

Modelling of data with Scilab

Interpolation	2
Data fitting of functions.....	2
Example 1. Linear interpolation example.....	3
Example 2. Spline interpolation.....	4
Example 3. Linear regression.....	6
Example 4. Fitting a function to data.....	7

Modelling of data with Scilab:

Interpolation

Interpolation curves go through all points in a data set. With interpolation one can estimate new data points between two adjacent original data points by plugging in a new x-coordinate that lies between two original points.

It is important that interpolation is not used to extrapolate (predict) a function. The most common ways to interpolate are :

- **Linear interpolation** (see help `interp`). Creates a linear interpolation between two adjacent points
- **Spline interpolation** (Commands '`splin`' together with '`interp`', i.e. two commands. Note '`interp`' is not the same as '`interp`', see help `splin`)
Spline interpolation fits a set of cubics through the points, using a new cubic in each interval. Both the slope and the curvature are required to be the same for the pair of cubics that join each point.

Data fitting of functions

In the case of experimental data, the measured values may have errors but the underlying function may be known from physical laws. Then one may try to fit this function to the data set. As a criterion of "goodness of fit " the least squares criterion is usually used. If the data set is large, the curve generally does not go through all points, or even any point at all. The general command is ***datafit*** (see also help `data fit`).

Fitting straight lines is so common that two special commands are available, ***regress*** and ***reglin***. The fitted curves may be used for extrapolation (prediction), but with care, as any stock market analyst or meteorologist can verify.

- ***datafit*** (see help `data fit`)
- ***regress*** and the equivalent ***reglin***.
`regress` gives the slope and intercept of a line, `reglin` also gives the standard deviation of the residuals but is otherwise equivalent to `regress`

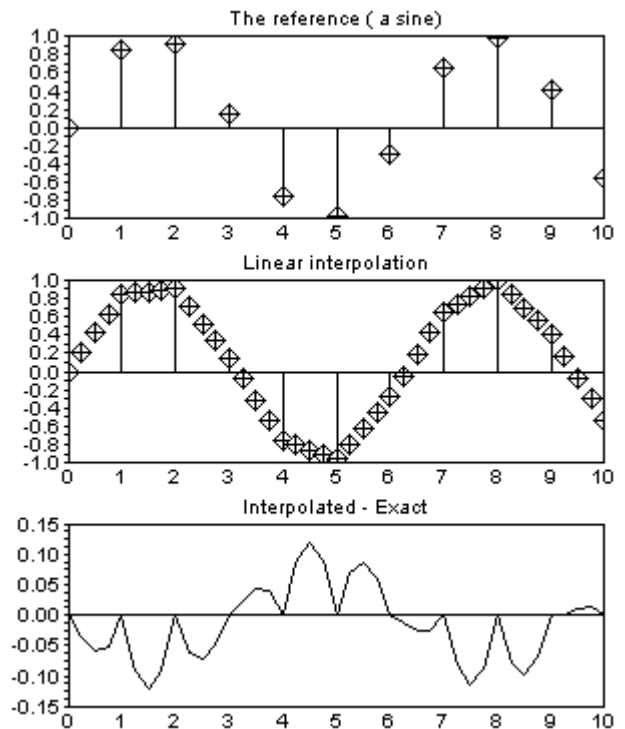
Example 1. Linear interpolation example

Linear interpolation, demo of the command

```
yi=interp1n([x;y],xi)
```

where x,y are the original data and xi, yi the interpolated data.

