Tutorials and worked examples for simulation, curve fitting, statistical analysis, and plotting.
https://simfit.org.uk

Smooth interpolation of a sparse set of exact (x_i, y_i) points consists of filling in gaps where function values are changing rapidly, by adding extra points in order to create a better curve when the data are plotted or printed as a smooth curve.

There are two cases.

1. The x, y data points were generated from a function $y = f(x)$ where y is a single–valued function of x.

   This situation is encountered when a best fit curve has been obtained by evaluating a model equation at equally spaced points and has a peak or valley where closer spaced points would have been better able to capture the turning points.

2. The x, y data points were generated as the solutions to an implicit function $g(x, y) = 0$, for instance parametrically as $x(t)$, $y(t)$.

   This may be necessary when orbits of a system of differential equations have been plotted and there may be multiple values of y given x.

Of course the problem can usually be rectified by simply repeating the calculation using more function values, but it is sometimes required to be wise after the event, e.g., when preparing results for publication when parameter values are no longer available. More trivially, when preparing a graph for publication the ability to arbitrarily smooth chosen curves specified by known mathematical functions can be much appreciated.

It must be admitted that we cannot get something for nothing and that any interpolation technique must be understood and controlled otherwise spurious excursions and undulations could be introduced.

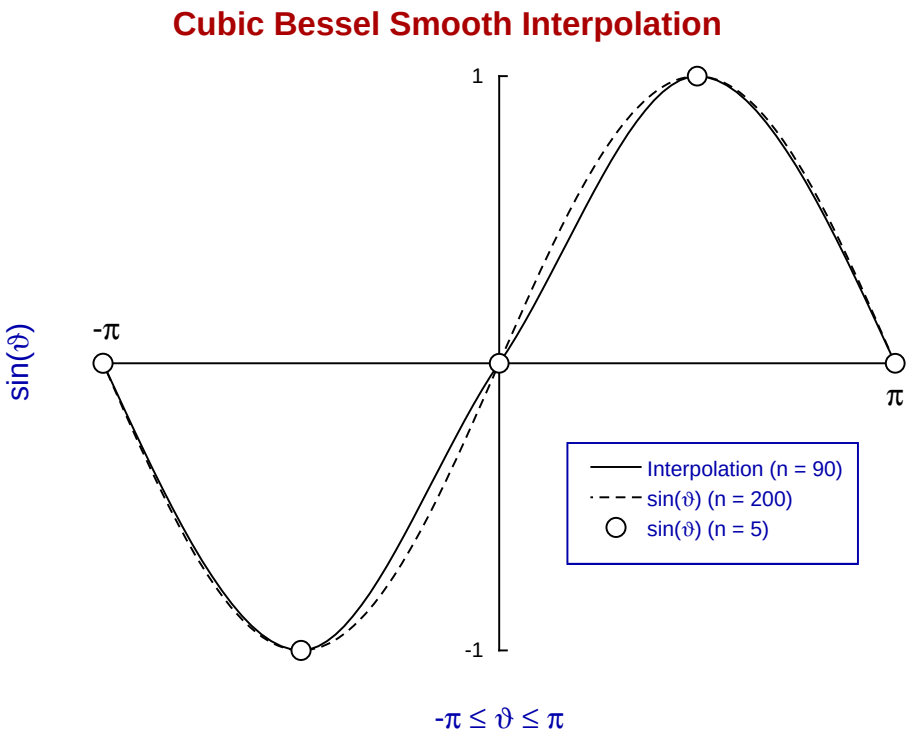
Potential problems with interpolation

Most methods for smooth interpolation use some sort of cubic polynomial smoothing and the methods provided by SimFIT are based on using cubics which are specified by treating neighboring points to generate local cubics in much the the same way as knots are used in global spline fitting.

