

Output Variables

LoadWave sets the following variables:

V_flag	Number of waves loaded.
S_fileName	Name of the file being loaded.
S_path	File system path to the folder containing the file.
S_waveNames	Semicolon-separated list of the names of loaded waves.

S_path uses Macintosh path syntax (e.g., "hd:FolderA:FolderB:"), even on Windows. It includes a trailing colon. If LoadWave is loading from the Clipboard, S_path is set to "".

When LoadWave presents an Open File dialog and the user cancels, V_flag is set to 0 and S_fileName is set to "".

See Also

The **ImageLoad** operation.

Loading Igor Binary Data on page II-154, **Loading Delimited Text Files** on page II-129, **Loading Fixed Field Text Files** on page II-137, **Loading General Text Files** on page II-138

LoadWave Generation of Wave Names on page II-142, **Specifying Characteristics of Individual Columns** on page II-145, **Loading Custom Date Formats** on page II-144, **LoadWave Text Encoding Issues** on page II-149

See **Importing Data** on page II-126 for further information on loading waves, including loading multidimensional data from HDF, PICT, TIFF and other graphics files. Check the "More Extensions:File Loaders" folder other file-loader extensions.

Setting Bit Parameters on page IV-12 for further details about bit settings.

Loess

```
Loess [flags] srcWave = srcWaveName [, factors = factorWaveName1
    [, factorWaveName2 ...] ]
```

The Loess operation smooths *srcWaveName* using locally-weighted regression smoothing. This algorithm is sometimes classified as a "nonparametric regression" procedure. The regression can be constant, linear, or quadratic. A robust option that ignores outliers is available. See **Basic Algorithm**, **Robust Algorithm**, and **References** for additional details and terminology.

This implementation works with waveforms, XY pairs of waves, false-color images, matrix surfaces, and multivariate data (one dependent data wave with multiple independent variable data waves).

Unlike the **FilterFIR** operation, Loess discards any NaN input values and will not generate a result that is wholly NaN.

Parameters

srcWaveName is the input data to be smoothed. It may be a one-dimensional or a two-dimensional wave, and it may contain NaNs.

When no /DEST flag is specified, Loess will overwrite *srcWaveName* with the smoothed result.

If *srcWaveName* is one-dimensional and no factors are provided, X values are derived from the X scaling of *srcWaveName*.

If *srcWaveName* is two-dimensional, the factors keyword is not permitted and the X and Y values are derived from the X and Y scaling of *srcWaveName*.

Higher dimensions of *srcWaveName* are not supported.

The optional factors parameter(s) provide the independent variable value(s) that correspond to the observed value in *srcWaveName*.

Note: Cleveland et al. (1992) use the term "multiple factors" instead of "multivariate", hence the keyword name "factors" is used to denote these waves.

Use one factors wave when *srcWaveName* is the one-dimensional Y wave of an XY data pair:

```
srcWaveName[i] = someFunction(factorWaveName1[i])
```

Use multiple factors waves when *srcWaveName* contains the values of a multivariate function. “Multivariate” means that *srcWaveName* contains the observed results of a process that combines multiple independent input variables:

```
srcWaveName[i] = someFunction(factorWaveName1[i], factorWaveName2[i],...)
```

A maximum of 10 factors waves is supported.

All factors wave(s) must be numeric, noncomplex, one-dimensional and have the same number points as *srcWaveName*.

Any NaN values in *srcWaveName*[i], *factorWaveName1*[i], *factorWaveName2*[i], ... cause all corresponding values to be ignored. Factors waves may contain NaN values only when /DFCT is specified.

Loess does not support NaNs in any of *destFactorWaveName1*, *destFactorWaveName2*,... and the results are undefined.

Flags

/CONF={*confInt*, *ciPlusWaveName* [,*ciMinusWaveName*]}

confInt specifies the confidence interval (a probability value from 0 to 1).

ciPlusWaveName and the optional *ciMinusWaveName* are the names of new or overwritten output waves to hold the fitted value \pm the confidence interval.

Note: /CONF uses large memory allocations, approximately $N*N*8$ bytes, where N is the number of points in *srcWaveName* (see **Memory Details**).

