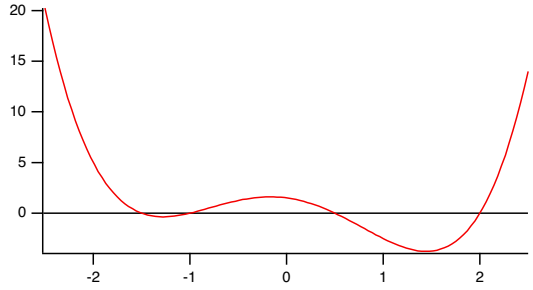


Now use FindRoots to find the roots:

```
FindRoots/P=PolyCoefs   // roots are now in W_polyRoots
Print W_polyRoots
```

This prints:

```
W_polyRoots[0] = {cmplx(0.5,0), cmplx(-1,0), cmplx(-1.5,0), cmplx(2,0)}
```

Note that the imaginary part of the roots are zero, because this polynomial was constructed from real factors. In general, this won't be the case.

The FindRoots operation uses the Jenkins-Traub algorithm for finding roots of polynomials:

Jenkins, M.A., "Algorithm 493, Zeros of a Real Polynomial", *ACM Transactions on Mathematical Software, 1*, 178-189, 1975, used by permission of ACM (1998).

Roots of a 1D Nonlinear Function

Unlike the case with polynomials, there is no general method for finding all the roots of a nonlinear function. Igor searches for a root of the function using Brent's method, and, depending on circumstances will find one or two roots in one shot.

You must write a user-defined function to define the function whose roots you want to find. Igor calls your function with values of X in the process of searching for a root. The format of the function is as follows:

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

   return <an arithmetic expression>
End
```

The wave *w* is a coefficient wave — it specifies constant coefficients that you may need to include in the function. It provides a convenient way to alter the coefficients so that you can find roots of members of a function family without having to edit your function code every time. Igor does not alter the values in w.

As an example we will find roots of the function y = a+b*sinc(c*(x-x0)). Here is a user-defined function to implement this:

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

   return w[0]+w[1]*sinc(w[2]*(x-w[3]))
End
```

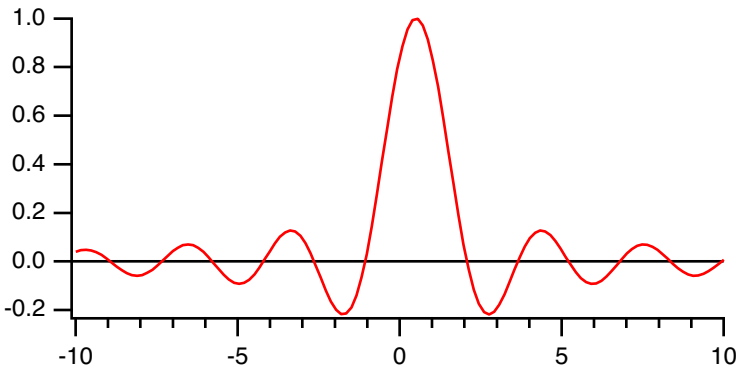
Enter this code into the Procedure window and then close the window.

Make a graph of the function:

```
Make/D/O SincCoefs={0, 1, 2, .5}    // a sinc function offset by 0.5
Make/D/O SincWave                   // a wave with 128 points
```

```
SetScale/I x -10,10,SincWave          // give it an X range of (-2, 2)
SincWave = mySinc(SincCoefs, x)       // fill it with function values

Display SincWave                      // and make a graph of it
ModifyGraph zero(left)=1              // add a zero line to show the roots
ModifyGraph minor(bottom)=1           // add minor ticks to the graph
```



Now we're ready to find roots.

The algorithm for finding roots requires that the roots first be bracketed, that is, you need to know two X values that are on either side of the root (that is, that give Y values of opposite sign). Making a graph of the function as we did here is a good way to figure out bracketing values. For instance, inspection of the graph above shows that there is a root between x=1 and x=3. The FindRoots command line to find a root in this range is

```
FindRoots/L=1/H=3 mySinc, SincCoefs
```

The /L flag sets the lower bracket and the /H flag sets the upper bracket. Igor then finds the root between these values, and prints the results in the history:

```
Possible root found at 2.0708
Y value there is -1.46076e-09
```

Igor reports to you both the root (in this case 2.0708) and the Y value at the root, which should be very close to zero (in this case it is -1.46×10^{-9}). Some pathological functions can fool Igor into thinking it has found a root when it hasn't. The Y value at the supposed root should identify if this has happened.

The bracketing values don't actually have to be at points of opposite sign. If the bracket encloses an extreme point, Igor will find it and then find two roots. Thus, you might use this command:

```
FindRoots/L=0/H=5 mySinc, SincCoefs
```

The Y values at X=0 and X=5 are both positive. Igor finds the minimum point at about X=2.7 and then uses that with the original bracketing values as the starting points for finding two roots. The result, printed in the history:

```
Looking for two roots...

Results for first root:
Possible root found at 2.0708
Y value there is -2.99484e-11

Results for second root:
Possible root found at 3.64159
Y value there is 3.43031e-11
```

Finally, it isn't always necessary to provide bracketing values. If the /L and /H flags are absent, Igor assigns them the values 0 and 1. In this case, there is no root between X=0 and X=1, and there is no extreme point. So Igor searches outward in a series of expanding jumps looking for values that will bracket a root (that is, for X values having Y values of opposite sign). Thus, in this case, the following simple command works: