the derivatives is used to more precisely locate the center and edges. The peak value is simply the greater of the two unsmoothed values surrounding the peak center (if /N, then the lesser value).

FindPeak is not a high-accuracy measurement routine; it is intended as a simple peak-finder. Use the **PulseStats** operation for more precise statistics.

Without /M, a peak is found where the derivative crosses zero, regardless of the peak height.

If you use the /M=*minLevel* flag, FindPeak ignores peaks that are lower than *minLevel* (i.e., the Y value of a found peak will exceed *minLevel*) in the box-smoothed input wave. If /N is also specified (search for minimum), FindPeak ignores peaks whose amplitude is greater than *minLevel* (i.e., the Y value of a found peak will be *less* than *minLevel*).

Without /I, a peak must have two successive values that exceed *minLevel*. Use /I when you are searching for peaks that may have only one value exceeding *minLevel*.

The search for the peak begins at *startX* (or the first point of the wave if /R is not specified), and ends at *endX* (or the last point of the wave if no /R). Searching backwards is permitted, and exchanges the values of V_LeadingEdgeLoc and V_TrailingEdgeLoc.

A simple automatic peak-finder is implemented in the "Peak Autofind.ipf" procedure file used in the Multipeak Fitting package. Execute `DisplayHelpTopic "Multipeak Fitting"` for details.

The FindPeak operation is not multidimensional aware. See **Analysis on Multidimensional Waves** on page II-95 for details.

See Also

The **PulseStats** operation, the **FindLevel** operation for details about the /B, /P, /Q, and /R flag values.

FindPointsInPoly

`FindPointsInPoly` ***`xWaveName, yWaveName, xPolyWaveName, yPolyWaveName`***

The FindPointsInPoly operation determines if points fall within a certain polygon. It can be used to write procedures that operate on a subset of data identified graphically in a graph.

Details

FindPointsInPoly determines which points in *yWaveName* vs *xWaveName* fall within the polygon defined by *yPolyWaveName* vs *xPolyWaveName*.

xWaveName must have the same number of points as *yWaveName* and *xPolyWaveName* must have the same number of points as *yPolyWaveName*.

FindPointsInPoly creates an output wave named W_inPoly with the same number of points as *xWaveName*. FindPointsInPoly indicates whether the point *yWaveName*[p] vs *xWaveName*[p] falls within the polygon by setting `W_inPoly[p]=1` if it is within the polygon, or `W_inPoly[p]=0` if it is not.

FindPointsInPoly uses integer arithmetic with a precision of about 1 part in 1000. This should be good enough for visually determined (hand-drawn) polygons but might not be sufficient for mathematically generated polygons.

The FindPointsInPoly operation is not multidimensional aware. See **Analysis on Multidimensional Waves** on page II-95 for details.

See Also

GraphWaveDraw, **PolygonOp**

FindRoots

`FindRoots` `[`***`flags`***`]` ***`funcspec, pWave`*** `[,` ***`funcspec, pwave`*** `[, …]]`

`FindRoots /P=`***`PolyCoefsWave`***

The FindRoots operation determines roots or zeros of a specified nonlinear function or system of functions. The function or system of functions must be defined in the form of Igor user procedures.

Using the second form of the command, FindRoots finds all the complex roots of a polynomial with real coefficients. The polynomial coefficients are specified by *PolyCoefsWave*.

