12. Curve Fitting and Interpolation

2014-06-29

```
$Version
```

```
9.0 for Mac OS X x86 (64-bit) (January 24, 2013)
```

Often it is necessary to find mathematical expressions for data given as discrete points only:

$$\{\{x_1,y_1\},\{x_2,y_2\}, ..., \{x_s,y_s\}\}.$$

There are several methods available for performing this task. One may either choose to have

1. one expression defined over the whole interval containing all the points: This is done by **fits** or **interpolating polynomials;**

   or

2. one may proceed such that different expressions are sought for the different intervals between the points; the objects so

   obtained are called **interpolating function objects** or **splines**.

12.1 Fitting Functions to Given Data

A function f(x) is defined as a linear sum of given functions $\phi_i(x)$ with unknown coefficients c_i :

$$f(x) := c_1\,\phi_1(x) + c_2\,\phi_2(x) + \ldots + c_n\,\phi_n(x)\,. \qquad (12.1)$$

The number of functions, n, must not be larger than the number of data pairs, s. In many cases it is be smaller. Setting y_i = f(x_i) for all pairs given above gives an (in most cases overdetermined) system of linear equations for the unknowncoefficients c_i . $\{y_1 , y_2 , ..., y_s\}$ is called the **response vector.** The values $\phi_r(x_k)$ are the elements of the **design matrix.** The latter is rectangular, in general. The number of columns is the number of functions, n. The number of rows is the number of points, s. Multiplying the response vector with the pseudoinverse (cf. § 8.9) of the design matrix gives the unknown coefficients c_i .

Fit[*data, funs, vars*]	finds a least square fit to a list of *data* as a linear combination of functions ***funs*** of variables in the list ***vars***

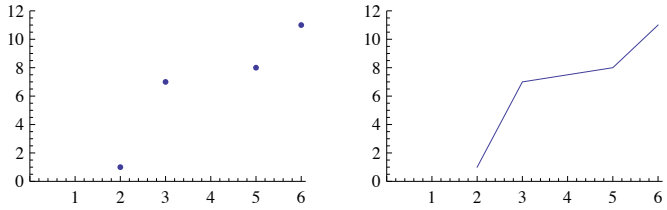
The function **Fit[]** finds a least-squares fit of data to a linear combination of user selected basis functions.

```
Clear[data];
data = {{2, 1}, {3, 7}, {5, 8}, {6, 11}}; TableForm[data]
```

```
2    1
3    7
5    8
6    11
```

```
Clear[d, f, p]
d = ListPlot[data, PlotRange → {{0, 6.1}, {0, 12}}, Prolog → PointSize[0.015`]];
c = ListPlot[data, PlotRange → {{0, 6.1}, {0, 12}}, Joined → True];
```

```
Show[GraphicsRow[{d, c}]]
```



```
f = Fit[data, {1, x}, x]
```

$-1.65 + 2.1\,x$

```
pl = Plot[f, {x, 0, 6}, PlotRange -> {{0, 6}, {0, 12}}];
```

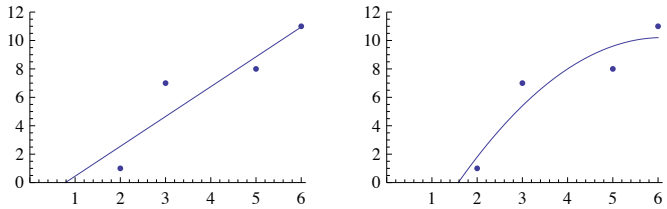
```
Clear[f]; f = Fit[data, {1, x, x^2}, x]
```

$-8.4 + 6.1\,x - 0.5\,x^2$

```
pq = Plot[f, {x, 0, 6}, PlotRange -> {{0, 6}, {0, 12}}];
```

```
ppl = Show[d, pl]; ppq = Show[d, pq];
```

```
Show[GraphicsRow[{ppl, ppq}]]
```

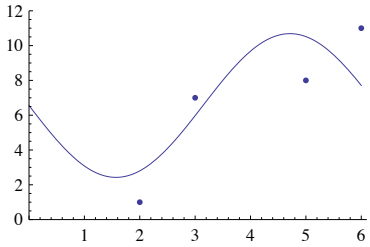


```
Clear[f]
f = Fit[data, {1, Sin[x]}, x]
```

$6.556 - 4.12927\,\text{Sin}[x]$

```
p = Plot[f, {x, 0, 6}, PlotRange -> {{0, 6}, {0, 12}}];
```

```
Show[d, p, PlotRegion -> {{.01, .95}, {.01, .95}}, ImageSize → 200]
```



Comparison of the various curves produced, shows that their shapes depend very strongly on the functions one assumes. In particular, the extrapolation, i.e. the evaluation of values outside the interval, where the data were given, is extremely sensitive to these assumptions. Some general knowledge on the behaviour of the function one wants to obtain is necessary to get meaningful results.

FindFit[*data, expr,par, var*] **_*expr*_**	finds numerical values of the parameters ***par*** so that gives a best fit to ***data***. ***var*** lists the variable(s)

```
Clear[a, b, c, d, t, x, y]
FindFit[data, a + b t, {a, b}, t]
```

```
{a → -1.65, b → 2.1}
```

```
data2 = Partition[{Range[Length[data]], Transpose[data]} // Flatten, 3]
```

```
{{1, 2, 3}, {4, 2, 3}, {5, 6, 1}, {7, 8, 11}}
```

