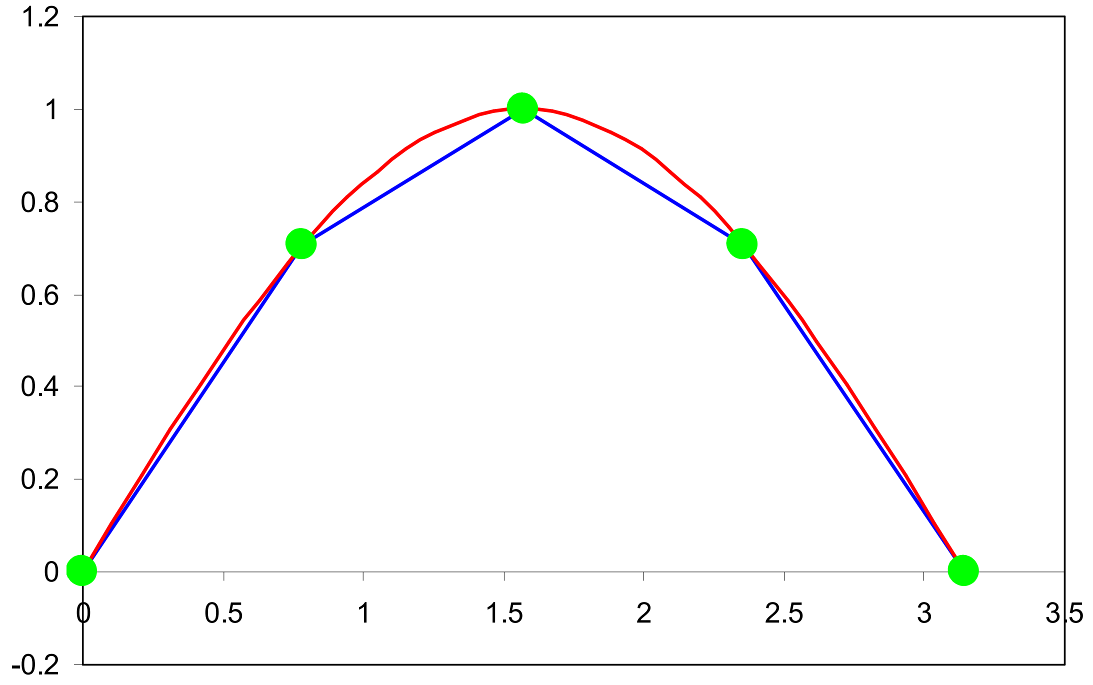
Interpolation

Fabio Cannizzo

Interpolation and Extrapolation

- We sometime know the value of a function f(x) at a set of point x_1, x_2, …, x_n, but we do not have an analytic expression for the function
  - For example, $f(x_i)$ may come from:
    - physical experiments
    - long and complicate numerical calculation that cannot be casted in simple functional form
  - x_i are often equally spaced, but that is not necessary
  - sometime we can choose the x_i, sometimes these are given
- We want to estimate f(x) for arbitrary x by, in some sense, drawing a smooth curve through the x_i such that **$f(x_i)=y_i$** (**interpolation**), and perhaps beyond (**extrapolation**) the x_i

Example of Interpolation



Two possible interpolation functions f(x) for the set of points in green

Functional Form

- Interpolation must model the data set via some plausible functional form
- This form should be able to approximate a large classes of functions which might arise in practice. Common choices are:
  - polynomials
  - rational (quotient of polynomials)
  - trigonometric

Interpolation

- Given a set of points (x_i, y_i), $i=1\dots N$, and chosen **a suitable parametric functional shape** for my interpolating function $f(x)$, we fix the parameters so that $y_i=f(x_i)$ for all given points
- Example:
  - Points: { (0,0), (1,1) }
  - Parametric Functional Shape: $y = m x + q$
  - Solution: $m=1$, $q=0$. We can verify it pass for both given points

Polynomial Interpolation

- The basis function we use to interpolate the set of N+1 points is a polynomial of degree N

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

- Theorem

  let x_0, x_1, … x_N, be distinct real values

  let y_0, y_1, … y_N, be the corresponding real values (not necessarily distinct)

  there exist one and only one polynomial of degree at most n such that $y_i=p(x_i)$ for i=0…N

Polynomial Interpolation

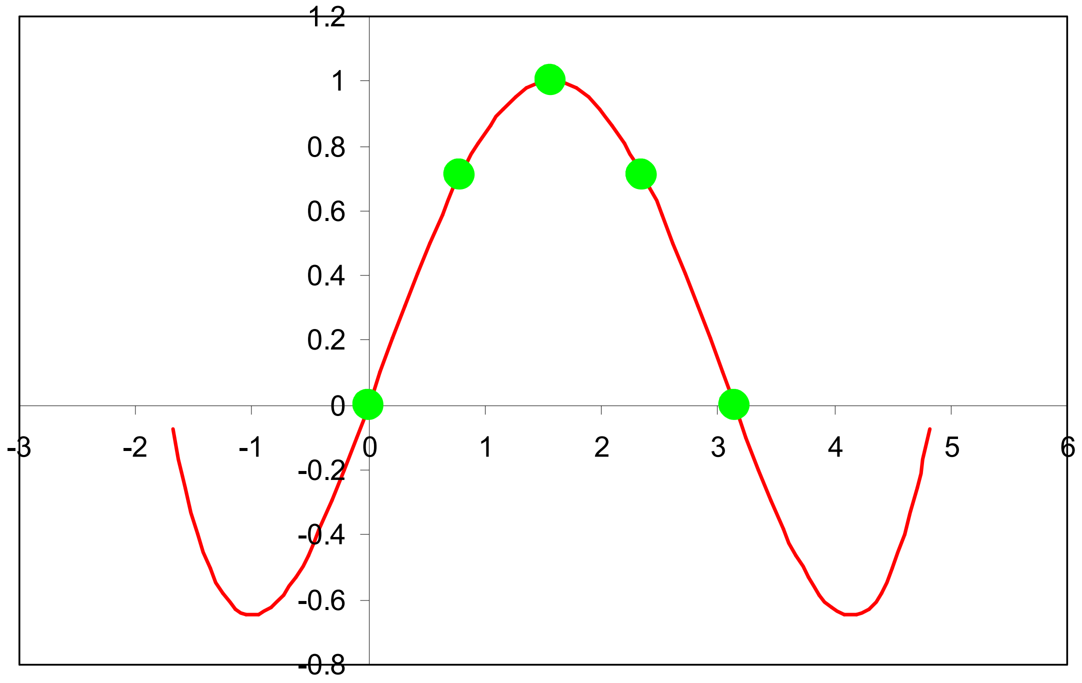
- To compute the coefficients a_i we could solve a linear system, imposing passage for all points

