate end conditions when information on the slope of the spline at the end points is not known.



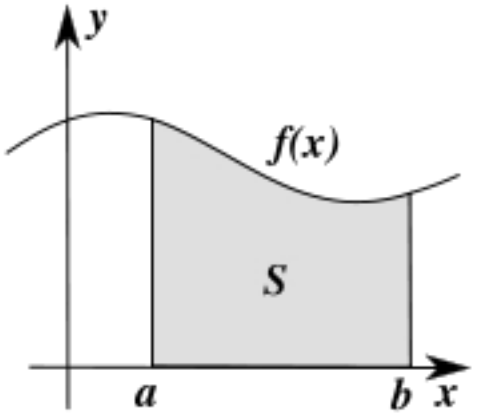



# MATLAB Built-in Functions

- Interp1/interp2/interp3: One/two/three-dimensional interpolation with piecewise polynomials
- interpft: one-dimensional interpolation of uniformly spaced data using Fourier series
- interpn: n-dimensional extension of methods used by interp3
- spline: one-dimensional interpolation with cubic-splines using not-a-knot or fixed-slope end conditons.

# Why Numerical Integration?

$$\int_a^b f(x)\,dx.$$




Estimate the area S numerically.

- The integrand f(x) may be known only at certain points, such as obtained by sampling.
- The integrant f(x) may be known, but it may be difficult or impossible to find an analytical expression of the integration. In other words, a closed-form expression for the integration involving elementary functions (not other integrals) cannot be found. An example of such an integrand is $f(x) = \exp(-x^2)$ when evaluating the error function.
- The analytical expression is available, but it may be easier to compute a numerical approximation than to compute the analytical expression, such as infinite series or product, or if its evaluation requires a special function which is not available.

# Numerical Integration

- A large class of numerical integration methods can be derived by constructing interpolating functions which are easy to integrate.
- Typically these interpolating functions are polynomials.
- Interpolation with polynomials evaluated at equally-spaced points in yields the Newton–Cotes formulas.
  - Rectangular rule (zero-order)
  - Trapezoidal rule (first-order)
  - Simpson rule (second- and third-order)

$$\int_a^b f(x)dx \cong \int_a^b P_n(x)dx$$

- If we allow the intervals between interpolation points to vary, we find another group of quadrature formulas, such as the Gaussian quadrature formulas.

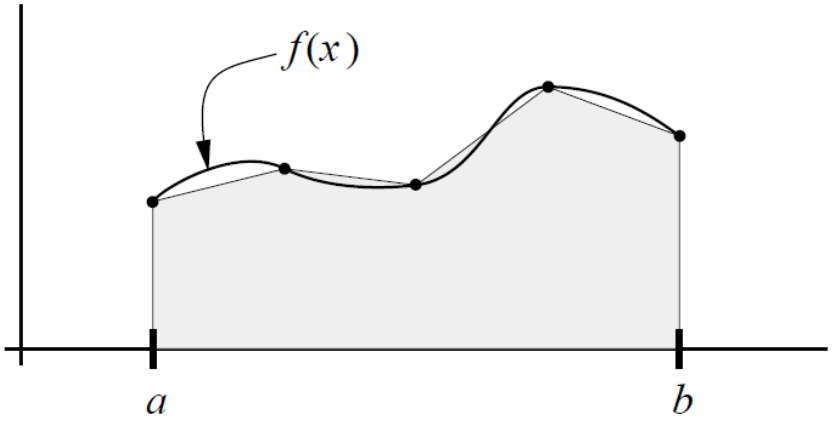
$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i).$$

