

## Analysis

Overview .....	115
Analysis of Multidimensional Waves .....	115
Waveform Versus XY Data .....	115
Converting XY Data to a Waveform .....	116
Using the XY Pair to Waveform Panel .....	117
Using the Interp Function .....	117
Using the Interpolate External Operation .....	118
Dealing with Missing Values .....	119
Replace the Missing Values With Another Value .....	120
Remove the Missing Values .....	120
Work Around Gaps in Data .....	120
Replace Missing Data with Interpolated Values .....	121
Replace Missing Data Using the Interpolate XOP .....	121
Replace Missing Data Using Median Smoothing .....	121
Interpolation .....	121
Differentiation and Integration .....	122
Areas and Means .....	122
X Ranges and the Mean, faverage, and area Functions .....	123
Finding the Mean of Segments of a Wave .....	124
Area for XY Data .....	124
Wave Statistics .....	124
Histograms .....	126
Histogram Caveats .....	129
Graphing Histogram Results .....	129
Histogram Dialog .....	131
Histogram Example .....	132
Curve Fitting to a Histogram .....	132
Computing a Histogram with Logarithmic Bins .....	134
Computing an "Integrating" Histogram .....	134
Sorting .....	134
Simple Sorting .....	135
Sorting to Find the Median Value .....	135
Multiple Sort Keys .....	136
Sorting Text .....	136
MakeIndex and IndexSort Operations .....	137
Decimation .....	137
Decimation by Omission .....	137
Decimation by Smoothing .....	138
Miscellaneous Operations .....	139
WaveTransform .....	139
Compose Expression Dialog .....	140
Table Selection Item .....	140
Create Formula Checkbox .....	140

## Chapter III-7 — Analysis

---

Matrix Math Operations .....	141
Normal Wave Expressions .....	141
matrixXXX Operations .....	141
MatrixOp Operation .....	141
Matrix Commands .....	141
Macintosh and LAPACK Library .....	142
Analysis Programming .....	142
Passing Waves to User Functions and Macros .....	142
Returning Created Waves from User Functions .....	142
Returning Created Waves from Macros .....	143
Writing Functions that Process Waves .....	143
WaveSum Example .....	144
RemoveOutliers Example .....	144
LogRatio Example .....	145
WavesMax Example .....	146
WavesAverage Example .....	146
Finding the Mean of Segments of a Wave .....	147
Computing a Logarithmic Histogram .....	149
Working with Mismatched Data .....	150
References .....	151

## Overview

Igor Pro is a powerful data analysis environment. The power comes from a synergistic combination of

- An extensive set of basic built-in analysis operations
- A fast and flexible waveform arithmetic capability
- Immediate feedback from graphs and tables
- Extensibility through an interactive programming environment
- Extensibility through external code modules (XOPs and XFUNCS)

Analysis tasks in Igor range from simple experiments using no programming to extensive systems tailored for specific fields. Chapter I-2, **Guided Tour of Igor Pro**, shows examples of the former. WaveMetrics' "Peak Measurement" procedure package is an example of the latter.

This chapter presents some of the basic analysis operations and discusses the more common analyses that can be derived from the basic operations. The end of the chapter shows a number of examples of using Igor's programmability for "number crunching".

Discussion of Igor Pro's more specialized analytic capabilities is in chapters that follow.

See the WaveMetrics procedures, technical notes, and sample experiments that come with Igor Pro for more examples.

## Analysis of Multidimensional Waves

Many of the analysis operations in Igor Pro operate on 1D (one-dimensional) data. However, Igor Pro does include the following capabilities for analysis of multidimensional data:

- Multidimensional waveform arithmetic
- Matrix math operations
- Multidimensional Fast Fourier Transform
- 2D and 3D image processing operations
- 2D and 3D interpolation operations and functions

Some of these topics are discussed in Chapter II-6, **Multidimensional Waves** and in Chapter III-11, **Image Processing**. The present chapter focuses on analysis of 1D waves.

There are many analysis operations that are designed only for 1D data. Multidimensional waves will not appear in dialogs for these operations. If you invoke them on multidimensional waves from the command line or from an Igor procedure, Igor may treat the multidimensional waves as if they were 1D. For example, the Histogram operation will treat a 2D wave consisting of  $n$  rows and  $m$  columns as if it were a 1D wave with  $n*m$  rows. In some cases (e.g., WaveStats), the operation will be useful. In other cases, it will make no sense at all.

## Waveform Versus XY Data

Igor is highly adapted for dealing with waveform data. In a waveform, data values are uniformly spaced in the X dimension. This is discussed under **Waveform Model of Data** on page II-77.

If your data is uniformly spaced, you can set the spacing using the SetScale operation. This is crucial because most of the built-in analysis operations and functions need to know this to work properly.

If your data is not uniformly spaced, you can represent it using an XY pair of waves. This is discussed under **XY Model of Data** on page II-78. Some of the analysis operations and functions in Igor can *not* handle XY pairs directly. To use these, you must either make a waveform representation of the XY pair or use Igor procedures that build on the built-in routines.

## Converting XY Data to a Waveform

Sometimes the best way to analyze XY data is to make a uniformly-spaced waveform representation of it and analyze that instead. Most analysis operations are easier with waveform data. Other operations, such as the FFT, can be done *only* on waveform data. Often your XY data set is nearly uniformly-spaced so a waveform version of it is a very close approximation.

Often your XY data set is nearly uniformly-spaced so a waveform version of it is a very close approximation.

In fact, often XY data imported from other programs has an X wave that is completely unnecessary in Igor because the values in the X wave are actually a simple "series" (values that define a regular intervals, such as 2.2, 2.4, 2.6, 2.8, etc), in which case conversion to a waveform is a simple matter of assigning the correct X scaling to the Y data wave, using **SetScale** (or the Change Wave Scaling dialog):

```
SetScale/P x, xWave[0], xWave[1]-xWave[0], yWave
```

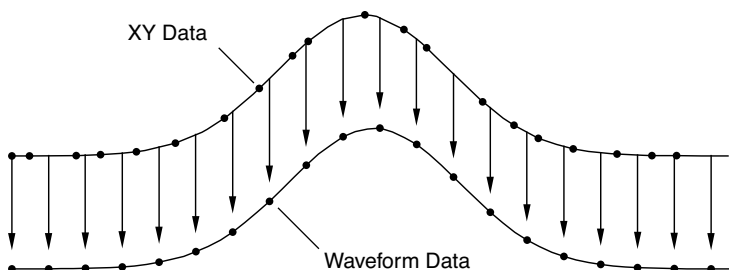
Now the X wave is superfluous and can be discarded:

```
KillWaves/Z xWave
```

The XY Pair to Waveform panel can be used to set the Y wave's X scaling when it detects that the X wave contains series data. The panel can be rather picky about what it considers a series, though, so you may need to use the SetScale command or dialog when the series is of low accuracy due to truncation or rounding in the original data set. See **Using the XY Pair to Waveform Panel** on page III-117.

If your X wave is not a series, then to create a waveform representation of XY data you need to use interpolation. To create a waveform representation of XY data you need to do interpolation. Interpolation creates a waveform from an XY pair by sampling the XY pair at uniform intervals. The diagram below shows how the XY pair defining the upper curve is interpolated to compute the uniformly-spaced waveform defining the lower curve.

Each arrow indicates an interpolated waveform value:



Igor provides three tools for doing this interpolation: The XY Pair to Waveform panel, the built-in **interp** function and the Interpolate external operation. To illustrate these tools we need some sample XY data. The following commands make sample data and display it in a graph:

```
Make/N=100 xData = .01*x + gnoise(.01)
Make/N=100 yData = 1.5 + 5*exp(-((xData-.5)/.1)^2)
Display yData vs xData
```

This creates a Gaussian shape. The x wave in our XY pair has some noise in it so the data is not uniformly spaced in the X dimension.

The x data goes roughly from 0 to 1.0 but, because our x data has some noise, it may not be monotonic. This means that, as we go from one point to the next, the x data usually increases but at some points may decrease. We can fix this by sorting the data.

```
Sort xData, xData, yData
```

This command uses the xData wave as the sort key and sorts both xData and yData so that xData always increases as we go from one point to the next.

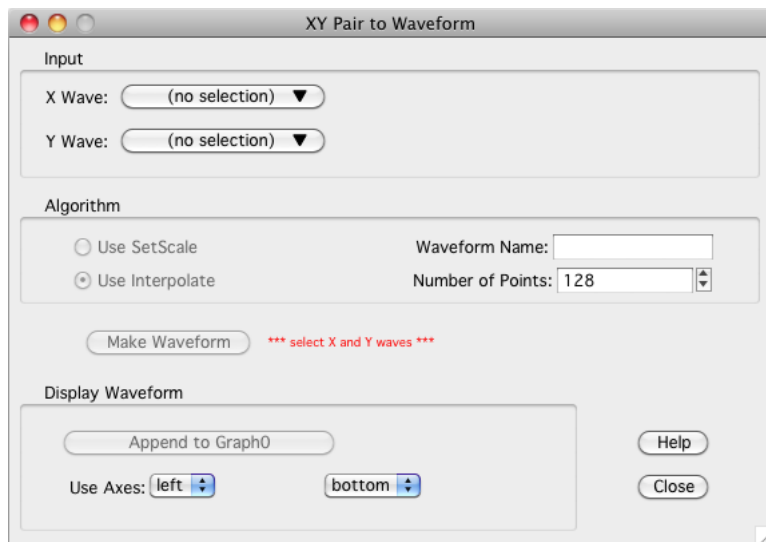
## Using the XY Pair to Waveform Panel

The XY Pair to Waveform panel creates a waveform from XY data using the **SetScale** or **Interpolate2** operations, based on an automatic analysis of the X wave's data.

The required steps are:

1. Select XY Pair to Waveform from Igor's Data→Packages submenu.

The panel is displayed:



2. Select the X and Y waves (xData and yData) in the popup menus. When this example's xData wave is analyzed it is found to be "not regularly spaced (slope error avg= 0.52...)", which means that SetScale is not appropriate for converting yData into a waveform.
3. Use Interpolate is selected here, so you need a waveform name for the output. Enter any valid wave name.
4. Set the number of output points. Using a number roughly the same as the length of the input waves is a good first attempt. You can choose a larger number later if the fidelity to the original is insufficient. A good number depends on how uneven the X values are - use more points for more unevenness.
5. Click Make Waveform.
6. To compare the XY representation of the data with the waveform representation, append the waveform to a graph displaying the XY pair. Make that graph the top graph, then click the "Append to <Name of Graph>" button.
7. You can revise the Number of Points and click Make Waveform to overwrite the previously created waveform in-place.

## Using the Interp Function

We can use the **interp** function (see page V-316) to create a waveform version of our Gaussian. The required steps are:

1. Make a new wave to contain the waveform representation.
2. Use the **SetScale** operation to define the range of X values in the waveform.
3. Use the **interp** function to set the data values of the waveform based on the XY data.

Here are the commands:

```
Duplicate yData, wData
SetScale/I x 0, 1, wData
wData = interp(x, xData, yData)
```

To compare the waveform representation to the XY representation, we append the waveform to the graph.

```
AppendToGraph wData
```

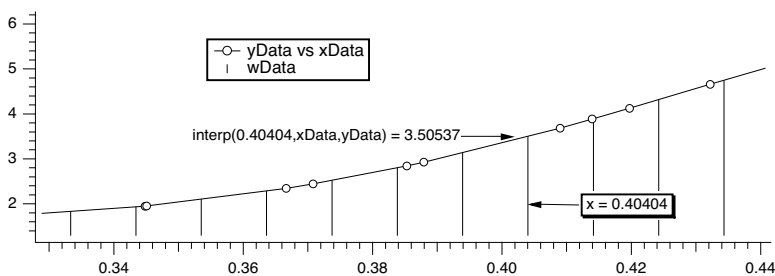
## Chapter III-7 — Analysis

Let's take a closer look at what these commands are doing.

First, we cloned `yData` and created a new wave, `wData`. Since we used `Duplicate`, `wData` will have the same number of points as `yData`. We could have made a waveform with a different number of points. To do this, we would use the `Make` operation instead of `Duplicate`.

The `SetScale` operation sets the X scaling of the `wData` waveform. In this example, we are setting the X values of `wData` to go from 0 up to and including 1.0. This means that our waveform representation will contain 100 values at uniform intervals in the X dimension from 0 to 1.0.

