

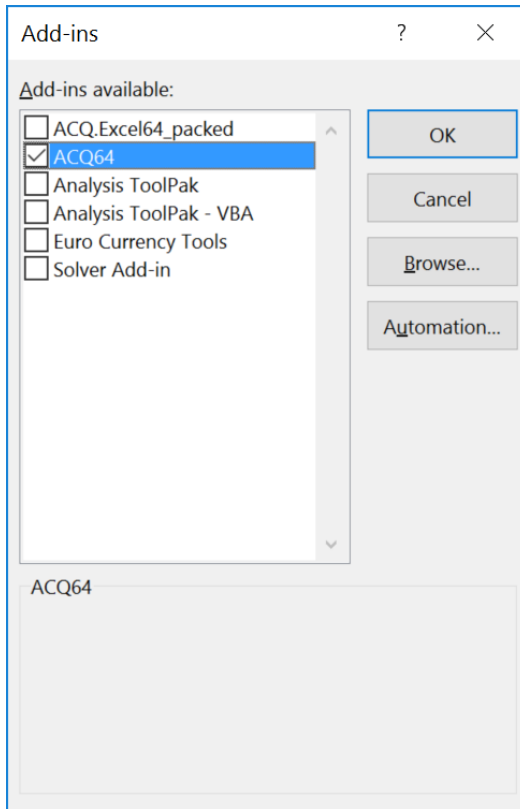
ACQ Excel Add-in

BASED ON EXCEL-DNA

Content

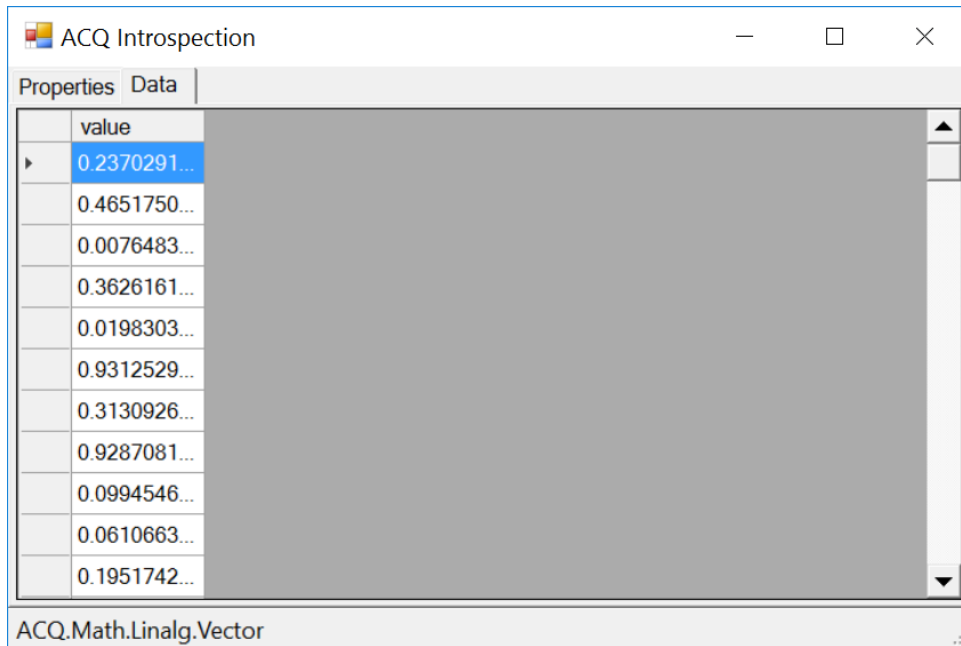
1. Installation procedure
2. Introspection and Logging
3. ACQ add-in functions:
 1. 1D Interpolation methods: Nearest, Linear, Cubic, Hermite, HermiteQS, Akima, Steffen, Multiquadrics.
 2. 2D Rectangular grid interpolation: Bilinear, BiCubic, BiHermite, BiSteffen, BiAkima
 3. Scattered data interpolation (N-D) based on radial basis functions: Linear, Cubic, Multiquadrics, Gaussian, Thinplate, InverseQuadratic, InverseMultiquadric
 4. Lowess locally-weighted linear regression
 5. Random number generator: based on Mersenne Twister (mt19937)

Installation

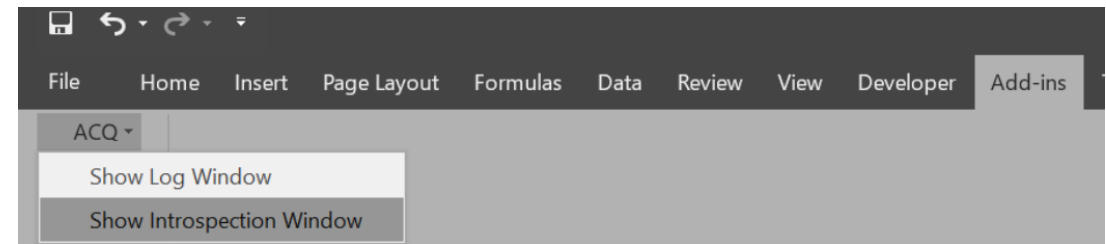


1. Start Excel. Click the File tab, click Options, and then click the Add-Ins category
2. In the Manage box, click Excel Add-ins, and then click Go. The Add-Ins dialog box appears
3. Browse to the location of the ACQ32.xll/ACQ64.xll files, and pick the xll file based on Excel bitness.
4. Make sure your Excel security settings allow you to run Add-ins

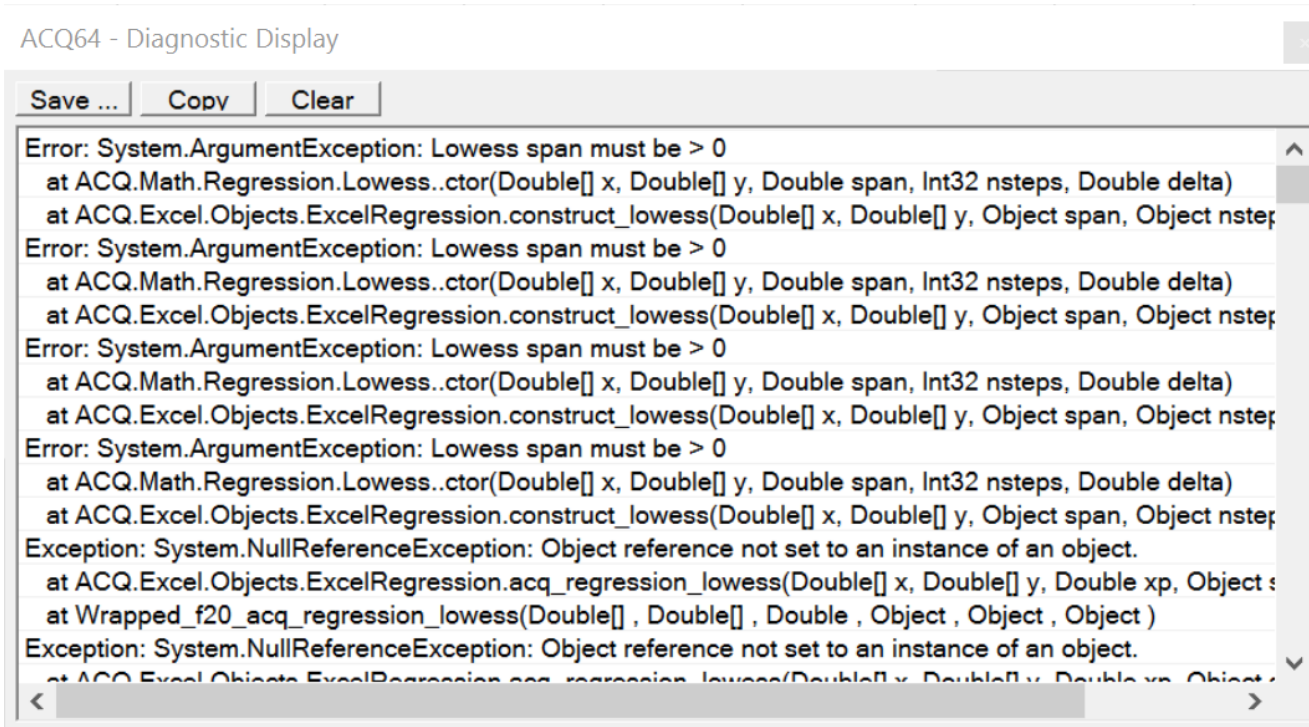
Introspection: Ctrl+Shift+A



1. ACQ Add-in supports introspection of the ACQ objects
2. ACQ object handles have the prefix #acq
3. Introspection window shows ACQ object in the currently selected Excel cell
4. The following shortcut, brings up introspection window: Ctrl+Shift+A
5. Introspection command is also accessible from the ribbon



Logging: Ctrl+Shift+H

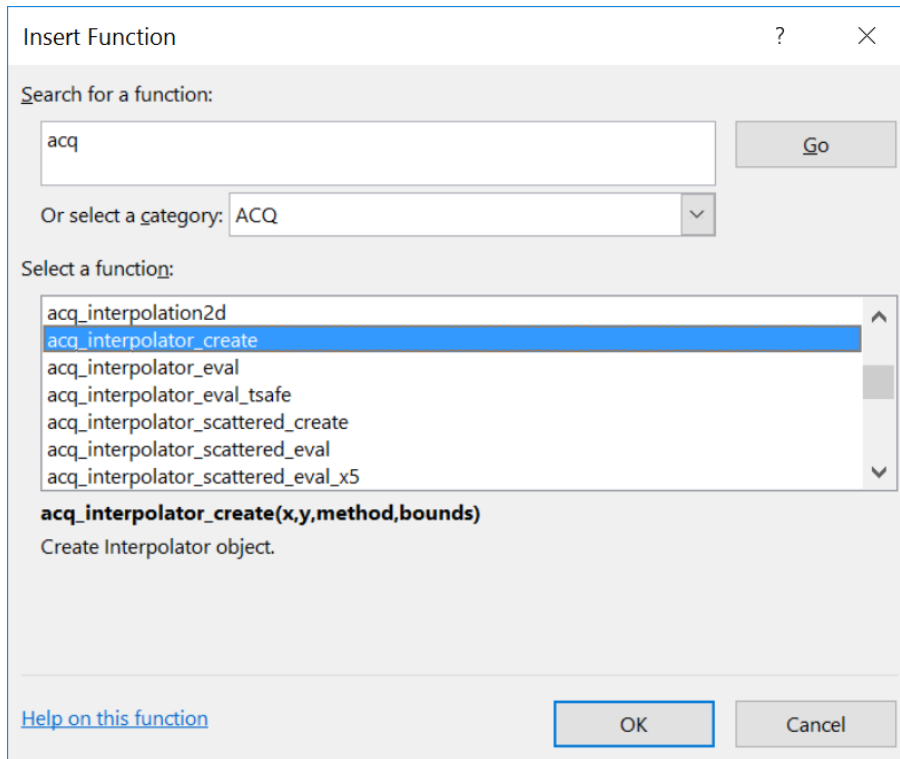


The screenshot shows a window titled "ACQ64 - Diagnostic Display". It has three buttons at the top: "Save ...", "Copy", and "Clear". Below the buttons is a text area containing the following log entries:

```
Error: System.ArgumentException: Lowess span must be > 0
  at ACQ.Math.Regression.Lowess..ctor(Double[] x, Double[] y, Double span, Int32 nsteps, Double delta)
  at ACQ.Excel.Objects.ExcelRegression.construct_lowess(Double[] x, Double[] y, Object span, Object nstep
Error: System.ArgumentException: Lowess span must be > 0
  at ACQ.Math.Regression.Lowess..ctor(Double[] x, Double[] y, Double span, Int32 nsteps, Double delta)
  at ACQ.Excel.Objects.ExcelRegression.construct_lowess(Double[] x, Double[] y, Object span, Object nstep
Error: System.ArgumentException: Lowess span must be > 0
  at ACQ.Math.Regression.Lowess..ctor(Double[] x, Double[] y, Double span, Int32 nsteps, Double delta)
  at ACQ.Excel.Objects.ExcelRegression.construct_lowess(Double[] x, Double[] y, Object span, Object nstep
Error: System.ArgumentException: Lowess span must be > 0
  at ACQ.Math.Regression.Lowess..ctor(Double[] x, Double[] y, Double span, Int32 nsteps, Double delta)
  at ACQ.Excel.Objects.ExcelRegression.construct_lowess(Double[] x, Double[] y, Object span, Object nstep
Exception: System.NullReferenceException: Object reference not set to an instance of an object.
  at ACQ.Excel.Objects.ExcelRegression.acq_regression_lowess(Double[] x, Double[] y, Double xp, Object s
  at Wrapped_f20_acq_regression_lowess(Double[] , Double[] , Double , Object , Object , Object )
Exception: System.NullReferenceException: Object reference not set to an instance of an object.
  at ACQ.Excel.Objects.ExcelRegression.acq_regression_lowess(Double[] x, Double[] y, Double xp, Object s
```

