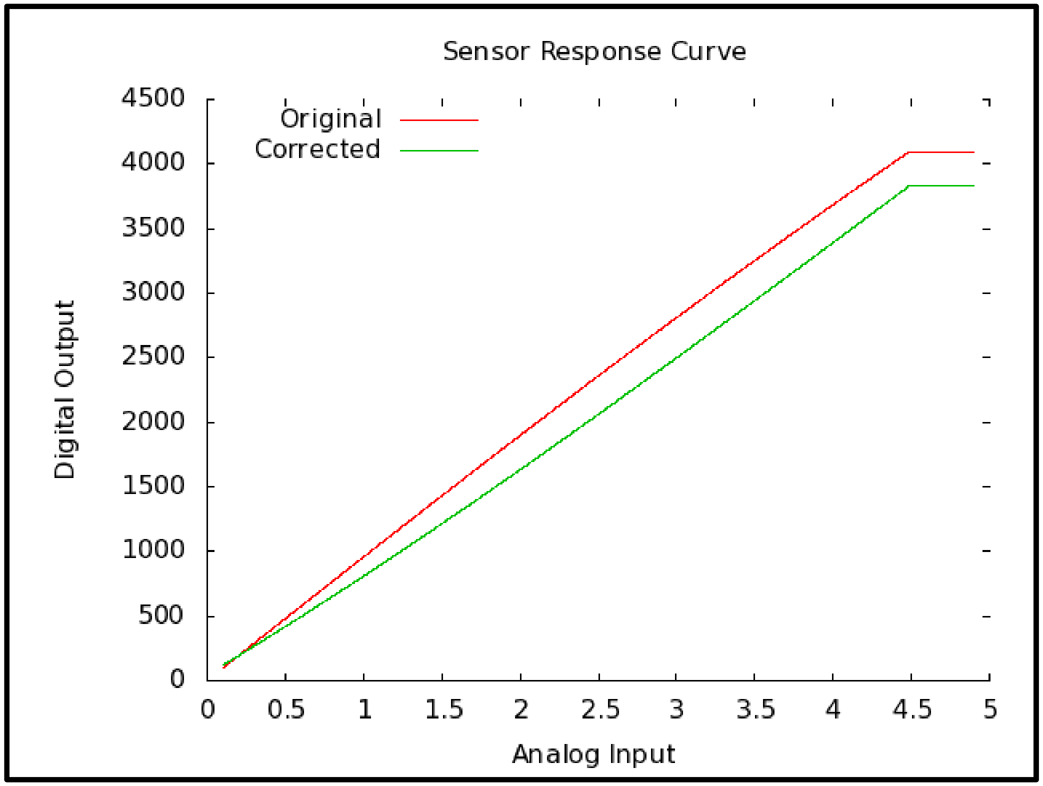
Applied Programming

gnuplot



gnuplot

- A command-driven, interactive, data plotting program.
  - Use interactively or via command file
    - gnuplot
    - gnuplot <gnuplotCmds.txt>
  - **Use as a service** (pipes)
    - popen(gnuplot)

- Only very high level summary given here
  - **In the order you need to implement**

help

- Produces pages and pages of detailed help
- Start gnuplot in interactive mode
  - E.g: gnuplot
  - **help set**

- Current state
  - show all   - shows the current settings

- Examples
  - http://www.gnuplot.info/demo/simple.html

General Commands

- Any number of commands may appear on a line separated by semicolons (;)
  - Or use “\n” in C at the end of each command
    - Use this one!
  - `load` or `call` must be the final command

- Strings are indicated with quotes.
  - Single or double
    - load "filename"
    - cd 'dir‘ - use this one in C

Popular Commands

- Set:
  - Used to setup or format the plot
  - Popular: **terminal, output, key, border, style, xrange, yrange, title, xlabel, ylabel**
- Plot:
  - Reads the data file and causes the plot to be rendered

1 - File Pipes

- You can “talk to” gnuplot using pipes:

```
if ((pipe = (FILE *) popen ("gnuplot -persist", "w")) == NULL)
 { fprintf (stderr,"Error: Unable to open pipe to gnuplot\n"); }
```

- Use the pipe as you would a file handle
  - fprintf (pipe, "set xlabel 'Time [sec]'\n");

- Note: popen is **not ANSI** (it is POSIX) compliant!
  Don’t compile this code with -ansi

Coding Hints

- Writing gnuplot code via pipes from C can be difficult to debug
  - A missing single quote or other minor string problem will generate a vague errors
  - A pipe IS a file handle

- E.g: to debug your code:
    - pipe = stdout;
    - fprintf (pipe, "set xlabel 'Time [sec]'\n");
  - Prints all your commands to stdout so you can see them!