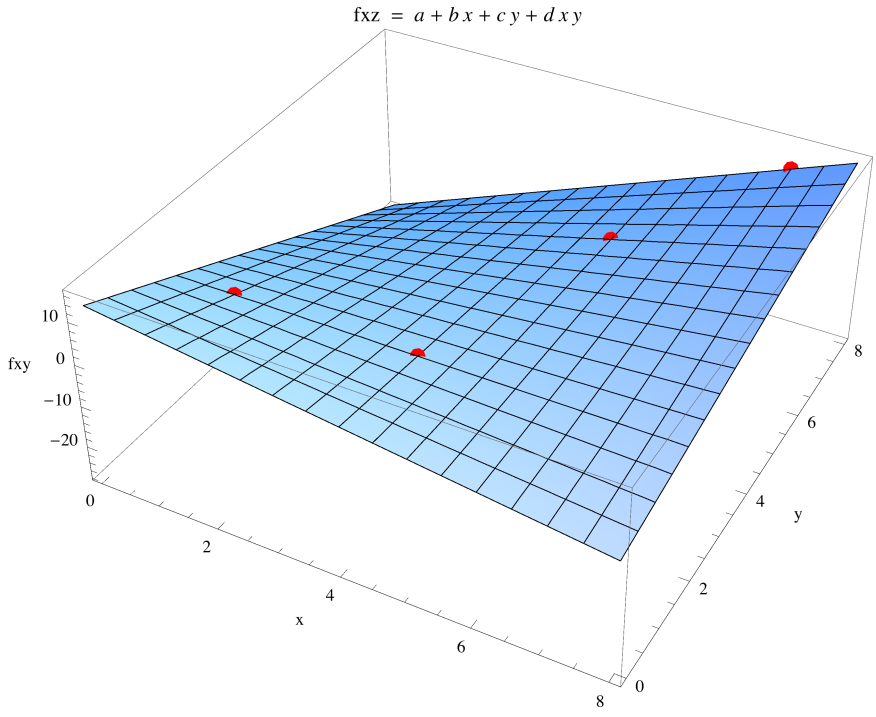
```
Map[Point, data2]
```

```
{Point[{1, 2, 3}], Point[{4, 2, 3}], Point[{5, 6, 1}], Point[{7, 8, 11}]}
```

```
fxy = a + b x + c y + d x y;
con = FindFit[data2, fxy, {a, b, c, d}, {x, y}]
```

```
{a → 13.1667, b → -1.83333, c → -5.08333, d → 0.916667}
```

```
p1 = Plot3D[Evaluate[fxy /.con], {x, 0, 8}, {y, 0, 8},
    AxesLabel → {"x", "y", "fxy"}, PlotLabel → Row[{"fxz  =  ", fxy}]];
p2 = Show[p1, Graphics3D[{PointSize[0.02`], Hue[0], Map[Point, data2]}],
  ImageSize → 450]
```



12.2 Interpolation by Polynomials

InterpolatingPolynomial[*data, var*] which provides	gives a polynomial in the variable ***var***
	an exact fit to the data given in the list ***data***

```
Clear[c, f, g, p, x, y];
data = {{2, 1}, {3, 7}, {5, 8}, {6, 11}}; TableForm[data]
```

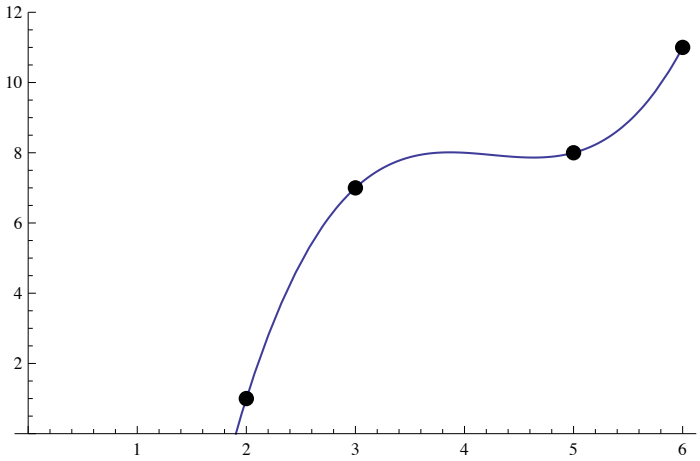
```
2    1
3    7
5    8
6    11
```

```
poi = {AbsolutePointSize[8], Map[Point, data]};
```

```
f = InterpolatingPolynomial[data, x] // Expand
```

$$-42+\frac{215\,x}{6}-\frac{17\,x^2}{2}+\frac{2\,x^3}{3}$$

```
Plot[f, {x, 0, 6}, PlotStyle → Thickness[0.003],
 PlotRange → {0, 12}, Epilog → poi]
```



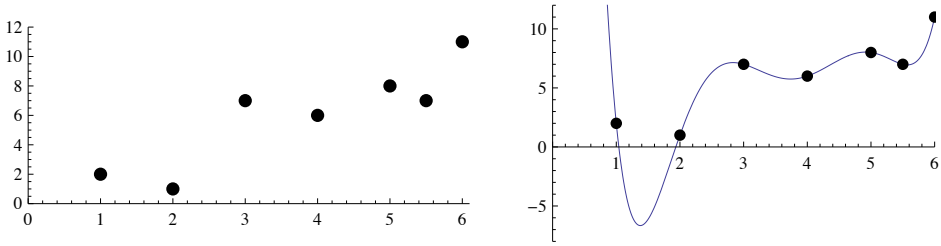
For these four given points the interpolating polynomial gives a fit which is very satisfactory from an esthetic point
of view. However, for larger sets of data points the curve representing the interpolating polynomials may oscillate
very strongly between the data points.

```
data1 = {{1, 2}, {2, 1}, {3, 7}, {4, 6}, {5, 8}, {5.5, 7}, {6, 11}};
{PointSize[0.03], Map[Point, data1]};
d =
  Graphics[%, Axes → True, PlotRange → {{0, 6.1}, {0, 12}}, AspectRatio → 0.4];
```

```
g[x_] = Expand[InterpolatingPolynomial[data1, x]]
```

$$276.762-641.633\,x+554.785\,x^2-234.611\,x^3+52.3519\,x^4-5.92222\,x^5+0.267725\,x^6$$

```
p = Plot[g[x], {x, 0, 6}, PlotRange → {{0, 6}, {-8, 12}}, Epilog → d[[1]]];
Show[GraphicsRow[{d, p}], ImageSize →  500]
```



So one may prefer to use again least square fits employing a few powers only.

```
f = Fit[data1, {1, x}, x]
```

$-0.240614 + 1.64846\,x$

```
g = Fit[data1, {1, x, x^2}, x]
```

$-0.0689979 + 1.5206\,x + 0.0180373\,x^2$

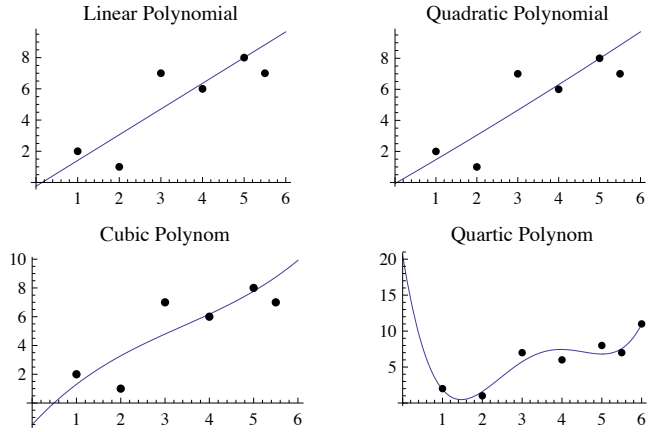
```
h = Fit[data1, {1, x, x^2, x^3}, x]
```