1. ACQ Add-in uses Excel-DNA logger
2. All ACQ exceptions and warnings are logged.
3. Ctrl+Shift+H (Help)

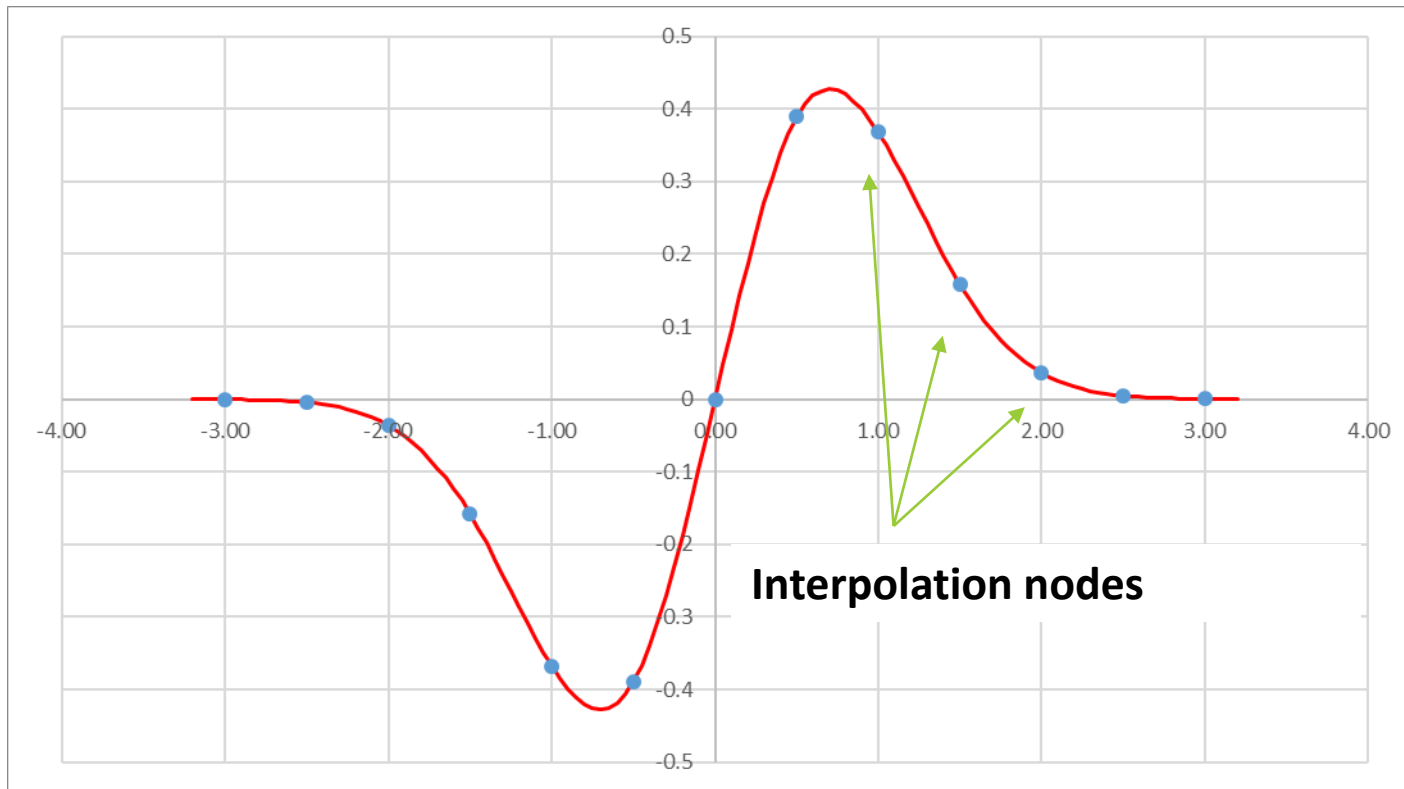
ACQ Functions



1. ACQ functions have prefix "acq_" and located in ACQ category.
2. Ctrl+Shift+H shows log window with error messages

Function	Description
acq_interpolator_create	Create Interpolator
acq_interpolator_eval	Compute interpolated value
acq_interpolation	Compute interpolated value: interpolator object is created on each call.
acq_xllpath	Returns path to the acq.xll
acq_version	ACQ version
acq_exceldna_version	Excel-DNA version

Interpolation



1. Given the location x and value of the function y at interpolation nodes, the function values between the nodes are estimated.
2. There are many interpolation methods. The choice of the method depends on underlying function.

Interpolation Procedure

=acq_interpolator_create(x,y,"Akima",TRUE)

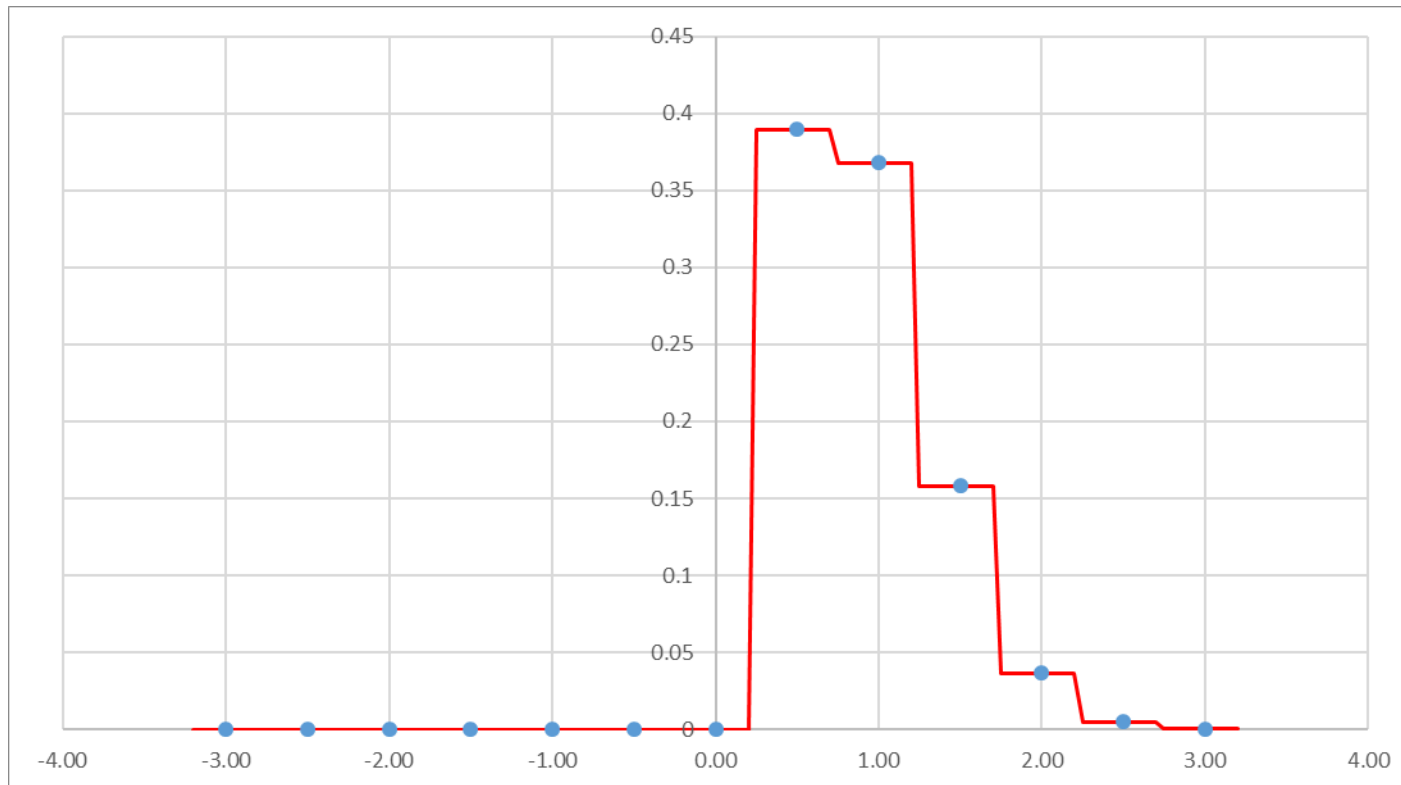
1. **x** – interpolation nodes (have to be arranged from small to large, no repeated values are allowed)
2. **y** – function values at interpolation nodes. (at least two nodes should be specified)
3. Interpolation method (string). Argument is optional, default is linear. Not case sensitive.
4. Bounds: TRUE or FALSE. Optional, default is true.

x	y
-3.0	0
-2.5	0
-2.0	0
-1.5	0
-1.0	0
-0.5	0
0.0	0
0.5	0.3894
1.0	0.367879
1.5	0.158099
2.0	0.036631
2.5	0.004826
3.0	0.00037

