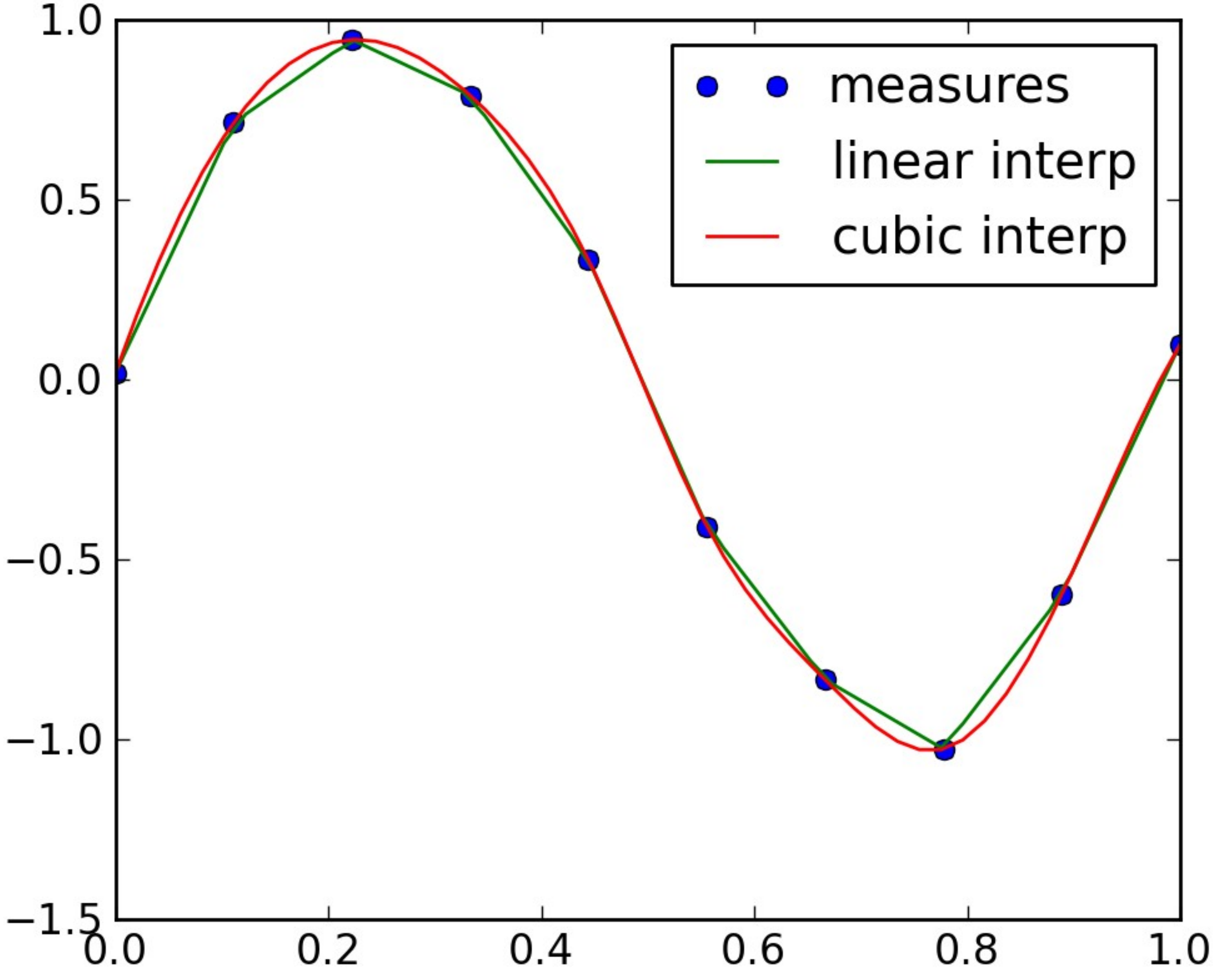
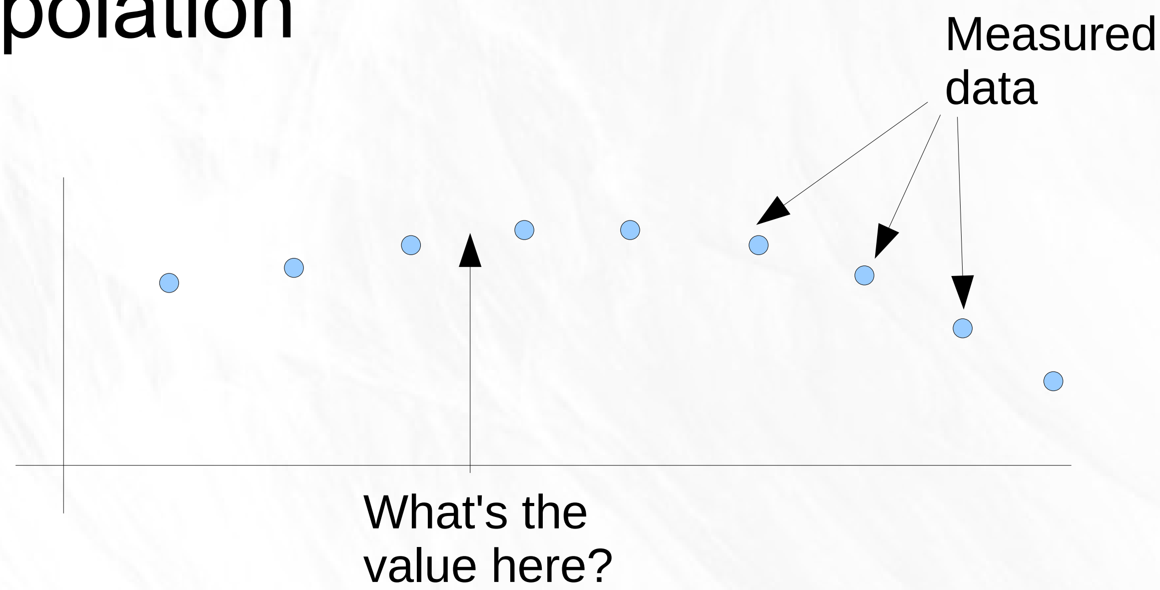
# Today: Interpolation



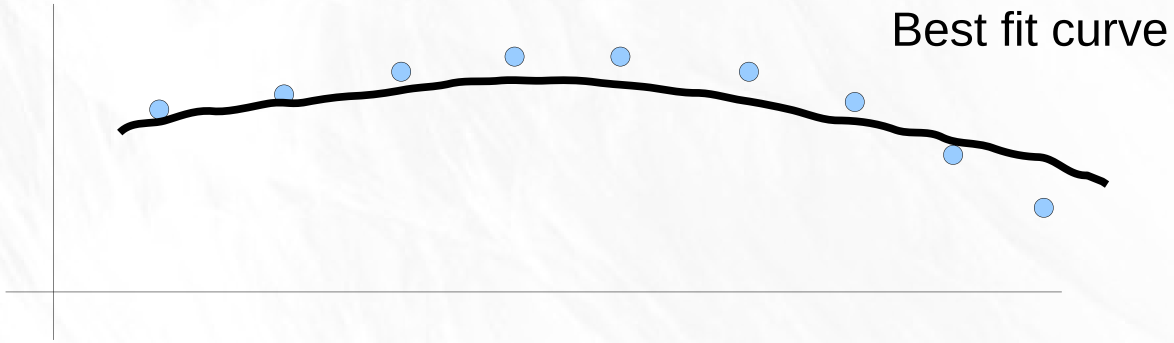
# Interpolation & Extrapolation

- Interpolation




- Extrapolation

# Different from regression




- Regression:
  - Fit some type of curve to the data
  - Curve might actually miss the data points
- Interpolation:  Curve goes through all data points

# Examples

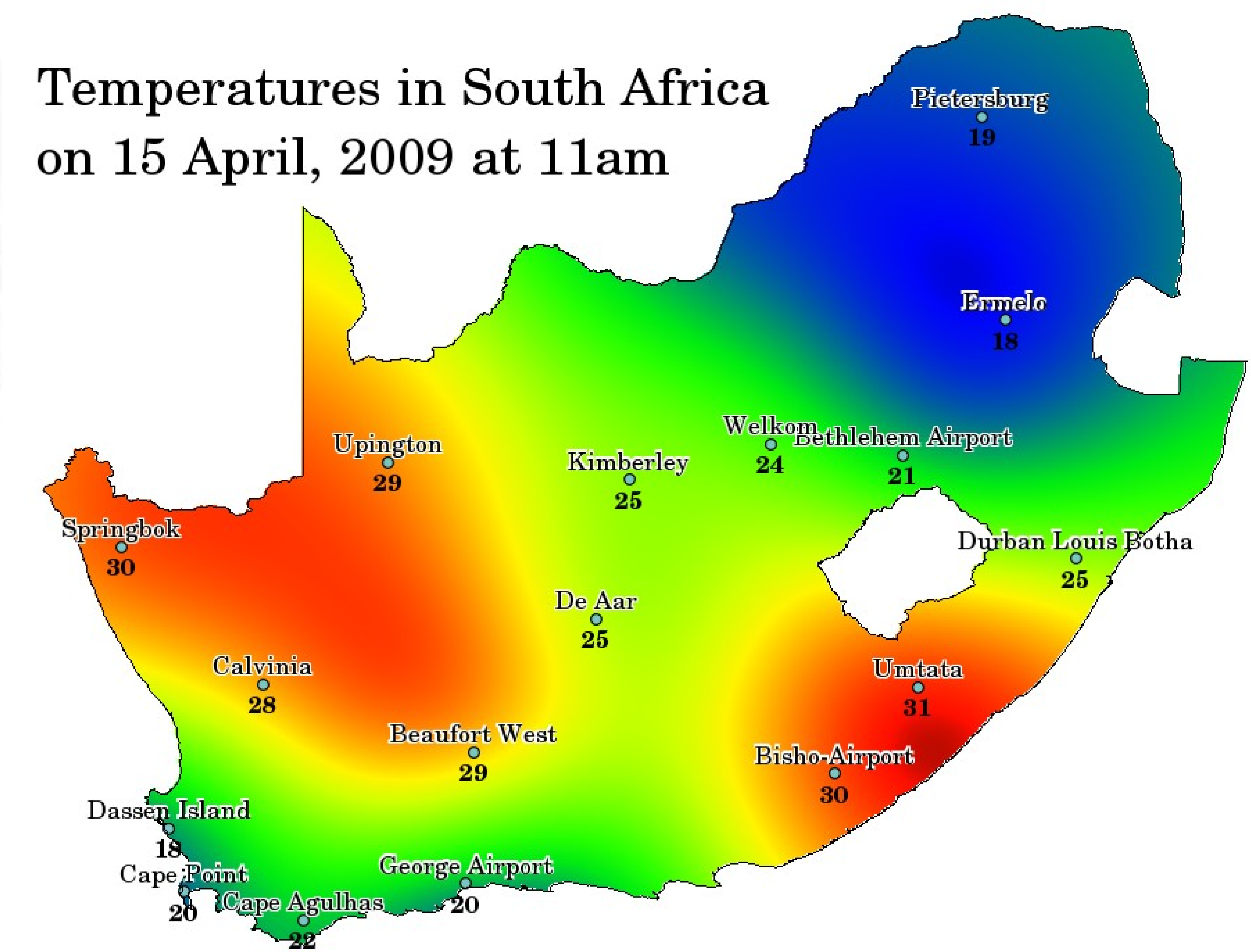
- Interpolation
  - Resize and redisplay an image
  - Upsample/downsample a sound file
  - Computation of special functions
- Extrapolation
  - What's tomorrow's stock price for IBM?
  - Where will the missile hit the ground?
- Regression
  - Fit a smooth line to noisy data
  - Fit complicated function to data

# 2 dimensional interpolation -- Upsampled image

Original image
(has "the jaggies")

Upsampled image
after interpolation

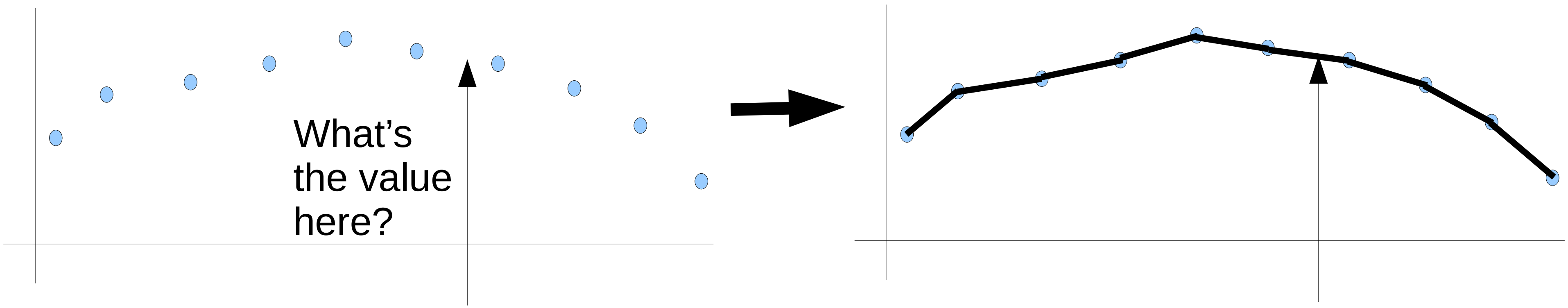
# Interpolation of temperature data



# Interpolation

- Two types of input -- two different strategies
  - Input points are evenly spaced (in x)
  - Input points are not evenly spaced (in x)
- Different interpolation schemes
  - Linear interpolation
  - Polynomial interpolation
  - Spline interpolation
- 1D vs. Multivariate

# Simplest: Linear interpolation



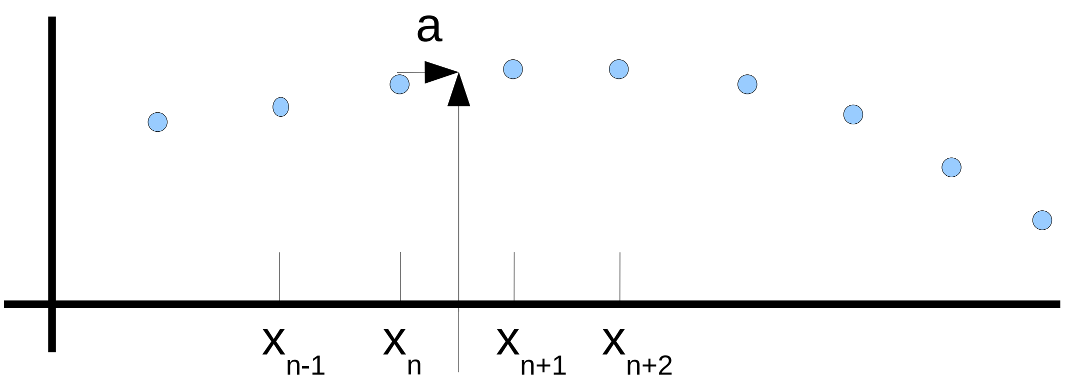

- Draw line between adjacent data points.
- Quick and dirty, but frequently suffices.
- Non-differentiable at x points.
- interp1 in matlab.