$-1.50006 + 3.3173\,x - 0.577117\,x^2 + 0.0569129\,x^3$

```
i = Fit[data1, {1, x, x^2, x^3, x^4}, x]
```

$20.5384 - 34.2162\,x + 19.342\,x^2 - 4.06955\,x^3 + 0.291998\,x^4$

```
pf = Plot[f, {x, 0, 6}, PlotLabel → "Linear Polynomial"];
pg = Plot[g, {x, 0, 6}, PlotLabel → "Quadratic Polynomial"];
ph = Plot[h, {x, 0, 6}, PlotLabel → "Cubic Polynom"];
pi = Plot[i, {x, 0, 6}, PlotLabel → "Quartic Polynom"];
pdf = Show[pf, d]; pdq = Show[pg, d];
pdh = Show[ph, d]; pdi = Show[pi, d];
Show[GraphicsGrid[{{pdf, pdq}, {pdh, pdi}}]]
```



12.3 Interpolation

```
?? Interpolation
```

Interpolation[{f_1, f_2, ...}] constructs an interpolation
of the function values f_i, assumed to correspond to x values 1, 2, ...
Interpolation[{{x_1, f_1}, {x_2, f_2}, ...}] constructs an interpolation of the
function values f_i corresponding to x values x_i.
Interpolation[{{{x_1, y_1, ...}, f_1}, {{x_2, y_2, ...}, f_2}, ...}] constructs an interpolation of multidimensional data.
Interpolation[{{{x_1, ...}, f_1, df_1, ...}, ...}] constructs
an interpolation that reproduces derivatives as well as function values.
Interpolation[*data*, x] find an interpolation of *data* at the point x. ≫

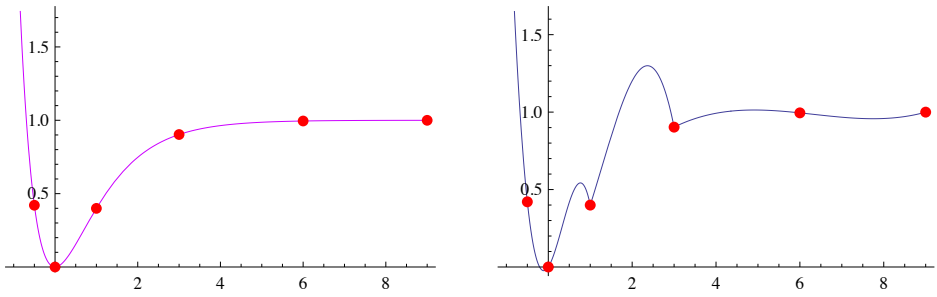
```
Attributes[Interpolation] = {Protected}

Options[Interpolation] =
 {InterpolationOrder → 3, Method → Automatic, PeriodicInterpolation → False}
```

Function values and derivatives may be real or complex, or arbitrary symbolic expressions. The x_i must be real numbers.
Interpolation works by fitting polynomial curves between successive data points. The degree of these is specified by the
option `InterpolationOrder;` whose default values is 3. The precision of the InterpolatingFunction object is that of the data.
Note: Lists comprising the data and the derivatives of the function(s) have been changed in going from *Mathematica*5.2
to *Mathematica7.0* !

```
data = {-1, -0.5, 0, 1, 3, 6, 9}; Clear[lx, f]
```

$f(x\_) = (e^{-x} - 1)^2;$

```
i1 = Interpolation[d1 = {data, f[data]} // Transpose];
epi = {Epilog → {Hue[1], PointSize[.025], Map[Point , d1]}};

p1 = Plot[f[x], {x, -1, 9}, Evaluate[epi], PlotStyle → Hue[0.8`]];
p2 = Plot[i1[t], {t, -1, 9}, Evaluate[epi]];
Show[GraphicsRow[{p1, p2}], ImageSize → 500]
```



Using derivatives in interpolation

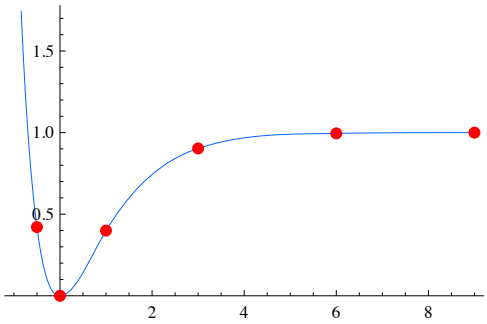
Comparison of the two drawings above shows that the interpolating function gives a curve lacking smoothness.
This can be improved by presribing also derivatives at the points.

```
d2 = Table[{{data[[k]]}, f[data[[k]]], f'[data[[k]]]} // N, {k, Length[data]}]
```

```
{{{-1.}, 2.95249, -9.34155}, {{-0.5}, 0.420839, -2.13912},
 {{0.}, 0., 0.}, {{1.}, 0.399576, 0.465088}, {{3.}, 0.902905, 0.0946166},
 {{6.}, 0.995049, 0.00494522}, {{9.}, 0.999753, 0.000246789}}
```

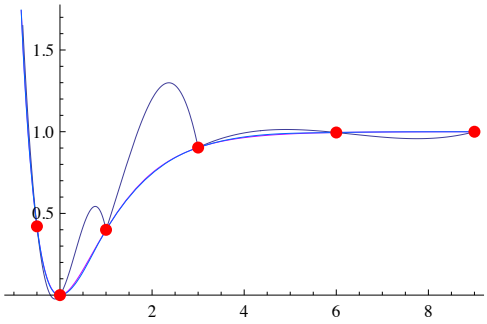
```
Clear[i2, t]
i2[t_] = Interpolation[d2][t];
```

```
p3 = Plot[i2[t], {t, -1, 9}, Evaluate[epi], PlotStyle → Hue[0.6], ImageSize →  250]
```



Alternatively, one can use splines.

```
Show[p1, p2, p3, ImageSize → 250]
```



FunctionInterpolation[*expr*, {*x*, *xmin*, *xmax*}] evaluates *expr* with *x* running from *xmin* to *xmax* and constructs an **InterpolatingFunction** object which represents an approximate function corresponding to the result.

