Neville's Algorithm

x_0	$y_0 = P_0(x)$			
		$P_{01}(x)$		
x_1	$y_1 = P_1(x)$		$P_{012}(x)$	
		$P_{12}(x)$		$P_{0123}(x)$
x_2	$y_2 = P_2(x)$		$P_{123}(x)$	
		$P_{23}(x)$		$P_{1234}(x)$
x_3	$y_3 = P_3(x)$		$P_{234}(x)$	
		$P_{34}(x)$		
x_4	$y_4 = P_4(x)$			

$$P_{123}(x) = \frac{(x - x_3)P_{12}(x) + (x_1 - x)P_{23}(x)}{x_1 - x_3}$$

*Call the **column** in our table **m** (from 0 to n) and the **location** in the column **i** (from 0 to n-m)*

➡ ***P_{123}: m = 2, i = 1***

$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}$$

Neville's Algorithm

x_0	$y_0 = P_0(x)$			
		$P_{01}(x)$		
x_1	$y_1 = P_1(x)$		$P_{012}(x)$	
		$P_{12}(x)$		$P_{0123}(x)$
x_2	$y_2 = P_2(x)$		$P_{123}(x)$	
		$P_{23}(x)$		$P_{1234}(x)$
x_3	$y_3 = P_3(x)$		$P_{234}(x)$	
		$P_{34}(x)$		
x_4	$y_4 = P_4(x)$			

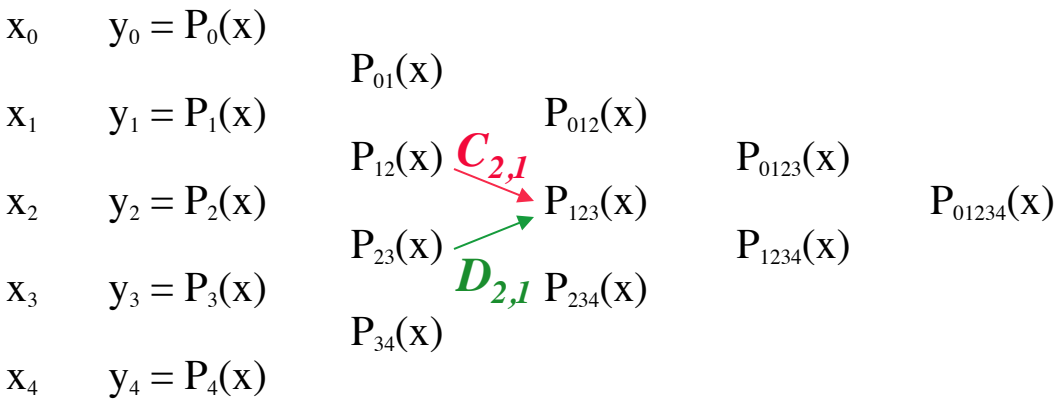
$$P_{1234}(x) = \frac{(x - x_4)P_{123}(x) + (x_1 - x)P_{234}(x)}{x_1 - x_4}$$

*Call the **column** in our table **m** (from 0 to n) and the **location** in the column **i** (from 0 to n-m)*

➡ ***P_{1234}: m = 3, i = 1***

$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}$$

Neville's Algorithm



$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}$$

Keep track of the differences between parents and daughters

$$C_{m,i} = P_{i\ldots(i+m)} - P_{i\ldots(i+m-1)} \qquad D_{m,i} = P_{i\ldots(i+m)} - P_{(i+1)\ldots(i+m)}$$

*At each level m, the **C**'s and **D**'s are the **corrections** that make the interpolation **one order higher**.*

Neville's Algorithm

$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}$$

$$C_{m,i} = P_{i\ldots(i+m)} - P_{i\ldots.(i+m-1)} \qquad D_{m,i} = P_{i\ldots(i+m)} - P_{(i+1)\ldots.(i+m)}$$

Using the above equations we can derive a ***recursive*** *form for the* ***C's*** *and* ***D's***

$$C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \qquad D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}$$

The C's and D's do not depend on the actual value of P(x)