2 - Set terminal

- Selects the basic graphics output type
- Popular terminal types:

Command	Description
dumb	ascii art for anything that prints text
gif	GIF images using libgd and TrueType fonts
jpeg	JPEG images using libgd and TrueType fonts
png	PNG images using libgd and TrueType fonts
X11	X11 Window System

- Typical command:

set terminal png enhanced font 'DejaVuSans.ttf' 12

Note: TrueType fonts allow for dynamic font resizing

Finding Fonts

- In your .bashrc file you need to add the following lines to enable fonts:

export GDFONTPATH=/usr/share/fonts/dejavu

export GNUPLOT_DEFAULT_DDFONT="DejaVuSans.ttf"

3 - Set Output

- Use to set the output file name
  - Must be done AFTER the set terminal command
    - filename must be enclosed in quotes
  - If the filename is omitted
    - The current output file will be closed
    - All new output will go to stdio
- Example: set output ‘test.png'

- LinuxNote: Linux machines
  - Output can be piped if the first character is '|'.

4 - Set Key

- Controls the plot key or legend
  - The key is placed in the upper right inside corner of the graph by default.
- Popular:
  - on|off
  - {no}box
  - `left`, `right`, `top`, `bottom`, `center`, `inside`, `outside`

- Example: set key box
  set key on

5- Set Border

- Controls the border around the plot

- Popular:
  - <integer>
    - Integer thickness of the line

- Example: set border 3

6 - Set Style

- Changes how data is displayed on the plot
- Popular:
  - data <plotting-style>
  - See what looks best

Plotting-Style	Description
lines	Connects adjacent points with straight line segments. – often the best choice
points	Displays a small symbol at each point.
linespoints	Does both `lines` and `points` (abbrev: lp)

- Sample: set style data lines

7 - Set Title

- Sets the plot title in the center top of plot

- Popular:
  - “<title-text>”

- Example: set title 'Sensor Response Curve'

8 - Set xlabel - ylabel

- Sets the x and y axes labels

- Popular:
  - "<label>“

- Example: set ylabel 'Digital Output'

Optional - Set xrange - yrange

- Controls horizontal and vertical range displayed
  - Useful to FORCE a series of plots to be identical

- Popular:
  - [<min>:<max>]
    - <min> and <max> terms are constants, or an asterisk "*" for autoscaling.

- Example: set xrange [0:25]
  set yrange [-1:*]

9 - Plot Command

- Reads data and generates the plot
  - Provides features to “parse” the data
- Popular:
  - ‘<datafile>’ using x:y lt c lw w t{itle} ‘title’ {,}
    - x:y - the colums x and y datafile in datafile
    - lt c - line color index, 1..n, for unique colors
    - lw w - The width of the line
    - , - Continue with another plot section
- Examples:

plot 'data' using 1:2 title 'plot1'

plot 'data' using 1:2 lt 1 lw 1.5 t 'plot1', 'data' using 1:4 lt 2 lw 2.5 t ‘plot2'

Note the comma to continue the command

Plot Feature

- Has the ability to read data AND perform simple calculations
  - Use “$x” to indicate the column data to use for calculation. (x is the column number)

- Example:

"plot ‘data' using 1:($3*57.2958) lt 1 lw 1.5 t "

Convert the radian data in column 3 into a “degree” form and plot it. (180/PI = 57.2958)

Column data

- You can combine multiple column text data files into ONE giant column text file by rows
  - Using the bash command **paste**

- E.g.

  paste file1.txt file2.txt > alldata.txt

Applied Programming

Numerical Interpolation

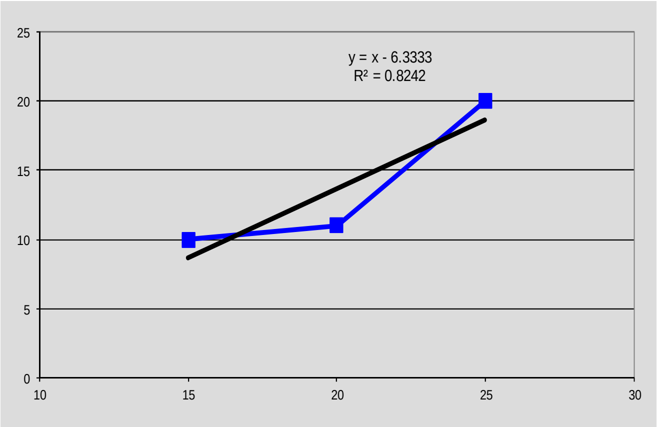
More details in: U. Ascher and C. Grief, "A First Course in Numerical Methods", chapters 10.1 – 10.5, 11.1, 11.3

Motivation: 21st Century Tables ?

