

## Data Smoothing Functions in Mathcad

A data smoother takes a set of data and returns a new set that contains less noise than the original set but retains the basic shape and important properties of the original data. Mathcad provides three data smoothing functions:

**medsmooth**, **ksmooth** and **supsmooth**.

Here's an example provided by Stuart Bruff of what the smoothers do. The matrix contains five sets of stress-strain data from one-dimensional compression tests on different specimens.

M :=

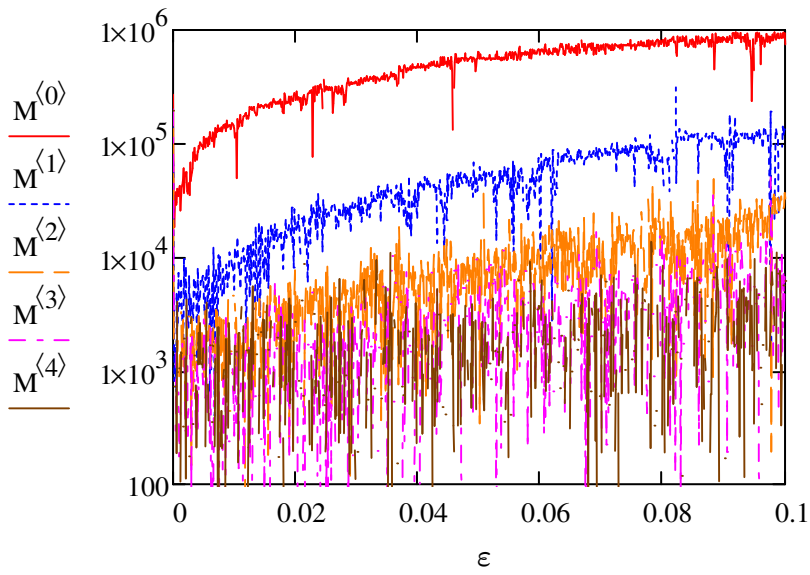
	0	1	2	3	4
0	2.698·10 <sup>5</sup>	1.396·10 <sup>5</sup>	1.352·10 <sup>5</sup>	1.999·10 <sup>5</sup>	1.943·10 <sup>5</sup>
1	2.483·10 <sup>4</sup>	8.23·10 <sup>3</sup>	1.905·10 <sup>3</sup>	2.897·10 <sup>3</sup>	-152.84
2	1.83·10 <sup>4</sup>	808.982	-578.111	-9.403·10 <sup>3</sup>	724.82
3	3.015·10 <sup>4</sup>	1.601·10 <sup>3</sup>	-2.749·10 <sup>3</sup>	-3.364·10 <sup>3</sup>	-5.141·10 <sup>3</sup>
4	3.721·10 <sup>4</sup>	1.004·10 <sup>3</sup>	-273.6	275.638	-2.75·10 <sup>3</sup>
5	2.884·10 <sup>4</sup>	643.342	1.808·10 <sup>3</sup>	1.438·10 <sup>3</sup>	688.809
6	3.047·10 <sup>4</sup>	4.857·10 <sup>3</sup>	-2.806	-143.205	...

$\varepsilon_{ww}$  :=

	0
0	0
1	1·10 <sup>-4</sup>
2	2·10 <sup>-4</sup>
3	3·10 <sup>-4</sup>
4	4·10 <sup>-4</sup>
5	5·10 <sup>-4</sup>
6	...

