mathcad data analysis extension pack

PRECONDITIONING DATA

Smoothing

medsmooth(x,n) **VSmooth(x,W)**

ksmooth(x,y,bw) **supsmooth(x)**

The smoothing functions remove noise from data in various ways. The **medsmooth** function performs median filtering, while the new **VSmooth** performs repeated median smoothing for a scalar or vector of window widths W. The **ksmooth** function performs normal kernel filtering over a filter width bw. The **supsmooth** function performs super smoothing, a fast algorithm that uses an adjustable window to calculate a localized linear fit to the data.

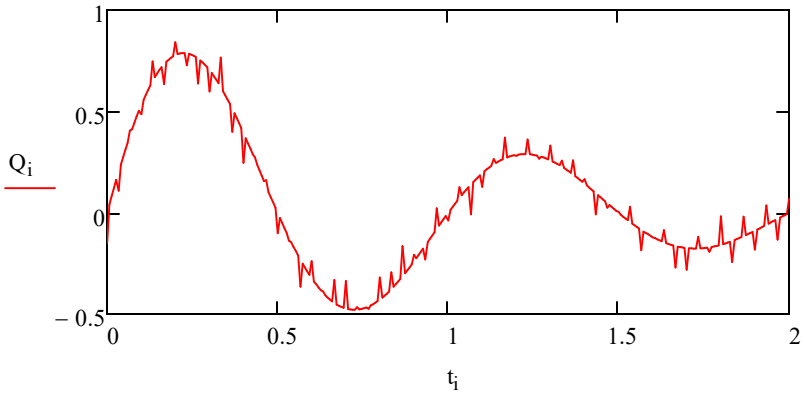
Consider some data:

$f(t) := e^{-t} \cdot \sin(2 \cdot \pi \cdot t)$ sampled at the times

$i := 0 .. 300$ $t_i := \frac{i}{150}$ $Q_i := f(t_i)$

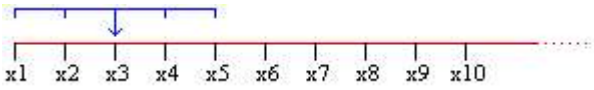
Inject noise every fifth sample,

$j := 0 .. 60$ $Q_{5 \cdot j} := Q_{5 \cdot j} + rnd(0.3) - 0.15$

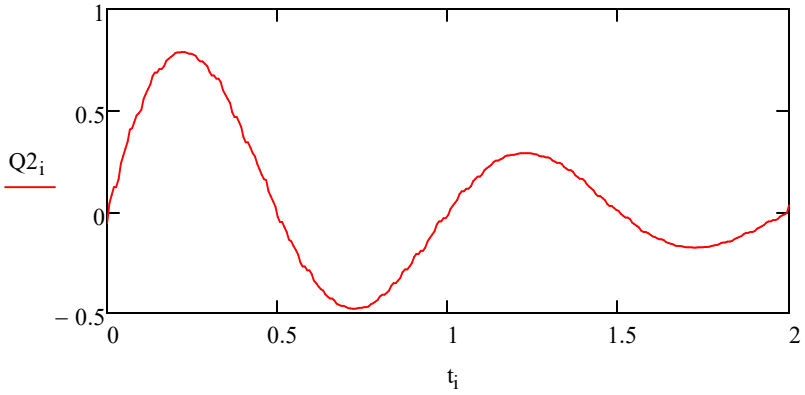


Median Smoothing

The **medsmooth** function takes a vector of real data, x, and smooths it using a window of length n. The **median** of the n points surrounding each data point is used to replace the data point, as is suggested in this diagram for a window of length 5:



$$Q2 := \text{medsmooth}(Q, 3)$$



Median smoothing is particularly useful in cases where there are sudden bursts of noise or incidents of corruption in the data. It is generally preferred over mean smoothing, which has a tendency to blur sharp features in data. You can write your own sliding window filters using a simple Mathcad program, as shown in the **Example on cosine smoothing**.

Normal kernel smoothing

Kernel smoothing takes the weighted average of the data points surrounding the point to be smoothed where weights are based both on a kernel function and on the relative distance from the central point. The **ksmooth** function uses weights given by the normal (Gaussian) distribution, scaled by the specified window bandwidth, a positive, real number. That is, given the kernel function

$$\sigma := 0.3708158988058244$$

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

which is the density function for a normal distribution, the **ksmooth** function calculates the normalized weighted quantity

$$\frac{\sum_{i=0}^{last(x)} \left(K\left(\frac{x_i - x_j}{bw} \right) \cdot y_i \right)}{\sum_{i=0}^{last(x)} K\left(\frac{x_i - x_j}{bw} \right)}$$

