Example 2: Integrating a 2D function using function limits

In this example we compute the volume of a sphere of radius 1. To do so we use symmetry and integrate the volume in one octant only. The limits of integration are [0,1] in the x direction and [0,sqrt(1-x^2)] in the y direction. In this case we need to define two user-defined functions: one for the integrand and one for the upper limit of the inner integral:

```
Function myIntegrand2(pWave,inX,inY)
   Wave/Z pWave
   Variable inx,iny
   Variable r2=inX^2+inY^2
   if(r2>=1)
      return 0
   else
      return sqrt(1-r2)
   endif
End

Function y2Func(pWave,inX)
   Wave pWave
   Variable inX
   return sqrt(1-inX^2)
End
```

To perform the integration, execute:

```
Integrate2D outerLowerLimit=0, outerUpperLimit=1, innerLowerLimit=0,
   innerUpperFunc=y2Func, integrand=myIntegrand2
Print/D 4*pi/3 -8*V_Value // Calculation error
```

Note that the integrand function tests that r2 does not exceed 1. This is because the sqrt function would return a NaN if r2>1 which can happen due to floating point rounding errors. Returning a NaN terminates the integration.

Example 3: Integrating a 2D wave using fixed limits

In this example we compute the volume between the surface defined by the wave my2DWave and the plane z=0 with integration limits [0,1] in the x-direction and [0,2] in the y-direction. For simplicity we set the wave's value to be a constant (pi).

```
Make/O/N=(5,9) my2DWave=pi
SetScale/P x 0,0.3,"", my2DWave
SetScale/P y 0,0.4,"", my2DWave
Integrate2D outerLowerLimit=0, outerupperLimit=1, innerlowerLimit=0, innerUpperLimit=2,
   srcWave=my2DWave
Print/D 2*pi-V_Value          // Calculation error
```

See Also

Integrate, **Differentiate**, **Integrate1D**, **area**, **areaXY**

IntegrateODE

IntegrateODE [***flags***] ***derivFunc***, ***cwaveName***, ***ywaveSpec***

The IntegrateODE operation calculates a numerical solution to a set of coupled ordinary differential equations by integrating derivatives. The derivatives are user-specified via a user-defined function, *derivFunc*. The equations must be a set of first-order equations; a single second-order equation must be recast as two first-order equations, a third-order equation to three first order equations, etc. For more details on how to write the function, see **Solving Differential Equations** on page III-322.

IntegrateODE offers two ways to specify the values of the independent variable (commonly called X or t) at which output Y values are recorded. You can specify the X values or you can request a "free-run" mode.

The algorithms used by IntegrateODE calculate results at intervals that vary according to the characteristics of the ODE system and the required accuracy. You can set specific X values where you need output (see the /X flag below) and arrangements will be made to get values at those specific X values. In between those values, IntegrateODE will calculate whatever spacing is needed, but intermediate values will not be output to you.

If you specify free-run mode, IntegrateODE will simply output all steps taken regardless of the spacing of the X values that results.

Parameters

cwaveName	Name of wave containing constant coefficients to be passed to *derivFunc*.
derivFunc	Name of user function that calculates derivatives. For details on the form of the function, see **Solving Differential Equations** on page III-322.
ywaveSpec	Specifies a wave or waves to receive calculated results. The waves also contain initial conditions. The *ywaveSpec* can have either of two forms:

ywaveName: *ywaveName* is a single, multicolumn wave having one column for each equation in your equation set (if you have just one equation, the wave will be a simple 1D wave).

{*ywave0*, *ywave1*, …}: The *ywaveSpec* is a list of 1D waves, one wave for each equation. The ordering is important — it must correspond to the elements of the y wave and dydx wave passed to *derivFunc*.

Unless you use the /R flag to alter the start point, the solution to the equations is calculated at each point in the waves, starting with row 1. You must store the initial conditions in row 0.

Flags

/CVOP={*solver*, *jacobian*, *extendedErrors* [, *maxStep*]}

Selects options that affect how the Adams-Moulton and BDF integration schemes operate. This flag applies only when using /M = 2 or /M = 3. These methods are based on the CVODE package, hence the flag letters “CV”.