Flags for roots of nonlinear functions

/B [= *doBracket*]	Specifies bracketing for roots of a single nonlinear function only. *doBracket*=0: Skips an initial check of the root bracketing values and the possible search for bracketing values. This means that you must provide good bracketing values via the /L and /H flags. See /L and /H flags for details on bracketing of roots. /B alone is the same as /B=0. *doBracket*=1: Uses default root bracketing.
/F=*trustRegion*	Sets the expansion factor of the trust region for the search algorithm when finding roots of systems of functions. Smaller numbers will result in a more stable search, although for some functions larger values will allow the search to zero in on a root more rapidly. Default is 1.0; useful values are usually between 0.1 and 100.
/I=*maxIters*	Sets the maximum number of iterations in searching for a root to *maxIters*. Default is 100.
/L=*lowBracket* /H=*highBracket*	/L and /H are used only when finding roots of a single nonlinear function. *lowBracket* and *highBracket* are X values that bracket a zero crossing of the function. A root is found between the bracketing values. If *lowBracket* and *highBracket* are on the same side of zero, it will try to find a minimum or maximum between *lowBracket* and *highBracket*. If it is found, and it is on the other side of zero, Igor will find two roots. If *lowBracket* and *highBracket* are on the same side of zero, but no suitable extreme point is found between, it will search outward from these values looking for a zero crossing. If it is found, Igor determines one root. If *lowBracket* and *highBracket* are equal, it adds 1.0 to *highBracket* before looking for a zero crossing. The default values for *lowBracket* and *highBracket* are zero. Thus, not using either *lowBracket* or *highBracket* is the same as /L=0/H=1.
/Q	Suppresses printout of results in the history area. Ordinarily, the results of root searches are printed in the history.
/T=*tol*	Sets the acceptable accuracy to *tol*. That is, the reported root should be within ±*tol* of the real root.
/X=*xWave* /X={*x1*, *x2*, …}	Sets the starting point for searching for a root of a system of functions. There must be as many X values as functions. The starting point can be specified with a wave having as many points as there are functions, or you can write out a list of X values in braces. If you are finding roots of a single function, use /L and /H instead. If you specify a wave, this wave is also used to receive the result of the root search.
/Z=*yValue*	Finds other solutions, that is, places where f(x) = *yValue*. FindRoots usually finds zeroes — places where f(x) = 0.

Flag for roots of polynomials

/P=*PolyCoefsWave*	Specifies a wave containing real polynomial coefficients. With this flag, it finds polynomial roots and does not expect to find user function names on the command line. The /P flag causes all other flags to be ignored. Use of this flag is not permitted in a thread-safe function.

Parameters

func specifies the name of a user-defined function.

pwave gives the name of a parameter wave that will be passed to your function as the first parameter. It is not modified. It is intended for your private use to pass adjustable constants to your function.

These parameters occur in pairs. For a one-dimensional problem, use a single *func*, *pwave* pair. An N-dimensional problem requires N pairs unless you use the combined function form (see **Combined Format for Systems of Functions**).

Function Format for 1D Nonlinear Functions

Finding roots of a nonlinear function or system of functions requires that you realize the function in the form of an Igor user function of a certain form. In the FindRoots command you then specify the functions with one or more function names paired with parameter wave names. See **Finding Function Roots** on page III-338 for detailed examples.

The functions must have a particular form. If you are finding the roots of a single 1D function, it should look like this:

```
Function myFunc(w,x)
    Wave w
    Variable x

    return f(x)          // an expression …
End
```

Replace “`f(x)`” with an appropriate expression. The FindRoots command might then look like this:

```
FindRoots /L=0 /H=1 myFunc, cw       // cw is a parameter wave for myFunc
```

Function Format for Systems of Multivariate Functions

If you need to find the roots of a system of multidimensional functions, you can use either of two forms. In one form, you provide N functions with N independent variables. You must have a function for each independent variable. For instance, to find the roots of two 2D functions, the functions must have this form:

```
Function myFunc1(w, x1, x2)
    Wave w
    Variable x1, x2

    return f1(x1, x2)       // an expression …
End

Function myFunc2(w, x1, x2)
    Wave w
    Variable x1, x2

    return f2(x1, x2)       // an expression …
End
```

In this case, the FindRoots command might look like this (where `cw1` and `cw2` are parameter waves that must be made before executing FindRoots):

```
FindRoots /X={0,1} myFunc1, cw1, myFunc2, cw2
```

You can also use a wave to pass in the X values. Make sure you have the right number of points in the X wave — it must have N points for a system of N functions.

```
Function myFunc1(w, xW)
    Wave w, xW

    return f1(xW[0], xW[1])       // an expression …
End

Function myFunc2(w, xW)
    Wave w, xW

    return f2(xW[0], xW[1])       // an expression …
End
```

Combined Format for Systems of Functions

For large systems of equations it may get tedious to write a separate function for each equation, and the FindRoots command line will get very long. Instead, you can write it all in one function that returns N Y values through a Y wave. The X values are passed to the function through a wave with N elements. The parameter wave for such a function must have N columns, one column for each equation. The parameters for equation N are stored in column N-1. FindRoots will complain if any of these waves has other than N rows.