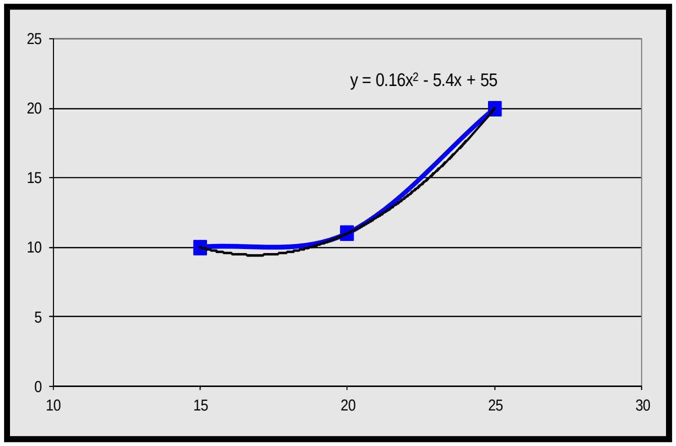
- In the past books with long “engineering tables” were essential for engineering practice.

- There are ***two main tasks*** usually performed with table entries:
  - ❑ Finding a value between two entries in the table (interpolation)
  - ❑ Finding a value outside the range of the table (**extrapolation**)

Interpolation vs Fitting



- Fitting
  - Good for "noisy" data
  - Finds an equation for a set of points
  - Does not normally "touch" any of the points



- Interpolation
  - Can't be used on "noisy" data
  - Finds an equation that "touches" points
  - Uses other points to "adjust" the equation

The Interpolation Problem

- Mathematical Description

Given a set of data points (x_i, y_i) for $i = 0, 1, \ldots, n$ where x_i is the independent variable. Find a function f such that $f(x_i) = y_i, i = 0, 1, \ldots, n$

- We say that a function $f(x)$ such that $f(x_i)=y_i$ ***interpolates*** the data points (x_i,y_i)
- The function $f(x)$ that interpolates the data is called the ***interpolant***.

Fundamental Assumption

The ***data is not corrupted by noise***, *i.e.*, it is "exact"

Interpolation

- The first step is to identify a "suitable function"

- Plot the data to "see" what the function "looks like" (*e.g.* we need to find a model for the data)

  - If possible, use ***prior*** knowledge to ***decide what type of interpolating function*** to use.

Interpolants

- For ease of computations, we will consider ***interpolants*** with ***linear parameters***

$$f(x) = c_o\phi_o(x) + c_1\phi_1(x) + \cdots + c_n\phi(x)$$

- The functions $\phi_i(x)$ are called ***basis functions***
- The constants c_i are the ***parameters.***

Important:

- The ***choice of basis is critical*** for the efficiency of the interpolation algorithm.

Interpolants

- Common basis functions (e.g., data models) used for interpolation are:
  - ❑Polynomials (General use)
  - ❑Complex exponentials (DSP)
  - ❑Radial Basis Functions (3D CAD)
  - ❑Splines
  - ❑…

Numerical Interpolation Algorithms

- Numerical Interpolation Algorithms usually involve ***two steps***:

  1. Construction of the interpolation function (e.g., find the parameters $\boldsymbol{c_k}$)

  2. Evaluation of the interpolation function at a desired point $\boldsymbol{x}$

General Interpolation Problem

- Finding the ***n+1 parameters*** of a function

$$f(x) = c_o\phi_o(x) + c_1\phi_1(x) + \cdots + c_n\phi(x)$$

that ***interpolates n+1 data points*** $\{(x_k, y_k)\}_{k=0}^{n}$ is equivalent to solving the linear *system*

$$\begin{bmatrix} \phi_o(x_o) & \phi_1(x_o) & \cdots & \phi_n(x_o) \\ \phi_o(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \phi_o(x_n) & \phi_1(x_n) & \cdots & \phi_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}$$

=> **Interpolation can be reduced to Linear Algebra**

Interpolation with Polynomials

- The most common functions used for interpolation are the *polynomials*
- Polynomials interpolants include:
  - ❑ Constants (0^{th} order polynomial)
  - ❑ Lines (1^{st} order polynomial)
  - ❑ Parabolas (2^{nd} order polynomial)
  - ❑ Cubics (3^{rd} order polynomial)
  - ❑ …

