Interpolation

Data types in science
- Discrete data (data tables)
- Experiment
- Observations
- Calculations
- Continuous data
  - Analytics functions
  - Analytic solutions

From continuous to discrete ...
soon after hurricane Isabel (September 2003)

From discrete to continuous?

What do we want from discrete sets of data?
\( \{f_i(x_i) \ i = 1, n\} \)

Quite often we need to know function values at any arbitrary point \( x \).
Can we generate values for any \( x \) between \( x_1 \) and \( x_n \) from a data table?

If you think that the data values \( f_i \) in the data table are free from errors, then interpolation lets you find an approximate value for the function \( f(x) \) at any point \( x \) within the interval \( x_1 \ldots x_n \).
Key point!!!
The idea of interpolation is to select a function \( g(x) \) such that

1. \( g(x) = f_i \) for each data point \( i \)
2. this function is a good approximation for any other \( x \) between original data points

Applications of approximating functions

- interpolation
- differentiation
- integration

What is a good approximation?
- What can we consider as a good approximation to the original data if we do not know the original function?
- Data points may be interpolated by an infinite number of functions

Important to remember

Interpolation = Approximation
There is no exact and unique solution
The actual function is NOT known and CANNOT be determined from the tabular data.

Two step procedure
- Select a function \( g(x) \)
- Find coefficients

Step 1: selecting a function \( g(x) \)
We should have a guideline to select a reasonable function \( g(x) \).
We need some ideas about data!!!
- \( g(x) \) may have some standard form
- or be specific for the problem
Some ideas for selecting $g(x)$

- Most interpolation methods are grounded on ‘smoothness’ of interpolated functions. However, it does not work all the time.
- Practical approach, i.e. what physics lies beneath the data.

Linear combination is the most common form of $g(x)$

- linear combination of elementary functions, or trigonometric, or exponential functions, or rational functions, ...

$$g(x) = a_1 h_1(x) + a_2 h_2(x) + a_3 h_3(x) + ...$$

Three of most common approximating functions

- Polynomials
- Trigonometric functions
- Exponential functions

Approximating functions should have following properties

- It should be easy to determine
- It should be easy to evaluate
- It should be easy to differentiate
- It should be easy to integrate

Linear Interpolation: Idea

The idea of linear interpolation is to approximate data at a point $x$ by a straight line passing through two data points $x_j$ and $x_{j+1}$ closest to $x$.

$$g(x) = a_0 + a_1 x$$

where $a_0$ and $a_1$ are coefficients of the linear functions. The coefficients can be found from a system of equations

$$g(x_j) = f_j = a_0 + a_1 x_j$$  \hspace{1cm} (4.2)$$
$$g(x_{j+1}) = f_{j+1} = a_0 + a_1 x_{j+1}$$  \hspace{1cm} (4.3)$$

Example: C++

```cpp
double int1(double x, double xi[], double yi[], int imax) {
    double y;
    int j;
    // if x is outside the xi[] interval
    if (x <= xi[0])      return y = yi[0];
    if (x >= xi[imax-1]) return y = yi[imax-1];
    // loop to find j so that xi[j] < x < xi[j+1]
    j = 0;
    while (j <= imax-1) {
        if (xi[j] >= x) break;
        j = j + 1;
    }
    y = yi[j-1]*(y1[j] - y1[j-1])/(x1[j] - x1[j-1]);
    return y;
}
```

bisection approach is much more efficient to search an array
Linear interpolation: conclusions

- The linear interpolation may work well for very smooth functions when the second and higher derivatives are small.
- It is worthwhile to note that for each data interval one has a different set of coefficients \( a_0 \) and \( a_1 \).
- This is the principal difference from data fitting where the same function, with the same coefficients, is used to fit the data points on the whole interval \([x_1, x_n]\).
- We may improve quality of linear interpolation by increasing number of data points \( x_i \) on the interval.
- HOWEVER!!! It is much better to use higher-order interpolations.