Neville's Algorithm

x_0 $C_{0,0}$, $D_{0,0}=y_0$

x_1 $C_{0,1}$, $D_{0,1}=y_1$

$x \longrightarrow$

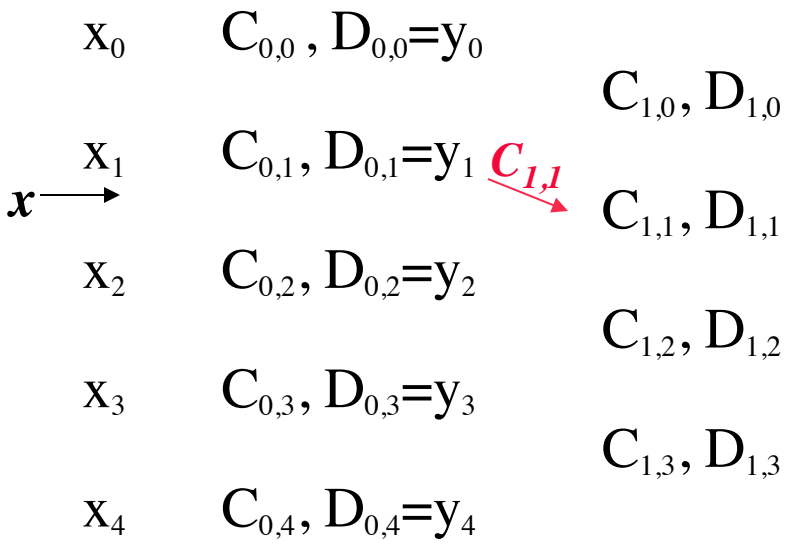
x_2 $C_{0,2}$, $D_{0,2}=y_2$

x_3 $C_{0,3}$, $D_{0,3}=y_3$

x_4 $C_{0,4}$, $D_{0,4}=y_4$

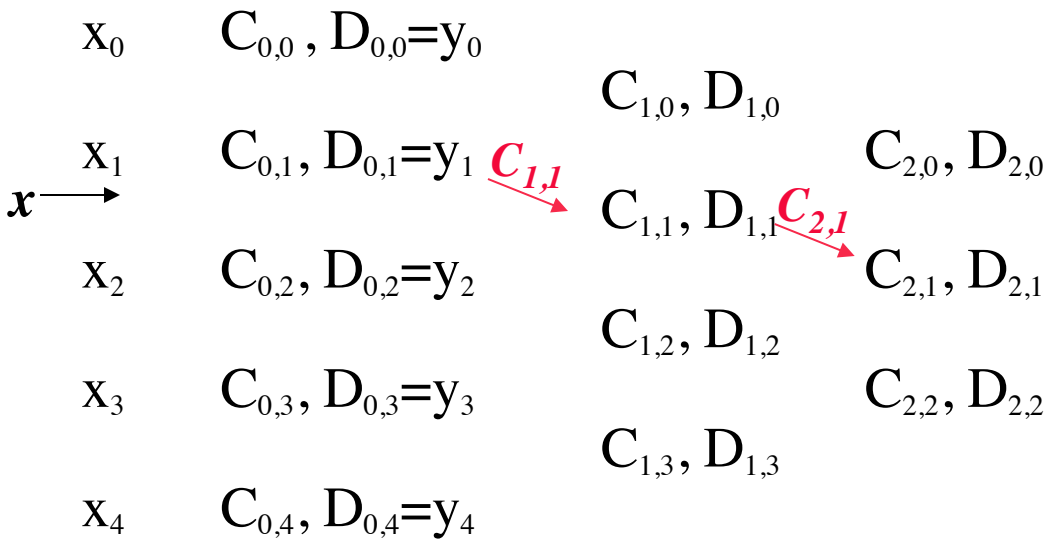
Solution: $y = y_1$

Neville's Algorithm



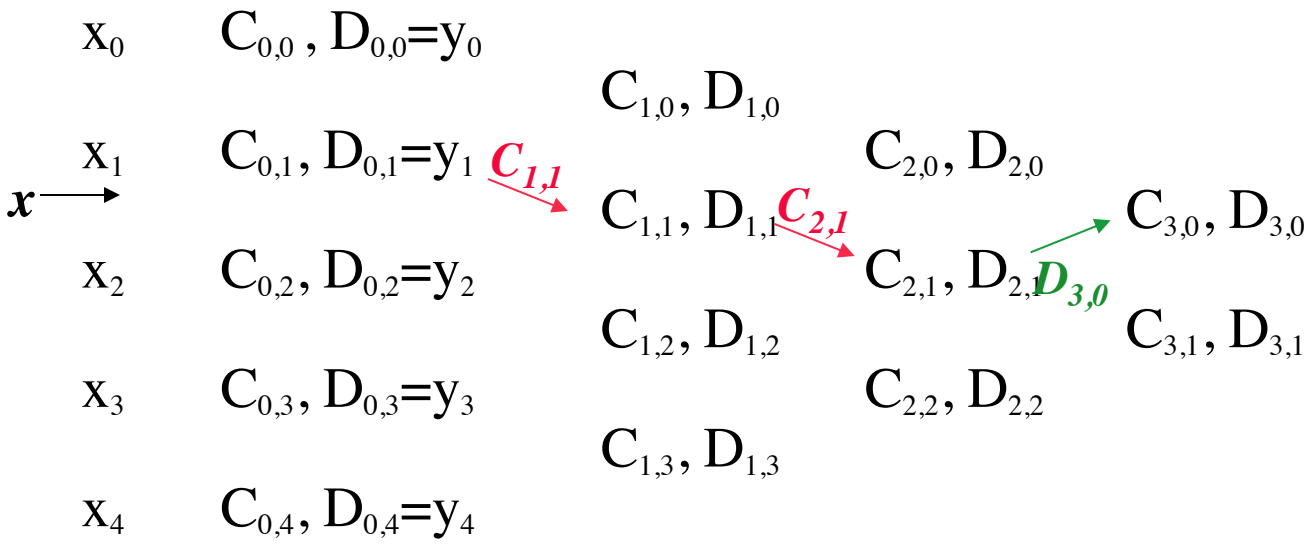
Solution: $y = y_1 + C_{1,1}$

Neville's Algorithm



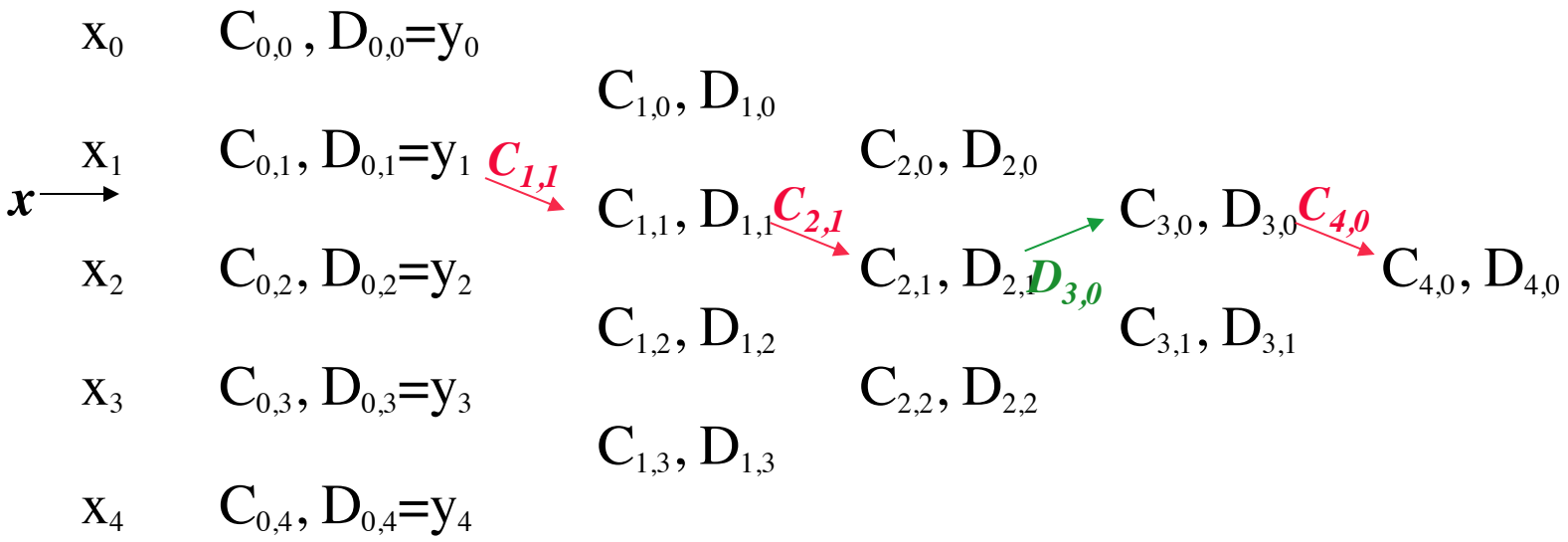
Solution: $y = y_1 + C_{1,1} + C_{2,1}$

Neville's Algorithm



Solution: $y = y_1 + C_{1,1} + C_{2,1} + D_{3,0}$

Neville's Algorithm



Solution: $y = y_1 + C_{1,1} + C_{2,1} + D_{3,0} + C_{4,0}$

The final answer $P_{0,\ldots n}(x)$ *is equal to the sum of any* y_i *plus a set of* ***C's*** *and/or* ***D's*** *that* ***form a path*** *through the tree to the top of the pyramid.*