The *solver* parameter selects a solver method for each step. The values of *solver* can be:

solver=0:	Select the default for the integration method. That is functional for /M=2 or Newton for /M=3.
solver=1:	Functional solver.
solver=2:	Newton solver.

The *jacobian* parameter selects the method used to approximate the jacobian matrix (matrix of d*f*/dy_i where *f* is the derivative function).

jacobian=0:	Full jacobian matrix.
jacobian=1:	Diagonal approximation.

In both cases, the derivatives are approximated by finite differences.

In our experience, *jacobian* = 1 causes the integration to proceed by much smaller steps. It might decrease overall integration time by reducing the computation required to approximate the jacobian matrix.

If the *extendedErrors* parameter is nonzero, extra error information will be printed to the history area in case of an error during integration using /M=2 or /M=3. This extra information is mostly of the sort that will be meaningful only to WaveMetrics software engineers, but may occasionally help you to solve problems. It is printed out as a bug message (`BUG:...`) regardless of whether it is our bug or yours.

If *maxStep* is present and greater than zero, this option sets the maximum internal step size that the CVODE package is allowed to take. This is particularly useful with /M=3, as the BDF method is capable of taking extremely large steps if the derivatives don't change much. Use of this option may be necessary to make sure that the CVODE package doesn't step right over a brief excursion in, say, a forcing function. If you have something in your derivative function that may be step-like and brief, set *maxStep* to something smaller than the duration of the excursion.

If you want to set *maxStep* only, set the other three options to zero.

/E=*eps*	Adjusts the step size used in the calculations by comparing an estimate of the truncation error against a fraction of a scaled number. The fraction is *eps*. For instance, to achieve error less than one part in a thousand, set *eps* to 0.001. The number itself is set by a combination of the /F flag and possibly the wave specified with the /S flag. See **Solving Differential Equations** on page III-322 for details. If you do not use the /E flag, *eps* is set to 10^{-6}. For details, see **Error Monitoring** on page III-332.
/F=*errMethod*	Adjusts the step size used in the calculations by comparing an estimate of the truncation error against a scaled number. *errMethod* is a bitwise parameter that specifies what to include in that number: bit 0: Add a constant from the error scaling wave set by the /S flag. bit 1: Add the current value of the results. bit 2: Add the current value of the derivatives. bit 3: Multiply by the current step size (/M=0 or /M=1 only). Each bit that you set of bits 0, 1, or 2 adds a term to the number; setting bit 3 multiplies the sum by the current step size to achieve a global error limit. Note that bit 3 has no effect if you use the Adams or BDF integrators (/M=2 or /M=3). See **Setting Bit Parameters** on page IV-12 for further details about bit settings. If you don't include the /F flag, a constant is used. Unless you use the /S flag, that constant is set to 1.0. For details, see **Error Monitoring** on page III-332.
/M=*m*	Specifies the method to use in calculating the solution. *m*=0: Fifth-order Runge-Kutta-Fehlberg (default). *m*=1: Bulirsch-Stoer method using Richardson extrapolation. *m*=2: Adams-Moulton method. *m*=3: BDF (Backwards Differentiation Formula, or Gear method). This method is the preferred method for stiff problems. If you don't specify a method, the default is the Runge-Kutta method (*m*=0). Bulirsch-Stoer (*m*=1) should be faster than Runge-Kutta for problems with smooth solutions, but we find that this is often not the case. Simple experiments indicate that Adams-Moulton (*m*=2 may be fastest for nonstiff problems. BDF (*m*=3) is definitely the preferred one for stiff problems. Runge-Kutta is a robust method that may work on problems that fail with other methods.
/Q [= *quiet*]	*quiet* = 1 or simply /Q sets quiet mode. In quiet mode, no messages are printed in the history, and errors do not cause macro abort. The variable V_flag returns an error code. See **Details** for the meanings of the V_flag error codes.
/R=(*startX*,*endX*)	Specifies an X range of the waves in *ywaveSpec*.
/R=[*startP*,*endP*]	Specifies a point range in *ywaveSpec*. If you specify the range as /R=[*startP*] then the end of the range is taken as the end of the wave. If /R is omitted, the entire wave is evaluated. If you specify only the end point (/R = [,*endP*]) the start point is taken as point 0. You must store initial conditions in *startP*. The first point is *startP*+1.
/S=*errScaleWaveName*	

If you set bit 0 of *errMethod* using the /F flag, or if you don't include the /F flag, a constant is required for scaling the estimated error for each differential equation. By default, the constants are simply set to 1.0.