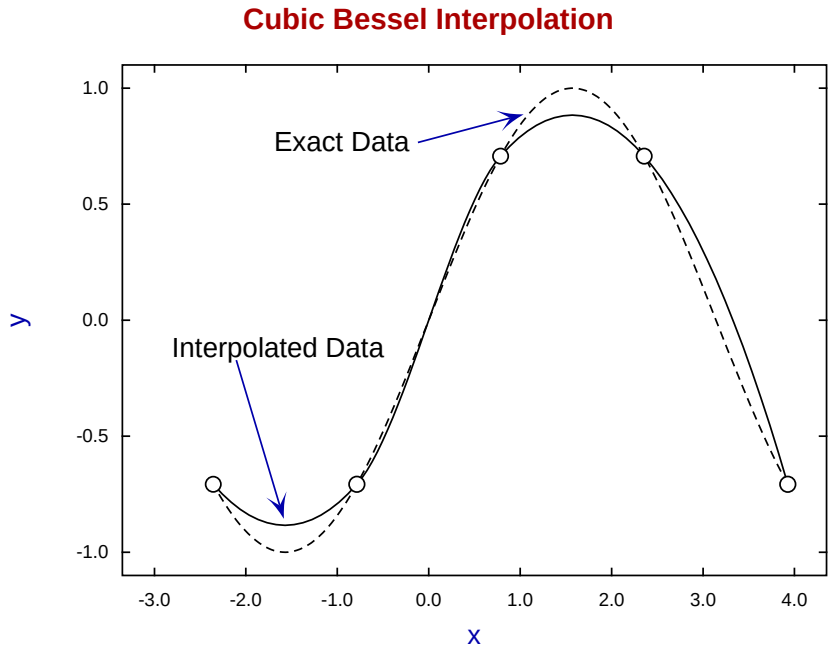
Evidentally, placing knots strategically is vital in order to avoid unwanted oscillations and we commence by using the example of a simple sine curve with the following data, which is to be smoothed using the cubic Bessel method.

x	y
-3.1415927	0.0
-1.5707963	-1.0
0.0000000	0.0
1.5707963	1.0
3.1415927	0.0

Now all computer generated curves are actually constructed by joining defined pixels by straight lines, and in the next graph we can see how the interpolated curve with 90 points generated from the original 5 five points is remarkably close to the exact curve with 200 points.

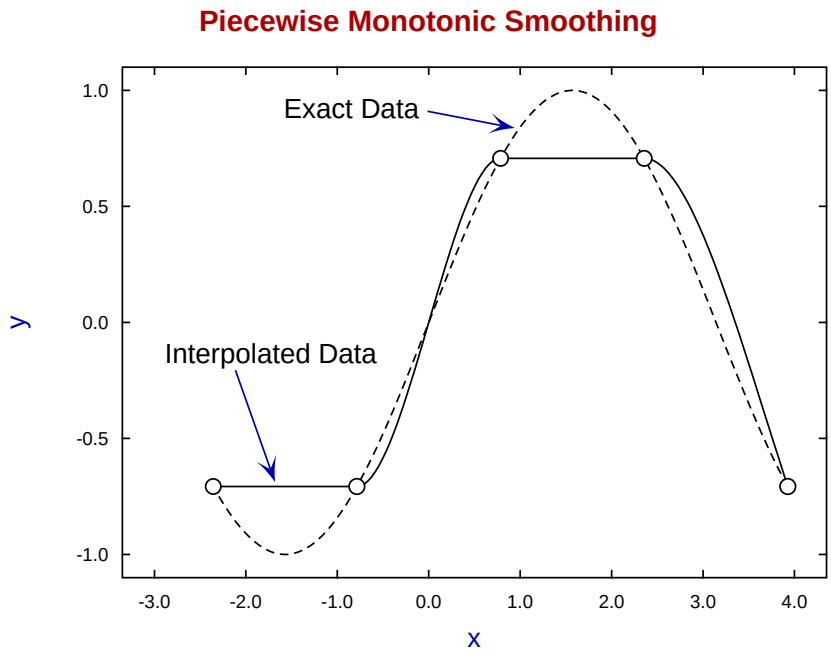


To illustrate the effect of displacing the original data from the extreme positions, consider next the result when the given x data are simply incremented by $\pi/4$. Because of this shift from the critical coordinates the next graph indicates a very much worse fit because the original data were not situated at the turning points leading to undershoot. This is one of the problems when interpolating sparse data with turning points when undershoot or overshoot can occur with the cubic Bessel technique.



As undershoot or overshoot can give a misleading impression of false turning points when observing the

closeness of data to a best–fit curve or when plotting the profile from numerical solution of differential equations, it is necessary to impose constraints by using the piecewise monotonic method which preserves the sign of $f'(x)$ between successive x values as shown next.