/DEST=*destWaveName* Specifies the name of the wave to hold the smoothed data. It creates *destWaveName* if it does not already exist or overwrites it if it does. The x (and possibly y) scaling of *destWave* determines the independent (factor) coordinates unless /DFCT={*destFactorWaveName1* [,*destFactorWaveName2*...]} is also specified.

/DFCT Specifies that the /DEST wave's x (and possibly y) scaling determines the independent (factor) coordinates at which to compute the smoothed data.

/DFCT={*destFactorWaveName1* [,*destFactorWaveName2*...]}

Specifies the names of one-dimensional waves providing the independent coordinates at which to compute the smoothed data.

If /DFCT={...} is used, the same number of waves must be specified for /DFCT and for factors = {*factorWaveName1* [, *factorWaveName2*...]}, though their lengths may (and usually will) be different. The length of *destFactorWaveName* waves must be the same as that of the *destWaveName* wave.

All destination factor waves must be numeric, noncomplex, and one-dimensional. The number of destination factor waves must match the number of source factor waves (if specified), or match the dimensionality of *srcWaveName* (one destination factor wave if *srcWaveName* is one-dimensional, two destination factors waves if *srcWaveName* is two-dimensional.)

The values in the destination factor waves may not be NaN.

/E=*extrapolate* Set *extrapolate* to nonzero to use a slower fitting method that computes values beyond the domain defined by the source factors. This is the “surface” parameter named “direct” in Cleveland et al. (1992). The default is *extrapolate* = 0, which uses the “interpolate” surface parameter, instead.

/N=*neighbors* Specifies the number of values in the smoothing window.

If *neighbors* is even, the next larger odd number is used. When *neighbors* is less than two, no smoothing is done.

The default is $0.5 * \text{numpts}(\text{srcWaveName})$ rounded up to the next odd integer or 3, whichever is larger.

Use either /N or /SMTH, but not both.

<code>/NORM [=norm]</code>	<p>Set <i>norm</i> to 0 when specifying multiple factors and they all have the same scale and meaning, for example multiple factors all in units of meters.</p> <p>The default is <i>norm</i> = 1, which normalizes each factor independently when computing the weighting function. This is appropriate when the factors are not dimensionally related, for example one factor measures wavelength and another measures temperature.</p>
<code>/ORD=order</code>	<p>Specifies the regression (fitting) order, the <i>d</i> parameter in Cleveland (1979):</p> <p><i>order</i>=0: Fits a constant to the locally-weighted neighbors around each point.</p> <p><i>order</i>=1: Fits a line to the locally-weighted neighbors around each point (Lowess smoothing).</p> <p><i>order</i>=2: Default; fits a quadratic (Loess smoothing).</p>
<code>/PASS=passes</code>	<p>The number of iterations of local weighting and regression fitting performed. The minimum is 1 and the default is 4. The <i>passes</i> parameter corresponds to the <i>t</i> parameter in Cleveland (1979).</p>
<code>/R [=robust]</code>	<p>Set <i>robust</i> to nonzero to use a robust fitting method that uses a bisquare weight function instead of the normal tricube weight function. This corresponds to the “symmetric” family in Cleveland et al. (1992). The robust method is less affected by outliers. The default is <i>robust</i> = 0, which is the “gaussian” family in Cleveland et al. (1992).</p>
<code>/SMTH=sf</code>	<p>Another way to express the number of values in the smoothing window, $0 \leq sf \leq 1$. The default is 0.5.</p> <p>To compute <i>neighbors</i> from <i>sf</i>, use:</p> <pre>neighbors = 1+floor(sf*numpts(srcWaveName)).</pre> <p>Use either <code>/N</code> or <code>/SMTH</code>, but not both.</p>
<code>/TIME=secs</code>	<p><i>secs</i> is the number of seconds allowed to complete the calculation before either warning (default) or stopping.</p> <p>If the stop bit (4) of <code>/V=verbose</code> is set, the calculation stops after the allotted time. If the diagnostic bit of <code>/V=verbose</code> is also set, warnings about the calculation exceeding the allotted time are printed to the history area.</p> <p>As an example, use <code>/TIME=30/V=6</code> to abort calculations longer than 30 seconds and print the warning to the history area.</p>
<code>/V [=verbose]</code>	<p>Controls how much information to print to the history area. <i>verbose</i> is a bitwise parameter with each bit controlling one aspect:</p> <p><i>verbose</i>=0: Prints nothing to the history area (default).</p> <p><i>verbose</i>=1: Prints the number of observations, equivalent number of parameters, and residual standard error.</p> <p><i>verbose</i>=2: Prints diagnostic information and error messages.</p> <p><i>verbose</i>=4: Use with <code>/TIME</code>. If this bit is set, calculations that exceed <i>secs</i> seconds are aborted.</p> <p>Set <i>verbose</i> to 6 to both limit the time and print diagnostic and error messages.</p> <p><code>/V</code> alone is the same as <code>/V=3</code>, which prints all information.</p> <p><code>S_info</code> contains all the informational messages regardless of the value of</p>
<code>/Z[=z]</code>	<p>Set <i>z</i> to nonzero to prevent an error from stopping execution. Use the <code>V_flag</code> variable to see if the smoothing succeeded.</p>

Basic Algorithm

The basic locally-weighted regression algorithm fits a constant, line, or quadratic to each point of the source data, using data that falls within the given span of neighbors over the factor data. Data outside of the span

is ignored (given a weight of zero), and data inside the span is given a weight that depends on the distance of the data from the point being evaluated: data closer to the point being evaluated have higher weights and have a greater affect on the fit.

The basic algorithm uses the “tricube” weighting function to emphasize near values and deemphasize far values. For the one-factor case (simple XY data), the weighting function can be expressed as:

$$w_i = \left(1 - \left| \frac{x - x_i}{\max_q (x - x_i)} \right|^3 \right)^3,$$

where $\max_q (x - x_i)$ is the maximum Euclidean distance of the q factor values within the given span from the factor point (x) whose observation (y) value is being evaluated.

The weights are applied to the factor values in the span to compute the constant, linear, or quadratic regression at x.

When multiple factors are used, the Euclidean distance is computed using one dimension per factor. The default is to normalize each factor’s range by the standard deviation of that factor’s values before computing the Euclidean distances. When factors are dimensionally equal, use the /NORM=0 option to skip this normalization. (See /NORM, about “dimensionally equal”.)

Robust Algorithm

The robust algorithm adds to the basic algorithm a method to identify and remove outliers by rejecting values that exceed a threshold related to the “median absolute deviation” of the basic regression’s residuals. The remaining values are used to compute robust “bisquare” weighting values:

$$r_i = \begin{cases} \left(1 - \left| \frac{e_i}{6 \cdot \text{median}(|e_i|)} \right|^2 \right)^2 & \text{for } 0 \leq |e_i| < 6 \cdot \text{median}(|e_i|) \\ 0 & \text{for } |e_i| > 6 \cdot \text{median}(|e_i|) \end{cases}$$

where e_i is the difference between the observed value and the regression’s fitted value, and $\text{median}(|e_i|)$ is evaluated for all the observed values.

These robust weighting values are multiplied with the original weighting values and a new regression (with new residuals) is computed. This process repeats 4 times by default. Use the /PASS flag to specify a different number of repetitions.

Details

Loess sets the variable V_flag to 0 if the smoothing was successful, or to an error code if not. Unlike other operations, the /Z flag allows execution to continue even if input parameters are in error.

Information printed to the history area when /V is set is always stored in the S_info string, even if /V=0 (the default). S_Info also contains the error message text if V_flag is an error code.

The error messages are described in Cleveland et al. (1992). They are often more dire than they seem.

The error message “Span too small. Fewer data values than degrees of freedom” usually means that the /SMTH or /N values are too small. The error code returned in V_Flag for this case is 1106.

The “Extrapolation not allowed with blending” (V_Flag = 1115) error usually means that the destination factors are trying to compute observations outside of the source factors domain without specifying /E=1. This happens if the /DEST destWaveName already exists and has X scaling that extends beyond the X scaling of srcWaveName. The solution is either kill the /DEST wave, limit the X scaling to the domain of the source wave, or use /E=1.