Example: test_linear_interpolation.m

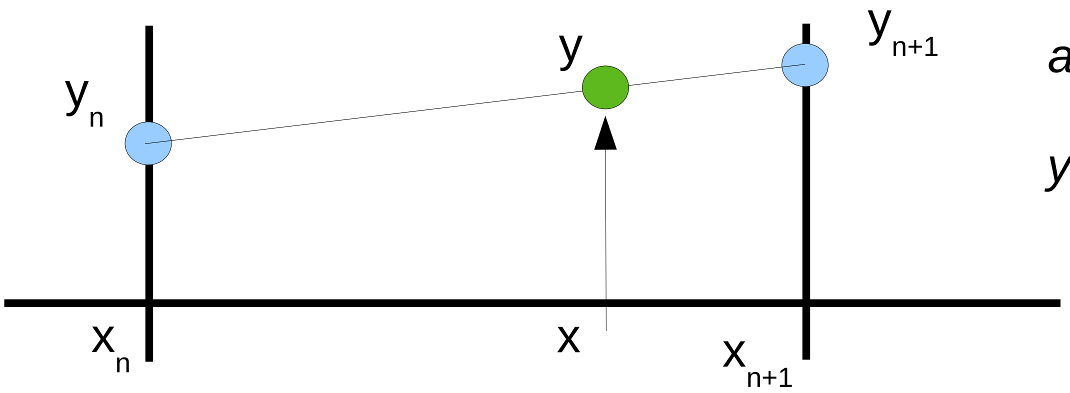
# Two pieces to algorithm

- Inputs: $(x_n, y_n)$ data (known), new point to interpolate, x.

- Find correct interval




- Interpolate inside interval

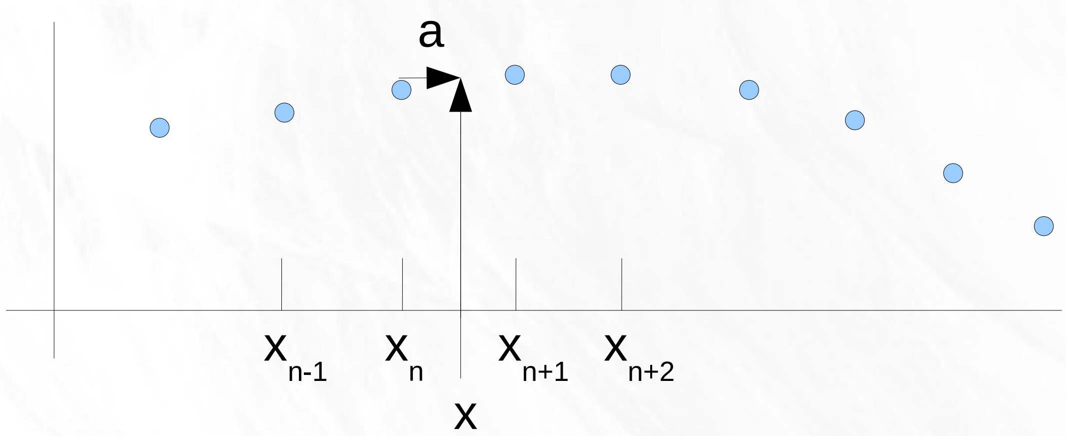



$a = (x - x_n)/(x_{n+1} - x_n)$

$y(x) = (1\text{-}a)y_n + ay_{n+1}$

# Linear interpolation algorithm

| | | | | | | |
|---|---|---|---|---|---|---|
| x0 | x1 | x2 | x3 | x4 | x5 | x6 |
| y0 | y1 | y2 | y3 | y4 | y5 | y6 |




1) Inputs: vectors $x_{dat}$ $y_{dat}$, scalar $x$

2) Find nearest point $x_n$ to left of input $x$

3) Compute $a = (x - x_n)/(x_{n+1} - x_n)$ (Note $0 \leq a \leq 1$)

4) Compute $y(x) = (1-a)y_n + ay_{n+1}$

# Octave program

```
function y = linear_interpolation(x_dat, y_dat, x)
  %  This function performs linear interpolation
  %  Inputs:  x_dat = vector of (evenly spaced) x data points
  %           The values are assumed sorted and increasing.
  %           y_dat = vector of corresopnding y values
  %           x = scalar x value where to computed interpolated y
  %  Outputs: y = scalar interpolated value at  x

  % Algorithm:  1.  Find adjacent points in x_dat on both sides of
  %                 input x.
  %             2.  Compute eta = fractional distance from point on left
  %             3.  Interpolate: y = (1-eta)*y1 + eta*y2

  % Find first x_dat greater than the input x (x_dat to the right
  % of the input x)
  t = x < x_dat;
  idx2 = (find(t))(1);

  % Verify input x is valid
  if (idx2 < 2 || idx2 > length(x_dat))
    % error -- Input x is outside of interpolation domain
    error("Input x is outside of interpolation domain.")
  end

  % Get index of x_dat value to the left of the input x value.
  idx1 = idx2-1;
  eta = (x - x_dat(idx1)) / (x_dat(idx2) - x_dat(idx1));

  % Compute interpolated value
  y = (1 - eta)*y_dat(idx1) + eta*y_dat(idx2);
return
```

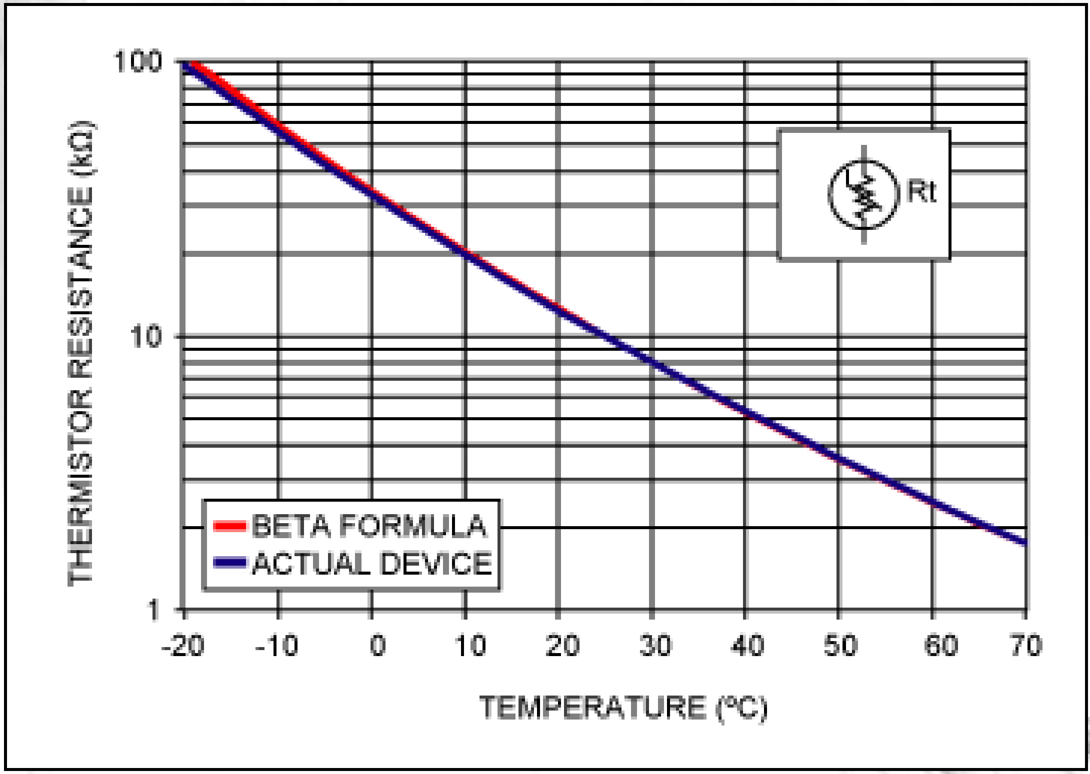
find $x_{n+1}$

$$\eta = \frac{x - x_n}{x_{n+1} - x_n}$$

$$y = (1-\eta)\, y_n + \eta\, y_{n+1}$$

# Example from real world: Thermistor

- Thermistor used to measure temperature.
- Resistance of device depends upon temperature.

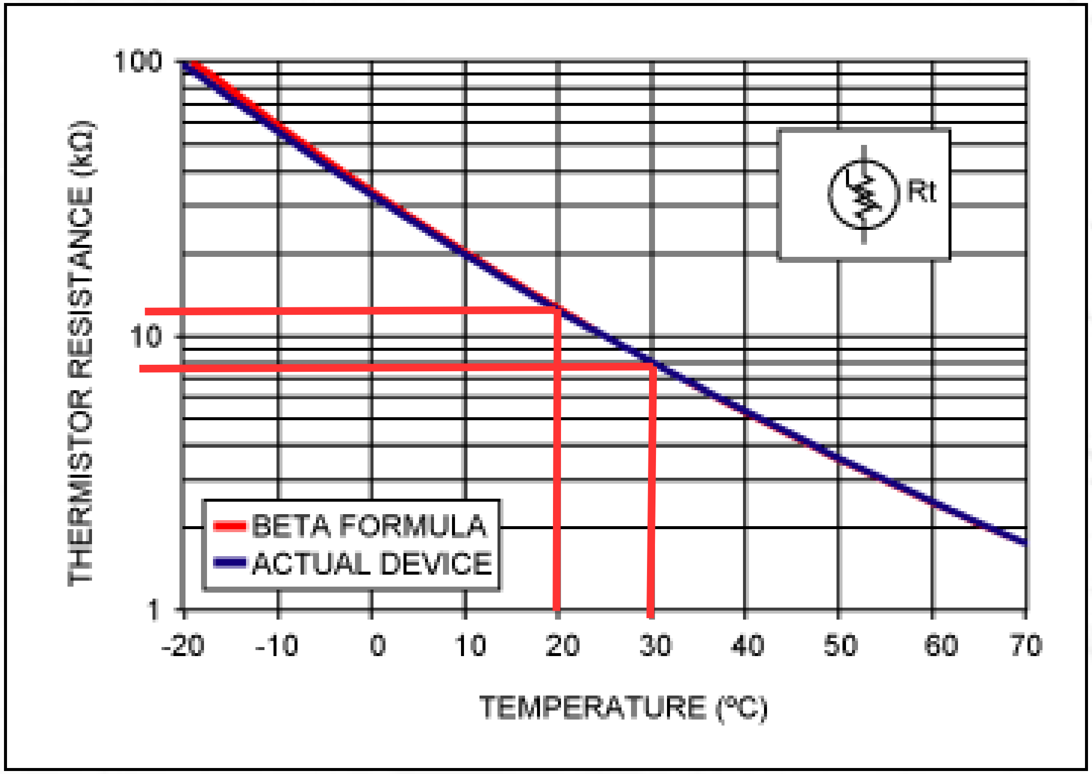



- Goal: report temperature from measured resistance.
- Often you get only a plot of resistance vs. Temp.