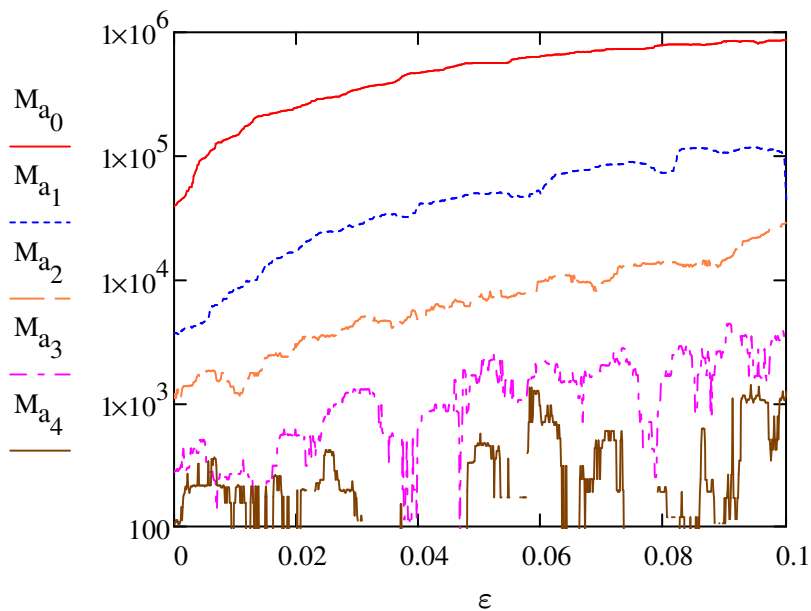
i := 0..cols(M) - 1

Each column of the matrix is plotted below providing very noisy data, but the trend is apparent.

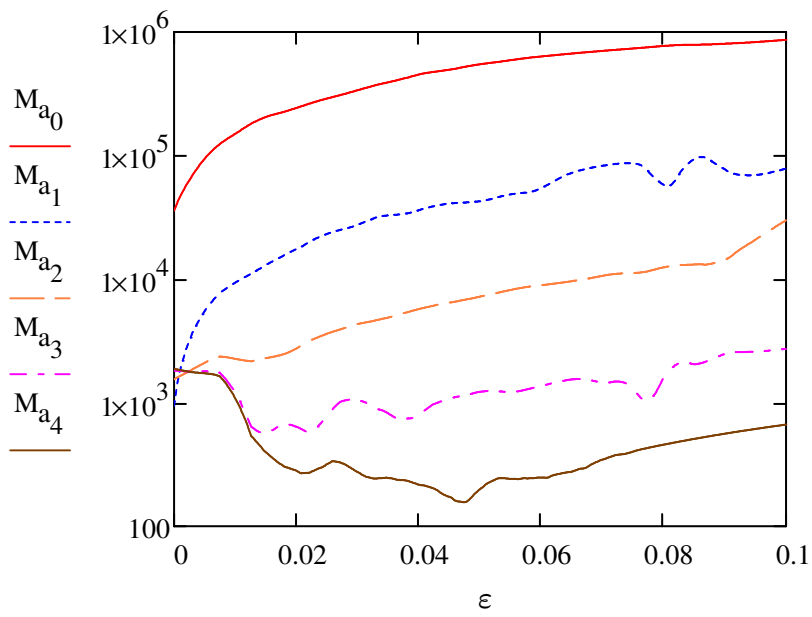


Now see what each of Mathcad's smoothing functions do to smooth out the curves in the plots: medsmooth, ksmooth, and supsmooth.

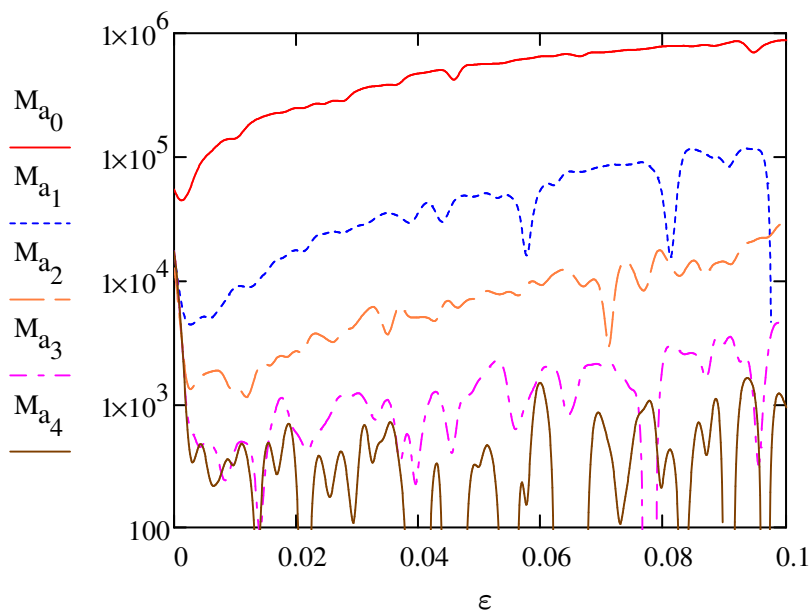
$$M_{a_i} := \text{medsmooth}(M^{(i)}, 55)$$