Method	Akima
Bounds	TRUE
Interpolator	#Interpolator:11

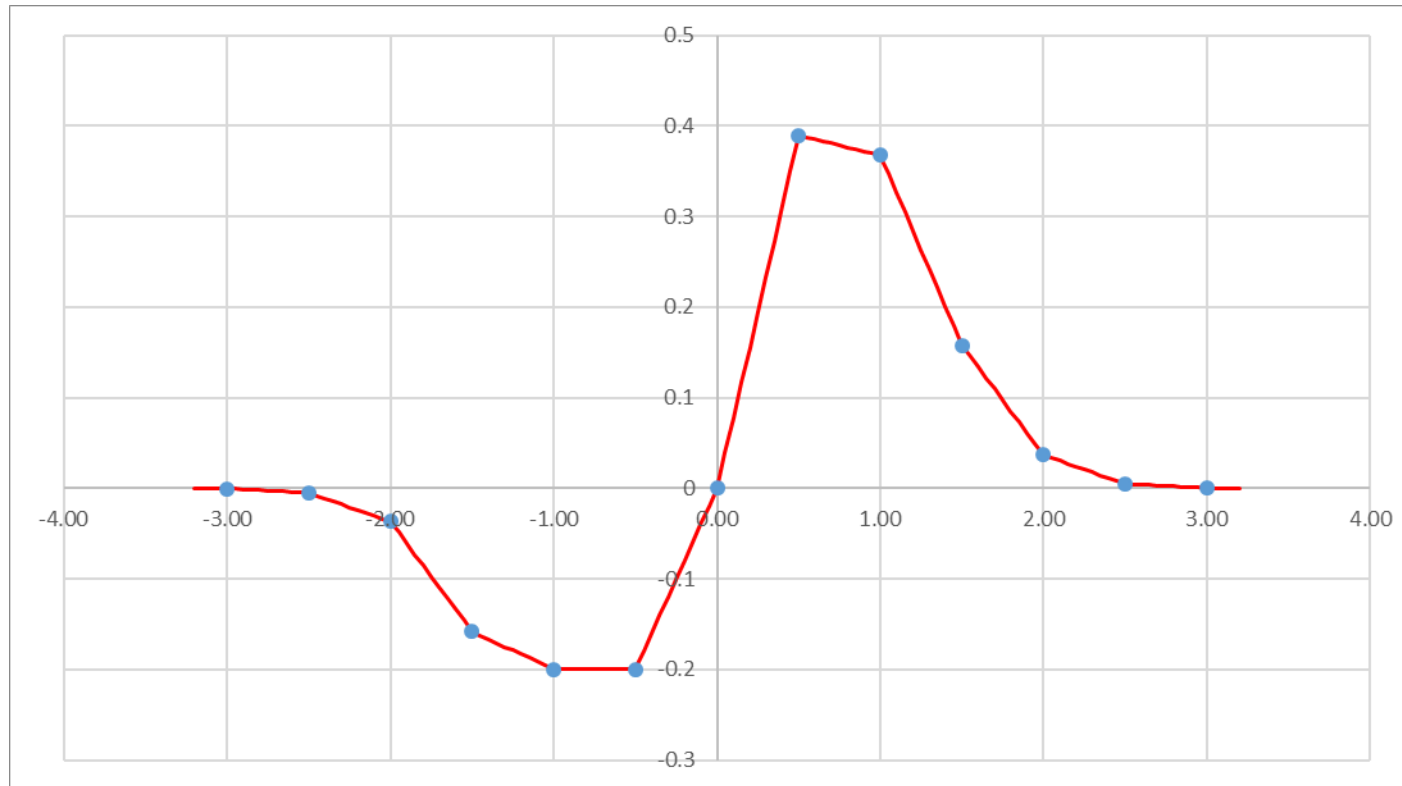
1. The function creates an interpolator object that can be used to evaluate interpolation at given point.
2. The handle will be automatically updated when any input argument change (if Automatic Calculation is enabled).

Nearest Interpolation



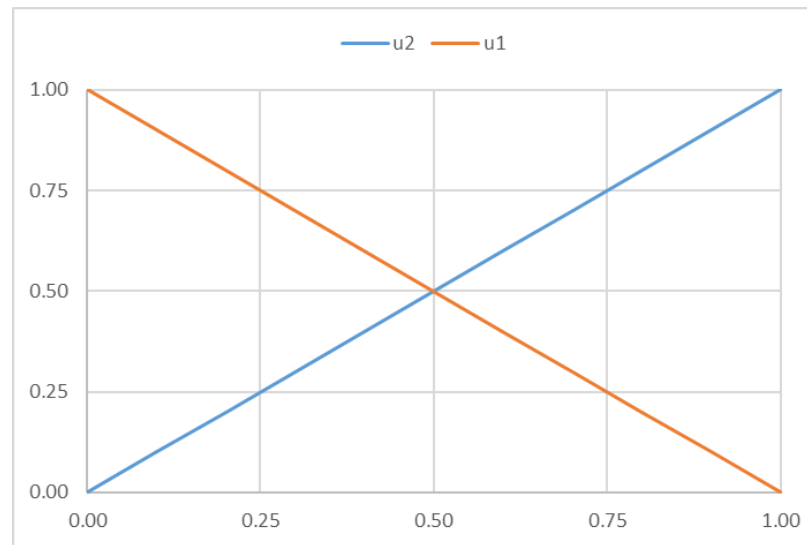
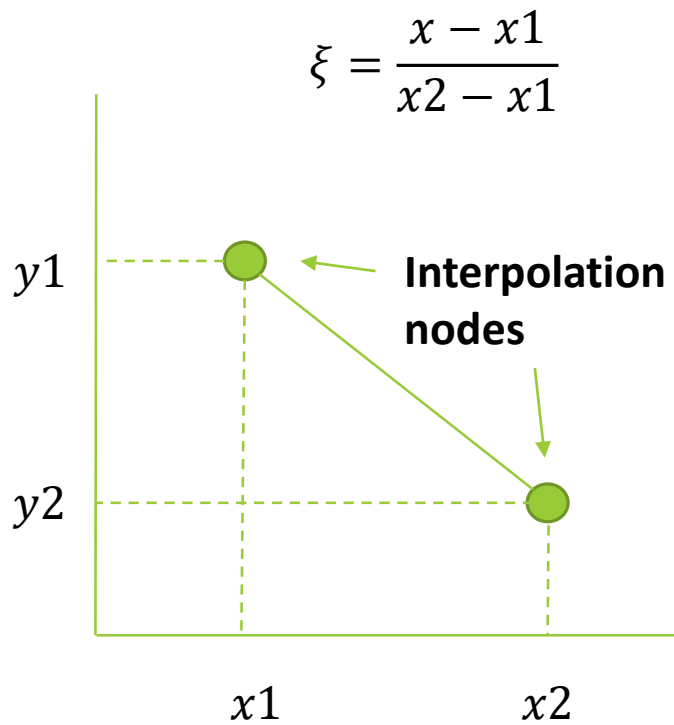
- Method: **Nearest**
- Nearest interpolation returns a value of the function at the node which is closest to the specified point.
- It is appropriate when interpolating function that can only take discrete values.

Linear Spline Interpolation



- Method: **Linear**
- Linear interpolation constructs a straight line between interpolation nodes.
- The line is a shortest distance between two points.
- The derivative of interpolation function is constant between nodes, and has discontinuity at the node points.
- Second derivative is zero on each interval.

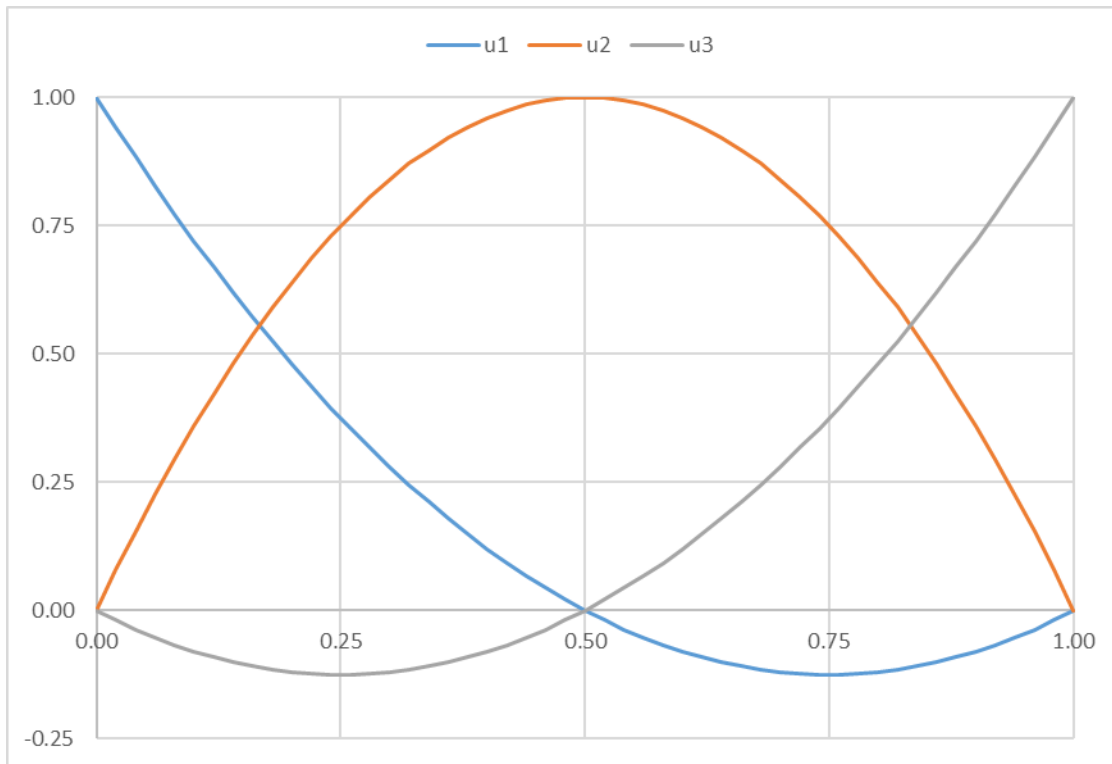
Linear Basis Functions



$$g(\xi) = y_1 \cdot u_1(\xi) + y_2 \cdot u_2(\xi)$$

- Interpolation on each interval is constructed as linear combination of basis functions
- Basis functions are defined on interval from 0 to 1, so each interpolation interval is normalized
- There are two linear basis function u_1 and u_2 (i.e. first order basis).
- Linear spline can be constructed using only function values at the nodes: y_1 and y_2 .

Quadratic Basis Functions



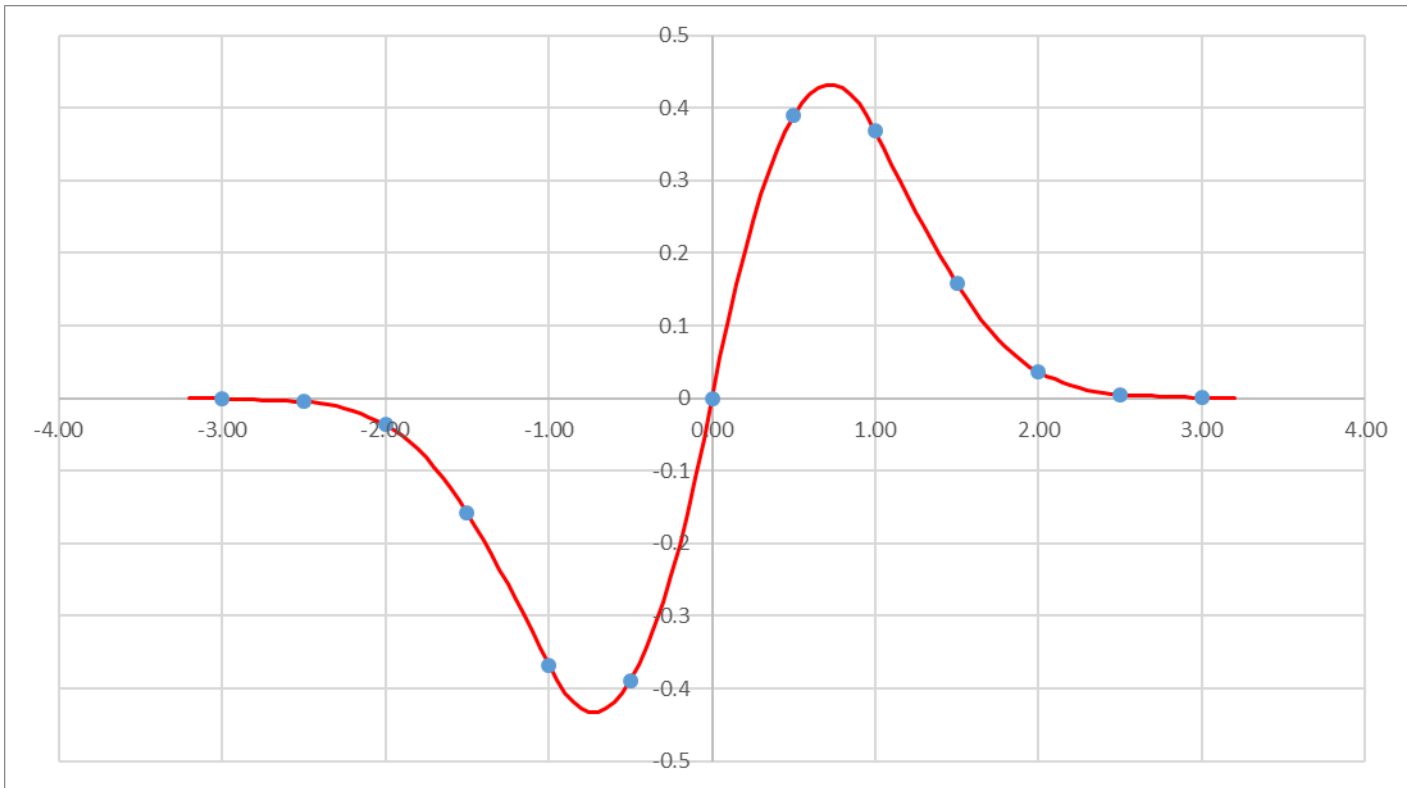
$$u_1(\xi) = 1 - 3\xi + 2\xi^2 = (1 - \xi)(1 - 2\xi)$$