# Create interpolation table

- Use pencil and ruler to create table of resistance vs. Temp.

| Temp | Res |
|---|---|
| -20 | 1.00E+005 |
| -10 | 5.50E+004 |
| 0 | 3.30E+004 |
| 10 | 2.00E+004 |
| 20 | 1.20E+004 |
| 30 | 8.00E+003 |

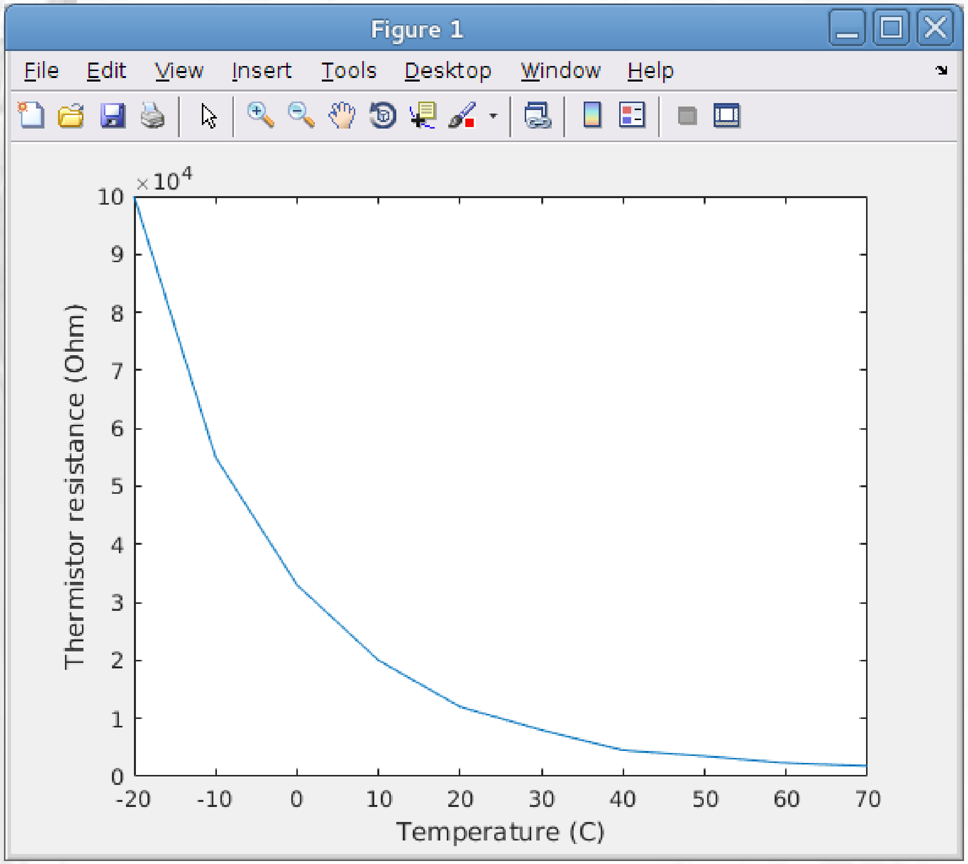


# Write interpolation function

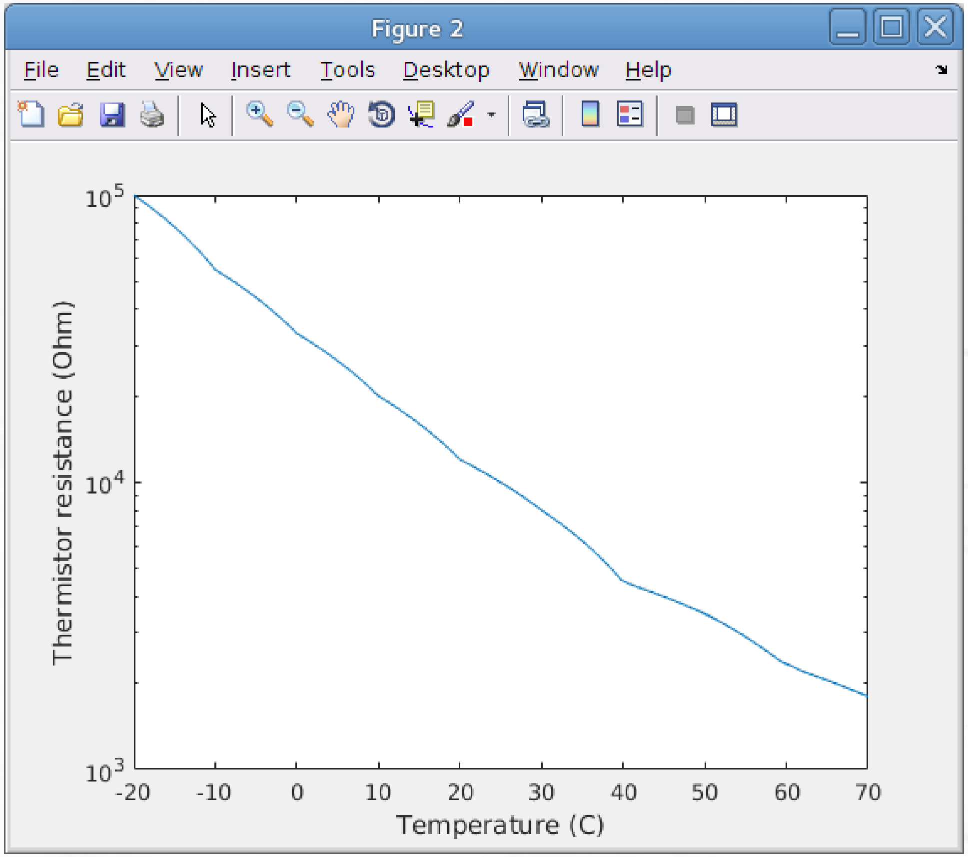
```
function T = get_temp(R)
  % Uses look-up table and interpolation to return
  % a value for temp when given an input R.

Rtab = [
    -20, 1e5;
    -10, 5.5e4;
    0, 3.3e4;
    10, 2.0e4
    20, 1.2e4;
    30, 8.0e3;
    40, 4.5e3;
    50, 3.5e3;
    60, 2.3e3;
    70, 1.8e3
    ];

  % Find first table R below input R
  for i = (N-1):-1:1
    % fprintf('Checking Rtab(%d, 2) = %f\n', i, Rtab(i,2))
    if (Rtab(i,2) >= R)
      % Found it.
      R1 = Rtab(i,2);
      R2 = Rtab(i+1,2);
      alpha = (R-R1)/(R2-R1);
      T = (1-alpha)*Rtab(i,1) + alpha*Rtab(i+1,1);
      return
    end
  end
end
```

Here's where I do the interpolation

# Interpolation results

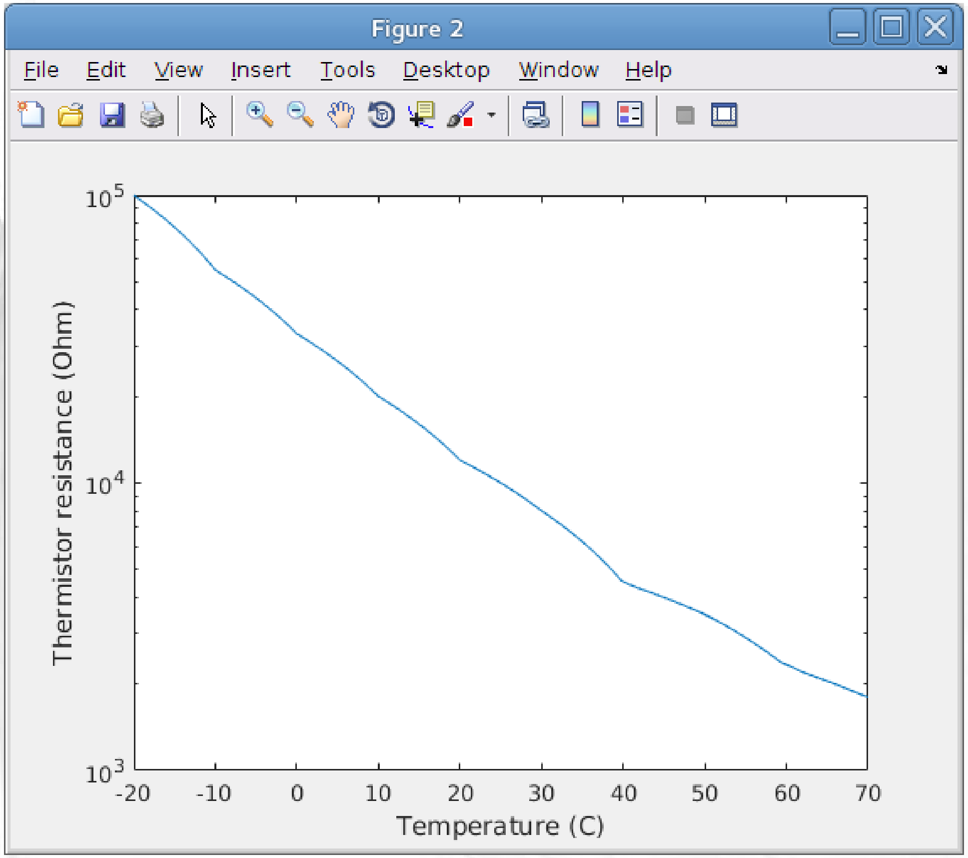


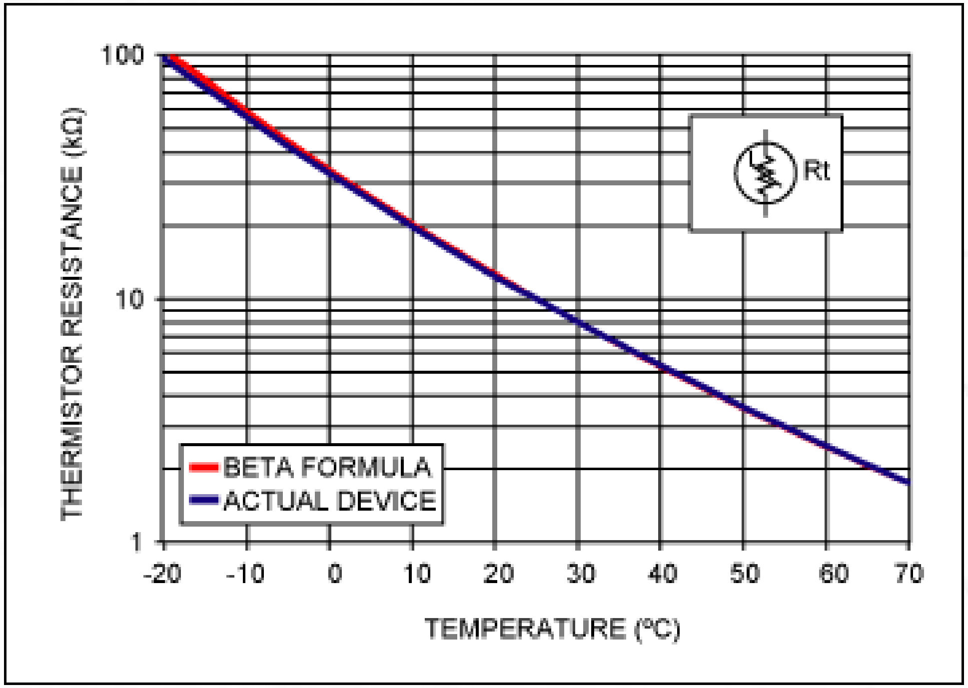

Lin plot




Log plot

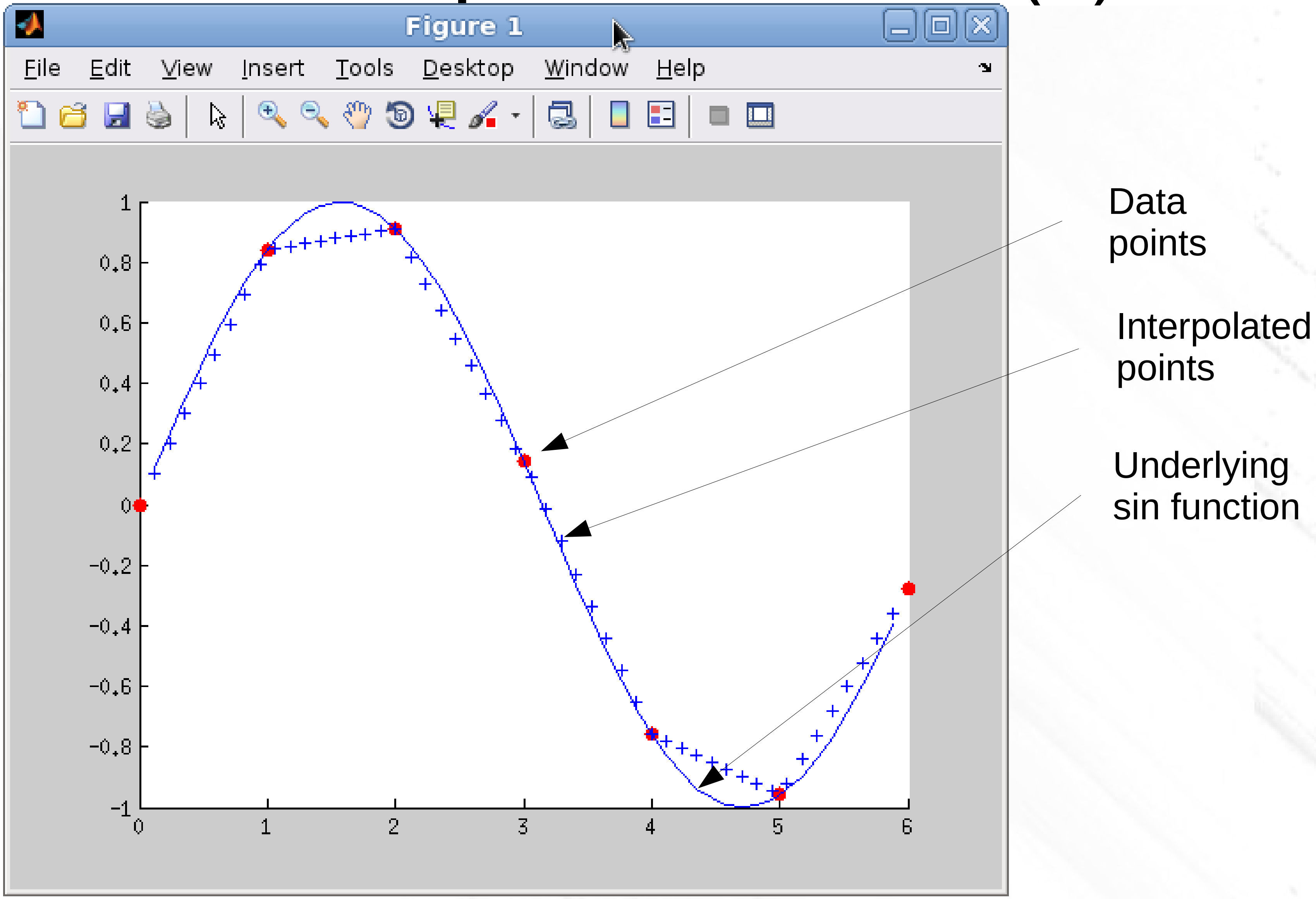
# Comparison with datasheet




- Note slope discontinuities at knot points.
- Whether you care about them (or not) depends upon your application.

