Lecture 5: Interpolation

- Polynomial interpolation
- Rational approximation
- Coefficients of the polynomial

Interpolation:

Sometime we know the values of a function f(x) for a finite set of points x_i. Yet we want to evaluate f(x) for other values perhaps contained within the range of x_i (interpolation) or outside it (extrapolation).

To do this, we need to model the behavior of the function (since we don't know it) for these other points, usually by shooting a smooth curve that will go through the available points.

The most common interpolation schemes are based on polynomials, or quotients of polynomials. Trigonometric functions are also used, but we will treat that subject later on in the course.

There is an extensive mathematical literature, stemming sometimes from centuries ago with powerful theorems about errors, etc, in interpolation situations. This literature is in general useless, because it requires knowledge of the function f(x) we don't usually have.

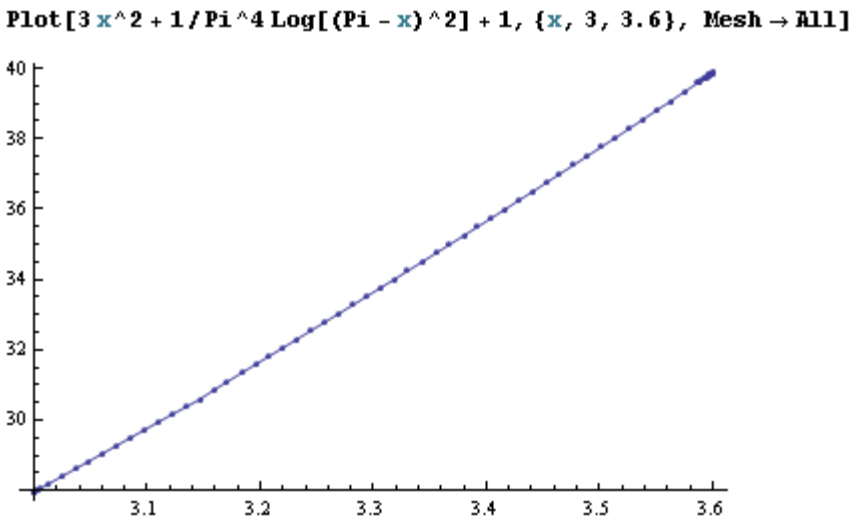
Some general issues:

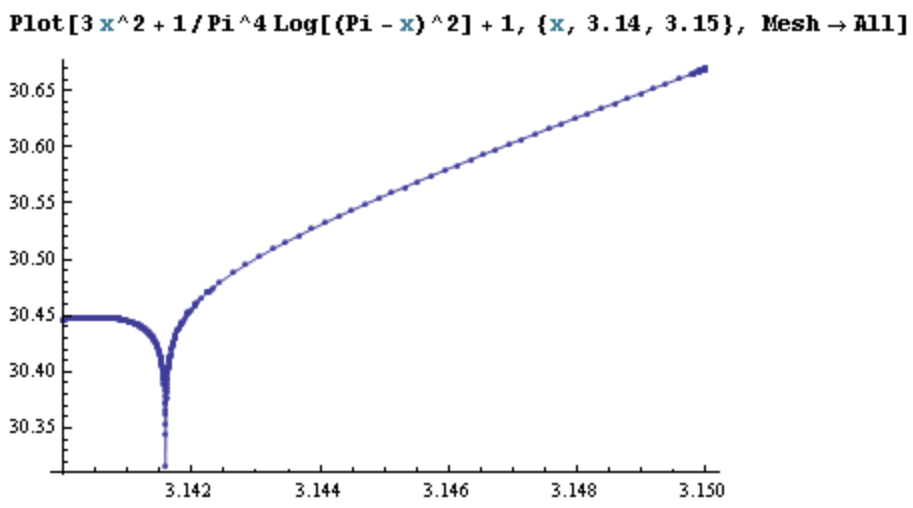
Conceptually, the interpolation process has two stages: determine the function that interpolates and then evaluate it at the point of interest. This is not the best procedure in practice: it usually is more costly and prone to round-off errors than to start with the value one has for $f(x_i)$ that is based on x_i closer to the point of interest and to add certain corrections to it.

The cardinality of the subset of points of x_i one uses to generate the interpolating function (minus one) is called the order of the interpolating scheme.