The ***C's*** *and/or* ***D's*** *that are used to go to the next higher order can be used to* ***track the error*** *that is made by truncating the series.*

Pseudo Code for Neville's Algorithm

1. *Find the x_i value that is closest to the value x.*
   *Do this by calculating the **absolute distance** between the point x and all x_i*

   ***Initialize** the smallest distance to the distance between x and x_0*

   *Loop from **I = 1 to n** (the number of x_i values - 1)*

      *calculate the **absolute distance** between x and x_i*

      Compare the distance with the smallest distance found so far.

      If distance is smaller than the smallest distance found so far then
         This is your new smallest distance
         Save index I as the index for the point with the smallest distance
      end if

   End do loop

Pseudo Code for Neville's Algorithm

2. ***Initialize*** *the value for y as the y_i-value of the nearest point found in step 1.*

3. ***Initialize*** *the values for the C's and D's (C and D are vectors of size n+1). They are initialized to be the values y_i (also a vector of size n+1)*

Loop from ***I = 0 to n***

$C_i = y_i$
$D_i = y_i$

End Looping over the n+1 values

Alternatively: Use whole array initialization

$C = Y$
$D = Y$

Pseudo Code for Neville's Algorithm

4. Calculate the C's and D's for each column

Loop from ***m = 1 to n over the columns in the table***

Loop over the n-m+1 rows in each column (i = 0, n-m)

Calculate the new C's and D's

End Looping over the rows

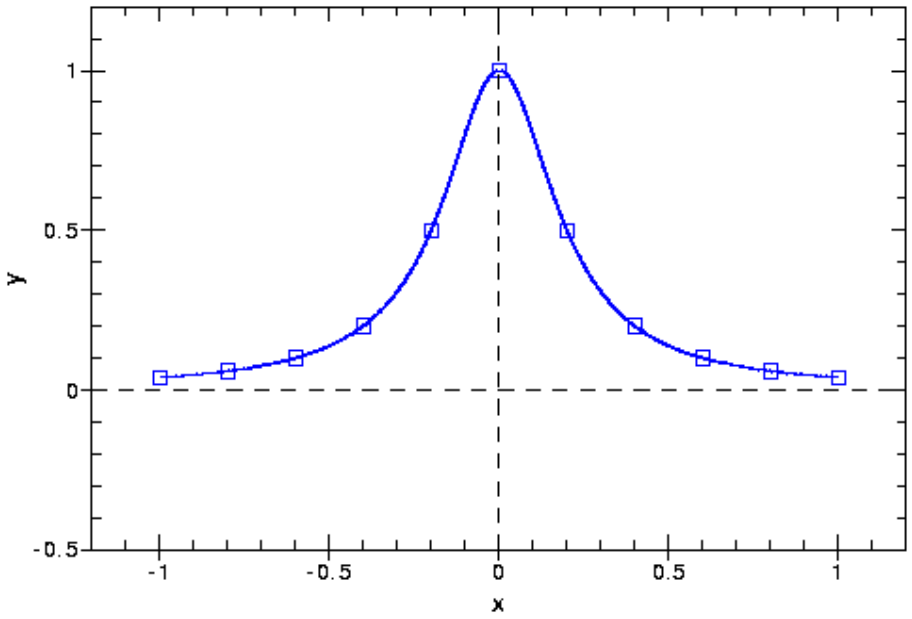
After the values for the C's and D's are calculated, we need to decide which correction, C or D, we want to add to our accumulating value of y, i.e., which path to take through the tree-forking up or down. We want to do this such that we take the most 'straight' line route through the table.

