[11.5] Using bounds and estimates with *nsolve()*

nsolve() is a function that is used to find one approximate real root of an equation. *nsolve()* is better than *solve()* in some cases, because *nsolve()* returns a single number, rather than an expression such as "x=4". This means that the result of *nsolve()* can be used in further calculations without parsing the number from the expression. Further, *nsolve()* is usually faster than *solve().*

The function format is

`nsolve(`*`equation,VarOrGuess`*`)`

where *equation* is the equation to be solved, and *VarOrGuess* specifies the variable for which to solve, and, optionally, a guess of the solution. As described in the manual, the command can be further modfied with solution bounds like this:

`nsolve(`*`equation,VarOrGuess`*`)|`*`bounds`*

where *bounds* is a conditional expression that specifies an interval over which to search for an answer. See the manual for examples.

Like any numeric solver, *nsolve()* can be faster if you supply a guess or bound the solution interval. Note that the guess need not a simple number, but can be a function itself.

With some functions, you *must* supply bounds to get the right solution. Consider $y = x^2$ as a simple example. If y = 4, then there are two solutions, x = 2 and x = -2. Since both are correct, and *nsolve()* cannot 'know' which solution you want, you need to supply the bound like this:

`nsolve(x^2=4,x)|x>0` to find the x = 2 root

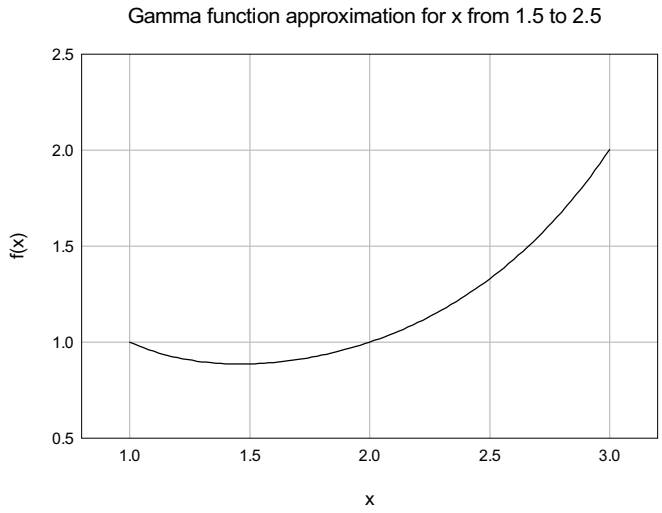
or

`nsolve(x^2=4,x)|x<0` to find the x = -2 root.

To test the performance of the numeric solver, I found an estimating polynomial for the gamma function over the range x = [1.5, 2.5]. The function is

$$f(x) = a + bx + cx^2 + dx^3 + ex^4 + fx^5 + gx^6 + hx^7 + ix^8$$

This graph shows the function approximation.



First, note that we must use bounds to limit the solution range, because the function has a minimum at about x = 1.4. Otherwise, if we try to solve for f(x) = 1, we may get the solution x = 1 or x = 2.

I wrote a simple test program to solve for x for 11 different values of f(x) over the range [1.5, 2]. I also found an estimating function to find a guess for the solver. The estimating function is a 5th order polynomial, with an error of about ±0.03. The *nsolve()* call to use the estimating function looks like this:

```
nsolve(polyeval(fclist,xx)=yy,xx=polyeval(fglist,yy))
```

Here, *fclist{}* is the list of the 8th-order function polynomial coefficients, and *fglist{}* is the list of the 5th-order estimating function coefficients.

I tested three different conditions:

1. No initial guess and bounds of [1.48, 2.52]: mean execution time = 4.3 seconds.
2. Initial guess function and same bounds as 1.: mean execution time = 5.4 seconds
3. Initial guess function, no bounds: mean execution time = 4.2 seconds.

The error for all three conditions was the same. While this is not an exhaustive test, it shows the same results that I have often seen when using the solver:

- *nsolve()* is very good at finding the correct solution, when only bounds are given.
- Supplying *both* bounds and an initial guess function can actually result in slower execution time.
- In terms of execution time, supplying just bounds is as good as supplying an estimating function.