MA3457/CS4033:
Numerical Methods for Calculus and Differential Equations

Course Materials

P A R T I

B'14
2014-2015

Course Objectives

Numerical Methods is the course about

> ***techniques by which mathematical problems are formulated so that they can be solved with arithmetic operations***.

The primary goal of the course:

(1) to introduce the students to a ***range of numerical algorithms related to problems in Calculus and Differential Equations***,

(2) to review their ***fundamental principles***, and

(3) to illustrate their ***applications***.

Upon completing this course you will be able

- *to use numerical procedures for solving applied problems* and [*when applying the algorithms to particular practical scenarios*] *control their performance*

Main Topics of the Course

1. Interpolation	7 classes
Polynomial & Spline Interpolation **Numerical Differentiation**	
2. Approximation	4 classes
3. Numerical Integration	6 classes
4. Initial Value Problems	4 classes
5. Boundary Value Problems	3 classes

Available MATLAB Resources

MATLAB Help: Video Tutorials/Demos on Specific Topics and Features

MathWorks Website: Interactive MATLAB & Simulink Based Tutorials
(http://www.mathworks.com/academia/student_center/tutorials/)
Strongly Recommended: Interactive MATLAB Tutorial

MathWorks Recorded Webinars
(http://www.mathworks.com/company/events/webinars/index.html)
Recommended: Introduction to MATLAB
(http://www.mathworks.com/videos/introduction-to-matlab-81592.html)

Introduction to MATLAB – Short Course (November, 2014)

Instructor: *Adriana Hera* (ahera@wpi.edu)

Details at: http://www.wpi.edu/webapps/regi/sesa.html

- First MA3457/CS4033 Conference (Wed, Oct 29, 2 pm) – **Intro to MATLAB** by *Kyle Dunn*

Key Introductory Notes

From the earliest times, one of the most important aspects of mathematical study – ***the search for solutions of real-world problems***.

In many applications:

- an exact solution may be unattainable, or
- a solution may not give the answer in a convenient form.

So people are usually OK with:

- finding ***good approximate results***, *and*
- *spending a* ***reasonable amount of computational effort*** to get them.

Historical Facts

Mathematics is the oldest science on Earth, and its branch related to ***Numerical Methods*** (NM) is one of the oldest in math.

Some NM may be ***older than many key elements of many other human cultures***:

- ***The Babylonians*** (3700+ years ago) knew how to find numerical solutions of quadratic equations and approximations to the square root of an integer.
- Contributions to NM were made in ***China*** between 100 and 1000.
- Important findings have been made by ***ancient Greeks*** (~1800 y. ago).
- ***Middle Eastern*** and particularly ***Persian scholars*** (900 and 600 y. ago)
- ***European mathematicians*** – especially from ~1200 to ~1800, e.g., Fibonacci, Kepler, Newton, Taylor, Lagrange, Leibniz, etc. etc.

Many issues that confront a scientist or engineer who uses numerical methods ***are the same today as throughout the history of the subject***.

In the past and nowadays, two primary considerations are:

- **the computational effort required, and**
- **the accuracy of the resulting solution.**

1. INTERPOLATION

Polynomial Interpolation

Methods of interpolations – ***methods for representing a function based on knowledge of its behavior at certain discrete points***.

From this information:

- obtain **estimates of functional values *at other points***, or
- use **the closed-form representation of the function as the basis for other numerical techniques** (e.g., numerical differentiation or integration).

Conceptually,

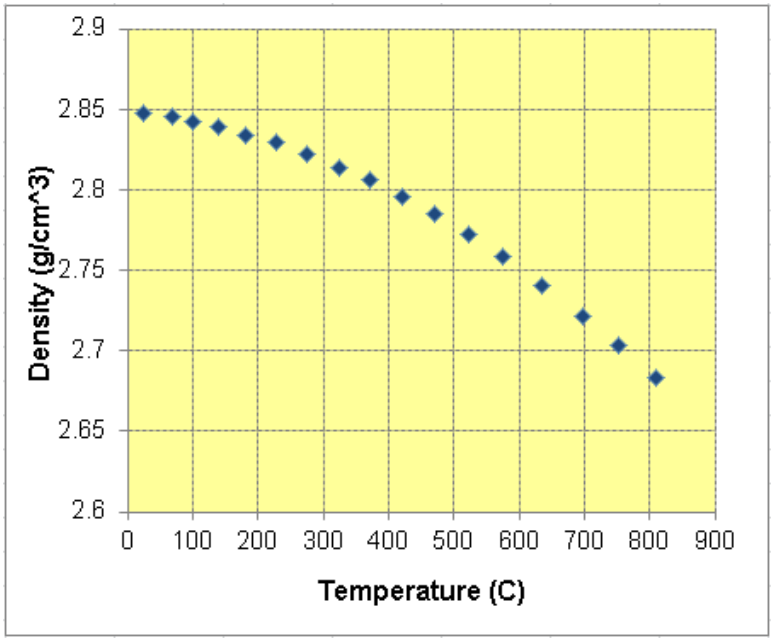
- interpolation produces ***a function that matches the given data exactly***;
- interpolating function provides a ***good approximation to the data values at intermediate points***.

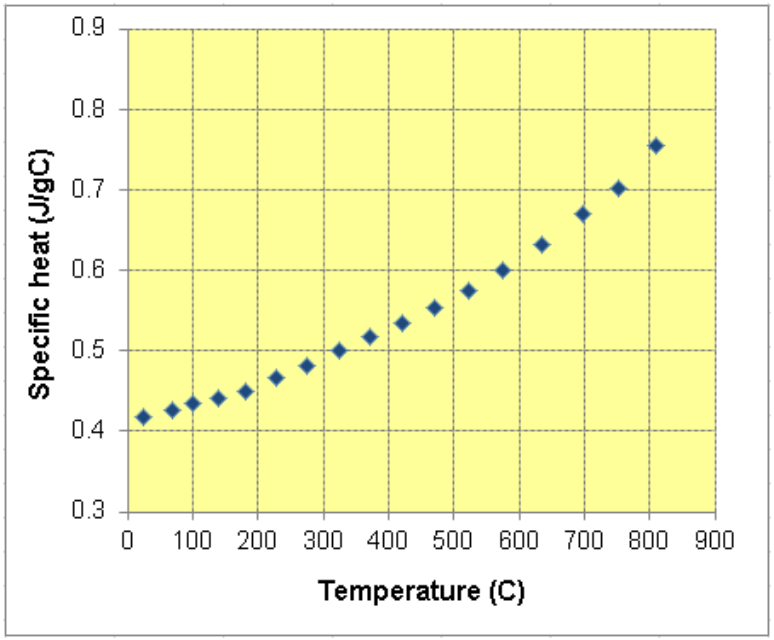
The data comes from *two sources*:

- measured (experimental) values, or
- computed values (obtained by other numerical methods).

Motivation for Interpolation

Temperature Characteristics of Thermal Parameters of Zirconia





Experimental data by Ceralink, Inc., from [1].

For modeling phenomena with variable/dynamic parameters, *COMSOL Multiphysics* uses ***functional/analytical representations***.