Linear vs. Quadratic interpolations

example from F.S. Acton “Numerical methods that work” “A table of \( \sin(x) \) covering the first quadrant, for example, requires 541 pages if it is to be linearly interpolable to eight decimal places. If quadratic interpolation is used, the same table takes only one page having entries at one-degree intervals.”

Polynomial Interpolation

Polynomial interpolation is a very popular method due, in part, to simplicity

\[
g(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n. \quad (4.6)
\]

The condition that the polynomial \( g(x) \) passes through sample points \( f_j(x_j) \)

\[
f_j(x_j) = g(x_j) = a_0 + a_1 x_j + a_2 x_j^2 + \ldots + a_n x_j^n. \quad (4.7)
\]

The number of data points minus one defines the order of interpolation. Thus, linear (or two-point interpolation) is the first order interpolation.

Properties of polynomials

Weierstrass theorem:

If \( f(x) \) is a continuous function in the closed interval

\( a \leq x \leq b \) then for every \( \varepsilon > 0 \) there exists a polynomial \( P_n(x) \), where the value on \( n \) depends on the value of \( \varepsilon \), such that for all \( x \) in the closed interval \( a \leq x \leq b \)

\[
|P_n(x) - f(x)| < \varepsilon
\]

System of equations for the second order

\[
f_2 = a_0 + a_1 x_2 + a_2 x_2^2
\]

\[
f_{j+1} = a_0 + a_1 x_{j+1} + a_2 x_{j+1}^2
\]

\[
f_{j+2} = a_0 + a_1 x_{j+2} + a_2 x_{j+2}^2
\]

⇒ We need three points for the second order interpolation

⇒ Freedom to choose points?

⇒ This system can be solved analytically
Solutions for the second order

\[ p(x) = f_1(x) = (x-x_1^2)(x-x_2^2) + f_2(x) = (x-x_1)(x-x_2) + \]

and any arbitrary order (Lagrange interpolation)

\[ f(x) = \sum_{i=1}^{n} y_i \lambda_i(x) \]

\[ \lambda_i(x) = \prod_{j=1, j \neq i}^{n} \frac{x-x_j}{x_i-x_j} \]

Lagrange Interpolation: C++

```cpp
double point(double x, double x[], double y[], int isize, int npoints)
{
    double lambda[isize];
    double y;
    int j, i, il;
    // check order of interpolation
    if (npoints > isize) npoints = isize;
    // if x is outside the x[] interval
    if (x <= x[0]) return y = y[0];
    if (x >= x[isize-1]) return y = y[isize-1];
    // loop to find j so that x[j-1] < x < x[j]
    j = 0;
    while (j <= isize-1)
    {
        if (x < x[j]) break;
        j = j + 1;
    }
    // shift j to correspond to (npoint-1)th interpolation
    j = j - npoints/2;
    // if j is outside the range [0, ... isize-1]
    if (j < 0) j=0;
    if (j+npoints-1 > isize-1 ) j=isize-npoints;
    y = 0.0;
    for (i = j; i <= j+npoints-1; i = i+1)
    {
        lambda[i] = 1.0;
        for (il = j; il <= j+npoints-1; il = il + 1)
        {
            if(il != i) lambda[i] = lambda[i] * (x-x[il])/(x[i]-x[il]);
        }
        y = y + y[i]*lambda[i];
    }
    return y;
}
```

Example: C++ (cont.)

```cpp
// shift j to correspond to (npoint-1)th interpolation
j = j - npoints/2;
// if j is outside the range [0, ... isize-1]
if (j < 0) j=0;
if (j+npoints-1 > isize-1 ) j=isize-npoints;
y = 0.0;
for (i = j; i <= j+npoints-1; i = i+1)
{
    lambda[i] = 1.0;
    for (il = j; il <= j+npoints-1; il = il + 1)
    {
        if(il != i) lambda[i] = lambda[i] * (x-x[il])/(x[i]-x[il]);
    }
    y = y + y[i]*lambda[i];
}
return y;
```

Note on Lagrange interpolation

- It is possible to use Lagrange formula straightforwardly, like in the example above.
- There is a better algorithm - Neville’s algorithm (for constructing the same interpolating polynomial).
- Sometimes Neville’s algorithm confused with Aitken’s algorithm (the latter now considered obsolete).