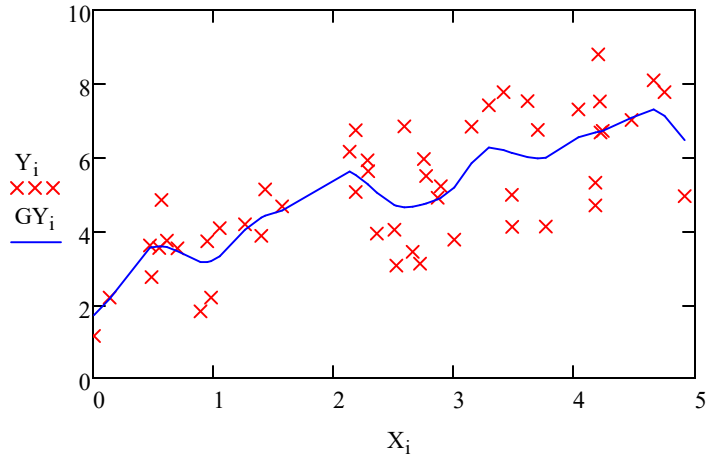
for each data point y over the range x, where bw is the bandwidth you select for third **ksmooth** argument.

$i := 0 .. 50$ $\quad X_i := rnd(5)$ $\quad X := sort(X)$

$Y_i := X_i + rnd(5)$

$bw := .5$ $\quad GY := ksmooth(X, Y, bw)$

It's important to select bandwidth carefully. Too large a bandwidth will wash out details as it averages over the whole data set. Too small a bandwidth may create artificial details in the smoothed data. Try changing bw above to numbers between 0.01 and 2 to see these effects.



Super smoothing

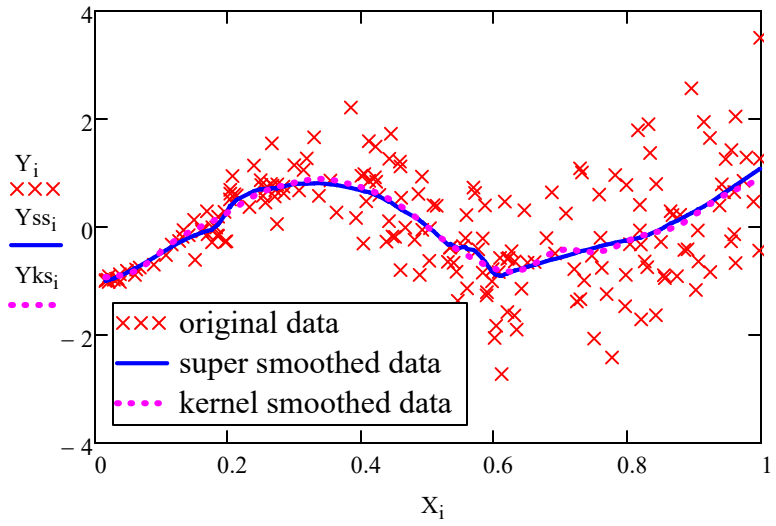
Consider a cloud-like data set created by corrupting a cosine curve:

$$M := \text{READPRN}(\text{"supex.prn"})$$

$$X := M^{\langle 0 \rangle} \qquad Y := M^{\langle 1 \rangle} \qquad i := 0 .. \text{last}(X)$$

$$Yss := \text{supsmooth}(X, Y) \qquad Yks := \text{ksmooth}(X, Y, .1)$$

The **supsmooth** function takes x data in strictly increasing order (no two x values can be the same). It implements a fast algorithm that is able to see through the cloud and pull out a periodic trend in the cloud somewhat. Similar results are obtained from kernel smoothing, but the calculation takes longer.



The **supsmooth** algorithm utilizes a local smoother that does a localized linear fit. A demonstration of the algorithm is shown in the **appendix of Mathcad programs**.

Like median smoothing, **LocLin** moves through the data, focusing on a window of values. The x- and y-values within a window are used to determine a local linear least-squares fit. The value of this linear function at the x-value centered in the window is used as the smoothed y-value. Window length is calculated for each x-value using cross-validation estimation. The localized window-adjustment makes **supsmooth** particularly useful in cases where data display different degrees of noise in different portions of the measurement.

Repeated smoothing

The smoothing process can be repeated until no additional smoothing is possible, that is, until a chosen method produces no additional changes on successive applications. The programmed function **Smooth** below demonstrates this technique. The program will also terminate after 100 applications of **medsmooth** even if this steady state is not reached.

