

Lesson: Finding Roots: Secant Method and Matlab's fzero

1. Secant Method

The secant method is very similar to Newton's method. The main difference is that it uses a secant line instead of a tangent line. This method involves constructing a secant line of the function and then finding out where the secant line has a root.

See example on handout

A secant line is simply a line that goes through two different points of a curve. Since it must go through two points then we need two beginning points, x_0 and x_1 . So the equation of the secant line will be

$$y - f(x_1) = m(x - x_1)$$

where

$$m = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

The secant method is often called a "quasi-newton" method since it follows the Newton method but instead of calculating the derivative $f'(x_1)$ it uses a finite difference approximation of the derivative i.e.

$$f'(x_1) \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Putting the secant method in the framework of the Newton method, we have that

$$x_n = x_{n-1} + \Delta x$$

$$\text{where } \Delta x = -f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

A general algorithm for the secant method is given below:

Secant Algorithm:

```

Inputs: f, x0, x1, xtol, ftol

iter = 0

while iter < maxit
    iter = iter + 1

    f0 = f(x0)
    f1 = f(x1)

    dx = -f1*(x1-x0) / (f1-f0)
    x2 = x1 + dx
    f2 = f(x2)

    if |x2 - x1| < xtol or |f2| < ftol
        break
    end
    x0 = x1
    x1 = x2
end

r = x2

```

Advantages/Disadvantages

- Converges rapidly, but not as fast as the Newton Method
- If the slope of the secant line is ever zero, it will fail.
- + This method does not require any knowledge of $f'(x)$
- + This method requires fewer (3) function evaluations than the Newton Method (4)

2. Matlab's **fzero** command

Matlab includes a function for root finding called **fzero**. This command combines a few different methods including bisection and the secant method.

fzero is generally called in the following ways:

```

>> r = fzero(funname, x0)
>> r = fzero(funname, x0, [], p1, p2, ...)

```

In the first command, *funname* should be the name of a function m-file which accepts x as an input and computes $f(x)$. In the second command, *funname* should be the name a function m-file which accepts x as an input along with the additional inputs $p1, p2, \dots$ etc. In both commands, x_0 is the starting point of **fzero**. x_0 may be a single point (like in the Newton method) or it may be two points (like in the secant method).

Example: Use **fzero** to find an approximation to a root of $y = e^{-x} + x - 2$. First we let our initial guess $x_0 = 3$.

```
>> x0 = 3; %this is the initial guess
>> funname = 'ch6ex1fun'; %function name that defines our function
>> r = fzero(funname, x0) % notice that ch6ex1fun only defines the function
                           and not the derivative since we are using the secant
                           method and not newton's method

r =
    1.84140566043696
```

Next we let the **fzero** command begin with the interval $[0, 6]$ since we know that there is a root in the interval.

```
>> x0 = [0,6];
>> r = fzero(funname, x0)

r =
    1.84140566043696
```

In both cases, it gave identical results. The limitation in using an interval as an input is shown in the following command:

```
>> r = fzero(funname, [3,6])
??? Error using ==> fzero
    The function values at the interval endpoints must differ in sign.
```

The problem here is that the interval $[3, 6]$ does not contain a root. If an interval is passed to **fzero** it must bracket a root for the function to work.

The advantage of giving an interval to **fzero** instead of a single point is that it will restrict its searching to within that interval. So if a function has several roots, you can single out the one you want by choosing an appropriate interval.