$$u_2(\xi) = 4\xi - 4\xi^2 = 4\xi(1 - \xi)$$

$$u_3(\xi) = 2\xi^2 - \xi = \xi(2\xi - 1)$$

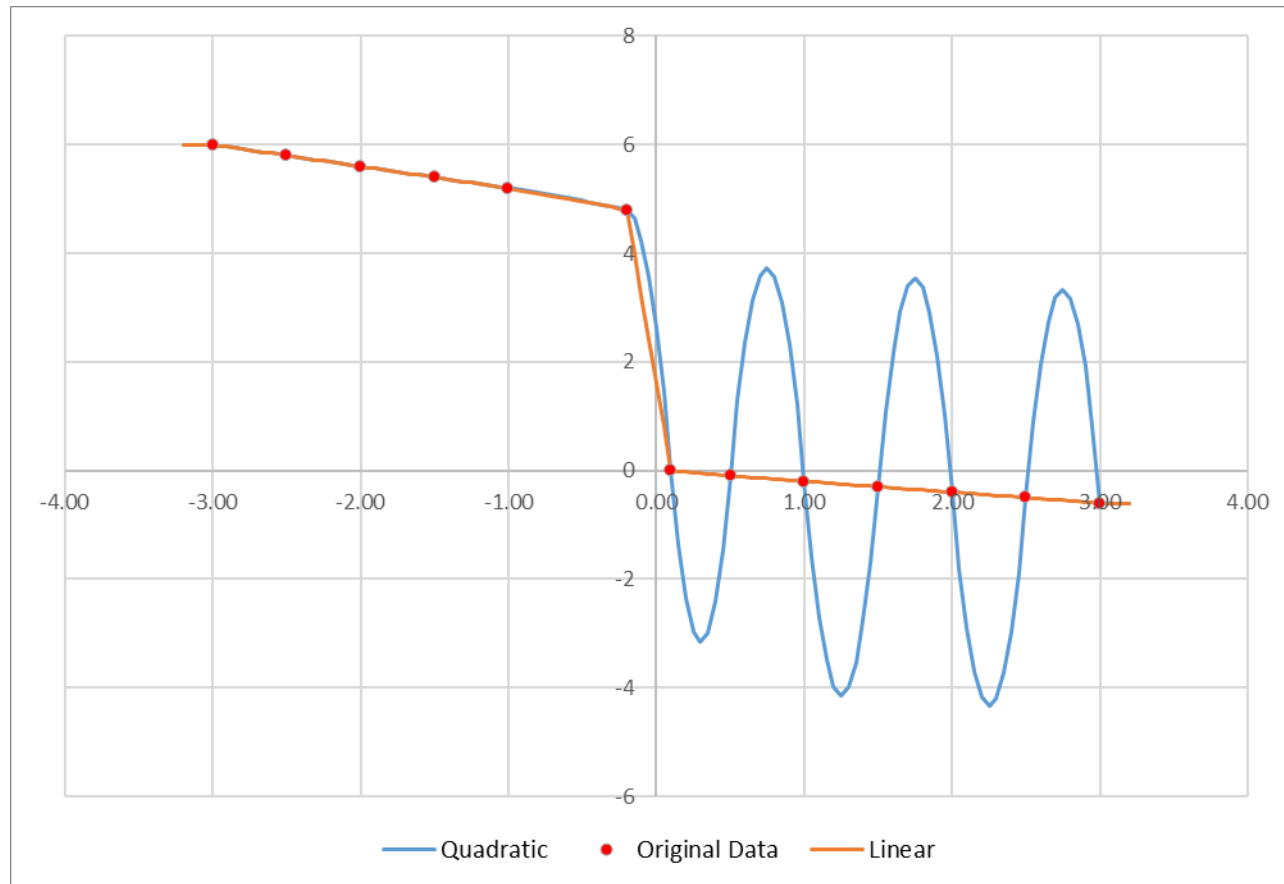
- U_1 – controls the value of the function at $\xi = 0$
- U_3 – controls the value of the function at $\xi = 1$
- U_2 – controls the value of the function at the midpoint (which we don't know).
- Coefficient for U_2 is selected by matching derivative at the end of the interval (i.e. derivative at the end of the first interval is assumed to be the same as derivative at the start of the next interval). The derivative at the start of the first interval is computed by fitting parabola to the first 3 points.

Quadratic Spline Interpolation



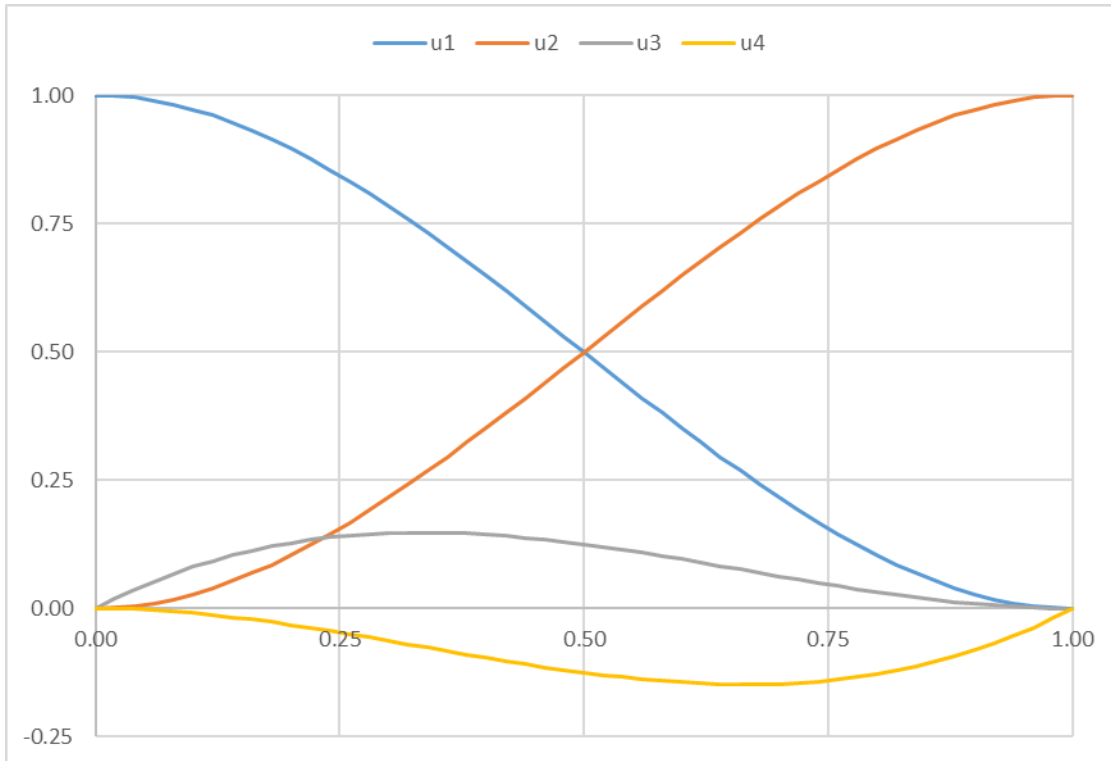
- Method: **Quadratic**
- Quadratic spline interpolation uses second order basis functions on each interval.
- First derivative is piecewise linear (continuous at the nodes)
- Second derivative is constant on each interval.
- The same as linear interpolation when two nodes are specified, fits parabola in case of 3 nodes.
- Suitable for very smooth function

Quadratic Spline Interpolation



- Stability issues become pronounced for functions with fast changing derivatives (i.e. large second derivatives).

Cubic Basis Functions (Hermite)



- Many different interpolation methods use cubic basis (aka Hermite basis). It is popular because it decouples function values from derivatives at the nodes.
- Akima, Steffen, Hermit, Cubic spline, and many other interpolation methods use Hermite basis functions.
- Coefficients for U1 and U2 are given by function values at the node.
- U3 and U4 control the derivatives at the nodes without affecting values of the function.
- There are a lot of different algorithms for choosing coefficients for U3 and U4.

Cubic Basis Functions (Hermite)