- The ***choice of basis*** functions leads to ***different interpolation algorithms***

The Monomial Basis

- The ***standard form*** of a polynomial of n^{th} degree occurs when we choose the ***monomial basis***

$$\phi_k(x) = x^k, \quad k = 0, \ldots, n$$

- Under this basis, the interpolating function

$$f(x) = c_o\phi_o(x) + c_1\phi_1(x) + \cdots + c_n\phi(x)$$

becomes the "usual" n^{th} *degree polynomial:*

$$p_n(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots + c_n x^n$$

Notation: $p_n(x)$ denotes an n^{th} *degree polynomial*

Monomial Basis: Parameters

- Fact: There is a ***unique n^th^ degree polynomial***

$$p_n(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots + c_n x^n$$

that interpolates *n+1* distinct data points.

- Its coefficients can (in principle) be obtained by solving:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}$$

Complexity

$$\mathcal{O}(\tfrac{2}{3}n^3)$$

This is called a ***Vandermode matrix***. For large degree polynomials it is difficult to solve it accurately.

Efficient Numerical Interpolation

- The ***monomial basis*** is ***rarely used*** because:
  - Better results can be obtained using other bases.
  - It is **very sensitive** to round-off errors.

- The two most common alternatives are:
  - ❑ **Lagrange basis** (Lagrange Interpolation Alg.)
  - ❑ **Newton's basis** (Newton's Interpolation Alg.)

❑**Lagrange** Interpolation

Linear Lagrange Interpolation

- We have two different data points (x_0,y_0) and (x_1,y_1)

- Assuming a *linear relation* between x and y we can write

$$p_1(x) = \underbrace{\left(\frac{x - x_1}{x_0 - x_1}\right)}_{1@x_0,0@x_1} y_0 + \underbrace{\left(\frac{x - x_0}{x_1 - x_0}\right)}_{1@x_1,0@x_0} y_1$$

- This is the Lagrange form of a straight line (1st deg. Poly.) through points (x_0,y_0) and (x_1,y_1)
  - We can use it to interpolate any value $\boldsymbol{x_0 < x < x_1}$
  - Note: You can't evaluate AT the end points (but why would you?)

Quadratic Lagrange Interpolation

- Given three different data points: (x_0,y_0), (x_1,y_1) , (x_2,y_2)

- Assuming a *quadratic relation* between x and y:

$$p_2(x) = \left(\frac{x - x_1}{x_0 - x_1}\frac{x - x_2}{x_0 - x_2}\right) y_0 + \left(\frac{x - x_0}{x_1 - x_0}\frac{x - x_2}{x_1 - x_2}\right) y_1 + \left(\frac{x - x_0}{x_2 - x_0}\frac{x - x_1}{x_2 - x_1}\right) y_2$$

- This is the Lagrange form of a 2nd order polynomial (e.g. a parabola)

Lagrange Basis Polynomial

The ***Lagrange basis polynomial of n^{th} degree***

at point x_k is:

$$\phi_k(x) = L_{n,k}(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}$$

n – the number of product terms

k – the index of the point

$$L_{2,0} = \left(\frac{x - x_1}{x_0 - x_1} \frac{x - x_2}{x_0 - x_2} \right)$$

So:
n =2, k = 0

$$L_{2,1} = \left(\frac{x - x_0}{x_1 - x_0} \frac{x - x_2}{x_1 - x_2} \right)$$

n=2, k = 1

Lagrange Basis: Parameters

- n^{th} degree polynomials interpolating the data set $\{(x_k, y_k)\}_{k=0}^{n}$

can be written in terms of the Lagrange basis as:

$$\boxed{p_n(x) = \sum_{k=0}^{n} y_k \phi_k(x)}$$

given $$\boxed{\phi_k(x) = L_{n,k}(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}}$$

Lagrange Basis: Parameters

- The ***Lagrange basis polynomial of n^{th} degree*** associated with the interpolation point x_k is

$$\phi_k(x) = L_{n,k}(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}$$

$$p_n(x) = \sum_{k=0}^{n} y_k \phi_k(x)$$

- To find the **unique polynomial of degree (at most) *n*** that interpolates $n+1$ distinct points we would need to "solve"

$$\begin{bmatrix} \phi_o(x_o) & \phi_1(x_o) & \cdots & \phi_n(x_o) \\ \phi_o(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \phi_o(x_n) & \phi_1(x_n) & \cdots & \phi_n(x_n) \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}$$