You provide custom values of the constants via the /S flag and a wave. Make a wave having one element for each derivative, set a reasonable scale factor for the corresponding equation, and set *errScaleWaveName* to the name of that wave.

If you don't use the /S flag, the constants are all set to 1.0.

/STOP = {*stopWave*, *mode*}

Requests that IntegrateODE stop when certain conditions are met on either the solution values (Y values) or the derivatives.

stopWave contains information that IntegrateODE uses to determine when to stop.

mode controls the logical operations applied to the elements of stopWave:

mode=0:	OR mode. If *stopWave* contains more than one condition, any one condition will stop the integration when it is satisfied.
mode=1:	AND mode. If *stopWave* contains more than one condition, all conditions must be satisfied to cause the integration to stop.

See Details, below, for more information.

/U=*u* Update the display every *u* points. By default, it will update the display every 10 points. To disable updates, set *u* to a very large number.

/X=*xvaluespec* Specifies the values of the independent variable (commonly called x or t) at which values are to be calculated (see parameter *ywaveSpec*).

You can provide a wave or *x0* and *deltaX*:

/X = *xWaveName*

Use this form to provide a list of values for the independent variable. They can have arbitrary spacing and may increase or decrease, but should be monotonic.

If you use the /XRUN flag to specify free-run mode, /X = *xWaveName* is required. In this case, the X wave becomes an output wave and any contents are overwritten. See the description of /XRUN for details.

/X = {*x0, deltaX*}

If you use this form, *x0* is the initial value of the independent variable. This is the value at which the initial conditions apply. It will calculate the first result at *x0+deltaX*, and subsequent results with spacing of *deltaX*.

deltaX can be negative.

If you do not use the xValues keyword, it reads independent variable values from the X scaling of the results wave (see *ywaveSpec* parameter).

/XRUN={*dx0*, *Xmax*}

If *dx0* is nonzero, the output is generated in a free-running mode. That is, the output values are generated at whatever values if the independent variable (*x* or *t*) the integration method requires to achieve the requested accuracy. Thus, you will get solution points at variably-spaced X values.

The parameter *dx0* sets the step size for the first integration step. If this is smaller than necessary, the step size will increase rapidly. If it is too large for the requested accuracy, the integration method will decrease the step size as necessary.

If *dx0* is set to zero, free-run mode is not used; this is the same as if the XRUN flag is not used.

When using free-run mode, you must provide an X wave using /X = *xWaveName*. Set the first value of the wave (this is usually point zero, but may not be if you use the /R flag) to the initial value of X.

As the integration proceeds, the X value reached for each output point is written into the X wave. The integration stops when the latest step taken goes beyond *Xmax* or when the output waves are filled.

Details

The various waves you may use with the IntegrateODE operation must meet certain criteria. The wave to receive the results (*ywaveSpec*), and which contains the initial conditions, must have exactly one column for each equation in your system of equations, or you must supply a list of waves containing one wave for each equation. Because IntegrateODE can't determine how many equations there are from your function, it uses the number of columns or the number of waves in the list to determine the number of equations.

If you supply a list of waves for *ywaveSpec*, all the waves must have the same number of rows. If you supply a wave containing values of the independent variable or to receive X values in free-run mode (using /X=*waveName*) the wave must have the same number of rows as the *ywaveSpec* waves.

The wave you provide for error scaling via the /S flag must have one point for each equation. That is, one point for each *ywaveSpec* wave, or one point for each column of a multicolumn *ywaveSpec*.

By default, the display will update after each tenth point is calculated. If you display one of your *ywaveSpec* waves in a graph, you can watch the progress of the integration.

The display update may slow down the calculation considerably. Use the /U flag to change the interval between updates. To disable the updates entirely, set the update interval to a number larger than the length of the waves in *ywaveSpec*.