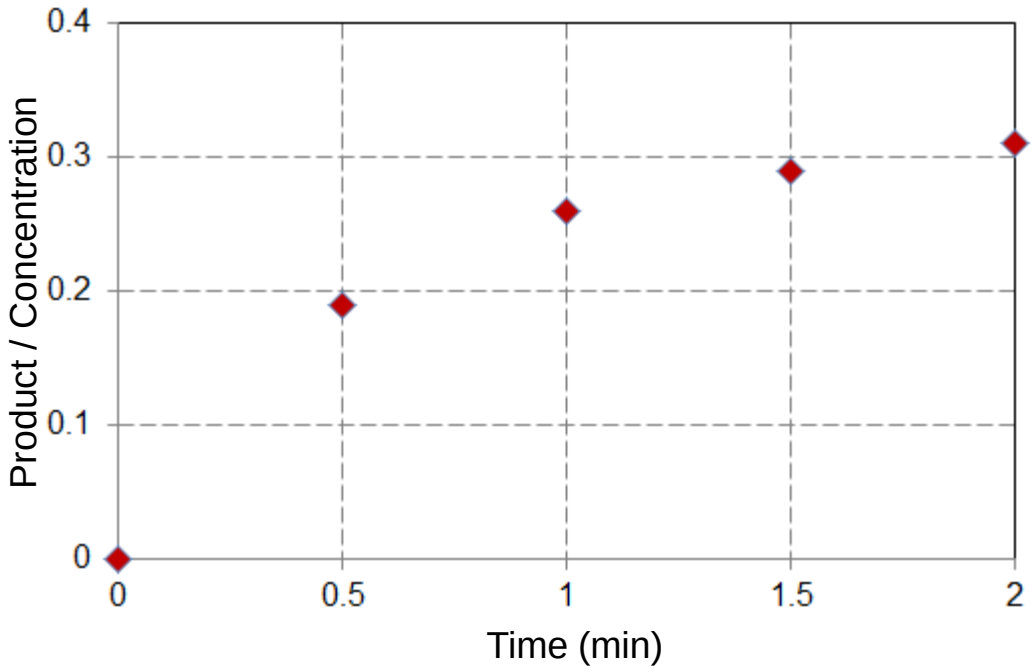
[1]. V.V. Yakovlev, S.M. Allan, M.L. Fall, and H.S. Shulman, Computational study of thermal runaway in microwave processing of zirconia, In: *Microwave and RF Power Applications*, J. Tao, Ed., Cépaduès Éditions, 2011, pp. 303-306.

PROJECT TEAMS

Desilva, Thomas Ryan	15	CS	Wen, Xi	15	ECS
Christiansen, Marc Anton	16	PHE	Wigell, Rachel Ellen	15	CS
LaRose, Danielle Marie	15	CS	Ma, Mingkun	17	MA
Gao, Jiali	15	MAC	Jackman, James Deane	15	CS
Murray, Nicholas D	15	CS	Taubert, Alex Wayne	15	MAC
Coia, Cecily Ann	16	MAC	Li, Xia	15	CS
Liu, Yuchen	16	CS	Ellison, Samuel Joseph	16	MAC
Bauer, Andrew C	TR	MA	Carmichael, Rose Tamara	16	MA
Xue, Zhaokun	15	MA	Graham, Alyssa Marie	14	CS
Minchev, Kliment Sashev	15	ME	Li, Tianyu	15	MA
Delisi, Eric Vincent	16	MA	Fung, Adria	15	RBE
Xie, Jiaxun	16	ME	Weber, Emily Catherine	17	MA
Perrone, Michael Alexande	16	MA	Van Herwarde, Grania Maev	15	MGE
Larkins, Riley David	15	RBE	Dymov, Khasan Muratovich	17	MA
Kowalski, Miranda Berube	17	MA	O'Hara, James Kevin	17	AE
Leiro, Alejandro Emmanuel	16	PH	Zhang, Xinzhe	16	MA
Jiang, Lin	17	IE			
Arnold, Zachary Philip	15	CS			

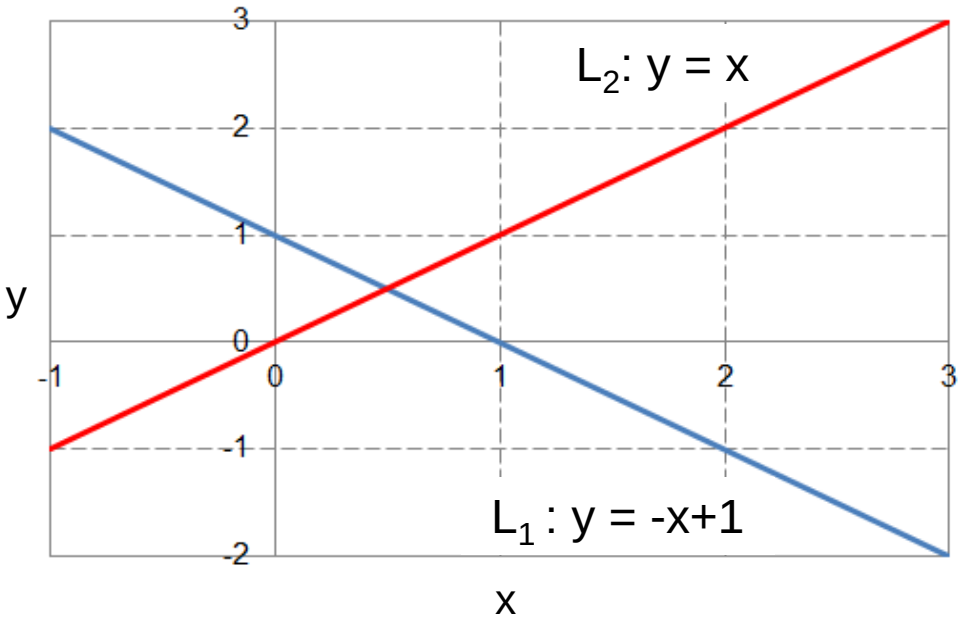
Introductory Illustration

Observed Concentration of the Product – Chemical Reaction



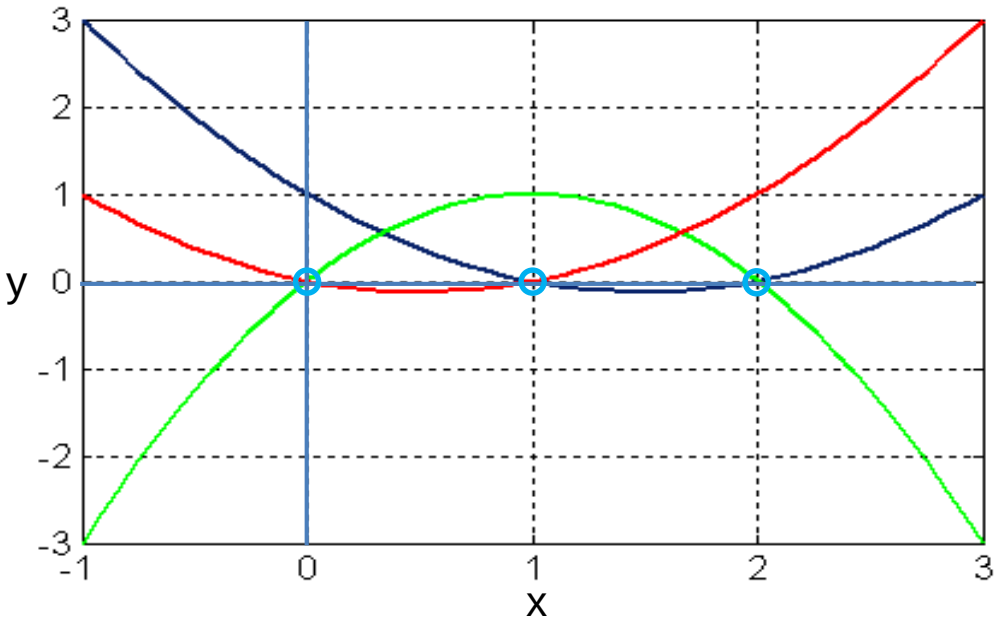
The Basis Functions – Linear Polynomials

Graphs of L_1 & L_2



The Basis Functions – Quadratic Polynomials

Graphs of C_1, C_2 and C_3



C_3 :
$y = x(x-1)/2$

C_1 :
$y = (x-1)(x-2)/2$

C_2:
$y = x(2-x)$

$x_1 = 0;\ \ x_2 = 1;\ x_3 = 2;$

Lagrange Interpolation – MATLAB Script (1)

LIBRARY OF MATLAB PROCEDURES

`Lagrange_coef`

Finds coefficients of the Lagrange interpolating polynomials

```
function c = Lagrange_coef(x,y)
%
% Calculate coefficients of Lagrange interpolating polynomial
%
n = length(x);
for k = 1 : n
      d(k) = 1;
      for i = 1 : n
            if i ~= k
                  d(k) = d(k) * (x(k) - x(i));
            end
            c(k) = y(k) / d(k);
      end
end
```

Lagrange Interpolation – MATLAB Script (2)

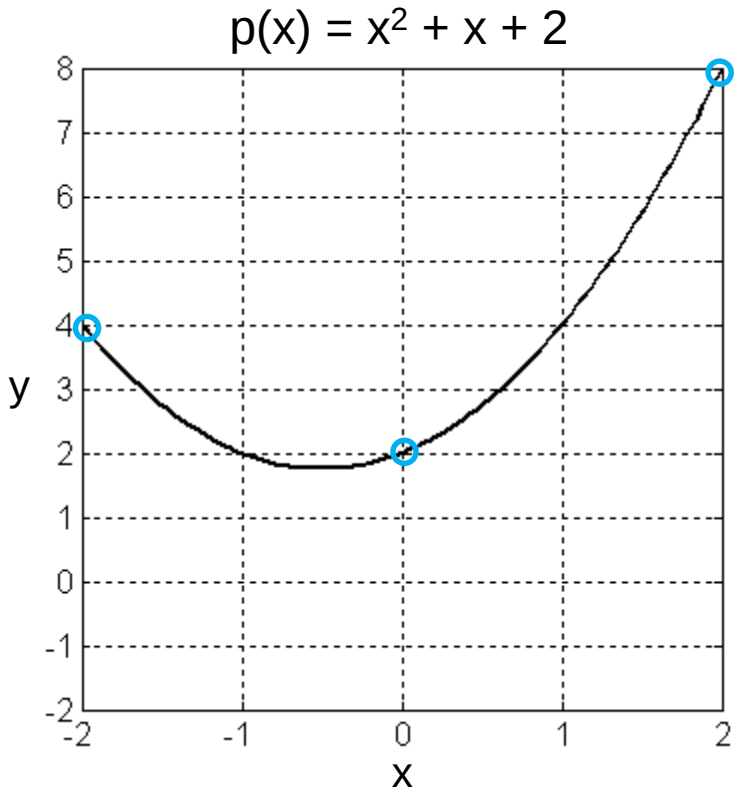
LIBRARY OF MATLAB PROCEDURES

Lagrange_Eval

Finds the Lagrange interpolating polynomial at $x = t$

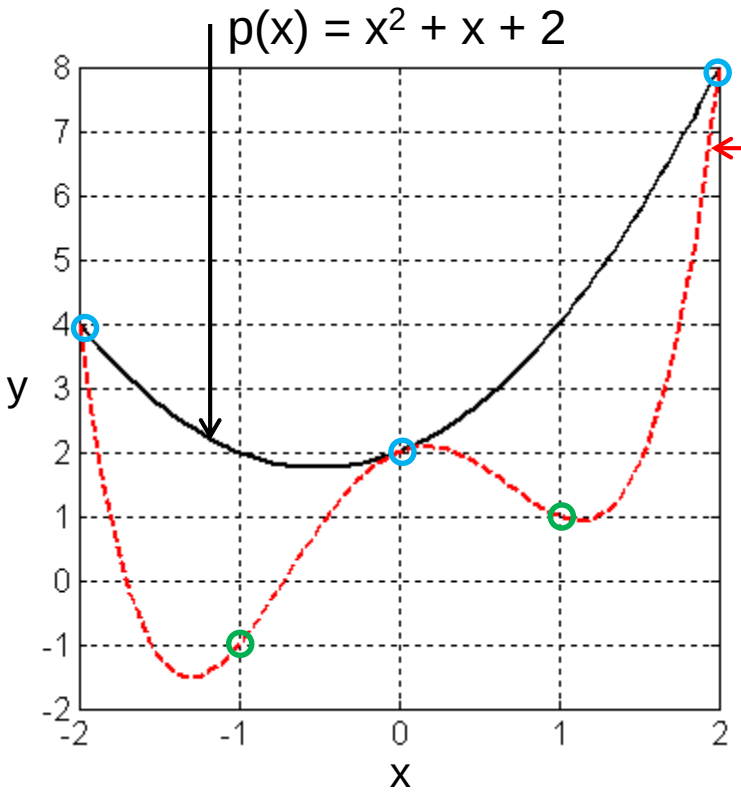
```
function p = Lagrange_Eval(t, x, c)
%
% Evaluate the Lagrange interpolation polynomial at point x = t
%
m = length(x);
%
for i = 1 : length(t)
      p(i) = 0;
      for j = 1 : m
            N(j) = 1;
            for k = 1 : m
                  if (j ~= k)
                        N(j) = N(j) * (t(i) - x(k));
                  end
            end
            p(i) = p(i) + N(j) * c(j);
      end
end
```

Lagrange Interpolation – Example with Three Points



$(x_1, y_1) = (-2, 4)$; $(x_2, y_2) = (0, 2)$; $(x_3, y_3) = (2, 8)$

Lagrange Interpolation – Example with Five Points



$$p(x) = \frac{1}{6}(x)(x+1)(x-1)(x-2) + \frac{1}{2}(x+2)(x+1)(x-1)(x-2) - \frac{1}{6}(x+2)(x)(x-1)(x-2) + \frac{1}{6}(x+2)(x)(x+1)(x-2) - \frac{1}{3}(x+2)(x)(x+1)(x-1)$$

$(x_1, y_1) = (-2, 4)$; **$(x_2, y_2) = (-1, -1)$;** $(x_3, y_3) = (0, 2)$;
$(x_4, y_4) = (1, 1)$; $(x_5, y_5) = (2, 8)$

Lagrange Interpolation – Summary

Observations

For the Lagrange Interpolation, the ***more points you take, the higher order of the polynomial you work with***.