But of course there is nothing to be solved for

$$L_{n,k}(x_i) = \begin{cases} 1 & , i = k \\ 0 & , i \neq k \end{cases}$$

Quadratic Lagrange Example

- Find ***y* at *x=23.1*** given

  $x_0=15, y_0=10$

  $x_1=20, y_1=11$

  $x_2=25, y_2=20$

Remember: n=2 quadratic

$$p_2(x) = \left(\frac{x-x_1}{x_0-x_1}\frac{x-x_2}{x_0-x_2}\right)y_0 + \left(\frac{x-x_0}{x_1-x_0}\frac{x-x_2}{x_1-x_2}\right)y_1 + \left(\frac{x-x_0}{x_2-x_0}\frac{x-x_1}{x_2-x_1}\right)y_2$$

$$\boxed{\begin{aligned}\phi_0(x) &= L_{2,0}(x)\\ \phi_0(x_0) &= 1\end{aligned}}$$

- Let $y=f(x)$. We want to find a function $f(x)$ that interpolates the data, i.e., such that

$$y_k = f(x_k)\ ,\ k=0,1,2 \qquad \text{(quadratic)}$$

Lagrange Interpolation

- We are given 3 points so we know that there is a unique 2nd order interpolating polynomial.

- Using the Lagrange interpolation formula

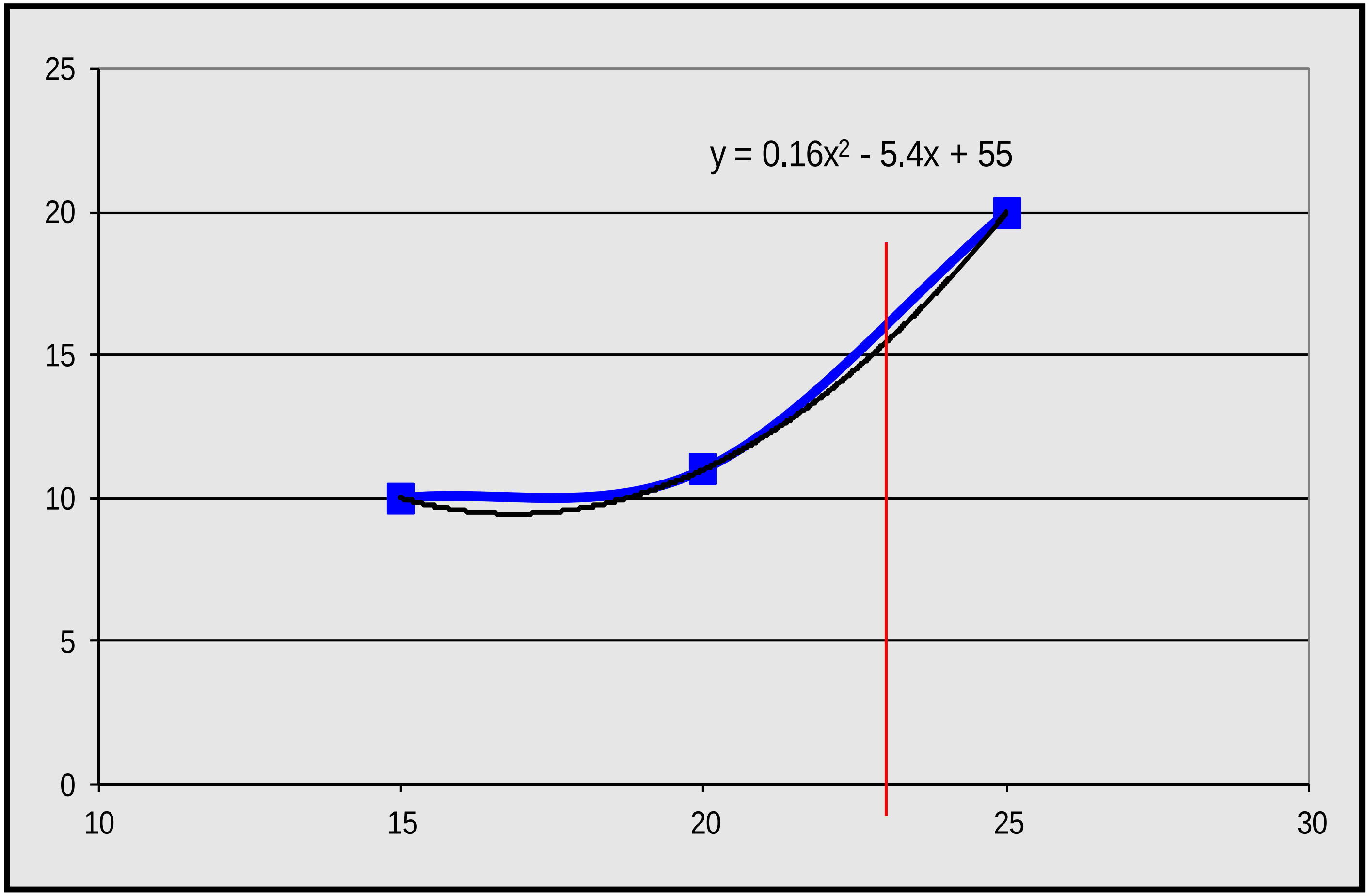
$$p_2(x) = L_{2,0}(x)y_0 + L_{2,1}(x)y_1 + L_{2,2}(x)y_2$$