The last step uses a waveform assignment to set the data values of `wData`. This assignment evaluates the right-hand expression once for each point in `wData`. For each evaluation, `x` takes on a different value from 0 to 1.0. The `interp` function returns the value of the curve `yData` versus `xData` at `x`. For instance, `x=.40404` (point number 40 of `wData`) falls between two points in the XY curve. The `interp` function linearly interpolates between those values to estimate a data value of 3.50537:



We can wrap these calculations up into an Igor procedure that can create a waveform version of any XY pair.

```
Function XYToWave1(xWave, yWave, wWaveName, numPoints)
    Wave/D xWave           // X wave in the XY pair
    Wave/D yWave           // Y wave in the XY pair
    String wWaveName       // Name to use for new waveform wave
    Variable numPoints     // Number of points for waveform

    Make/O/N=(numPoints) $wWaveName // Make waveform.
    Wave wWave= $wWaveName
    WaveStats/Q xWave       // Find range of x coords.
    SetScale/I x V_min, V_max, wWave // Set X scaling for wave.
    wWave = interp(x, xWave, yWave) // Do the interpolation.
End
```

This function uses the **WaveStats** operation to find the X range of the XY pair. `WaveStats` creates the variables `V_min` and `V_max` (among others). See **Accessing Variables Used by Igor Operations** on page IV-103 for details.

The function makes the assumption that the input waves are already sorted. We left the sort step out because the sorting would be a side-effect and we prefer that procedures not have nonobvious side effects.

To use the WaveMetrics-supplied `XYToWave1` function, include the “XY Pair To Waveform” procedure file. See **The Include Statement** on page IV-145 for instructions on including a procedure file.

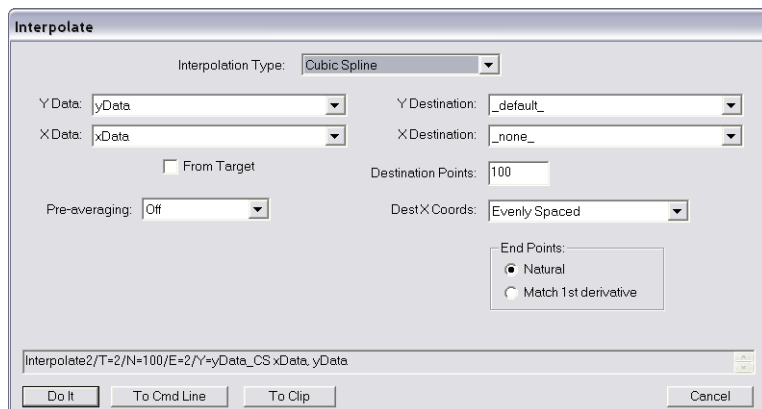
If you have blanks (NaNs) in your input data, the `interp` function will give you blanks in your output waveform as well. The `Interpolate XOP`, discussed in the next section, interpolates across gaps in data and does not produce gaps in the output.

### Using the Interpolate External Operation

The `Interpolate XOP` provides not only linear but also cubic and smoothing spline interpolation. Furthermore, it does not require the input to be sorted and can automatically make the destination waveform and set its X scaling. It also has a dialog that makes it easy to use interactively.

The Interpolate XOP is automatically installed by the Igor installer in the “Igor Extensions” folder. It adds an Interpolate item to the Analysis menu.

To use it on our sample XY data, choose Interpolate and set up the dialog as shown below.



Choosing “\_default\_” as the “Y destination” auto-names the destination wave by appending “\_CS” to the name of the input “Y data” wave. Choosing “\_none\_” as the “X destination” creates a waveform from the input XY pair rather than a new XY pair.

Here is a rewrite of the XYToWave1 function that uses the Interpolate XOP rather than the interp function.

```
Function XYToWave2(xWave, yWave, wWaveName, numPoints)
    Wave/D xWave           // x wave in the XY pair
    Wave/D yWave           // y wave in the XY pair
    String wWaveName       // name to use for new waveform wave
    Variable numPoints     // number of points for waveform

    Interpolate2/T=2/N=(numPoints)/E=2/Y=$wWaveName xWave, yWave
End
```

Gaps in the input data are ignored. Most often, this is what is desired but if you want to maintain gaps in the output data you will have to install them yourself. You can use the fact that the interp function maintains gaps to restore gaps in the Interpolate XOP output. For example:

```
yDest= yDest + 0*interp(x, xSource, ySource)
```

Only when the output of interp is a NaN is data in the destination wave affected. This works because NaN multiplied by anything (even zero) is NaN.

To use the WaveMetrics-supplied XYToWave2 function, include the “XY Pair To Waveform” procedure file. See **The Include Statement** on page IV-145 for instructions on including a procedure file.

The cubic spline algorithm used by the Interpolate XOP is derived from the cubic spline function in *Numerical Recipes in C* (see **References** on page III-151). The XOP also provides a smoothing spline based on Reinsch (1967).

## Dealing with Missing Values

A missing value is represented in Igor by the value NaN which means “Not a Number”. A missing value is also called a “blank”, because it appears as a blank cell in a table.

When a NaN is combined arithmetically with any value, the result is NaN. To see this, execute the command:

```
Print 3+NaN, NaN/5, sin(NaN)
```

By definition, a NaN is not equal to anything. Consequently, the condition in this statement:

```
if (myValue == NaN)
```

is always false.

The workaround is to use the **numtype** function:

```
if (NumType(myValue) == 2)          // Is it a NaN?
```

See also **NaNs, INFs and Missing Values** on page II-100 for more about how NaN values.

Some routines deal with missing values by ignoring them. The **CurveFit** operation (see page V-85) is one example. Others may produce unexpected results in the presence of missing values. Examples are the **FFT** operation and the **area** and **mean** functions.

Here are some strategies for dealing with missing values.

### Replace the Missing Values With Another Value

You can replace NaNs in a wave with this statement:

```
wave0 = NumType(wave0)==2 ? 0:wave0    // Replace NaNs with zero
```

If you're not familiar with the **?:** operator, see **Operators** on page IV-5.

For multi-dimensional waves you can replace NaNs using **MatrixOp**. For example:

```
Make/O/N=(3,3) matNaNTest = p + 10*q
Edit matNaNTest
matNaNTest[0][0] = NaN; matNaNTest[1][1] = NaN; matNaNTest[2][2] = NaN
MatrixOp/O matNaNTest=ReplaceNaNs(matNaNTest,0)    // Replace NaNs with 0
```

### Remove the Missing Values

For 1D waves you can remove NaNs using **WaveTransform** **zapNaNs**. For example:

```
Make/N=5 NaNTest = p
Edit NaNTest
NaNTest[1] = NaN; NaNTest[4] = NaN
WaveTransform zapNaNs, NaNTest
```

There is no built-in operation to remove NaNs from an XY pair if the NaN appears in either the X or Y wave. You can do this, however, using the **RemoveNaNsXY** procedure in the "Remove Points" **WaveMetrics** procedure file which you can access through **Help**→**Windows**→**WM Procedures Index**.

There is no operation to remove NaNs from multi-dimensional waves as this would require removing the entire row and entire column where each NaN appeared.

### Work Around Gaps in Data

Many analysis routines can work on a subrange of data. In many cases you can just avoid the regions of data that contain missing values. In other cases you can extract a subset of your data, work with it and then perhaps put the modified data back into the original wave.

Here is an example of extract-modify-replace (even though **Smooth** properly accounts for NaNs):

```
Make/N=100 data1= sin(P/8)+gnoise(.05); data1[50]= NaN
Display data1
Duplicate/R=[0,49] data1,tmpdata1    // start work on first set
Smooth 5,tmpdata1
data1[0,49]= tmpdata1[P]           // put modified data back
Duplicate/O/R=[51,] data1,tmpdata1  // start work on 2nd set
Smooth 5,tmpdata1
data1[51,]= tmpdata1[P-51]
KillWaves tmpdata1
```

## Replace Missing Data with Interpolated Values

You can replace NaN data values prior to performing operations that do not take kindly to NaNs by replacing them with smoothed or interpolated values using the **Smooth** operation (page V-577), the **Loess** operation (page V-357), or the Interpolate XOP.

## Replace Missing Data Using the Interpolate XOP

By using the same number of points for the destination as you have source points, you can replace NaNs without modifying the other data.

If you have waveform data, simply duplicate your data and perform linear interpolation using the same number of points as your data. For example, assuming 100 data points:

```
Duplicate data1,data1a
Interpolate/T=1/N=100/Y=data1a data1
```

If you have XY data, the XOP has the ability to include the input x values in the output X wave. For example:

```
duplicate data1, yData1, xData1
xData1 = x
Display yData1 vs xData1
Interpolate2/T=1/N=100/I/Y=yData1a/X=xData1a xData1,yData1
```

If, after performing an operation on your data, you wish to put the modified data back in the source wave while maintaining the original missing values you can use a wave assignment similar to this:

```
yData1 = (numtype(yData1) == 0) ? yData1 : yData1a
```

This technique can also be applied using interpolated results generated by the **Smooth** operation (page V-577) or the **Loess** operation (page V-357).

## Replace Missing Data Using Median Smoothing

You can use the Smooth dialog to replace each NaN with the median of surrounding values.

Select the Median smoothing algorithm, select "NaNs" from the Replace popup, and choose "Median" for the "with:" radio button. Enter the number of surrounding points used to compute the median (an odd number is best).

You can choose to overwrite the NaNs or create a new waveform with the result. The Smooth dialog produces commands like this:

```
Duplicate/O data1,data1_smth;DelayUpdate
Smooth/M=(NaN) 5, data1_smth
```

## Interpolation

Igor Pro has a number of interpolation tools that are designed for different applications. We summarize these in the table below.

Data	Operation/Function	Interpolation Method
1D waves	wave assignment, e.g., val=wave(x)	Linear
1D waves	<b>Smooth</b>	Running median, average, binomial, Savitsky-Golay
1D XY waves	<b>interp()</b>	Linear
1D single or XY waves	Interpolate XOP	Linear, cubic spline, smoothing spline
1D or 2D single or XY	<b>Loess</b>	Locally-weighted regression
Triplet XYZ waves	<b>ImageInterpolate</b>	Voronoi

Data	Operation/Function	Interpolation Method
1D X, Y, Z waves	Data→Packages→XYZ to Matrix	Voronoi
1D X, Y, Z waves	<b>Loess</b>	Locally-weighted regression
2D waves	<b>ImageInterpolate</b>	Bilinear, splines, Kriging, Voronoi
2D waves	<b>Interp2D()</b>	Bilinear
2D waves (points on the surface of a sphere)	<b>SphericalInterpolate</b>	Voronoi
3D waves	<b>Interp3D(), Interp3DPath, ImageTransform extractSurface</b>	Trilinear
3D scatter data	<b>Interpolate3D</b>	Barycentric

All the interpolation methods in this table consist of two common steps. The first step involves the identification of data points that are nearest to the interpolation location and the second step is the computation of the interpolated value using the neighboring values and their relative proximity. You can find the specific details in the documentation of the individual operation or function.

## Differentiation and Integration

The **Differentiate** operation (see page V-114) and **Integrate** operation (see page V-309) provide a number of algorithms for operation on one-dimensional waveform and XY data. These operations can either replace the original data or create a new wave with the results. The easiest way to use these operations is via dialogs available from the Analysis menu. These handy dialogs even provide for graphing the results.

For most applications, trapezoidal integration and central differences differentiation are appropriate methods. See the reference section for details. However, when operating on XY data, the different algorithms have different requirements for the number of points in the X wave. If your X wave does not show up in the dialog, try choosing a different algorithm or click the help button to see what the requirements are.

When operating on waveform data, X scaling is taken into account (although this can be turned off using the /P flag). You can use the **SetScale** operation (see page V-564) to define the X scaling of your Y data wave.

Although these operations work along just one dimension, they can be targeted to operate along rows or columns of a matrix (or even higher dimensions) using the /DIM flag.

The Integrate operation replaces or creates a wave with the numerical integral. For finding the area under a curve, see **Areas and Means** on page III-122.

## Areas and Means

You can compute the area and mean value of a wave in several ways using Igor.

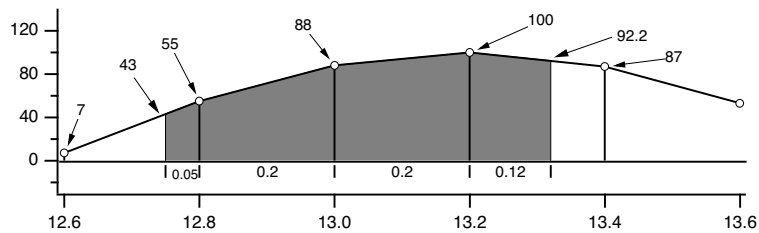
Perhaps the simplest way to compute a mean value is with the Wave Stats dialog in the Analysis menu. The dialog is pictured under **Wave Statistics** on page III-124. You select the wave, type in the X range (or use the current cursor positions), click Do It, and Igor prints several statistical results to the history area. Among them is V\_avg, which is the average, or mean value. This is the same value that is returned by the **mean** function (see page V-388), which is faster because it doesn't compute any other statistics. The mean function will return NaN if any data within the specified range is NaN. The WaveStats operation, on the other hand, ignores such missing values.

WaveStats and the mean function use the same method for computing the mean: find the waveform values within the given X range, sum them together, and divide by the number of values. The X range serves only to select the values to combine; the range is rounded to the nearest point numbers.

If you consider your data to describe discrete values, such as a count of events, then you should use either WaveStats or the mean function to compute the average value. You can most easily compute the total number of events, which is an area of sorts, using the **sum** function (see page V-682). It can also be done easily by multiplying the WaveStats outputs V\_avg and V\_npnts.