$$u_1(\xi) = (1 + 2\xi)(1 - \xi)^2$$

$$u_2(\xi) = \xi^2(3 - 2\xi)$$

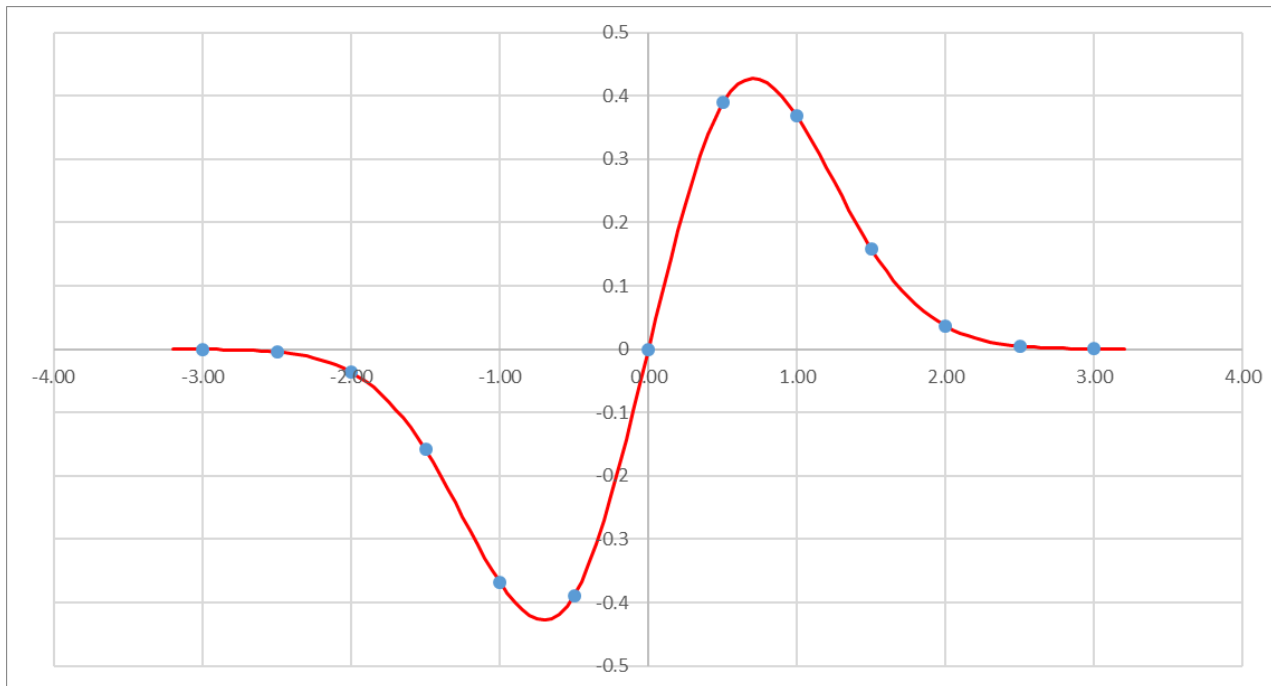
$$u_3(\xi) = \xi(1 - \xi)^2$$

$$u_4(\xi) = -\xi^2(1 - \xi)$$

$$f(\xi) = \sum_{i=1}^4 c_i \cdot u_i(\xi)$$

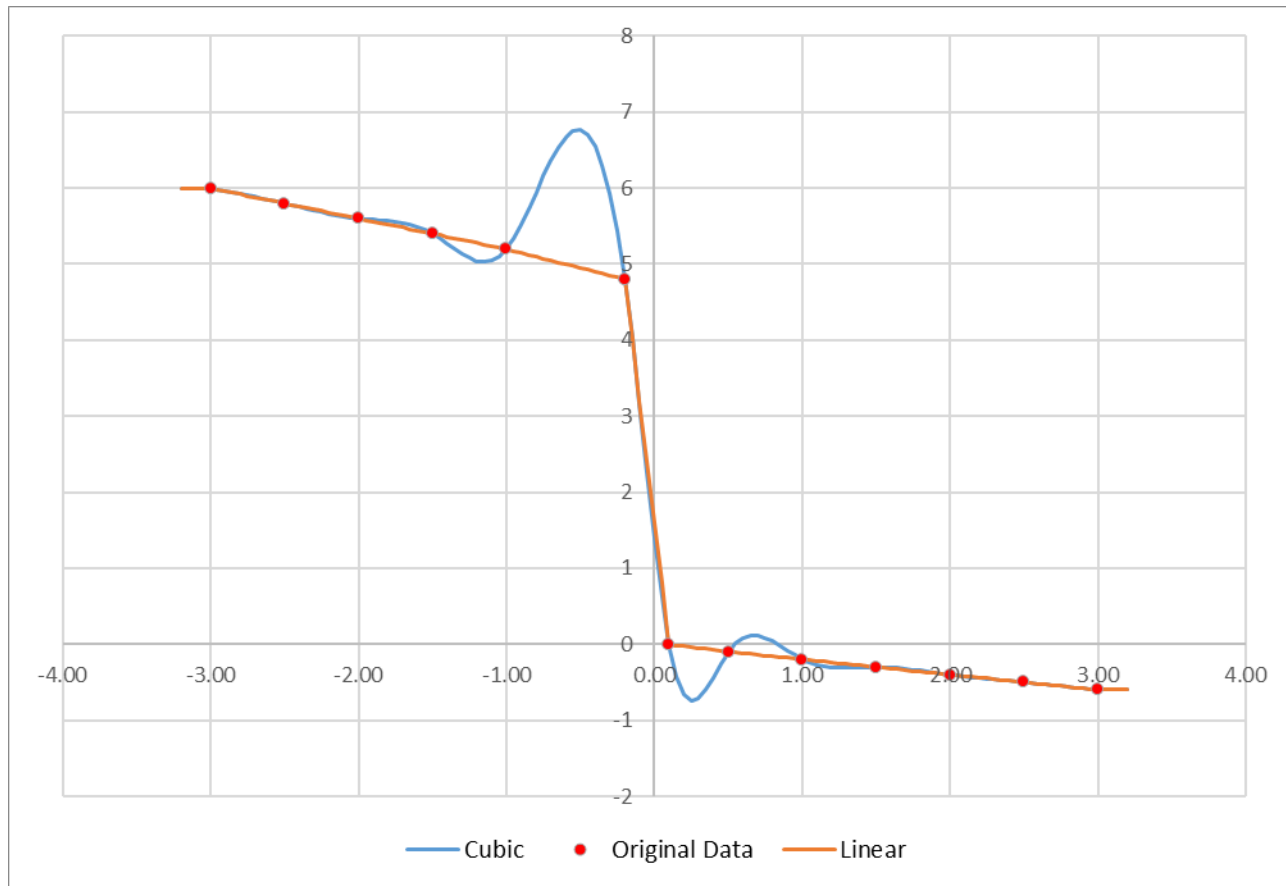
- U_3 and U_4 have zero values at the nodes, and determine derivatives at the node.
- U_1 and U_2 have zero derivatives at the nodes and determine function values
- Coefficients for U_1 and U_2 are set to the values at the nodes
- Therefore many algorithms for setting derivatives at the nodes using U_3 and U_4 .
- This convenient representation essentially decouples function values from derivatives at the nodes.

Cubic Spline Interpolation



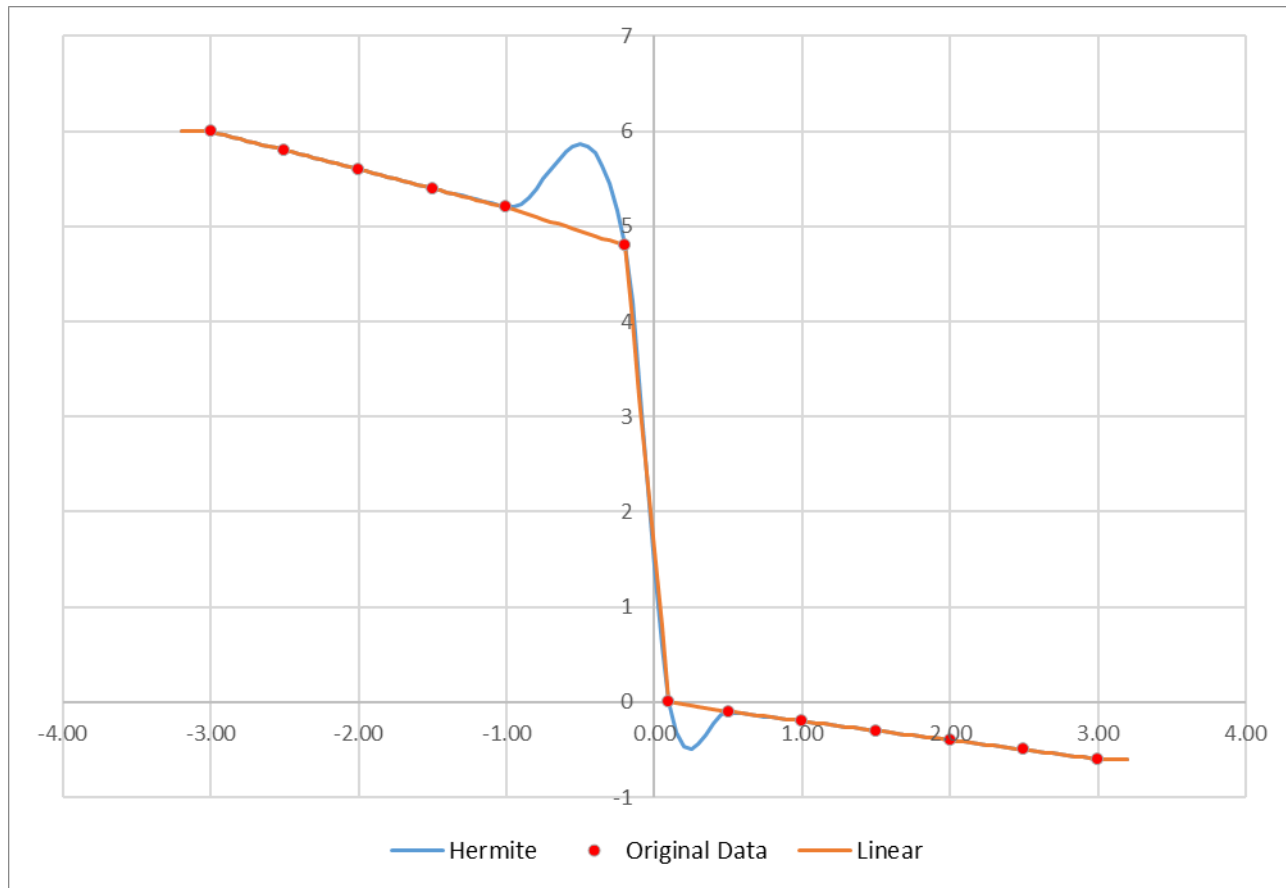
- Method: **Cubic**
- Cubic spline or Natural Cubic spline each interval is constructed such that first and second derivatives are continuous at the nodes.
- This is done by considering all interpolation points (i.e. solving tridiagonal system of equations).
- Therefore interpolation is not local, changes to function values at one node affect all interpolated points.
- Produces very smooth but sometimes wiggly interpolation curve.

Cubic Spline Interpolation



- Method: **Cubic**
- Functions with sharp bends tend to produce wiggles

Hermite Spline Interpolation



- Method: **Hermite**
- Cubic spline interpolation with derivatives at the nodes determined by finite differences
- Second derivative is not continuous at the nodes.
- Interpolation is local (compared with natural cubic spline at $x < 0$).

Hermite and Hermite QS

In **Hermite** method derivatives at the nodes are set to average finite difference derivative. Both methods produce the same results for equally spaced points. But Hermite is a bit more stable for non-equally spaced nodes.

$$s_i = \frac{1}{2} \left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}} + \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \right)$$

In **HermiteQS** method derivatives at the nodes are computed by fitting parabola and then differentiating it at the central node.
QS – quadratic slopes

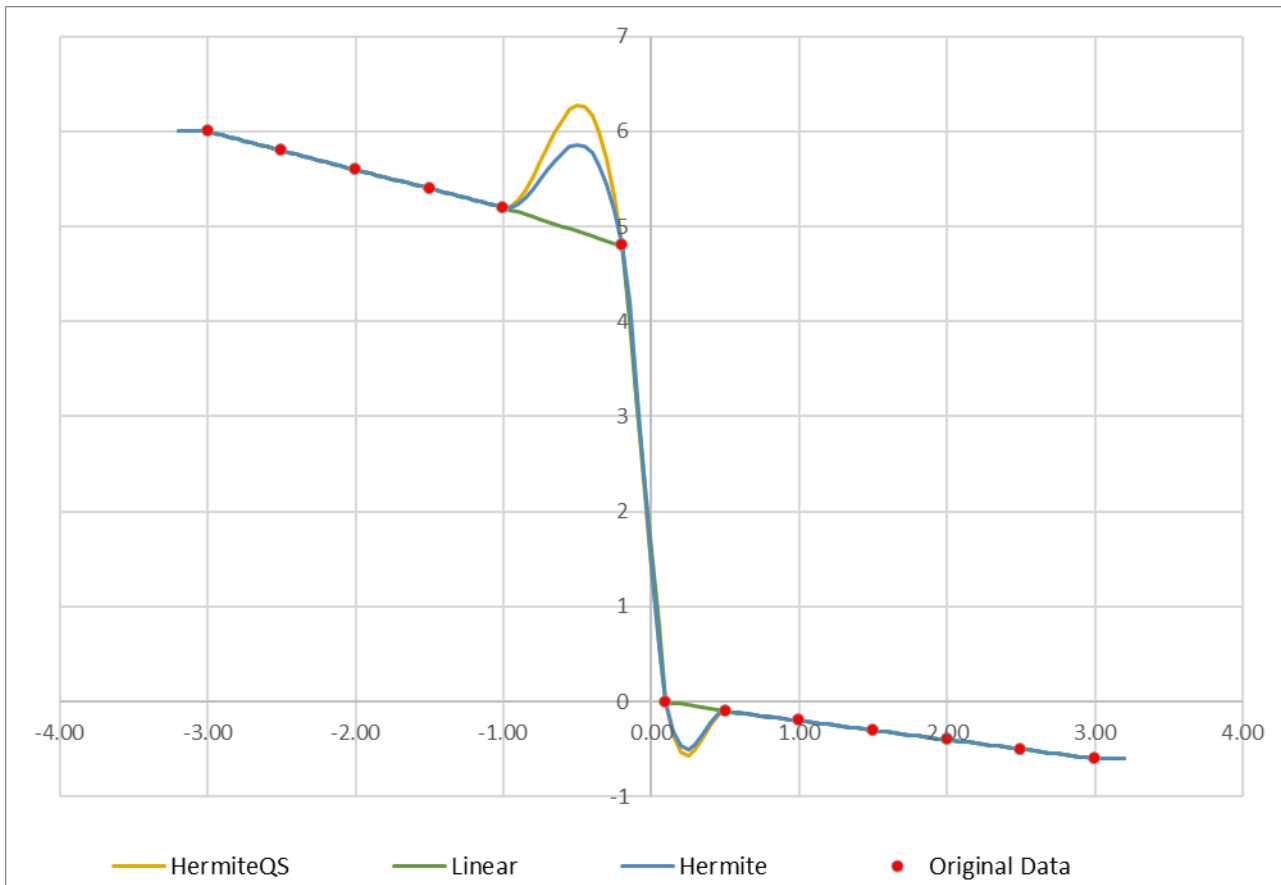
$$s_i = \left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}} \frac{x_{i+1} - x_i}{x_{i+1} - x_{i-1}} + \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \frac{x_i - x_{i+1}}{x_{i+1} - x_{i-1}} \right)$$

