The **Execute** operation.

OperationList

OperationList(*matchStr*, *separatorStr*, *optionsStr*)

The OperationList function returns a string containing a list of internal (built-in) or external operation names corresponding to *matchstr*.

Parameters

Only operation names that match *matchStr* string are listed. Use `"*"` to match all names. See **WaveList** for examples.

separatorStr is appended to each operation name as the output string is generated. *separatorStr* is usually ";" for list processing (See **Processing Lists of Waves** on page IV-198 for details).

Use *optionsStr* to further qualify the list of operations. *optionsStr* is a (case-insensitive) string containing one of these values:

`"internal"`	Restricts the list to built-in operations.
`"external"`	Restricts the list to external operations (see **Igor Extensions** on page III-511).

Any other value for *optionsStr* (`"all"` is recommended) will return both internal and external operations.

See Also

The **DisplayProcedure** operation and the **FunctionList**, **MacroList**, **StringFromList**, and **WinList** functions.

Optimize

Optimize [*flags*] *funcSpec*, *pWave*

The Optimize operation determines extrema (minima or maxima) of a specified nonlinear function. The function must be defined in the form of an Igor user function.

Use the first form for univariate functions (one dimensional functions; functions taking just one variable). Use the second form with multivariate functions (functions in more than one dimension; functions of more than one variable).

Optimize uses Brent's method for univariate functions. For multivariate functions you can choose several variations of quasi-Newton methods or simulated annealing.

Flags

/A [= *findMax*]	Finds a maximum (/A=1 or /A) or minimum (/A=0 or no flag).
/D=*nDigits*	Specifies the number of good digits returned (default is 15) by the function being optimized. If you use /X=*xWave* with a single-precision wave, the default is seven. Ignored with simulated annealing (/M={3,0}).
/DSA=*destWave*	Sets a wave to track the current best model only found with simulated annealing (/M={3,0}). *destWave* must have the same number of points as the X vector.
/F=*trustRegion*	Sets the initial trust region when /M={1 or 2, …} with multivariate functions. The value is a scaled step size (see **Multivariate Details**). After the first iteration the trust region is adjusted according to conditions. Ignored with simulated annealing (/M={3,0}).
/H= *highBracket* /L= *lowBracket*	Find the extrema of a univariate function. *lowBracket* and *highBracket* are X values on either side of the extreme point. An extreme point is found between the bracketing values. If *lowBracket* and *highBracket* are equal, Optimize adds 1.0 to *highBracket* before looking for an extreme point. Default values for *lowBracket* and *highBracket* are zero. Thus, if neither *lowBracket* nor *highBracket* is present, this is the same as /L=0/H=1. Ignored with simulated annealing (/M={3,0}).

/I=*maxIters*

Sets the maximum number of iterations in searching for an extreme point to *maxIters*. Default is 100 for *stepMethod* (/M flag) 0-2, 10000 for *stepMethod* = 3 (simulated annealing).

If you use this form of the /I flag with simulated annealing, *maxItersAtT* is set to *maxIters*/2 and *maxAcceptances* is set to *maxIters*/10.

/I={*maxIters*, *maxItersAtT*, *maxAcceptances*}

Specifies the number of iterations for simulated annealing. The maximum number of iterations is set by *maxIters*, *maxItersAtT* sets the maximum number of iterations at a given temperature in the cooling schedule, and *maxAcceptances* sets the total number of accepted changes in the X vector (whether they increase or decrease the function) at a given temperature.

If you use this form of the flag with any *stepMethod* (/M flag) other than 3, *maxItersAtT* and *maxAcceptances* are ignored.

Defaults for *stepMethod* = 3 are {10000, 5000, 500}.

/M={*stepMethod*, *hessMethod*}

Specifies the method used for selecting the next step (*stepMethod*) and the method for calculating the Hessian (matrix of second derivatives) with multivariate functions.

stepMethod	Method
0	Line Search
1	Dogleg
2	More-Hebdon
3	Simulated Annealing

hessMethod	Method
0	secant (BFGS)
1	finite differences

Default values are {0,0}. The *hessMethod* variable is ignored if you select *stepMethod* = 3.