# Recall example: Linear interpolation of sin(x)




Obviously, this stinks

# Polynomial interpolation – General

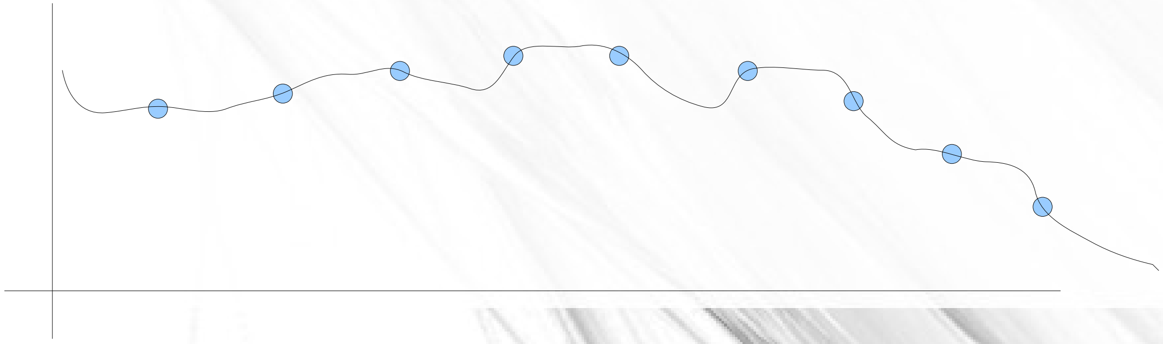
We want to find polynomial

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n$$

Where at each data point *j*

$$P(x_j) = a_0 + a_1 x_j + a_2 x_j^2 + a_3 x_j^3 + \ldots + a_n x_j^n = y_j$$

Between the data points $x_j$ the polynomial will interpolate the function.



# How many points? What degree poly?

- N data points => polynomial of order n-1.
  - 1 data point => $P(x) = a_0$ (constant)
  - 2 data points => $P(x) = a_0 + a_1 x$ (line)
  - 3 data points => $P(x) = a_0 + a_1 x + a_2 x^2$ (quadratic)
- In general, you can pass a polynomial of degree n-1 through n points.
- Also, that polynomial is unique.

*Explanation on blackboard*

# Consider degree N polynomial

- Degree N → N+1 coefficients

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_N x^N$$

- N+1 unknown coefficients → require N+1 equations (points) to determine all coeffs.

- Equations will be something like

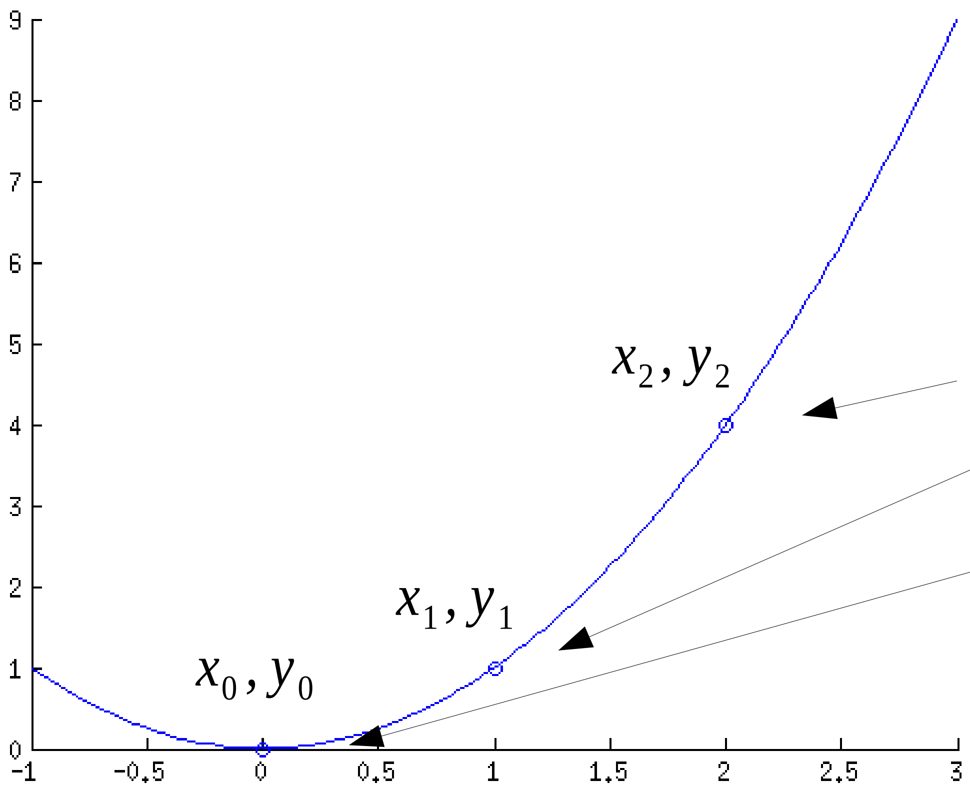
$$y_1 = P(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + \cdots + a_N x_1^N$$

$$y_2 = P(x_2) = a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 + \cdots + a_N x_2^N$$

... etc ...

# Example

- Three points, 2nd degree polynomial




This line must pass through all three points. This gives three equations:

$$a_0 + a_1 x_2 + a_2 x_2^2 = y_2$$

$$a_0 + a_1 x_1 + a_2 x_1^2 = y_1$$

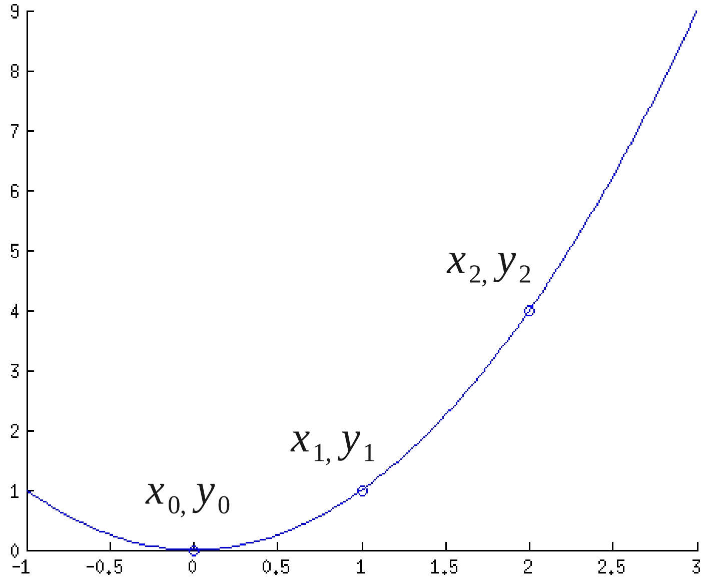
$$a_0 + a_1 x_0 + a_2 x_0^2 = y_0$$

Solve equations to get $a_n$ coefficients.

# How to get the coefficients?

- Three equations, three unknowns:

$$a_0 + a_1 x_0 + a_2 x_0^2 = y_0$$
$$a_0 + a_1 x_1 + a_2 x_1^2 = y_1$$
$$a_0 + a_1 x_2 + a_2 x_2^2 = y_2$$




- Rewrite as matrix expression

$$\begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

- Solve for $a_0$, $a_1$, $a_2$.

# General interpolating polynomial satisfies a matrix equation

$$\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \ldots \\ 1 & x_1 & x_1^2 & x_1^3 & \ldots \\ 1 & x_2 & x_2^2 & x_2^3 & \ldots \\ 1 & x_3 & x_3^2 & x_3^3 & \ldots \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \ldots \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \ldots \end{bmatrix}$$

- This is a “Vandermonde matrix”
  - Appears commonly in numerical analysis. Examples: interpolation (here), and discrete Fourier transforms.
- We know x and y, so we can solve for a, right?

/home/sdb/Northeastern1/Class8/vandermonde.m

# Problems...

- Vandermonde matrix badly conditioned for real x and high N (i.e. it has a high condition number).
- Conditioning gets worse as the degree of the polynomial increases.
- This means coefficient vector [a] will depend sensitively upon the x and y values in the data.

# Why is the Vandermonde matrix ill-conditioned?

$$\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \ldots \\ 1 & x_1 & x_1^2 & x_1^3 & \ldots \\ 1 & x_2 & x_2^2 & x_2^3 & \ldots \end{bmatrix}$$

$$x_0=5, x_1=6, x_2=7, \ldots$$

$$\begin{bmatrix} 1 & 5 & 25 & 125 & \ldots \\ 1 & 6 & 36 & 216 & \ldots \\ 1 & 7 & 49 & 343 & \ldots \end{bmatrix}$$

Rows not very orthogonal

Matrix becomes more ill-conditioned as the polynomial order increases.

- Badly conditioned matrix – row vectors are not very orthogonal.
- Therefore, doing high-degree interpolation over many points is a bad idea.

# Recall what we are trying to do

- You have a set of N {x, y} pairs (data)
- You want to find an expression representing a line which passed through all {x, y} points.
- You want to use this expression to interpolate the data.
- The points can be evenly spaced, or unevenly spaced.

# Lagrange polynomial interpolation

Interpolation polynomial:

$$L(x) = \sum_j y_j l_j(x)$$

Individual terms:

$$l_j(x) = \prod_{m \neq j} \frac{x - x_m}{x_j - x_m}$$

Note term involving $x_j$ is missing

$$= \frac{x - x_1}{x_j - x_1} \dots \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} \dots \frac{x - x_m}{x_j - x_m}$$

Consider the $l_j(x)$ as basis functions in an expansion for $L(x)$.

# Example

$$L(x) = \sum_j y_j l_j(x)$$

$$l_j(x) = \prod_{m \neq j} \frac{x - x_m}{x_j - x_m}$$

Take x = [1 2 3],   y = [1 4 9]

$$L(X) = \underbrace{1}_{y_1} \cdot \underbrace{\frac{1}{2}(x-2)(x-3)}_{l_1(x)} + \underbrace{4}_{y_2} \cdot \underbrace{(-1)(x-1)(x-3)}_{l_2(x)} + \underbrace{9}_{y_3} \cdot \underbrace{\frac{1}{2}(x-1)(x-2)}_{l_3(x)}$$

Lagrange interpolation polynomial

– *Note: 3 data points, 2nd degree polynomial*

Check:

- L(1) -> 1
- L(2) -> 4
- L(3) -> 9
- L(2.5) -> 6.25

# Python example code

$$L(x) = \sum_j y_j l_j(x) \qquad l_j(x) = \prod_{m \neq j} \frac{x - x_m}{x_j - x_m}$$

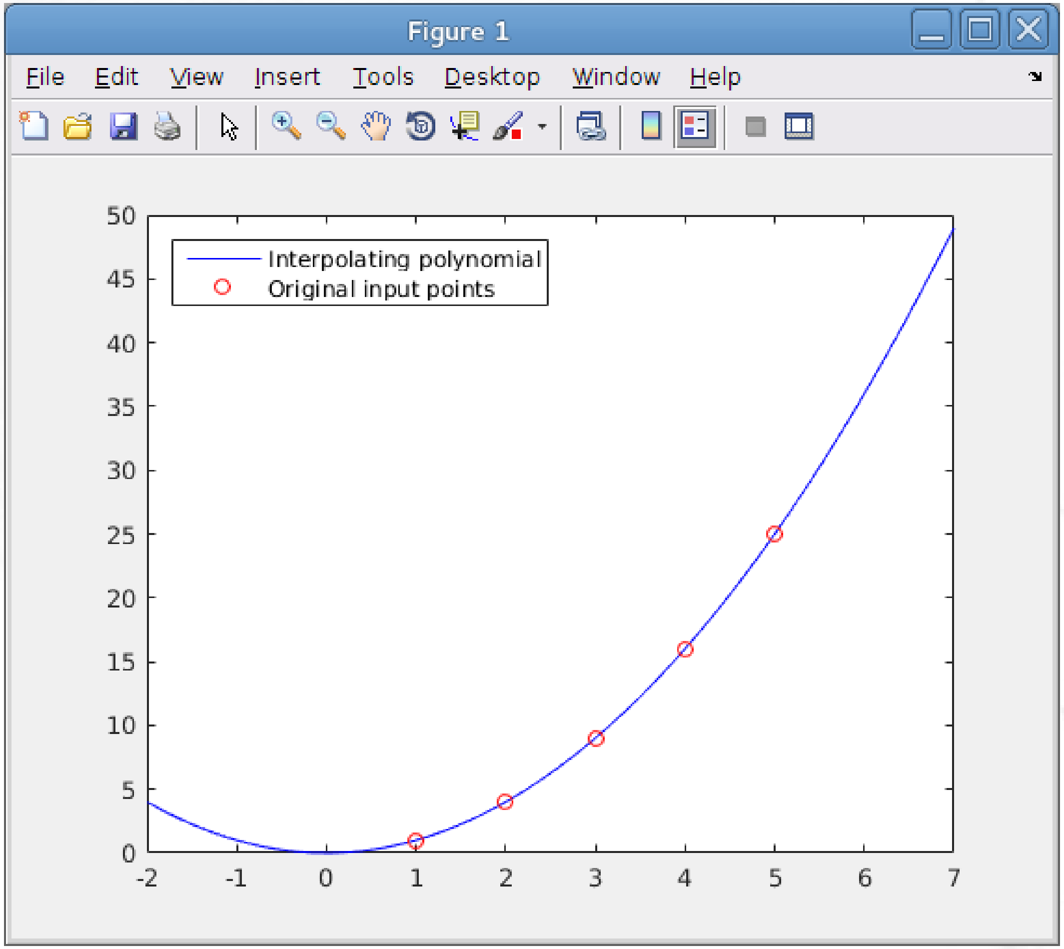
```
import numpy

def LagrangeInterpolate(xvec, yvec, x):

    L = 0
    N = len(xvec)
    for j in range(N):
        l_j = 1
        for m in range(N):
            if j != m:
                l_j = l_j * (x - xvec[m])/(xvec[j] - xvec[m])
        L = L + l_j*yvec[j]
    return L
```

What's the complexity of this algorithm?