$$\begin{cases} y_0 = a_0 + a_1 x_0 + a_2 x_0{}^2 + \ldots + a_n x_0{}^n \\ y_1 = a_0 + a_1 x_1 + a_2 x_1{}^2 + \ldots + a_n x_1{}^n \\ \vdots \\ y_{n+1} = a_0 + a_1 x_n + a_2 x_n{}^2 + \ldots + a_n x_n{}^n \end{cases} \Rightarrow \underbrace{\begin{bmatrix} 1 & x_0 & \cdots & x_0{}^n \\ 1 & x_1 & \cdots & x_1{}^n \\ \vdots & & \ddots & \vdots \\ 1 & x_n & \cdots & x_n{}^n \end{bmatrix}}_{\text{VANDERMONDE MATRIX}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Polynomial Coefficients

- Since the N+1 equations are linearly independent, the Vandemonde system has always a unique solution
- Solution of a linear system is expensive (about n^3 operations)
- There is an easier way …

Example of Interpolation



Polynomial interpolation and extrapolation

Extrapolation is dangerous!

Math Review

$$\|g(x)\|_1 = \int_a^b w(x)|g(x)|dx$$ norm - 1

$$\|g(x)\|_2 = \sqrt{\int_a^b w(x)g(x)^2 dx}$$ norm - 2 (Euclidean)

$$\|g(x)\|_p = \sqrt[p]{\int_a^b w(x)|g(x)|^p dx}$$ norm - p (generale case)

$$\|g(x)\|_\infty = \max_{[a,b]}|g(x)|$$ norm - ∞ (infinity or maximum)

The function w(x) is a weitgh function which can be used if accuracy in certain regions is more important than other regions. Normally w(x)≡1

Polynomial Approximation

- Weierstrass Theorem

  if f(x) is infinitely smooth in [a,b], for any ε>0 there exist *n* such that the polynomial approximation of degree *n* satisfies

  $$\left\| f(x) - p_n(x) \right\|_\infty < \varepsilon$$

- By growing the order of the polynomial, we can expect the maximum error in absolute value to become smaller and smaller
- There is a catch: high order polynomial may have undesired oscillations

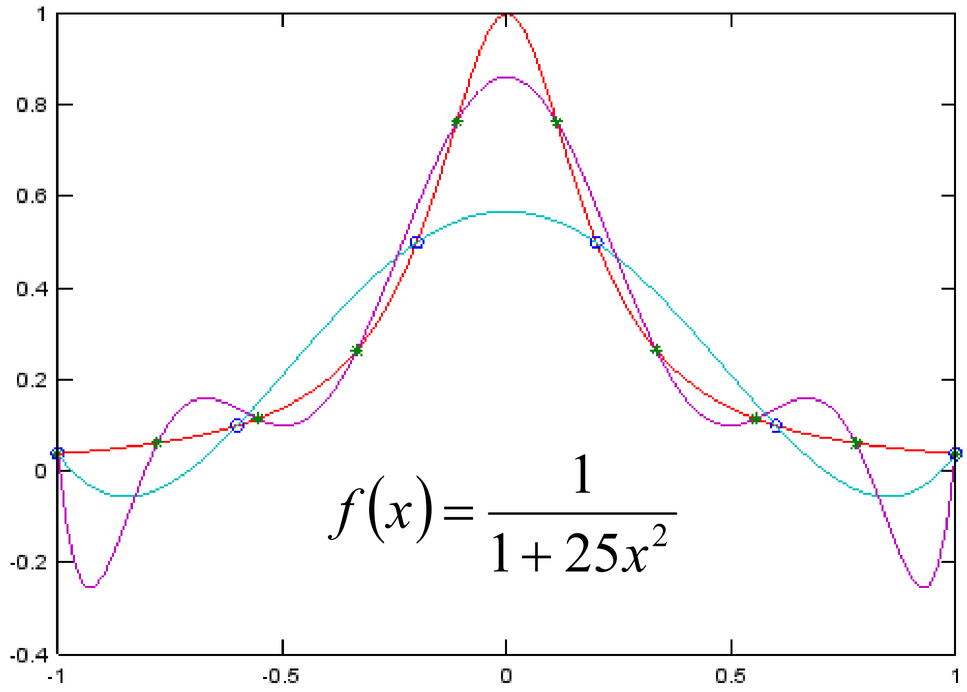
Runge’s Phenomenon

We want to approximate f(x) in [1-,1]

Equally spaced points would seem a good choice, but…

Runge's phenomenon is a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree

In the picture, polynomials of degree 5 and 9



```
>> function y=runge(x), y=1./(1+25*x.*x); endfunction
>> xi6=[-1:0.4:1]';yi6=runge(xi6); xi10=[-1:2/9:1]';yi10=runge(xi10);
>> vdm5=ones(6,6);  for i=5:-1:1, vdm5(:,i)=vdm5(:,i+1).*xi6;  endfor
>> vdm9=ones(10,10);for i=9:-1:1, vdm9(:,i)=vdm9(:,i+1).*xi10; endfor
>> p5=vdm5\yi6; p9=vdm9\x10;
>> x=[-1:0.02:1]; y=runge(x);
>> plot(xi6,yi6,"o",xi10,yi10,"*",x,y,x,polyval(p5,x),x,polyval(p9,x))
```

Error in Polynomial Approx

When a smooth function *f*(*x*) is approximated with a polynomial *p*(*x*) of degree *n*, the error can be expressed analytically:

$$e(x) = f(x) - p_n(x) = \frac{\pi(x)}{(n+1)!} f^{(n+1)}(\xi), \quad \xi \in [a,b]$$

where $\pi(s) = (x - x_0)(x - x_1)\ldots(x - x_n)$

A bound of the error is :

$$E(x) = \frac{|\pi(x)|}{(n+1)!} \left\| f^{(n+1)}(t) \right\|_\infty, \quad t \in [a,b]$$