$$s_0 = \frac{y_1 - y_0}{x_1 - x_0}$$

$$s_n = \frac{y_n - y_{n-1}}{x_n - x_{n-1}}$$

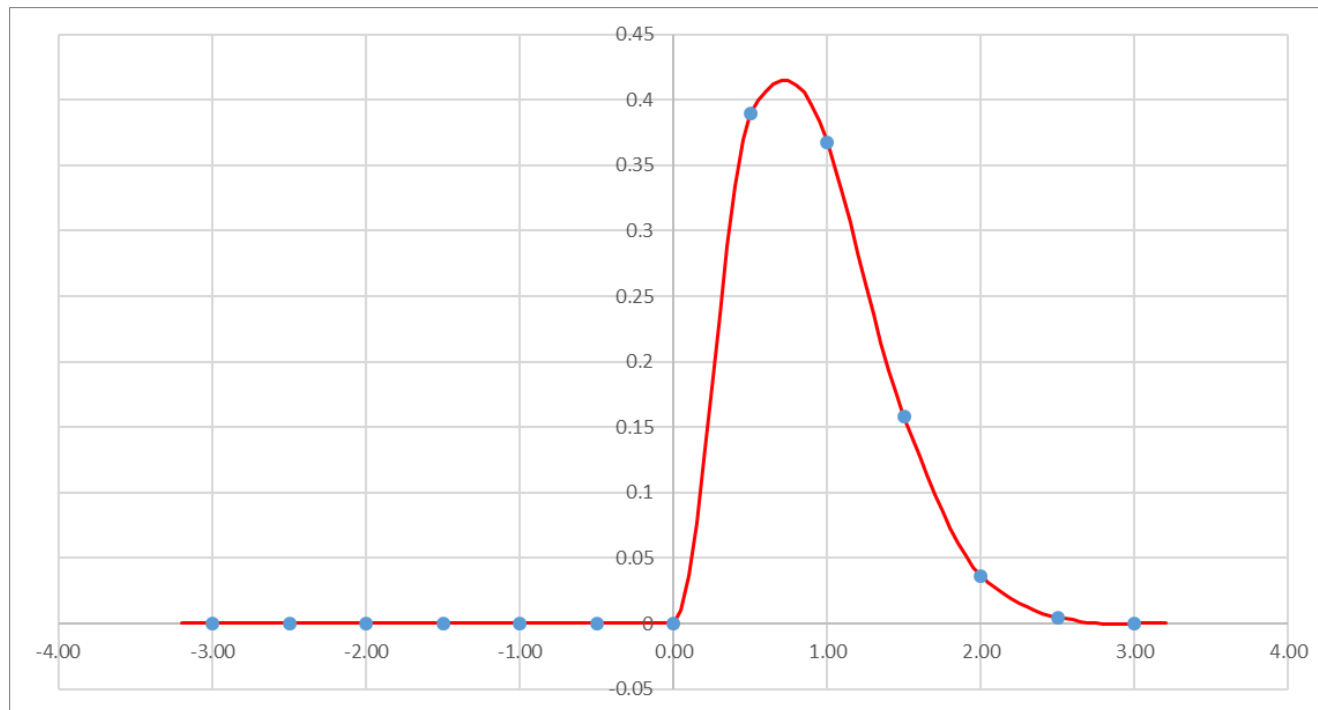
derivatives at the end points are the same for both methods

Hermite and Hermite QS



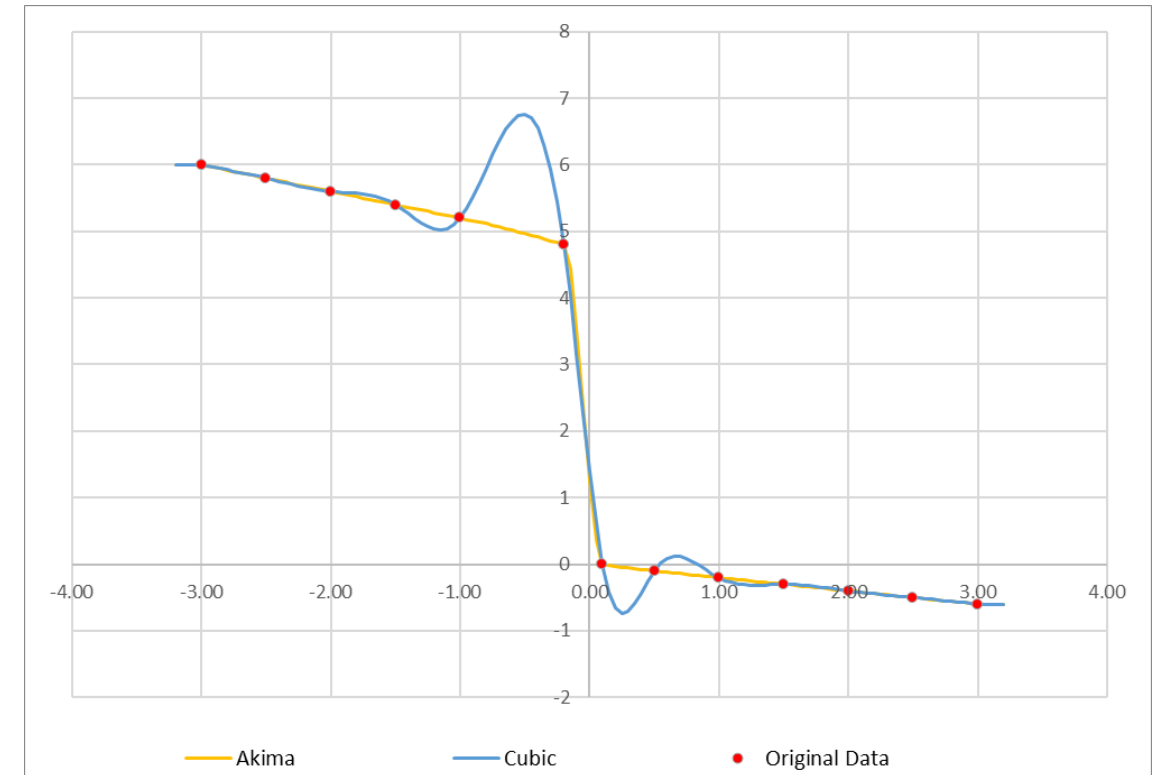
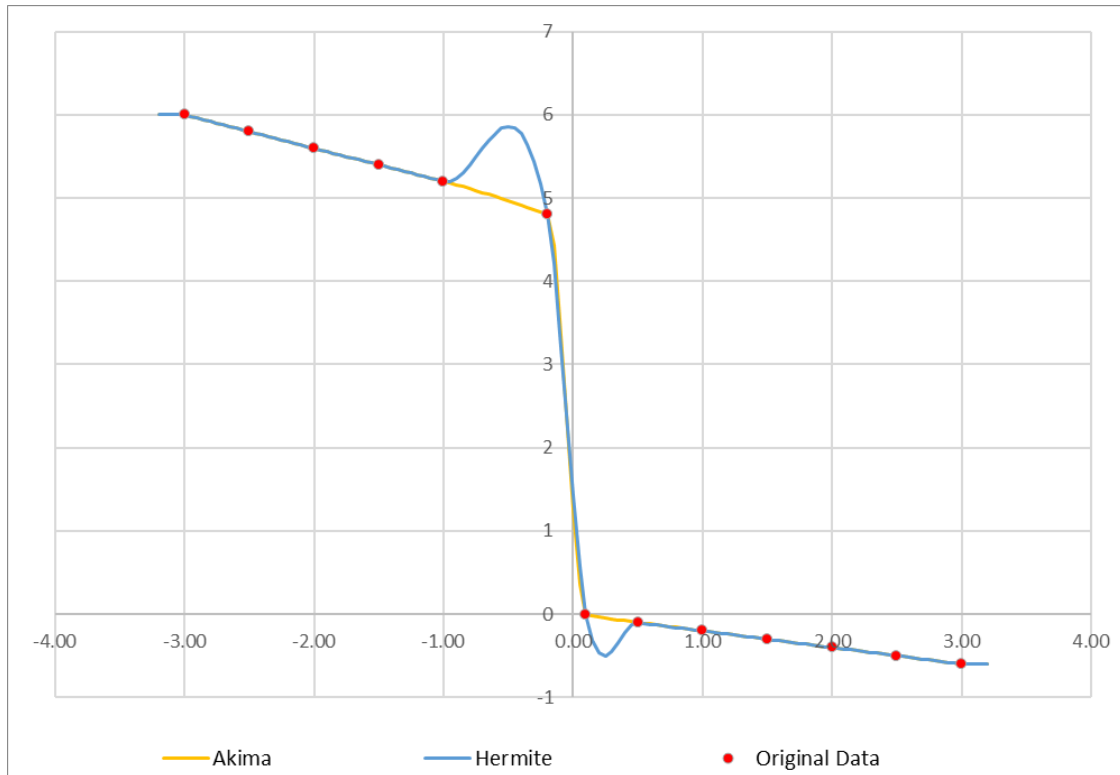
Both methods produce the same results for equally spaced points. But Hermite is a bit more stable for non-equally spaced nodes.