If your data is a sampled representation of a continuous process such as a sampled audio signal, you should use the faverage function to compute the mean, and the area function to compute the area. These two functions use the same linear interpolation scheme as does trapezoidal integration to estimate the waveform values between data points. The X range is *not* rounded to the nearest point; partial X intervals are included in the calculation through this linear interpolation.

The diagram below shows the calculations each function performs for the data shown. The two values 43 and 92.2 are linear interpolation estimates.



Comparison of area, faverage and mean functions over interval (12.75,13.32)

$$\text{WaveStats/R}=(12.75,13.32) \text{ wave} \\ \text{V\_avg} = (55+88+100+87)/4 = 82.5$$

$$\text{mean}(\text{wave}, 12.75, 13.32) = (55+88+100+87)/4 = 82.5$$

$$\begin{aligned} \text{area}(\text{wave}, 12.75, 13.32) &= 0.05 \cdot (43+55) / 2 && \text{first trapezoid} \\ &+ 0.20 \cdot (55+88) / 2 && \text{second trapezoid} \\ &+ 0.20 \cdot (88+100) / 2 && \text{third trapezoid} \\ &+ 0.12 \cdot (100+92.2) / 2 && \text{fourth trapezoid} \\ &= 47.082 \end{aligned}$$

$$\begin{aligned} \text{faverage}(\text{wave}, 12.75, 13.32) &= \text{area}(\text{wave}, 12.75, 13.32) / (13.32-12.75) \\ &= 47.082/0.57 = 82.6 \end{aligned}$$

Note that only the area function is affected by the X scaling of the wave. faverage eliminates the effect of X scaling by dividing the area by the same X range that area multiplied by.

One problem with these functions is that they can not be used if the given range of data has missing values (NaNs). See **Dealing with Missing Values** on page III-119 for details.

### X Ranges and the Mean, faverage, and area Functions

The X range input for the mean, faverage and area functions are optional. Thus, to include the entire wave you don't have to specify the range:

```
Make/N=10 wave=2; Edit wave.xy // X ranges from 0 to 9
Print area(wave) // entire X range, and no more
18
```

Sometimes, in programming, it is not convenient to determine whether a range is beyond the ends of a wave. Fortunately, these functions also accept X ranges that go beyond the ends of the wave.

## Chapter III-7 — Analysis

---

```
Print area(wave, 0, 9)           // entire X range, and no more
18
```

You can use expressions that evaluate to a range beyond the ends of the wave:

```
Print leftx(wave),rightx(wave)
0 10
Print area(wave,leftx(wave),rightx(wave)) // entire X range, and more
18
```

or even an X range of  $\pm x$ :

```
Print area(wave, -Inf, Inf) // entire X range of the universe
18
```

### Finding the Mean of Segments of a Wave

Under **Analysis Programming** on page III-142 is a function that finds the mean of segments of a wave where you specify the length of the segments. It creates a new wave to contain the means for each segment.

### Area for XY Data

To compute the area of a region of data contained in an XY pair of waves, use the **areaXY** function (see page V-29). There is also an XY version of the **faverage** function; see **faverageXY** on page V-152.

Technical Note 018, “Area and Integration” discusses XY area computations in greater detail and provides routines for cubic spline area and integration, area of exponential data and integration of spectroscopic or chromatographic peaks.

## Wave Statistics

The **WaveStats** operation (see page V-729) computes various descriptive statistics relating to a wave and prints them in the history area of the command window. It also stores the statistics in a series of special variables or in a wave so you can access them from a procedure.

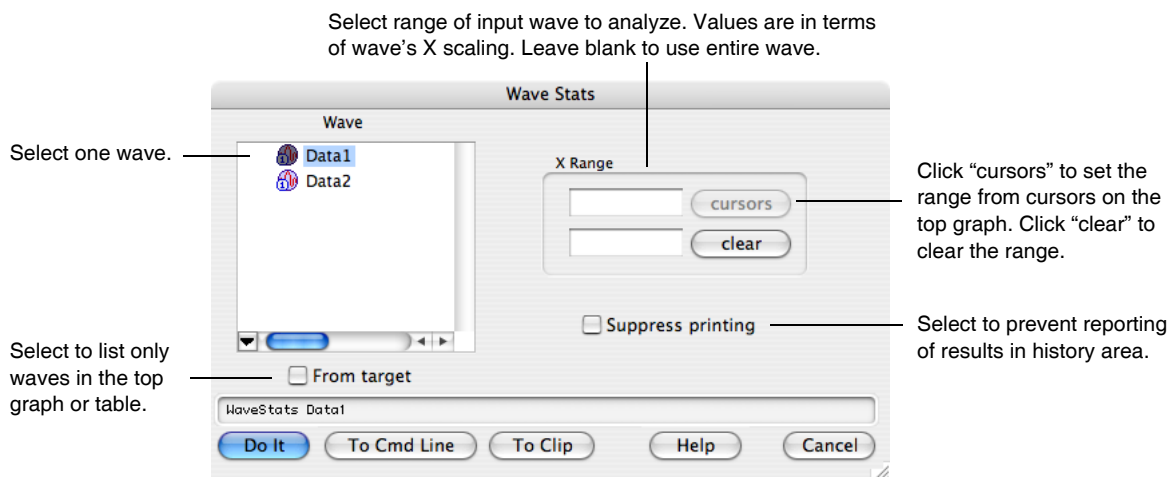
The statistics printed and the corresponding special variables are:

Variable	Meaning
V_npnts	Number of points in range, not including points whose value is NaN or INF.
V_numNaNs	Number of NaNs in range.
V_numINFs	Number of INFs in range.
V_avg	Average of data values.
V_sdev	Standard deviation of data values, $\sigma = \sqrt{\frac{1}{V\_npnts - 1} \sum (Y_i - V\_avg)^2}$ (“Variance” is $V\_sdev^2$ .)
V_rms	RMS (Root Mean Square) of Y values = $\sqrt{\left(\frac{1}{V\_npnts} \sum Y_i^2\right)}$
V_adev	Average deviation = $\frac{1}{V\_npnts} \sum_{i=0}^{V\_npnts - 1}  x_i - \bar{x} $

---

Variable	Meaning
V_skew	Skewness = $\frac{1}{V\_npnts} \sum_{i=0}^{V\_npnts-1} \left[ \frac{x_i - \bar{x}}{\sigma} \right]^3$
V_kurt	Kurtosis = $\frac{1}{V\_npnts} \sum_{i=0}^{V\_npnts-1} \left[ \frac{x_i - \bar{x}}{\sigma} \right]^4 - 3$
V_minloc	X location of minimum data value.
V_min	Minimum data value.
V_maxloc	X location of maximum data value.
V_max	maximum data value.
V_minRowLoc	Row containing minimum Z value (2D or higher waves).
V_minColLoc	Column containing minimum Z value (2D or higher waves).
V_maxColLoc	Column containing maximum Z value (2D or higher waves).
V_maxRowLoc	Row containing maximum Z value (2D or higher waves).
V_minLayerLoc	Layer containing minimum Z value (3D or higher waves).
V_maxLayerLoc	Layer containing maximum Z value (3D or higher waves).
V_minChunkLoc	Chunk containing minimum Z value (4D waves only).
V_maxChunkLoc	Chunk containing maximum Z value (4D waves only).
V_startRow	First wave point. Zero if you do not use /R.
V_endRow	Last wave point. Last point if you do not use /R.

To use the WaveStats operation, choose Wave Stats from the Analysis menu.



The WaveStats dialog expects that the range you specify, if any, be in terms of the X values of the source wave. You set this using the Change Wave Scaling dialog or **SetScale** operation (see page V-564). The WaveStats operation can use a point range, too (see page V-729).

## Chapter III-7 — Analysis

The “Suppress printing” option is normally used when you call WaveStats from an Igor procedure. The procedure uses the special variables set by WaveStats.

Igor ignores NaNs and INFs in computing the average, standard deviation, RMS, minimum and maximum. NaNs result from computations that have no defined mathematical meaning. They can also be used to represent missing values. INFs result from mathematical operations that have no finite value.

Following is a macro that illustrates the use of WaveStats. The macro shows the average and standard deviation of a source wave, assumed to be displayed in the top graph. It draws lines to indicate the average and standard deviation.

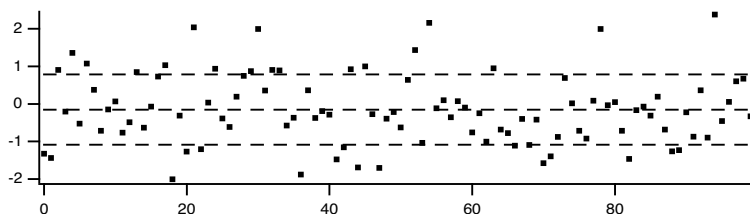
```
#pragma rtGlobals=1      // Use modern global access method.

Function ShowAvgStdDev(source)
    Wave source          // source waveform

    Variable left=leftx(source),right=rightx(source) // source X range
    WaveStats/Q source
    SetDrawLayer/K ProgFront
    SetDrawEnv xcoord=bottom,ycoord=left,dash= 7
    DrawLine left, V_avg, right, V_avg // show average
    SetDrawEnv xcoord=bottom,ycoord=left,dash= 7
    DrawLine left, V_avg+V_sdev, right, V_avg+V_sdev // show +std dev
    SetDrawEnv xcoord=bottom,ycoord=left,dash= 7
    DrawLine left, V_avg-V_sdev, right, V_avg-V_sdev // show -std dev
    SetDrawLayer UserFront
End
```

You could try this function using the following commands.

```
Make/N=100 wave0 = gnoise(1)
Display wave0; ModifyGraph mode(wave0)=2, lsize(wave0)=3
ShowAvgStdDev(wave0)
```



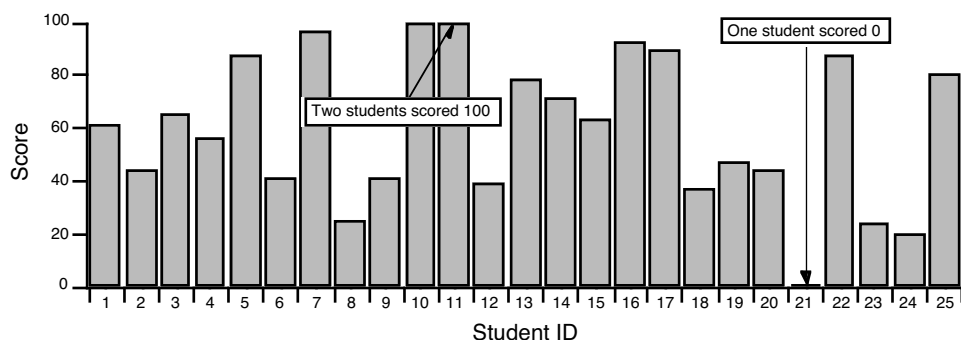
When you use WaveStats with a complex wave, you can choose to compute the same statistics as above for the real, imaginary, magnitude and phase of the wave. By default WaveStats only computes the statistics for the real part of the wave. When computing the statistics for other components, the operation stores the results in a multidimensional wave `M_WaveStats`.

If you are working with large amounts of data and you are concerned about computation speed you might be able to take advantage of the `/M` flag that limits the calculation to the first order moments.

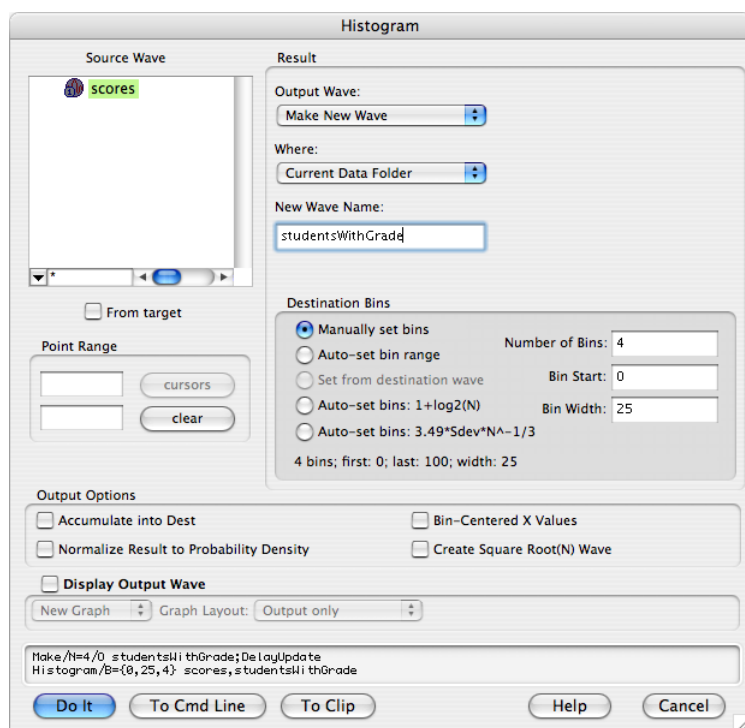
If you are working with 2D or 3D waves and you want to compute the statistics for a domain of an arbitrary shape you should use the **ImageStats** operation (see page V-287) with an ROI wave.