- The upper figure shows points from a reference signal with markers and vertical bars. The reference is a sinusoid.
- The middle figure shows a linear interpolation with four times denser points.
- The lower figure shows the difference between the interpolated and true values.



```
//Here is the script, copy into Scipad and run:
// ----- Start of code -----
// 'interp1nlinear.sce', linear interpolation demo
// * Create a reference signal with a few points and plot it
// * Interpolate it to, say, 4 times the point density and plot it
// together with the reference points
// * Compute the true points 4 times denser with the reference and
// subtract the corresponding interpolated points
// * Plot the difference to see the errors
//***** Create a reference signal *****
// A sinusoid with 11 points, one radian apart.
x=0:10; y=sin(x);
//**** Interpolate the reference points linear *****
xi=0:.25:10; // Create a 'denser' x-axis
yi=interp1n([x;y],xi); // This is the command for linear interpolation
// ***** Do some plotting *****
f1=scf(1);clf(1);
width=320; height=600; f1.figure_size=[width,height];
// We use 3 subplots for compactness
subplot(3,1,1); // Plot the reference
plot2d(x,y,style=-8) // plot reference as measurement points
plot2d3(x,y) // and also as vertical lines
plot2d(x,0*y) // Make a zero line
xlabel('The reference ( a sine)')
subplot(3,1,2); //Plot the result here
plot2d(xi,yi,style=-8); // The interpolation
plot2d3(x,y) // Also the reference for comparison
plot2d(x,0*y) // Add a zero axis
xlabel('Linear interpolation')
// We know the reference, what do the errors look like?
yy=sin(xi); // compute the reference 4 times denser.
ydiff1=yi-yy; // and take the difference
subplot(3,1,3)
plot2d(xi,ydiff1) // Plot the difference
plot2d(x,0*y) // zero line
xlabel('Interpolated - Exact')
// ----- end of code-----
```

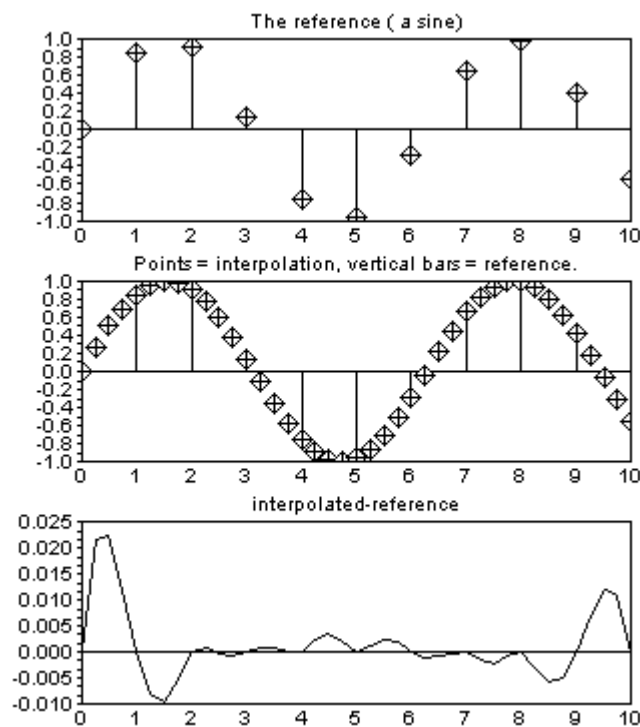
Example 2. Spline interpolation

Demo of command:

```
yd=splin(x,y); and yi=interp(xi,x,y,yd) (used together)
```

where x and y are the original data and x_i and y_i are the new interpolated values.

- The upper figure shows points from a reference signal with markers and vertical bars. The reference is a sinusoid.
- The middle figure shows an interpolation with four times denser points.
- The lower figure shows the difference between the interpolated and true values. Note start 'transients', at the endpoints derivatives are given zero value by 'spline' (typically), so the first and last segments tend to be 'wiggly'.



Here is the script, copy into Scipad and run :

```
// ----- Start of code -----
// 'interpol spline.sce', spline interpolation demo
// * Create a reference signal with a few points and plot it
// * Interpolate it to, say, 4 times the point density and plot it
//   together with the reference points
// * Compute the true points 4 times denser with the reference and
//   subtract the corresponding interpolated points
// * Plot the difference to see the errors

//***** Create a reference signal *****
// A sinusoid with 11 points, one radian apart.

x=0:10;
y=sin(x);

//**** Interpolate the reference points with splines *****
xi=0:.25:10; // Create a 'denser' x-axis
// ----- These two commands do the interpolation-----
yd=splin(x,y);
yi=interp(xi,x,y,yd); // xi and yi are the new 'denser' values

// ***** Do some plotting *****

f1=scf(1); clf(1); width=350; height=600;f1.figure_size=[width,height];
// We use 3 subplots for compactness

subplot(3,1,1); // Plot the reference
plot2d(x,y,style=-8) // plot reference as measurement points
plot2d3(x,y) // and also as vertical lines
plot2d(x,0*y) // Make a zero line
xtitle('The reference ( a sine)')

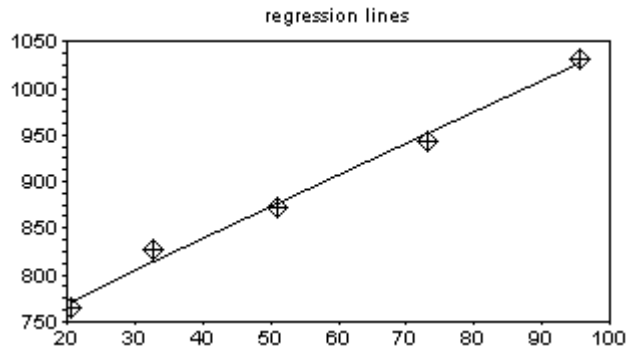
subplot(3,1,2); // Interpolation and reference
plot2d(x,0*y) // zero line
plot2d(xi,yi,style=-8); // The interpolation
plot2d3(x,y); // The reference signal
xtitle('Points = interpolation, vertical bars = reference.')
```