Add the correction C or D to the value of y

End Looping over all columns

Difficult Data

The Runge Function: $f(x) = \dfrac{1}{1+25x^2}$

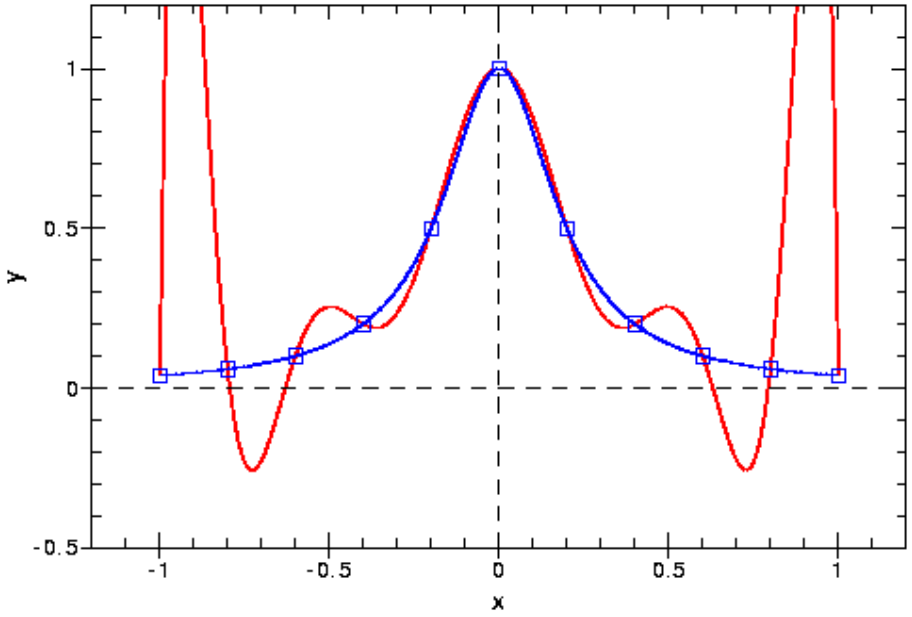


11 Points

The Runge function is a smooth function.

Difficult Data

The Runge Function: $f(x) = \dfrac{1}{1+25x^2}$

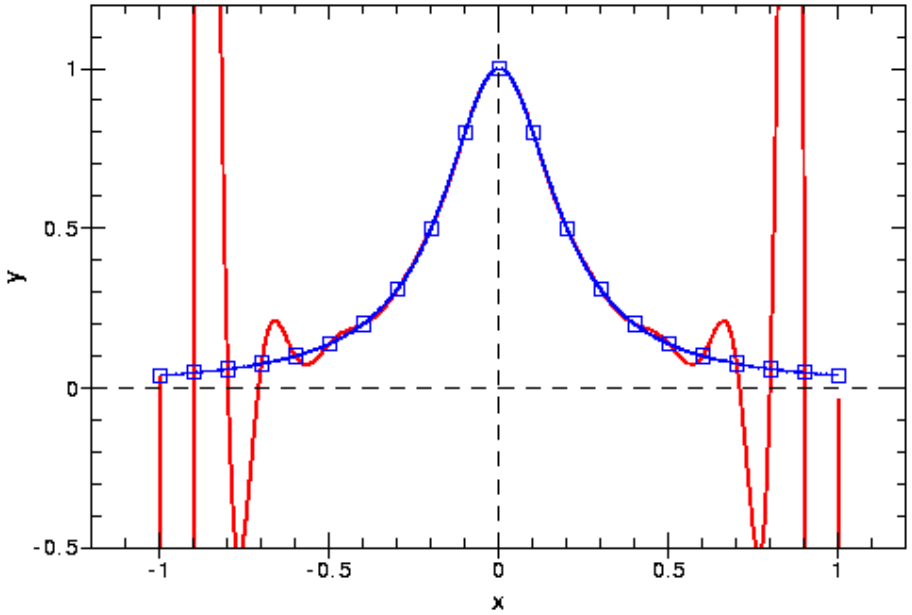


11 Points

The interpolating polynominal oscillates toward the end of the interpolation interval.