## Histograms

A histogram totals the number of input values that fall within each of a number of value ranges (or “bins”) usually of equal extent. For example, a histogram is useful for counting how many data values fall in each range of 0-10, 10-20, 20-30, etc. This calculation is often made to show how students performed on a test:



The usual use for a histogram in this case is to figure out how many students fall into certain numerical ranges, usually the ranges associated with grades A, B, C, and D. Suppose the teacher decides to divide the 0-100 range into 4 equal parts, one per grade. The **Histogram** operation (see page V-245) can be used to show how many students get each grade by counting how many students fall in each of the 4 ranges. Let's use the Histogram dialog and enter the obvious values:



The Histogram operation analyzes the source wave (*scores*), and puts the histogram result into a destination wave (*studentsWithGrade*).

**Note:** The Histogram operation does not produce a “bar chart”. For information on how to make a bar chart, see **Bars** on page II-252, or Chapter II-13, **Category Plots**. Also see **Graphing Histogram Results** on page III-129.

The first of the “bins” has been set manually to start at 0 and to count values up to (but not including) 25. We expect the four bins to span the range of 0–100. The Histogram dialog created the needed *studentsWithGrade* destination wave:

```
Make/N=4/D/O studentsWithGrade; DelayUpdate
```

and then used it in the Histogram operation:

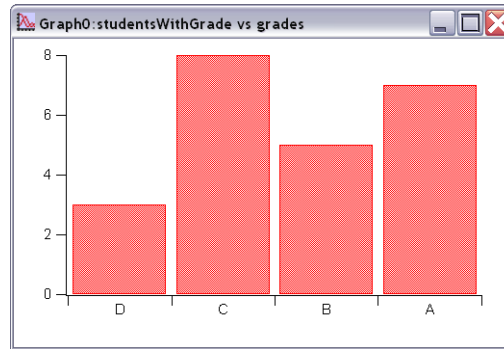
```
Histogram/B={0,25,4} scores,studentsWithGrade
```

Let's create a text wave of grades to plot *studentsWithGrade* versus a grade letter in a category plot:

## Chapter III-7 — Analysis

```
Make/O/T grades= {"D", "C", "B", "A"}
Display studentsWithGrade vs grades
SetAxis/A/E=1 left
```

Point	grades	studentsWithGrade	studentsWithGrade
0	D	0	3
1	C	25	8
2	B	50	5
3	A	75	7
4			



Everything *looks* good in the category plot. Let's double-check that all the students made it into the bins:

```
•print sum(studentswithgrade)
  23
```

There are two missing students. They are ones who scored 100 on the test. The four bins we defined are actually:

```
Bin 1:      0 - 24.99999
Bin 2:      25 - 49.99999
Bin 3:      50 - 74.99999
Bin 4:      75 - 99.99999
```

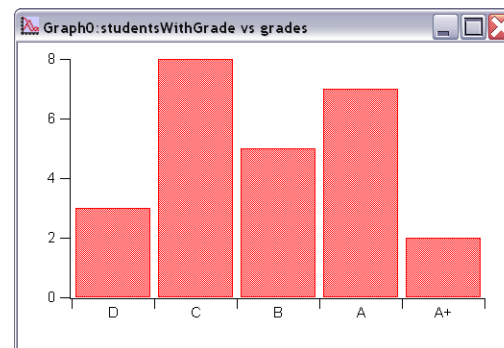
The problem is that the test scores actually encompass 101 values, not 100. To include the perfect scores in the last bin, we could add a small number such as 0.001 to the bin width:

```
Bin 1:      0 - 25.00999
Bin 2:      25.001 - 49.00199
Bin 3:      50.002 - 74.00299
Bin 4:      75.003 - 100.0399
```

The students who scored 25, 50 or 75 would be moved down one grade, however. Perhaps the best solution is to add another bin for perfect scores:

```
Make/O/T grades= {"D", "C", "B", "A", "A+"}
Histogram/B={0,25,5} scores,studentsWithGrade
```

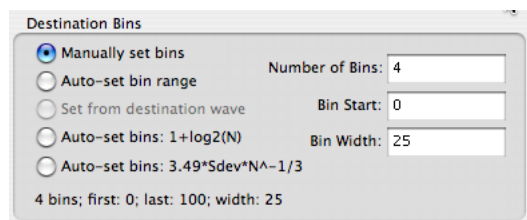
Point	grades	studentsWithGrade	studentsWithGrade
0	D	0	3
1	C	25	8
2	B	50	5
3	A	75	7
4	A+	100	2
5			



This example was intended to point out the care needed when choosing the histogram binning. Our example used "manual binning". The Histogram operation actually sets the binning in five ways:

Bin Mode	What It Does
Manual bins	Sets number of points and X scaling of the destination (output) wave based on parameters that you explicitly specify.
Auto-set bins	Sets X scaling of destination wave to cover the range of values in the <b>source</b> wave. Does not change the number of points (bins) in the destination wave. Thus, you must set the number of destination wave points before computing the histogram. When using the Histogram Dialog, if you select Make New Wave or Auto from the Output Wave menu, the dialog must be told how many points the new wave should have. It displays the Number of Bins box to let you specify the number.
Set bins from destination wave	Does not change the X scaling or the number of points in the destination wave. Thus, you need to set the X scaling and number of points of the destination wave before computing the histogram.  When using the Histogram Dialog, the Set from destination wave radio button is only available if you choose Select Existing Wave from the Output Wave menu.
Auto-set bins: $1+\log_2(N)$	Examines the input data and sets the number of bins based on the number of input data points. Sets the bin range the same as if Auto-set bin range were selected.
Auto-set bins: $3.49*Sdev*N^{-1/3}$	Examines the input data and sets the number of bins based on the number of input data points and the standard deviation of the data. Sets the bin range the same as if Auto-set bin range were selected.

These five options correspond to the radio buttons in the Histogram dialog:



**Note:** The option “Set from destination wave” is not available because the dialog Output Wave menu is set to Make New Wave. To use Set from destination wave you must choose Existing Wave in the Output Wave menu.

## Histogram Caveats

You must create the destination wave, using the Make operation or the Histogram dialog, before computing the histogram. Depending on how you want to set the histogram’s bins, you may need to set the X scaling of the destination wave also, using the **SetScale** operation (see page V-564).

The Histogram operation does not distinguish between 1D waves and multidimensional waves. If you use a multidimensional wave as the source wave, it will be analyzed as if the wave were one dimensional. This may still be useful- you will get a histogram showing counts of the data values from the source wave as they fall into bins.

If you would like to perform a histogram of 2D or 3D image data, you may want to use the **ImageHistogram** operation (see page V-261), which supports specific features that apply to images only.

## Graphing Histogram Results

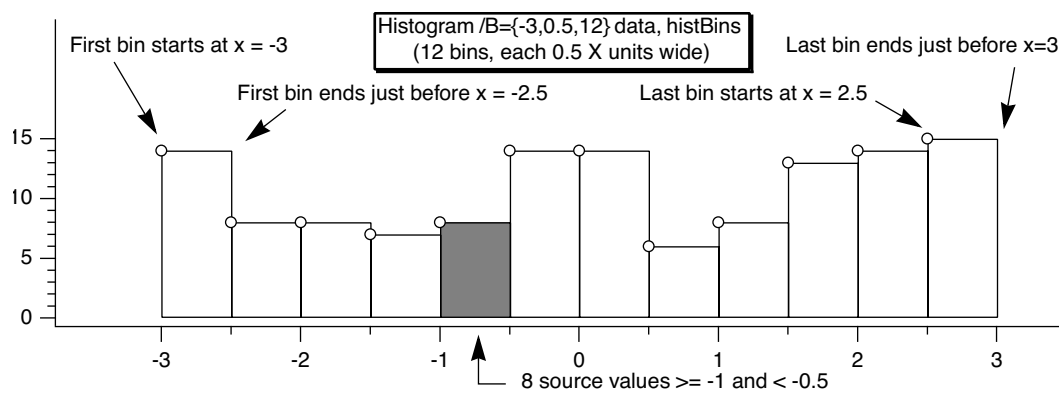
Our example above displayed the histogram results as a category plot because the bins corresponded to text values. Often histogram bins are displayed on a numeric axis. In this case you need to know how Igor displays a histogram result.

## Chapter III-7 — Analysis

For example, this histBins destination wave has 12 points (bins), the first bin starting at -3, and each bin is 0.5 wide. The X scaling is shown in the table:

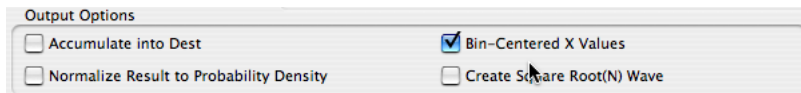
Point	histBins.x	histBins.d
0	-3	14
1	-2.5	8
2	-2	8
3	-1.5	7
4	-1	8
5	-0.5	14
6	0	14
7	0.5	6
8	1	8
9	1.5	13
10	2	14
11	2.5	15
12		

When histBins is graphed in both bars and markers modes, it looks like this:



Note that the markers are positioned at the start of the bars. You can offset the marker trace by half the bin width if you want them to appear in the center of the bin.

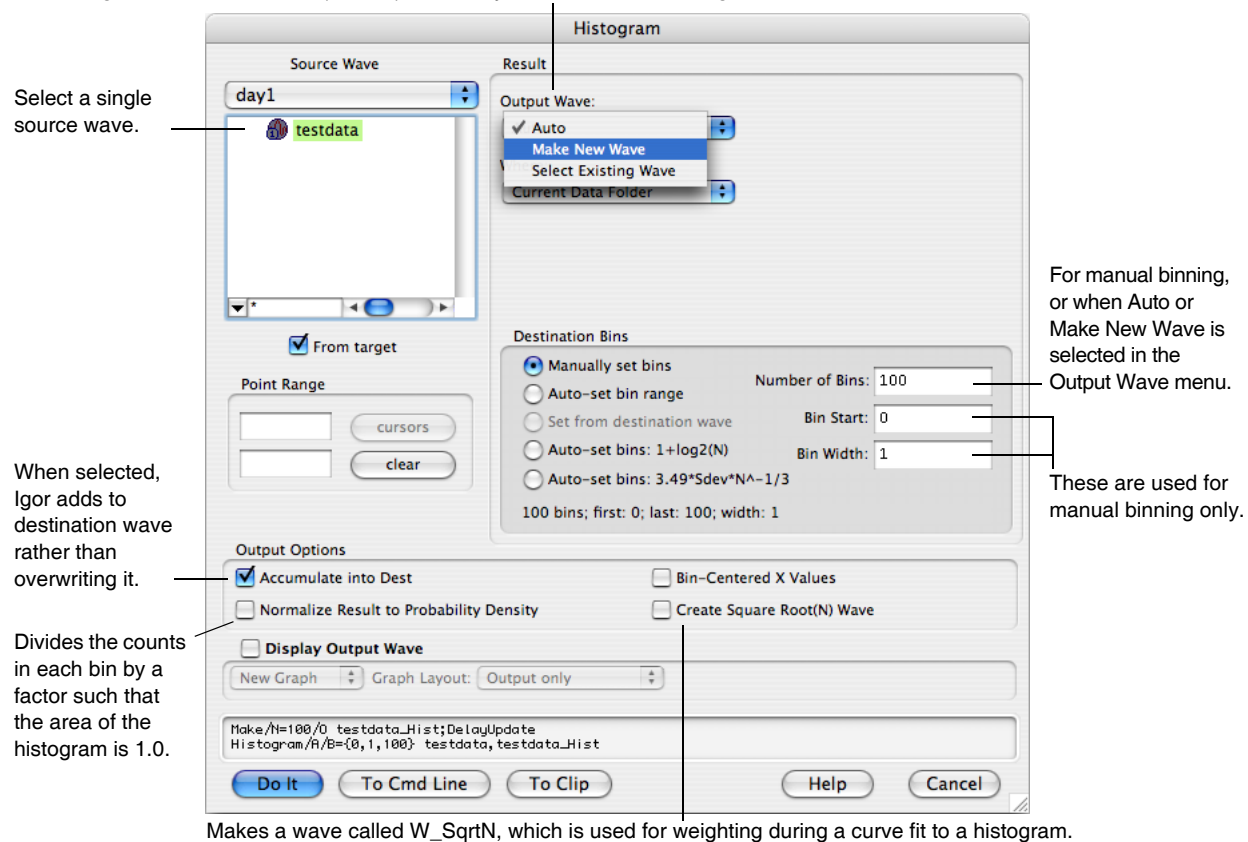
Alternatively, you can make a second histogram using the Bin-Centered X Values option:



## Histogram Dialog

To use the Histogram operation, choose Histogram from the Analysis menu.

The dialog creates a destination (“result”) wave for you, or select an existing wave.



Makes a wave called `W_SqrtN`, which is used for weighting during a curve fit to a histogram.