Akima Spline Interpolation

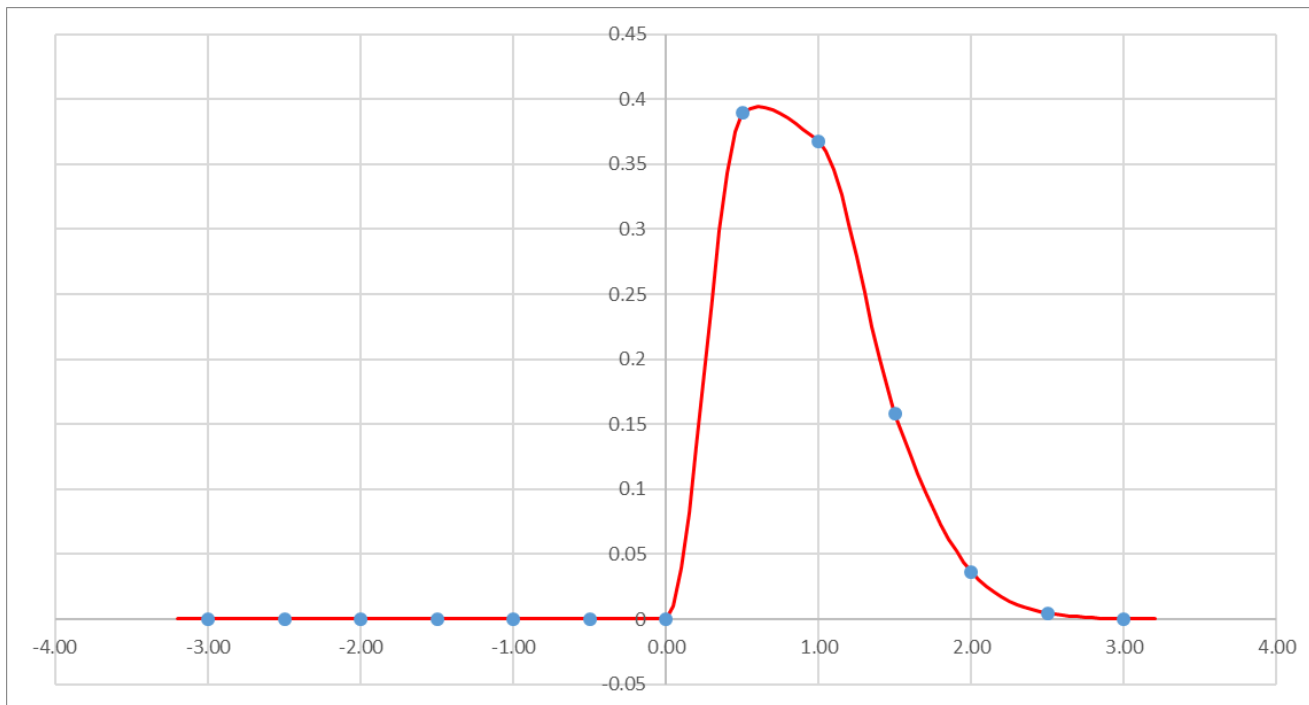


- Method: **Akima**
- Interpolation based on cubic spline but uses extra heuristic conditions to limit derivatives.
- It is popular method, but it can be unstable under certain conditions (i.e. small changes in node values lead to very large changes in interpolated values).
- Notice that wiggle at $x=-0.5$ is completely eliminated (compared to Hermite and Cubic).

Akima Spline vs Hermite and Cubic



Steffen Spline Interpolation



- Method: **Steffen**
- Interpolation based on cubic spline but limit the derivatives to enforce monotonicity.
- It is stable but tend to produce interpolation function with large second derivatives.
- Steffen interpolation is similar to Akima, but more stable and elegant.

2D Rectangular Grid Interpolation

`=acq_interpolator2d_create(x1,x2,y,"Bilinear")`

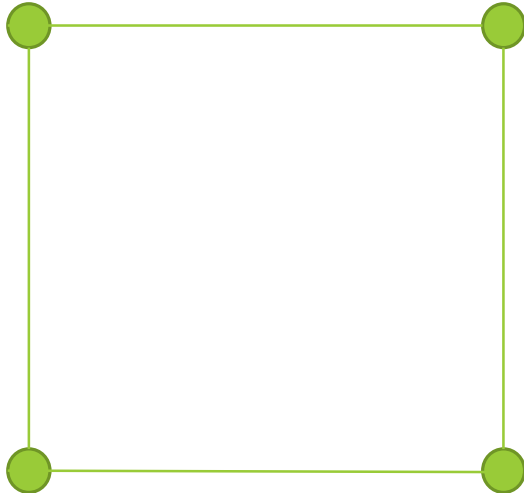
`=acq_interpolator2d_eval (handle, x1p, x2p)`

		x1		
x2	y	0	0.5	1
	0	1	0.8	0.5
	1	0	0.5	1

- **acq_interpolator2d_create** – creates 2D interpolator object
- **acq_interpolator2d_eval** – evaluates the interpolator value at specified point
- **acq_interpolation2d** - evaluates interpolation at specified point(in-situ without constructing interpolator object)

- Rectangular grid is specified by the nodes locations x1, and x2. Nodes have to be ordered and unique
- The values of the function are specified as rectangular range y.
- Number of rows y should match the size of x2, and number of columns should match the size of x1
- Currently implemented methods are: **Bilinear, Bicubic, BiHermite, BiAkima, BiSteffen**.

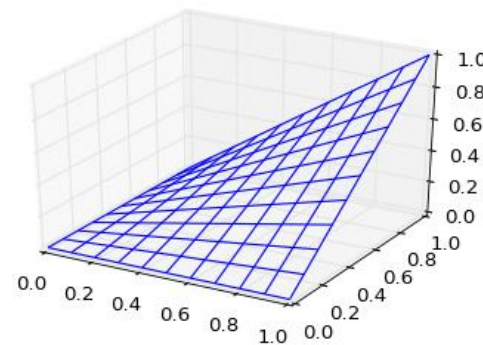
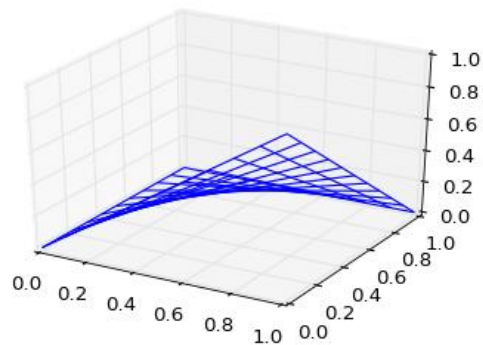
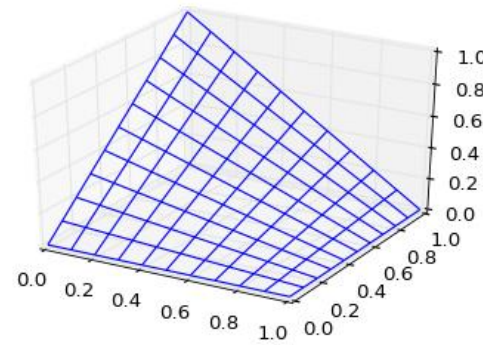
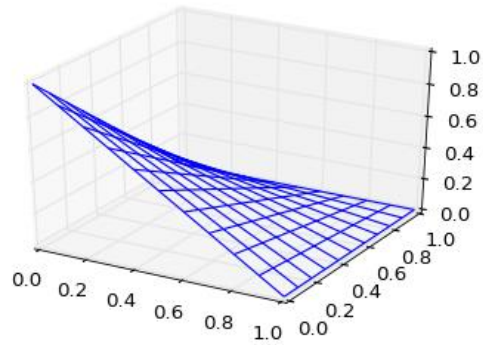
Bilinear Interpolation



$$\begin{aligned}u_1(\xi, \eta) &= \xi \cdot \eta \\u_2(\xi, \eta) &= (1 - \xi) \cdot \eta \\u_3(\xi, \eta) &= \xi \cdot (1 - \eta) \\u_4(\xi, \eta) &= (1 - \xi) \cdot (1 - \eta)\end{aligned}$$

- Method: **Bilinear**
- Basis functions are simply a product of linear basis functions for each dimension. So dimensions are treated independently and the method is fully equivalent to applying linear interpolation in each dimension sequentially.
- Note that interpolation is not linear but linear in each dimension.
- Basis function coefficients are equal to function values at the nodes

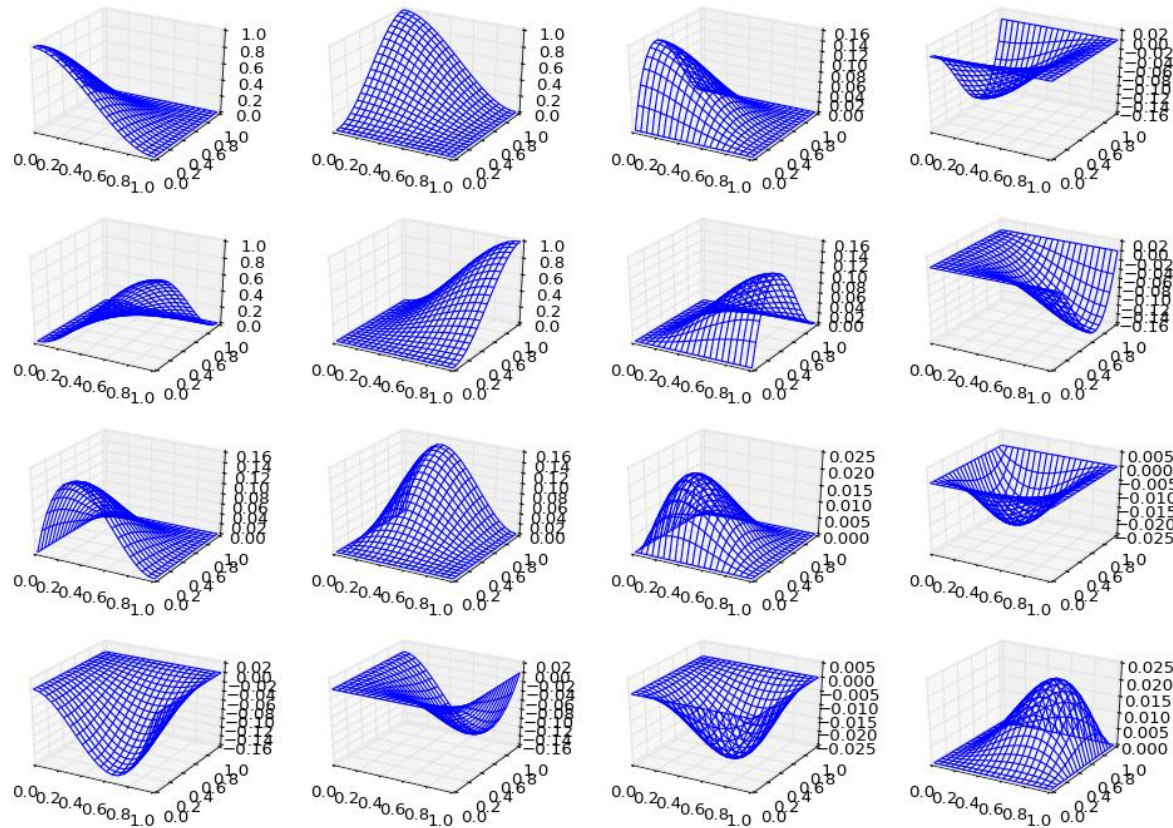
Bilinear Basis Functions



- Basis function for bilinear interpolations
- Basis function coefficients are equal to function values at the nodes
- Note that functions are not linear (i.e. surface is not a plane)
- Interpolation function is linear combination of basis functions with coefficients equal to the values at the nodes.