As this method does not allow the interpolated data to exceed obvious turning points by setting $f'(x) = 0$ at extreme y_i points it avoids overshoot and undershoot, but at the expense of straight line sections.

Example 1: smoothing a single valued function

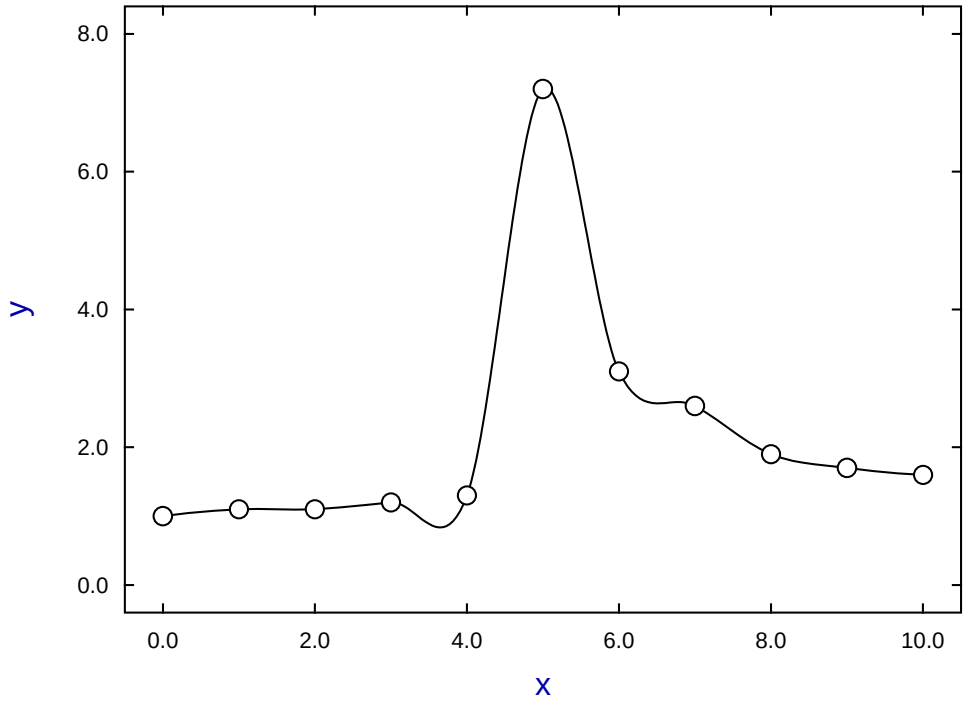
From the main SimFIT menu choose the [Statistics] option then the [Data smoothing] option and select smooth interpolation of discrete data for $y = f(x)$, which uses the following data contained in the test file `j07caf.tf1`.

x	y
0.0	1.0
1.0	1.1
2.0	1.1
3.0	1.2
4.0	1.3
5.0	7.2
6.0	3.1
7.0	2.6
8.0	1.9
9.0	1.7
10.0	1.6

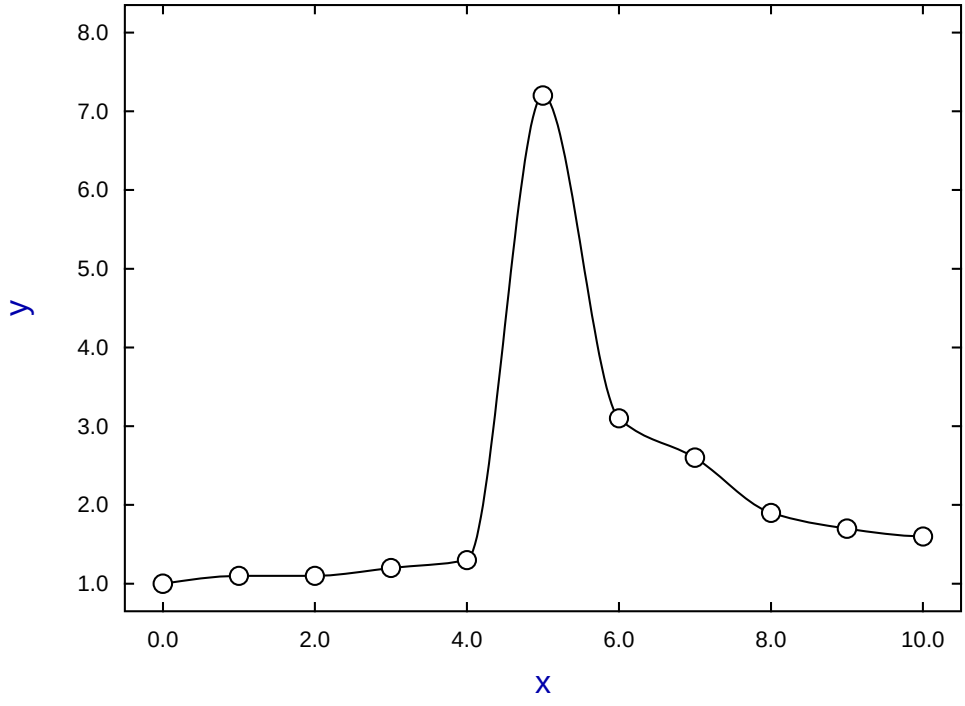
Note that you can decide whether to use the cubic Bessel or piecewise monotonic method, and also you can specify the tolerance factor. This would usually be satisfactory with the default value of 2000. Note that decreasing this factor will result in less smooth curves but with fewer extra interpolated points.