$$M_{a_i} := \text{supsmooth}(\varepsilon, M^{(i)})$$



$$M_{a_i} := \text{ksmooth}(\varepsilon, M^{(i)}, \varepsilon_{23} - \varepsilon_0)$$



## How the Smoothers Work

To see how each of these smoothers works, use the following set of data.

$$\text{ORIGIN} := 1$$

$$i := 1..300$$

$$t_i := \frac{i}{150}$$

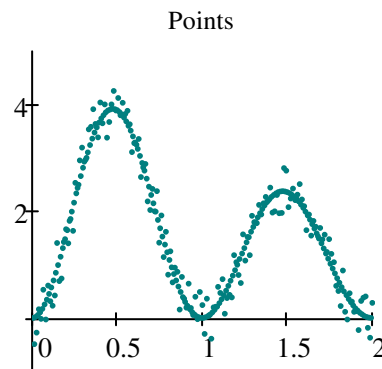
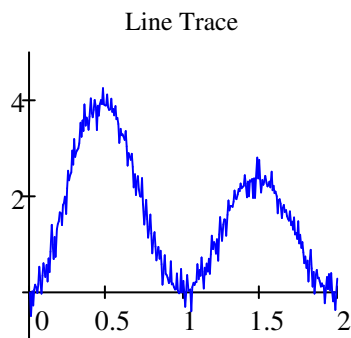
$$j := 1..150$$

$$f(t) := 5 \cdot e^{-\frac{t}{2}} \cdot \sin(\pi \cdot t)^2$$

$$Q_i := f(t_i)$$

The **rnd** function adds random noise to every other point in the data set. You get a different set of data each time this worksheet is calculated.

$$Q_{2:j} := Q_{2:j} + \text{rnd}(1) - \frac{1}{2}$$



## Median Smoothing: The medsmooth function

Medsmooth is moving window smoothing, using a symmetric window. But rather than using a mean or a polynomial fit it uses a median as the smoothed value. Median smoothing is particularly useful in cases where there are sudden bursts of noise or incidents of corruption in the data.

The **medsmooth** function takes two arguments:

medsmooth(y, n)

y - a real vector of data

n - the window size (number of points considered when Mathcad modifies each point)

**Medsmooth** returns a modified vector of the same size as the original data.

For almost all points in a data set, **Medsmooth** takes a window of data around a given data point and replaces that point with the median of the values in the window. The window size, n, must be an odd number so that there are (n-1)/2 points on either side of the point being replaced.

Apply a window size of 5 to the Q data to see what happens:

$$Q_{\text{med}_5} := \text{medsmooth}(Q, 5)$$

Q =

	1
1	$2.186 \cdot 10^{-3}$
2	-0.49
3	0.02
4	-0.272
5	0.054
6	0.162
7	0.104
8	-0.014
9	0.17
10	0.532
11	0.251
12	-0.029
13	0.346
14	0.609
15	0.454
16	...

$Q_{\text{med}_5} =$

	1
1	$2.186 \cdot 10^{-3}$
2	-0.135
3	$2.186 \cdot 10^{-3}$
4	0.02
5	0.054
6	0.054
7	0.104
8	0.162
9	0.17
10	0.17
11	0.251
12	0.346
13	0.346
14	0.346
15	0.454
16	...

