Chapter 4. Interpolation & Extrapolation

§1. Introduction

(a) Basic Strategy: Given a data table (x_i, y_i), find $y(\bar{x})$ for an $\bar{x}$ which is not one of the data points.

* Naive approach: come up with a function $y(x)$ to go through all (or some) points; then calculate $y(\bar{x})$.
* This is inefficient and subject to roundoff error; inferior to the practice of constructing $y(\bar{x})$ directly from y_i.

— A philosophical question: why is extrapolation more "risky" than interpolation? Fundamentally no difference. For latter $\bar{x}$ bounded in (x_j, x_{j+1}).

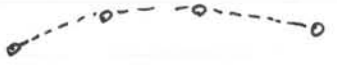
(b) Order of approximation & error estimate. How many points to use? Will additional points always increase accuracy? Two extremes: use 2 points or all points.

Order of interpolation = # of points used − 1.

Higher order does not guarantee higher accuracy!

(c) Local vs. nonlocal ~~fitting~~ interpolation

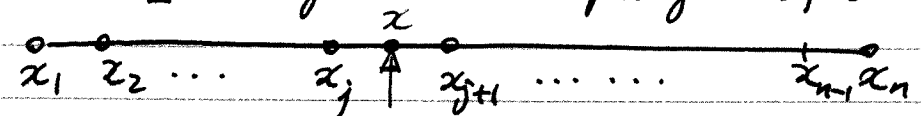
Local interp.: a finite # of nearest neighbors. This implies discontinuity in derivative as $\bar{x}$ crosses x_i. Very easy to see in 2-point linear interpolation, but generally true. "Nearest neighbors" change 1 member as $\bar{x}$ crosses x_i.



* Interpolation by "spline function": to ensure continuity to a certain order of derivative. "Stiffer" interpolation.

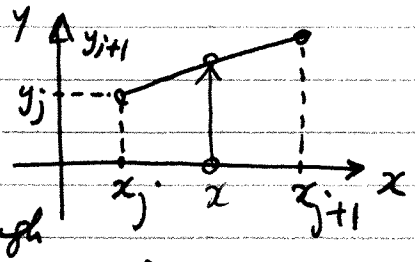
§2. Local interpolation

(a) Search table for the nearest neighbors, e.g. by bisection. [Can you write a program for this?]



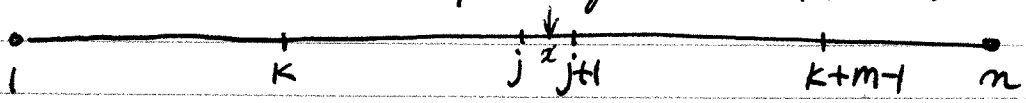
(b) Linear interpolation:

$$y = y_j + \frac{y_{j+1} - y_j}{x_{j+1} - x_j}(x - x_j)$$



(c) Polynomial interpolation through m points surrounding $x \in (x_j, x_{j+1})$.

- Find the indices of the points, $k, k+1, \cdots, k+m-1$.



$$K = \min\left(\max\left(j - \frac{m-1}{2}, 1\right), n+1-m\right)$$

Takes care of m being odd or even, and both ends.

* Lagrange formula:

$$P(x) = \frac{(x-x_2)(x-x_3)\cdots(x-x_m)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_m)} y_1 + \frac{(x-x_1)(x-x_3)\cdots(x-x_m)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_m)} y_2$$
$$+ \cdots + \frac{(x-x_1)\cdots(x-x_{m-1})}{(x_m-x_1)\cdots(x_m-x_{m-1})} y_m$$
$$= \sum_{j=1}^{m} \frac{\prod_{i=1, i\neq j}^{m} (x - x_i)}{\prod_{i=1, i\neq j}^{m} (x_j - x_i)} y_j \qquad \left((m-1)^{th} \text{ order polynomial.}\right)$$

- Straightforward, but not numerically efficient. One has to multiply the same factors many times.

(d) Neville's algorithm: a better implementation of the Lagrange formula. [Ref. N. R. §3.1, p 102.]

- Saves calculation; gives error estimation.

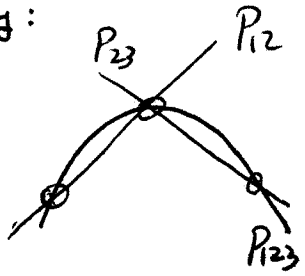
(Explain pyramidalong with formula)

- Pyramid:

Order	0	1	2	...	m−1
x_1	$P_1=y_1$	$C_{1,1}$, $D_{1,1}$ → P_{12}	$C_{2,1}$, $D_{2,1}$ → P_{123}	$C_{3,1}$ → ...	$P_{123\cdots m}$
x_2	$P_2=y_2$				
x_3	$P_3=y_3$	P_{23}			
⋮	⋮				
x_{m-1}	$P_{m-1}=y_{m-1}$	$P_{(m-1)m}$			
x_m	$P_m=y_m$				

P is polynomial that goes through all indexed points.

- How to construct P's algebraically:



$$P_{12}(x) = \frac{(x-x_2)P_1 + (x_1-x_1)P_2}{x_1-x_2}$$

$$P_{23}(x) = \frac{(x-x_3)P_2 + (x_2-x_2)P_3}{x_2-x_3}$$

⋮

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

$$P_{234}(x) = \frac{(x-x_4)P_{23} + (x_2-x_2)P_{34}}{x_2-x_4}$$

⋮

Note: (i) Two parents already agree on all internal points. Each "linear combination" adds the end points by raising polynomial power by 1.

(ii) Given an x value, these are NOT polyn., but #'s!

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

⋮

$$P_{i(i+1)\cdots(i+l)} = \frac{(x-x_{i+l})P_{i(i+1)\cdots(i+l-1)} + (x_i-x)P_{(i+1)\cdots(i+l)}}{x_i - x_{i+l}}$$

(Left (upper); right (lower))

- Error of interpolation & reformulation.

Going back to pyramid: each interpolation can be seen as correction of either "parent":

$$C_{l,i} = \boxed{P_{i\cdots(i+l)}}_{\text{new (child)}} - P_{i\cdots(i+l-1)} \leftarrow \text{upper}$$

$$D_{l,i} = \boxed{P_{i\cdots(i+l)}}_{\text{new (child)}} - P_{(i+1)\cdots(i+l)} \leftarrow \text{lower}$$

(l: level of interpolation; i: location in table.)

The C's and D's can be computed easily via a recursive formula:

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

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

Proof: $C_{l+1,i} = P_{i\cdots(i+l+1)} - P_{i\cdots(i+l)}$

$$= \frac{(x - x_{i+l+1})P_{i\cdots(i+l)} + (x_i - x)P_{(i+1)\cdots(i+l+1)}}{x_i - x_{i+l+1}} - P_{i\cdots(i+l)}$$

$$= \frac{(x - x_i)P_{i\cdots(i+l)} + (x_i - x)P_{(i+1)\cdots(i+l+1)}}{x_i - x_{i+l+1}}$$

$$= \frac{x_i - x}{x_i - x_{i+l+1}}\left[P_{(i+1)\cdots(i+l+1)} - P_{i\cdots(i+l)}\right]$$

two half siblings sharing one parent: $C_{l,i+1}$, $D_{l,i}$, $P_{i\cdots(i+l)}$

$$= \frac{x_i - x}{x_i - x_{i+l+1}}\left[C_{l,i+1} - D_{l,i}\right].$$

Code implementation:

- Cumulate the errors to arrive at final value.

* Why are C's & D's errors? Each P is an interpolation for $y(x)$ based on different set of points. C, D are differences between one order of interpolation & the next. Last ones used as criteria.

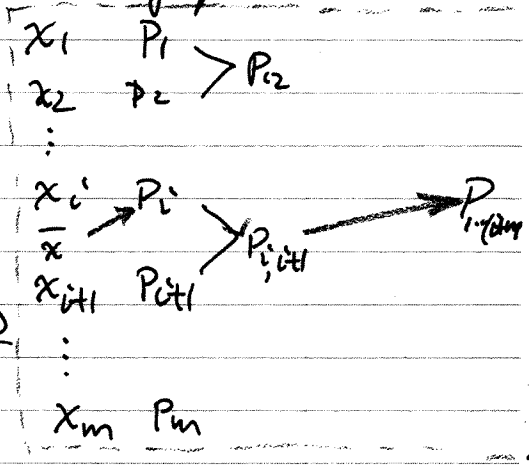
a) Where to start? Start from data point closest to target x. That way, we use the closest data points for interpolation. The errors are smaller, and we can hope to terminate within fewer layers.