Polynomials of very high degree raise the difficulties:

- *It is necessary to* ***completely rework the problem with the new expanded data set*** – a bigger computational effort is required, and
- ***Appearance of the curve is not improved*** - rather, it becomes fluctuating/non-smooth

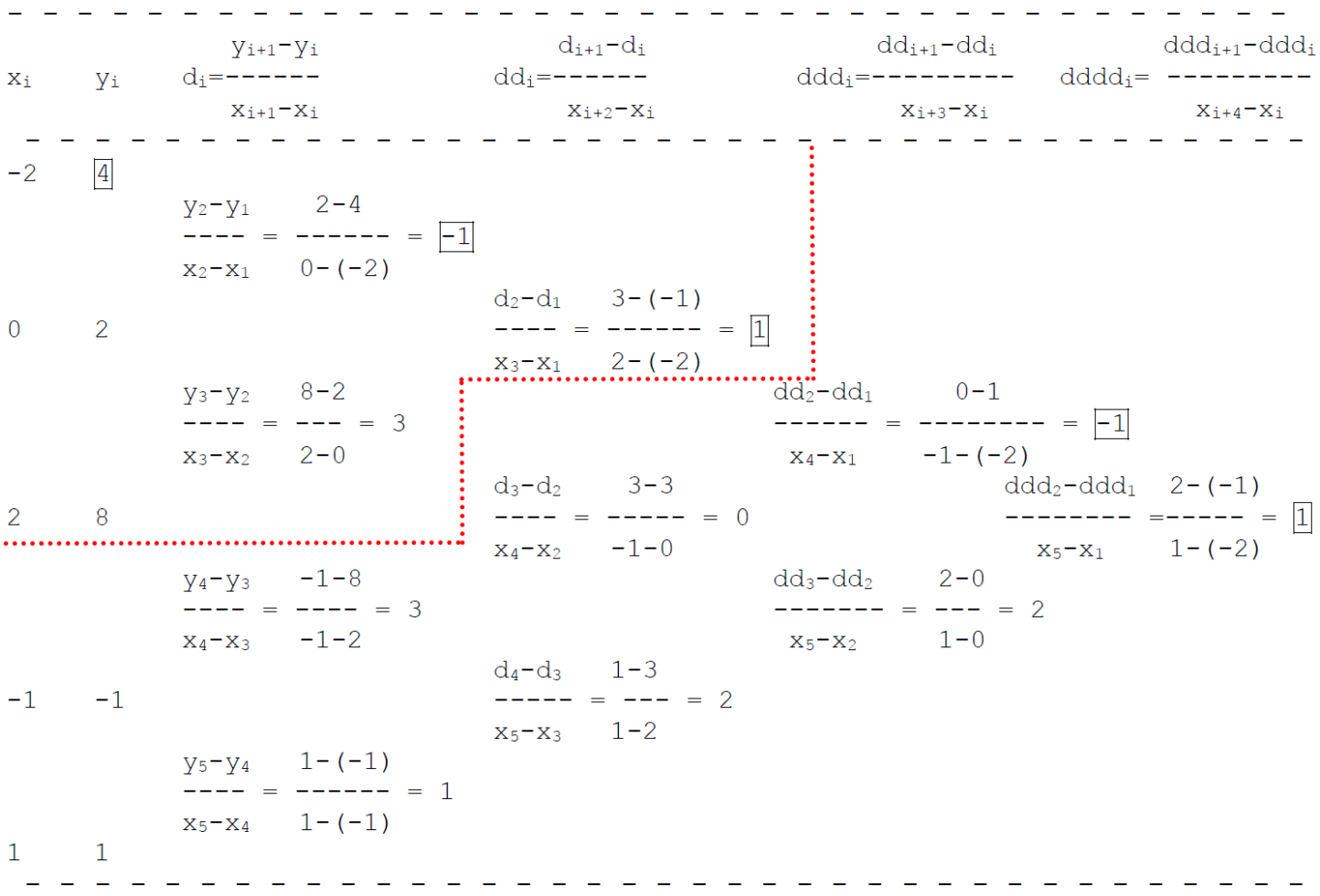
Preliminary Conclusion

1) The Lagrange Interpolation is particularly convenient ***when the same values of the independent variable*** x may occur in different applications (with only y values changed).
2) Lagrange form is not convenient when:
   - *additional data points may be added to the problem,* or
   - *appropriate degree of the interpolating polynomial is unknown* [i.e., when it may be better to use less than a full set of data].

Newton Interpolation – 5-Point Example

$(x_1, y_1) = (-2, 4)$; **$(x_2, y_2) = (-1, -1)$;** $(x_3, y_3) = (0, 2)$; **$(x_4, y_4) = (1, 1)$;** $(x_5, y_5) = (2, 8)$

Divided-Difference Table



$$p(x) = a_1 + a_2(x-x_1) + a_3(x-x_1)(x-x_2) + a_4(x-x_1)(x-x_2)(x-x_3) + a_5(x-x_1)(x-x_2)(x-x_3)(x-x_4)$$

$$= 4 - (x+2) + (x+2)(x) - (x+2)(x)(x-2) + (x+2)(x)(x-2)(x+1)$$

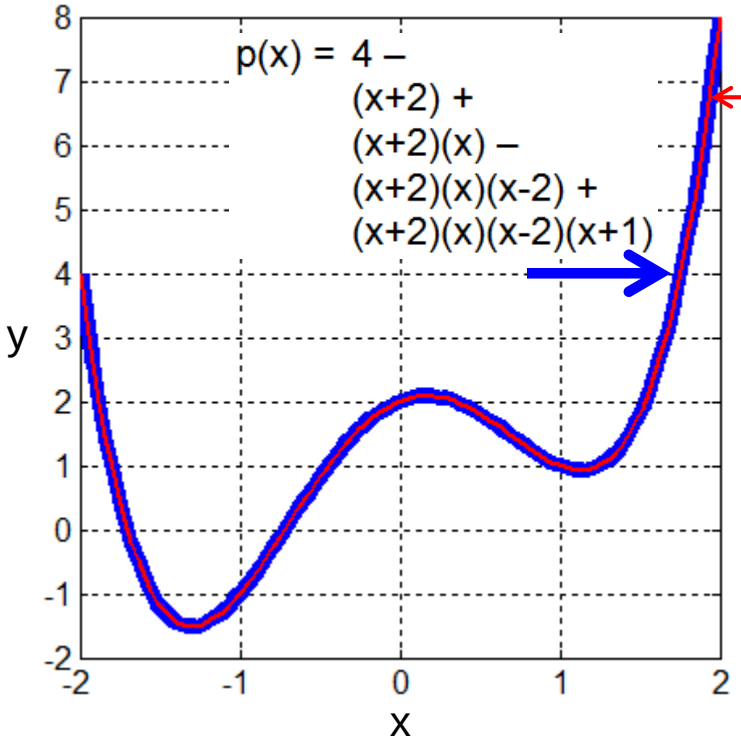
Newton Interpolation – Example with Five Points

Newton form:

$$p(x) = 4 - (x+2) + (x+2)(x) - (x+2)(x)(x-2) + (x+2)(x)(x-2)(x+1)$$

Lagrange form:

$$p(x) = \frac{1}{6}(-x)(x+1)(x-1)(x-2) + \frac{1}{2}(-(x+2)(x+1)(x-1)(x-2) + \frac{1}{6}(-(x+2)(x)(x-1)(x-2) - \frac{1}{6}(-(x+2)(x)(x+1)(x-2) + \frac{1}{3}(-(x+2)(x)(x+1)(x-1)$$



- *Compared to the polynomial obtained with the Lagrange approach*, Newton interpolating expression looks *different*, however, ***the two are in fact equivalent***. This can be shown algebraically; here, the equivalence is demonstrated *graphically* by plotting both functions (**Newton Polynomial** and **Lagrange Polynomial**) in the same axes.