/home/sdb/Northeastern1/Class8/test_lagrange.py

## Implementation

```
function L = lagrange_poly(xvec, yvec, x)
   L = 0;
  N = length(xvec);
  for j=1:N
    lj = 1;
    for m=1:N
      if (j ~= m)
        lj = lj*(x-xvec(m))/(xvec(j)-xvec(m));
      end
    end
    L = L+lj*yvec(j);
  end
end
```

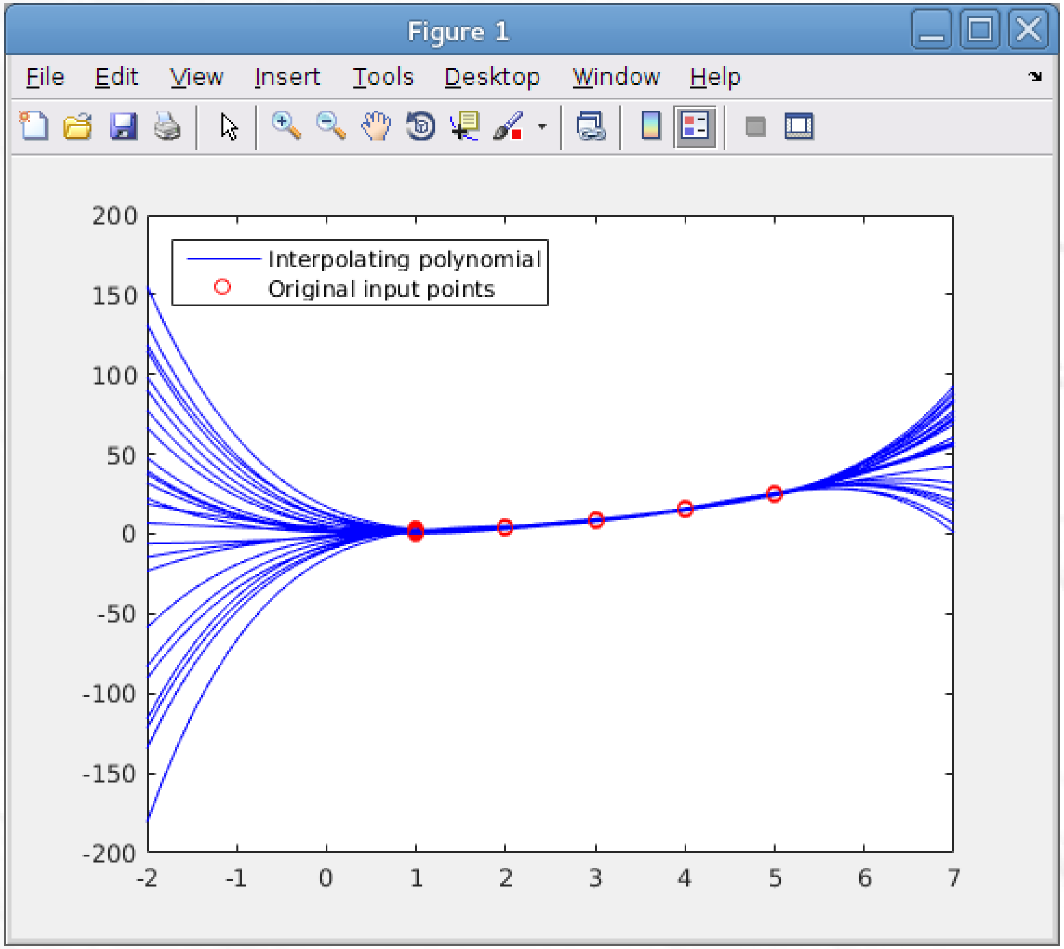



## Test

```
% Try interpolating quadratic
xn = [1., 2., 3., 4., 5.];
fn = [1., 4., 9., 16., 25.];

N = 50;
y = zeros(N,1);
x = linspace(-2, 7, N);
for idx=1:N
  y(idx) = lagrange_poly(xn, fn, x(idx));
end

plot(x, y, 'b')
hold on
plot(xn, fn, 'ro')
legend('Interpolating polynomial', 'Original input points', 'Location', 'northwest')
```

# Adding noise to data points

```
% Try interpolating quadratic
xn = [1., 2., 3., 4., 5.];
fn = [1., 4., 9., 16., 25.];

for cnt = 1:25

  % Add some noise
  nn = 0.3*randn(1, 5);
  fn = fn+nn;

  N = 50;
  y = zeros(N,1);
  x = linspace(-2, 7, N);
  for idx=1:N
    y(idx) = lagrange_poly(xn, fn, x(idx));
  end

  plot(x, y, 'b')
  hold on
  plot(xn, fn, 'ro')

end
```




- Interpolating polynomial goes haywire outside of domain

# Lagrange Interpolation Problems with Algorithm

$$L(x)=\sum_j y_j l_j(x) \qquad l_j(x)=\prod_{m\neq j}\frac{x-x_m}{x_j-x_m}$$

- Algorithm is $O(N^2)$ (2 loops).
- You need to run both loops for ***each*** data point you want to interpolate.
- If you add a new {x, y} pair (new datapoint), you need to redo the entire computation (i.e. get new $l_j$ coefficients).
- *Increasingly ill-conditioned as the number of data points increases (and distance between ends grows).*

# Improvement to Lagrange algorithm

- Recall expression for Lagrange basis terms

$$l_j(x) = \prod_{m \neq j} \frac{x - x_m}{x_j - x_m} \qquad j = 0, 1, \cdots, n$$

- Define expression for *l(x)* (new function)

$$l(x) = (x - x_0)(x - x_1)(x - x_2) \cdots (x - x_n)$$

- Define "barycentric weights"

$$w_j(x) = \frac{1}{\prod_{m \neq j} (x_j - x_m)} \qquad j = 0, 1, \cdots, n$$

- Then we can write interpolating polynomial as

$$L(x) = l(x) \sum_{j=0}^{n} \left( \frac{w_j}{x - x_j} y_j \right)$$

# Barycentric interpolation formula

- Now consider interpolating the function 1:

$$1 = l(x) \sum_{j=0}^{n} \left( \frac{w_j}{x - x_j} \right)$$

- Divide *L(x)* by this to get:

$$L(x) = \frac{\sum_{j=0}^{n} \left( \frac{w_j}{x - x_j} y_j \right)}{\sum_{j=0}^{n} \left( \frac{w_j}{x - x_j} \right)}$$

*Note this is simply a rewrite of the Lagrange formula – not a different formula.*

# Barycentric Interpolation

$$L(x)=\frac{\sum_{j=0}^{n}\left(\frac{w_j}{x-x_j}y_j\right)}{\sum_{j=0}^{n}\left(\frac{w_j}{x-x_j}\right)}$$

$w_j$ computed upfront

- Idea: Split algorithm into two:
  - Upfront preparation (i.e. compute $w_j$). $O(N^2)$
  - Quick computation for each x (i.e. one loop). O(N)
- OO approach: Create object carrying around the weights. Then invoke an “evaluate method” to compute individual interpolations.

# Python Code -- preparation

$$w_j(x) = \frac{1}{\prod_{m \neq j} (x_j - x_m)}$$

```
class LagrangeInterpolate:

    # Constructor
    def __init__(self, xn, fn):
        """
        Constructor takes data points xn and fn, and
        creates weights vector.
        """
        self.N = len(xn)
        self.fn = fn
        self.xn = xn
        self.w = numpy.zeros(self.N)
        for j in range(self.N):
            tmp = 1
            for k in range(self.N):
                if (j != k):
                    tmp = tmp*(xn[j] - xn[k])
            self.w[j] = 1/tmp
        return
```

# Python Code -- interpolator

$$L(x)=\frac{\sum_{j=0}^{n}\left(\frac{w_j}{x-x_j}y_j\right)}{\sum_{j=0}^{n}\left(\frac{w_j}{x-x_j}\right)}$$

```
# Interpolator
def Interpolate(self, x):
    """
    This uses interpolation formula in Trefethen paper, eq. 4.2.
    """
    # If input lies exactly on an xn, return stored fn since if we
    # try to do computation, it will return nan.
    idx = numpy.where(x == self.xn)[0]
    if (idx):
        return self.fn[idx]

    # Compute interpolated value
    num = 0.0
    denom = 0.0
    for j in range(self.N):
        tmp = self.w[j]/(x - self.xn[j])
        num = num + tmp*self.fn[j]
        denom = denom + tmp
    return num/denom
```

# Invocation

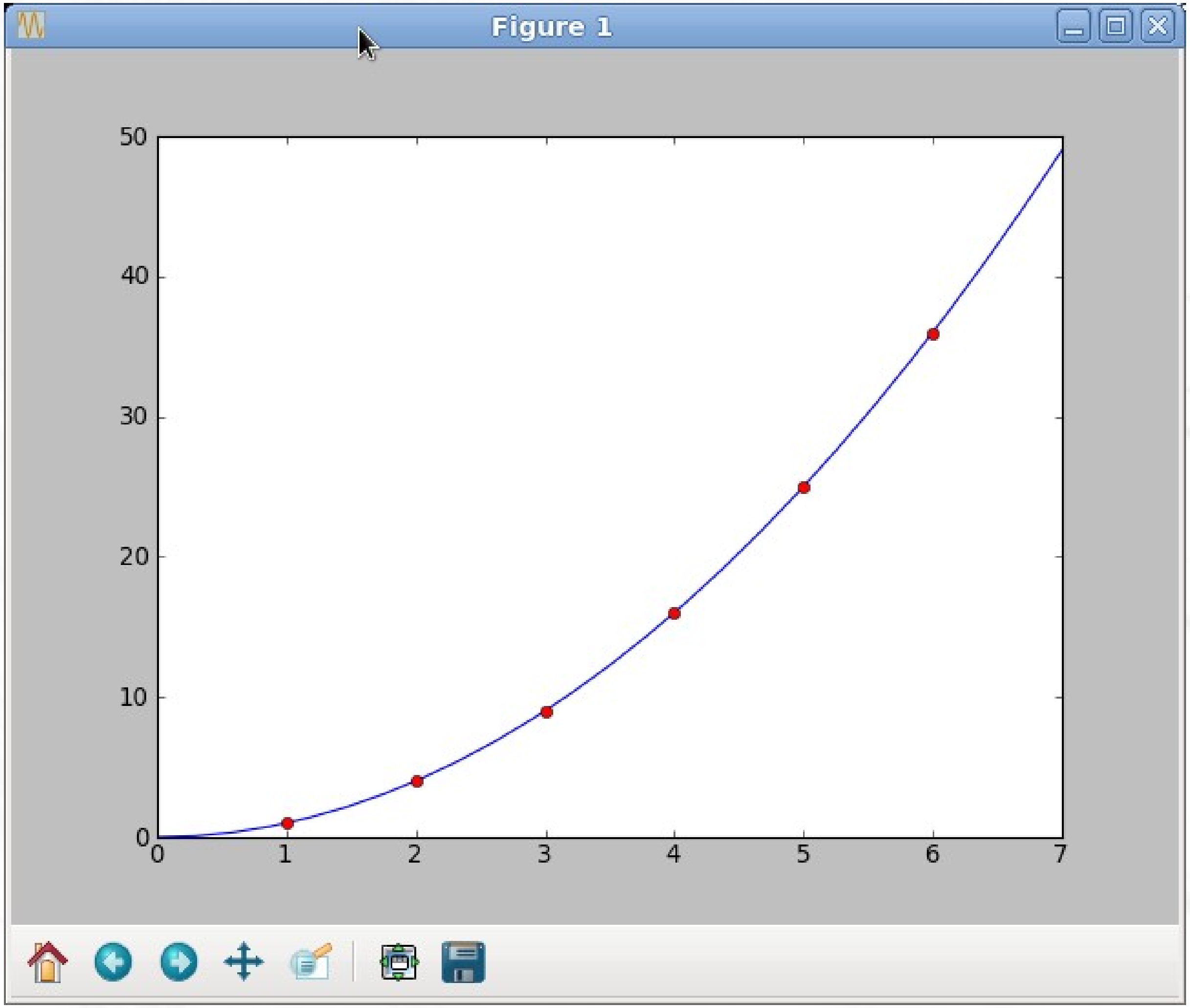
```
import numpy
import Barycentric
import matplotlib.pyplot as plt

# Try interpolating quadratic
x = numpy.array([1., 2., 3., 4., 5.])
f = numpy.array([1., 4., 9., 16., 25.])

# Create interpolation object
int1 = Barycentric.LagrangeInterpolate(x, f)

# Invoke Interpolation method on object
int1.Interpolate(2.5)
int1.Interpolate(3)

x = numpy.linspace(0, 10, 50)
y = map(int1.Interpolate, x)

line = plt.plot(x, y)
plt.show()
```

Note that my old test function fails on newer Pythons – must use list(map()) ...