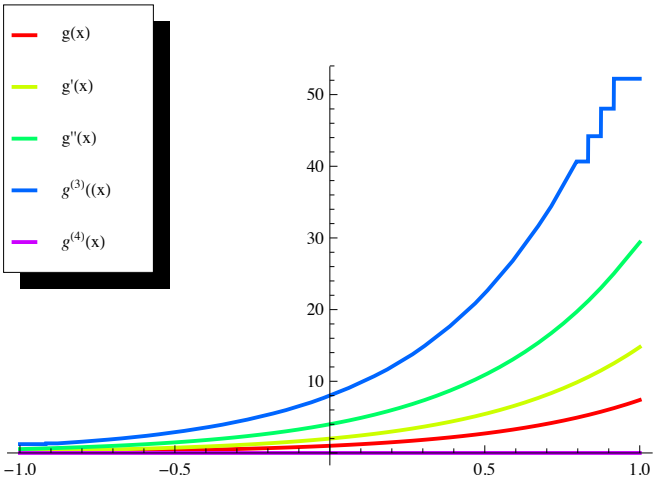
Simple Interpolation

In general, **InterpolatingFunction** objects are continuous but not smooth, as in the following example:

```
f[x_] = Exp[2x];
```

```
g = FunctionInterpolation[f[x],{x,-1,1}] ;
```

```
Plot[Evaluate[Table[D[g[x], {x, n}], {n, 0, 4}]], {x, -1, 1}, PlotRange → All,
 ImageSize → 350, PlotStyle → Table[{Thick, Hue[n/5]}, {n, 0, 4}],
 PlotLegend → {"g(x)", "g'(x)", "g''(x)", "g⁽³⁾((x)", "g⁽⁴⁾(x)"},
 LegendPosition → {-1, 0}]
```



It can be seen that these **InterpolatingFunction** objects are piecewise polynomial functions.

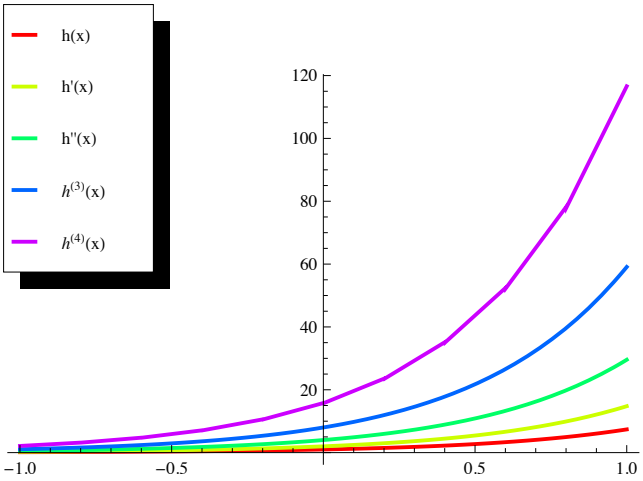
Smooth Interpolation

If the derivatives of the function are also available, one can use **FunctionInterpolation[]** as follows:

```
h = FunctionInterpolation[{f[x],f'[x],f''[x]},{x,-1,1}]
```

```
InterpolatingFunction[{{-1., 1.}}, <>]
```

```
Plot[Evaluate[Table[∂{x,n} h[x], {n, 0, 4}]], {x, -1, 1}, PlotRange → All,
 ImageSize → 350, PlotStyle → Table[{Thick, Hue[n/5]}, {n, 0, 4}],
 PlotLegend → {"h(x)", "h'(x)", "h''(x)", "h⁽³⁾(x)", "h⁽⁴⁾(x)"},
 LegendPosition → {-1, 0}]
```



In general, the derivatives of an **InterpolatingFunction** object are continuous up to the minimum order

which is contained in its data.

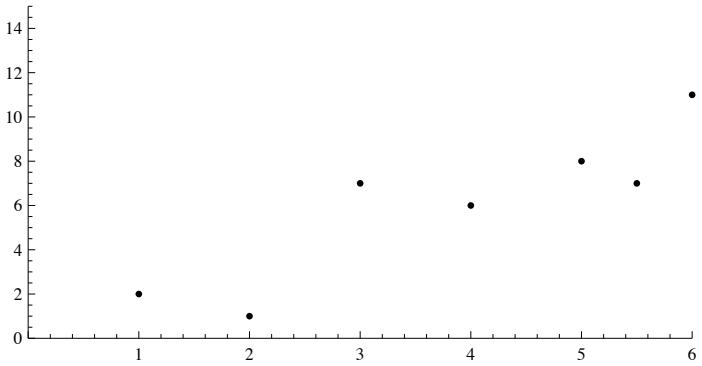
Robert Knapp, rknapp@wolrfram.com
TMJ, 8 (#3, 2002) 394/5.

12.4 Approximation by Spline Functions

Splines are polynomials interpolating between two data points. Depending on the degree of the spline also continuity
of higher derivatives is prescriped. In particular, cubic splines pass through all the points, they have continuous first
and second derivatives at all inner points. The second derivative is zero at the end points.
There are two packages for splines: Graphics`Spline` and NumericalMath`SplineFit`. The first package calls and
uses the second one.

