Interpolation

In regression the fitted function does not pass through the specified knots, only approximates them. In interpolation, however, graph of the function does pass through the given knots.

Interpolation with one polynomial

A polynomial can be given as

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

with real coefficients $a_n, a_{n-1}, \cdots, a_1, a_0$, where $n$ is degree of the polynomial. A first-degree polynomial is a linear function.

Let us have a data set of $n$ points. This data set may be approximated by polynomials of increasing degree up to at most of degree $(n-1)$. Fit with a lower degree polynomial is a regression, while fit with polynomial of degree $(n-1)$ is interpolation and this polynomial passes through all the points or knots. This is called **global** polynomial interpolation, since one function is fitted to all the data.

**Problem**

Wind turbine output power depends on wind velocity. Experimental data for five different velocities are shown in the following table:

<table>
<thead>
<tr>
<th>Wind velocity [km/h]</th>
<th>22</th>
<th>35</th>
<th>48</th>
<th>61</th>
<th>74</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power [W]</td>
<td>320</td>
<td>490</td>
<td>540</td>
<td>500</td>
<td>480</td>
</tr>
</tbody>
</table>

A fourth degree polynomial can be fitted to these five data points. Let us interpolate wind turbine output power for wind velocities 42 and 68 km/h. What wind velocity produces 400 W output power?

Load and plot the data file `windturbine.txt`.

% wind turbine data
clear all; close all; clc;
data = load('windturbine.txt')
data =
  22  320
  35  490
  48  540
  61  500
  74  480

v = data(:,1) % wind velocity

v =
  22
  35
  48
  61
  74

p = data(:,2) % output power

p =
  320
  490
  540
  500
  480

figure(1)
plot(v,p,'r*')

Coefficients of the fourth degree interpolation polynomial may be computed by solving a system of five linear equations specified for the five points, just like we learned in case of regression. Matrix $A$ of the system is called Vandermonde matrix.

\[
\begin{align*}
   y_1 &= a_4 x_1^4 + a_3 x_1^3 + a_2 x_1^2 + a_1 x_1 + a_0 \\
   y_2 &= a_4 x_2^4 + a_3 x_2^3 + a_2 x_2^2 + a_1 x_2 + a_0 \\
   y_3 &= a_4 x_3^4 + a_3 x_3^3 + a_2 x_3^2 + a_1 x_3 + a_0 \\
   y_3 &= a_4 x_4^4 + a_3 x_4^3 + a_2 x_4^2 + a_1 x_4 + a_0 \\
   y_5 &= a_4 x_5^4 + a_3 x_5^3 + a_2 x_5^2 + a_1 x_5 + a_0
\end{align*}
\]

Therefor:

\[
A = \begin{pmatrix}
  x_1^4 & x_1^3 & x_1^2 & x_1 & 1 \\
  x_2^4 & x_2^3 & x_2^2 & x_2 & 1 \\
  x_3^4 & x_3^3 & x_3^2 & x_3 & 1 \\
  x_4^4 & x_4^3 & x_4^2 & x_4 & 1 \\
  x_5^4 & x_5^3 & x_5^2 & x_5 & 1
\end{pmatrix} ;
\quad b = \begin{pmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4 \\
  y_5
\end{pmatrix}
\]

Alternatively built-in Matlab functions `polyfit`, `polyval` can be used to solve our problem and
make a plot:

```matlab
a = polyfit(v,p,4)  %  0.0001  -0.0171   0.5627   12.0190  -62.0517

a =
    0.0001  -0.0171   0.5627   12.0190  -62.0517

polyval(a,42)  %  531.7853
ans = 531.7853

fp = @(x) polyval(a,x)

fp = function_handle with value:
    @(x)polyval(a,x)

fp(68)  %  476.5008
ans = 476.5008

hold on;
fplot(fp,[min(v) max(v)])
```

If we want to compute wind velocity for output power of 400 W we need to solve a nonlinear equation. The required initial value can be found from the plot of the function.

```matlab
h = @(x) fp(x)-400
```
h = function_handle with value:
    @(x)fp(x) - 400

x200 = fzero(h, 30) % 27.1296

x200 = 27.1296

To fit a polynomial of degree \( n \), which in standard form is written as
\[
    f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0
\]
a system of \((n+1)\) linear equations must be solved. To set up the equations coordinates of all available data points have to be substituted into the standard form. In practice it is not advisable to solve this system, especially for higher degree polynomials, since often we get a badly conditioned coefficient matrix. We may use Matlab’s special command to compute the Vandermonde matrix. Both give the same results:

\[ A = [v.^4 \ v.^3 \ v.^2 \ v.^1 \ v.^0] \] % usual specification

\[
A =
\begin{bmatrix}
  234256 & 10648 & 484 & 22 & 1 \\
  1508025 & 42875 & 1225 & 35 & 1 \\
  5308416 & 110592 & 2304 & 48 & 1 \\
  13845841 & 226981 & 3721 & 61 & 1 \\
  29986576 & 405224 & 5476 & 74 & 1 \\
\end{bmatrix}
\]

\[ A = \text{vander}(v) \] % alternative: Vandermonde matrix

\[
A =
\begin{bmatrix}
  234256 & 10648 & 484 & 22 & 1 \\
  1508025 & 42875 & 1225 & 35 & 1 \\
  5308416 & 110592 & 2304 & 48 & 1 \\
  13845841 & 226981 & 3721 & 61 & 1 \\
  29986576 & 405224 & 5476 & 74 & 1 \\
\end{bmatrix}
\]

\[ \text{cond}(A) \] % 1.5378e+09

\[ \text{ans} = 1.5378e+09 \]

\[ x = A \backslash p \] % 0.0001; -0.0171; 0.5627; 12.0190; -62.0517

\[ x =
\begin{bmatrix}
  0.0001 \\
  -0.0171 \\
  0.5627 \\
  12.0190 \\
  -62.0517 \\
\end{bmatrix}
\]

**Lagrange and Newton interpolating polynomials**

There is only one interpolating polynomial for a given set of data points, but there are several different formulations. Let us briefly review two other forms besides the general one that are often better, easier to use and do not lead to a badly conditioned system. One such formulation is due to Lagrange, the other is to Newton.
Lagrange interpolating polynomial

This polynomial can be specified directly from the coordinates of points without any computation or solution of a system of equations, as given below, from \( n \) points:

\[
 f(x) = \sum_{i=1}^{n} y_i L_i(x) = \sum_{i=1}^{n} y_i \prod_{j=1 \atop j \neq i}^{n} \left( \frac{x-x_j}{x_i-x_j} \right)
\]

where \( L_i(x) = \prod_{j=1 \atop j \neq i}^{n} \left( \frac{x-x_j}{x_i-x_j} \right) \) are called Lagrange functions. For two points:

\[
 f(x) = \frac{(x-x_2)}{(x_1-x_2)} y_1 + \frac{(x-x_1)}{(x_2-x_1)} y_2,
\]

for 3 points:

\[
 f(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} y_3.
\]

- It is clear that the interpolation polynomial can be specified without any calculation, from coordinates of data points only.
- It is uneasy to work with the formula since for each interpolation point the whole equation must be written again, unlike the standard form where we need to substitute the coefficients only.
- If a new point is added to the set of points, all Lagrange functions have to be recomputed. This is different from Newton's form where only some new terms must be computed when a new point is added.

Newton interpolating polynomial

Newton's polynomial can recursively be written with so called divided differences. The general form of the polynomial is

\[
 f(x) = a_1 + a_2 (x-x_1) + a_3 (x-x_1)(x-x_2) + \cdots + a_n (x-x_1)(x-x_2)\cdots(x-x_{n-1}).
\]

Coefficients can be computed from divided differences. Generally a \( k \)-th degree divided difference:

\[
 f[x_{i+k},\ldots,x_i] = \frac{f[x_{i+k},\ldots,x_{i+1}]-f[x_{i+k-1},\ldots,x_i]}{x_{i+k}-x_i}, \quad (k = 1, 2, \ldots, n) \text{ is } (i = 0, \ldots, n-k).
\]

- It is clear that in this case the interpolation polynomial can also be specified without any calculation, from coordinates of data points only.
- It is not so inconvenient to work with, since if coefficients are already computed these can be reused later for interpolation of any point.
- If a new point is added to the set of points, we do not need to start from scratch, only a new coefficient have to be calculated. Hence it is easy to add a new point and points do not
have to be ordered.

**Interpolation of reservoir water height-area-volume curves**

Let us consider a problem in hydraulic engineering. A basic problem in planning of water storage reservoirs is the morphological characteristic curve of the reservoir, which gives volume and area variation as a function of water height. We have the following data:

<table>
<thead>
<tr>
<th>Water level H [cm]</th>
<th>Volume V [10^6 m^3]</th>
<th>Area F [km^2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>336</td>
<td>0.16</td>
<td>0.05</td>
</tr>
<tr>
<td>504</td>
<td>0.36</td>
<td>0.09</td>
</tr>
<tr>
<td>714</td>
<td>0.79</td>
<td>0.19</td>
</tr>
<tr>
<td>976</td>
<td>1.73</td>
<td>0.37</td>
</tr>
<tr>
<td>1302</td>
<td>3.31</td>
<td>0.62</td>
</tr>
<tr>
<td>1628</td>
<td>5.83</td>
<td>0.90</td>
</tr>
<tr>
<td>1812</td>
<td>7.72</td>
<td>1.05</td>
</tr>
<tr>
<td>1932</td>
<td>8.98</td>
<td>1.16</td>
</tr>
<tr>
<td>2142</td>
<td>11.50</td>
<td>1.27</td>
</tr>
</tbody>
</table>

Let us make a water height – volume graph as usual by plotting volume on the horizontal, height on the vertical axis. Interpolate volume for water level of 15 m and conversely, water level belonging to 12 million m^3 volume.

Load data from `reservoir.txt` and make a plot

```matlab
clc; clear all; close all;
data = load('reservoir.txt')
```

```matlab
data =
1.0e+03 *
0.3360 0.0002 0.0001
0.5040 0.0004 0.0001
0.7140 0.0008 0.0002
0.9760 0.0017 0.0004
1.3920 0.0033 0.0006
1.6280 0.0058 0.0009
1.8120 0.0077 0.0011
1.9320 0.0090 0.0012
2.1420 0.0115 0.0013

H = data(:,1); % cm
V = data(:,2); % million m^3
F = data(:,3); % km^2
figure(1)
plot(V,H,'*'); hold on;
ylabel('water level [cm]')
xlabel('water volume [10^6 m^3]')```
For our 9 data points fit an 8 degree polynomial:

\[ n = \text{length}(V) \mod 9 \]

\[ n = 9 \]

\[ a1 = \text{polyfit}(V,H,n-1) \]

\textbf{Warning: Polynomial is badly conditioned. Add points with distinct X values, reduce the degree.}

\[ a1 = 1.0e+03 \]

\[ -0.0000 0.0004 -0.0075 0.0649 -0.3178 0.8842 -1.3829 1.3947 0.1448 \]

\[ p1 = @(x) \text{polyval}(a1,x) \]

\[ p1 = \text{function_handle with value:} \]

\[ @(x)\text{polyval}(a1,x) \]

\[ g = \text{fplot}(p1, [\theta, \text{max}(V)]) \]

\[ g = \text{FunctionLine with properties:} \]

- \text{Function:} @(x)\text{polyval}(a1,x)
  - \text{XRange:} [0 11.5000]
  - \text{Color:} [0.8500 0.3250 0.0980]
  - \text{LineStyle:} '-'
  - \text{LineWidth:} 0.5000

\text{Show all properties}

\[ \text{set}(g, 'Color', 'r') \]
Is this curve acceptable for interpolation? Certainly it is not. This is over-fitting. In this case a high-degree polynomial reproduces data perfectly but there are oscillations in between (especially at the two sides). The interpolation deviates significantly from the trend of data, hence it is unsuitable for interpolation or extrapolation. This wavy, oscillating behavior is called Runge effect. For few data points when degree of the polynomial is low interpolating polynomials are of good use, but for many points and high degree polynomials we must look for another solution.

Incidentally, Matlab issues a warning after running polyfit:

Warning: Polynomial is badly conditioned. Add points with distinct X values, reduce the degree of the polynomial, or try centering and scaling as described in HELP POLYFIT.

When we ask for the condition number of Vandermonde matrix the answer is a very big number (order of $10^{10}$) which indicates a badly conditioned matrix.

\[
\begin{align*}
A &= \text{vander}(V); \\
\text{cond}(A) &\approx 2.7260\times10^{10} \\
\text{ans} &= 2.7260\times10^{10}
\end{align*}
\]

**Spline interpolation**

Generally a **spline** of degree $k$ and order $m$ is an interpolation function where piecewise polynomials of degree $k$ are fitted and these have order $m$ continuity at the joints. One of the most
frequently used spline is cubic spline interpolation of second order. In this case third degree polynomials are fitted between data points that have common derivatives up to second order at the joints. There is a cubic spline interpolation of first order as well. In this case also third degree polynomials are fitted, but only the first derivatives are required to be the same (e.g. Hermite polynomial interpolation).

**Linear spline interpolation**

Let us fit linear functions between data points. If \( n \) points are given there are \( n-1 \) intervals and so \( n-1 \) lines have to be determined. Matlab function `interp1` can be used for spline interpolation. It has a parameter for specifying which kind of spline interpolation is required. If not specified it defaults to linear interpolation. Possible interpolation methods for `interp1` are: 'linear' – default, 'nearest' – nearest neighbor, 'cubic' – cubic first order spline, 'spline' – cubic second order spline.

```matlab
figure(2);plot(V,H,'r*');hold on
ylabel('water level [cm]');
sp = @(x) interp1(V,H,x)

sp = function_handle with value:
 @(x)interp1(V,H,x)

g=fplot(sp,[0,max(V)])

g =
 FunctionLine with properties:

  Function: @(x)interp1(V,H,x)
  XRange: [0 11.5000]
  Color: [0 0.4470 0.7410]
  LineStyle: '-'
  LineWidth: 0.5000

Show all properties

xlabel('water volume \([10^6 \text{ m}^3]\)');
set(g,'Color','k')
```

Linear function fit gives a continuous interpolation, however its slope will have sudden changes. If smoother interpolation is required a higher order spline must be used.

**Quadratic spline interpolation**

Quadratic (second degree) spline interpolation works by fitting quadratic polynomials into given data points and prescribing even continuity of first order derivatives computed from left or right.

We must specify \( (3n-4) \) equations for quadratic spline fitting. Cubic second order splines are used more frequently, partly because also second derivatives (curvatures) are continuous hence interpolation is also smoother, and partly because there is a formulation where only \( (n-2) \) equations have to be solved. Matlab has no built-in functions for quadratic splines, only for cubics.
Cubic second order spline interpolation

This is one of the most frequently used form of spline interpolation. In this case third degree polynomials are fitted into the given points such that both first and second order derivatives are continuous at the joints. We have \((n-1)\) intervals in case of \(n\) points and for these intervals cubic polynomials have to be determined. Since each cubic polynomial has 4 unknown coefficients, there are \(4(n-1)=4n-4\) unknowns for which we must prescribe equations or conditions.

One possibility is to set second derivatives to 0 at the endpoints, this spline is called natural cubic spline. Another possibility that also used by Matlab is the “not-a-knot” condition. This means the continuity of third order derivatives at the second and next to the last points.

Let us make cubic third degree spline interpolation of characteristic curve of the reservoir in Matlab. We will use `interp1` function again, but in this case we change the default ‘linear’ method to ‘spline’.