Important!

- Moving from the first order to the third and fifth order improves interpolated values to the original function.
- However, the seventh order interpolation instead being closer to the function f(x) produces wild oscillations.
- This situation is not uncommon for high-order polynomial interpolation.
- Rule of thumb: do not use high order interpolation. Fifth order may be considered as a practical limit.
- If you believe that the accuracy of the fifth order interpolation is not sufficient for you, then you should rather consider some other method of interpolation.
Cubic Spline

- The idea of spline interpolation is reminiscent of very old mechanical devices used by draftsmen to get a smooth shape.
- It is like securing a strip of elastic material (metal or plastic ruler) between knots (or nails).
- The final shape is quite smooth.
- Cubic spline interpolation is used in most plotting software. (cubic spline gives most smooth result)

Spline vs. Polynomials

- One of the principal drawbacks of the polynomial interpolation is related to discontinuity of derivatives at data points $x_i$.
- The procedure for deriving coefficients of spline interpolations uses information from all data points, i.e. nonlocal information to guarantee global smoothness in the interpolated function up to some order of derivatives.

Equations

The interpolated function on $[x_j, x_{j+1}]$ interval is presented in a cubic spline form

$$g(x) = f_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

What is different from polynomial interpolation?
... the way we are looking for the coefficients!

Polynomial interpolation: 3 coefficient for 4 data points
Spline interpolation: 3 coefficients for the each interval

Coefficients for spline interpolation

- For the each interval we need to have a set of three parameters $b_j, c_j$, and $d_j$.
  Since there are $(n-1)$ intervals, one has to have $3n-3$ equations for deriving the coefficients for $j=1,n-1$.
- The fact that $g(x_j)=f(x_j)$ imposes $(n-1)$ equations.

Central idea to spline interpolation

- the interpolated function $g(x)$ has continuous the first and second derivatives at each of the $n-2$ interior points $x_j$.

$$g'_{j-1}x_j = g_j'x_j$$
$$g''_{j-1}x_j = g''_jx_j.$$
Cubic spline boundary conditions

Possibilities to fix two more conditions

• Natural spline - the second order derivatives are zero on boundaries.
  \[ f''(x_i) = 0 \]

• Input values for the first order \( f'(x) \) derivates at boundaries
  \[ f'(x_0) = f'(x_N) \]

• Input values for the second order \( f''(x) \) derivates at boundaries
  \[ f''(x_0) = f''(x_N) \]

\[ s_i(x) = a_i + b_i(x-x_i) + \frac{c_i}{2}(x-x_i)^2 + \frac{d_i}{6}(x-x_i)^3 \]

\[ x_{i-1} \leq x \leq x_i \quad i = 1, 2, \ldots, N \]

Need to find \( a_i, b_i, c_i, d_i \):

\[ s_i'(x) = b_i + c_i(x-x_i) + \frac{d_i}{2}(x-x_i)^2 \]
\[ s_i''(x) = c_i + d_i(x-x_i) \]
\[ s_i'''(x) = d_i \]

by the definition (interpolation) \( a_i = f(x_i) \)

1) \( s(x) \) must be continuous at \( x_i \):
\[ s_{i-1}(x_i) = s_i(x_i) \]

\[ a_i = a_{i-1} + b_{i-1}(x_i-x_{i-1}) + \frac{c_{i-1}}{2}(x_i-x_{i-1})^2 + \frac{d_{i-1}}{6}(x_i-x_{i-1})^3 \]

The system of equations to find coefficients

\[ h_i d_i = c_i - c_{i-1} \quad i = 1, 2, \ldots, N \quad c_0 = c_N = 0 \]

\[ h_i c_i - \frac{h_i^2}{2} d_i = b_i - b_{i-1} \quad i = 2, 3, \ldots, N \]

\[ h_i c_i - \frac{h_i^2}{6} d_i = f_i - f_{i-1} \quad i = 1, 2, \ldots, N \]

Solve the system for \( c_i \) \( i = 1, 2, \ldots, N-1 \)

Consider two equations (7) for points \( i \) and \( i-1 \)

\[ b_i = \frac{h_i^2}{2} c_i + \frac{h_i^2}{6} d_i = \frac{f_i - f_{i-1}}{h_i} \]
\[ b_{i-1} = \frac{h_{i-1}^2}{2} c_{i-1} + \frac{h_{i-1}^2}{6} d_{i-1} = \frac{f_{i-1} - f_{i-2}}{h_{i-1}} \]

and subtract the second equation from the first

\[ b_i - b_{i-1} = \frac{1}{2}(h_i c_i - h_{i-1} c_{i-1}) - \frac{1}{6}(h_i^2 d_i - h_{i-1}^2 d_{i-1}) + \frac{f_i - f_{i-1}}{h_i} - \frac{f_{i-1} - f_{i-2}}{h_{i-1}} \]

Use the difference \( h_i - h_{i-1} \) in the right side of (6)

\[ h_i c_i + h_{i-1} c_{i-1} + \frac{h_i^2}{2} d_i - \frac{2h_i^2}{3} d_i = 2 \left( \frac{f_i - f_{i-1}}{h_i} - \frac{f_{i-1} - f_{i-2}}{h_{i-1}} \right) \]

Then from (5)

\[ h_i^2 d_i = h_i (c_i - c_{i-1}) \]
\[ h_i^2 d_{i-1} = h_{i-1} (c_{i-1} - c_{i-2}) \]

Substitute in (8)

\[ h_i c_i + 2h_i c_{i-1} + h_{i-1} c_{i-2} + h_{i-1} c_i = 6 \left( \frac{f_i - f_{i-1}}{h_i} - \frac{f_{i-1} - f_{i-2}}{h_{i-1}} \right) \]

for \( i = 1, 2, \ldots, N-1 \), \( c_0 = c_N = 0 \)

This is a tridiagonal system of equations (use Thomas method)

then \( d_i = \frac{c_i - c_{i-1}}{h_i} \), \( b_i = \frac{h_i}{2} c_i + \frac{h_i^2}{6} d_i + \frac{f_i - f_{i-1}}{h_i} \) for \( i = 1, 2, \ldots, N \),

Implementation

Recommendation – do not write your own spline program but get one from ….

- Advanced program libraries!
- Books
Comments to Spline interpolation

Generally, spline does not have advantages over polynomial interpolation when used for smooth, well behaved data, or when data points are close on x scale.

The advantage of spline comes into the picture when dealing with “sparse” data, when

- there are only a few points for smooth functions
- or when the number of points is close to the number of expected maximums.

Example: Spline interpolation

![Graph showing spline and polynomial interpolation](image)

Divided Differences

The first divided difference at point i:

\[ f[x_i, x_{i+1}] = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} \]

The second divided difference:

\[ f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} \]

In general:

\[ f[x_i, x_{i+1}, ..., x_n] = \frac{f[x_{i+1}, x_{i+2}, ..., x_n] - f[x_i, x_{i+1}, ..., x_{i+2}]}{x_n - x_i} \]

Other notations:

\[ f_i^{(0)} = f_i, \quad f_i^{(1)} = f[x_i, x_{i+1}], \quad f_i^{(2)} = f[x_i, x_{i+1}, x_{i+2}] \]

Divided Difference Polynomials

Let’s define a polynomials \( P_n(x) \) where the coefficients are divided differences:

\[ P_n(x) = f_i^{(0)} + (x - x_i) f_i^{(1)} + (x - x_i)(x - x_{i+1}) f_i^{(2)} + \cdots + (x - x_i)(x - x_{i+1}) \cdots (x - x_{i+n-1}) f_i^{(n)} \]

It is easy to show that \( P_n(x) \) passes exactly through the data points:

\[ P_n(x_i) = f_i, \quad P_n(x_{i+1}) = f_{i+1}, \cdots \]
Comments

Polynomials satisfy a uniqueness theorem – a polynomial of degree \( n \) passing exactly through \( n+1 \) points is unique.

The data points in the divided difference polynomials do not have to be in any specific order. However, more accurate results are obtained for interpolation if the data are arranged in order of closeness to the interpolated point.

Example: Fortran

```fortran
Double precision d(n,n), x(n), f(n)
integer i,j
............... 
\! d(\,0,0) \! 
| initialization of d(n,1) 
\! do i=1,n 
\! d(i,1) = f(i) 
\! end do 
| calculations 
\! do j=2,n 
\! do i=1,n-j+1 
\! d(i,j)=(d(i+1,j-1)-d(i,j-1))/(x(i+1+j-2)-x(i)) 
\! end do 
\! end do 
| print results 
\! do i=1,n 
\! write(*,200) (d(i,j),j=1,n-i+1) 
\! end do 
200 format (5f10.6)
```

Example:

<table>
<thead>
<tr>
<th>(x)</th>
<th>3.20, 3.30, 3.35, 3.40, 3.50, 3.60, 3.65, 3.70</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td>0.312500, 0.303030, 0.298507, 0.294118, 0.285714, 0.277778, 0.273973, 0.270270</td>
</tr>
</tbody>
</table>

Equally spaced \( x_i \)

Fitting approximating polynomials to tabular data is much easier when the values \( x_i \) are equally spaced:

the forward difference relative to point \( i \)

\[ f_{i+1} - f_i = \Delta f_i \]

the backward difference relative to point \( i+1 \)

\[ f_{i-1} - f_i = \nabla f_i \]

the central difference relative to point \( i+1/2 \)

\[ f_{i+1} - f_{i-1} = \delta f_{i+1/2} \]
Difference tables are useful for evaluating the quality of a set of tabular data.

- Smooth differences => "good" data
- Not monotonic differences => possible errors in the original data, or the step in \( x \) is too large, or may be a singularity in \( f(x) \).

### The Newton Difference Polynomials

The Newton forward difference polynomials

\[
P_n(x) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_0 + \ldots + \frac{s(s-1)(s-2)\ldots(s-[n-1])\Delta^n f_0}{n!}
\]

The Newton backward difference polynomials

\[
P_n(x) = f_0 + s\nabla f_0 + \frac{s(s+1)}{2!}\nabla^2 f_0 + \frac{s(s+1)(s+2)}{3!}\nabla^3 f_0 + \ldots + \frac{s(s+1)(s+2)\ldots(s+[n-1])\nabla^n f_0}{n!}
\]

where \( s = (x - x_0)/h \), \( x = x_0 + sh \).

A major advantage of the Newton forward and backward difference polynomials is that each higher order polynomial is obtained from the previous lower-degree polynomials simply by adding the new term.

Other difference polynomials:

- # Strirling centered-difference polynomials
- # Bessel centered-difference polynomials

### Example: compare Lagrange and divided differences interpolations

#### Quality of Lagrange interpolation:

<table>
<thead>
<tr>
<th>average difference from ( f(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
</tr>
<tr>
<td>0.1410</td>
</tr>
</tbody>
</table>