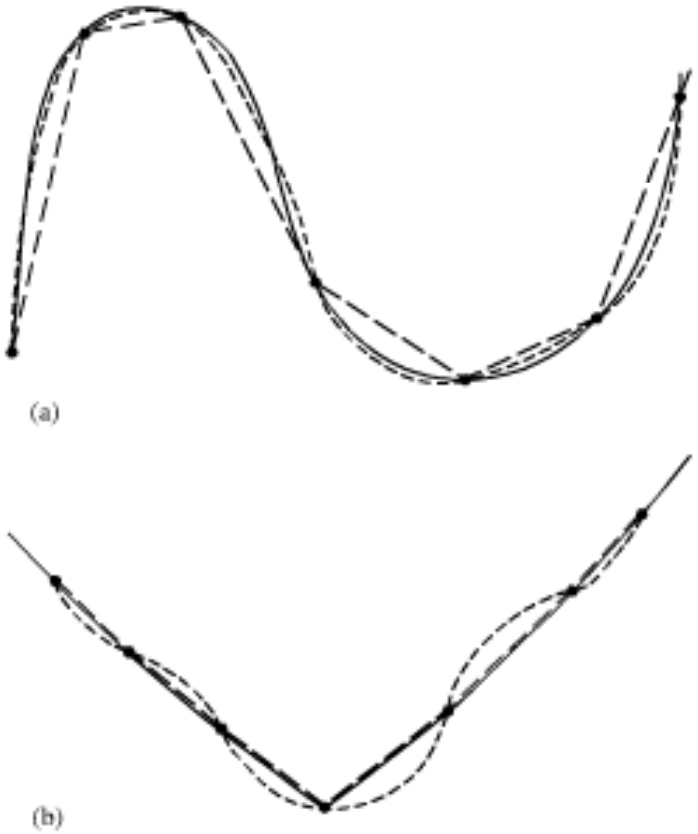
One can always find functions that make a mockery of any interpolation scheme at a given point. It is good practice therefore to construct methods that can somehow reliably estimate the error in the interpolation.

$$f(x) = 3x^2 + \frac{1}{\pi^4}\,\mathrm{Ln}[(\pi - x)^2] + 1$$





Higher order does not guarantee better accuracy. For instance, higher order polynomials tend to “wiggle” between the prescribed points (especially far away from each other) rather than produce a smoother function. If one uses additional points closer to the one of interest things get better, but this is not always an option.



The function that results from a local interpolation is usually not well behaved when one computes derivatives, since moving from point to points switches the set of points used and therefore does not yield smooth derivatives. If the purpose of the interpolation is to compute (especially high order) derivatives then one needs a scheme that at each point is able to supply such derivative accurately.
An example is the spline method that assigns a polynomial locally using non-local values. We will discuss it next class.

When the total number of x_i's used is much larger than the order of interpolation, a non-trivial part of the task is to find where within the table of x_i's to apply the method.

Polynomial interpolation:

Through any two points goes a straight line, through any three points a quadratic, etc. In general this is the content of Lagrange's interpolating formula. Given $y_1=f(x_1)$, $y_2=f(x_2)$, $y_N=f(x_N)$, we can get a polynomial of degree N-1 by,

$$P(x)=\frac{(x-x_2)(x-x_3)...(x-x_N)}{(x_1-x_2)(x_1-x_3)...(x_1-x_N)}y_1+\frac{(x-x_1)(x-x_3)...(x-x_N)}{(x_2-x_1)(x_2-x_3)...(x_2-x_N)}y_2$$
$$+...+\frac{(x-x_1)(x-x_2)...(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)...(x_N-x_{N-1})}y_N$$

We have N terms, all of degree N-1, and all designed to be zero at all of the x_i except one, at which it has the value y_i.

Let us work on an error estimate for this formula.

Error estimate of Lagrange's formula:

Construct:

$$g(t) = f(t) - P(t) - [f(x) - P(x)] \prod_{i=0}^{n} \frac{(t - x_i)}{(x - x_i)}$$

Now, it is true that $g(x_k)=0$

Vanish

It is also true that $g(x)=0$

Then g(t) vanishes for n+2 points: $x, x_0, x_1, \ldots, x_N$

If f belongs to C^{n+1}, then one can apply Rolle's theorem, which states that somewhere at a point ξ in the interval defined by the above points it has to happen that, $g^{(n+1)}(\xi) = 0$

$$g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - P^{(n+1)}(\xi) - [f(x) - P(x)] \frac{d^{n+1}}{dt^{n+1}} \prod_{i=0}^{n} \frac{(t - x_i)}{(x - x_i)}$$

So this term vanishes

And this term is a polynomial of degree n+1, therefore,

$$\prod_{i=0}^{n}\frac{(t-x_i)}{(x-x_i)}=\frac{1}{\prod_{i=0}^{n}(x-x_i)}t^{n+1}+(\text{lower degree terms})$$

Therefore, $$\frac{d^{n+1}}{dt^{n+1}}\prod_{i=0}^{n}\frac{(t-x_i)}{(x-x_i)}=\frac{(n+1)!}{\prod_{i=0}^{n}(x-x_i)}$$

Putting all this together, we get,

$$0=f^{(n+1)}(\xi)-0-[f(x)-P(x)]\frac{(n+1)!}{\prod_{i=0}^{n}(x-x_i)}$$

Or, $$f(x)=P(x)+\frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x-x_i)$$

It resembles the formula for the error in the Taylor formula.

$$f(x) = P(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x - x_i)$$

This formula exhibits why these kinds of results are usually not that useful.