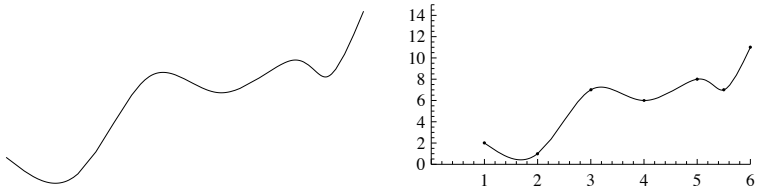
12.4.1 Drawing a spline curve through the data points.

```
<< Splines`;

data1 = {{1, 2}, {2, 1}, {3, 7}, {4, 6}, {5, 8}, {5.5`, 7}, {6, 11}};
d = Graphics[{PointSize[0.01], Map[Point, data1]}, Axes → True,
  PlotRange → {{0, 6}, {0, 15}}, AspectRatio →  0.5]
```



Much more information and types of splines can be found at Help/ Find Selected Function /Splines
.

```
sp = Show[Graphics[Spline[data1, Cubic]], AspectRatio → 0.5];
dp = Show[d, sp];
Show[GraphicsRow[{sp, dp}], ImageSize → 400]
```



12.4.2 Computing and using a spline

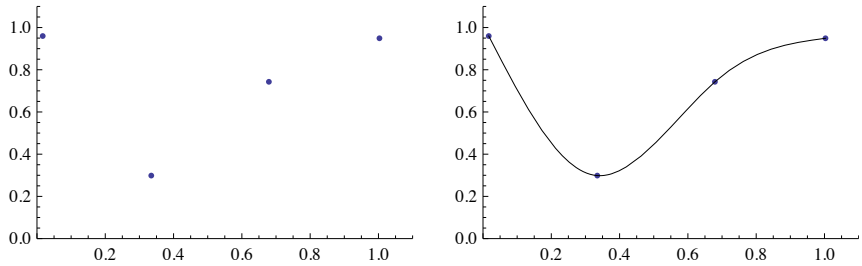
COMPATIBILITY ISSUE

The data points need not be equidistant. For this demonstration we choose 4 data points:

```
np =  4 ; (*  = number of data points *)

data = Table[{x + RandomReal[] 0.1`, RandomReal[]}, {x, 0, 1, 1/(np - 1)}]
```

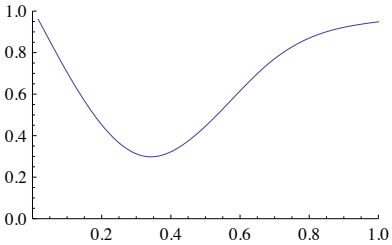
```
{{0.0168349, 0.95978}, {0.334696, 0.298598},
 {0.67912, 0.742919}, {1.0029, 0.949139}}
```

```
dp = ListPlot[data, PlotRange → {{0, 1.1`}, {0, 1.1`}}, Prolog → PointSize[0.02`]];
sp = Graphics[Spline[data, Cubic]];
psd = Show[dp, sp, Axes → True];

Show[GraphicsRow[{dp, psd}], ImageSize → 450]
```



A spline function object is obtained by the following command. It depends on a parameter, which runs
from 0 to np. It may be used to plot a curve and to get values:

```
ip = SplineFit[data, Cubic];

pip = ParametricPlot[ip[u], {u, 0, np - 1},
  PlotRange → {0, 1}, ImageSize →  200, AspectRatio → 0.6]
```



Gives the same curve as above at the right. These are the coordinates of a point on the spline curve:

```
ip[2.23]
```

```
{0.75587, 0.833355}
```

14.4.2.1 The splinefunction object

```
ip = SplineFit[data, Cubic];
fu = FullForm[ip]
```

```
SplineFunction[Cubic, List[0.`, 3.`],
 List[List[0.016834882778006554`, 0.9597796814213382`],
  List[0.33469596654690364`, 0.2985979572886228`],
  List[0.6791201672510354`, 0.7429185474703968`],
  List[1.0028963642106037`, 0.9491392007773733`]],
 List[List[List[0.016834882778006554`, 0.3094010523365304`,
    -1.1102230246251565`*^-16, 0.008460031432366788`],
   List[0.9597796814213382`, -0.9718556704082324`,
    1.5265566588595902`*^-16, 0.31067394627551687`]],
  List[List[0.33469596654690364`, 0.33478114663363057`, 0.02538009429710053`,
    -0.015737040226599386`], List[0.2985979572886228`,
    -0.03983383158168145`, 0.9320218388265509`, -0.44786741706309546`]],
  List[List[0.6791201672510354`, 0.3383302145480336`, -0.02183102638269796`,
    0.007277008794232653`], List[0.7429185474703968`,
    0.48060759488213384`, -0.4115804123627359`, 0.13719347078757865`]]]]
```

The first argument characterizes the spline, the second one gives the range of the parameter, the third one
the data points. The fourth argument gives for each of the np - 1 intervals bounded by two adjacent data points
two lists; these are the coefficients a0, a1, a2, a3 and b0, b1, b2, b3
of cubic polynomials representing the spline curve in parametric form:

$$\{x[t] = a0 + a1\, t + a2\, t^2 + a3\, t^3,\ y[t] = b0 + b1\, t + b2\, t^2 + b3\, t^3\},\quad 0 \le t \le 1.$$

```
ip[[4]] // Chop
```

```
{{{0.0168349, 0.309401, 0, 0.00846003}, {0.95978, -0.971856, 0, 0.310674}},
 {{0.334696, 0.334781, 0.0253801, -0.015737},
  {0.298598, -0.0398338, 0.932022, -0.447867}},
 {{0.67912, 0.33833, -0.021831, 0.00727701},
  {0.742919, 0.480608, -0.41158, 0.137193}}}
```

```
tl = Table[t^n, {n, 0, 3}]
```

$$\{1,\ t,\ t^2,\ t^3\}$$

```
px = Table[ip[[4, n, 1]].tl, {n, np - 1}] // Chop
```

$$\{0.0168349 + 0.309401\, t + 0.00846003\, t^3,\ 0.334696 + 0.334781\, t + 0.0253801\, t^2 - 0.015737\, t^3,\ 0.67912 + 0.33833\, t - 0.021831\, t^2 + 0.00727701\, t^3\}$$

```
py = Table[ip[[4, n, 2]].tl, {n, np - 1}] // Chop
```

$$\{0.95978 - 0.971856\, t + 0.310674\, t^3,\ 0.298598 - 0.0398338\, t + 0.932022\, t^2 - 0.447867\, t^3,\ 0.742919 + 0.480608\, t - 0.41158\, t^2 + 0.137193\, t^3\}$$

Properties of the splines

The above polynomials are used to show the properties of the splines at the data points.
They are continuous at the inner points:

```
{px[[1]] , py[[1]] } /. t → 0
```

```
{0.0168349, 0.95978}
```

```
Table[{(px[[k + 1]] /. t → 0) - (px[[k]] /. t → 1),
    (py[[k + 1]] /. t → 0) - (py[[k]] /. t → 1)} // Chop, {k, np - 2}]
```

```
{{0, 0}, {0, 0}}
```

```
{px[[-1]] , py[[-1]] } /. t → 1
```

```
{1.0029, 0.949139}
```

The first derivatives are continuous at the inner points:

```
{D[px[[1]], t] , D[py[[1]], t] } /. t → 0 // Chop
```

```
{0.309401, -0.971856}
```

```
Table[{(D[px[[k + 1]] , t] /. t → 0) - (D[px[[k]], t] /. t → 1),
    (D[py[[k + 1]], t] /. t → 0) - (D[py[[k]], t] /. t → 1)} // Chop, {k, np - 2}]
```

```
{{0, 0}, {0, 0}}
```

```
{D[px[[-1]], t] , D[py[[-1]], t] } /. t → 1 // Chop
```

```
{0.316499, 0.0690272}
```

The second derivatives are continuous at the inner points, zero at the end points:

```
{D[px[[1]], {t, 2}] , D[py[[1]], {t, 2}] } /. t → 0 // Chop
```

```
{0, 0}
```

```
Table[{(D[px[[k + 1]] , {t, 2}] /. t → 0) - (D[px[[k]], {t, 2}] /. t → 1),
    (D[py[[k + 1]], {t, 2}] /. t → 0) -
     (D[py[[k]], {t, 2}] /. t → 1)} // Chop, {k, np - 2}]
```

```
{{0, 0}, {0, 0}}
```

```
{D[px[[-1]], {t, 2}] , D[py[[-1]], {t, 2}] } /. t → 1 // Chop
```

```
{0, 0}
```

The first derivative of the spline curve

The first derivative is dy/dx = $\dot{y}[t]/\dot{x}[t]$; the corresponding fractions are combined with x[t] so that we get a parametric representation of the derivative of the spline curve:

```
fr = D[py, t] / D[px, t]
```

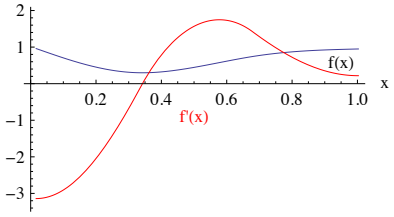
$$\left\{\frac{-0.971856 + 0.932022\ t^2}{0.309401 + 0.0253801\ t^2},\ \frac{-0.0398338 + 1.86404\ t - 1.3436\ t^2}{0.334781 + 0.0507602\ t - 0.0472111\ t^2},\ \frac{0.480608 - 0.823161\ t + 0.41158\ t^2}{0.33833 - 0.0436621\ t + 0.021831\ t^2}\right\}$$

```
fd[u_] = Apply[Which, Flatten[Table[{k - 1 ≤ u ≤ k,
        {px[[k]], fr[[k]]} /. t → u - k + 1}, {k, np - 1}] ] // ExpandAll // Together,
```

$$\text{Which}\left[0 \le u \le 1,\ \left\{0.0168349 + 0.309401\ u + 0.00846003\ u^3,\ \frac{36.7226\ \left(-1.04274 + 1.\ u^2\right)}{12.1907 + 1.\ u^2}\right\},\right.$$
$$1 \le u \le 2,\ \left\{0.041032 + 0.23681\ u + 0.0725912\ u^2 - 0.015737\ u^3,\right.$$
$$\left.\frac{28.4594\ \left(2.41699 - 3.38735\ u + 1.\ u^2\right)}{-5.01598 - 3.07517\ u + 1.\ u^2}\right\},\ 2 \le u \le 3,$$
$$\left.\left\{-0.14308 + 0.512978\ u - 0.0654931\ u^2 + 0.00727701\ u^3,\ \frac{18.853\ \left(9.16771 - 6.\ u + 1.\ u^2\right)}{23.4977 - 6.\ u + 1.\ u^2}\right\}\right]$$

```
pd = ParametricPlot[fd[u], {u, 0, np - 1}, PlotStyle → Hue[0]];
```

```
Show[pip, pd, AxesLabel → {"x", ""}, PlotRange → {-3.5`, 2.1`},
 Epilog → {Text["f(x)", {0.95`, 0.6`}], Hue[0], Text["f'(x)", {0.5`, -0.9`}]}]
```



Second derivatives may be found from the formula: $d^2y/dx^2 = (\dot{x}\,\ddot{y} - \ddot{x}\,\dot{y})/\dot{x}^3$.

12.5 Approximation by Chebyshev Expansions

A very economic method to represent functions is by expansions with respect to Chebyshev polynomials $T_n(x)$ (the German transliteration is: Tschebischeff Polynome):

$$T_n(x) : T_n(\cos\vartheta) = \cos(n\,\vartheta), \quad x = \cos\vartheta\,, \; -1 \le x \le 1;$$

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_2(x) = x^2 - 1/2\,, T_3(x) = x^3 - 3/4\,x, \ldots.$$

These fulfil the following orthonormality relation:

$$\int_{-1}^{1} dx\, T_n(x)\, T_k(x) / \sqrt{(1 - x^2)} = \delta_{nk}\,(1 + \delta_{n0})\,\pi/2.$$

$T_n(x)$ is the polynomial of n-th degree with leading coefficient 1 whose maximum absolute value is smallest in
the intervall $-1 \le x \le 1$; i.e. it deviates least from zero. In series expansions the shifted Chebyshev polynomials are needed defined as:

$$T^*_n(x) = T_n(2x - 1), \; 0 \le x \le 1;$$

with the following orthonormality properties:

$$\int_0^1 dx \; T^*_n(x)\, T^*_k(x) / \sqrt{(x - x^2)} = \delta_{nk}\,(1 + \delta_{n0})\,\pi/2.$$

The corresponding series exansion is:

$$f(x) = \Sigma\; c_n\; T^*_n(x), \quad 0 \le x \le 1.$$

The advantage of the Chebyshev expansion is that it gives an approximation more uniform over the intervall, while Taylor's expansion is very accurate in the neighbourhood of the expansion point but becomes the less accurate the larger the distance from this point. This is shown in a simple example (due to Lanczos) where comparison with the exact result is easy.

```
f = 1 / (1 + x);
```

```
f6 = Series[f, {x, 0, 6}] // Normal
```

$$1 - x + x^2 - x^3 + x^4 - x^5 + x^6$$

```
f7 = Series[f, {x, 0, 7}] // Normal
```

$$1 - x + x^2 - x^3 + x^4 - x^5 + x^6 - x^7$$

The expansion coefficients for the Chebyshev series are found as is usual for a complete orthonormal system. The necessary integrations are done by numeric integration. The factor 1/2 for $n = 0$ is taken into account in a second step.