$$f(\xi, \eta) = \sum_{i=1}^4 c_i \cdot u_i(\xi, \eta)$$

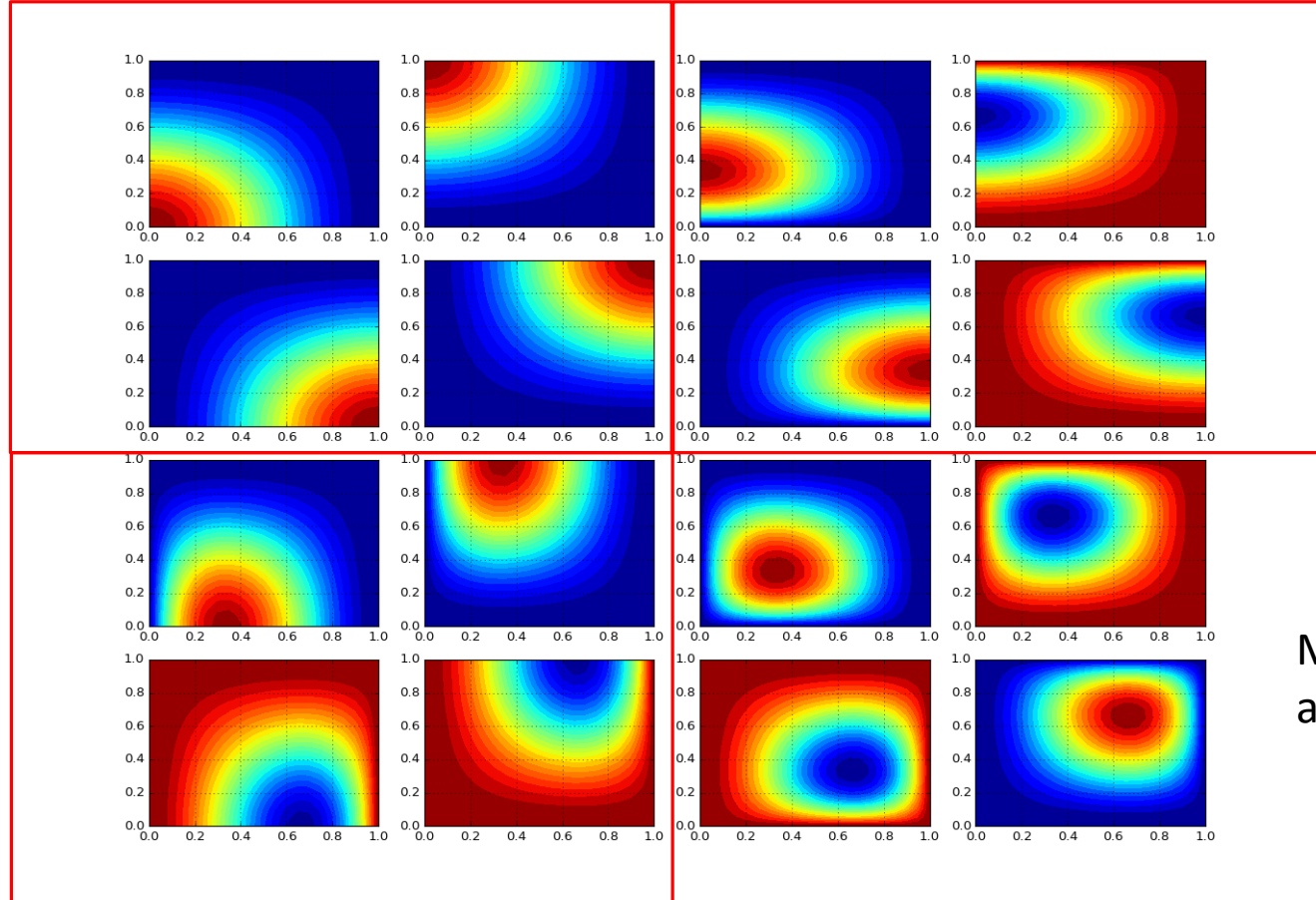
Bicubic Interpolation



- Method: **bicubic**
- Basis functions are simply a product of cubic (Hermite) basis function for each dimension.
- There are four cubic basis functions per dimension, therefore in 2D, there are 16 cubic basis functions (shown on the left).
- ACQ uses finite difference derivatives to find coefficients for cubic basis functions. Therefore algorithm is equivalent to Hermit spline (i.e. not natural cubic spline).

Bicubic Basis Functions

Function values at the nodes

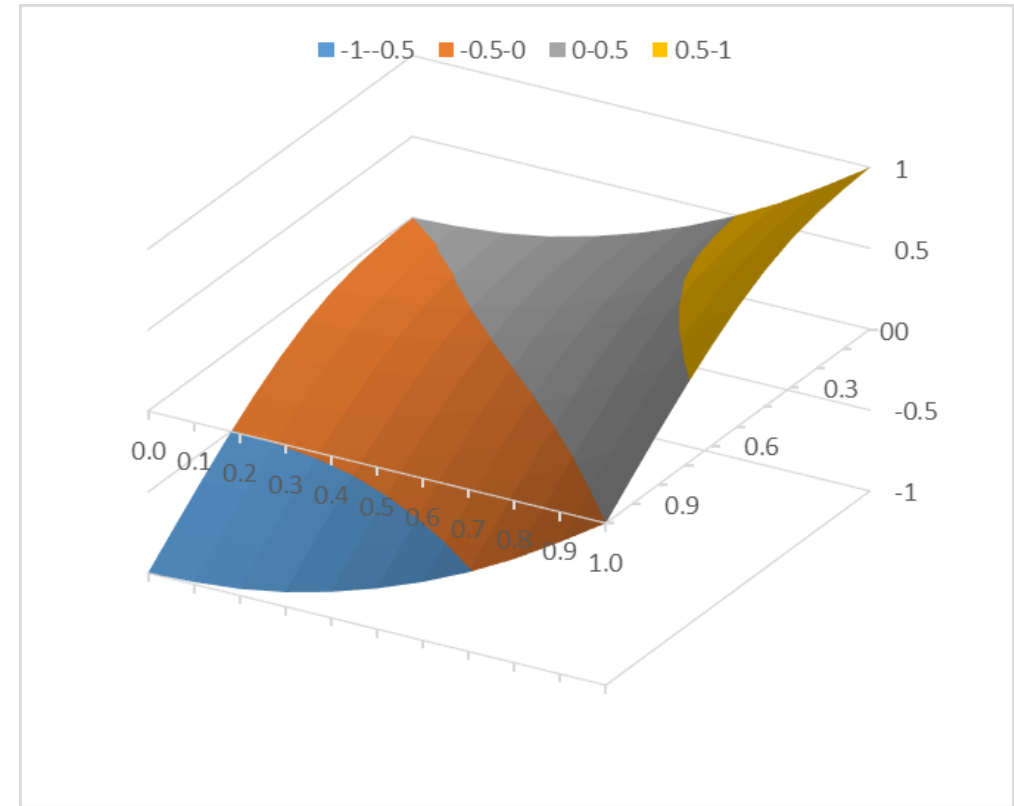
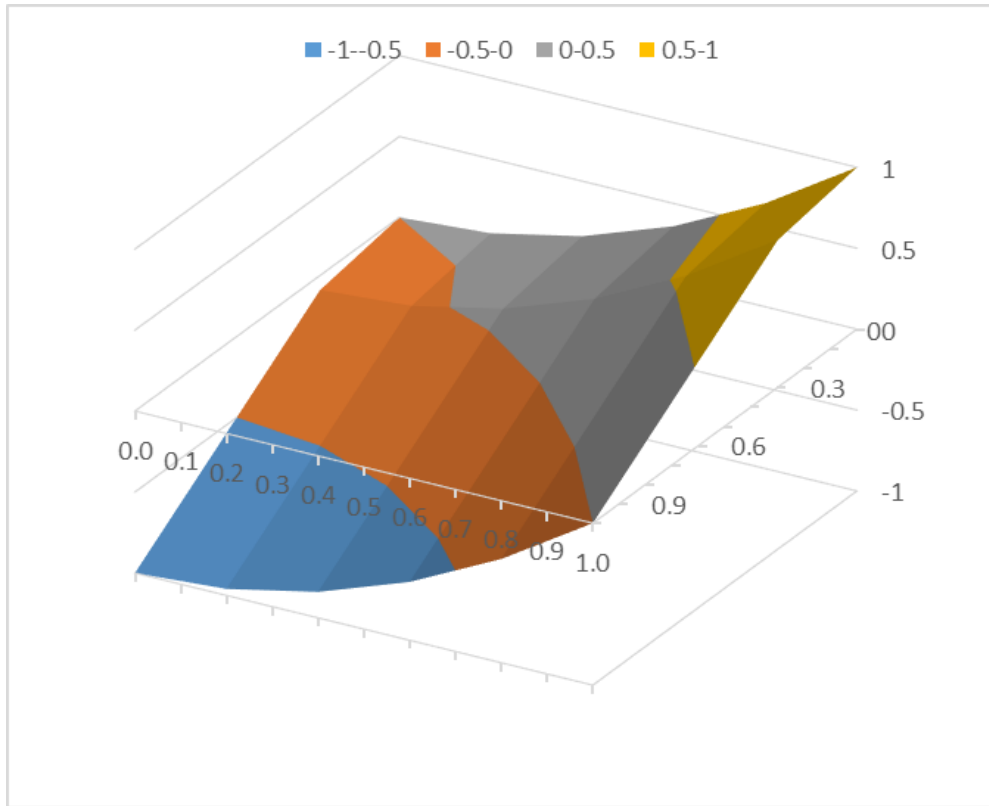


Function derivatives at the nodes (along vertical edges)

Function derivatives at the nodes (along horizontal edges)

Mixed derivatives at the nodes.

Bilinear vs Bicubic



Scattered Data Interpolation

x1	x2	y
-1.85088	-0.8531	0.015709
-0.01402	-2.92255	0.000195
0.732653	0.005971	0.584607
1.907031	-0.079	0.026174
-0.37363	1.100778	0.2589
0.672671	-1.01391	0.227525

x: range of node coordinates one column per dimension

y: column of node values (the same number of rows as x range)

scales: optional argument, scale applied to each dimension.

method: name of radial basis function e.g. linear, multiquadrics

- Interpolation nodes don't have to be on the regular grid
- There is no restriction on number of dimensions
- Interpolation is based on radial basis functions.
- Currently implemented radial basis functions are: **Linear, Cubic, Multiquadrics, Gaussian, Thinplate, InverseQuadratic, InverseMultiquadric**

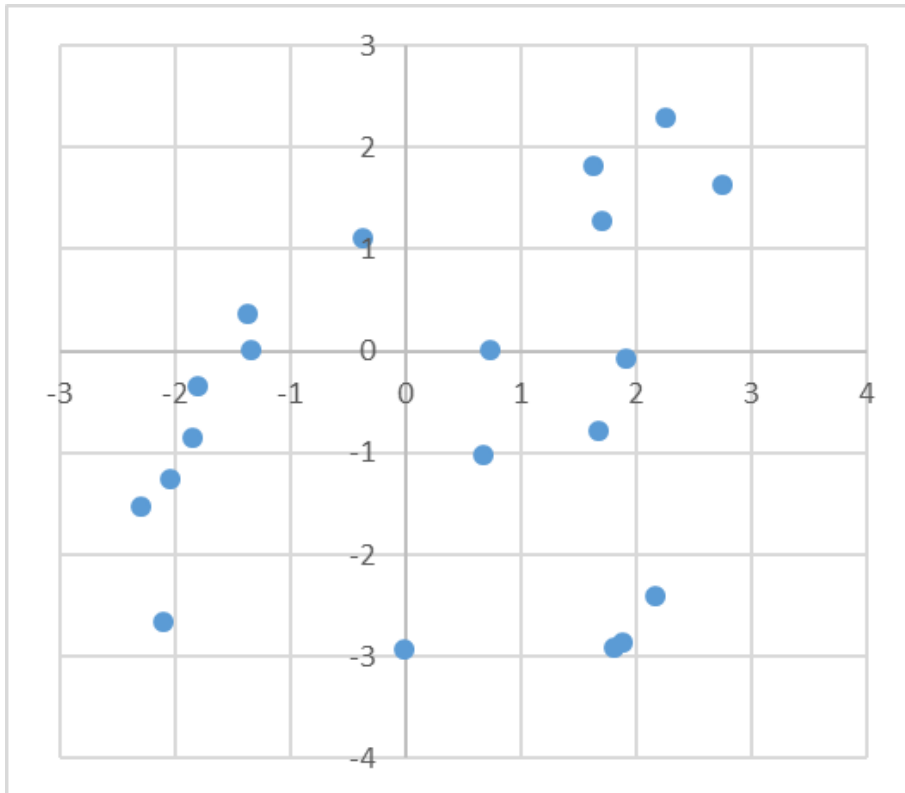
`=acq_interpolator_scattered_create(x, y, scales, method)`

`=acq_interpolator_scattered_eval(x)`

`=acq_interpolator_scattered_eval_x5(x1, x2, x3, x4, x5)`

Scattered Data Interpolation

2D scattered data



2D interpolation on scattered data