$$p_2(x) = \left(\frac{x-20}{15-20}\frac{x-25}{15-25}\right)10 + \left(\frac{x-15}{20-15}\frac{x-25}{20-25}\right)11 + \left(\frac{x-15}{25-15}\frac{x-20}{25-20}\right)20$$

$$p_2(23.1) = 15.6376$$

- Result: $f(23.1) \approx p_2(23.1) = 15.6376$

Lagrange Interpolation: Graph



The **blue line** in the plot represents the unknown function *f(x) that generated the data*

Barycentric Algorithm

- An efficient algorithm for the construction of Lagrange interpolation n^th order polynomials.

$$\{(x_i, y_i)\}_{i=0}^{n}$$

Given data $\{(x_i, y_i)\}$ $i = 0,..,n$ compute $\boldsymbol{w_j}$ (***barycentric weights***)

$$\boldsymbol{w_j} = \frac{1}{\prod_{i \neq j}(x_j - x_i)} = \frac{1}{\rho_j}, j = 0, 1, \ldots, n$$

(This requires about n^2 FLOP)

Lagrange "Construction Algorithm"

Input: $\{(x_i)\}_{i=0}^{n}$

Output: $\boldsymbol{\rho} = (\rho_0; \rho_1; \ldots; \rho_n)$ (an $n+1$ vector)

Compute inverse weights

$$\rho_i = \prod_{\substack{i=0 \\ i \neq j}}^{n} (x_j - x_i)$$

Notes:

- Uses only ***data for the independent variable x_i*** (for efficiency we compute the reciprocal)

Evaluation of Interpolant

General Lagrange Polynomial Evaluation

- Given an evaluation point ***x*** (***not in the data set***)

$$p_n(x) = \frac{\sum_{j=0}^{n} y_j w_j \dfrac{1}{(x - x_j)}}{\sum_{j=0}^{n} w_j \dfrac{1}{(x - x_j)}}$$

$w_i = 1/p_i$

$$\rho_i = \prod_{\substack{i=0 \\ i \neq j}}^{n} (x_j - x_i)$$

Warning: if ***x*** is in the data set division by zero will occur

- Lets organize this in a efficient algorithm

Lagrange "Evaluation Algorithm"

Input: $\{\rho_i\}_{i=0}^n, \{(x_i, y_i)\}_{i=0}^n,$ and $x \neq x_i$

Output: $p_n(x)$

$$\rho_i = \prod_{\substack{i=0 \\ i \neq j}}^{n} (x_j - x_i)$$

1. Compute $\psi(x) = \prod_{i=0}^{n} (x - x_i)$
2. Compute $\theta(x) = \sum_{i=0}^{n} \frac{y_i}{(x - x_i)\rho_i}$
3. Find $p(x) = \theta(x)\psi(x)$

(All of this takes about ***5n* FLOP**)

Lagrange Interpolation - Summary

- **Limitations**:
  - **Not recursive -** if more data points become available, we must recompute everything

- **Advantages**:
  + Simple to derive (isolates contribution of each point)
  + Useful when abscissas **(*x*) are fixed** but function values **(*y*) change**

- **Newton's** Interpolation

Linear Newton Interpolation

- We have two data points (x_0,y_0) and (x_1,y_1)

- Assuming a linear relation between x and y we can write

$$\boxed{p_1(x) = a_0 + a_1(x - x_0)}$$

- This is the ***Newton polynomial form of a straight line*** through points (x_0,y_0) and (x_1,y_1), where

$$a_0 = y_0, \quad a_1 = \frac{y_1 - y_0}{x_1 - x_0}$$

- As before, can use it to **interpolate** any value $\boldsymbol{x_0 < x < x_1}$

Quadratic Newton Interpolation

- Given three different data points: (x_1,y_1), (x_2,y_2) , (x_3,y_3)

- Assuming a quadratic relation between x and y we can write

$$p_2(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$$

where:

$$a_0 = y_0, a_1 = \frac{y_1 - y_0}{x_1 - x_0}, a_2 = \frac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0}$$

Note the recursive nature of the coefficients !!