Difficult Data

The Runge Function: $f(x) = \dfrac{1}{1+25x^2}$



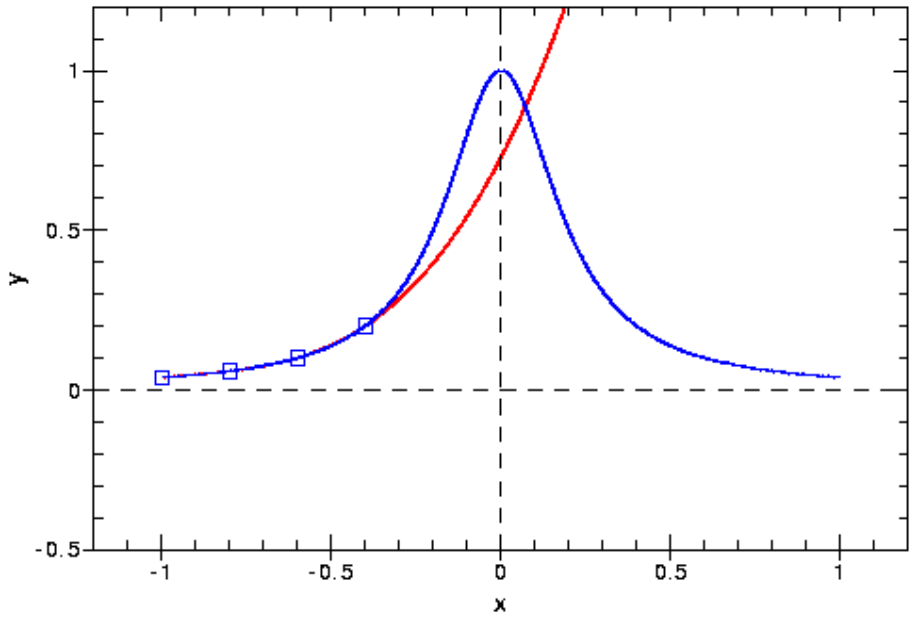
21 Points

With increasing number of points the problem is not resolved.

Reason: *The magnitude of* ***higher order derivatives*** *of the Runge function get even larger.*

Difficult Data

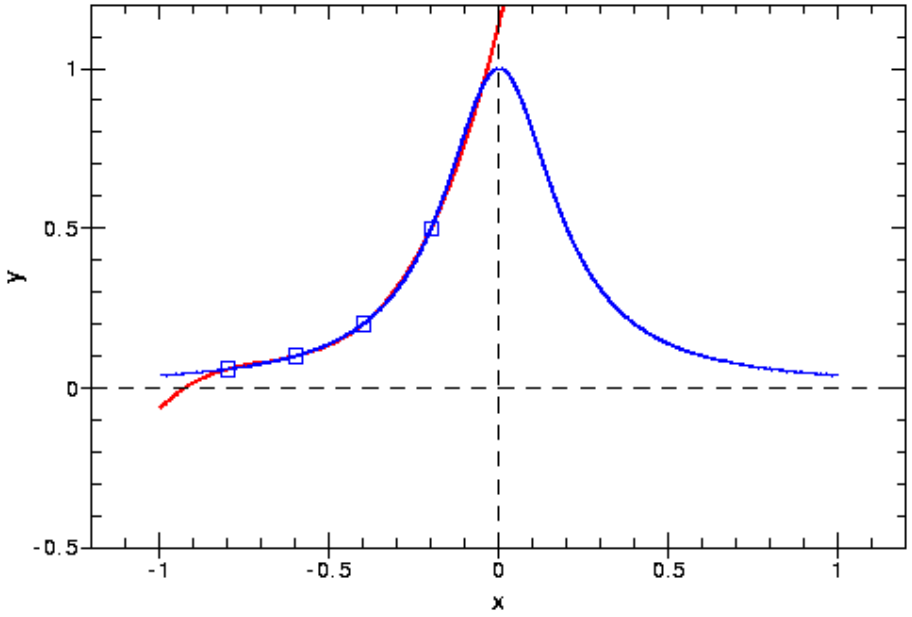
The Runge Function: $f(x) = \dfrac{1}{1+25x^2}$