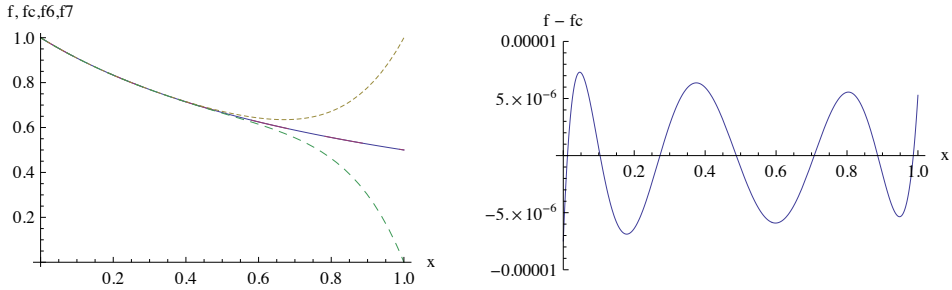
```
cn = Table[ NIntegrate[ChebyshevT[n, 2 x - 1] / ((1 + x) Sqrt[x (1 - x)]), {x, 0, 1}],
                {n, 0, 6}] 2 / Pi // N ;
cn[[1]] = cn[[1]] / 2; cn
```

```
{0.707107, -0.242641, 0.0416306,
 -0.00714267, 0.00122549, -0.000210261, 0.000036075}
```

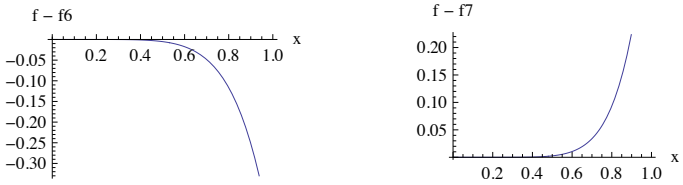
```
fc = Expand[Sum[ cn[[k]] ChebyshevT[k - 1, 2 x - 1], {k, Length[cn]}]];  fc
```

$$0.999993 - 0.99922\,x + 0.986378\,x^2 - 0.907076\,x^3 + 0.675347\,x^4 - 0.329299\,x^5 + 0.0738817\,x^6$$

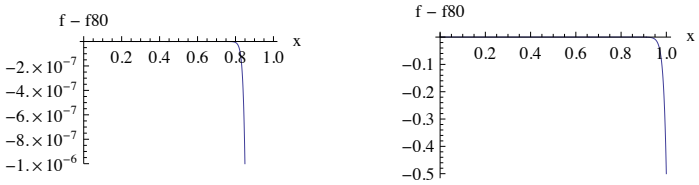
```
SetOptions[Plot, DisplayFunction → Identity];
pt = Plot[{f, fc, f6, f7}, {x, 0, 1}, AxesLabel → {"x", "f, fc,f6,f7"}, PlotStyle →
    {Dashing[{}], Dashing[{0.1`}], Dashing[{0.01`}], Dashing[{0.02`}]}];
pc = Plot[{fc - f}, {x, 0, 1}, PlotRange → {-1, 1}/10^5, AxesLabel → {"x", "f - fc"}];
Show[GraphicsRow[{pt, pc}], ImageSize → 500]
```




```
p6 = Plot[{f - f6}, {x, 0, 1}, AxesLabel → {"x", "f - f6"}];
p7 = Plot[{f - f7}, {x, 0, 1}, AxesLabel → {"x", "f - f7"}];
Show[GraphicsRow[{p6, p7}, Spacings → {Scaled[0.5`], Scaled[0]}]]
```




```
f80 = Normal[Series[f, {x, 0, 80}]];
SetOptions[Plot, AxesLabel → {"x", "f - f80"}];
pt = Plot[{f - f80}, {x, 0, 1}, PlotRange → {0, -1/10^6}];
pa = Plot[{f - f80}, {x, 0, 1}, PlotRange → All];
Show[GraphicsRow[{pt, pa}, Spacings → {Scaled[0.5`], Scaled[0]}]]
```



Note that 80 (!) terms of the Taylors expansion give high precision up to x ≤ .8, but thereafter the accuracy deterioriates rapidly. - In this simple example the expansion coefficients can be found by analytic integration; in this way one gets:

```
Clear[p]
fc = Sqrt[2] (1/2 + Sum[(-p)^n ChebyshevT[n,2x-1],{n,6}]);
p = 3 - 2 Sqrt[2] ; N[p]
tc = N[fc]//Expand
```

```
0.171573
```

$$0.999993 - 0.99922\,x + 0.986378\,x^2 - 0.907076\,x^3 + 0.675347\,x^4 - 0.329299\,x^5 + 0.0738817\,x^6$$

```
Chop[fc - tc, 10^-8]
```

$$-0.999993 + 0.99922\,x - 0.986378\,x^2 + 0.907076\,x^3 - 0.675347\,x^4 + 0.329299\,x^5 - 0.0738817\,x^6 + \sqrt{2}\left(\frac{1}{2} - \left(3-2\sqrt{2}\right)(-1+2x) + \left(3-2\sqrt{2}\right)^2\left(-1+2(-1+2x)^2\right) - \left(3-2\sqrt{2}\right)^3\left(-3(-1+2x)+4(-1+2x)^3\right) + \left(3-2\sqrt{2}\right)^4\left(1-8(-1+2x)^2+8(-1+2x)^4\right) - \left(3-2\sqrt{2}\right)^5\left(5(-1+2x)-20(-1+2x)^3+16(-1+2x)^5\right) + \left(3-2\sqrt{2}\right)^6\left(-1+18(-1+2x)^2-48(-1+2x)^4+32(-1+2x)^6\right)\right)$$

```
dn = Table[
Integrate[ChebyshevT[n,2x-1]/((1+x) Sqrt[x(1-x)]),
    {x,0,1}], {n,0,6}] 2/Pi ;
dn[[1]] = dn[[1]]/2; Cancel[dn]
N[dn]
```

$$\left\{\frac{1}{\sqrt{2}}, 4-3\sqrt{2}, -24+17\sqrt{2}, 140-99\sqrt{2}, -816+577\sqrt{2}, 4756-3363\sqrt{2}, -27720+19601\sqrt{2}\right\}$$

```
{0.707107, -0.242641, 0.0416306,
 -0.00714267, 0.00122549, -0.000210261, 0.000036075}
```

```
Prepend[Table[Sqrt[2] (-p)^k // Expand, {k, 6}], 1/Sqrt[2]]
```

$$\left\{\frac{1}{\sqrt{2}}, 4-3\sqrt{2}, -24+17\sqrt{2}, 140-99\sqrt{2}, -816+577\sqrt{2}, 4756-3363\sqrt{2}, -27720+19601\sqrt{2}\right\}$$