Newton Interpolation – MATLAB Script (1)

LIBRARY OF MATLAB PROCEDURES

`Newton_coef`

Finds coefficients of the Newton interpolating polynomials

```
function a = Newton_coef(x, y)
n = length(x);
%
% Calculate coefficients of Newton interpolating polynomial
%
a(1) = y(1);
for k = 1 : n-1
     d(k, 1) = (y(k+1) - y(k)) / (x(k+1) - x(k));
               % 1st divided diff
end
for j = 2 : n-1
     for k = 1 : n-j
          d(k, j) = (d(k+1, j-1) - d(k,j-1)) / (x(k+j) - x(k));
               % 2nd divided diff
     end
end
d
for j = 2 : n
     a(j) = d(1, j-1);
end
```

Newton Interpolation – MATLAB Script (2)

LIBRARY OF MATLAB PROCEDURES

Newton_Eval

Finds the Newton interpolating polynomial at $x = t$

```
function p = Newton_Eval(t, x, a)
n = length(x);
%
for i = 1 : length(t)
      ddd(1) = 1;            % Compute 1st term
      c(1) = a(1);
      for j = 2 : n
            ddd(j) = (t(i) - x(j-1)).*ddd(j-1);
                        % Compute jth term
            c(j) = a(j) .* ddd(j);
      end
      p(i) = sum(c);
end
```

Alternative Notation (1)

The 0th DD of f with respect to x_i:

Notation: $f[x_i]$ and Def.: $f[x_i] = f(x_i)$

Other DD are defined recursively:

The 1st DD of f with resp. to x_i and x_{i+1}:

Notation: $f[x_i, x_{i+1}]$ and

$$\text{Def.:}\quad f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

The 2nd DD of f with resp. to x_i and x_{i+1}:

Notation: $f[x_i, x_{i+1}, x_{i+2}]$ and

$$\text{Def.:}\quad f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i};$$ and so on.

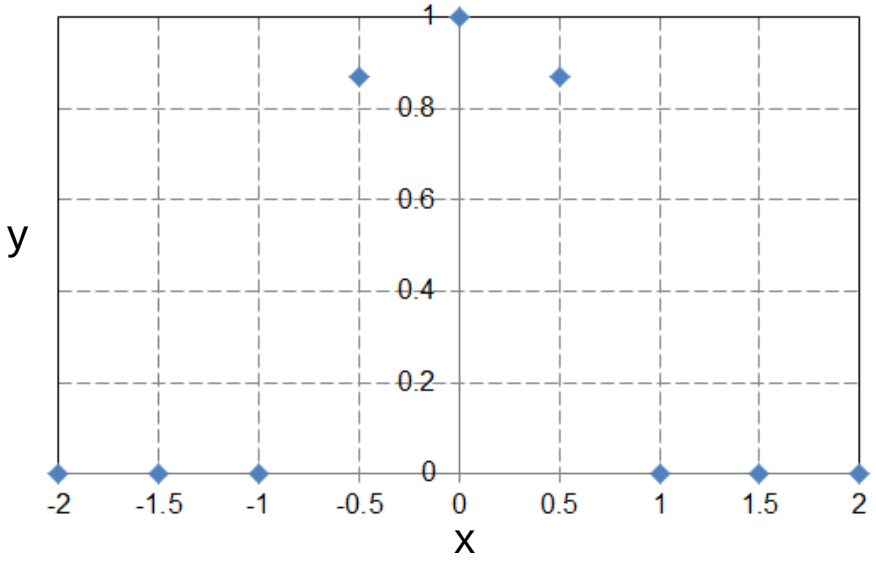
Alternative Notation (2)

DD's, when put in the DD Table, appear in the following positions:

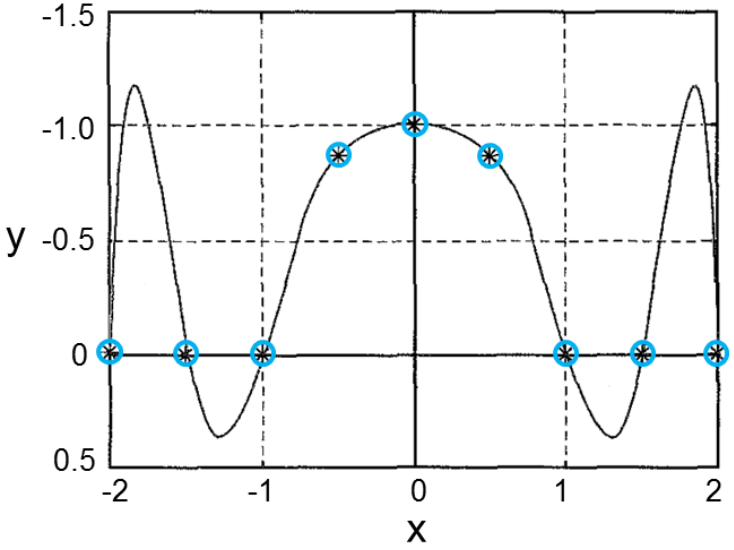
x	f(x)	1st DD	2nd DD	.....
x_0	$f[x_0]$			
		$f[x_0, x_1]$		
x_1	$f[x_1]$		$f[x_0, x_1, x_2]$ -> etc.	
		$f[x_1, x_2]$		
x_2	$f[x_2]$			
| etc.				

Difficulties with Polynomial Interpolation (1)