To use the “Manually set bins” or “Set from destination wave” bin modes, you need to decide the range of data values in the source wave that you want the histogram to cover. You can do this visually by graphing the source wave or you can use the WaveStats operation to find the exact minimum and maximum source values.

The dialog requires that you enter the starting bin value and the bin width. If you know the starting bin value and the ending bin value then you need to do some arithmetic to calculate the bin width. One way to do this is to click the To Cmd Line button and edit the Histogram command in Igor’s command line. For example, if you want 12 bins from -3 to +3, you could execute

```
Histogram/B={-3, (3 - (-3))/12, 12} test, hist
```

A line of text at the bottom of the Destination Bins box tells you the first and last values, as well as the width and number of bins. This information can help with trial-and-error settings.

If you use the “Manually set bins” or any of the “Auto-set” modes, Igor will set the X units of the destination wave to be the same as the Y units of the source wave.

If you enable the Accumulate checkbox, Histogram does not clear the destination wave. Use this to accumulate results from several histograms in one destination. If you want to do this, don’t use the “Auto-set bins” option since it makes no sense to change bins in mid-stream. Instead, use the “Set from destination wave” mode. To use the Accumulate option, the destination wave must be double- or single-precision and real.

The “Bin-Centered X Values” and “Create Square Root(N) Wave” options are useful for curve fitting to a histogram. If you do not use Bin-Centered X Values, any X position parameter in your fit function will be shifted by half a bin width. The Square Root(N) Wave creates a wave that estimates the standard deviation of the histogram data; this is based on the fact that counting data have a Poisson distribution. The wave created by this option does not try to do anything special with bins having zero counts, so if you use the

## Chapter III-7 — Analysis

---

square root(N) wave to weight a curve fit, these zero-count bins will be excluded from the fit. You may need to replace the zeroes with some appropriate value.

The binning modes were added in Igor Pro. In earlier versions of Igor, the accumulate option had *two* effects:

- Did not clear the destination wave.
- Effectively used the “Set bins from destination wave” mode.

To maintain backward compatibility, the Histogram operation still behaves this way if the accumulate (“/A”) flag is used and no bin (“/B”) flag is used. This dialog always generates a bin flag. Thus, the accumulate flag just forces accumulation and has no effect on the binning.

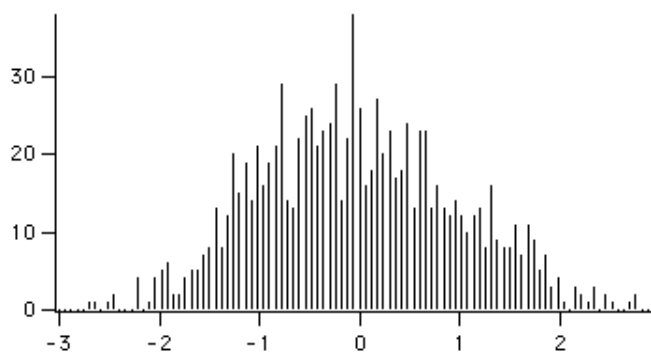
You can use the Histogram operation on multidimensional waves but they are treated as though the data belonged to a single 1D wave. If you are working with 2D or 3D waves you may prefer to use the **Image-Histogram** operation (see page V-261), which computes the histogram of one layer at a time.

### Histogram Example

The following commands illustrate a simple test of the histogram operation.

```
Make/N=1024 noise = gnoise(1) // make raw data
Make hist // make destination for histogram
Histogram/B={-3, (3 - -3)/100, 100} noise, hist // do histogram
Display hist; Modify mode(hist)=1
```

These commands produce the following graph.



### Curve Fitting to a Histogram

By default, a histogram wave has the X scaling set such that an X value gives the value at the left end of a bin. Usually if you are going to fit a curve to a histogram, you want centered X values. You can change the X scaling to centered values with this command:

```
setscale/P x leftx(hist)+deltax(hist)/2, deltax(hist), hist
```

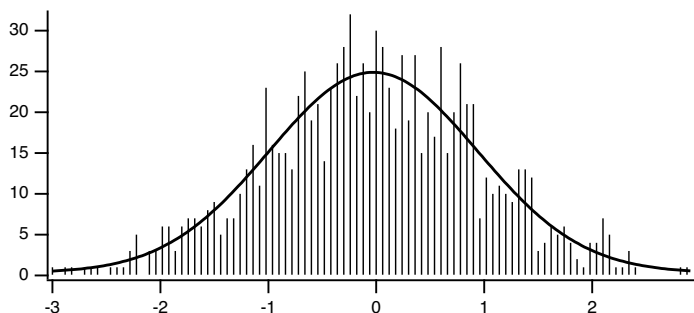
In this command, *hist* is the name of a wave containing a histogram. Substitute whatever name your wave has.

It is easier to simply use the option to have the Histogram operation produce output with bin-centered X values. Using the example from the previous section, add the /C flag:

```
Histogram/C/B={-3, (3 - -3)/100, 100} noise, hist // do histogram
```

Because the values in the source wave are Gaussian deviates generated by the *gnoise* function, the histogram should have the familiar Gaussian bell-shape. You can estimate the characteristics of the population the samples were taken from by fitting a Gaussian curve to the data. First try fitting a Gaussian curve to the example histogram:

```
CurveFit gauss hist /D // curve fit to histogram
```



The solution from the curve fit was (if you try this example on your computer, the results will be somewhat different because the random noise added by `gnoise` will be different):

```
y0      = -0.19106 ± 0.78
A       = 24.618 ± 0.871
x0      = -0.037059 ± 0.031
width   = 1.4361 ± 0.0748
```

Note that the peak position (`x0`) is shifted approximately half a bin below zero. Since the `gnoise()` function produces random numbers with mean of zero, we would expect `x0` to be close to zero. The shifted value of `x0` is a result of Igor's way of storing the X values for histogram bins. Setting the X value to the left edge is good for displaying a bar chart, but bad for curve fitting.

One solution to the problem is simply to correct the curve fit result after the fact:

```
W_coef[2] += 0.03    // add half a bin width to the peak position
```

This solution has many drawbacks, including the fact that the fit curve on the graph is still wrong because it was calculated using the uncorrected value of `x0`.

Another solution is to use the bin-centered X value option before doing the curve fit, as was done above:

```
Histogram/C/B={-3, (3 - -3)/100, 100} noise, hist // do histogram
CurveFit gauss hist /D                          // curve fit to histogram
```

The result of the curve fit is closer to what we expect:

```
y0      = -0.19106 ± 0.78
A       = 24.618 ± 0.871
x0      = -0.0070593 ± 0.031
width   = 1.4361 ± 0.0748
```

But this shifts the trace showing the histogram by half a bin on the graph. If the trace is displayed using markers or dots, this may be what is desired, but if you have used bars, the display is incorrect.

Another possibility is to make an X wave to go with the histogram data. This X wave would contain X values shifted by half a bin. Use this X wave as input to the curve fit, but don't use it on the graph:

```
Duplicate hist, hist_x
hist_x = x + deltax(hist)/2
CurveFit gauss hist /X=hist_x/D
```

Use this method to graph the original histogram wave without modifying the X scaling, so a graph using bars is correct. It also gives a curve fit that uses the center X values, giving the correct `x0`. You could also use the Histogram operation twice, once with the `/C` flag to get bin-centered X values, and once without to get the shifted X scaling appropriate for bars. Both methods have the drawback of creating an extra wave that you must track.

There is one last refinement to curve fitting to a histogram. Since the histogram represents counts, the values in a histogram should have uncertainties described by a Poisson distribution. The standard deviation of a Poisson distribution is equal to the square root of the mean, which implies that the estimated error of a histogram bin depends on the magnitude of the value. This, in turn, implies that the errors are not constant and a curve fit will give a biased solution.

## Chapter III-7 — Analysis

---

The correct solution is to use a weighting wave; use the /N flag with the Histogram operation to get the appropriate wave. This example makes a new data set using gnoise to make gaussian-distributed values, makes a histogram with bin-centered X values and the appropriate weighting wave, and then does two curve fits, one without weighting and one with:

```
Make/N=1024 gdata=gnoise(1)
Make/N=20/O gdata_Hist
Histogram/C/N/B=4 gdata,gdata_Hist
Display gdata_Hist
ModifyGraph mode=3,marker=8
CurveFit gauss gdata_Hist /D
CurveFit gauss gdata_Hist /W=W_SqrtN /I=1 /D
```

Note the “/W=W\_SqrtN /I=1” addition to the second CurveFit command; this adds the weighting using the weighting wave created by the Histogram operation. Also, /B=4 was used to have the Histogram operation set the number of bins and bin range appropriately for the input data.

The results from the unweighted fit:

```
y0    =-3.3383 ± 2.98
A      =133.26 ± 3.51
x0     =0.024088 ± 0.0252
width=1.5079 ± 0.0578
```

And from the weighted fit:

```
y0    =0.33925 ± 0.804
A      =135.21 ± 5.25
x0     =0.0038416 ± 0.031
width=1.3604 ± 0.0405
```

### Computing a Histogram with Logarithmic Bins

**Analysis Programming** on page III-142 describes how you can compute a histogram with logarithmic bins. The built-in Histogram operation can not do this.

### Computing an “Integrating” Histogram

In a histogram, each bin of the destination wave contains a count of the number of occurrences of values in the source that fell within the bounds of the bin. In an *integrating* histogram, instead of *counting* the occurrences of a value within the bin, we *add* the value itself to the bin. When we’re done, the destination wave contains the sum of all values in the source which fell within the bounds of the bin.

Igor comes with an example experiment called “Integrating Histogram” that illustrates how to do this with a user function. This experiment is in the “Examples:Analysis” folder.

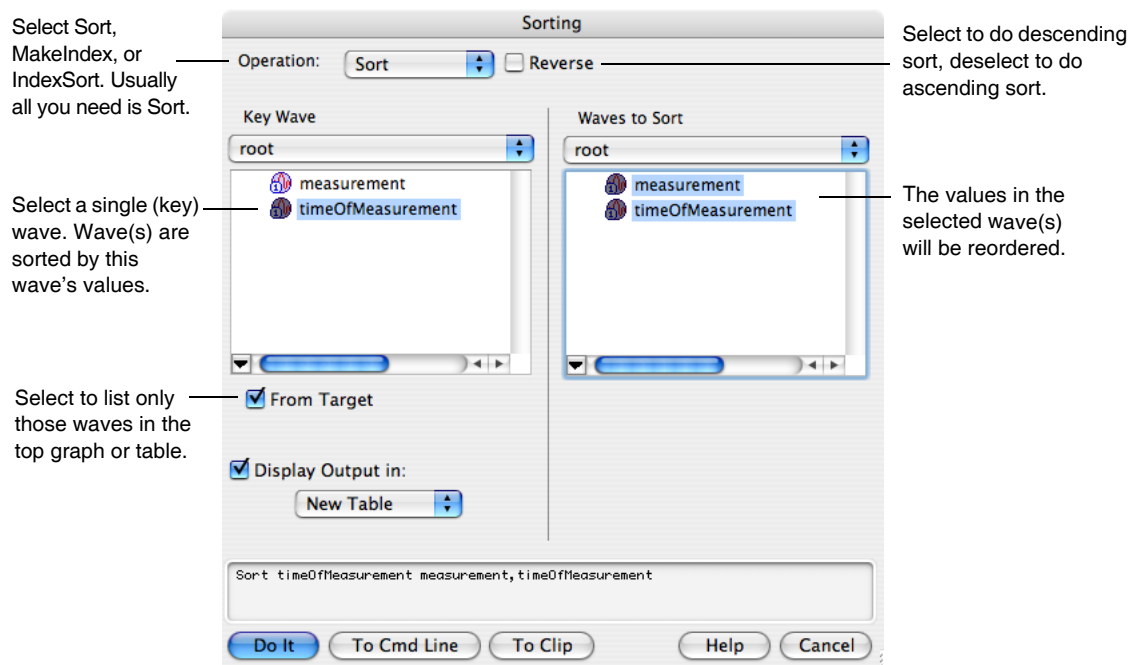
## Sorting

The **Sort** operation (see page V-581) sorts one or more 1D numeric or text waves in ascending or descending order.

The Sort operation is often used to prepare a wave or an XY pair for subsequent analysis. For example, the **interp** function assumes that the X input wave is monotonic.

There are other sorting-related operations: **MakeIndex** and **IndexSort**. These are used in rare cases and are described the section **MakeIndex and IndexSort Operations** on page III-137. Also see the **SortList** operation (page V-581).

To use the Sort operation, choose Sort from the Analysis menu.



The sort key wave controls the reordering of points. However, the key wave itself is not reordered unless it is also selected as a destination wave in the “Waves to Sort” list.

The number of points in the destination wave or waves must be the same as in the key wave. When you select a wave from the dialog’s Key Wave list, Igor shows only waves with the same number of points in the Waves to Sort list.

The key wave can be a numeric or text wave, but it must not be complex. The destination wave or waves can be text, real or complex except for the MakeIndex operation in which case the destination must be text or real.