It requires to know that the derivative of n+1 order exists, and it requires a bound for it in the interval.

Implementing Lagrange's formula directly in a program is also kind of awkward to program in an economical way. Let us now take a look at an alternative formula that is more efficient and has an easier error estimate. It is called *Neville's algorithm*.

Let $P_{i_1 i_2 \ldots i_n}$ be the polynomial of degree n - 1 that passes through the points $i_1 \ldots i_n$

Neville's observation is that these objects form a "tableau" of "ancestors" and "progeny"

$$\begin{array}{lllll}
x_1: & y_1 = P_1 & & & \\
 & & P_{12} & & \\
x_2: & y_2 = P_2 & & P_{123} & \\
 & & P_{23} & & P_{1234} \\
x_3: & y_3 = P_3 & & P_{234} & \\
 & & P_{34} & & \\
x_4: & y_4 = P_4 & & &
\end{array}$$

Neville's formula is

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

You can check the formula explicitly, however one can quickly convince oneself that it works by noticing that the progenitors coincide for $x_{i+1} \ldots x_{i+m-1}$

One can get a very efficient algorithm by considering the small differences between progenitors and progeny,

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

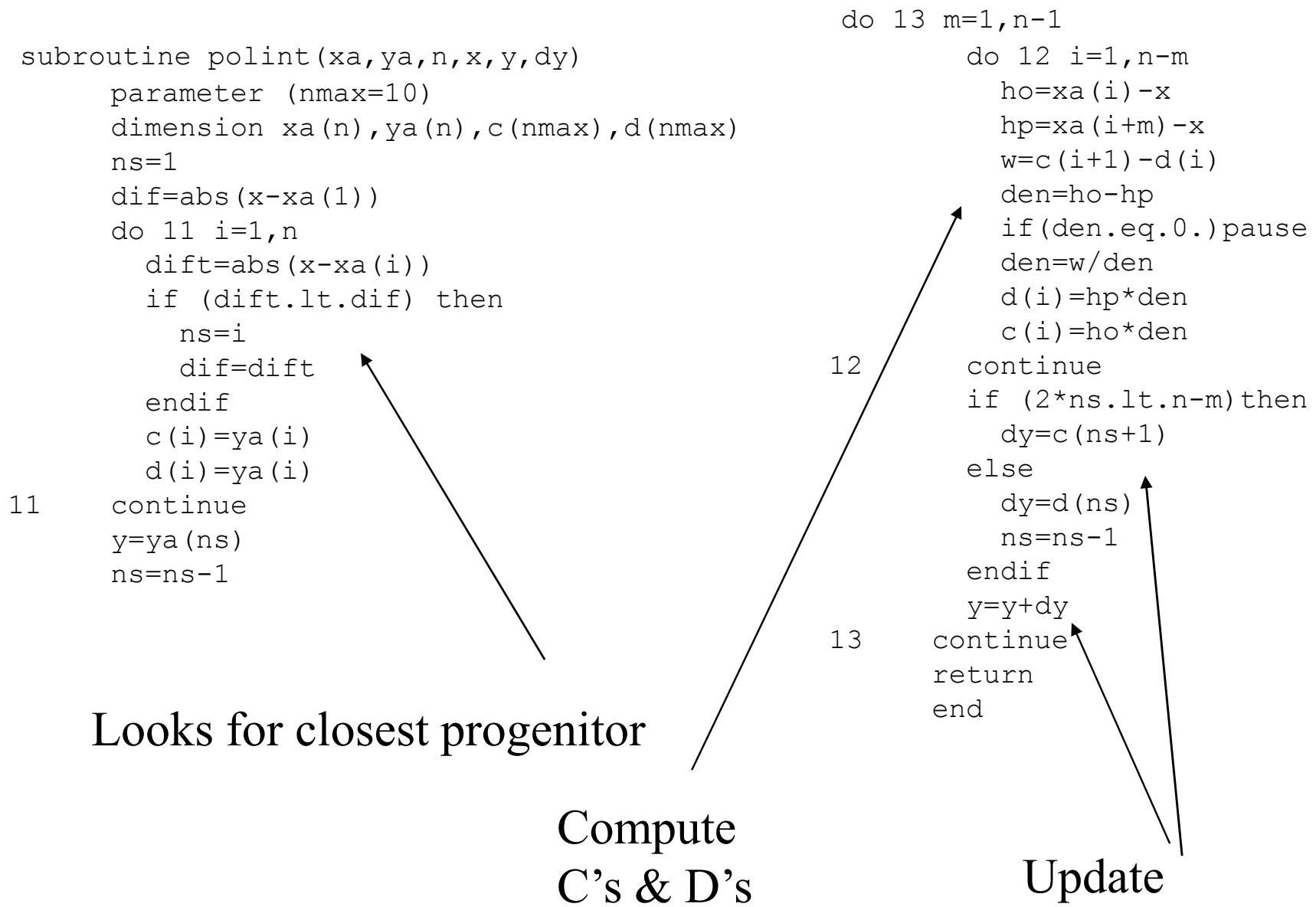
$$D_{m,i} = P_{i\ldots(i+m)} - P_{i\ldots(i+m)}$$