# Interpolating quadratic curve

- Red dots: data points
- Blue line:

  Line drawn using interpolating polynomial

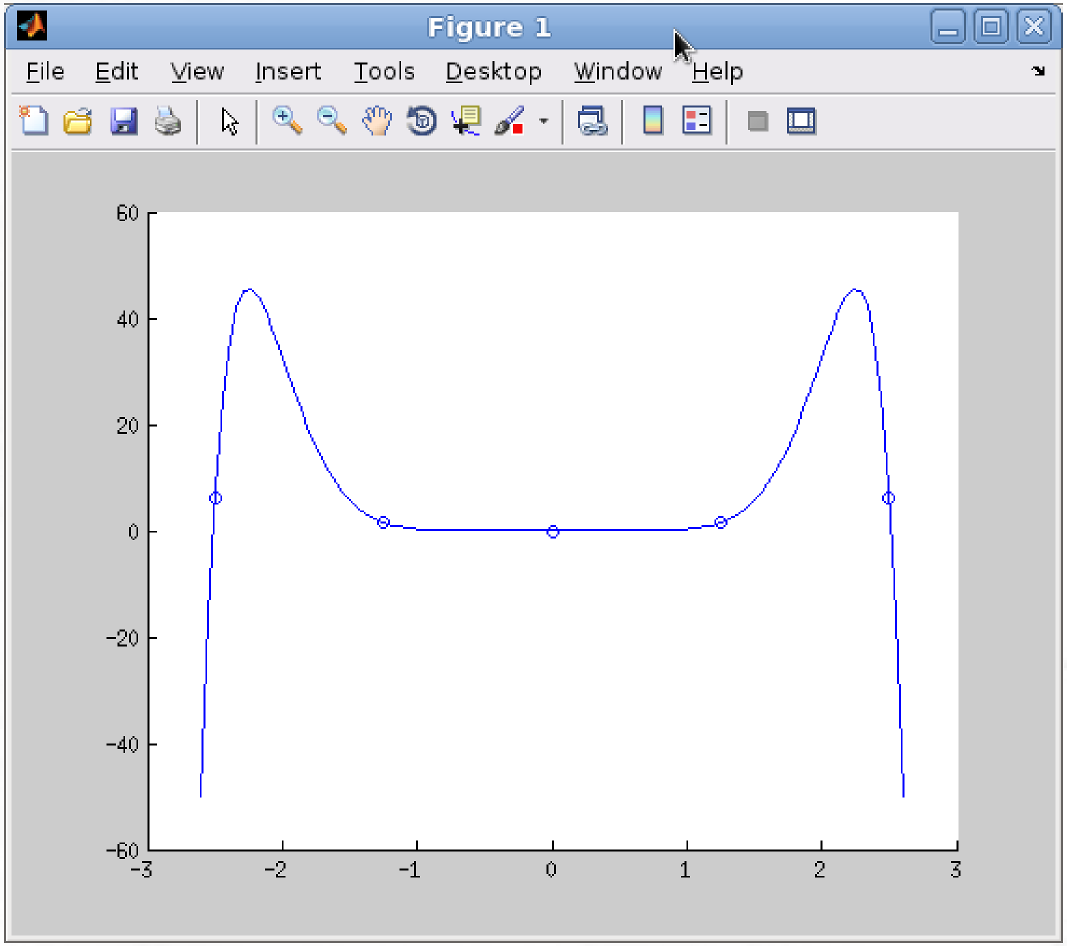


# Barycentric interpolation advantages

- Preparation is $O(N^2)$.
- Computation is $O(N)$ at each point.
- Useful if you want to call the interpolator multiple times with different inputs.
- But your code must trap input $x = x_j$ and return $y_j$ in this case. (Otherwise, you get NaN.)

# Higher Order Polynomials – Runge Phenomenon

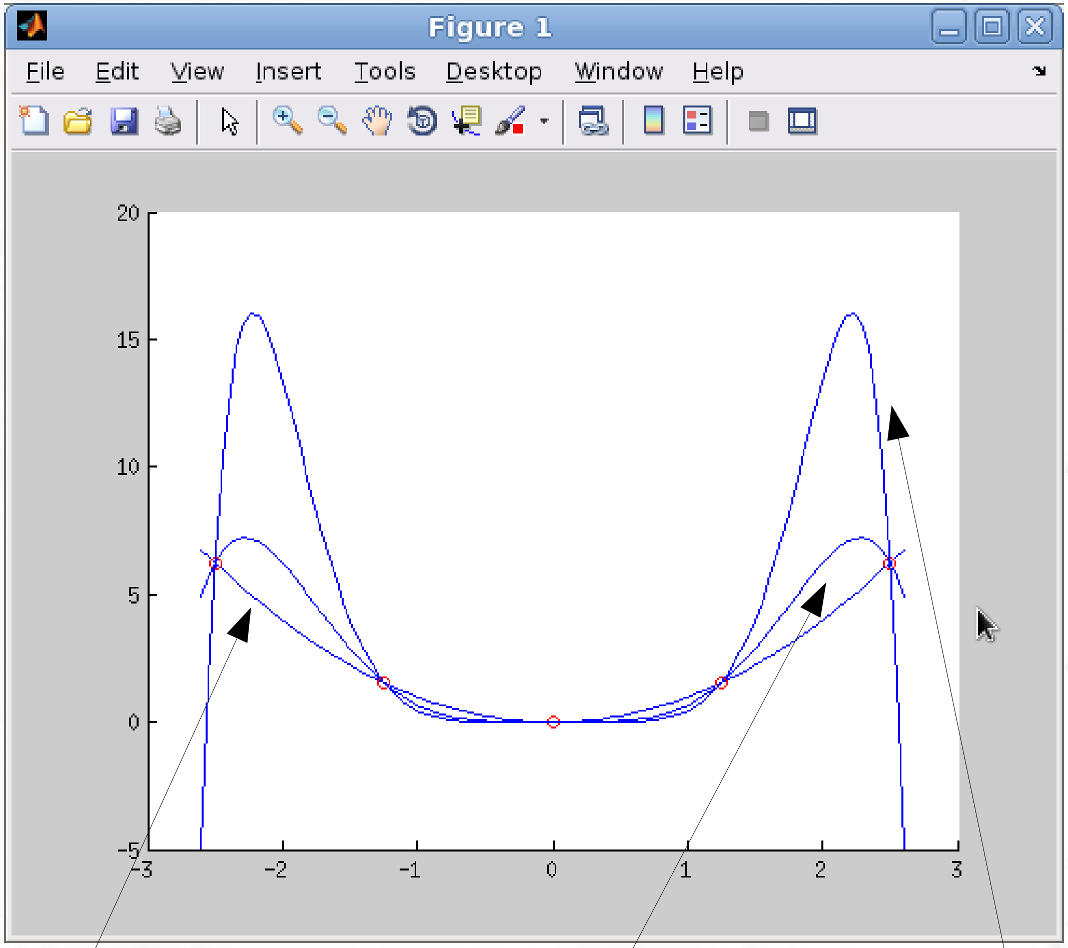
- Use higher-order polynomial to fit larger intervals and more points, right?
- Wrong!
- Runge phenomenon: Misbehavior of interpolation near ends of domain.




$10^{th}$ order polynomial fit to 5 points. Points are generated from quadratic, $y = x^2$.

# Runge Phenomenon

- Interpolating poly goes crazy at end of domain due to high order terms.
- This is called the Runge Phenomenon
- In this case, the original function is quadratic.
- Use deg = N-1 for N data points.




Run code in Vandermonde

$4^{th}$ degree interpolating poly

$6^{th}$ degree interpolating poly

$9^{th}$ degree interpolating poly

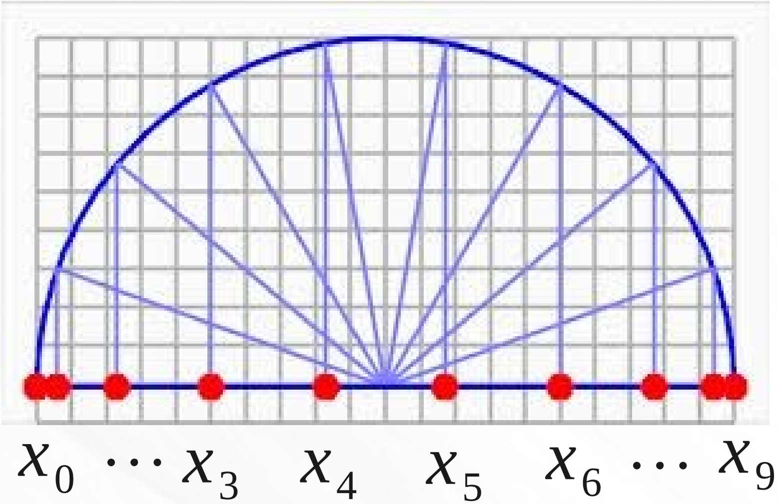
# Remarks

- Don't interpolate high degree polynomials over large numbers of points.
- Chop up your interval and interpolate low degree polynomials over short intervals (spline).
- Don't use uniformly-spaced x values unless they are forced on you -- in situations where you have the leeway to choose the x values, use Chebyshev nodes.
  - Example: Computing special fcns in hand calculators.

# A better way: Interpolation using Chebyshev nodes

- Use Lagrange interpolation formula, but choose x values to be Chebyshev nodes.
- For N point interpolation, the Chebyshev nodes are

$$x_i = \cos\left(\frac{2i+1}{2N+2}\pi\right) \quad 0 \le i \le N$$




- These are the roots of the Chebyshev polynomial $T_{N+1}(x)$ given *N* = point count
- They are also found via the circle construction above.

# Lagrange polynomial on Chebyshev nodes

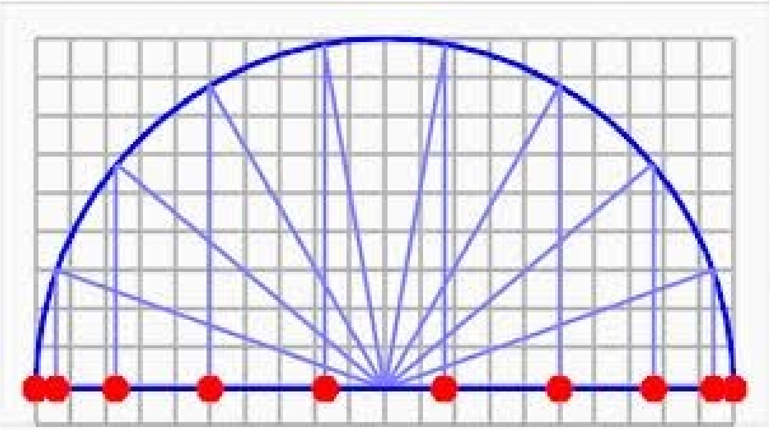
- Lagrange polynomial:

$$L(x)=\sum_j y_j l_j(x) \qquad l_j(x)=\prod_{m\neq j}\frac{x-x_m}{x_j-x_m}$$

Don’t forget to use barycentric formulation of this polynomial

- Chebyshev nodes:

$$x_m=\cos\left(\frac{2m+1}{2N+2}\pi\right) \qquad 0\leq m\leq N$$

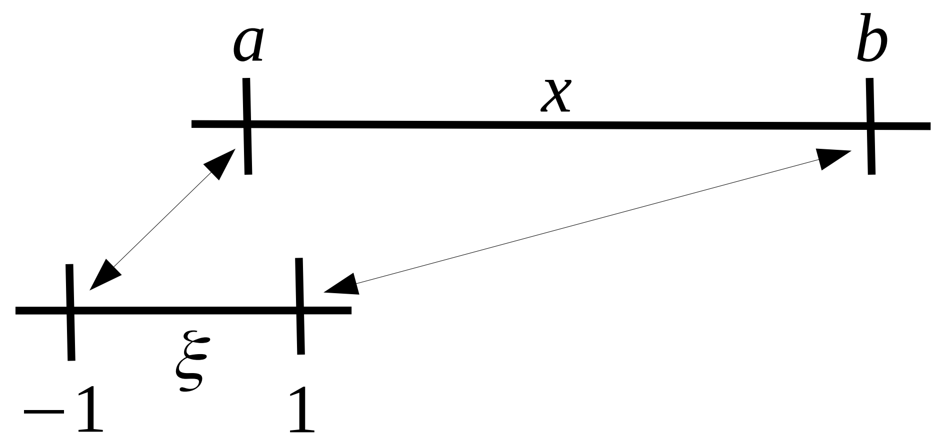



- Domain of nodes

$$x_m\in(-1,1)$$

# My function lives on [a,b], Chebyshev nodes live on [-1,1]

- What to do?
- Use linear map ...




$$x = s\,\xi + t$$

slope

offset

- Now insert info about end points and get coeffs.

$$\boxed{\begin{aligned} a &= -s + t \\ b &= s + t \end{aligned}} \Rightarrow \boxed{\begin{aligned} t &= (b+a)/2 \\ s &= (b-a)/2 \end{aligned}} \Rightarrow \boxed{\begin{aligned} x &= s\,\xi + t \\ \xi &= \frac{x-t}{s} \end{aligned}}$$

You can go back and forth

# Chebyshev polynomials (1st kind)

- Form an orthogonal set on the interval [-1, 1]

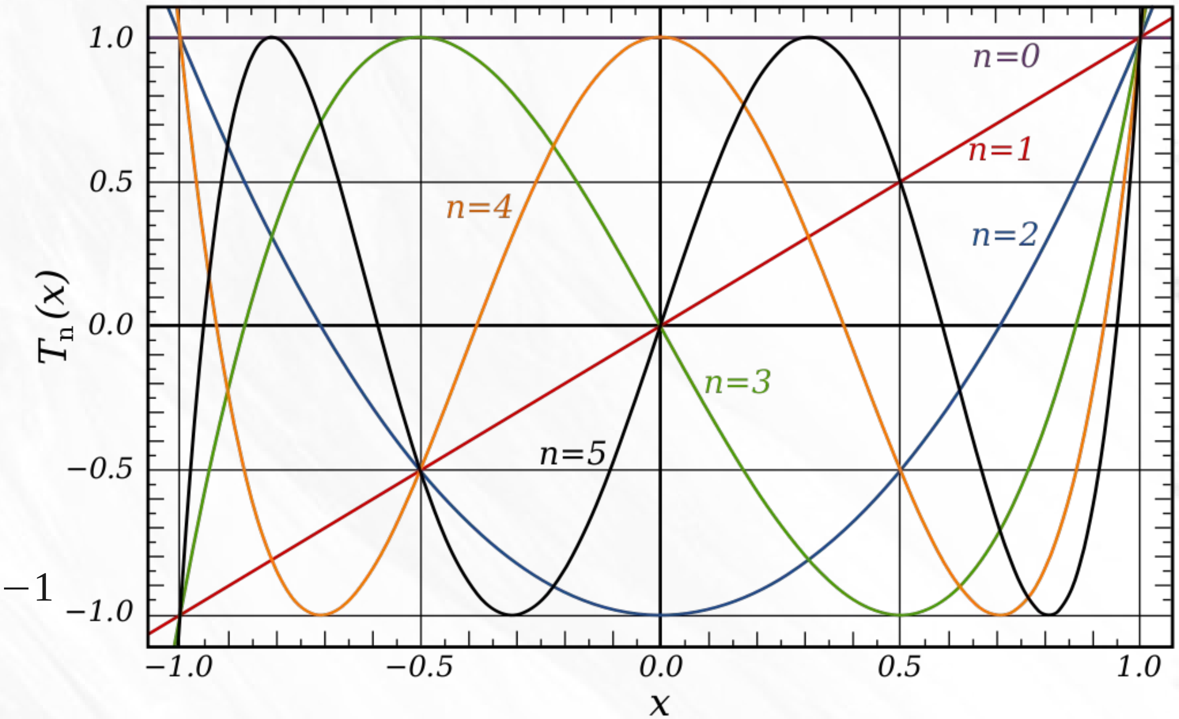
$$T_0(x)=1$$

$$T_1(x)=x$$

$$T_2(x)=2x^2-1$$

$$T_3(x)=4x^3-3x$$

$$T_{n+1}(x)=2x\,T_n-T_{n-1}$$




- They have the nice property that the are bounded by +1 and -1. Therefore, it's easy to estimate the error in expansions using Chebyshev polynomials.

