```
// Normalize the waves
Variable aMin = WaveMin(waveA)
Variable bMin = WaveMin(waveB)
waveA -= aMin
waveB -= bMin
Variable aMax = WaveMax(waveA)
Variable bMax = WaveMax(waveB)
waveA /= aMax
waveB /= bMax
```

Note the use of the temporary variables aMin and bMin. They are needed for two reasons. First, if we wrote `waveA -= WaveMin(waveA)`, then **WaveMin** would be called once for each point in waveA, which would be a waste of time. Worse than that, the minimum value in waveA would change during the course of the waveform assignment statement, giving incorrect results.

There are sometimes faster ways to do waveform arithmetic. For large waves, the **FastOp** and **MatrixOp** operations provide increased speed:

```
waveA -= aMin                       // FastOp does not support wave-variable
FastOp waveA = (1/aMax) * waveA

MatrixOp/O waveA = waveA - aMin
MatrixOp/O waveA = waveA / aMax
```

Example: Converting XY Data to Waveform Data

There are some times when it is desirable to convert XY data to uniformly spaced waveform data. For example, the Fast Fourier Transform requires uniformly spaced data. If you have measured XY data in the time domain, you would need to do this conversion before doing an FFT on it.

We can make some sample XY data as follows:

```
Make/N=1024 xWave, yWave
xWave = 2*PI*x/1024 + gnoise(.001)
yWave = sin(xwave)
```

xWave has values from 0 to 2π with a bit of noise in them. Our data is not uniformly spaced in the x dimension but it is monotonic — always increasing, in this case. If it were not monotonic we could sort the XY pair.

We can create a waveform representing our XY data as follows:

```
Duplicate ywave, wave0
SetScale x 0, 2*PI, wave0
wave0 = interp(x, xwave, ywave)
```

The SetScale command sets the scaling of wave0 so that its X values run from 0 to 2π. Its data values are generated by picking a value off the curve represented by ywave versus xwave at each of these X values using linear interpolation.

See **Converting XY Data to a Waveform** on page III-109 for a discussion of other interpolation techniques.

Example: Concatenating Waves

Concatenating waves can be done much more easily using the **Concatenate** operation (see page V-82). This simple example serves mainly to illustrate a use of wave assignment statements.

Suppose we have three waves of 100 points each: wave1, wave2 and wave3. We want to create a fourth wave, wave4, which is the concatenation of the three original waves. Here is the sequence of commands to do this.

```
Make/N=300 wave4
wave4[0,99] = wave1[p]              // Set first third of wave4
wave4[100,199] = wave2[p-100]       // Set second third of wave4
wave4[200,299] = wave3[p-200]       // Set last third of wave4
```