/Q

Suppresses printout of results in the history area. Ordinarily, the results of root searches are printed in the history.

/R={*typX1*, *typX2*, …}

/R=*typXWave*

Specifies the expected size of X values with multivariate functions. These values are used to scale X values. If the X values you expect are very different from one, you will get more accurate results if you can give a reasonable estimate. Optimize will use *typXi* to scale X_i to reduce floating-point truncation error.

You must provide the same number of values in either a wave or a list of values as you provide to the /X flag.

Ignored with simulated annealing (/M={3,0}).

/S=*stepMax*

Limits the largest scaled step size allowed with multivariate functions. Optimize will stop if five consecutive steps exceed *stepMax*.

Ignored with simulated annealing (/M={3,0}).

/SSA=*stepWave*

Name of a 3-column wave having number of rows equal to the length of the X vector only when used with simulated annealing (/M={3,0}). *stepWave* sets information about the step size used to generate new X vectors. The step sizes are in terms of normalized X values. The normalization is such that the X_i ranges from -1 to 1 based on the ranges set by the /XSA flag.

Column zero sets the step size used when creating new trial X vectors. Default is 1.0.

Column one sets the minimum step size. Default is 0.001.

Column two sets the maximum step size. Default is 1.0.

/T=*tol*
Sets the stopping criterion with univariate functions. Optimize will attempt to find a minimum within ± *tol*.

When this form is used with a multivariate function, *gradTol* is set to *tol* and *stepTol* is set to $gradTol^2$.

Ignored with simulated annealing (/M={3,0}).

/T={*gradtol*, *stepTol*}
Sets the stopping criteria for multivariate functions. Iterations stop if a point is found with estimated scaled gradient less than *gradTol*, or if an iteration takes a scaled step shorter than *stepTol*. Default values are $\{8.53618 \times 10^{-6}, 7.28664 \times 10^{-11}\}$. These values are $(6.022 \times 10^{-16})^{1/3}$ and $(6.022 \times 10^{-16})^{2/3}$ as suggested by Dennis and Schnabel. 6.022×10^{-16} is the smallest double precision floating point number that, when added to 1, is different from 1.

Ignored with simulated annealing (/M={3,0}).

/TSA={*InitialTemp*, *CoolingRate*}
Used only with simulated annealing (/M={3,0}).

InitialTemp sets the initial temperature. If *InitialTemp* is set to zero, Optimize calls your function 100 times to estimate the best initial temperature. This is the recommended setting unless your function is very expensive to evaluate (in which case, you may not want to use simulated annealing at all).

CoolingRate sets the factor by which the temperature is decreased.

/X=*xWave*
/X={*x1*, *x2*, …}
Sets the starting point for searching for an extreme point with multivariate functions or with simulated annealing (/M={3,0}). The starting point can be specified with a wave having as many points as the number of independent variables, or you can write out a list of X values in braces. If you are finding extreme points of a univariate function, use /L and /H instead unless you are using the simulated annealing method. If you specify a wave, this wave is also used to receive the result of the extreme point search.

/XSA=*XLimitWave*
Name of a 2-column wave having number of rows equal to the length of the X vector only when used with simulated annealing (/M={3,0}).

Column zero sets the minimum value allowed for each element of the X vector.

Column one sets the maximum value allowed for each element of the X vector.

Default is $\pm X_i * 10$ if X_i is nonzero, or ±1 if X_i is zero. While a default is provided, it is highly recommended that you provide an *XLimitWave*.

/Y=*funcSize*
Specifies expected sizes of function values with multivariate functions. If you expect your function will return values very different from one, you should set *funcSize* to the expected size. Optimize will use this value to scale the function results to reduce floating-point truncation error.

Parameters

func specifies the name of your user-defined function that will be optimized.