Choice of the Points

- A natural choice for x_i is to have them equally spaced in [a,b]
- Is there a better choice?
- Can we minimize the error bound?
- The only term in the error bound formula that we can hope to reduce is $\pi(x)$, by changing the choice of the points x_i

Error in Polynomial Approximation

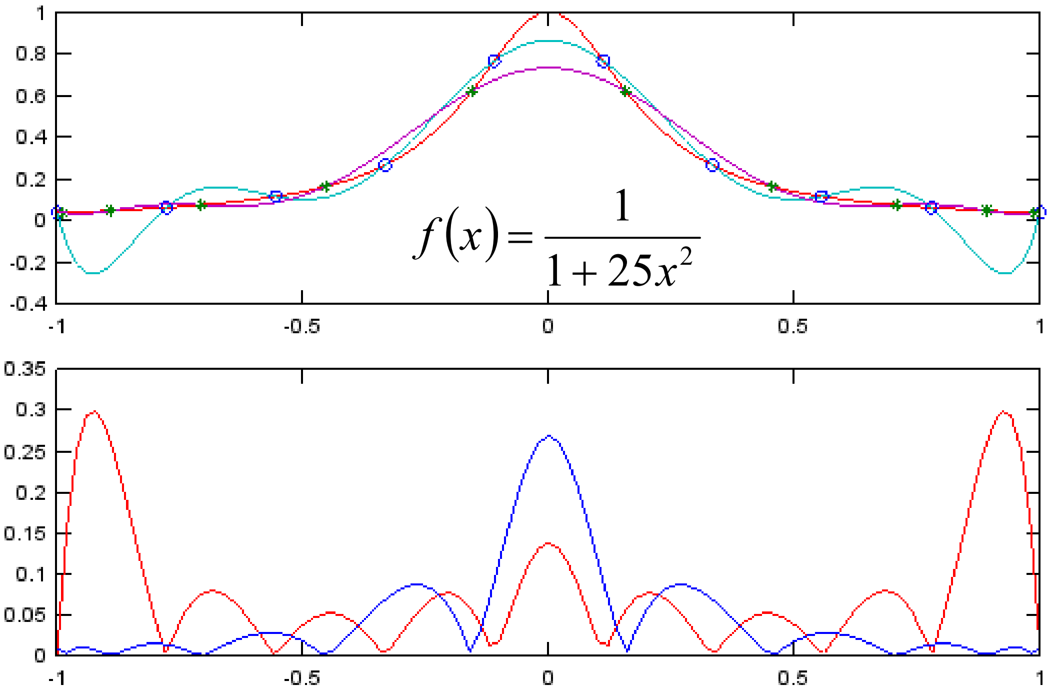
Equally spaced points

$$x_i = -1 + \frac{2}{n-1}(i-1), \quad i = 1 \ldots n$$

Chebyshev points

$$x_i = \cos\left(\pi - \frac{2i-1}{2n}\pi\right), \quad i = 1 \ldots n$$

- less oscillations
- smaller largest error



```
>> function y=runge(x), y=1./(1+25*x.*x); endfunction
>> n=10; j=[1:n]'; xe=-1+2/(n-1)*(j-1); xc=cos(pi()-(2*j-1)/(2*n)*pi());
>> ye=runge(xe); yc=runge(xc);
>> pe= vander(xe)\ye; pc=vander(xc)\yc;
>> x=[-1:0.02:1]; y=runge(x); yee=polyval(pe,x); ycc=polyval(pc,x);
>> subplot(2,1,1); plot(xe,ye,"o",xc,yc,"*",x,y,x,yee,x,ycc)
>> subplot(2,1,2); plot(x,abs(y-yee),”r”,x,abs(y-ycc),”b”)
```

Chebishev Polynomials

- The points used in the previous page are the roots of Chebyshev polynomials of degree n-1
- Chebyshev polynomials are a special family of polynomials defined recursively as

  $T_{n+1}(x)=2x\,T_n(x) - T_{n-1}(x)$, with $T_0=1$ and $T_1=x$
- These points have the property to minimize the approximation error in infinite norm, and to reduce oscillation, i.e. they minimize $\pi(x)$ in infinite norm (we won't prove that)

Chebishev Polynomial

- Roots of Chebyshev polynomial are all in the interval [-1,+1]
- If we need to approximate f(x) in a generic interval [a,b], we choose a linear change of variable which transforms [a,b] in [-1,+1]
- This is the line passing for (a,-1) and (b,+1), i.e.:

  $z=-1+2/(b-a)\ (x-a)$
- So, approximating f(x) in [a,b], is the same as approximating f(x(z))=g(z) in [-1,1]
- Therefore we can simply take the n Chebishev points z_i in [-1,1] and anti-transform them into the equivalent points x_i in [a,b] using the change of variable above

Polynomial Interpolation Summary

- The choice of interpolation points is important to reduce the error
- Although Chebishev reduce oscillation, polynomial of high orders have wild oscillations in between points, especially in the wings of the region
- In practice, we usually do not use high order polynomial interpolation when we have an interpolation problem
- So why is it important?
  - Polynomial interpolation is at the very heart of many other numerical methods: finite differences, quadrature, ODEs, root search, PDEs, …

Splines

- A very common interpolation techniques is to use ***piecewise*** interpolation functions
- Given *n*+1 points, and one interpolation function g(x : θ), which depends on *m* parameters θ, we construct an interpolator G(x) as a sequence of *n* interpolators g(x : θ_i), one in each interval [x_{i-1},x_i]

$$G(x) = \sum_{i=1}^{n} g(x, \theta_i)\, I_{x \in [x_i, x_{i+1})}$$

where I_a is the indicator function, defined as: $I_a = \begin{cases} 1, & \text{if } a \text{ is true} \\ 0, & \text{otherwise} \end{cases}$

Number of Free Parameters

- The interpolator G(x) has m*n degrees of freedom, which we can use to fit the points and satisfy other properties (e.g. smoothness of the interpolating function)
- What other properties we decide to satisfy is arbitrary, but there are some common choices
- We will only treat cases where g(x) is a polynomial