Bulge-Flat Data

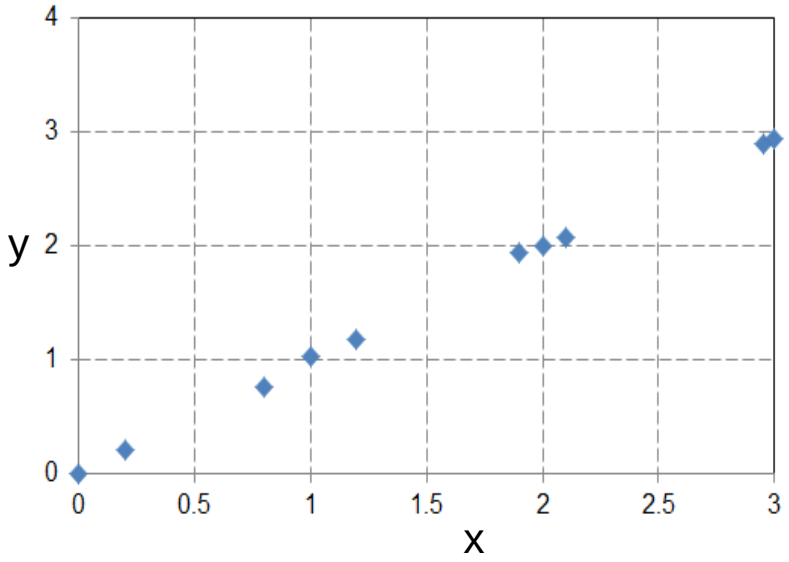


Newton Interpolation

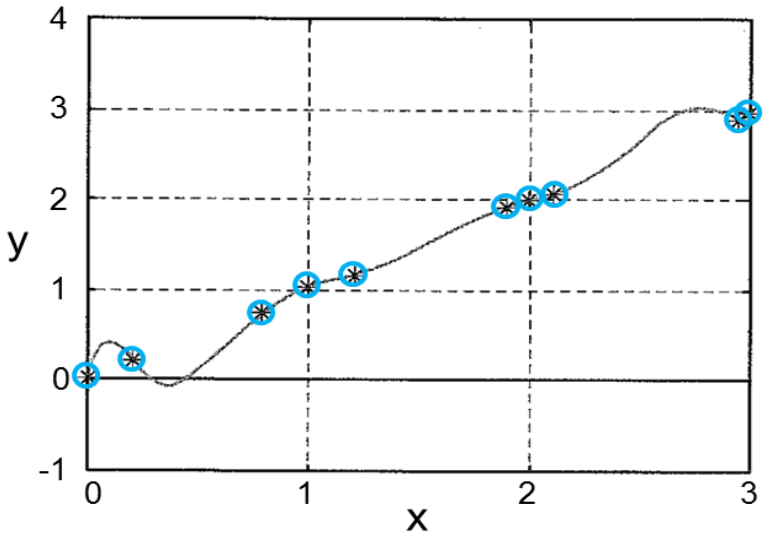


Difficulties with Polynomial Interpolation (2)

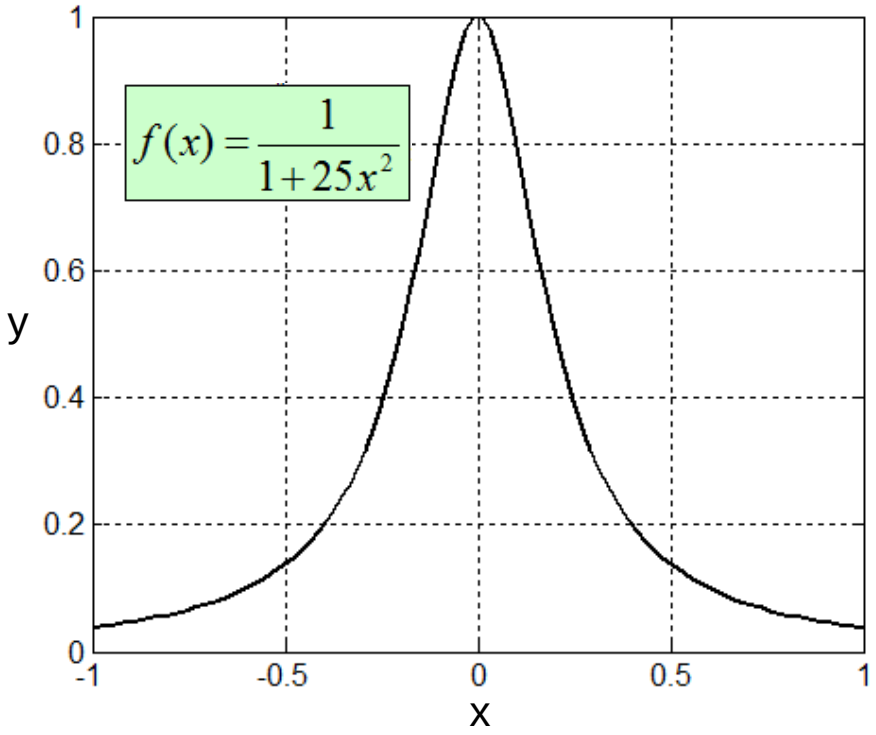
"Noisy" Straight Line



Newton Interpolation



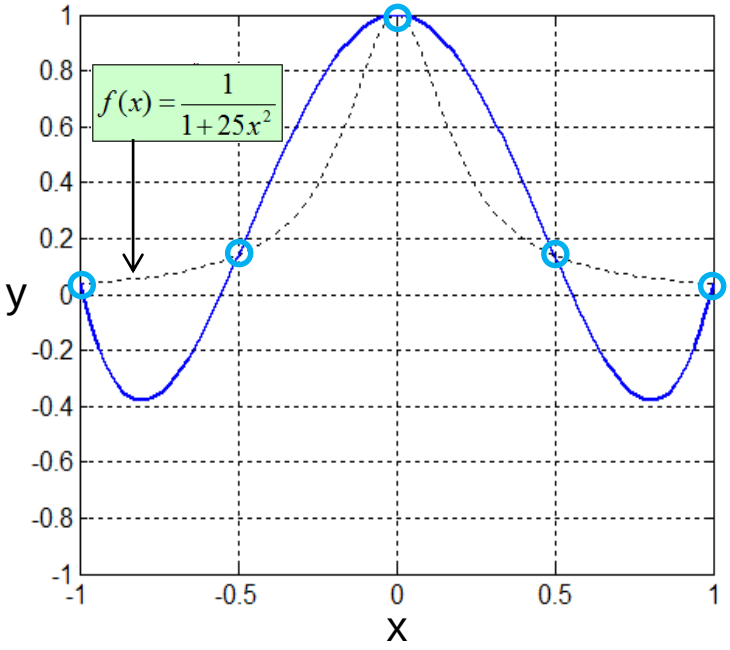
The Runge's Function



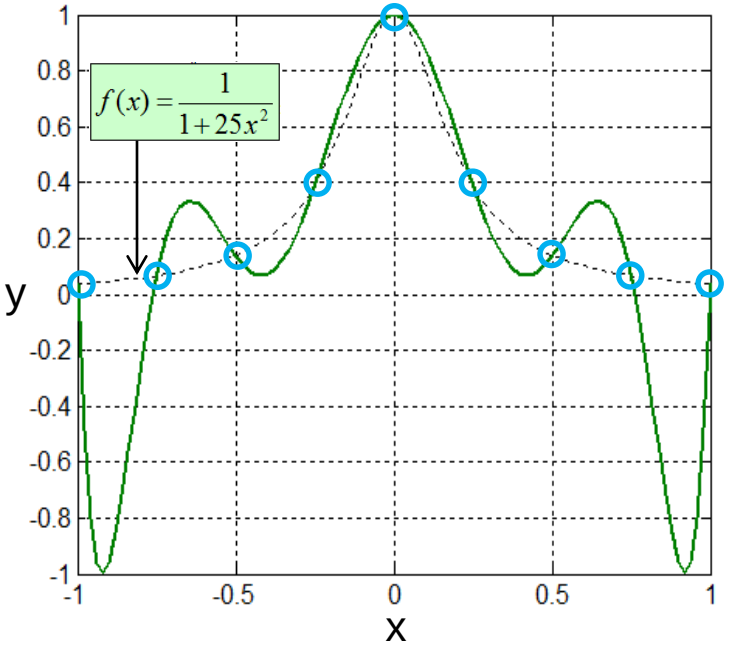
Polynomial Interpolation of the Runge's Function (1)