// We know the reference, what do the errors look like?
yy=sin(xi); // // compute the reference 4 times denser.
ydiff=yi-yy; // and take the difference

```
subplot(3,1,3); // The errorplot
plot2d(x,0*y) // zero line
plot2d(xi,ydiff)// Plot the difference
xtitle('interpolated-reference')
// ----- end of code -----
```

Example 3. Linear regression

The graph shows some 'measurement data' and the least-square line obtained with the commands **regress** and **reglin**, both equivalent, although **reglin** also outputs the standard deviation of the residuals.



Here is the script, copy and paste Scipad and run.

```
// ----- Start of code -----
// regresslines.sce , demo of linear regression
// (least square lines) with regress and reglin
// ***** Measurement data *****
x=[20.5  32.7  51.0  73.2  95.7]; // measurement x-values
y=[765   826   873   942  1032]; // corresponding y-values

// **** demo of command regress *****
coefs=regress(x,y) // linecoefficients =regress(x,y)

// Note that:
// coefs(2) is the slope of the line
// coefs(1) is the line intercept
// so the resulting line is : y = coefs(2)*x + coefs(1)

// ***** The command reglin (used the same way, gives the same + std dev) *****
[a,b,sig]=reglin(x,y)

// a slope of line
// b intercept of line
// so the resulting line is : y = a*x+b
// sig ->sigma = standard deviation of the distances from measurement y:s to the
line

// Plot
f1=scf(1); clf(1); width=350; height=600; f1.figure_size=[width,height];
plot2d(x,y,style=-8) // measurement points
plot2d(x,coefs(2)*x+coefs(1)) // the regress line
plot2d(x,a*x+b) // 'the other' reglin line, the same ...
xtitle('regression lines');

// Show that the coefficients are the same
[ coefs(2) coefs(1) a b sig]

// ----- end of code -----

// Result:
// ans =
// 3.3948725 702.17206 3.3948725 702.17206 7.9378316
```

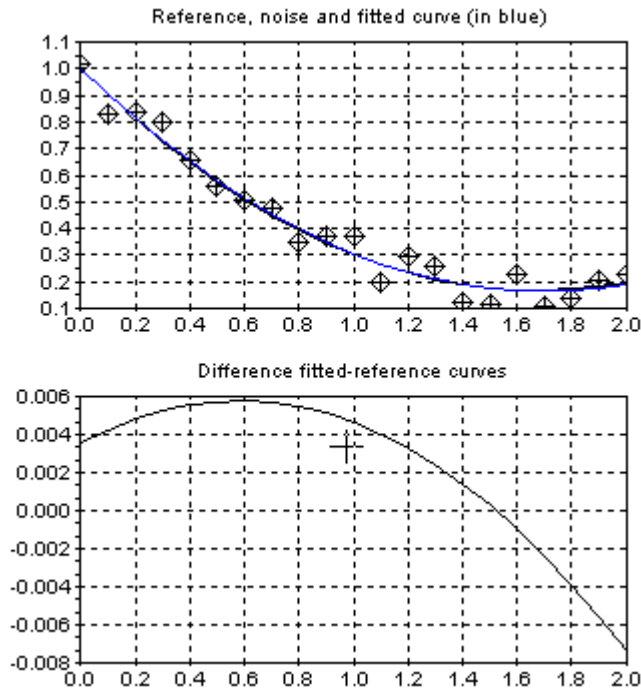
Example 4. Fitting a function to data

Fitting a second degree polynomial $y = a + b x + c x^2$
to a set of data that looks to be curved. This is a demo of the command datafit:

```
[koeff,SSE]=datafit(errors,measdata,koeff0)
```

The upper curve = dataset, reference in black and fitted curve in blue.