Piecewise Constant Interpolation

- The simplest possible type of spline use as basis function a polynomial of degree zero: $y_j = a_{0,j}$
- There is 1 free parameter per bucket, i.e. n degrees of freedom in total
- We can define it in many ways, for example:

$$G(x) = y_i \; I_{x=x_n} + \sum_{i=0}^{n-1} y_i \; I_{x \in [x_i, x_{i+1})}$$

Continuous from the right

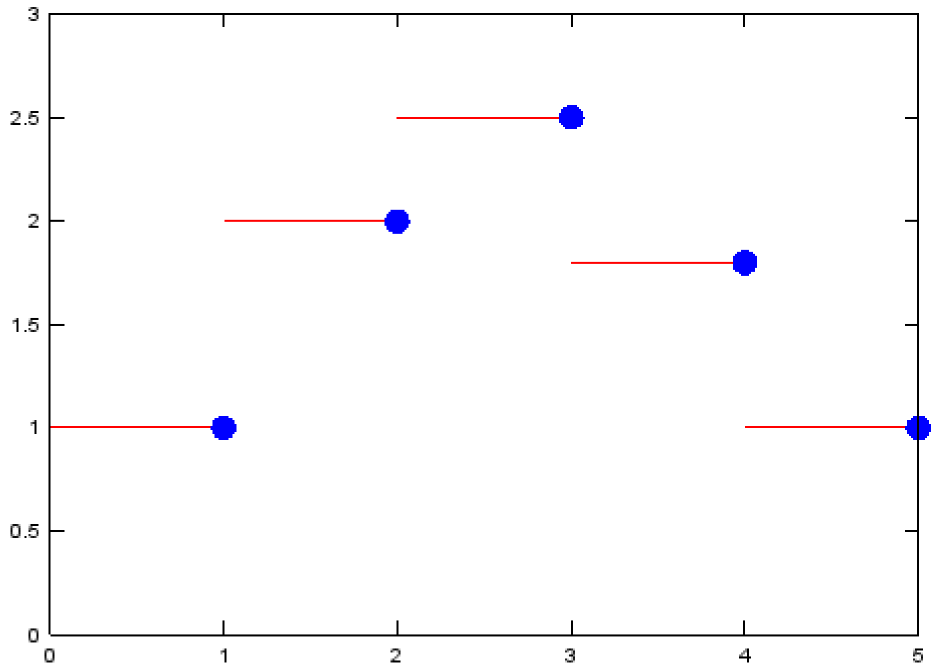
or

$$G(x) = y_0 \; I_{x=x_0} + \sum_{i=0}^{n-1} y_{i+1} \; I_{x \in (x_i, x_{i+1}]}$$

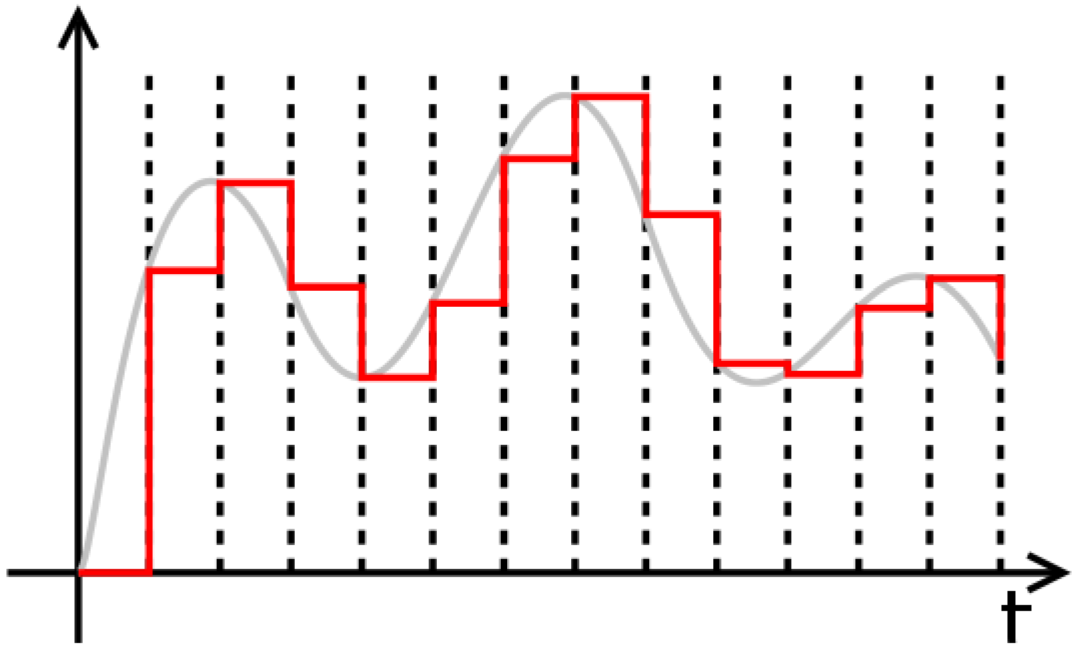
Continuous from the left

- Need to define the extrapolation behavior: usually the standalone point is not used
- The second formulation is used very frequently in finance, in the bootstrapping of interest rates or implied volatilities.