Newton Interpolation

5 Equally Spaced Points

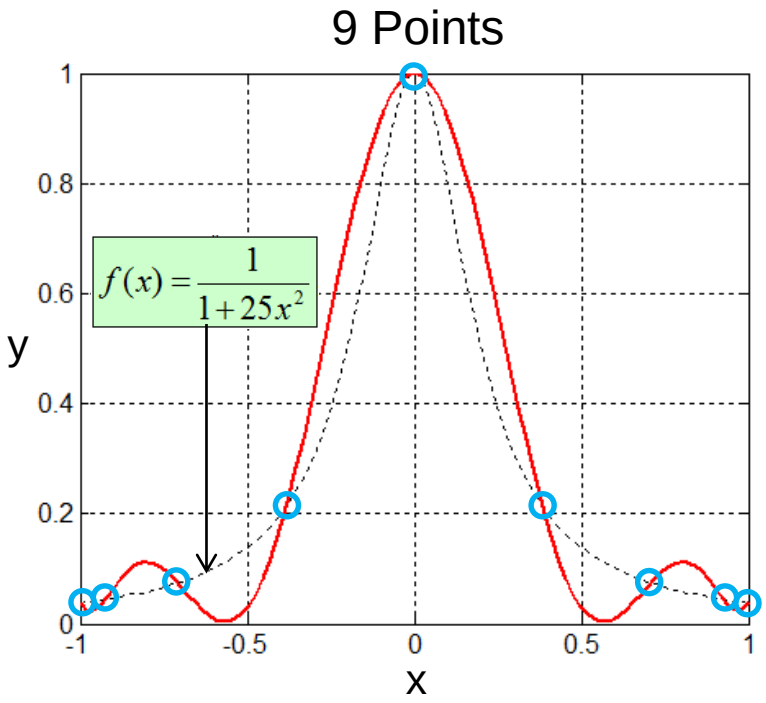


9 Equally Spaced Points



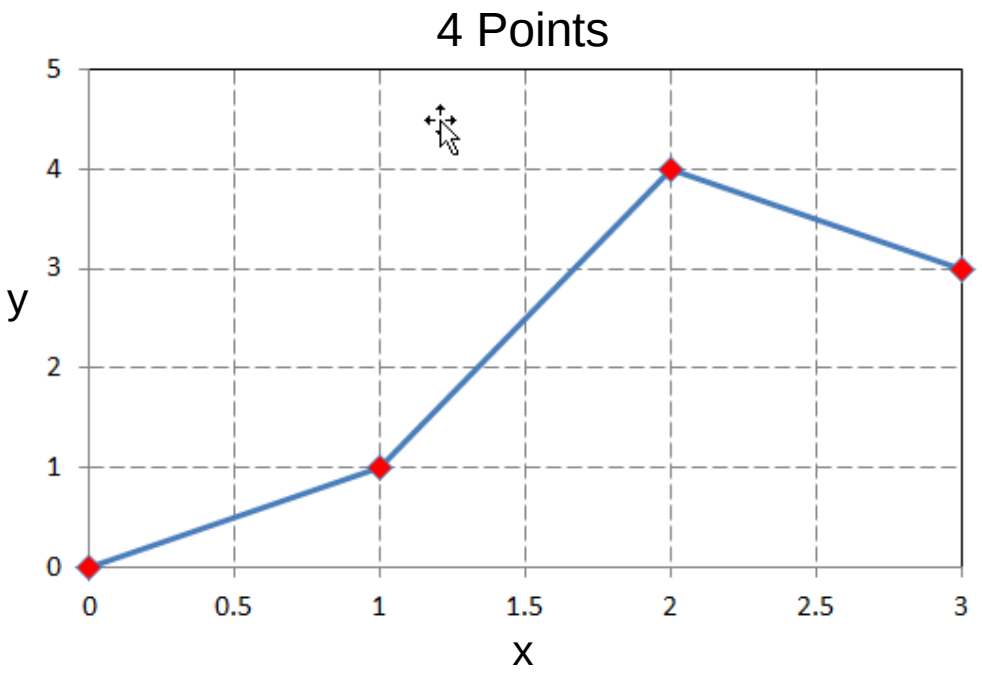
Polynomial Interpolation of the Runge's Function (2)

Newton Interpolation with Chebyshev Nodes



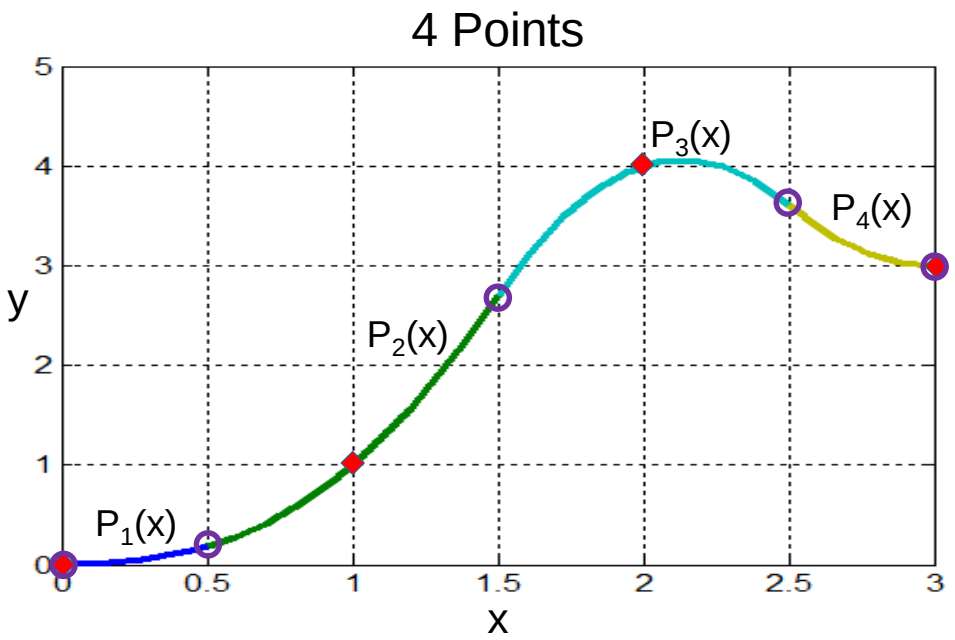
Spline Interpolation (1)

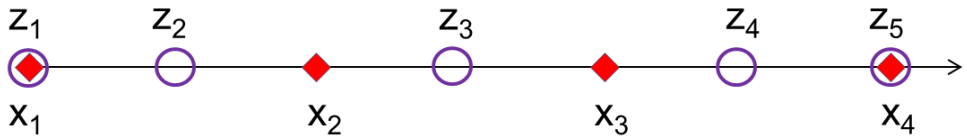
Piecewise Linear Interpolation



Spline Interpolation (2)

Piecewise Quadratic Interpolation





Cubic Spline Interpolation

Practical Problems from Computational Viewpoint

(1) Only some ***simplest problems*** dealing with cubic spline interpolation could be *fully solved manually*

(2) Other ***simple problems*** require a computer *for a central part of the solution*: *corresponding linear system can be derived manually*, but a computer (e.g., a procedure of Gaussian elimination) is needed ***to solve it*** for the coefficients.

(3) For other problems (with more than 3-4 points – i.e., actually, *for all **applied** problems*) a computer is needed to implement the algorithm at all stages.

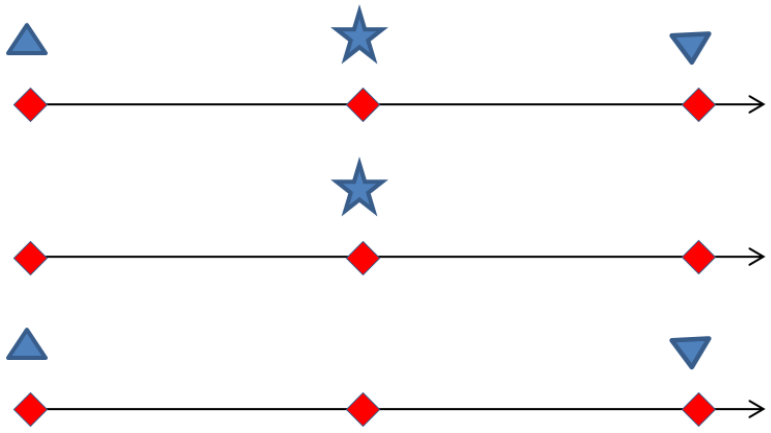
Problem of Type (1)

Section 3.5, Example 1

Construct a natural cubic spline that passes through the points (1, 2), (2, 3), and (3, 5).