Notice that elements 1 and 2 of the smoothed data remain the same as the original data. This is because there are too few elements on either side of these points for Mathcad to evaluate the entire window. For a window size  $n$ , the first  $(n-1)/2$  data points are unchanged.

Element 3 has changed. Where does this new value come from?

Since the window size is 5, Mathcad replaces element 3 by the median of elements 1, 2, 3, 4, and 5.

$$\text{median}(Q_1, Q_2, Q_3, Q_4, Q_5) = 2.186 \times 10^{-3}$$

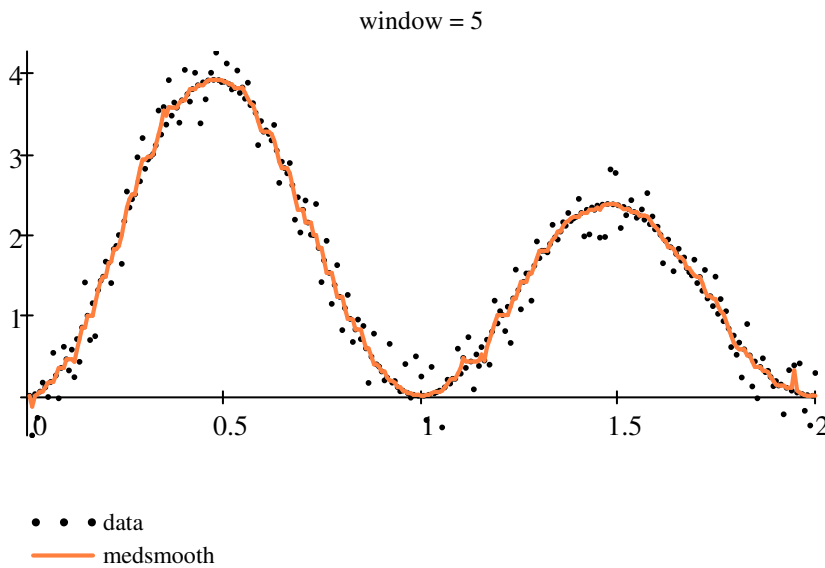
It then finds a new value for  $Q_4$  using this same technique:

$$\text{median}(Q_2, Q_3, Q_4, Q_5, Q_6) = 0.02$$

and so on.

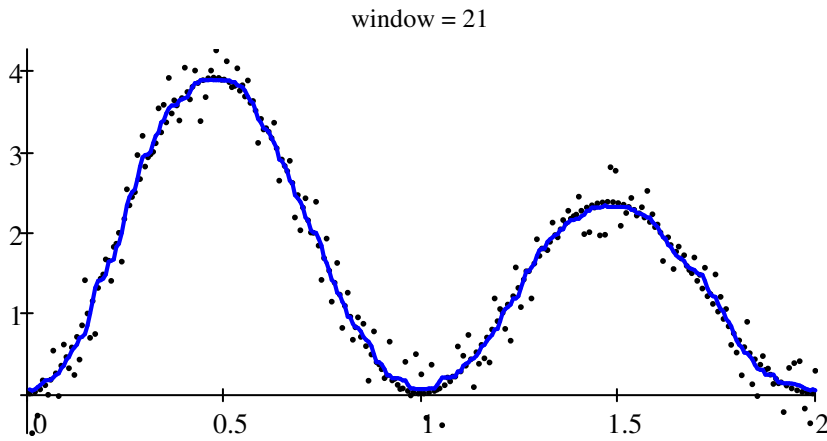
This process continues until the window size is too large to accommodate all the necessary points at the end of the data. As with the beginning of the data, the last  $(n-1)/2$  points are also unchanged.

The new data set is much smoother than the original. However, there still are several bumpy areas.