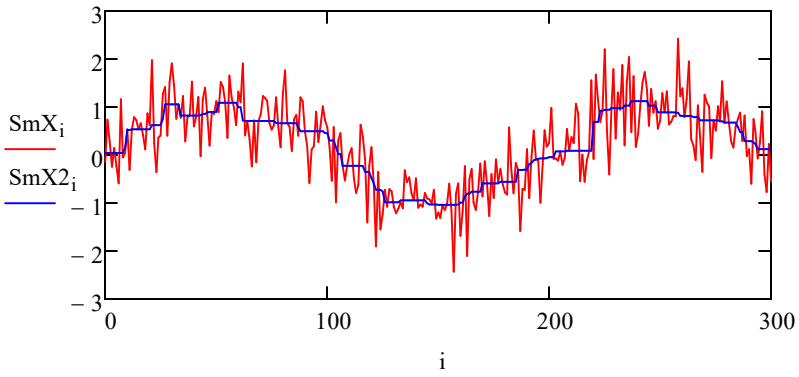
```
Smooth(x, w) := | maxit ← 100
                | sx ← medsmooth(x, w)
                | sx2 ← medsmooth(sx, w)
                | counter ← 1
                | while [1 − (sx = sx2)]·(counter < maxit)
                |     | sx ← sx2
                |     | sx2 ← medsmooth(sx, w)
                |     | counter ← counter + 1
                | sx2
```

For the data,

$$i := 0 .. 300 \qquad X_i := \sin\left(3 \cdot \pi \cdot \frac{i}{300}\right)$$

$$X := X + \text{rnorm}(301, 0, 0.5)$$

$$\text{SmX} := \text{Smooth}(X, 1) \qquad \text{SmX2} := \text{Smooth}(X, 11)$$

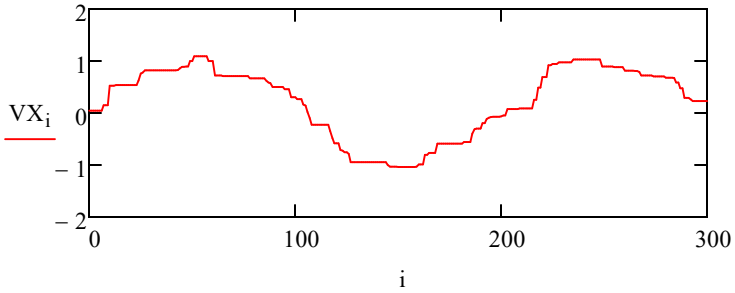


You can see the effect of different window lengths, and also the plateaus that arise from "sanding" the data too much.

A variation on the preceding method, the new built-in function **VSmooth** takes a data set x and a vector of window values W1. The function **Smooth** is applied successively to the data for each value in the window vector.

$$W1 := \begin{pmatrix} 13 \\ 9 \\ 5 \end{pmatrix}$$

$$\text{VX} := \text{VSmooth}(X, W1)$$



The data is first smoothed repeatedly until there's no change using a window of

$$W1_0 = 13$$

then applies the process to

$$W1_1 = 9$$

and finally smooths as much as possible with

$$W1_2 = 5$$

as the window width. You can also provide a scalar value for W, in which case only a single window width is used repeatedly.

$$VX1 := VSmooth(X, 9)$$

Time series

If you have time series data with no particular independent variable, other than the index of the data point itself, it is straightforward to construct another vector for the x axis for the smoothers.

data :=

	0
0	9.206
1	9.3
2	9.278
3	9.175
4	...

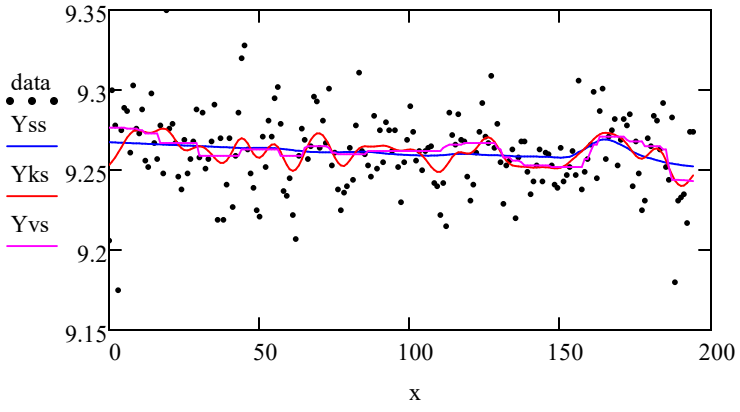
"heatflow1.txt"

$$i := 0 .. last(data) \qquad x_i := i$$

$$Yss := supsmooth(x, data)$$

$$Yks := ksmooth(x, data, 9)$$

$$Yvs := VSmooth(data, 11)$$



Other smoothers

You may wish to consider various **regression** techniques as smoothing methods, since fitting a curve to a set of data amounts to finding a smooth function that approximates the data. If you evaluate the function at every original data point in x, you will get a smoothed version of the y data. In particular, the **loess** function for fitting windowed piecewise quadratics results in a curve that is more like a smoothed version of data than a classical polynomial fit.

These smoothers form a core of basic techniques. Other special-purpose built-in functions for smoothing one and two-dimensional data sets are available in the **Signal Processing Extension Pack** and the **Image Processing Extension Pack**, available separately from PTC.