Here is an template for such a function:

```
Function myCombinedFunc(w, xW, yW)
    Wave w, xW, yW

    yW[0] = f1(w[0][...], xW[0], xW[1],..., xW[N-1])
    yW[1] = f2(w[1][...], xW[0], xW[1],..., xW[N-1])
```

```
   …
   yW[N-1] = fN(w[N-1][...], xW[0], xW[1],..., xW[N-1])
End
```

When you use this form, you only have one function and parameter wave specification in the FindRoots command:

```
Make/N=(nrows, nequations) paramWave
fill in paramWave with values
Make/N=(number of equations) guessWave
guessWave = {x0, x1, …, xN}
FindRoots /X=guessWave myCombinedFunc, paramWave
```

FindRoots has no idea how many actual equations you have in the function. If it doesn't match the number of rows in your waves, your results will not be what you expect!

Coefficients for Polynomials

To find the roots of a polynomial, you first create a wave with the correct number of points. For a polynomial of degree N, create a wave with N+1 points. For instance, to find roots of a cubic equation you need a four-point wave.

The first point (row zero) of the wave contains the constant coefficient, the second point contains the coefficient for X, the third for X^2, etc.

There is no hard limit on the maximum degree, but note that there are significant numerical problems associated with computations involving high-degree polynomials. Roundoff error most likely limits reasonably accurate results to polynomials with degree limited to 20 to 30.

Ultimately, if you are willing to accept very limited accuracy, the numerical problems will result in a failure to converge. In limited testing, we found no failures to converge with polynomials up to at least degree 100. At degree 150, we found occasional failures. At degree 200 the failures were frequent, and at degree 500 we found no successes.

Note that you really can't evaluate a polynomial with such high degree, and we have no idea if the computed roots for a degree-100 polynomial have any practical relationship to the actual roots.

While FindRoots is a thread-safe operation, finding polynomial roots is not. Using FindRoots/P=polyWave in a ThreadSafe function results in a compile error.

Results for Nonlinear Functions and Systems of Functions

The FindRoots operation reports success or failure via the V_flag variable. A nonzero value of V_flag indicates the reason for failure.

`V_flag`=0:	Success, but check V_YatRoot, V_YatRoot2 or W_YatRoot to make sure that the convergence point is sufficiently small to indicate a true root. It is very unlikely for the Y values to be exactly zero. "Sufficiently small" depends on the scale of your problem.
`V_flag`=1:	User abort.
`V_flag`=3:	Exceeded maximum allowed iterations
`V_flag`=4:	/T=*tol* was too small. Reported by the root finder for systems of nonlinear functions.
`V_flag`=5:	The search algorithm wasn't making sufficient progress. It may mean that /T=*tol* was set to too low a value, or that the search algorithm has gotten trapped at a false root. Try restarting from a different starting point.
`V_flag`=6:	Unable to bracket a root. Reported when finding roots of single nonlinear functions.
`V_flag`=7:	Fewer roots than expected. Reported by the polynomial root finder. This may indicate that roots were successfully found, but some are doubled. Happens only rarely.
`V_flag`=8:	Decreased degree. Reported by the polynomial root finder. This indicates that one or more of the highest-order coefficients was zero, and a lower degree polynomial was solved.
`V_flag`=9:	Convergence failure or other numerical problem. Reported by the polynomial root finder. This indicates that a numerical problem was detected during the computation. The results are not valid.