$$\int_a^b f(x) \cong \sum_{i=1}^{n} w_i\, f(x_i)$$

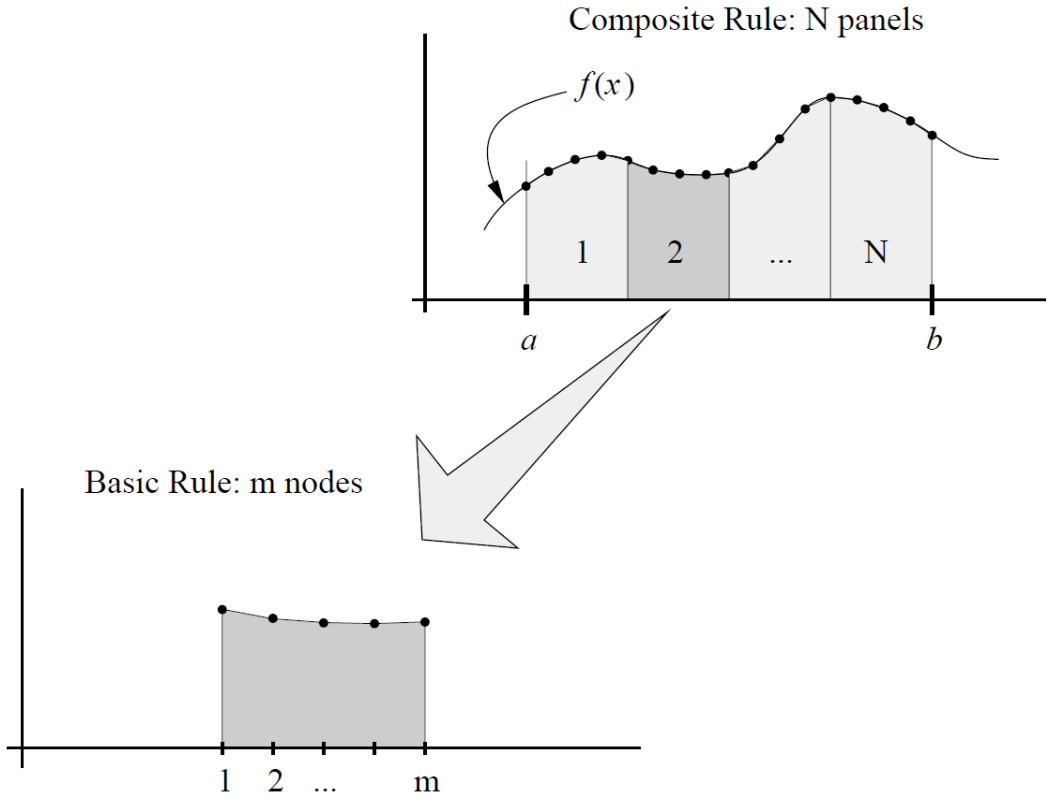
# Basic Ideas and Nomenclature

- Numerical integration is also called numerical quadrature.
- The term quadrature originally referred to the process determining the dimension of a square having the same area as other planar shape.
- This suggests a basic computational strategy of numerical integration:
  - To evaluate an integration, approximate the curve y=f(x) with a simple function that is easy to integrate;
  - The area under the simple curve is approximately equal to the area under f(x).

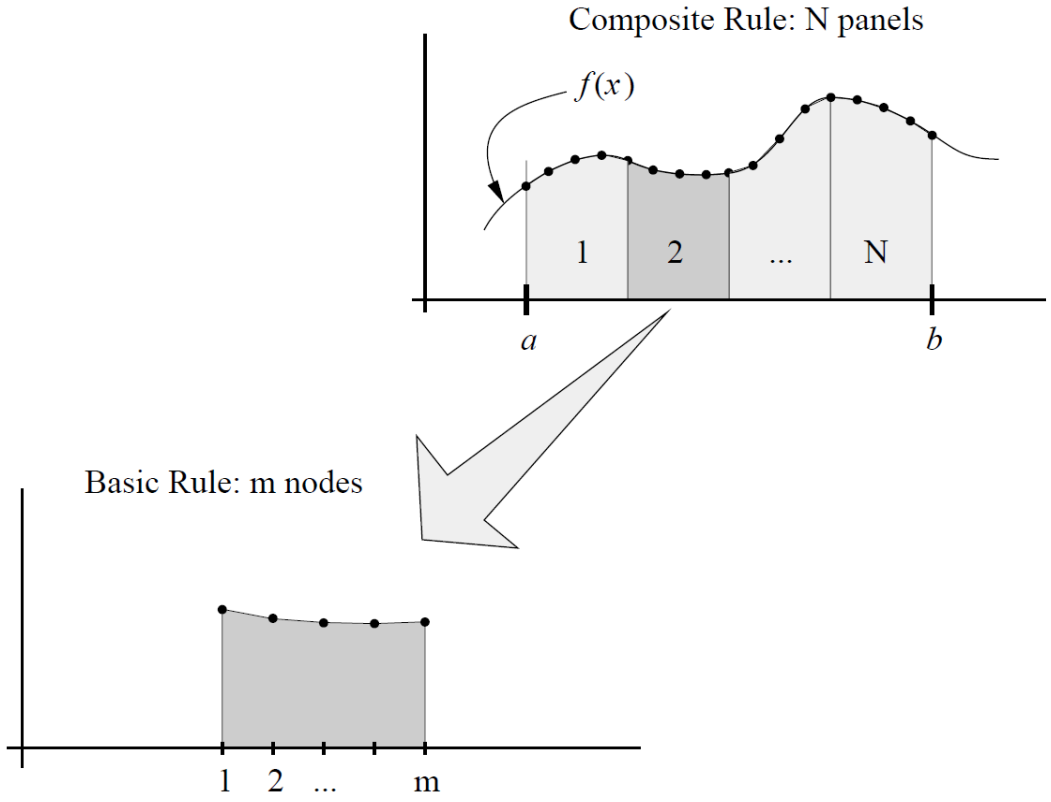
# Basic Ideas and Nomenclature




- The figure shows a piecewise-linear approximation to a function.
- The area under this approximation can be computed by summing the areas of the trapezoidal regions between the piecewise approximation and the x-axis.
- Polynomials are very easy to integrate, and theory of polynomial interpolation is well understood.
- Most numerical integration schemes involve constructing a polynomial interplant to f(x0 and then integrate the interpolant to obtain an approximation to the integral of f(x).

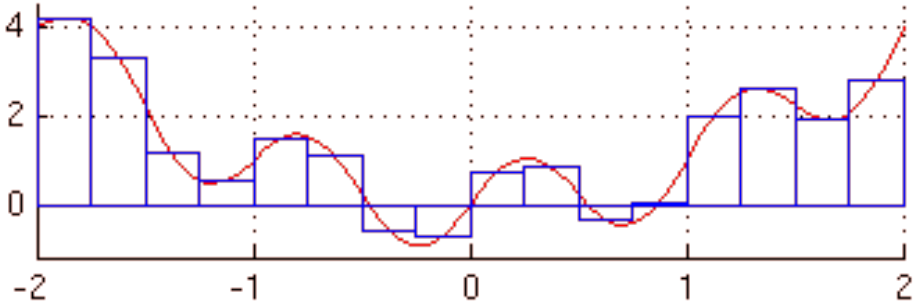



- A global interval [a,b] is divided into N panels.
- On each panel, a relatively low-degree polynomial approximation to f(x) is created.
- Integrating the polynomial approximation in the panel gives what is called a basic rule. A basic rule involves just enough (x,f(x)) pairs to define one segment of the piecewise polynomial.
- Applying the basic rule to each of the N panels and adding together the results gives what is called a composite rule or an extended rule.




- The location and number of the nodes within a panel determine many important characteristics of the basic rule.
- When the nodes are equally spaced, the resulting integration formulas are known as Newton-Cotes rules.
- In Contrast, Gaussian quadrature rules use nodes chosen as the zeros of orthogonal polynomials.
- Gaussian quadrature rules have a much smaller truncation error than the corresponding Newton-Cotes rules using the same number of nodes.
- Although the Gaussian quadrature rules are more complex to derive, they are not significantly harder to implement in a program.

# Rectangular Rule




For each interval Δx

$$\int_{x_i}^{x_{i+1}} d(x)dx \approx (x_{i+1} - x_i) f\left(\frac{x_i + x_{i+1}}{2}\right)$$

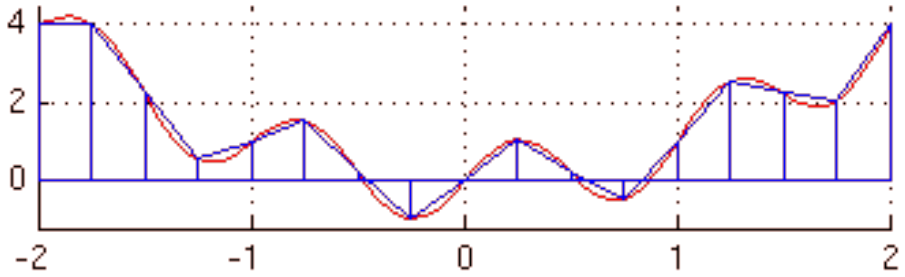
$$= f\left(\frac{x_i + x_{i+1}}{2}\right)\Delta x$$

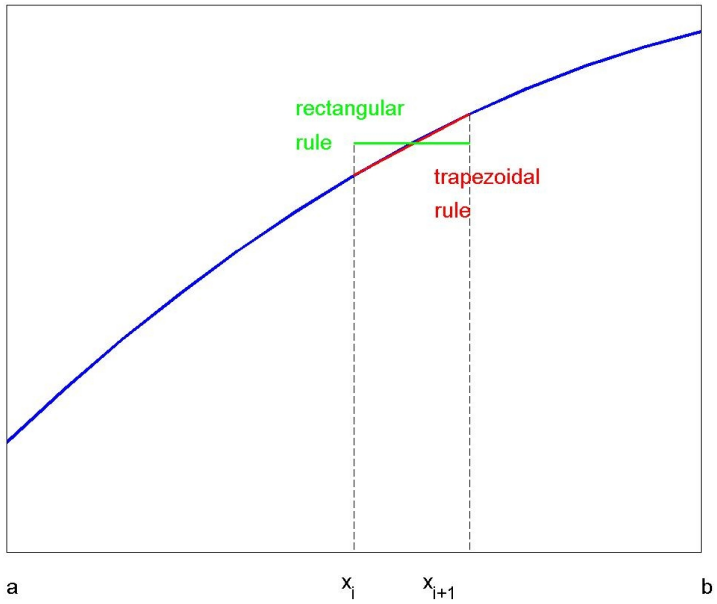
- The simplest approach to numerically integrating a function *f* over [*a*,*b*] is to divide the interval into *n* subdivisions of equal width, $\Delta x=(b-a)/n$ and approximate *f* in each interval.
- The simplest method of this type is to let the interpolating function be a constant function (a polynomial of order zero) which passes through the point $((x_i+x_{i+1})/2, f((x_i+x_{i+1})/2))$.

# Composite Trapezoidal Rule

Trapezoidal rule: for each interval Δx

$$\int_{x_i}^{x_{i+1}} f(x)dx \cong \frac{f(x_i)+f(x_{i+1})}{2}\Delta x = \frac{\Delta x}{2}(f_i + f_{i+1})$$



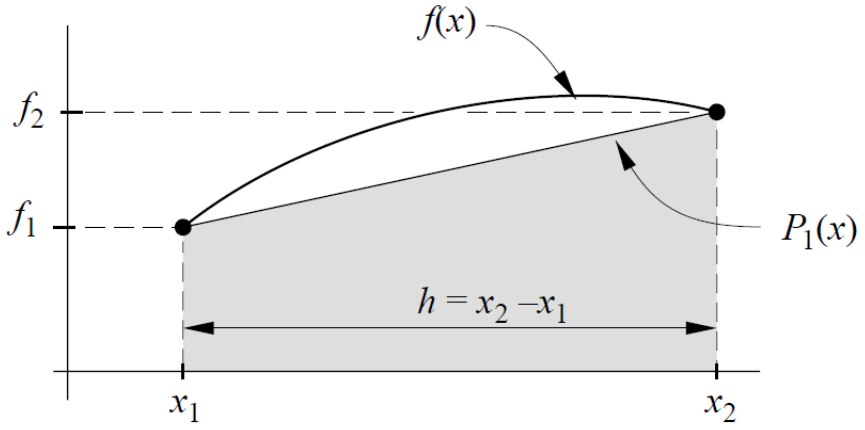

Composite trapezoidal rule:

If the interval [*a,b*] is subdivided into *n* subintervals of size Δ*x*, then, over the whole interval, the trapezoidal rule gives

$$\int_a^b f(x)dx = \frac{\Delta x}{2}(f_a + 2f_2 + 2f_3 + \ldots + 2f_n + f_b)$$

The formula is beautifully simple, and it can be applied to both equally spaced or non-equally spaced points.

Derive the trapezoidal rule from the Lagrange interpolating polynomial?




$$P_1(x) = \frac{x - x_2}{x_1 - x_2} f_1 + \frac{x - x_1}{x_2 - x_1} f_2 \qquad f(x) = P_1(x) + error$$

$$I = \int_{x_1}^{x_2} f(x)dx \approx \int_{x_1}^{x_2} P_1(x)dx = \int_{x_1}^{x_2} \left( \frac{x - x_2}{x_1 - x_2} f_1 + \frac{x - x_1}{x_2 - x_1} f_2 \right) dx$$

$$= -\frac{f_1}{h} \int_{x_1}^{x_2} (x - x_2)\, dx + \frac{f_2}{h} \int_{x_1}^{x_2} (x - x_1)\, dx = \frac{1}{2}(f_1 + f_2)h$$

# Derive Trapezoidal Rule Using Newton-Cotes Formulas

- Newton-Cotes formulas: integrate the interpolating polynomial for equally spaced points.
- Trapezoidal rule is to approximate the function in each interval by a linear segment.
- The interpolating polynomial is of the first order.

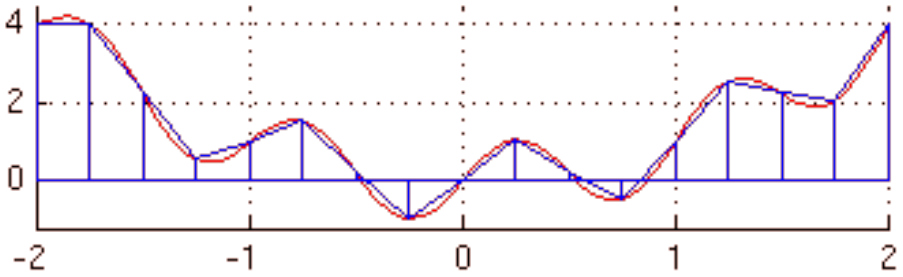
$$P_1(x) = f_i + s\Delta f_i \qquad s=(x-x_i)/\Delta x \qquad dx=\Delta x\, ds.$$

$$\int_{x_0}^{x_1} f(x)dx \cong \Delta x \int_{s=0}^{s=1} (f_0 + s\Delta f_0)ds = \Delta x f_0 s\,|_0^1 + \Delta x \Delta f_0 \frac{s^2}{2}|_0^1 = \Delta x(f_0 + \frac{1}{2}\Delta f_0)$$

$$= \frac{\Delta x}{2}[2f_0 + (f_1 - f_0)] = \frac{\Delta x}{2}(f_0 + f_1)$$

You can use the same technique for higher order polynomials

# Local Error and Global Error




- Truncation errors have an important role in numerical
  - Local error for each interval
  - Global error for the entire interval
- The truncation error of a quadrature rule can be measured by applying it to an integral that has a known analytical formula.
- Theoretical formulas for the truncation error are also available from numerical analysis.
- Theoretical formulas show that the truncation error for each method decreases in a precise way as the number of nodes used in the integration is increased.
- If we found the limiting value of the sum as the widths of the intervals approach zero, we would have the exact value of the integral.
- Comparing the measured error with the theoretical formula for the error allows for the verification of the code implementing a numerical integration method.

# Local Error of a Single Subinterval

$$P_1(x) = f_i + s\Delta f_i$$

First-order polynomial

$$\int_{x_0}^{x_1} f(x)dx \cong \Delta x \int_{s=0}^{s=1} (f_0 + s\Delta f_0)ds$$

$$P_2(x) = f_i + s\,\Delta f_i + \frac{s(s-1)}{2}\Delta^2 f_i$$

Second-order polynomial

The error is given by $(\Delta x^2/2)(s)(s-1)f''(\xi)$, where $\xi$ is some point in the interval $(x_0,x_1)$.

$$Local\ Error = \int_{x_0}^{x_1} \frac{s(s-1)}{2}(\Delta x)^2 f''(\xi)dx = (\Delta x)^3 f''(\xi)\int_0^1 \frac{s(s-1)}{2}ds$$

$$= (\Delta x)^3 f''(\xi)\left(\frac{s^3}{6} - \frac{s^2}{4}\right)\Big|_0^1 = -\frac{1}{12}(\Delta x)^3 f''(\xi) \quad x_0 \le \xi \le x_1$$

The local error is $O(\Delta x^3)$

# Global Error of the Entire Interval

$$Gloabl\ Error = -\frac{1}{12}(\Delta x)^3\left[ f''(\xi_1) + f''(\xi_2) + ... + f''(\xi_n)\right]$$

Each $\xi_i$ is found in the n successive subinterval. If we assume that f''(x) is continuous on (a,b), there is some value of x in (a,b) – say x=ξ – at which the value of the sum is equal to n f''(ξ), then

$$Gloabl\ Error = -\frac{1}{12}(\Delta x)^3 n f''(\xi) = -\frac{(b-a)}{12}(\Delta x)^2\ f''(\xi) = O(\Delta x)^2$$

This explains why the numerical integral becomes more accurate when the width of the subinterval becomes smaller. The global error is larger than the local error.

# Exercise

- Calculate the analytical solution of $\int_{1.8}^{3.4} e^x dx$
- Use the trapezoidal rule to calculate

  Take the width of each subinterval as 0.2.
- Calculate the exact error.
- Analyze the global error, and compare it with the exact error.
- Use the Simpson's 1/3 rule with 2, 4, 6, … panels, until the numerical integration converge to the three decimal places.
- Apply the Simpson's 3/8 rule with 3, 6, 9,…panels, until the numerical integration converge to the three decimal places.