```matlab
sp2 = @(x) interp1(V,H,x,'spline')

sp2 = function_handle with value:
 @(x)interp1(V,H,x,'spline')

g=fplot(sp2,[0,max(V)])

g =
 FunctionLine with properties:

    Function: @(x)interp1(V,H,x,'spline')
    XRange: [0 11.5000]
      Color: [0.8500 0.3250 0.0980]
  LineStyle: '-'
     LineWidth: 0.5000

Show all properties

set(g,'Color','r')
```
sp2 = @(x) spline(V,H,x)

sp2 = function_handle with value:
@(x)spline(V,H,x)

% What water height corresponds to 12 million m^3 volume?
H15 = sp2(12) % 2174 cm

H15 = 2.1741e+03

% what volume belongs to 1500 cm water height?
f = @(x) sp2(x)-1500

f = function_handle with value:
@(x)sp2(x)-1500

V1500 = fzero(f,5) % 4.6699 million m^3

V1500 = 4.6699

**Cubic first order spline interpolation**

Matlab has another kind of cubic spline interpolation. The function `interp1` may as well be called with method 'cubic'. This is cubic first order interpolation where still cubic polynomials are fitted inside each interval but only first order continuity is required. This is cubic Hermite interpolation.

Let us consider an example where it may be useful.
Our problem now relates to geodesy. To compute the Earth's gravity field we must know the Earth's internal density distribution. Earth's density ($\rho$) changes in terms of the radius (R) approximately like this:

<table>
<thead>
<tr>
<th>Radius [km]</th>
<th>0</th>
<th>800</th>
<th>1200</th>
<th>1400</th>
<th>2000</th>
<th>3000</th>
<th>3400</th>
<th>3600</th>
<th>4000</th>
<th>5000</th>
<th>5500</th>
<th>6370</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density [kg/m$^3$]</td>
<td>13000</td>
<td>12900</td>
<td>12700</td>
<td>12000</td>
<td>11650</td>
<td>10600</td>
<td>9900</td>
<td>5500</td>
<td>5300</td>
<td>4750</td>
<td>4500</td>
<td>3300</td>
</tr>
</tbody>
</table>

Let us make a spline fit of the radius-density data and interpolate Earth's density at the radius of 3200 km, and radius where the density is exactly 4000 kg/m$^3$. First load earth_density.txt file and make cubic second order and cubic first order spline interpolation. Compare both interpolations in terms of best fit.

```matlab
clear all; close all
data = load('earth_density.txt')

r = data(:,1) % radius in km
ro = data(:,2) % density in kg/m^3
```
% cubic first order Hermite interpolation
fsp1 = @(x) interp1(r,ro,x,'cubic') % or 'pchip'

fsp1 = function_handle with value:
 @(x) interp1(r,ro,x,'cubic')

fsp1(3200) % 10361

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
ans = 1.0361e+04

figure(1); subplot(1,2,1);
plot(r,ro,'r*'); hold on;
fplot(fsp1,[0,max(r)]);

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.

title('Cubic first order Hermite interpolation')
% cubic second order spline interpolation
fsp2 = @(x) spline(r,ro,x)

fsp2 = function_handle with value:
    @(x)spline(r,ro,x)

fsp2(3200) % 11350

ans = 1.1350e+04

subplot(1,2,2)
plot(r,ro,'r*'); hold on;
fplot(fsp2,[0,max(r)]);
title('Cubic second order spline interpolation')
The densities given by the two methods differ by much for radius of 3200 km, the first yielded 10361 kg/m3, the second gave 11350 kg/m3.

It is clear from the plots that cubic first order spline ('cubic') gave a much better fit than cubic second order spline ('spline'). It is because there are strong discontinuities, jumps in the data. When interpolating smooth functions, however, second order splines yield better results.

Hence to answer the second question we should use our first spline to interpolate radius belonging to 4000 kg/m3 density.

```
% What is the radius for 4000 kg/m^3 density?
f = @(x) fsp1(x)-4000

f = function_handles with value:
 @(x)fsp1(x)-4000

R4000 = fzero(f,5500) % 5958.9 km
```

Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
Warning: INTERP1(...,'CUBIC') will change in a future release. Use INTERP1(...,'PCHIP') instead.
So approximately at 6000 km radius will density be 4000 kg/m³.