The number of destination waves is limited by the 400 character limit in Igor’s command buffer. To sort a very large number of waves, use several Sort commands in succession, being careful not to sort the key wave until the very last.

## Simple Sorting

In the simplest case, you would select a single wave as both the source and the destination. Then, Sort would merely sort that wave.

If you want to sort an XY pair such that the X wave is in order, you would select the X wave as the source and both the X and Y waves as the destination.

## Sorting to Find the Median Value

The following user-defined function illustrates a simple use of the Sort operation to find the median value of a wave.

```
Function/D Median(w, x1, x2) // Returns median value of wave w
    Wave w
    Variable x1, x2 // range of interest

    Variable result

    Duplicate/R=(x1,x2) w, tempMedianWave // Make a clone of wave
    Sort tempMedianWave, tempMedianWave // Sort clone
    SetScale/P x 0,1,tempMedianWave
    result = tempMedianWave((numpts(tempMedianWave)-1)/2)
    KillWaves tempMedianWave
```

## Chapter III-7 — Analysis

---

```
    return result
End
```

We used the name `tempMedianWave` rather than just `temp` to minimize the chance of a conflict. It is possible that a procedure in the chain of procedures leading to `Median` has created a wave named `temp`.

It is easier and faster to use the **StatsMedian** operation (page V-640) to find the median value in a wave.

### Multiple Sort Keys

If the key wave has two or more identical values, you may want to use a secondary source to determine the order of the corresponding points in the destination. This requires using multiple sort keys. The Sorting dialog does not provide a way to specify multiple sort keys but the Sort operation does. Here is an example demonstrating the difference between sorting by single and by multiple keys. Notice that the sorted wave (`tdest`) is a text wave, and the sort keys are text (`tsrc`) and numeric (`nw1`):

```
Make/O/T tsrc={"hello","there","hello","there"}
Duplicate/O tsrc,tdest
Make nw1= {3,5,2,1}
tdest= tsrc + " " + num2str(nw1)
Edit tsrc,nw1,tdest
```

Point	tsrc	nw1	tdest
0	hello	3	hello 3
1	there	5	there 5
2	hello	2	hello 2
3	there	1	there 1
4			

Single-key text sort:

```
Sort tsrc,tdest // nw1 not used
```

Point	tsrc	nw1	tdest
0	hello	3	hello 3
1	there	5	hello 2
2	hello	2	there 1
3	there	1	there 5
4			

Execute this to scramble `tdest` again:

```
tdest= tsrc + " " + num2str(nw1)
```

Execute this to see a two key sort (`nw1` breaks ties):

```
Sort {tsrc,nw1},tdest
```

Point	tsrc	nw1	tdest
0	hello	3	hello 2
1	there	5	hello 3
2	hello	2	there 1
3	there	1	there 5
4			

The reason that “hello 3” sorts after “hello 2” is because `nw1[0] = 3` is greater than `nw1[2] = 2`.

You can sort by more than two keys by specifying more than two waves inside the braces.

### Sorting Text

You can sort text waves with the Sort operation or the Sorting dialog. See **Multiple Sort Keys** on page III-136 for an example.

By default, text sorting is case-insensitive; “hello” sorts equally with “HELLO”. You can make the sorting case-sensitive by adding the `/C` flag to the generated command. If you use the Sorting dialog, use the To Cmd Line button, and type `/C` after Sort in the generated command.

## MakeIndex and IndexSort Operations

The MakeIndex and IndexSort operations are infrequently used. You will normally use the Sort operation.

Applications of MakeIndex and IndexSort include:

- Sorting large quantities of data
- Sorting individual waves from a group one at a time
- Accessing data in sorted order without actually rearranging the data
- Restoring data to the original ordering

The MakeIndex operation creates a set of index numbers. IndexSort can then use the index numbers to rearrange data into sorted order. Together they act just like the Sort operation but with an extra wave and an extra step.

The advantage is that once you have the index wave you can quickly sort data from a given set of waves at any time. For example, if you have hundreds of waves you can not use the normal sort operation on a single command line. Also, when writing procedures it is more convenient to loop through a set of waves one at a time than to try to generate a single command line with multiple waves. This is particularly true when not all waves from a given set will fit into memory at one time.

You can also use the index values to access data in sorted order without using the IndexSort operation. For example, if you have data and index waves named wave1 and wave1index, you can access the data in sorted order on the right hand side of a wave assignment like so:

```
wave1 [wave1index [p] ]
```

If you create an index wave, you can undo a sort and restore data to the original order. To do this, simply use the Sort operation with the index wave as the source.

To understand the MakeIndex operation, consider that the following commands

```
Duplicate data1,data1index
MakeIndex data1,data1index
```

are identical in effect to

```
Duplicate data1,data1index
data1index= P
Sort data1,data1index
```

Like the Sort operation, the MakeIndex operation can handle multiple sort keys.

## Decimation

If you have a large data set it may be convenient to deal with a smaller but representative number of points. In particular, if you have a graph with hundreds of thousands of points, it probably takes a long time to draw or print the graph. You can probably do without many of the data points without altering the graph much. Decimation is one way to accomplish this.

There are at least two ways to decimate data:

1. Keep only every nth data value. For example, keep the first value, discard 9, keep the next, discard 9 more, etc. We call this **Decimation by Omission** (see page III-137).
2. Replace every nth data value with the result of some calculation such as averaging or filtering. We call this **Decimation by Smoothing** (see page III-138).

### Decimation by Omission

To decimate by omission, create the smaller output wave and use a simple assignment statement (see **Waveform Arithmetic and Assignments** on page II-93) to set their values. For example, If you are decimating by a factor of 10 (omitting 9 out of every 10 values), create an output wave with 1/10th as many points as the input wave.

## Chapter III-7 — Analysis

For example, make a 1000 point test input waveform:

```
Make/O/N=1000 wave0
SetScale x 0, 5, wave0
wave0 = sin(x) + gnoise(.1)
```

Now, make a 100 point waveform to contain the result of the decimation:

```
Make/O/N=100 decimated
SetScale x 0, 5, decimated // preserve the x range
decimated = wave0[p*10] // for(p=0;p<100;p+=1) decimated[p] = wave0[p*10]
```

Decimation by omission can be obtained more easily using the Resample operation and dialog by using an interpolation factor of 1 and a decimation factor of (in this case) 10, and a filter length of 1.

```
Duplicate/O wave0, wave0Resampled
Resample/DOWN=10/N=1 wave0Resampled
```

**Note:** Before Igor 6.01, the minimum filter length was 3. For backwards compatibility with Igor 6.0 you must instead use /N=3 and a window type whose first and last values are 0 and whose middle value is 1. Most windows do this, including the default Hanning and the Bartlett windows; /WINF=None doesn't. The result is the same as if /N=1 were specified with Igor 6.01 or later.

```
Resample/DOWN=10/N=3 wave0Resampled // Works with Igor 6.0 and 6.01 or later
```

### Decimation by Smoothing

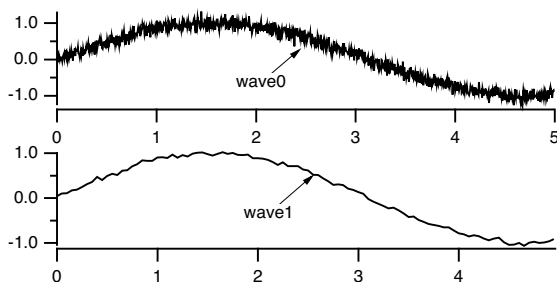
While decimation by omission completely discards some of the data, decimation by smoothing combines all of the data into the decimated result. The smoothing can take many forms: from simple averaging to various kinds of lowpass digital filtering.

The simplest form of smoothing is averaging (sometimes called “boxcar” smoothing). You can decimate by averaging some number of points in your original data set. If you have 1000 points, you can create a 100 point representation by averaging every set of 10 points down to one point. For example, make a 1000 point test waveform:

```
Make/O/N=1000 wave0
SetScale x 0, 5, wave0
wave0 = sin(x) + gnoise(.1)
```

Now, make a 100 point waveform to contain the result of the decimation:

```
Make/O/N=100 wave1
SetScale x 0, 5, wave1
wave1 = mean(wave0, x, x+9*deltax(wave0))
```



Notice that the output wave, wave1, has one tenth as many points as the input wave.

The averaging is done by the waveform assignment

```
wave1 = mean(wave0, x, x+9*deltax(wave0))
```

This evaluates the right-hand expression 100 times, once for each point in wave1. The symbol “x” returns the X value of wave1 at the point being evaluated. The right-hand expression returns the average value of wave0 over the segment that corresponds to the point in wave1 being evaluated.

It is essential that the X values of the output wave span the same range as the X values of the input range. In this simple example, the SetScale commands satisfy this requirement.

There is a WaveMetrics-supplied macro for decimation. To use it, include the “Decimation” file. **The Include Statement** on page IV-145 describes how to include a procedure file.

Results similar to the example above can be obtained more easily using the **Resample** operation (page V-525) and dialog.

Resample is based on a general sample rate conversion algorithm that optionally interpolates, low-pass filters, and then optionally decimates the data by omission. The lowpass filter can be set to “None” which averages an odd number of values centered around the retained data points. So decimation by a factor of 10 would involve averaging 11 values centered around every 10th point.

The decimation by averaging above can be changed to be 11 values centered around the retained data point instead 10 values from the beginning of the retained data point this way:

```
Make/O/N=100 wave1Centered
SetScale x 0, 5, wave1Centered
wave1Centered = mean(wave0, x-5*deltax(wave0), x+5*deltax(wave0))
```

Each decimated result (each average) is formed from different values than wave1 used, but it isn’t any less valid as a representation of the original data.

Using the Resample operation:

```
Duplicate/O wave0, wave2
Resample/DOWN=10/WINF=None/N=11 wave2 // no /UP means no interpolation
```

gives nearly identical results to the wave1Centered = mean(...) computation, the exceptions being only the initial and final values, which are simple end-effect variations.

The /WINF and /N flags of Resample define simple low-pass filtering options for a variety of decimation-by-smoothing choices. The default /WINF=Hanning window gives a smoother result than /WINF=None. See the **WindowFunction** operation (page V-738) for more about these window options.

## Miscellaneous Operations

### WaveTransform

When working with large amounts of data (many waves or multiple large waves), it is frequently useful to replace various wave assignments with wave operations which execute significantly faster. The **WaveTransform** operation (see page V-732) is designed to help in these situations. For example, to flip the data in a 1D wave you can execute the following code:

```
Function flipWave(inWave)
    wave inWave

    Variable num=numPnts(inWave)
    Variable n2=num/2
    Variable i,tmp
    num-=1
    Variable j
    for(i=0;i<n2;i+=1)
        tmp=inWave[i]
        j=num-i
        inWave[i]=inWave[j]
        inWave[j]=tmp
    endfor
End
```

## Chapter III-7 — Analysis

You can obtain the same result (about 250 times faster) using the command

```
WaveTransform/O flip waveName
```

In addition to “flip”, WaveTransform can also fill a wave with point index or the inverse point index, shift data points, normalize, convert to complex-conjugate, compute the squared magnitude or the phase, etc.

## Compose Expression Dialog

The Compose Expression item in the Analysis menu brings up the Compose Expression dialog.

The screenshot shows the 'Compose Expression' dialog box. It has several sections: 'Destination' with radio buttons for 'Wave', 'Variable', and 'String', and a dropdown menu currently showing '\_table selection\_'; 'Assignment Operator' with radio buttons for 'Assign to', 'Add to', 'Subtract from', 'Create Formula', 'Multiply by', and 'Divide by'; an 'Insert' section with dropdown menus for 'Operator', 'Wave', 'String', 'Function', 'Variable', and 'Values'; and an 'Expression:' text field containing 'NaN'. Below the text field is a preview box showing 'wave1 [0,3] = NaN'. At the bottom are buttons for 'Do It', 'To Cmd Line', 'To Clip', 'Help', and 'Cancel'. A 'Test' button is located to the right of the expression field. Four numbered annotations point to: 1. The 'Wave' radio button and the '\_table selection\_' dropdown. 2. The 'Assign to' radio button. 3. The 'Test' button. 4. The 'Test' button. A separate annotation points to the 'Insert' section, stating: 'The Insert pop-up menus insert object names and functions here.'

This dialog generates a command that sets the value of a wave, variable or string based on a numeric or string expression created by pointing and clicking. Any command that you can generate using the dialog could also be typed directly into the command line.

The command that you generate with the Compose Expression dialog consists of three parts: the destination, the assignment operator and the expression. The command resembles an equation and is of the form:

```
<destination> <assignment-operator> <expression>
```

For example:

```
wave1 = K0 + wave2           // a wave assignment command
K0 += 1.5 * K1               // a variable assignment command
str1 = "Today is" + date()   // a string assignment command
```

### Table Selection Item

The Destination Wave pop-up menu contains a “\_table selection\_” item. When you choose “\_table selection\_”, Igor assigns the expression to whatever is selected in the table. This could be an entire wave or several entire waves, or it could be a subset of one or more waves.