One can derive

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

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

The final $P_{1\ldots N}$ is given by the sum of any y_i plus all the C's that form a path from y_i to the desired $P_{1\ldots N}$ through the "family tree".


subroutine polint(xa,ya,n,x,y,dy)
parameter (nmax=10)
dimension xa(n),ya(n),c(nmax),d(nmax)
ns=1
dif=abs(x-xa(1))
do 11 i=1,n
dift=abs(x-xa(i))
if (dift.lt.dif) then
ns=i
dif=dift
endif
c(i)=ya(i)
d(i)=ya(i)
11 continue
y=ya(ns)
ns=ns-1
do 13 m=1,n-1
do 12 i=1,n-m
ho=xa(i)-x
hp=xa(i+m)-x
w=c(i+1)-d(i)
den=ho-hp
if(den.eq.0.)pause
den=w/den
d(i)=hp*den
c(i)=ho*den
12 continue
if (2*ns.lt.n-m)then
dy=c(ns+1)
else
dy=d(ns)
ns=ns-1
endif
y=y+dy
13 continue
return
end
Looks for closest progenitor
Compute
C's & D's
Update

Rational approximations:

$$R_{i(i+1)\ldots(i+m)} = \frac{p_0 + p_1 x + \ldots + p_r x^r}{q_0 + q_1 x + \ldots + q_s x^s} \qquad m+1 = r+s+1$$

The main advantage of rational approximations is that they allow to model functions with poles (zeros of the denominator).

This can be more important than one thinks: even if a function of a real variable does not have a pole, it might have one if extended to the complex plane. If a function has a pole in the complex plane, a power series will only be a good approximation far away from the pole, even if we restrict ourselves only to the real axis.

Rational approximations can be used in interpolations, but also in analytic work, for instance choosing a rational approximation that expanded in power series agrees with the power series of the function f(x) of interest. Such approximation is called *Padé approximation.* We will discuss this approximation later on in the course.

The formulae for rational approximations are similar to those for polynomial ones. Bulirsch and Stoer have come up with a Neville type relation for the R coefficients. In Numerical Recipes there is a coding of this relationship. The concepts are all the same as before, introducing the C's, D's, progenitors, progeny, etc, so we will not repeat them here.

Coefficients of the interpolating polynomial:

"A polynomial is nothing more than its coefficients" (My algebra teacher, ca. 1983)

This is true in principle, but numerically you should make sure that it is the coefficients what you need. The coefficients in general can be determined to much less accuracy than the value of the polynomial at some point. The Neville method we described, for instance, returns the exact values for x_i's, whereas if one first computes the coefficients and then evaluates the polynomial for the x_i's roundoff errors will give less accurate results.

One might need the coefficients if one wants to obtain something else, like a derivative of the polynomial, or perhaps its convolution against another function whose moments are known.

To get the coefficients one simply writes:

$$y = c_1 + c_2 x + c_3 x^2 + \ldots + c_N x^{N-1}$$

And evaluating the polynomial for all the points given, one gets a system of linear equations for the c_i's,

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{N-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^{N-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Matrices like the one on the left are called *Vandermonde* matrices

And one can handle it using the standard techniques for linear systems that we will discuss in the course later on.

A trick for handling *Vandermonde* matrices:

Consider the polynomial, $P(x) = \prod_{n=1, n\neq j}^{N} \frac{x - x_n}{x_j - x_n} = \sum_{k=1}^{N} A_{jk} x^{k-1}$

This polynomial is constructed so it vanishes for all x_i's, except for x_j, for which it takes the value one. In other words,

$$P(x_i) = \delta_{ij} = \sum_{k=1}^{N} A_{jk} x^{k-1}$$

But this implies that A is the matrix inverse of the Vandermonde matrix! Efficient algorithms can be constructed to build the coefficients of the polynomial and therefore A explicitly (see Numerical Recipes).

Summary

- It is better to use techniques that approximate the value than to compute the interpolating polynomial and evaluate.
- As in finite differences, higher order does not mean higher accuracy for interpolation.
- Neville's algorithms of "progenitors" and "progeny" are efficient.