# Simpson's Rules

- Apply the Newton-Cotes formulas to the second- and third-order interpolating polynomials.
- Simpson's 1/3 rule ($O(\Delta x^5)$)

$$\int_{x_0}^{x_2} f(x)dx \cong \Delta x \int_0^2 (f_0 + s\Delta f_0 + \frac{s(s-1)}{2}\Delta^2 f_0)ds = \Delta x(2f_0 + 2\Delta f_0 + \frac{1}{3}\Delta^2 f_0)$$

$$= \frac{\Delta x}{3}(f_0 + 4f_1 + f_2)$$

- Composite formulas ($O(\Delta x^4)$)

$$\int_a^b f(x)dx = \frac{\Delta x}{3}(f_a + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \ldots + 4f_{n-1} + f_b)$$

Because the method uses pairs of panels, the number of panels (subintervals) must be even.

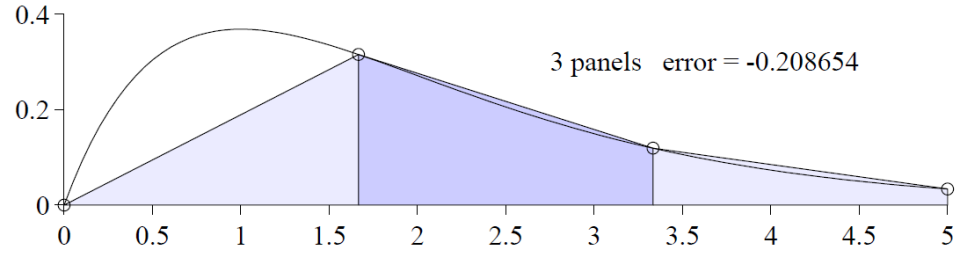
# Simpson's Rules

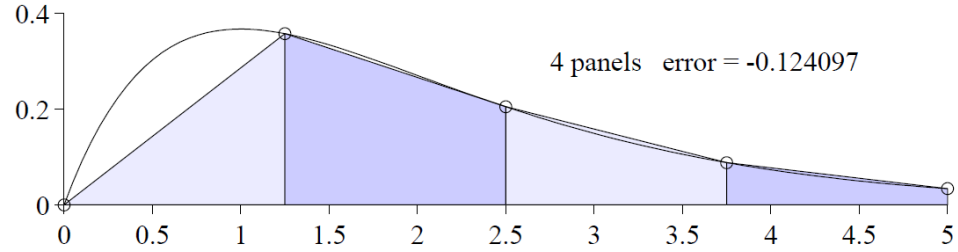
- Apply the Newton-Cotes formulas to the third-order interpolating polynomial gives the Simpson's 3/8 rule ($O(\Delta x^5)$).

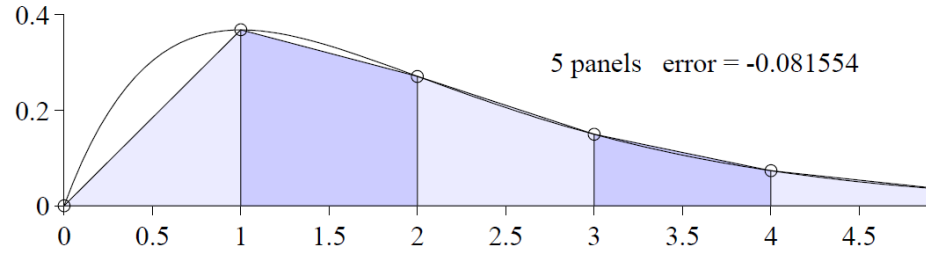
$$\int_{x_0}^{x_3} f(x)dx = \int_{x_0}^{x_3} P_3(x)dx = \frac{3\Delta x}{8}(f_0 + 3f_1 + 3f_2 + f_3)$$

- Adding the extra point into the formula does not increase the order of accuracy of the approximation.