Newton basis polynomial

- The ***Newton basis polynomial of k^{th} degree*** associated with the interpolation point x_k is

$$\phi_k(x) = N_k(x) = \begin{cases} 1 & , k = 0 \\ \prod_{i=0}^{k-1}(x - x_i) & , k > 0 \end{cases}$$

$$p_n(x) = \sum_{k=0}^{n} a_k \, \phi_k(x)$$

Newton Basis: Parameters

- To find the unique polynomial of degree (at most) *n* that interpolates n+1 distinct points we would need to solve

$$\begin{bmatrix} N_o(x_o) & N_1(x_o) & \cdots & N_n(x_o) \\ N_o(x_1) & N_1(x_1) & \cdots & N_n(x_1) \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ N_o(x_n) & N_1(x_n) & \cdots & N_n(x_n) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}$$

- This lower triangular system can be solved in $O(n^2)$

Newton Basis: Parameters

- To **evaluate a *3rd* order polynomial at *x*:**

$$
\begin{aligned}
p_0(x) &= a_3 \\
p_1(x) &= a_2 + (x - x_2)p_0(x) \\
p_2(x) &= a_1 + (x - x_1)p_1(x) \\
p_3(x) &= a_0 + (x - x_0)p_2(x)
\end{aligned}
$$

Notice the recursive nature

- To **evaluate an *nth* order polynomial at *x*:**

$$
\begin{aligned}
p_0(x) &= a_n \\
&\vdots \\
p_k(x) &= a_{n-k} + (x - x_{n-k})p_{k-1}(x) \\
&\vdots \\
p_n(x) &= a_0 + (x - x_0)p_{n-1}(x)
\end{aligned}
$$

Newton Divided Difference Table

- The Newton coefficients $a_0, a_1, a_2, \ldots$ can be found by repeated difference calculations
  - Populate a table with the given points: x_i, y_i (with extra space for difference terms)
  - Process each column, calculating the Newton difference pairs.
  - No linear algebra ☺

$$
\begin{array}{llll}
x_0 & f[x_0] & & \\
 & & f[x_0;x_1] = \dfrac{f[x_1]-f[x_0]}{x_1-x_0} & \\
x_1 & f[x_1] & & f[x_0;x_1;x_2] = \dfrac{f[x_1;x_2]-f[x_0;x_1]}{x_2-x_0} \\
 & & f[x_1;x_2] = \dfrac{f[x_2]-f[x_1]}{x_2-x_1} & \\
x_2 & f[x_2] & &
\end{array}
$$

Newton and Divided Difference Table

- The coefficients $a_0, a_1, a_2, \ldots$ can be read from the **"top row"** of a ***"divided difference table"***

$$
\begin{array}{llll}
x_0 & f[x_0] & & \\
 & & f[x_0;x_1] = \dfrac{f[x_1]-f[x_0]}{x_1-x_0} & \\
x_1 & f[x_1] & & f[x_0;x_1;x_2] = \dfrac{f[x_1;x_2]-f[x_0;x_1]}{x_2-x_0} \\
 & & f[x_1;x_2] = \dfrac{f[x_2]-f[x_1]}{x_2-x_1} & \\
x_2 & f[x_2] & &
\end{array}
$$

Note: The ***number of operations*** to compute the table is:

$$\text{nops} = 3\frac{n(n+1)}{2} = O\left(\tfrac{3}{2}n^2\right)$$

Newton Divided Difference Example

x0	y0		
		f[x0;x1]= (y1-y0)/(x1-x0)	
x1	y1		f[x0;x1;x2] = $\frac{f[x1;x2] - f[x0;x1)]}{x2-x0}$
		f[x1;x2] = (y2 - y1)/(x2-x1)	
x2	y2		

(Difference in the cols)
(difference in the x)

data set {(15, 10), (20, 11), (25, 20)}

Solution: Construct a divided difference table

15	**10**		
		f[x0;x1]= (11-10)/(20-15) = .2	
20	11		f[x0;x1;x2] = $\frac{1.8 - .2 =}{25-15}$ .16
		f[x1;x2] = (20 - 11)/(25-20) = 1.8	
25	20		