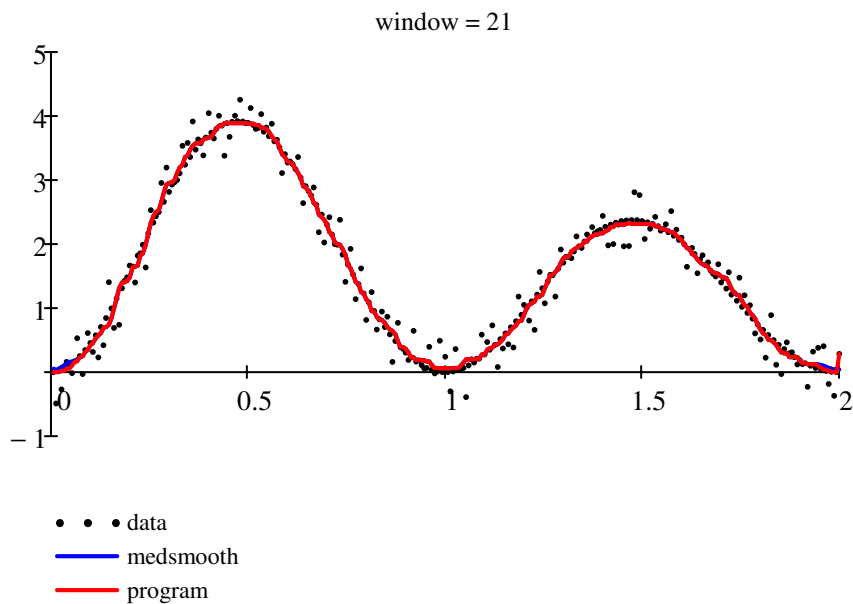
A larger window may help, but then a larger number of values at the endpoints are not smoothed.

$$Q_{\text{med\_21}} := \text{medsmooth}(Q, 21)$$



Here's a Mathcad program that uses a smaller window size when necessary.

$$\text{smooth\_ends}(\text{data}, \text{window}) := \left| \begin{array}{l} \text{for } n \in 1 \dots \text{last}(\text{data}) \\ \left| \begin{array}{l} w \leftarrow 2n - 1 \quad \text{if } n \leq \frac{\text{window} - 1}{2} \\ w \leftarrow 2 \cdot (\text{last}(Q) - n) + 1 \quad \text{if } n \geq \text{last}(Q) - \frac{\text{window} - 1}{2} \\ w \leftarrow \text{window} \quad \text{otherwise} \\ \text{smoothed}_n \leftarrow \text{medsmooth}(\text{data}, w)_n \end{array} \right. \\ \text{smoothed} \end{array} \right.$$



### Kernel Smoothing: the `ksmooth` function

Like `medsmooth`, `ksmooth` replaces each point in the data set with a modified version of itself based on the values of the surrounding points.

The `ksmooth` function takes 3 arguments:

`ksmooth(x, y, bandwidth)`

`x` - vector of real numbers, must be in ascending order

`y` - vector of real numbers

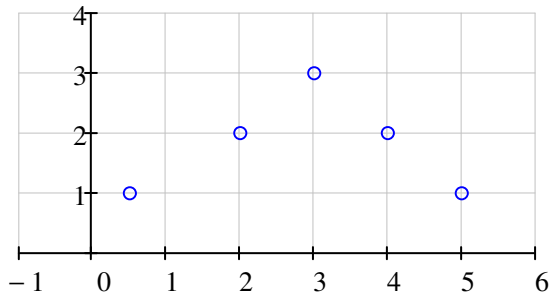
`bandwidth` - size of the smoothing window. Typically a few times the spacing between the `x` data points

There are several factors involved with kernel smoothing:

**Weighted averaging** - the current point has the most impact on the final value. As you move a greater number of points away from the central point, each point has a smaller contribution. The weights can be considered a percent each point contributes to the total, so the sum of the weights should add to 1.