The lower curve = difference between fitted curve and reference (that we know in this example)



Here is the script, copy and paste into Scipad and run

```
// ----- start of code-----
// datafitpolynome.sce
// demonstrates fitting a polynome to (simulated) measurement data
//
// * Create some reference data, a disturbed polynomial of degree 2.
// * Plot the data and pretend they are measurement points
// * They look like a 2:d degree polynomial
// * Fit a 2:d degree polynomial
// * Plot the fitted polynomial together with the data
// * Since (in this example) we know the reference, compare the
// computed and known coefficients, also plot the error
//
// ----- Create reference -----
X=[0:0.1:2]; //note capital X and Y to separate them from computed x and y
Y=1-X+0.3*X .*X;
// Add some noise
rand("seed",1); // to be able to reproduce the values
noise=.15*(rand(1,length(X))-0.5);
Y1=Y+noise; // Y1 are the ' measurement data '
//
// ----- Fit a polynomial of degree 2 -----
// A second order polynomial can be written
// y = a + b*x + c*x^2
// Where datafit computes a, b och c 'as good as possible'
// The difference between each measurement and the value of the
// fitted curve at that point i is
// e(i) = Y1(i) - a - b*x(i) -c*x(i)^2 (i = an index)
```

```

//
// Since i stands for every measurement point, a large overdetermined
// system of equations has to be approximately solved. The math routines
// for this require matrix-formulation.
// Therefore indexed variables are used, for example
// z(1) for x och z(2) for Y (The choice for the name z is your own)
// Then the equation  $e = y - a - b*x - c*x^2$ 
// becomes  $e(i)=z(2)- a - b*z(1) - c*z(1)^2$ 
// The variables x and y are now in the vector z. Coeficcients a, b and c
// are called koeff, i.e. koeff(1)=a, koeff(2) = b, koeff(3) = c.
//
// Also mesurement data have to be written in matrix form, can be done like this:
// measdata=[X;Y1];
// It is practical to define the error e as a function :

function [e]=errors(koeff,z)
e=z(2)-koeff(1)-koeff(2)*z(1)-koeff(3)*z(1)^2;
endfunction

measdata=[X;Y1]; // measurement data as a matrix
koeff0=[1;2;3]; // This is a starting guess

// And here the command datafit:
[koeff,SSE]=datafit(errors,measdata,koeff0); //SSE=SumSquaredErrors

// Now the result is in koeff and the fitted polynomial becomes :
y=koeff(1)+koeff(2)*X+koeff(3)*X^2;

// ----- Do some plotting -----
f1=scf(1); clf(1); width=350; height=600; f1.figure_size=[width,height];

subplot(2,1,1) // we use subplots
plot2d(X,Y1,style=-8); // The measurements
plot2d(X,Y); // The reference curve
plot2d(X,y,style=2); //The fitted curva
xgrid(); // add grid
xlabel('Reference, noise and fitted curve (in blue) ');

// Plot the difference fitted-reference curve in subplot 2
subplot(2,1,2)
plot2d(X,y-Y);xgrid()
xlabel('Difference fitted-reference curves')

// Here we just print some values for comparison
refcoefficients=[ 1, -1, 0.3]
initialguess=[koeff0(1) koeff0(2) koeff0(3)] // Gussed start values
computedcoeffs=koeff' // Computed coeffs
SumSquaredErrors=SSE // SSE=SumSquaredErrors
//
//----- end of code -----

//----- Result: -----

// refcoefficients = ! 1. - 1. 0.3 !
// initialguess = ! 1. 2. 3. !
// computedcoeffs = ! 1.0035442 - 0.9923257 0.2934324 !
// SumSquaredErrors = 0.0469948

```