Root Finding

Carver J. Bierson

July 11, 2017

Lets say we have some function $f(x)$ and we want to find a value of x such that $f(x) = 0$. In most cases you would just solve this by hand but there are some forms of $f(x)$ where this can't be done. One such example is $f(x) = (5 - x)e^x - 5$. Here we will use two common methods for finding roots.

1 Bisection

Lets say we know our root is between two points, $a < x_0 < b$. We will assume that $f(a)$ and $f(b)$ have opposite sign (implying the function passes 0 in the middle. We will then use the following algorithm to find the root.

1. Given two points a and b that bracket a root (i.e. $f(a)$ and $f(b)$ have opposite signs), take the halfway point between them $h = (a + b)/2$, and evaluate $f(h)$.
2. If $f(h)$ is within some specified tolerance of 0, then end – we have found the root.
3. If not, then check if f(h) and f(a) have opposite signs. If they do, then there must be a root between $x = a$ and $x = h$, so set $b = h$ and go back to step 1.
4. If $f(h)$ and $f(a)$ have the same sign, then $f(h)$ and $f(b)$ must have opposite signs, because we know that $f(a)$ and $f(b)$ have opposite signs. Thus there must be a root between h and b, so set $a = h$ and go back to step 1.

This algorithm is great because it is guaranteed to find the root on that interval. We just keep stepping closer to the root with each step until we decide we are close enough. The downside of this method is it can be slow on a large interval as we only get a factor of 2 improvement with each step.

```
import numpy as np

# Written by Aldo Batta
# This is the function we wrote in class but with a few changes.
# It uses the bisection method to obtain the root
# of the function "f" we give as input
# "f" has to be defined before calling the function
def myBisection(f,a,b,ytol=1e-10,printall=True):
''' This function uses the bisection method to find the roots of a
    function.
    This program receives as input:
    f -> function we want to solve
    a -> beginning of interval bracketing the root
    a -> end of interval bracketing the root
    ytol -> tolerance for the root (has a default value of 1e-10)
    printall -> boolean indicating if we want to print info.
    '''
    fc = 0.1  i=0
    if(printall):
        print '# iterations | f(c) | c'
        print '--------------------------------'
    while np.abs(fc) > ytol: # This conditions determines how close
     we want # to be to zero. The smaller the number, the
      c = (a + b)/2.
      fa = f(a)
      fb = f(b)
      fc = f(c)
    i=i+1
    if fc * fa < 0:
      b=c
    else:
      a=c
    if(printall):
      print i,'      ',fc,'     ',c
  return c
```

Listing 1: Example Bisection function

The built in scipy bisection function is

```
scipy.optimize.bisect(f, a, b, args=(), xtol=1e-12, rtol
    =4.4408920985006262e-16, maxiter=100, full_output=False, disp=
    True)
```

2 Newton's method

Newtons method tries to outdo bisection by using the slope of our curve to guess where the root is. Let us define $df(x)/dx = f'(x)$. Then if our function were a straight line the root would be at

$$x_1 = x_0 - f(x_0)/f'(x_0) \tag{1}$$

Our function is not generally a straight line but by repeating this process we can usually jump to the correct solution in far fewer steps than bisection. Note that this method requires some knowledge of $f'(x)'$. If we are writing our own Newton's method function it makes sense to request $f(x)$ and $f'(x)$ from the user. The built in scipy function can use either a user provided $f'(x)$ or it can numerically estimate it by evaluating the function at a few nearby points. The Wikipedia page has a nice animation for how this method advances (`https://en.wikipedia.org/wiki/Newton%27s_method`).

Newton's method does have its own drawbacks. The worst of these is that there is no guarantee that it will find the solution. If there is a local minimum that is not near the root it can get stuck and never find its way back.

```
import numpy as np

# Written by Aldo Batta
# This function uses Newton's method to obtain the root of the
# function "f"
# It needs the definition of the derivative of "f"
# To use it with another function we need to define "fprime" first
def NewtonsMethodImproved(f,fprime,xguess, ftol=1.e-10,
                          maxiterations=100,
                          printall=True):
    """
    Find a root using newton's method. We need to specify the
    function,
    f, it's derivative, fprime, and a guess of x value.
    Optional arguments include ytol (default is 1.e-10)
    and maxiterations (default is 100)
    """
  t0 = xguess
  for i in range(maxiterations): # update x1
    t1 = t0 - f(t0)/fprime(t0)
    # reset t0 = t1 : get ready to go round again t0 = t1
        # only print the intermediate steps if we ask for them
    if(printall):
      print i+1, t1 # # of times we repeated the operation, and
        # exit if we're close enough
    if np.abs(f(t0))<ftol:
      break
    # Some error handling on the result
  if i==maxiterations-1:
    print "ERROR: Newton's Method did not converge"
    return None
  else:
    return t1
```

Listing 2: Example Newtons method function

```
scipy.optimize.newton(func, x0, fprime=None, args=(), tol=1.48e-08,
     maxiter=50, fprime2=None)
```

3 Brent's method

Brent's method takes the both worlds from bisection and Newtons method. Like the bracket method, Brent's method takes an interval to search in the input. From there it tries to estimate where the root is with a algorithm similar to Newton's method. However if it gets stuck it jumps to a different part of the interval instead of just failing. This should be your goto method for numerical root finding.

```
scipy.optimize.brentq(f, a, b, args=(), xtol=1e-12, rtol
    =4.4408920985006262e-16, maxiter=100, full_output=False, disp=
    True)
```