To use this feature, start by selecting in a table the numeric wave or waves to which you want to assign a value. Next, choose Compose Expression from the Analysis menu. Choose “\_table selection\_” in the Destination Wave pop-up menu. Next, enter the expression that you want to assign to the waves. Notice the command that Igor has created which is displayed in the command box toward the bottom of the dialog. If you have selected a subset of a wave, Igor will generate a command for that part of the wave only. Finally, click Do It to execute the command.

### Create Formula Checkbox

The Create Formula checkbox in the Compose Expression dialog generates a command using the := operator rather than the = operator. The := operator establishes a dependency such that, if a wave or variable on the right hand side of the assignment statement changes, Igor will reassign values to the destination (left hand side). We call the right hand side a formula. Chapter IV-9, **Dependencies**, provides details on dependencies and formulas.

## Matrix Math Operations

There are three basic methods for performing matrix calculations: normal wave expressions, the `matrixXXX` operations, and the **MatrixOp** operation.

### Normal Wave Expressions

You can add matrices to other matrices and scalars using normal wave expressions. You can also multiply matrices by scalars. For example:

```
Make matA={{1,2,3},{4,5,6}}, matB= {{7,8,9},{10,11,12}}
matA= matA+0.01*matB
```

gives new values for

```
matA ={{1.07,2.08,3.09},{4.1,5.11,6.12}}
```

### matrixXXX Operations

A number of matrix operations are implemented in Igor; most have names starting with the word “matrix”. For example, you can multiply a string of matrices (and column and row vectors) using the **MatrixMultiply** operation (page V-376). This operation. The `/T` flag allows you to specify that a given matrix’s data should be transposed before being used in the multiplication.

Many of Igor’s matrix operations use the LAPACK library. To learn more about LAPACK see:

*LAPACK Users’ Guide*, 3rd ed., SIAM Publications, Philadelphia, 1999.

or the LAPACK web site:

[http://www.netlib.org/lapack/lug/lapack\\_lug.html](http://www.netlib.org/lapack/lug/lapack_lug.html)

Unless noted otherwise, LAPACK routines support real or complex, IEEE single and double precision matrix waves. Most matrix operations create the variable `V_flag` and set it to zero if the operation is successful. If the flag is set to a negative number it indicates that one of the parameters passed to the LAPACK routines is invalid. If the flag value is positive it usually indicates that one of the rows/columns of the input matrix caused the problem.

### MatrixOp Operation

The **MatrixOp** operation (page V-377) improves the execution efficiency and simplifies the syntax of matrix expressions. For example, the expression

```
MatrixOp matA = (matD - matB x matC) x matD
```

is equivalent to matrix multiplications and subtraction following standard precedence rules.

### Matrix Commands

Here are the matrix math operations and functions. For full documentation, see Chapter V-1, **Igor Reference**.

*General:*

```
MatrixConvolve coefMatrix, dataMatrix
MatrixCorr [flags] waveA [, waveB]
MatrixDet(matrixA)
MatrixDot(waveA, waveB)
MatrixFilter [flags] Method dataMatrix
MatrixMultiply matrixA[/T], matrixB[/T] [, additional matrices]
MatrixOp [/O] destwave = matrixExpression
MatrixRank(matrixA [, maxConditionNumber])
MatrixTrace(matrixA)
MatrixTranspose [/H] matrix
```

*EigenValues, eigenvectors and decompositions:*

```
MatrixEigenV [flags] matrixWave
```

```
MatrixInverse [flags] srcWave  
MatrixLUD matrixA  
MatrixSchur [/Z] srcMatrix  
MatrixSVD matrixA
```

*Linear equations and least squares:*

```
MatrixGaussJ matrixA, vectorsB  
MatrixLinearSolve [flags] matrixA matrixB  
MatrixLLS [flags] matrixA matrixB  
MatrixLUBkSub matrixL, matrixU, index, vectorB  
MatrixSolve method, matrixA, vectorB  
MatrixSVBkSub matrixU, vectorW, matrixV, vectorB
```

### Macintosh and LAPACK Library

Any matrix operation that uses the LAPACK library will use Apple's veclib implementation if it is available. On computers that include Velocity Engine, single-precision matrix operations may use the Velocity Engine. It is possible to disable use of veclib using the **SetIgorOption** operation (see page V-562):

```
SetIgorOption UseVeclib=[1 or 0 or ?]
```

## Analysis Programming

This section contains data analysis programming examples. There are many more examples in the Wave-Metrics Procedures, Igor Technical Notes, and Sample Experiments folders.

### Passing Waves to User Functions and Macros

As you look through various examples you will notice two different ways to pass a wave to a function: using a Wave parameter or using a String parameter.

Using a Wave Parameter	Using a String Parameter
Function Test1(w) Wave w	Function Test2(wn) String wn
Usable in functions, not in macros.	Usable in functions and macros.
w is a "formal" name. Use it just as if it were the name of an actual wave.	Use the \$ operator to convert from a string to wave name.

The string method is used in macros and in user functions for passing the name of a wave that the function is to create or for passing the base name of a family of waves. The wave parameter method is used in user functions when the wave will always exist before the function is called. For details, see **Accessing Waves in Functions** on page IV-65.

### Returning Created Waves from User Functions

A function can return only a number or string, not a wave. But functions can return the name (or better, the full path) of the created wave(s). See also **Accessing Waves in Functions** on page IV-65.

A function that creates a single wave can be defined as a string function that returns the path to the created wave. The calling routine uses that string to refer to the wave:

```
Function CallingFunction()  
String pathToWave= fCreateANoiseWave(5)  
WAVE w = $pathToWave  
WaveStats w  
Print NameOfWave(w) // Prints (only) the name of the created wave  
End
```

```
Function/S fCreateANoiseWave(noiseValue)
  Variable noiseValue

  Make/O theNoiseWave= gnoise(noiseValue) // The same name, or pass a name
  return GetWavesDataFolder(theNoiseWave,2) // String is full path to wave
End
```

To return paths to more than one wave, use **Pass-By-Reference** on page IV-45:

```
Function CallingFunction()
  String pw1, pw2
  pbrCreateTwoNoiseWaves(5,3,pw1,pw2)
  WAVE w1 = $pw1
  WAVE w2 = $pw2
End

Function/S pbrCreateTwoNoiseWaves(noise1,noise2,path1,path2)
  Variable noise1, noise2 // Inputs
  String &path1, &path2 // Outputs (pass-by-reference ala Fortran)

  Make/O noiseWave1= gnoise(noise1)
  path1= GetWavesDataFolder(noiseWave1,2)

  Make/O noiseWave2= gnoise(noise2)
  path1= GetWavesDataFolder(noiseWave2,2)

  return 0 // Or some other useful value
End
```

## Returning Created Waves from Macros

A Macro or Proc can not return any value and functions can return only a number or string, so how can you write one that passes back to the calling routine a wave created in the subroutine?

For a macro, you'll need to take advantage of the fact that waves are global objects with names. Pass to the subroutine a name parameter to be used to create the wave. The calling routine then knows what the name of the created wave is because it supplied the name:

```
Macro CallingRoutine()
  String name="noiseWave"
  CreateANoiseWave(name,5)
  WaveStats $name
End

Proc CreateANoiseWave(name,noiseValue)
  String name // Create a wave with this name
  Variable noiseValue // with this much noise

  Make/O $name= gnoise(noiseValue)
End
```

If CreateANoiseWave needs to create more than one wave, pass more than one name parameter. This technique can also be used by functions as in the **WavesAverage Example** on page III-146.

A method that works (but is bug-prone) is to just document the name of the wave created by the called routine and use that name in the calling routines. The problem with that is when you change the name of the created wave you need to update each calling routine.

## Writing Functions that Process Waves

The user function is a powerful, general-purpose analysis tool. You can do practically any kind of analysis. However, complex analyses require programming skill and patience.

It is useful to think about an analysis function in terms of its input parameters, its return value and any side effects it may have. By return value, we mean the value that the function directly returns. For example, a function might return a mean or an area or some other characteristic of the input. By side effects, we mean

## Chapter III-7 — Analysis

---

changes that the function makes to any objects. For example, a function might change the values in a wave or create a new wave.

This table shows some of the common types of analysis functions.

Input Parameters	Return Value	Side Effects	Example Function
A source wave	A number	None	WaveArea
A source wave	Not used	The source wave is modified	RemoveOutliers
A source wave	String	A new destination wave is created	LogRatio
A source wave and a destination wave	Not used	The destination wave is modified	
The base name of a family of waves	A number	None	WavesMax
The base name of a family of waves	String	One or more new waves are created	WavesAverage

It is also possible to write an analysis function that has both a meaningful return value and side effects.

The following example functions are intended to show you the general form for some common analysis function types. We have tried to make the examples useful while keeping them simple.

### WaveSum Example

Input: Source wave  
Return value: Number  
Side effects: None

```
// WaveSum(w)
// Returns the sum of the entire wave, just like Igor's sum function.
Function WaveSum(w)
    Wave w

    Variable i, n=numpts(w), total=0
    for(i=0;i<n;i+=1)
        total += w[i]
    endfor

    return total
End
```

To use this, you would execute something like

```
Print "The sum of wave0 is:", WaveSum(wave0)
```

### RemoveOutliers Example

Input: Source wave  
Return value: Number  
Side effects: Source wave is modified

Often a user function used for number-crunching needs to loop through each point in an input wave. The following example illustrates this.

```
// RemoveOutliers(theWave, minVal, maxVal)
// Removes all points in the wave below minVal or above maxVal.
// Returns the number of points removed.
Function RemoveOutliers(theWave, minVal, maxVal)
    Wave theWave
    Variable minVal, maxVal
```

```

Variable i, numPoints, numOutliers
Variable val
numOutliers = 0
numPoints = numpnts(theWave)      // number of times to loop

for (i = 0; i < numPoints; i += 1)
  val = theWave[i]
  if ((val < minVal) || (val > maxVal)) // is this an outlier?
    numOutliers += 1
  else // if not an outlier
    theWave[i - numOutliers] = val // copy to input wave
  endif
endifor

// Truncate the wave
DeletePoints numPoints-numOutliers, numOutliers, theWave
return numOutliers
End

```

To test this function, try the following commands.

```

Make/O/N=10 wave0= gnoise(1); Edit wave0
Print RemoveOutliers(wave0, -1, 1), "points removed"

```

RemoveOutliers uses the for loop to iterate through each point in the input wave. It uses the built-in numpnts function to find the number of iterations required and the local variable p as the loop index. This is a very common practice.

The line “if ((val < minVal) || (val > maxVal))” decides whether a particular point is an outlier. || is the logical OR operator. It operates on the logical expressions “(val < minVal)” and “(val > maxVal)”. This is discussed in detail under **Bitwise and Logical Operators** on page IV-33.

To use the WaveMetrics-supplied RemoveOutliers function, include the Remove Points.ipf procedure file:

```
#include <Remove Points>
```

See **The Include Statement** on page IV-145 for instructions on including a procedure file.

### LogRatio Example

Input:           Source waves  
Return value:   String  
Side effects:    Destination wave created

```

// LogRatio(source1, source2)
// Creates a new wave that is the log of the ratio of input waves.
// Returns full path to destination wave as a string.
Function/S LogRatio(source1, source2)
  Wave source1, source2

  String destName = NameOfWave(source1) + " " + NameOfWave(source2)
  destName = CleanupName(destName,1) // obey name length limitation

  Duplicate/O source1, $destName
  WAVE dest = $destName
  dest = log(source1/source2)
  return GetWavesDataFolder(dest,2) // string is full path to wave
End

```

To use this in a macro, you would execute something like

```
Display $LogRatio(wave0, wave1)
```

To use this in a function, you would execute something like

```
String pathToWave= LogRatio(wave0, wave1)
Display $pathToWave
```

## Chapter III-7 — Analysis

---

The “Wave dest = \$destName” line creates a local wave reference which refers to the wave whose name is in the destName string variable. This is necessary because “\$destName = <expression>” is not allowed in user functions.

This simple function illustrates two commonly used techniques.

The first technique is the algorithmic derivation of a destination wave name based on a source wave name. The algorithm used here is to concatenate the two wave names. This could result in a name longer than the 31 character wave name limit, which is corrected here by the **CleanupName** function (see page V-50). A better function would check for this and use an alternate name if necessary. Another common algorithm is to derive the destination wave name by appending a suffix to the source wave name.

The second technique is the use of Duplicate/O to generate a destination wave. Using Duplicate guarantees that the destination wave has the same number of points, precision, and scaling as the source wave. Using /O (overwrite) prevents an error if the destination wave already exists.