The difference between the possibly over flexible cubic Bessel result and rather stiffer piecewise monotonic curve will be clear from the next two graphs.

Cubic Bessel



Piecewise monotonic

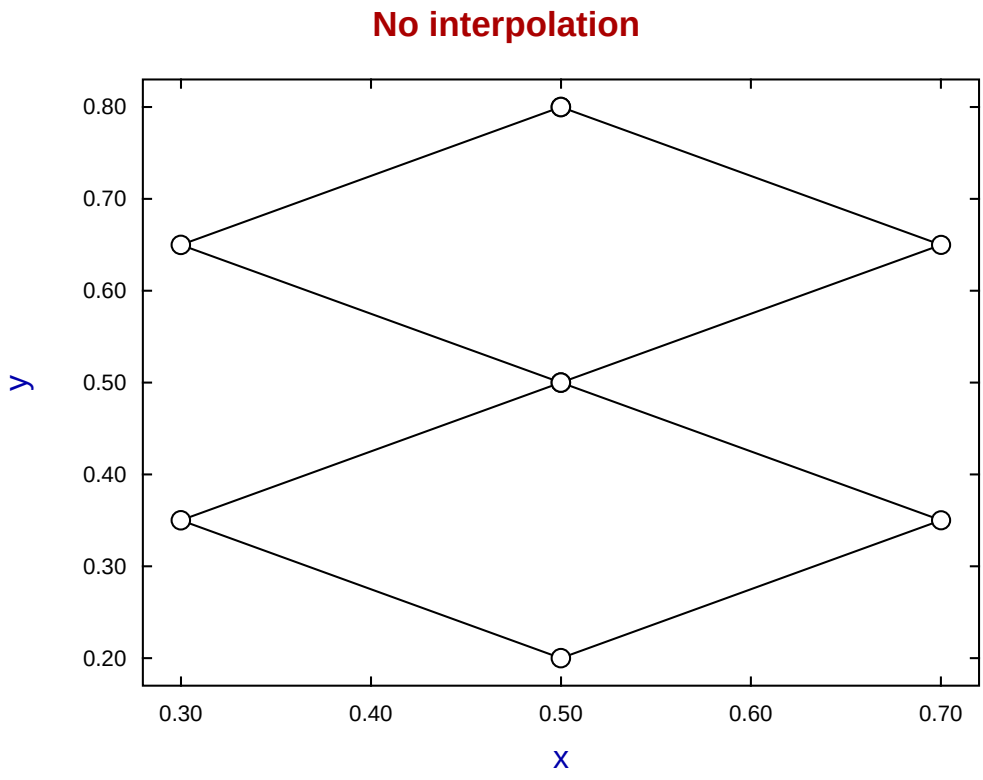


Example 2: Smoothing a parametric curve

From the main SIMFIT menu choose the [Statistics] option then the [Data smoothing] option and select smooth interpolation of discrete data for $x(t), y(t)$) which uses the following data contained in the test file `j07ccf.tf1`.