b) How to initialize C's & D's? It turns out that for the 1st column, we use the successive formulas for C_{11}, C_{12}, etc as if $C_{01}=D_{01}=P_1, \cdots, C_{0i}=D_{0i}=P_i$. This is as if the ancestors on the left of $P_1, \cdots, P_i$ are 0's.

Test: $C_{11} = \frac{(x_1-x)(C_{02}-D_{01})}{x_1-x_2} = \frac{x_1-x}{x_1-x_2}(P_2-P_1)$.

$= \frac{(x_1-x)P_2+(x-x_2)P_1}{x_1-x_2} - P_1 = P_{12}-P_1$.

c) Choose path that is straight from x to the tip, a path centered around the starting point. (Using closest neighbors)



d) Amount of calculation:

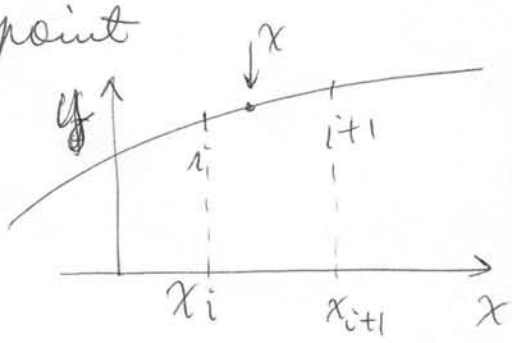
- Each P has 2 multiplication and 1 division $\Rightarrow$ 3 calculations
- Each level has $(i-1)$ P's: $(m-1)+(m-2)+\cdots+2+1 = \frac{m(m-1)}{2}$
- Total: $\frac{3}{2}m(m-1)$.
- Lagrange: $2(m-1)+1+1 = 2m$ for each point; total $2m^2$ calculations.

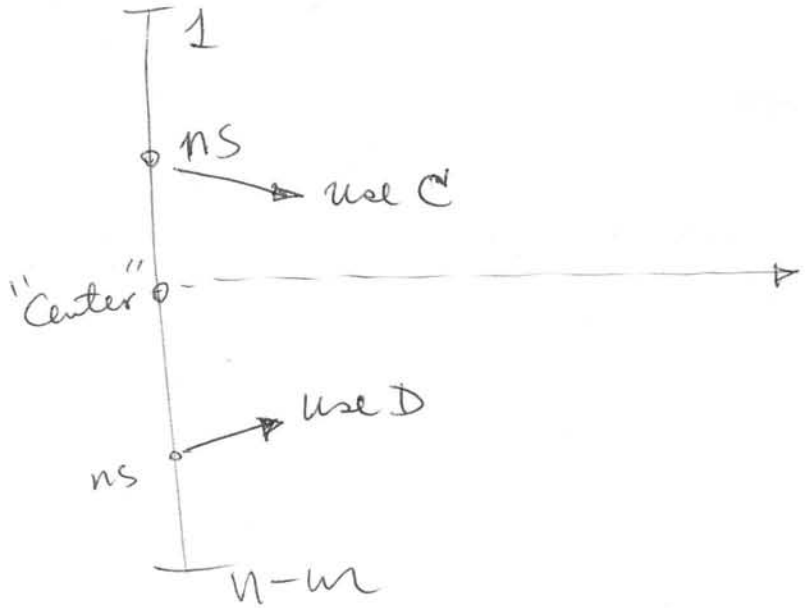
Neville: saves more that 25%, & gives error estimation

```
      SUBROUTINE polint(xa,ya,n,x,y,dy)
      INTEGER n,NMAX
      REAL dy,x,y,xa(n),ya(n)
      PARAMETER (NMAX=10)
      INTEGER i,m,ns
      REAL den,dif,dift,ho,hp,w,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 'failure in polint'
          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
```

← Find closest point

} First column of errors = y_i.



update c and d.

← which direction to go? Up or down?