Newton Divided Difference Example 1

- The divided difference table is

15	10		
		0.2	
20	11		0.16
		1.8	
25	20		

- Therefore: $a_0 = 10$, $a_1 = 0.2$, $a_2 = 0.16$

  and the interpolating polynomial is

$$\begin{aligned} p_2(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_2) \\ &= 10 + 0.2(x - 15) + 0.16(x - 15)(x - 20) \end{aligned}$$

Newton Divided Difference Example 2

- Generate the divided difference table for:

  (-1, 5), (0, 1), (1, 1), (2, 11)

- The table:

x	y			
-1	5			
		-4		
0	1		2	
		0		1
1	1		5	
		10		
2	11			

- Therefore: $a_0 = 5$, $a_1 = -4$, $a_2 = 2$, $a_3 = 1$

Newton: Main Advantage

- Newton's interpolation method can handle additional data points without recomputing all the coefficients (**it is recursive or "adaptive"**)

Example:

Suppose we already interpolated the data points

x	-1	0	1	2
y	5	1	1	11

and we want to add two more points

x	-2	3
y	5	35

Find the ***new interpolating polynomial***

Newton: Main Advantage

- We only need to update the last two "rows" of the divided difference table

-1	5					
		-4				
0	1		2			
		0		1		
1	1		5		**- 1/12**	
		10		**1 1/12**		**0**
2	11		**2 5/6**		**- 1/12**	
		1.5		**5/6**		
-2	**5**		**4 1/2**			
		6				
3	**35**					

They don't even have to be in "order"!

$$p(x) = 5 - 4(x+1) + 2(x+1)(x) + (x+1)(x)(x-1) - \frac{1}{12}(x+1)(x)(x-1)(x-2)$$

Error Estimates

- Let $f(x)$ be a smooth function and $P_n(x)$ the unique polynomial that interpolates $n+1$ distinct points of $f(x)$, *e.g.*, such that

$$y_k = f(x_k) = p_n(x_k)\ , \quad k=0,\ldots n$$

- Then the ***interpolation error at any point x*** is

$$e(x) = f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{n} (x - xk)$$

Note: This expression has limited practical value since we rarely know $f(x)$ and its $n+1$ derivative

Summary: Numerical Interpolation:

Basis	Construction	Evaluation
Monomial	$O(\frac{2}{3}n^3)$	$O(2n)$
Lagrange	$O(n^2)$	$O(5n)$
Newton	$O(\frac{3}{2}n^2)$	$O(2n)$

Complexity of Interpolation Approaches

- The **main advantage of Lagrange** interpolation is its **numerical stability**
- The **main advantage of Newton**'s method is its adaptivity (i.e., **recursive structure**), we can add more points *without recomputing all the coefficients*.

Summary: Polynomial Interpolation

- The two most common to numerical interpolation approaches are Lagrange and Newton
- The number of data points determines the degree of the polynomial to be interpolated (n+1 points require an n^{th} degree polynomial)
- ***Do not*** interpolate polynomials of ***order 4 or higher,*** they tend to ***oscillate.***

If more than 4 data points must be interpolated use

Piecewise Interpolation

Problem 1

- Given the data set {(1,6), (2, 1), (3, 3)}
- Write the Newton's polynomial that interpolates the data uniquely. Don't solve for a_x.

General Newton polynomial:

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

For this problem

$$p(x) = a_0 + a_1(x-1) + a_2(x-1)(x-2)$$

Problem 2

- Given the data{(1,6), (2, 1), (3, 3)}, complete the divided difference table to find the interpolating polynomial coefficients.
- Give your final answer in Newton's form.

1	6		
		f[x0;x1]= (1-6)/(2-1) = -5	
2	1		f[x0;x1;x2] = $\frac{2-(-5)}{3-1}$ = $\frac{7}{2}$
		f[x1;x2] = (3-1)/(3-2) = 2	
3	3		

$a_0 = 6;$
$a_1 = -5;$
$a_2 = 7/2;$

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

$$p(x) = 6 -5(x-1) + 7/2(x-1)(x-2)$$

Problem 3

- Given the following divided difference table, add (4,4) to it and find the new interpolating coefficients.
  - Give your answer in Newton's form.

1	6		
		-5	
2	1		3.5
		2	
3	3		

1	6			
		-5		
2	1		3.5	
		2		(-.5-3.5)/(4-1) = -1.33
3	3		(1-2)/(4-2) = -.5	
		(4-3)/(4-3) = 1		
4	4			

$a_0 = 6;$

$a_1 = -5;$

$a_2 = 3.5;$

$a_3 = -1.33;$

$$p(x) = 6 - 5(x-1) + 3.5(x-1)(x-2) - 1.33(x-1)(x-2)(x-3)$$