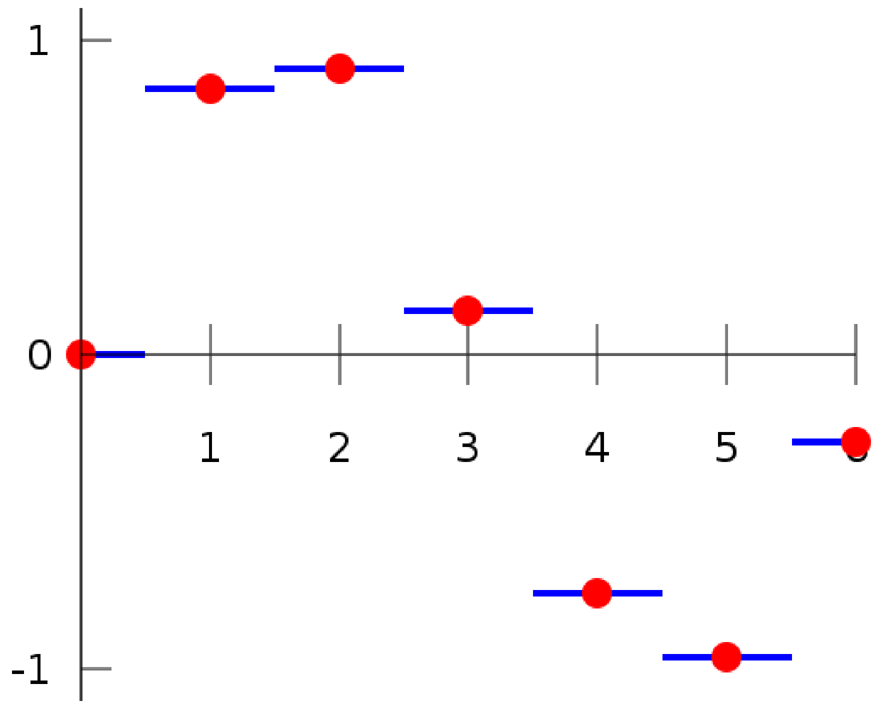
Piecewise Constant Interpolation



Piecewise Constant Interpolation



Piecewise Constant Interpolation



Searching for the Right Bucket

- After defining G(x), we want to use it for interpolating at generic points x
- We need to find which is the correct $g_i(x)$ to use, i.e. we want to know which is the pair of points x_i, x_{i+1}, such that $x_i \leq x < x_{i+1}$
- We can scan the points in order (**linear search**)
- Since the points x_i are sorted, we can use **binary search**, i.e. we proceed by bisecting the total interval, until we are left with an interval bounded by 2 contiguous points
- Binary search completes in $O(\log_2 n)$, as opposed to linear search which completes in O(n)

Piecewise Linear Interpolation

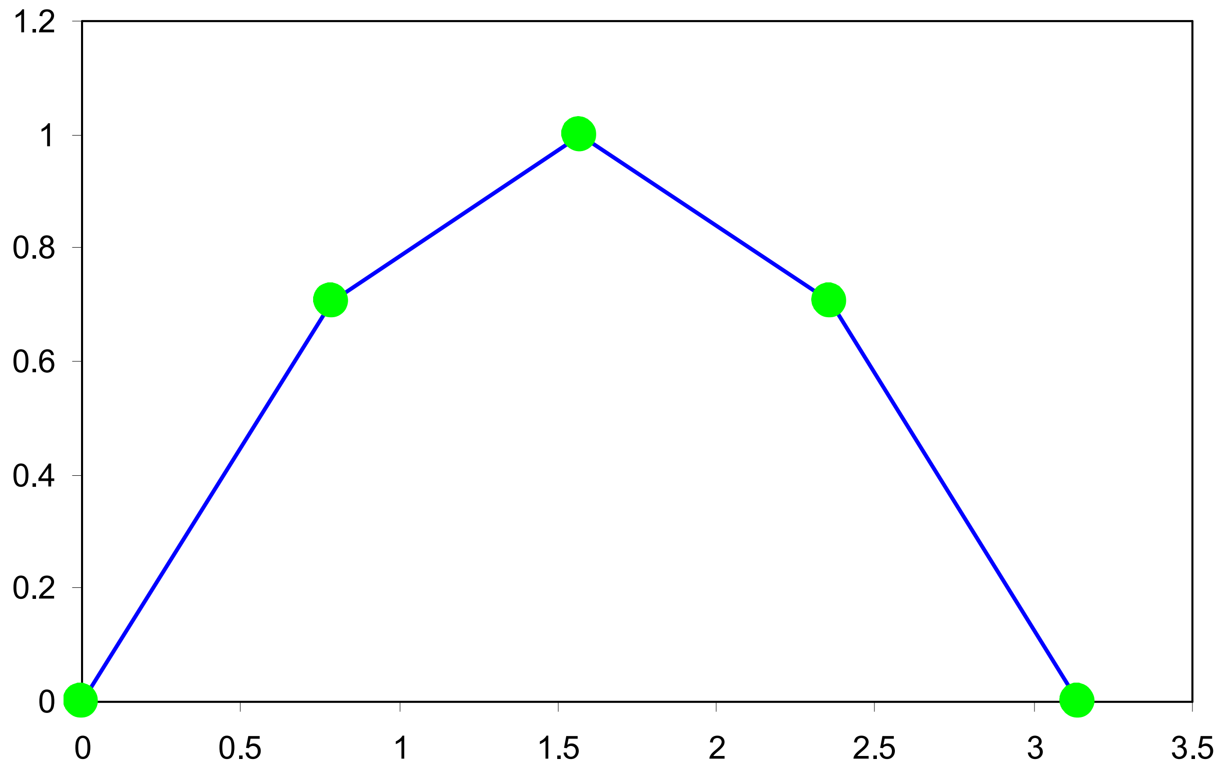
- Another commonly used basis are polynomials of degree 1: $y=a_0+a_1x$
- There are 2 free parameters per interval, i.e. 2n degrees of freedom in total
- Imposing that every polynomial pass through the two points delimiting the correspondent interval, the spline is completely defined:

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

Straight line passing through the points: (x_i,y_i), (x_{i+1}, y_{i+1})

$$G(x) = \sum_{i=0}^{n-1} g_i(x) I_{x\in[x_i,x_{i+1})}$$

Piecewise Linear Interpolation



Piecewise Cubic Interpolation

- Another commonly used basis are polynomials of degree 3: $y=a_0+a_1x+a_2x^2+a_3x^3$
- There are 4 free parameters per interval, i.e. 4n degrees of freedom in total
- Imposing that every polynomial pass through the two points delimiting the correspondent interval fixes 2n conditions
- To fill the other conditions we could impose that the 1st derivative and the 2nd derivative are continuous at all internal knots. I.e. $g'_i(x_{i+1})=g'_{i+1}(x_{i+1})$ and $g''_i(x_{i+1})=g''_{i+1}(x_{i+1})$ for $i = 1…n-1$
- This defines 2(n-1) equations, therefore we are still left with 2 degrees of freedom

Piecewise Cubic Interpolation

- The most common way to fix the 2 residual degree of freedom is to require that the second derivative at the extreme points of the interval is null, i.e. the function is linear (**natural boundaries)**
- An alternative is to specify the 1st derivative at the extreme points of the interval (**clamped boundaries)**
- Or we could use a combination of the above
- Or any other condition which might be better suited to your problem