Consider a small sample set of data:

$i := 1..5$	$x_i :=$	$y_i :=$										
	<table border="1" style="border-collapse: collapse; margin: auto;"> <tr><td style="padding: 2px 10px;">0.5</td></tr> <tr><td style="padding: 2px 10px;">2</td></tr> <tr><td style="padding: 2px 10px;">3</td></tr> <tr><td style="padding: 2px 10px;">4</td></tr> <tr><td style="padding: 2px 10px;">5</td></tr> </table>	0.5	2	3	4	5	<table border="1" style="border-collapse: collapse; margin: auto;"> <tr><td style="padding: 2px 10px;">1</td></tr> <tr><td style="padding: 2px 10px;">2</td></tr> <tr><td style="padding: 2px 10px;">3</td></tr> <tr><td style="padding: 2px 10px;">2</td></tr> <tr><td style="padding: 2px 10px;">1</td></tr> </table>	1	2	3	2	1
0.5												
2												
3												
4												
5												
1												
2												
3												
2												
1												



Look at the point at  $x = 3$ , and use the following relative weights:

$$w := \begin{pmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 1 \end{pmatrix} \quad \text{The point at } x = 3 \text{ is weighted three times more heavily than the points at } x = 1 \text{ and } x = 5.$$

Since the weights are actually percentages of the total contribution, divide by the sum of the weights.

$$w := \frac{w}{\sum w} \quad w = \begin{pmatrix} 0.111 \\ 0.222 \\ 0.333 \\ 0.222 \\ 0.111 \end{pmatrix}$$

The weighted average of element 3 is:

$$w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4 + w_5 \cdot x_5 = 2.944$$

See how this differs from the median method used in **medsmooth**:

$$\text{median}(x_1, x_2, x_3, x_4, x_5) = 3$$

**Kernels** - The distance to each point affects how much of an impact an element has on the final result. In the example above, elements 1 and 5 have the same weight because they are each 2 elements away from element 3. However, element 1 is farther away from the point being modified (element 3), it has a smaller effect on the results.

Manually changing the vector of weights is not efficient because when you modify other elements, they may not fall into this same pattern.

A kernel function can be used to provide a dynamic weighting mechanism that adjusts itself automatically for each point.

Kernel functions must satisfy the following criteria:

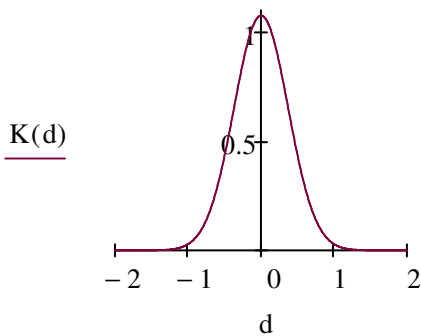
$$K(d) \geq 0 \text{ for all } d$$

$$\int_{-\infty}^{\infty} K(x) dx = 1 \quad \text{All weights "add" to 1}$$

$$K(-d) = K(d) \quad \text{It doesn't matter if you are on the left side (+d) or right side (-d) of a point, only the distance matters.}$$

**ksmooth** uses a Gaussian (Normal) kernel

$$\sigma := 0.3708158988058244 \quad K(d) := \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot e^{-\frac{d^2}{2 \cdot \sigma^2}}$$



**Bandwidth** controls the number of points considered. Unlike the window size used in **medsmooth**, this is not a number of points. Rather, it is the actual size of the window on the x-axis. This value is typically be a few times the spacing between values on the x-axis so that several points are considered.

The kernel function used in **ksmooth** can be manually defined as follows:

$$\hat{S}(j, bw) := \frac{\sum_{i=1}^{\text{last}(x)} \left( K\left(\frac{x_i - x_j}{bw}\right) \cdot y_i \right)}{\sum_{i=1}^{\text{last}(x)} K\left(\frac{x_i - x_j}{bw}\right)}$$