### WavesMax Example

Input:           Base name  
Return value:   Number  
Side effects:    None

```
// WavesMax(baseName)
// Returns the maximum value in all waves whose names start with
// the specified base name.
Function WavesMax(baseName)
    String baseName

    String wn            // contains the name of a particular wave
    String wl            // contains a list of wave names
    Variable theMax
    Variable index=0

    // get list of waves whose names start with baseName
    wl = WaveList(baseName+"*", ";", "")

    theMax = -INF
    do
        wn = StringFromList(index, wl, ";") // get next wave
        if (strlen(wn) == 0)                // no more names in list?
            break                            // break out of loop
        endif
        WaveStats/Q $wn                      // WaveStats finds max value
        theMax = max(V_max, theMax)         // and puts it in V_max
        index += 1
    while (1)            // do unconditional loop

    return theMax
End
```

This function illustrates the common technique of iterating through a list of waves.

### WavesAverage Example

Input:           Base name  
Return value:   String  
Side effects:    Creates destination wave

```
// WavesAverage(baseName, destName)
// Produces a new wave, each point of which contains the average of the
// corresponding points of a number of source waves.
// All waves whose name starts with the specified base name are source waves.
// This function assumes that all waves that start with the base name have
// the same number of points and that there is at least one such wave.
```

```

// Returns full path to destination wave as a string.
Function/S WavesAverage(baseName, destName)
  String baseName      // name for source wave
  String destName      // name for destination wave

  String wn            // contains the name of a particular wave
  String wl            // contains a list of wave names
  Variable index=0

  // get list of waves whose names start with baseName
  wl = WaveList(baseName+"*", ";", "")

  // Make destination wave based on the first source wave
  wn = StringFromList(0, wl)
  Duplicate/O $wn, $destName

  WAVE dest = $destName // create wave reference for destination
  dest = 0

  do
    wn = StringFromList(index, wl) // get next wave
    if (strlen(wn) == 0) // no more names in list?
      break // break out of loop
    endif
    WAVE source = $wn // create wave reference for source
    dest += source // add source to dest
    index += 1
  while (1) // do unconditional loop

  dest /= index // divide by number of waves
  return GetWavesDataFolder(dest,2)// string is full path to wave
End

```

The name of the destination wave is passed in as a parameter. A wave with that name is created using Duplicate. We iterate through the list of waves using the **StringFromList** operation (see page V-675). We need to use WAVE references for both the source waves and the destination wave because of the limitations on the use of the \$ operator in a function.

## Finding the Mean of Segments of a Wave

An Igor user who considers each of his waves to consist of a number of segments with some number of points in each segment asked us how he could find the mean of each of these segments. We wrote the FindSegmentMeans function to do this.

```

Menu "Macros"
  "Find Segment Means", FindSegmentMeans()
End

Function FindSegmentMeans()
  String source // name of wave that we want to analyze
  Variable n // number of points in each segment
  Prompt source, "Source wave", popup WaveList("*", ";", "")
  Prompt n, "Number of points in each segment"
  DoPrompt "Find Segment Means", source, n

  SegmentMeans($source, n)
End

Function/S SegmentMeans(source, n)
  Wave source
  Variable n

```

## Chapter III-7 — Analysis

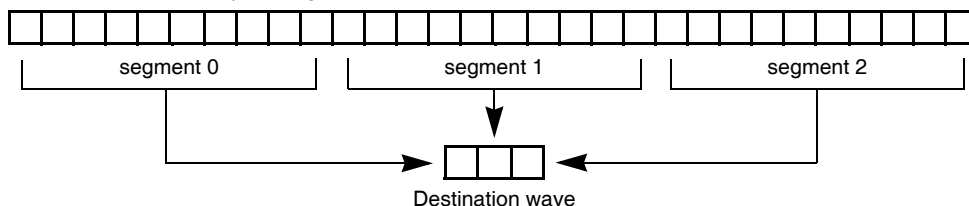
```
String dest // name of destination wave
Variable segment, numSegments
Variable startX, endX, lastX

dest = NameOfWave(source)+"_m" // derive name of dest from source
numSegments = trunc(numpts(source) / n)
if (numSegments < 1)
    DoAlert 0, "Destination must have at least one point"
    return ""
endif
Make/O/N=(numSegments) $dest
WAVE destw = $dest
lastX = pnt2x(source, numpts(source)-1)

for (segment = 0; segment < numSegments; segment += 1)
    startX = pnt2x(source, segment*n) // start X for segment
    endX = pnt2x(source, (segment+1)*n - 1) // end X for segment
    // this handles case where numpts(source)/n is not an integer
    endX = min(endX, lastX)
    destw[segment] = mean(source, startX, endX)
endfor
return GetWavesDataFolder(destw,2) // string is full path to wave
End
```

This diagram illustrates a source wave with three ten-point segments and a destination wave that will contain the mean of each of the source segments. The macro makes the destination wave.

Source wave, three 10 point segments



To test FindSegmentMeans, try the following commands.

```
Make/N=100 wave0=p+1; Edit wave0
FindSegmentMeans (wave0,10)
Append wave0_m
```

The loop index is the variable “segment”. It is the segment number that we are currently working on, and also the number of the point in the destination wave to set.

Using the segment variable, we can compute the range of points in the source wave to work on for the current iteration:  $\text{segment} \times n$  up to  $(\text{segment}+1) \times n - 1$ . Since the mean function takes arguments in terms of a wave’s X values, we use the pnt2x function to convert from a point number to an X value.

We wrote this in three parts: two functions and a menu definition. The FindSegmentMeans function uses Prompt statements along with DoPrompt to make a simple dialog for entering the parameters for the function. Users of previous versions of Igor will recognize that this capability used to be available only in macros.

The SegmentMeans function does the actual work. The two functions are partitioned this way so that you can call SegmentMeans in another function or on the command line without having to use the dialog.

Finally, the menu definition makes an entry in the Macros menu so that it is convenient to invoke the dialog.

If it is guaranteed that the number of points in the source wave is an integral multiple of the number of points in a segment, then the function can be speeded up and simplified by using a waveform assignment statement in place of the loop. Here is the statement.

```
destw = mean(source, pnt2x(source,p*n), pnt2x(source, (p+1)*n-1))
```

The variable `p`, which Igor automatically increments as it evaluates successive points in the destination wave, takes on the role of the segment variable used in the loop. Also, the `startX`, `endX` and `lastX` variables are no longer needed.

Using the example shown in the diagram, `p` would take on the values 0, 1 and 2 as Igor worked on the destination wave. `n` would have the value 10.

## Computing a Logarithmic Histogram

The built-in Histogram operation always uses bins of equal width. Here is some code that uses logarithmic bins.

This code is split into two user functions, `LogHist` and `DoLogHist`, using the same organization that we used for the previous example. The function `LogHist` calls `DoLogHist` to do the low-level, point-by-point computations. `LogHist` supplies a graphical user interface using `Prompt` and `DoPrompt`, while the `DoLogHist` function can be called from a user function without invoking a dialog. The menu definition provides a convenient way to invoke the `LogHist` function.

`LogHist` produces two destination waves - an XY pair. The X wave holds the coordinates of the start of the bins and the Y wave holds the counts. The waves are named using the name of the source wave with “\_hx” and “\_hy” suffixes.

```
#pragma rtGlobals=1          // Use modern global access method.

Menu "Analysis"
  "Log Histogram...", LogHist()
End

// DoLogHist(sw, dwX, dwY, startX, logDeltaX)
// Creates the logarithmic histogram of the source wave by summing the
// appropriate numbers into the destination y wave.
// sw is the source wave.
// dwX is the destination x wave.
// dwY is the destination y wave.
// startX, logDeltaX are explained below in LogHist().
Function DoLogHist(sw, dwX, dwY, startX, logDeltaX)
  Wave sw, dwX, dwY
  Variable startX, logDeltaX
  Variable pt, pp, dpnts, spnts

  // first find bin edges and put them in dwX
  dpnts = numpnts(dwX)
  for (pt = 0; pt < dpnts; pt += 1)
    dwX[pt] = 0^(pt*logDeltaX+startX) // this value is 10^startX when p == 0
  endfor

  // now find which bin of dwY each Y value in sw belongs in and increment it.
  spnts = numpnts(sw)
  for (pt = 0; pt < spnts; pt += 1)
    pp = (log(sw[pt]) - startX) / logDeltaX
    if ( pp == limit(pp,0,dpnts) ) // unless it is out of range or NaN
      dwY[pp] += 1
    endif
  endfor
End

// LogHist(sourceWave, numDecades, startDecade, binsPerDecade)
// Creates XY pair of waves that represent the logarithmic histogram of the
// source wave. If the source wave is named "data" then the output waves will
// be named "data_hx" and "data_hy".
// The product of numDecades and binsPerDecade specifies the number of bins in
// the histogram.
// startDecade specifies the X coordinate of the left edge of first bin. The bin
// starts at 10^startDecade.
// binsPerDecade specifies the bin width. Values less than 1 result in bins that
// span multiple decades. For example, set binsPerDecade to 0.5 to create bins
// that span two decades.
// Example
// Make/N=100 test = 10^(1+abs(gnoise(3)))
// Display test; ModifyGraph log(left)=1, mode=8,msize=2
// LogHist("test",12,0,1)
```

## Chapter III-7 — Analysis

---

Function LogHist()

```
String sourceWave= StrVarOrDefault("root:Packages:LogHist:sourceWave","_demo_")
Variable numDecades = NumVarOrDefault("root:Packages:LogHist:numDecades",10)
// first bin at 0.0001
Variable startDecade = NumVarOrDefault("root:Packages:LogHist:startDecade",-4)
Variable binsPerDecade= NumVarOrDefault("root:Packages:LogHist:binsPerDecade",1)
Prompt sourceWave, "Source wave", popup "_demo_";+WaveList(";", ";", "")
Prompt startDecade, "start decade (first bin starts at 10^startDecade)"
Prompt numDecades, "Number of decades in destination waves"
Prompt binsPerDecade, "bins per decade"
DoPrompt "Log Histogram", sourceWave, numDecades, startDecade, binsPerDecade

if( CmpStr(sourceWave,"_demo_") == 0 )
  sourceWave= "demoData"
  Make/O/N=100 $sourceWave=0.0001+10^(gnoise(2)) // about 10^-4 to 10^6
  CheckDisplayed/A $sourceWave
  if( V_Flag == 0 )
    Display $sourceWave; ModifyGraph log(left)=1, mode=8,msize=2
  endif
  startDecade=-4
  numDecades=10
  binsPerDecade=1
endif
if( binsPerDecade < 0 )
  binsPerDecade = 1
endif
// Save values for next attempt
NewDataFolder/O root:Packages
NewDataFolder/O root:Packages:LogHist
String/G root:Packages:LogHist:sourceWave = sourceWave
Variable/G root:Packages:LogHist:numDecades= numDecades
Variable/G root:Packages:LogHist:startDecade= startDecade
Variable/G root:Packages:LogHist:binsPerDecade= binsPerDecade
String destXWave, destYWave
// Concoct names for dest waves.
// This does not work if sourceWave is a full or partial path requiring single
// quotes (e.g., root:Data:'wave 0').
Variable numBins= numDecades * binsPerDecade
Variable logDeltaX=1/binsPerDecade // Log delta X (1 gives 1 decade per bin)
destXWave = sourceWave + "_hx"
destYWave = sourceWave + "_hy"
Make/O/N=(numBins+1) $destXWave=0, $destYWave=0
DoLogHist($sourceWave, $destXWave, $destYWave, startDecade, logDeltaX)
CheckDisplayed/A $destYWave
if( V_Flag == 0 )
  Display $destYWave vs $destXWave
  AutoPositionWindow/E/M=1
  ModifyGraph mode=4, marker=19, log(bottom)=1
endif
End
```

To test LogHist, choose “Log Histogram” from the Analysis menu, and choose “\_demo\_” from the Source wave pop-up menu in the resulting dialog.

To use the WaveMetrics-supplied logarithmic histogram procedures, include the “Log Histogram” procedure file. See **The Include Statement** on page IV-145 for instructions on including a procedure file.

### Working with Mismatched Data

Occasionally, you may find yourself with several sets of data each sampled at a slightly different rate or covering a different range of the independent variable (usually time). If all you want to do is create a graph showing the relationship between the data sets then there is no problem.

However, if you want to subtract one from another or do other arithmetic operations then you will need to either:

- Create representations of the data that have matching X values. Although each case is unique, usually you will want to use the Interpolate XOP (see **Using the Interpolate External Operation** on page III-118) or the interp function (see **Using the Interp Function** on page III-117) to create data sets with common X values. You can also use the **Resample** operation (page V-525) to create a wave to match another.

- Properly set each wave's X scaling, and perform the waveform arithmetic using X scaling values and Igor's automatic linear interpolation. See **Mismatched Waves** on page II-99.

The WaveMetrics procedure file Wave Arithmetic Panel uses these techniques to perform a variety of operations on data in waves. You can access the panel by choosing Packages→Wave Arithmetic from the Analysis menu. This will open the procedure file and display the control panel. Click the help button in the panel to learn how to use it.

## References

Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in C*, 2nd ed., 994 pp., Cambridge University Press, New York, 1992.

Reinsch, Christian H., Smoothing by Spline Functions, *Numerische Mathematic*, 10, 177-183, 1967.