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

Function Format

Finding extreme points of a nonlinear function requires that you realize the function as a Igor user function of a certain form. See **Finding Minima and Maxima of Functions** on page III-343 for detailed examples.

Your function must look like this:

```
Function myFunc(w,x1, x2, …)
   Wave w
   Variable x1, x2
```

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

A univariate function would have only one X variable.

A multivariate function can use a wave to pass in the X values:

```
Function myFunc(w,xw)
    Wave w
    Wave xw

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

Replace "f(…)" with an appropriate numerical expression.

Univariate Details

The method used by Optimize to find extreme points of univariate functions requires that the point be bracketed before starting. If you don't use /L and /H to specify the bracketing X values, the defaults are zero and one. Optimize first attempts to find the requested extreme point using the bracketing values (or the default). If that is unsuccessful, it attempts to bracket an extreme point by expanding the bracketing interval. If a suitable interval is found (the search is by no means perfectly reliable), then the search for an extreme point is made again.

Optimize uses Brent's method for univariate functions, which requires no derivatives. This combines a quadratic extrapolation with checking for wild results. In the case of wild results (points beyond the best current bracketing values) the method reverts to a golden section bisection algorithm. For well-behaved functions, the quadratic extrapolation converges superlinearly. The golden section bisection algorithm converges more slowly but features global convergence, that is, if an extremum is there, it will be found.

The stopping criterion is

$$\left| x + \frac{a+b}{2} \right| + \frac{a-b}{2} \le \frac{2}{3} tol.$$

In this expression, *a* and *b* are the current bracketing values, and *x* is the best estimate of the extreme point within the bracketing interval.

The left side of this expression works out to being simply the distance from the current solution to the boundary of the bracketing interval.

Note: Optimizing a univariate function with the simulated annealing method (/M={3,0}) works like a multivariate function, and this section does not apply. See the sections devoted to simulated annealing.

Multivariate Details

With multivariate functions, Optimize scales certain quantities to reduce floating point truncation error. You enter scaling factors using the /R and /Y flags. The /R flag specifies the expected magnitude of X values; Optimize then uses $X_i/typX_i$ in all calculations. Likewise, /Y specifies the expected magnitude of function values.

This scaling can be important for maintaining accuracy if your X's or Y's are very different from one, and especially if your X's have values spanning orders of magnitude.

The Optimize operation uses a quasi-Newton method with derivatives estimated numerically. The function gradient is calculated using finite differences. For estimation of the Hessian (second derivative matrix) you can use either a secant method (*hessMethod* = 0) or finite differences (*hessMethod* = 1). The finite difference method gives a more accurate estimate and may succeed with difficult functions but requires more function evaluations per iteration. The finite difference method's greater accuracy may reduce the total number of iterations required, so the overall number of function evaluations depends on details of the problem being solved. Usually the secant method requires fewer function evaluations and is preferred for functions that are expensive to evaluate.

Once a Newton step is calculated, there are three choices for the method used to find the best next value-line search along the Newton direction (*stepMethod* = 0), double dogleg (*stepMethod* = 1), or More-Hebdon (*stepMethod* = 2). The best method can be found only by experimentation. See Dennis and Schnabel (cited in **References**) for details.

The /F=*trustRegion*, /S=*stepMax* and /T={…, *stepTol*} all refer to scaled step sizes. That is,

$$stepX_i = \frac{|\Delta x_i|}{\max(|x_i|, typX_i)}.$$

The Optimize operation presumes that an extreme point has been found when either the gradient at the latest point is less than *gradTol* or when the last step taken was smaller than *stepTol*. These criteria both refer to scaled quantities:

$$\max_{1\le i\le n}\left\{|g_i|\frac{\max(|x_i|, typX_i)}{\max(|f|, funcSize)}\right\} \le gradTol,$$

or

$$\max_{1\le i\le n}\left\{|g_i|\frac{|\Delta x_i|}{\max(|x_i|, typX_i)}\right\} \le stepTol.$$

Simulated Annealing Introduction