Memory Details

Loess requires a lot of memory, especially with the /CONF flag. Even without /CONF, the memory allocations exceed this approximation:

Number of bytes allocated = number of points in *srcWaveName* * 216

With /CONF, Loess can allocate large amounts of memory, approximately $N*N*8$ bytes, where N is the number of points in *srcWaveName*. The 2GB memory limit of 32-bit addressing limits *srcWaveName* to approximately 10,000 points when using /CONF.

More precisely, the memory allocation may be approximated by this function:

```
Function ComputeLoessMemory(srcPoints, numFactorsWaves, doConfidence)
  Variable srcPoints          // number of points in srcWave, aka N
  Variable numFactorsWaves    // 1 or number of factors (independent variables)
  Variable doConfidence       // true if /CONF is specified

  Variable doubles= 9 * srcPoints // 9 allocated double arrays
  doubles += 5 * numFactorsWaves * srcPoints // 5 more arrays
  doubles += (1+numFactorsWaves) * srcPoints // another array
  doubles += (1+numFactorsWaves) * srcPoints // another array
  doubles += (4+5) * srcPoints // two more arrays
  if( doConfidence )
    doubles += srcPoints*srcPoints // one HUGE array
  endif
  Variable bytes= doubles * 8
  return bytes
End

Macro DemoLoessMemory()
  Make/O wSrcPoints={10,100,1000,2000,3000,5000,7500,10000,12500,15000,20000}
  Duplicate/O wSrcPoints, loessMemory, loessMemory3, loessMemoryConf
  SetScale d, 0,0, "Points", wSrcPoints
  SetScale d, 0,0, "Bytes", loessMemory, loessMemory3, loessMemoryConf
  loessMemory= ComputeLoessMemory(wSrcPoints[p],1, 0)// 1 factor (X) no /CONF
  loessMemory3= ComputeLoessMemory(wSrcPoints[p],3, 0)// 3 factors (X,Y,Z) no /CONF
  loessMemoryConf= ComputeLoessMemory(wSrcPoints[p],1, 1)// 1 factor (X)with /CONF
  Display loessMemory vs wSrcPoints; Append loessMemory3 vs wSrcPoints
  ModifyGraph highTrip(bottom)=1e+08, rgb(loessMemory3)=(0,0,65535)
  ModifyGraph lstyle(loessMemory3)=2
  Legend
  Display loessMemoryConf vs wSrcPoints
  AutoPositionWindow
  ModifyGraph highTrip(bottom)=1e+08
End
```

Examples

1-D, factors are X scaling, output in new wave:

```
Make/O/N=200 wv=2*sin(x/8)+gnoise(1)
KillWaves/Z smoothed // ensure Loess creates a new wave
Loess/DEST=smoothed srcWave=wv // 21-point loess.
Display wv; ModifyGraph mode=3,marker=19
AppendtoGraph smoothed; ModifyGraph rgb(smoothed)=(0,0,65535)
```

1-D, output in existing wave with more points than original data:

```
Make/O/N=100 short=2*cos(x/4)+gnoise(1)
Make/O/N=300 out; SetScale/I x, 0, 99, "" out // same X range
Loess/DEST=out/DFCT/N=30 srcWave=short
Display short; ModifyGraph mode=3,marker=19
AppendtoGraph out
ModifyGraph rgb(out)=(0,0,65535),mode(out)=2,lsize(out)=2
```

1-D Y vs X wave data interpolated to waveform (Y vs X scaling) with 99% confidence interval outputs:

```
// NOx = f(EquivRatio)
// Y wave
// Note: The next 2 Make commands are wrapped to fit on the page.
Make/O/D NOx = {4.818, 2.849, 3.275, 4.691, 4.255, 5.064, 2.118, 4.602, 2.286, 0.97,
3.965, 5.344, 3.834, 1.99, 5.199, 5.283, 3.752, 0.537, 1.64, 5.055, 4.937, 1.561};

// X wave (Note that the X wave is not sorted)
Make/O/D EquivRatio = {0.831, 1.045, 1.021, 0.97, 0.825, 0.891, 0.71, 0.801, 1.074,
1.148, 1, 0.928, 0.767, 0.701, 0.807, 0.902, 0.997, 1.224, 1.089, 0.973, 0.98, 0.665};
```