bw - bandwidth

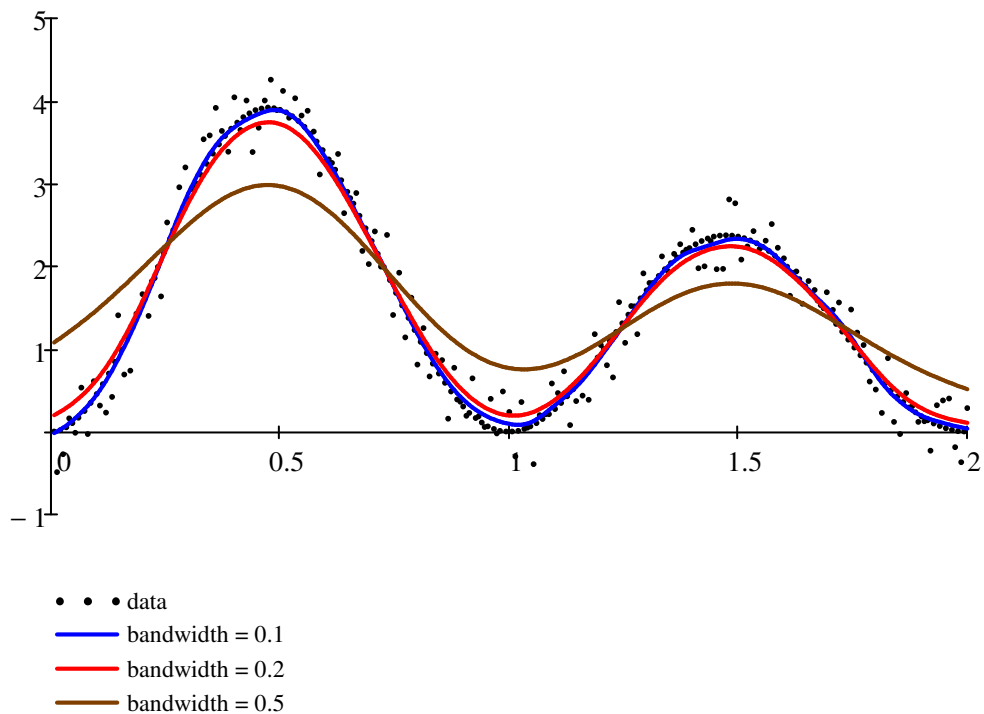
$x_j$  - the point being replaced

Below is a plot of the smoothed Q data for several different bandwidths.

$Q_{k0.1} := \text{ksmooth}(t, Q, 0.1)$

$Q_{k0.2} := \text{ksmooth}(t, Q, 0.2)$

$Q_{k0.5} := \text{ksmooth}(t, Q, 0.5)$



## Super Smoothing: the supsmooth function

The **supsmooth** function takes two arguments:

$\text{supsmooth}(x, y)$

$x$  - vector of real numbers, must be in strictly increasing order (no repeating  $x$  values)

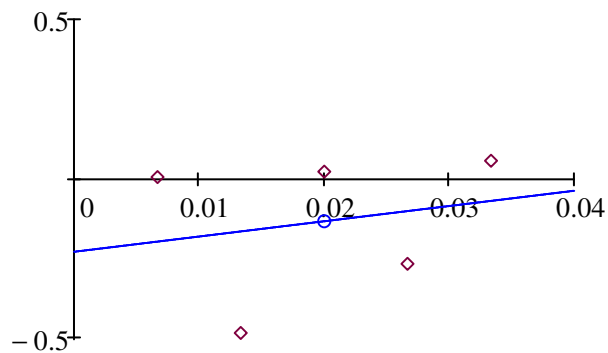
$y$  - vector of real numbers

The **supsmooth** algorithm uses a local smoother that does a localized linear fit.

Consider the first five points in  $Q$  and  $t$ . The **line** function can be used to find a best-fit line through these points. Essentially, just as **medsmooth** replaced the point with the median value of the 5 points, **supsmooth** replaces the point with the value of the best-fit line evaluated at the same  $x$  value.

$$\begin{pmatrix} b \\ m \end{pmatrix} := \text{line} \left[ \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{pmatrix}, \begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \end{pmatrix} \right]$$

$$f(x) := m \cdot x + b$$



Unlike the other smoothers, **supsmooth** does not take an argument such as window size or bandwidth. The **supsmooth** function adaptively adjusts the size of the window based on the behavior of the data.

$Q_{\text{sup}} := \text{supsmooth}(t, Q)$