#### Quality of interpolation: average difference from \( f(x) \)

<table>
<thead>
<tr>
<th>Orders of divided difference interpolation</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st interpolation</td>
<td>0.1410</td>
<td>0.1484</td>
<td>0.0848</td>
<td>0.1091</td>
<td>0.1433</td>
</tr>
<tr>
<td>2nd interpolation</td>
<td>0.1410</td>
<td>0.1484</td>
<td>0.0848</td>
<td>0.1091</td>
<td>0.1433</td>
</tr>
<tr>
<td>3rd interpolation</td>
<td>0.1410</td>
<td>0.1484</td>
<td>0.0848</td>
<td>0.1091</td>
<td>0.1433</td>
</tr>
<tr>
<td>4th interpolation</td>
<td>0.2022</td>
<td>0.2808</td>
<td>0.3753</td>
<td>0.4526</td>
<td>0.5293</td>
</tr>
<tr>
<td>5th interpolation</td>
<td>0.2022</td>
<td>0.2808</td>
<td>0.3753</td>
<td>0.4526</td>
<td>0.5293</td>
</tr>
</tbody>
</table>

#### Rational Function Interpolation

- A rational function \( g(x) \) is a ratio of two polynomials.
- Often rational function interpolation is a more powerful method compared to polynomial interpolation.

\[
g(x) = \frac{a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n}{b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m}
\]
When using rational functions is a good idea?

Rational functions may well interpolate functions with poles

\[ g(x) = \frac{a_0 + a_1 x + a_2 x^2 + \ldots a_n x^n}{b_0 + b_1 x + b_2 x^2 + \ldots b_m x^m} \]

that is with zeros of the denominator

\[ b_0 + b_1 x + b_2 x^2 + \ldots b_m x^m = 0 \]

The procedure has two principal steps

- On the first step we need to choose powers for the numerator and the denominator, i.e. n and m. You may need to attempt a few trials before coming to a conclusion.
- Once we know the number of parameters we need to find the coefficients

Example for n=2 and m=1

\[ g(x) = \frac{a_0 + a_1 x + a_2 x^2}{b_0 + b_1 x} \]

- We need five coefficients i.e. 5 data points
- We should fix one of the coefficients since only the ratio makes sense.
- If we choose, for example, \( b_0 \) as a fixed number \( c \) then we need 4 data points to solve a system of equations to find the coefficients

Finding coefficients using 4 data points

\[ \begin{align*}
    f(x_1)(c + b_1 x_1) &= a_0 + a_1 x_1 + a_2 x_1^2 \\
    f(x_2)(c + b_1 x_2) &= a_0 + a_1 x_2 + a_2 x_2^2 \\
    f(x_3)(c + b_1 x_3) &= a_0 + a_1 x_3 + a_2 x_3^2 \\
    f(x_4)(c + b_1 x_4) &= a_0 + a_1 x_4 + a_2 x_4^2
\end{align*} \]

- After a few rearrangements the system may be written in the traditional form for a system of linear equations.
- Very stable and robust programs for rational function interpolation can be found in many standard program libraries.

Interpolation in two or more dimensions

- bilinear interpolation
- bicubic interpolation
- bicubic spline
- ...

Applications for Interpolation

- Interpolation has many applications both in physics, science, and engineering.
- Interpolation is a corner’s stone in numerical integration (integrations, differentiation, Ordinary Differential Equations, Partial Differential Equations).
- Two-dimensional interpolation methods are widely used in image processing, including digital cameras.
**Are we there yet?**

**What is extrapolation?**
- If you are interested in function values outside the range $x_1...x_n$ then the problem is called **extrapolation**.
- Generally this procedure is much less accurate than interpolation.
- You know how it is difficult to extrapolate (foresee) the future, for example, for the stock market.

**What is data fitting?**
- If data values $f(x_i)$ are a result of experimental observation with some errors, then **data fitting** may be a better way to proceed.
- In data fitting we let $g(x_i)$ to differ from measured values $f$, at $x_i$ points having one function only to fit all data points; i.e. the function $g(x)$ fits all the set of data.
- Data fitting may reproduce well the trend of the data, even correcting some experimental errors.