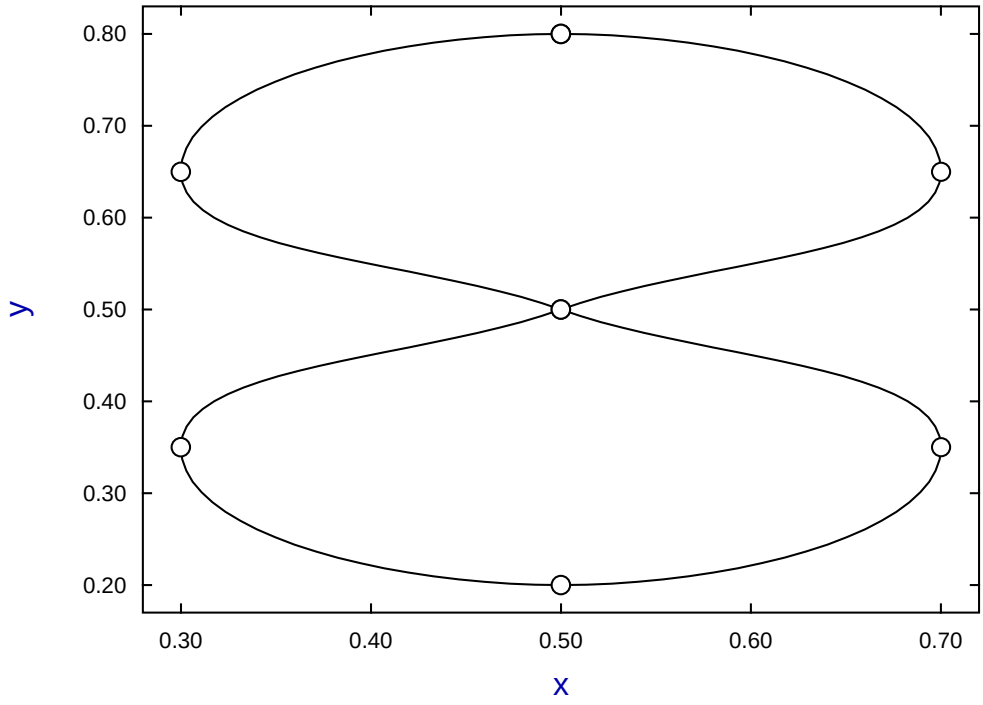
x	y
0.5	0.80
0.7	0.65
0.5	0.50
0.3	0.35
0.5	0.20
0.7	0.35
0.5	0.50
0.3	0.65
0.5	0.80

Note that these data have the last pair of coordinates equal to the first pair, as this technique is required to indicate a closed loop. Also observe that the parameter t does not occur in the data file, only the (x_i, y_i) coordinates. For instance, a circle defined parametrically by $x = r\cos\theta$, $y = r\sin\theta$ can also be expressed in the implicit form $x^2 + y^2 = r^2$ without reference to the parameter θ. First consider simply joining up these coordinates with no smoothing.