# Cosine formula for Chebyshev polynomials

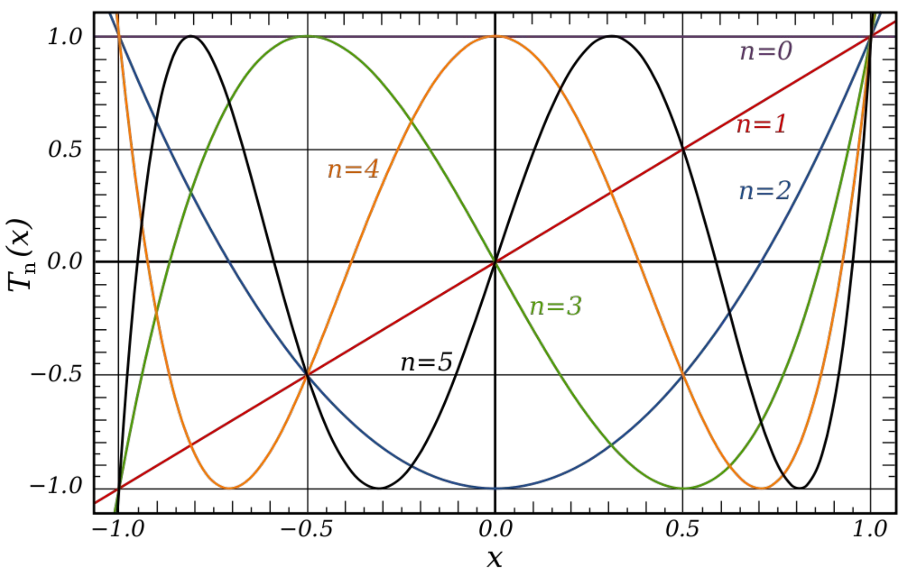
- One can show $T_n(x) = \cos(n \arccos(x))$

$$n = 0 \rightarrow \cos(0) = 1$$

$$n = 1 \rightarrow \cos(\arccos(x)) = x$$

$$n = 2 \rightarrow \cos(2 \arccos(x)) = 2x^2 - 1$$

Shown on blackboard



# Chebyshev polynomial of degree 2

- For n=2,

$$\cos(2\arccos(x)) = \cos(\arccos(x) + \arccos(x))$$

Use identity

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$$

$$= \cos(\arccos(x))\cos(\arccos(x)) - \sin(\arccos(x))\sin(\arccos(x))$$

$$= \cos^2(\arccos(x)) - \sin^2(\arccos(x))$$

$$= x^2 - (1 - \cos^2(\arccos(x)))$$

$$= 2x^2 - 1$$ Q.E.D.

# Chebyshev polynomials form orthogonal set

- Orthogonality:

$$\frac{1}{\pi}\int_{-1}^{1} T_n(x)T_m(x)\frac{dx}{\sqrt{1-x^2}} \quad \begin{aligned} &= 0 \; if \; n \neq m \\ &= 1 \; if \; n = m = 0 \\ &= \frac{1}{2} \; if \; n = m \neq 0 \end{aligned}$$

Valid for continuous case

Note weight

- Therefore, you can do expansion of arbitrary function *f(x)* on interval [-1, 1]:

$$f(x) = \sum_{n=0}^{\infty} a_n T_n(x)$$

$$a_0 = \frac{1}{\pi}\int_{-1}^{1} f(x)T_n(x)\frac{dx}{\sqrt{1-x^2}}$$

$$a_n = \frac{2}{\pi}\int_{-1}^{1} f(x)T_n(x)\frac{dx}{\sqrt{1-x^2}} \quad for \; n = 1, 2, \cdots$$

- These expansion frequently offer faster convergence than e.g. Fourier expansions.

# Optimal sampling points for interpolation: Chebyshev nodes

- These are roots of Chebyshev polynomial $T_{N+1}(x)$

$$x_i = \cos\left(\frac{2i+1}{2N+2}\pi\right) \qquad 0 \le i \le N$$

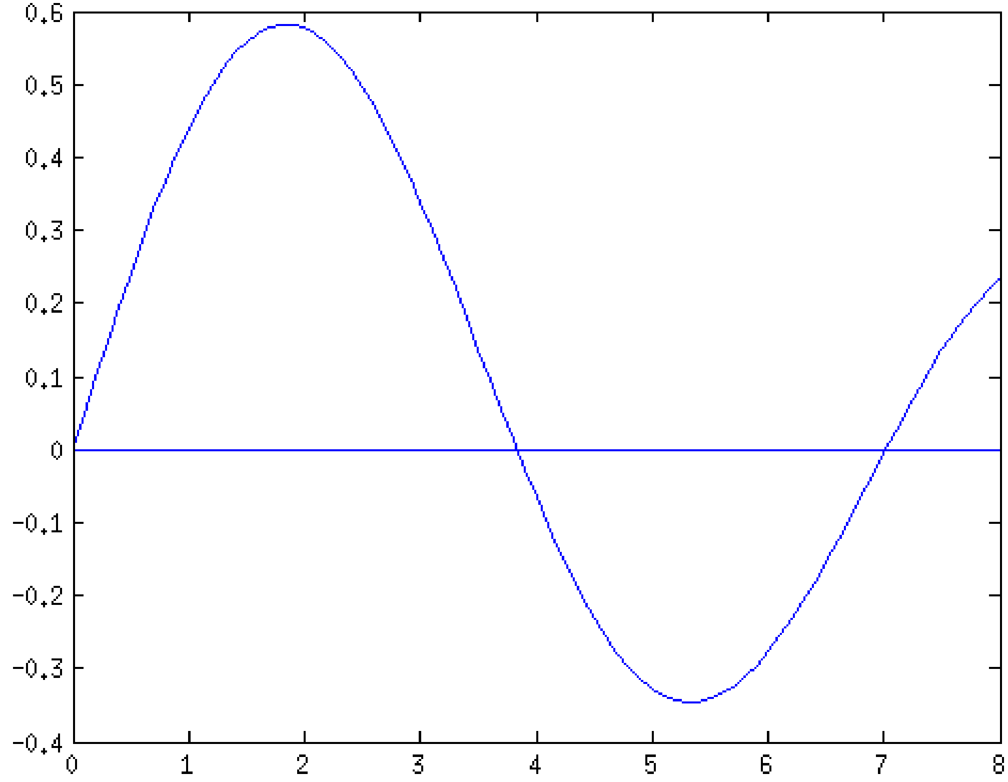
| **Order** | **Zeros** $x_i$ |
|---|---|
| 2 | -0.7071, 0.7071 |
| 3 | -0.8600, 0.0000, 0.8600 |
| 4 | -0.9239, -0.3827, 0.3827, 0.9239 |
| 5 | -0.9511, -0.5978, 0.0000, 0.5878, 0.9511 |

- If you function lives on interval [a, b], you must shift and shrink nodes from interval [-1, 1]:

$$z_i = \frac{1}{2}(b+a) + \frac{1}{2}(b-a)x_i$$

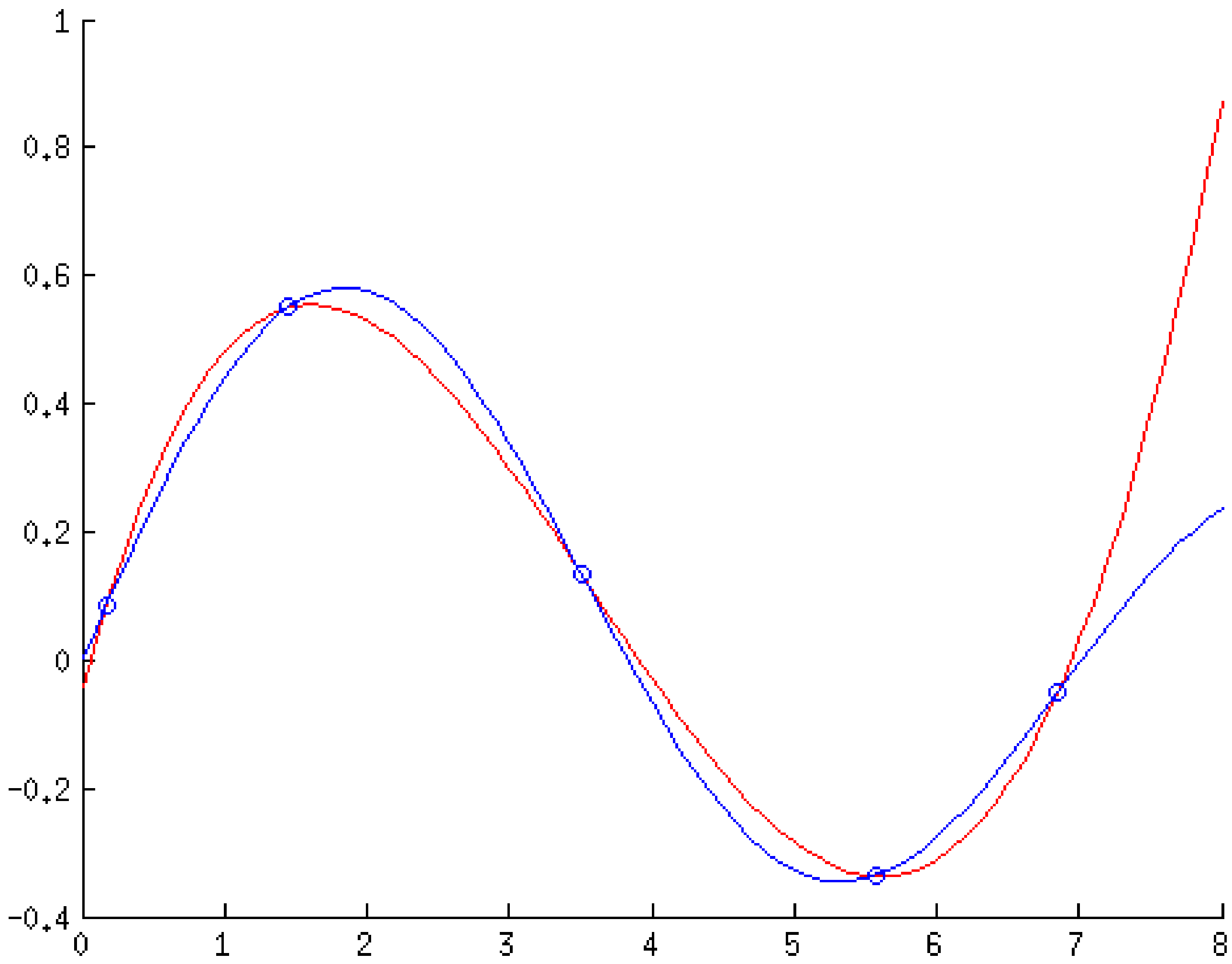
# Interpolating $J_1(x)$

- Interpolate besselj(1, x)
- Domain [0, 7.0155]
- Use Chebyshev points to sample function.



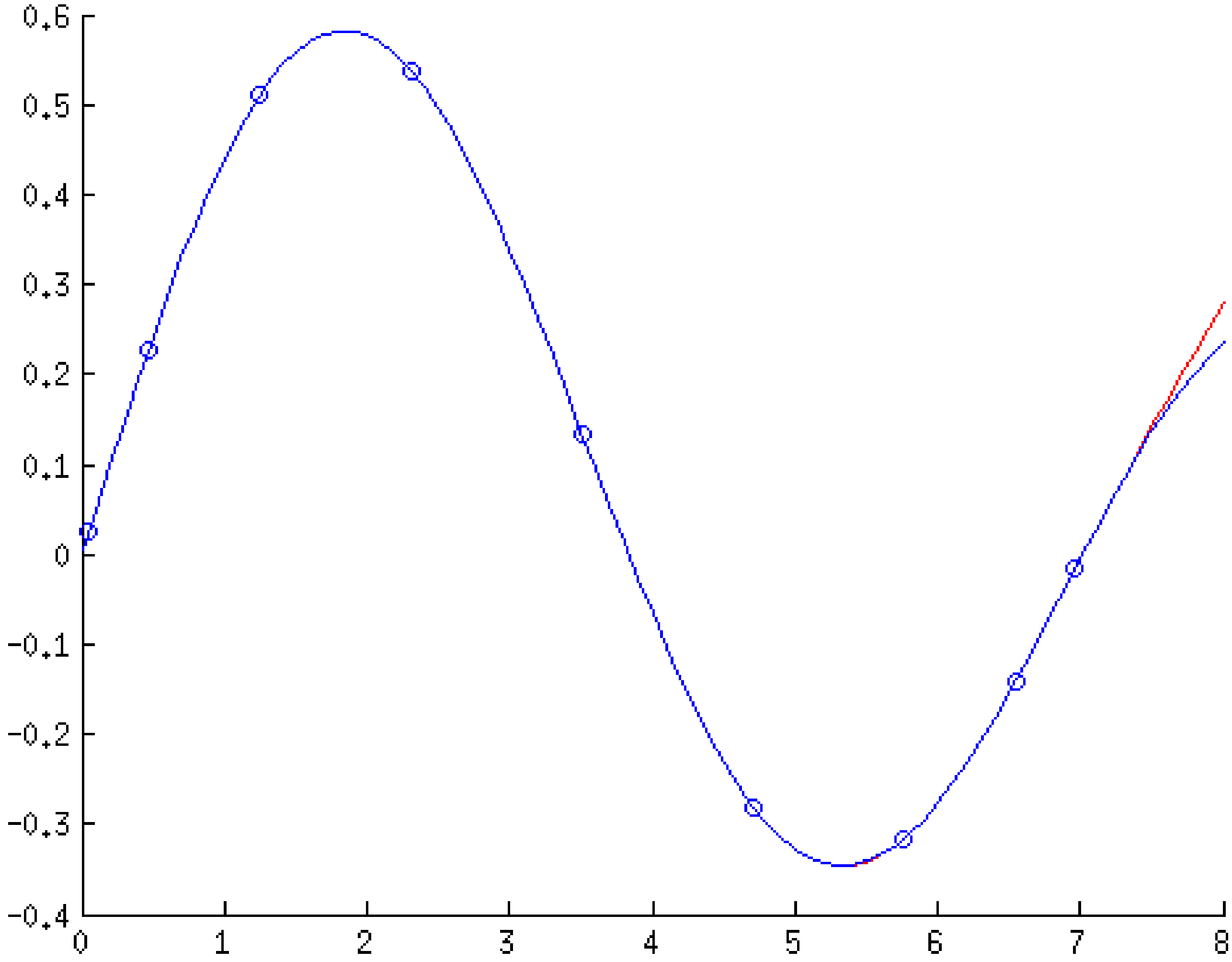

- Use Lagrange interpolation formula
- Derivations on blackboard

# 5 Point Interpolation



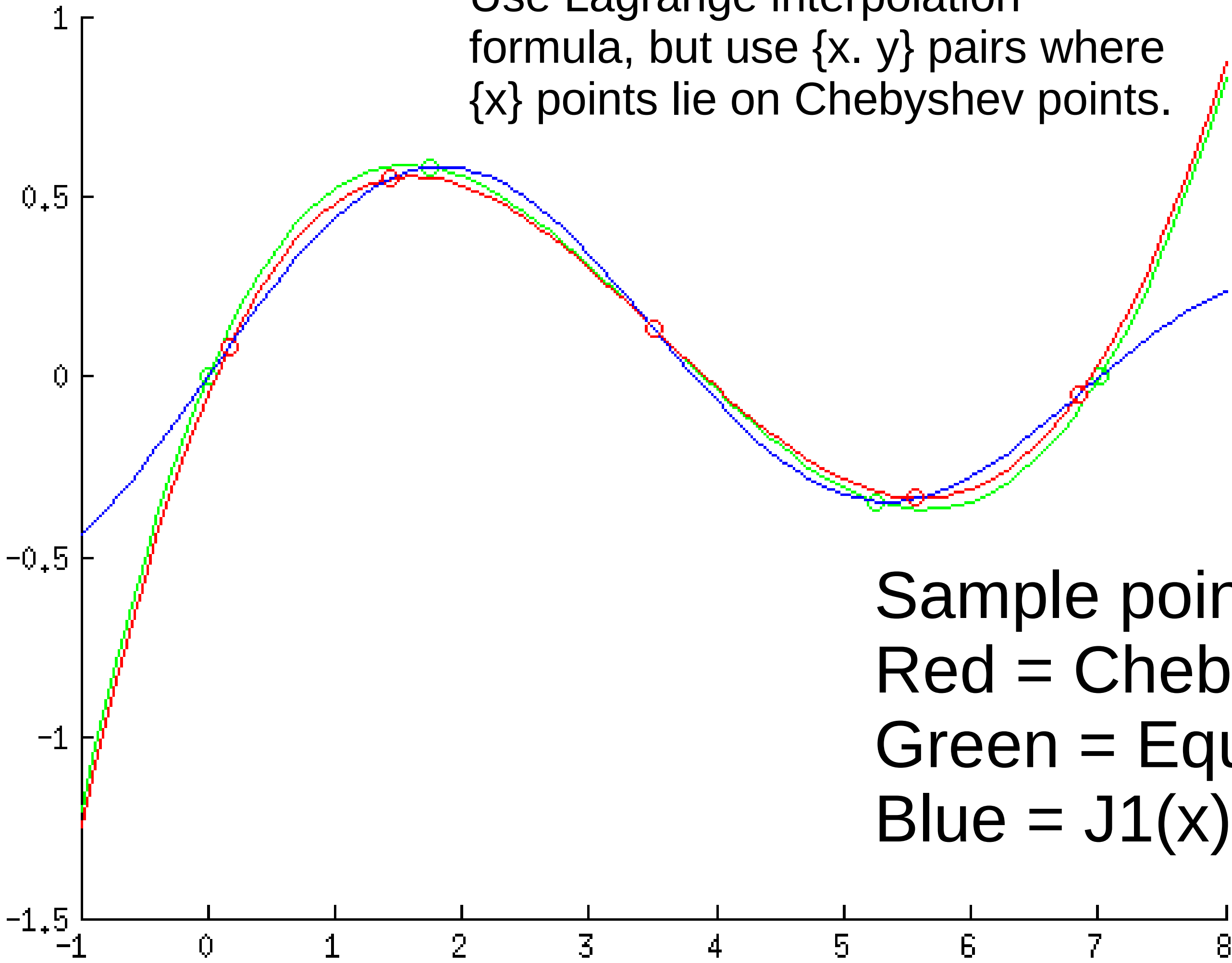

/home/sdb/Northeastern/Class8/Chebyshev/plot_besselj1_interpolation.m

# 9 Point Interpolation



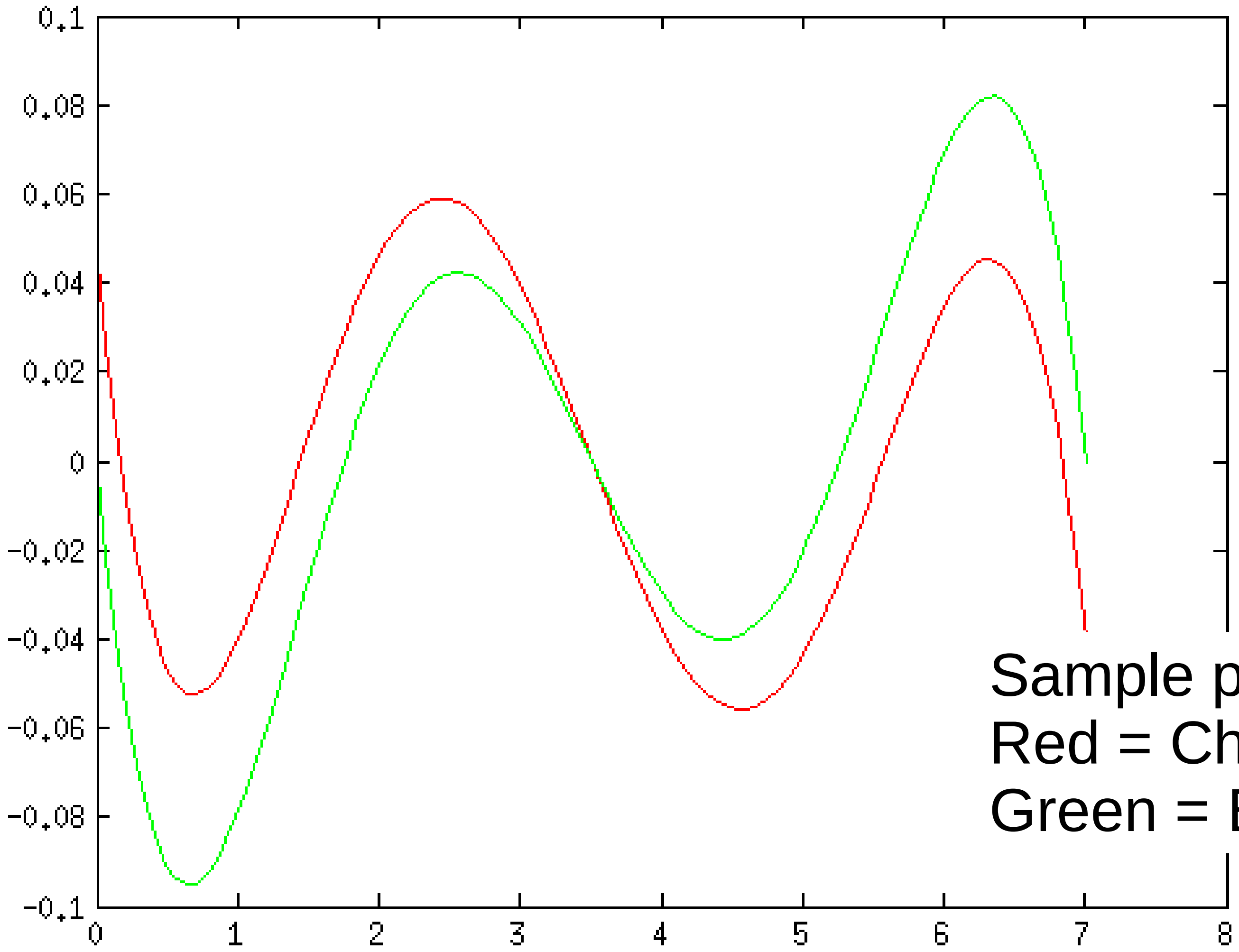
# Chebyshev vs. Equidistant

Important point:
Use Lagrange interpolation formula, but use {x. y} pairs where {x} points lie on Chebyshev points.




Sample points
Red = Chebyshev
Green = Equidistant
Blue = J1(x)

# Error: Chebyshev vs. Equidistant

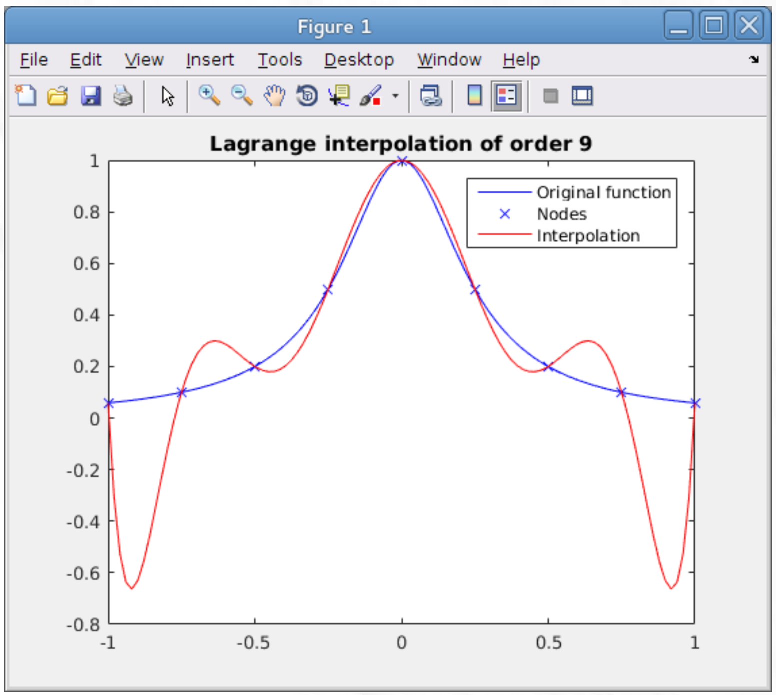


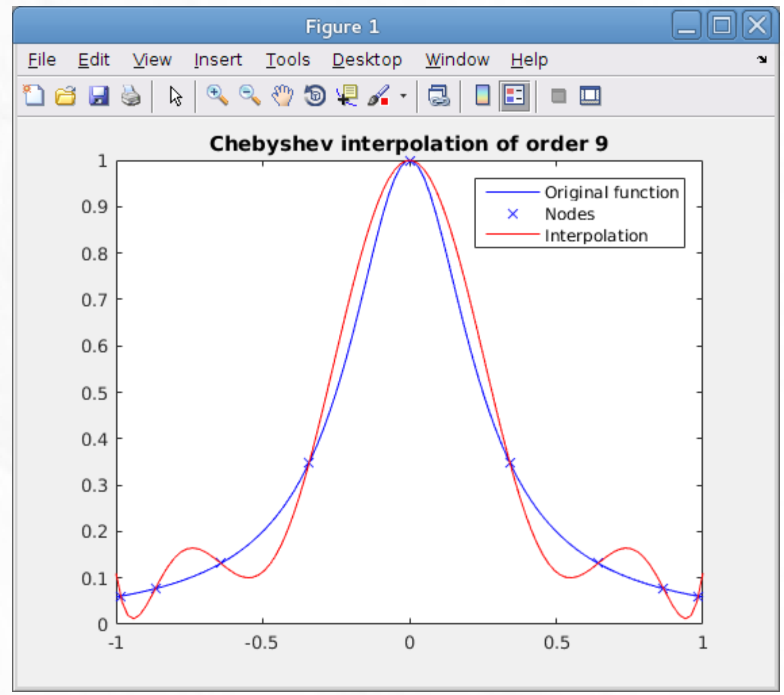

Sample points
Red = Chebyshev
Green = Equidistant

Chebyshev interpolation is better behaved at ends of domain

# Another example

- Interpolate $\dfrac{1}{1+16x^2}$




- Chebyshev interpolation handles ends of domain better.

/home/sbrorson/Northeastern1_Spring2017/Class9/ChebyshevInterpolation

# Major points from lecture

- 1D interpolation
  - Linear interpolation
  - Polynomial interpolation
  - Interpolation using Lagrange polynomial
  - Lagrange Barycentric formula
  - Chebyshev polynomials
  - Chebyshev interpolation: choose x = Chebyshev points and use Lagrange polynomial. (Works only if you can choose the interpolation points.)