In free-run mode, it is impossible to predict how many output values you will get. IntegrateODE will stop when either your waves are filled, or when the X value exceeds *Xmax* set in the /XRUN flag. The best strategy is to make the waves quite large; unused rows in the waves will not be touched. To avoid having “funny” traces on a graph, you can prefill your waves with NaN. Make sure that you don't set the initial condition row and initial X value row to Nan!

Stopping IntegrateODE

In some circumstances it is useful to be able to stop the integration early, before the full number of output values has been computed. You can do this two ways: using the /STOP flag to put conditions on the solution, or by returning 1 from your derivative function.

When using /STOP={*stopWave*, *mode*}, *stopWave* must have one column for each equation in your system or, equivalently, a number of columns equal to the order of your system. Each column represents a condition on either the solution value or the derivatives for a given equation in your system.

Row 0 of *stopWave* contains a flag telling what sort of condition to apply to the solution values. If the flag is zero, that value is ignored. If the flag is 1, the integration is stopped if the solution value exceeds the value you put in row 1. If the flag is -1, integration is stopped when the solution value is less than the value in row 1.

Rows 2 and 3 work just like rows 0 and 1, but the conditions are applied to the derivatives rather than to the solution values.

If *stopWave* has two rows, only the solution values are checked. If *stopWave* has four rows, you can specify conditions on both solution values and derivatives.

You can set more than one flag value non-zero. If you do that, then *mode* determines how the multiple conditions are applied. If *mode* is 0, then any one condition can stop integration when it is satisfied. If *mode* is 1, all conditions with a non-zero flag value must be satisfied at the same time. If row 0 and row 2 have nothing but zeroes, then *stopWave* is ignored.

For further discussion, see **Stopping IntegrateODE on a Condition** on page III-335.

Output Variables

The IntegrateODE operation sets a variety of variables to give you information about the integration. These variables are updated at the same time as the display so you can monitor an integration in progress. They are:

V_ODEStepCompleted	Point number of the last result calculated.
V_ODEStepSize	Size of the last step in the calculation.
V_ODETotalSteps	Total number of steps required to arrive at the current result. In free-run mode, this is the same as V_ODEStepCompleted.
V_ODEMinStep	Minimum step size used during the entire calculation.
V_ODEFunctionCalls	The total number of calls made to your derivative function.
V_Flag	Indicates why IntegrateODE stopped.

The values for V_Flag are:

0:	Finished normally.
1:	User aborted the integration.
2:	Integration stopped because the step size became too small. That is, dX was so small that X + dX = X.
3:	IntegrateODE ran out of memory.
4:	In /M=2 or /M=3, the integrator received illegal inputs. Please report this to WaveMetrics (see **Technical Support** on page II-4 for contact details).
5:	In /M=2 or /M=3, the integrator stopped with a failure in the step solver. The method chosen may not be suitable to the problem.
6:	Indicates a bug in IntegrateODE. Please report this to WaveMetrics (see **Technical Support** on page II-4 for contact details).
7:	An error scaling factor was zero (see /S and /F flags).
8:	IntegrateODE stopped because the conditions specified by the /STOP flag were met.
9:	IntegrateODE stopped because the derivative function returned a value requesting the stop.

See Also

Solving Differential Equations on page III-322 gives the form of the derivative function, details on the error specifications and what they mean, along with several examples.

References

The Runge-Kutta (/M=0) and Bulirsh-Stoer (/M=1) methods are based on routines in Press, William H., *et al.*, *Numerical Recipes in C*, 2nd ed., 994 pp., Cambridge University Press, New York, 1992, and are used by permission.

The Adams-Moulton (/M=2) and BDF methods (/M=3) are based on the CVODE component of the SUNDIALS package developed at Lawrence Livermore National Laboratory:

SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans. Math. Softw. 31, 3 (September 2005), 363-396. Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. 2005. DOI=http://dx.doi.org/10.1145/1089014.1089020

Cohen, Scott D., and Alan C. Hindmarsh, *CVODE User Guide*, LLNL Report UCRL-MA-118618, September 1994.