on a value of the solution for the equation corresponding to the column, or stopping on a value of the derivative corresponding to that equation, or both. Each row has different significance:

Row	Meaning
0	Stop flag for solution value 0: Ignore condition on solution for this equation 1: Stop when solution value is greater than the value in row 1 -1: Stop when solution value is less than the value in row 1
1	Value of solution at which to stop
2	Stop flag for derivative 0: Ignore condition on derivative for this equation 1: Stop when derivative value is greater than the value in row 3 -1: Stop when derivative value is less than the value in row 3
3	Value of derivative at which to stop

In the chemical kinetics example above (see **A System of Coupled First-Order Equations** on page III-327) the system has four equations so you need a stop wave with four columns. This wave:

Row	ChemKin_Stop[][0]	ChemKin_Stop[][1]	ChemKin_Stop[][2]	ChemKin_Stop[][3]
0	0	0	-1	1
1	0	0	0.15	0.4
2	0	0	-1	0
3	0	0	0	0

will stop the integration when the concentration of species C (column 2) is less than 0.15, or when the concentration of species D (column 3) is greater than 0.4, or when the derivative of the concentration of species C is less than zero.

When you have multiple stopping criteria, as in this example, you can specify either OR stopping mode or AND stopping mode using the mode parameter of the /STOP flag. If *mode* is 0, OR mode is applied — any of the conditions with a non-zero flag will stop the integration. If *mode* is 1, AND mode is applied — all conditions with a non-zero flag must be satisfied in order for the integration to be stopped.

The second way to stop the integration is by returning a value of 1 from the derivative function. You can apply any condition you like in the function so it is possible to make much more complex stopping conditions this way than using the /STOP flag. However, the derivative function is called for a many intermediate points during a single step, some of which aren't necessarily even on the eventual solution trajectory. That means that you could be applying your stopping criterion to values that are not meaningful to the final solution. That may be particularly true at a time when the internal step size is contracting — the derivative function may be called for points beyond the eventual solution point as the solver tries a step size that doesn't succeed.

Integrating a User Function

You can use the Integrate1D function to numerically integrate a user function. For example, if you want to evaluate the integral

$$I(a,b) = \int_a^b \exp[-x^3 \sin(2\pi/x^2)]dx,$$

you need to start by defining the user function

```
Function userFunc(v)
   Variable v
```

```
   return exp(-v^3*sin(2*pi/v^2))
End
```

The Integrate1D function supports three integration methods: Trapezoidal, Romberg and Gaussian Quadrature.

```
Printf "%.10f\r" Integrate1D(userfunc,0.1,0.5,0)      // default trapezoidal
0.3990547412
Printf "%.10f\r" Integrate1D(userfunc,0.1,0.5,1)      // Romberg
0.3996269165
Printf "%.10f\r" Integrate1D(userfunc,0.1,0.5,2,100)  // Gaussian Q.
0.3990546953
```

For comparison, you can also execute:

```
Make/O/N=1000 tmp
Setscale/I x,.1,.5,"" tmp
tmp=userfunc(x)
Printf "%.10f\r" area(tmp,-inf,inf)
0.3990545084
```

Integrate1D can also handle complex valued functions. For example, if you want to evaluate the integral

$$I(a,b) = \int_a^b \exp\{ix\sin x\}dx,$$

a possible user function could be

```
Function/C cUserFunc(v)
   Variable v
   Variable/C arg=cmplx(0,v*sin(v))
   return exp(arg)
End
```

Note that the user function is declared with a /C flag and that Integrate1D must be assigned to a complex number in order for it to accept a complex user function.

```
Variable/C complexResult=Integrate1D(cUserFunc,0.1,0.2,1)
print complexResult
(0.0999693,0.00232272)
```

Also note that if you just try to print the result without using a complex variable as shown above, you need to use the /C flag with print:

```
Print/C Integrate1D(cUserFunc,0.1,0.2,1)
```

in order to force the function to integrate a complex valued expression.

You can also evaluate multidimensional integrals with the help of Integrate1D. The trick is in recognizing the fact that the user function can itself return a 1D integral of another function which in turn can return a 1D integral of a third function and so on. Here is an example of integrating a 2D function: $f(x,y) = 2x + 3y + xy$.

```
Function do2dIntegration(xmin,xmax,ymin,ymax)
   Variable xmin,xmax,ymin,ymax
   Variable/G globalXmin=xmin
   Variable/G globalXmax=xmax
   Variable/G globalY
   return Integrate1D(userFunction2,ymin,ymax,1)
End

Function userFunction1(inX)
   Variable inX
   NVAR globalY=globalY
   return (3*inX+2*globalY+inX*globalY)
End
```