12.6 Nonlinear Fits

The function **Fit[]** contained in the kernel finds a least-squares fit of data to linear combination of given basis functions. Often a more complicated formula is needed, which is nonlinear in the parameters. The function **NonlinearModelFit[]** can be used for that purpose.

```
?? NonlinearModelFit
```

NonlinearModelFit[{y_1, y_2, ...}, *form*, {β_1, ...}, x] constructs a nonlinear model with structure *form* that fits the y_i for successive x values 1, 2, ... using the parameters β_1, ...
NonlinearModelFit[{{x_{11}, x_{12}, ..., y_1}, {x_{21}, x_{22}, ..., y_2}, ...}, *form*, {β_1, ...}, {x_1, ...}] constructs a nonlinear model where *form* depends on the variables x_k.
NonlinearModelFit[*data*, {*form*, *cons*}, {β_1, ...}, {x_1, ...}] constructs a nonlinear model subject to the parameter constraints *cons*. ≫

```
Attributes[NonlinearModelFit] = {Protected, ReadProtected}
```

```
data = {{0, 1}, {1, 0}, {3, 2}, {5, 4}, {6, 4}, {7, 5}};
```

```
nlm = NonlinearModelFit[data, Log[a + b x^2], {a, b}, x]
```

```
FittedModel[ Log[1.50632 + 1.42633 x^2] ]
```

Obtain the functional form:

```
Normal[nlm]
```

```
Log[1.50632 + 1.42633 x^2]
```

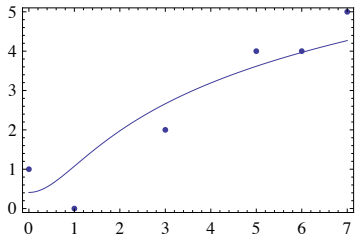
Evaluate the model at a point:

```
nlm[2.3]
```

```
2.20294
```

Visualize the fitted function with the data:

```
Show[ListPlot[data], Plot[nlm[x], {x, 0, 7}], Frame → True]
```

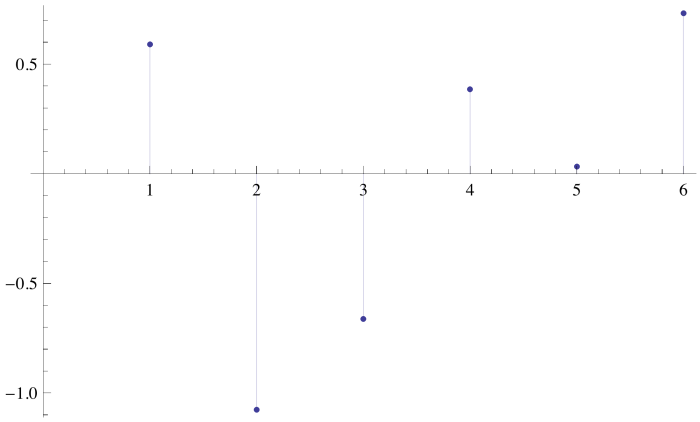


Extract and plot the residuals:

```
nlm["FitResiduals"]
```

```
{0.59033, -1.07591, -0.663282, 0.384644, 0.0324629, 0.731751}
```

```
ListPlot[%, Filling → Axis]
```



{{x11,x12,...,y1}, {x21,x22,...,y2}, ...} each sublist gives a data point.

The data set below consists of triples {x1,x2,y} describing a chemical reaction rate y as functions of two inputs x1 and x2. The list gives 5 such sets of data. The model is given by

$$y = \frac{\theta 1\, \theta 3\, x1}{x1\, \theta 1 + x2\, \theta 2 + 1.}$$

The parameters $\theta 1$, $\theta 2$, $\theta 3$ must be determined by a least-squares fit from the following data set:

```
data =

  {{1., 1., .126}, {1., 2., .076}, {2., 1., .219}, {2., 2., .126}, {.1, 0., .186}};

nlm = NonlinearModelFit[data,(θ1*θ3*x1)/(x1*θ1 + x2*θ2 + 1),{θ1, θ2, θ3},{x1,x2}]
```

$$\text{FittedModel}\left[\frac{2.44277\, x1}{1+3.13151\, x1+15.1594\, x2}\right]$$

```
Normal[nlm]
```

$$\frac{2.44277\, x1}{1 + 3.13151\, x1 + 15.1594\, x2}$$

12.7 Exercises

12.1. Find a curve composed of cubic splines for the set of data named **data** in section 12.1 .

12.2. Generate a list of 5 complex random numbers located in the square, whose corners are at the origin and at 5 + 5i.
Tranform each of this complex numbers in a pair of real numbers; each represents a point in the real plane.
Fit a polynomial of first, second, third degree and a curve of cubic splines to these points.
Draw each of these curves together with the data points.

12.3 Find a Chebyshev expansion comprising 7 terms for sin(x) in the intervall $0 \le x \le \pi/2$; compare its accuracy to that of a Taylor's expansions performed around x = 0 and $x = \pi/4$.

12.4 Fit the data given below to the implicit function 1/x = exp(b y/ x). Determine **b** such that it gives the best fit.

```
data = {{.5, -.693147}, {1.,0.}, {2.,2.7759}, {3.,6.592}};
```

Hint: Consider a function f = 1/x - exp(b y/ x) = 0. Augment data to new list data1 for {x,y,f} as input for **NonLinearModelFit[].**

12.5 Plot the surface corresponding to the final model function given at the end of sect.12.6
together with points representing the input data.
Compute the residuals and show them together with the curcumference of a square at height h.

12.6 A set of points is given in 3 dimensions. Find the best fit plane determined by these points.
The equation of this plane is: a x + b y + c z = d; find values for a, b, c, d.