Solution: ***Use lower order polynominals (less support points)***

Difficult Data

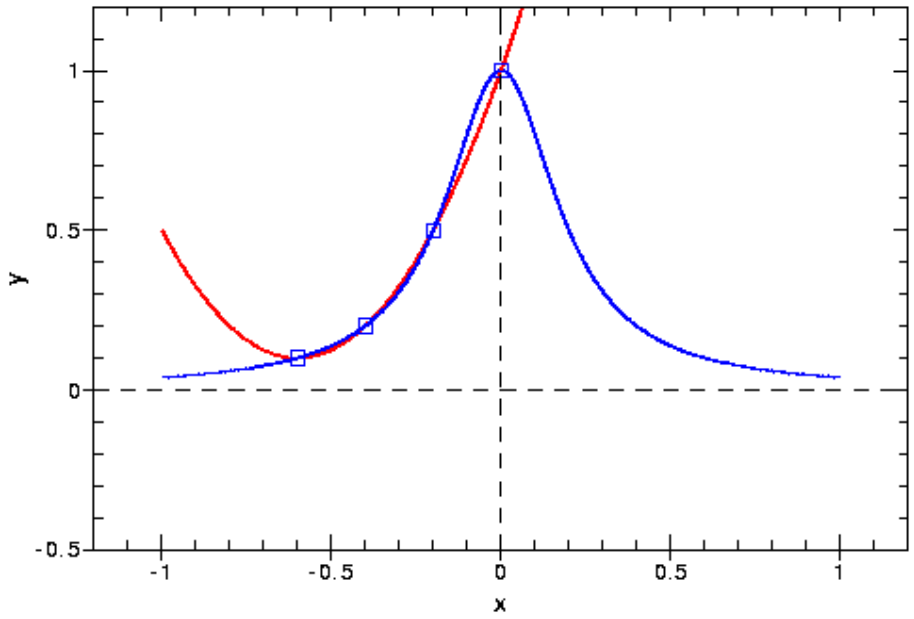
The Runge Function: $f(x) = \dfrac{1}{1+25x^2}$



Solution: ***Use lower order polynominals (less support points)***

Difficult Data

The Runge Function: $f(x) = \dfrac{1}{1+25x^2}$



Solution: ***Use lower order polynominals (less support points)***

Problem: ***Derivatives are NOT continuous at the merging points from one polynominal to the next!***

An Alternative Approach

Construct an approximation for f(x), namely g(x), by imposing the following conditions:

1. g′(x) is a smoothly varying function between two support points and is continuous at the support points

2. g′′(x) is continuous at the support points

This results a cubic (third order) polynominal in each interval while enforcing the above conditions.

---> Cubic Spline

Cubic Splines

Verify that number of imposed conditions equals the number of available coefficients:

There are n+1 support points (knots) → n subintervals.

On each subinterval we shell have a different cubic polynominal. Since each cubic polynominal has four coefficients

→ total of 4n coefficients available.

Within each interval cubic polynominal must go through two points
→ 2n conditions.

The 1st and 2nd derivative must be continuous at n-1 interior points
→ 2(n-1) conditions

The missing two conditions need to be provided as boundary conditions for the 1st or 2nd derivatives at the end points.

Types of Cubic Splines

Examples of Boundary conditions:

1. ***"Natural" Spline:*** $g_0''(x_0) = 0 \qquad g_n''(x_n) = 0$

   No curvature at the endpoints
   → equivalent to assuming that the end cubics approach linearity at their extremities.

2. ***"Clamped" Spline:*** $g_0'(x_0) = f'(x_0) \qquad g_n'(x_n) = f'(x_n)$

   Specify the 1st derivative of the interpolating function

3. ***Assume that*** $g_0''(x_0) = g_1''(x_1) \qquad g_n''(x_n) = g_{n-1}''(x_{n-1})$

   Equivalent to assuming that the end cubics approach parabolas at their extremeties.