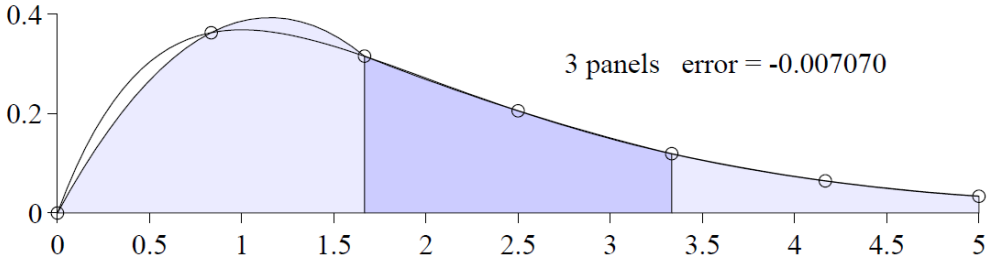
# Trapezoidal rule approximation

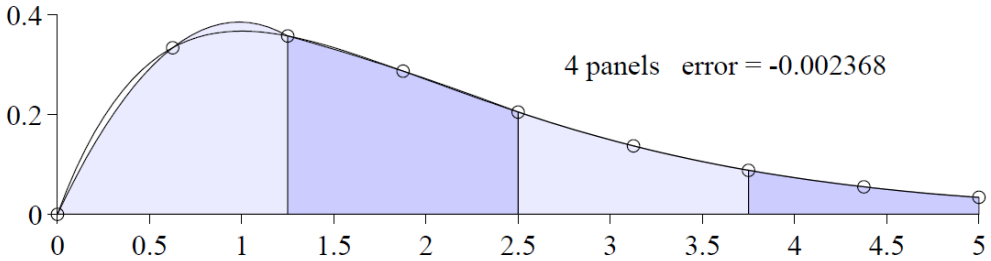


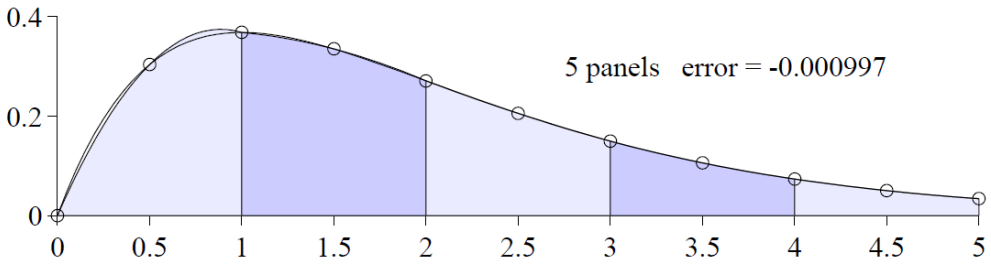




# Simpson rule approximation



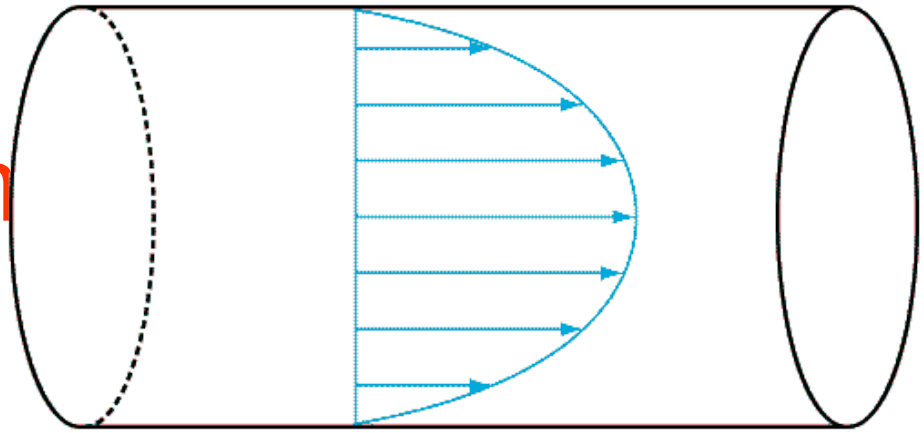



# Exercise

- Calculate the analytical solution of $\int_{1.8}^{3.4} e^x dx$
- Use the trapezoidal rule to calculate

  Take the width of each subinterval as 0.2.
- Calculate the exact error.
- Analyze the global error, and compare it with the exact error.
- Use the Simpson's 1/3 rule with 2, 4, 6, … panels, until the numerical integration converge to the three decimal places.
- Apply the Simpson's 3/8 rule with 3, 6, 9,…panels, until the numerical integration converge to the three decimal places.

# Velocity Distribution



- Laminar flow

  Still parabolic, but the maximum is at the surface

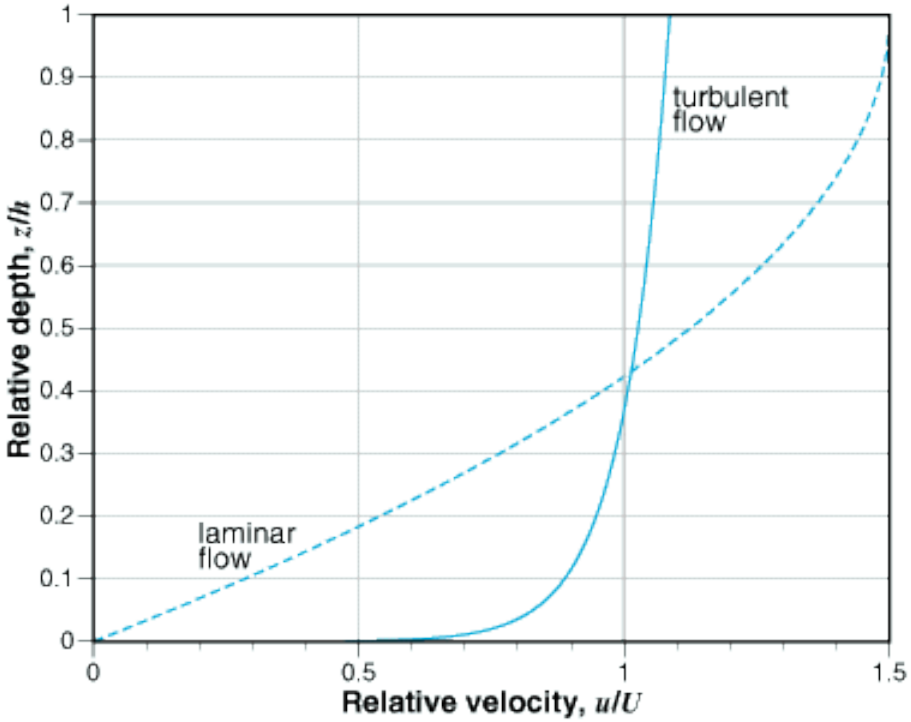
- Turbulent flow

  Logarithmic

$$u(z) = \sqrt{gR_H S}\left[2.5 ln\left(\frac{z}{k_r}\right) + 8.5\right]$$

$k_r$ is a channel roughness parameter

Gives u distribution in z direction



# Mean Velocity

Mean velocity in a channel is the sum, or integral, of the velocity at each point in a cross-section divided by the cross-sectional area:

$$U = \frac{1}{A}\int_0^w \int_0^h u(y,z)\,dz\,dy$$

If the channel is wide and rectangular, then we can reasonably assume that the velocity is the same at each point across the channel. In this case integrating across the channel is equivalent to multiplying by channel width

$$U = \frac{1}{wh} w \int_0^h u(z)\,dz = \frac{1}{h}\int_0^h u(z)\,dz$$

$$U = \sqrt{ghS}\left[2.5\ln\left(\frac{h}{k_r}\right)+6.0\right]$$

# Gaussian Quadrature

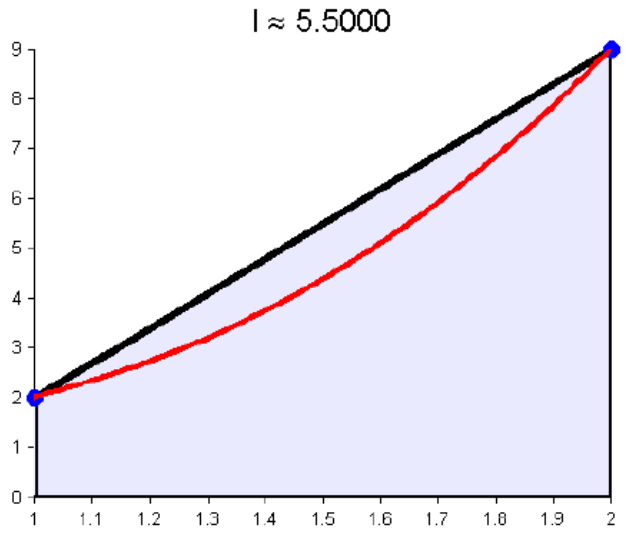
- In the Newton-Cotes methods, the $x_i$ values are predetermined before the numerical integration.
- The general form is $$\int_a^b f(x) \cong \sum_{i=1}^{n} w_i f(x_i)$$

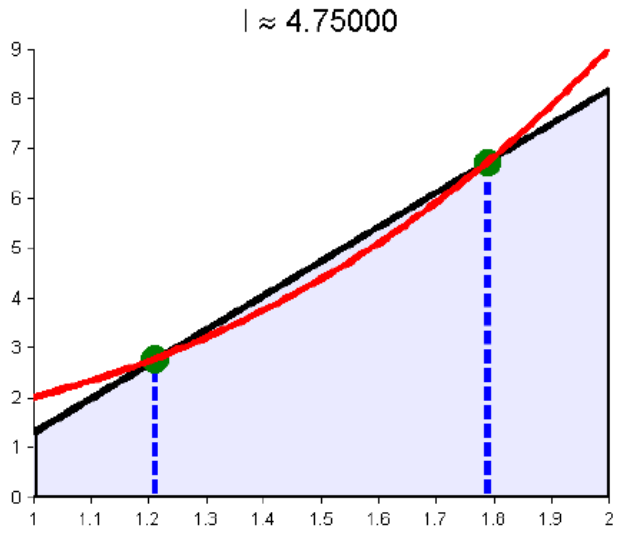
where $w_i$ are weights assigned to values of $f(x_i)$ in the integration formulas.

- Gauss observed if we remove the requirement that the function be evaluated at predetermined x-values, a three-term formula will contain six parameters (the three x-values are now unknown, plus three weights) and should correspond to an interpolating polynomial of degree 5.
- It can be applied only when f(x) is known explicitly so that f(x) can be evaluated at any desired value of x.

Consider

$$\int_1^2 x^3 + 1\, dx = 4.75$$




Gaussian Quadrature:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

- We have n points of $x_i$
- We have n real coefficients
- 2n unknowns can be used to exactly interpolate and integrate polynomials of degree up to 2n-1.

# Gaussian Quadrature

- To simplify the calculations, we will evaluate the integrals over the interval [-1 1].
- A simple case: two-term formulas with four unknowns

$$\int_{-1}^{1} f(x)dx = w_1 f(x_1) + w_2 f(x_2)$$

should correspond to an interpolating polynomial of degree 3, including $f(x)=x^3$, $f(x)=x^2$, $f(x)=x$, and $f(x)=1$. Assume the function is a cubic polynomial: $f(x)=a_0+a_1x+a_2x^2+a_3x^3$.

This implies that:

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{1} (a_0 + a_1x + a_2x^2 + a_2x^3)dx$$

$$= w_1(a_0 + a_1x_1 + a_2x_1^2 + a_2x_1^3) + w_2(a_0 + a_1x_2 + a_2x_2^2 + a_2x_2^3)$$

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{1} (a_0 + a_1 x + a_2 x^2 + a_2 x^3)dx$$

$$= w_1(a_0 + a_1 x_1 + a_2 x_1^2 + a_2 x_1^3) + w_2(a_0 + a_1 x_2 + a_2 x_2^2 + a_2 x_2^3)$$

$$a_0\left(w_1 + w_2 - \int_{-1}^{1} dx\right) + a_1\left(w_1 x_1 + w_2 x_2 - \int_{-1}^{1} x dx\right) +$$

$$a_2\left(w_1 x_1^2 + w_2 x_2^2 - \int_{-1}^{1} x^2 dx\right) + a_3\left(w_1 x_1^3 + w_2 x_2^3 - \int_{-1}^{1} x^3 dx\right) = 0$$

Since the coefficients of a are arbitrary, the terms in parenthesis must be zero.