The problem is about 3 points, 2 sub-intervals – thus, about 2 equations with 4 coefficients each, so – ***8 unknown total***.

8 conditions for 8 unknowns:



What to Set	No of conditions
Splines agree with data	[1 + 2 + 1 =] 4
1st & 2nd derivatives at the mid-point	2
2nd derivatives at the end points	2
TOTAL:	**8**

Cubic Spline Interpolation – MATLAB Script

LIBRARY OF MATLAB PROCEDURES

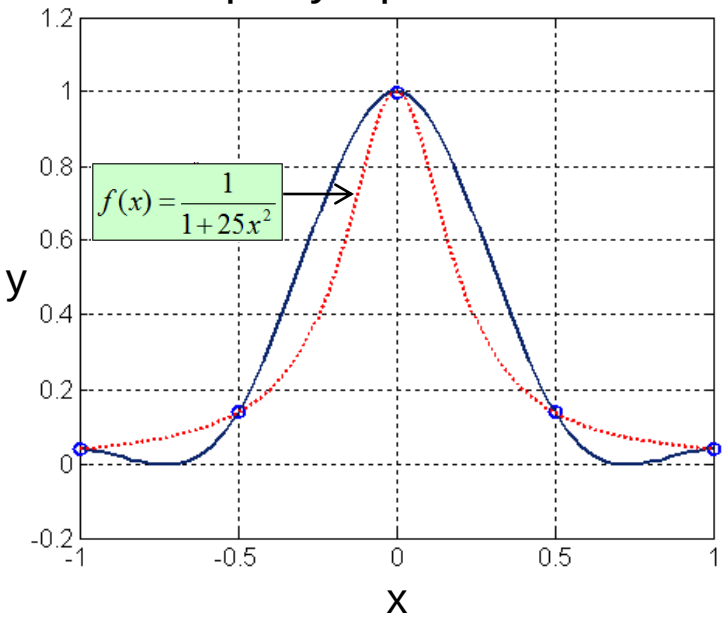
Spline_test

Generates and plots cubic spline interpolation for a test function – the Runge function with $a = 25$

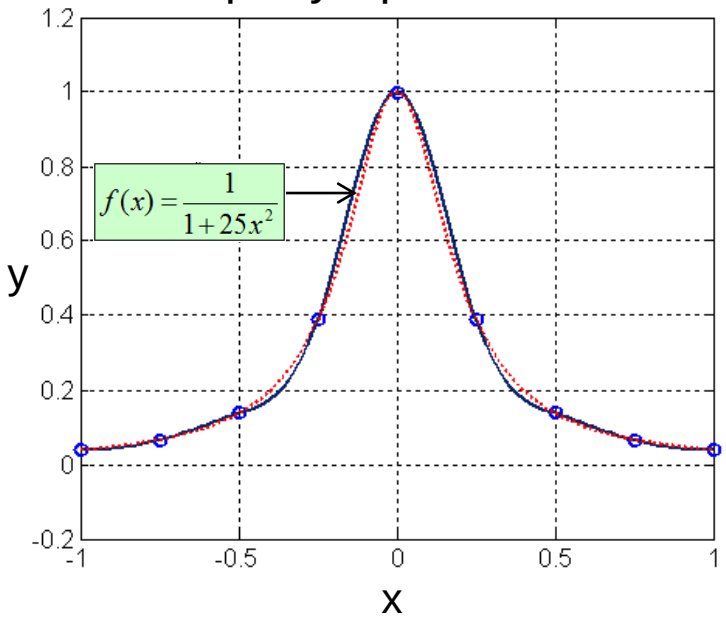
```
% Testing cubic spline interpolation by MATLAB's "spline" function
%
% Interpolation of the Runge Function
%
xR = -1 : 0.01 : 1;
yR = 1./(1 + 25*xR.*xR);
%
% Five data points
%
x = [-1 -0.5 0.0 0.5 1.0];
y = [ 0.0385 0.1379 1.0 0.1379 0.0385 ];
%
cs = spline(x, [0 y 0]);
%
xx = linspace(-1, 1, 101);
plot(x, y, 'o', xx, ppval(cs, xx), '-', xR, yR, 'k-');
```

Cubic Spline Interpolation of the Runge's Function

5 Equally Spaced Points



9 Equally Spaced Points

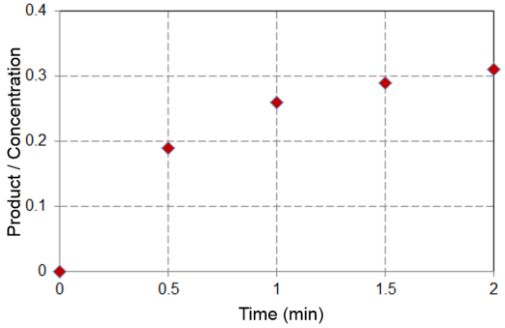


Polynomial Interpolation – Summary

Main Idea

Find a function which

- *matches exactly the data points* (which represent a process/phenomenon and are obtained either experimentally, or computationally) and
- *provides good estimates of the intermediate points* not included in the original data set.



Uniqueness / Equivalence of Interpolations

The polynomials generated by Lagrange & Newton approaches may look different, but *they are in fact identical* and represent ***the same function which is unique for the set of n distinct point***; the order of this polynomial is at most n-1.

Dealing with Additional Points

- ✓ In ***Lagrange Interpolation***, when adding more points to the data set, one has to rework the whole problem: *each term in the Lagrange polynomial contains information about all points*.
- ✓ There is no need to rework the problem with ***Newton Interpolation*** – *with new points, other terms in the polynomial are specified* and those already determined are used.

Polynomial Interpolation – Summary (*cont'd*)

Key Concept

- **Increasing the number of data points** ***does not improve interpolation***.
  - Moreover, there are *functions for which interpolation is always poorly performed*; unequally spaced data points (e.g., Chebyshev nodes) could be of help.

What is So Special About Cubic Splines?

- ***CSI***, the product of XX century, appears to be, overall, ***most efficient, flexible and accurate***.

Computer Implementations

- *All interpolation algorithms are convenient for implementations in computer codes and practical use* in different computational environments (MATLAB, etc.)

Why Numerical Differentiation?

Differentiation gives a measure of the rate at which a quantity changes.

Rates of change appear in many disciplines, practical applications, science, engineering, etc.

A function to be differentiated can be given as:

a) *An analytical expression*

b) *A set of discrete points* (obtained from experimentation or prior computation)

- For a), the derivative can be determined *analytically*
- When *analytical differentiation difficult/not possible*, ***Num. Diff.*** has to be used
- When the function is specified as *a set of discrete points*, the derivative also can be found by ***Num. Diff.***

Objective of Numerical Differentiation

- ***find estimates for the derivatives***, or *a slope of a function*, ***by using the function values at a set of discrete points***