The simulated annealing or Metropolis algorithm optimizes a function using a random search of the X vector space. It does not use derivatives to guide the search, making it a good choice if the function to be optimized is in some way poorly behaved. For instance, it is a good method for functions with discontinuities in the function value or in the derivatives.

Simulated annealing also has a good chance of finding a global minimum or maximum of a function having multiple local minima or maxima.

Because simulated annealing uses a random search method, it may require a large number of function evaluations to find a minimum, and it is not guaranteed that it will stop at an actual minimum. For these reasons, it is best to use one of the other methods unless those methods have failed.

The simulated annealing method generates new trial solutions by adding a random vector to the current X vector. The elements of the random vector are set to $stepsize_i * R_i$, where R_i is a random number in the interval (-1, 1). As the solution progresses, the *stepsize* is gradually decreased.

Bad trials, that is, those that change the function value in the wrong direction are accepted with a probability that depends on the simulated temperature. It is this aspect that allows simulated annealing to find a global minimum.

Function values are generated and accepted or rejected for some number of iterations at a given temperature, then the temperature is reduced. The probability of a bad iteration being accepted decreases with decreasing temperature. A too-fast cooling rate can freeze in a bad solution.

Simulated Annealing Details

It is highly recommended that you use the XSA flag to specify *XLimitWave*. This wave sets bounds on the values of the elements of the X vector during the random search. The defaults may be adequate but are totally *ad hoc*. You are better off to specify bounds that make sense to the problem you are solving.

The values of *XLimitWave* in addition to bounding the search space also scale the X vector during computations of probabilities, temperatures, etc. Consequently, the X limits can affect the performance of the algorithm.

A large number of iterations is required to have a good probability of finding a reasonable solution.

It is recommended that you set the initial temperature to zero so that Optimize can estimate a good initial temperature. If you can't afford the 100 function evaluations required, you probably shouldn't be using simulated annealing.

Optimize uses an exponential cooling schedule in which $T_{i+1} = CoolingRate * T_i$ (see the /TSA flag). *CoolingRate* must be in the range 0 to 1. A fast cooling rate (small value of *CoolingRate*) can cause simulated quenching; that is, a bad solution can be frozen in. Very slow cooling will result in slow convergence.

When simulated annealing is selected, the optimization is treated as multivariate even if your function has only a single X input. That is, the output variables and waves are the ones listed under multivariate functions.

Variables and Waves for Output

The Optimize operation reports success or failure via the V_flag variable. A nonzero value is an error code.

Variables for a univariate function:

V_flag	0:	Search for an extreme point was successful.
	57:	User abort.
	785:	Function returned NaN.
	786:	Unable to find bracketing values for an extreme point.

If you searched for a minimum:

V_minloc	X value at the minimum.
V_min	Function value (Y) at the minimum.

If you searched for a maximum:

V_maxloc	X value at the maximum.
V_max	Function value (Y) at the maximum.

For simulated annealing only:

V_SANumIncreases	Number of "bad" iterations accepted.
V_SANumReductions	Number of iterations resulting in a better solution.

Variables for a multivariate function:

V_flag	0:	Search for an extreme point was successful.
	57:	User abort.
	788:	Iteration limit was exceeded.
	789:	Maximum step size was exceeded in five consecutive iterations.
	790:	The number of points in the typical X size wave specified by /R does not match the number of X values specified by the /X flag
	791:	Gradient nearly zero and no iterations taken. This means the starting point is very nearly a critical point. It could be a solution, or it could be so close to a saddle point or a maximum (when searching for a minimum) that the gradient has no useful information. Try a slightly different starting point.

V_OptTermCode Indicates why Optimize stopped. This may be useful information even if V_flag is zero. Values are:

1:	Gradient tolerance was satisfied.
2:	Step size tolerance was satisfied.
3:	No step was found that was better than the last iteration. This could be because the current step is a solution, or your function may be too nonlinear for Optimize to solve, or your tolerances may be too large (or too small), or finite difference gradients are not sufficiently accurate for this problem.
4:	Iteration limit was exceeded.