$w_1 = w_2 = 1$, and $x_2 = -x_1 = \sqrt{1/3} = 0.5773$.

$$\int_{-1}^{1} f(x)dx \cong f\left(\frac{-1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

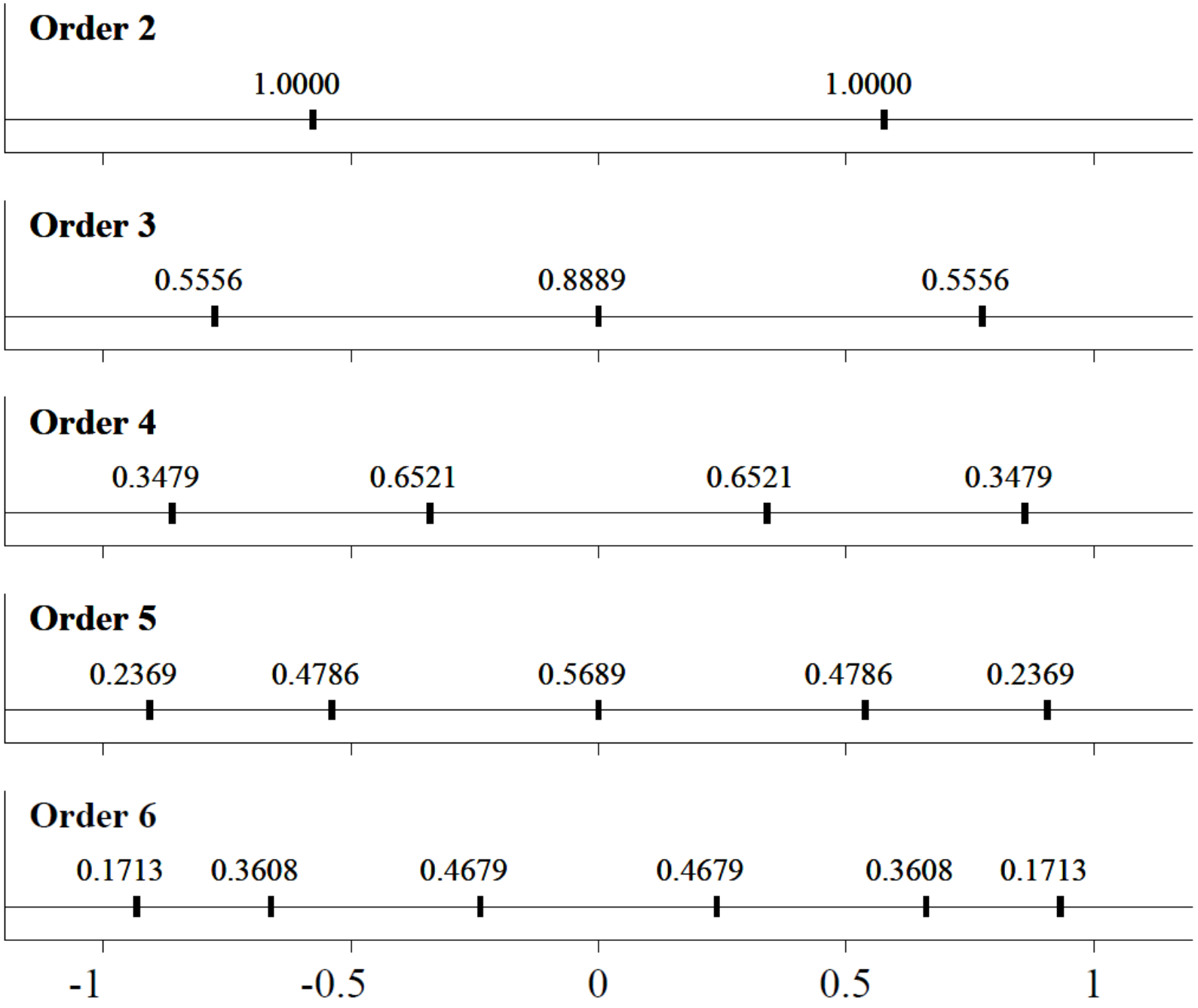
$$\int_{-1}^{1} x^3 dt = 0 = w_1 x_1^{\,3} + w_2 x_2^{\,3}$$

$$\int_{-1}^{1} x^2 dx = 2/3 = w_1 x_1^{\,2} + w_2 x_2^{\,2}$$

$$\int_{-1}^{1} x dx = 0 = w_1 x_1 + w_2 x_2$$

$$\int_{-1}^{1} dx = 2 = w_1 + w_2$$

Determine the one-term Gaussian Quadrature formulas


Order 2
1.0000
1.0000
Order 3
0.5556
0.8889
0.5556
Order 4
0.3479
0.6521
0.6521
0.3479
Order 5
0.2369
0.4786
0.5689
0.4786
0.2369
Order 6
0.1713
0.3608
0.4679
0.4679
0.3608
0.1713
-1
-0.5
0
0.5
1

# General Expression

- If the limits are [*a,b*] rather than [-1,1], then it is necessary to use the linear transformation

  t= [(*b*-*a*)*x*+*b*+*a*]/2, *dt* = [(*b*-*a*)/2]*dx*

$$\int_a^b f(t)\,dt = \frac{(b-a)}{2}\int_{-1}^{1} f\left(\frac{(b-a)\,x+b+a}{2}\right)dx$$

General Expression

$$\int_{-1}^{1} f(x)\,dx \cong \sum_{i=1}^{n} w_i f_i(x_i) = \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right)$$

# Exercise

Use the two-term Gaussian quadrature formulas to evaluate

$$\int_0^{\pi/2} \sin x dx$$

Divide the domain into two panels (use equal size for convenience), and use the two-term Gaussian quadrature formulas in each panel to evaluate the integral.