Piecewise Cubic Interpolation

- In general, we need to specify 4n linearly independent equations.
  - If we do not need the derivatives to be continuous, we can use also these degrees of freedom to achieve “something else”
  - If we do not need the spline to strictly pass by the points, we could replace also these conditions for “something else”
  - One variation are Bezier curves (C^1), often used in graphical applications for charts (e.g. Excel)

Cubic Splines

- "Cubic splines" very often refers to a C^2 interpolant with natural condition at the boundaries
- There is a fast construction procedure to obtain the coefficients, both for natural and clamped boundaries. I.e., the linear system with 4n equations and 4n unknowns can be simplified a lot, allowing for very quick solution

Cubic Splines Construction

Given the points $(x_i, y_i)_{i=1,2,\dots,n}$

linearpolation between x_i and x_{i+1} gives

$l_i(x) = A_i y_i + B_i y_{i+1}, \quad i = 1 \dots n-1$

where

$$A_i = \frac{x_{i+1} - x}{h_i}, \quad B_i = 1 - A_i = \frac{x - x_i}{h_i}, \quad h_i = x_{i+1} - x_i$$

Suppose we also know the 2nd derivative at every x_i we can add to $l_i(x)$ a cubic polynomial such that the 2nd derivative change linearly between x_i and x_{i+1}. This polynomial should also have zero values at the x_i

$g_i(x) = A_i y_i + B_i y_{i+1} + C_i y_i^{(2)} + D_i y_{i+1}^{(2)}$

where

$$C_i = \frac{1}{6}\left(A_i^3 - A_i\right) h_i^2, \quad D_i = \frac{1}{6}\left(B_i^3 - B_i\right) h_i^2$$

Cubic Splines Construction

We still have not imposed continuity of 1st derivative

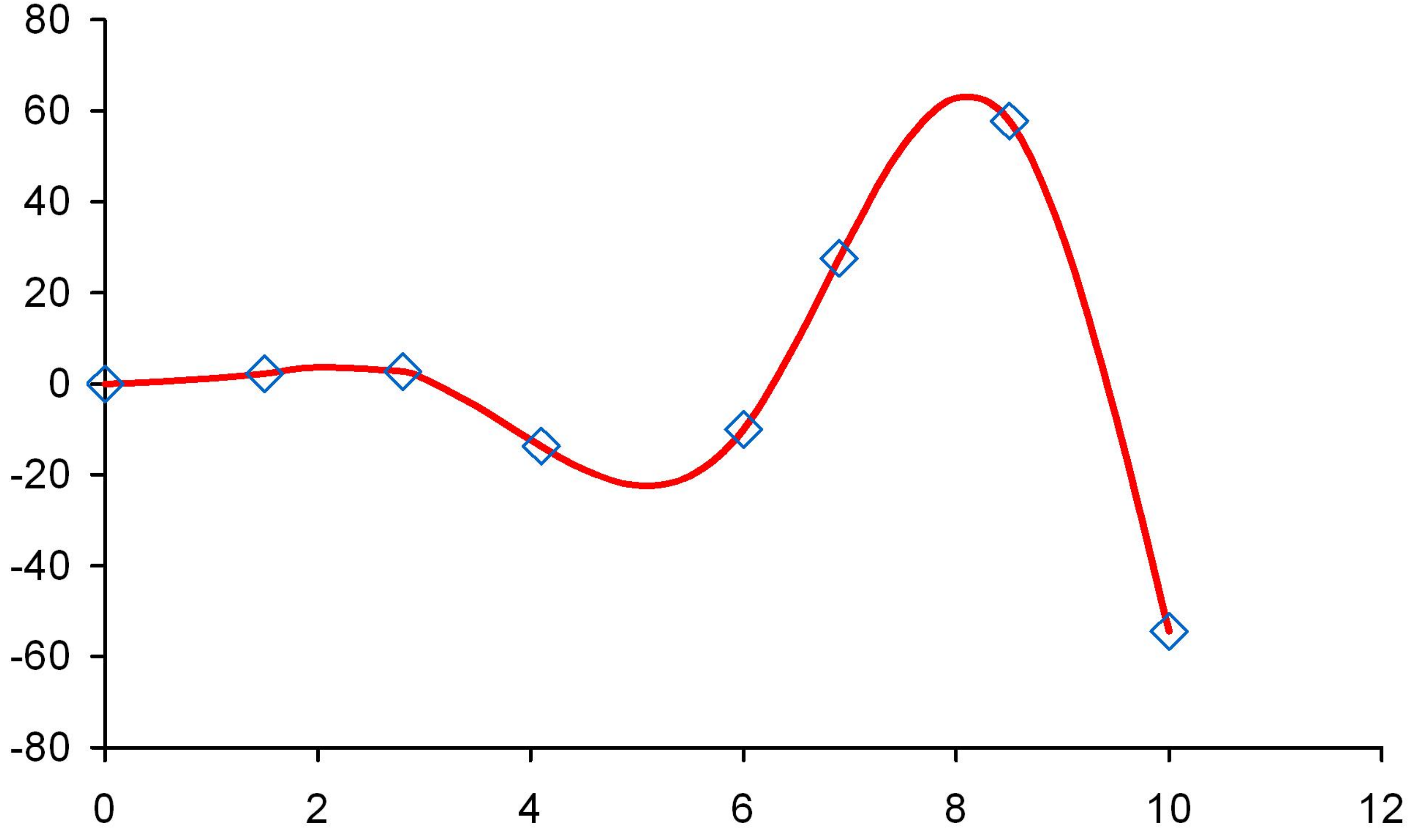
$$\frac{dg_i(x)}{dx} = \frac{y_{i+1} - y_i}{h_i} + \frac{1}{6}\left(-3A_i^2 + 1\right) h_i y_i^{(2)} + \frac{1}{6}\left(3B_i^2 - 1\right) h_i y_{i+1}^{(2)}$$