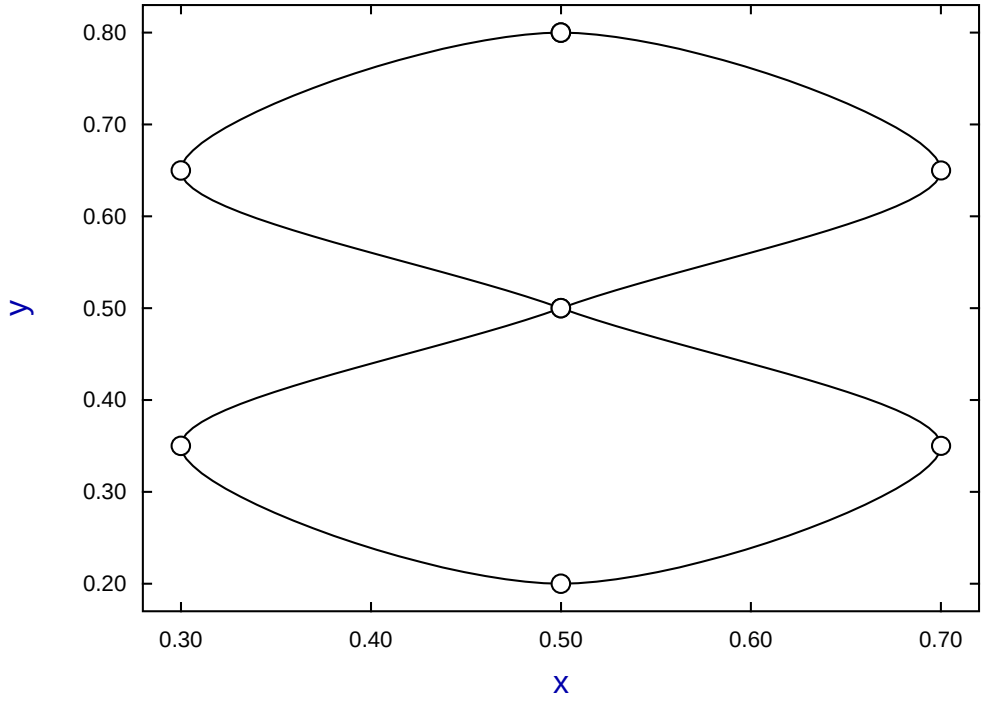


Now the McConalogue method which is analogous to the cubic Bessel providing a flexible curve and the Butland method which is similar to the piecewise monotonic method and yields a stiffer curve.

McConalogue method



Butland method



Theory

The SIMFIT program **spline** can be used to make a global spline interpolation for a single valued function but the user must then specify the additional interpolation points. The reason to use these local methods is that they can be used interactively to achieve satisfactory visual smoothness from within the SIMFIT graphics utilities. They are perfectly adequate for this purpose as will be explained elsewhere for advanced curve fitting using program **qnfit**, differential equation simulation and fitting using program **deqsol**, and advanced graph plotting and contouring using program **simplot**.

The code used is based on development of the two routines `j06caf` and `j06ccf` from the discontinued NAG graphics library which differ from the original NAG routines by supplying the discrete data on input but then returning the enlarged and interpolated data set. They can be called by programmers who use the Simdem package with the following details.

- For single valued functions

```
subroutine smooth$(n, nmax, x, y)
integer,          intent (in)    :: nmax
integer,          intent (inout) :: n
double precision, intent (inout) :: x(nmax), y(nmax)
```

  Here we must have `x` monotonically increasing (or decreasing). The value of `n` returned is increased and the coordinates returned contain the interpolated values, so the dimension `nmax` for `x` and `y` in the calling program must be large enough to contain the enlarged data set.

  For configuration use

```
subroutine j06cfg_1(isend, itolf, method)
integer, intent (in)    :: isend
integer, intent (inout) :: itolf, method
```

  where `isend = 1` sets the parameters directly otherwise an interactive interface is provided, while `method = 1` uses the piecewise monotonic method and `method = 2` uses the cubic Bessel technique.

- For parametric functions

```
subroutine contr1$(n, nmax, x, y)
integer,          intent (in)    :: nmax
integer,          intent (inout) :: n
double precision, intent (inout) :: x(nmax), y(nmax)
```

  To use the end points for a closed polygon the first and last coordinate pairs must be equal.

  For configuration use

```
subroutine j06cfg(isend, itolf, method)
integer, intent (in)    :: isend
integer, intent (inout) :: itolf, method
```

  where `isend = 1` sets the parameters directly otherwise an interactive interface is provided, while `method = 1` uses the Butland method and `method = 2` uses the McConalogue technique.

The tolerance parameter `ITOLF` controls the number of additional interpolated points required before interpolation is satisfactory, and the default value is 2000. Increasing `ITOLF` beyond this value is not likely to have much effect, while decreasing `ITOLF` down to about 200 generates fewer additional interpolation points and progressively decreases the smoothness.

This will be clear from the details as follows.

For `smooth$` the cubic $c(x)$ and piecewise linear polynomial $p(x)$ must satisfy

$$|c(x) - p(x)| \leq \frac{|YMAX - YMIN|}{ITOLF}.$$

For `contr1$` the piecewise cubic $(x(t), y(t))$ and piecewise linear polynomial $(p_x(t), p_y(t))$ must satisfy

$$|x(t) - p_x(t)| \leq \frac{|XMAX - XMIN|}{ITOLF}$$
$$\text{and } |y(t) - p_y(t)| \leq \frac{|YMAX - YMIN|}{ITOLF}.$$