$$\frac{dg_{i-1}(x_i)}{dx} = \frac{dg_i(x_i)}{dx}$$

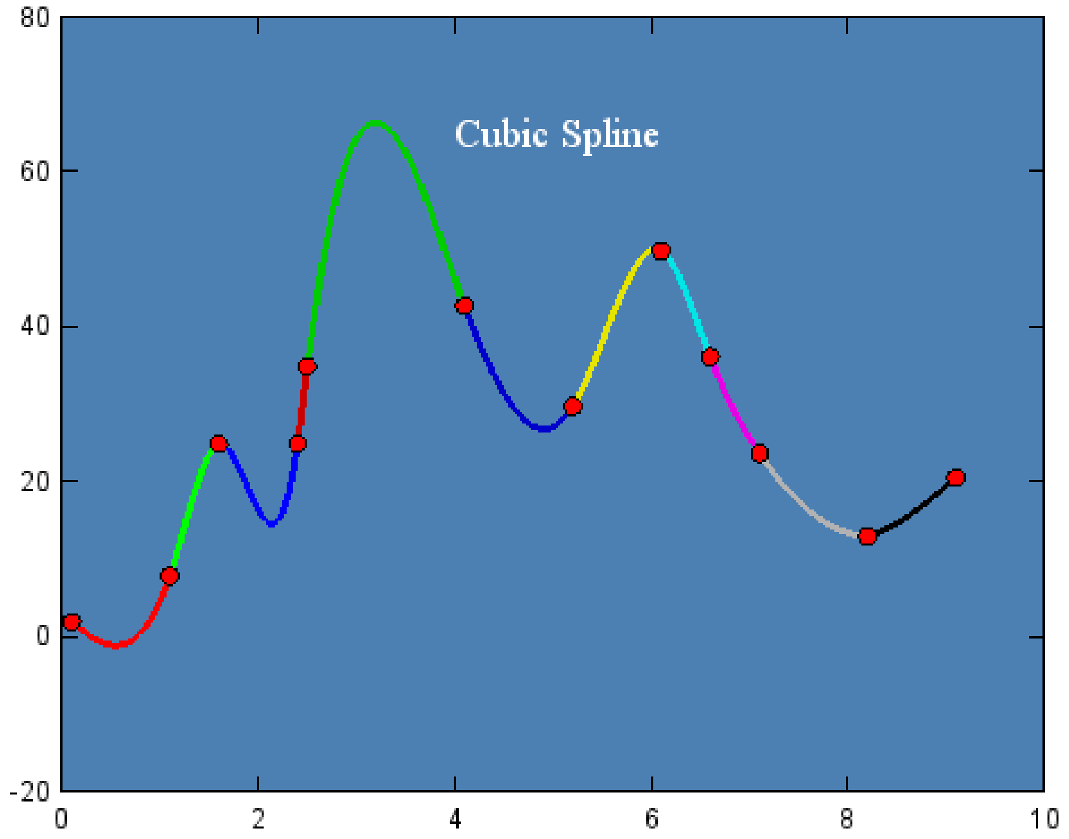
$$\frac{h_{i-1}}{6} y_{i-1}^{(2)} + \frac{1}{3}\left(h_{i-1} + h_i\right) y_i^{(2)} + \frac{h_i}{6} y_{i+1}^{(2)} = \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}, \quad i = 2 \ldots n\text{-}1$$

- we have *n*-2 equations in *n* unknowns (the *n* 2nd derivatives)
- using the natural (or clamped) boundary conditions we provide with the missing equations
- note the system is tridiagonal, therefore there are very fast direct solution techniques O(n)

Cubic Splines Example



Cubic Spline Example



Note the oscillations (1st blue segment and 2nd green segment)

In general curves with sharp corners are better approximated by low order polynomials, curves very smooth are better approximated by high order polynomials

Octave graphical capabilities are quite cool!!! (see *ColoredSpline.m*)

Cubic Splines Properties

- For any g(x) interpolating the points (x_i,y_i), the natural cubic spline is the C^2 function which minimize the integral of the 2nd derivative squared, i.e. is the one which oscillates less (esthetically nicer)
- Given a function f(x) and any g(x) interpolating the points (x_i,y_i), the clamped cubic spline is the C^2 function which minimize the integral of the square of the error between the two 2nd derivatives
- Maximum interpolation error tend to zero by increasing the number of points

Dangers

- Generally high order is good if the function is very smooth
- If the function is C^n, then we should not try to interpolate with order higher than *n*
- In the example we saw in the context of finite differences, the function is C^∞ everywhere except in x=2, where it is just C^1, and we were trying to estimate the derivative in x=2 using finite differences with an approximation of type C^2
- Some functions are very difficult to interpolate and will make a mock of any interpolation scheme
- In finance is very common to deal with functions which are discontinuous or only C^0.

Further Readings

- Prof Binegar’s lecture notes
  http://www.math.okstate.edu/~binegar/4513-F98/4513-l15.pdf
  http://www.math.okstate.edu/~binegar/4513-F98/4513-l16.pdf
  http://www.math.okstate.edu/~binegar/4513-F98/4513-l17.pdf
- “*Numerical Recipes in C++*”
- Prof Rutger’s lecture notes
  http://www.math.rutgers.edu/~falk/math573/lecture5.pdf

Exercise

- Chebishev vs equally spaced vs spline
- Alogorithm for computing spline coefficients and for binary search

Approximation

- It is a similar problem to interpolation: we want to find a simple function to use instead of a more complicate one
- How is this different from before?
  - We can compute $f(x_i)$, therefore we can chose the points (x_i, y_i) which fits as best
  - After I chose the x_i and compute the $y_i=f(x_i)$, it becomes an interpolation problem
- Taylor expansion is an example of polynomial approximation of a function

Interpolating vs Fitting

- Interpolating spline: it pass for the control points (knots)
- Fitting spline: the control points influence the shape of the spline, but it does not necessarily pass through them

Polynomial in Newton Form

1 Construct a polynomial of order 0 as

$P_0(x)=y_0$

This obviously pass for the 1st point (x_0,y_0)

2 Construct a polynomial of order 1 as

$P_1(x)=P_0(x)+c_1(x-x_0)$

Like P_0, this also pass for the 1st point. For it to pass also for the 2nd point we impose

$y_1=P_0(x_1)+c_1(x_1-x_0)$

so we obtain

$c_1=[y_1-P_0(x_1)]/(x_1-x_0)$

Polynomial in Newton Form

3 Construct a polynomial of order 2 as

$P_2(x) = P_1(x)+c_2(x-x_1)(x-x_0)$

This also pass for the 1st and 2nd point. For it to pass also for the 2nd point we impose:

$y_2 = P_1(x_2)+c_2(x_2-x_1)(x_2-x_0)$

so we obtain

$c_2 = [y_2-P_1(x_2)]/[(x_2-x_1)(x_2-x_0)]$

4 …

Polynomial in Newton Form

- Construct a polynomial of order n as

  $P_n(x) = P_{n-1}(x) + c_n(x-x_{n-1})\ldots(x-x_1)(x-x_0)$

  This also pass for the 1st n points. For it to pass also for the n+1th point we impose:

  $y_n = P_{n-1}(x_n) + c_n(x-x_{n-1})\ldots(x_n-x_1)(x_n-x_0)$

  so we obtain

  $c_n = [y_n-P_{n-1}(x_n)] / [(x_n-x_{n-1})\ldots(x_n-x_1)(x_n-x_0)]$

In compact form:

$$p(x) = c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + \ldots + c_n(x-x_0)(x-x_1)\ldots(x-x_{n-1})$$
$$= \sum_{i=0}^{n} c_i \prod_{j=0}^{i-1} (x-x_j)$$

Polynomial in Lagrange Form

- Remember the coefficients can be obtained by solving a linear system, with the Vandemonde matrix: Va=y ➔ a=V^{-1}y
- The coefficients of the polynomial depend linearly on y_i
- Collecting terms depending on each y_i we could write the polynomial as:

$$p(x) = y_0\, l_0(x) + y_1\, l_1(x) + \ldots + y_n\, l_n(x)$$

where the polynomials $l_0(x)$ depends only on the points x_i

Example

$$\begin{bmatrix}1 & x_0 & {x_0}^2\\ 1 & x_1 & {x_1}^2\\ 1 & x_2 & {x_2}^2\end{bmatrix}\begin{bmatrix}a_0\\ a_1\\ a_2\end{bmatrix}=\begin{bmatrix}y_0\\ y_1\\ y_n\end{bmatrix} \Rightarrow \begin{bmatrix}a_0\\ a_1\\ a_2\end{bmatrix}=\underbrace{\begin{bmatrix}1 & x_0 & {x_0}^2\\ 1 & x_1 & {x_1}^2\\ 1 & x_2 & {x_2}^2\end{bmatrix}^{-1}}_{Z=V^{-1}}\begin{bmatrix}y_0\\ y_1\\ y_n\end{bmatrix}=\begin{bmatrix}z_{0,0}y_0+z_{0,1}y_1+z_{0,2}y_2\\ z_{1,0}y_0+z_{1,1}y_1+z_{1,2}y_2\\ z_{2,0}y_0+z_{2,1}y_1+z_{2,2}y_2\end{bmatrix}$$

$$\begin{aligned}p(x)&=a_0+a_1x+a_2x^2\\ &=\left(z_{0,0}y_0+z_{0,1}y_1+z_{0,1}y_2\right)+\left(z_{1,0}y_0+z_{1,1}y_1+z_{1,1}y_2\right)x+\left(z_{2,0}y_0+z_{2,1}y_1+z_{2,1}y_2\right)x^2\\ &=y_0\underbrace{\left(z_{0,0}+z_{1,0}x+z_{2,0}x^2\right)}_{l_0(x)}+y_1\underbrace{\left(z_{0,1}+z_{1,1}x+z_{2,1}x^2\right)}_{l_1(x)}+y_2\underbrace{\left(z_{0,2}+z_{1,2}x+z_{2,2}x^2\right)}_{l_2(x)}\end{aligned}$$

Polynomial in Lagrange Form

- Because the polynomial $l_i(x)$ do not depend on y_i, but only on x_i, they must be the same for any arbitrary choice of y_i
- For example, if for a given index j, we choose

$$y_i = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

we would have the polynomial: $p_j(x) = y_j \, l_j(x) = l_j(x)$

because the polynomial must pass by all the points (x_i, y_i), it must be:

$$\begin{cases} p_j(x_i) = l_j(x_i) = y_i = 0 \text{ for all } i \neq j, \text{ i.e. } l_j(x) \text{ has } n \text{ distinct roots } \{x_i\}_{i \neq j} \\ p_j(x_j) = l_j(x_j) = y_j = 1 \end{cases}$$

we conclude that:

$$l_j(x) = \prod_{i \neq j} \frac{x - x_i}{x_j - x_i} \quad \text{and the polynomial is: } p(x) = \sum_{j=0}^{n} y_j \prod_{i \neq j} \frac{x - x_i}{x_j - x_i}$$

Error in Polynomial Approx

- Let's consider the polynomial approximation p(x) for the function f(x) passing through the points $(x_i, f(x_i))$
- For the points x_i the error is null. Can we estimate the error in a generic point $x \neq x_i$?
- Let's construct the auxiliary function:

$$w(t) = f(t) - p_n(t) - [f(x) - p_n(x)] \frac{\pi(t)}{\pi(x)}$$

$$\text{where } \pi(s) = (s - x_0)(s - x_1)\ldots(s - x_n)$$

$$\text{this has } n+2 \text{ roots: } t = x_0, x_1, \ldots, x_n, x$$

- The 1st derivative must have at least n+2-1 roots, the 2nd derivative at least n+2-2 roots, …, the n+1th derivative at least 1 root

Error in Polynomial Approx

The $(n+1)^{\text{th}}$ derivative is:

$$w^{(n+1)}(t) = f^{(n+1)}(t) - \underbrace{p_n^{(n+1)}(t)}_{0} - \underbrace{\left[f(x) - p_n(x)\right]}_{e(x)} \frac{\pi^{(n+1)}(t)}{\pi(x)}$$

$$= f^{(n+1)}(t) - e(x)\frac{(n+1)!}{\pi(x)}$$

we know there must be at least one point $\xi \in [a,b]$ where $w^{(n+1)}(\xi) = 0$

this allows us to conclude:

$$e(x) = \frac{\pi(x)}{(n+1)!} f^{(n+1)}(\xi), \quad \xi \in [a,b]$$

A bound of the error is:

$$E(x) = \frac{|\pi(x)|}{(n+1)!} \left\| f^{(n+1)}(t) \right\|_{\infty}, \quad t \in [a,b]$$

Divided Differences

- We have seen the Newton form and the Lagrange form.
- There is another form to describe a polynomial and find its coefficients called Divided Differences
- We won’t cover it in this course