Supplemental material – PHYS 6260
Prof. John Wise
Newman Chapter 6: Solution of Linear and Non-linear Equations

---

6.3: Non-linear Equations

In general, solving non-linear equations is more difficult than their linear counterparts. Solving systems of non-linear equations is even harder, and solving them in multiple dimensions even more so. In this section, we look at several different methods on solving non-linear equations. The best choice of method depends on the function (closed form or data), number of roots, slopes, and so on. No numerical method is fully robust, and it is best to have several tools at your disposal to choose the most useful given a problem.

The relaxation method

Suppose that we have a single non-linear equation in one dimension,

$$x = 2 - e^{-x}. \tag{1}$$

This equation has no analytic solution, so we turn to computational methods. The most basic method is called the *relaxation method* where one makes an initial guess at the solution, plug it into the right-hand side of the equation and receive a new value for x' on the left-hand side. Then we repeat the process, refining the solution in most cases. Using $x = 1$ as an initial solution, we would perform the following.

$$x' = 2 - e^{-1} \simeq 1.632 \tag{2}$$

$$x'' = 2 - e^{-1.632} \simeq 1.804. \tag{3}$$

If we keep repeating (and we're lucky), the value will converge to a fixed value (1.8414).

This method is the easiest to implement, but there exists some problems with it. First, one needs to rearrange the equation into the form, $x = f(x)$, which is not always trivial. Second, for equations with multiple solutions, the relaxation method will converge only to a single solution.

Third, this method might oscillate around the solution, for example consider

$$x = e^{1-x^2} \tag{4}$$

that has a solution $x = 1$ (by inspection). If use the relaxation method with an initial guess of $x = 1/2$, we find that it jumps between 0 and e, never converging. However, we can rearrange this equation into

$$x = \sqrt{1 - \log x} \tag{5}$$

and try again. Now the relaxation method converges to the known solution $x = 1$.

But why does the relaxation method work? And when does it converge or diverge from the solution? We turn back to the trusty Taylor expansion for an equation $x = f(x)$ that has a solution $x = x^*$. The refined value after one iteration is given in terms of the previous value x by

$$x' = f(x) = f(x^*) + (x - x^*)f'(x^*) + \dots \tag{6}$$

By definition x^* is a solution of the original equation, meaning that $x^* = f(x^*)$, which can be written as

$$x' - x^* = (x - x^*)f'(x^*). \tag{7}$$

This tells us that the relaxation method multiples the distance from the true solution $(x' - x^*)$ by its derivative f'. If the derivative is less than 1, then the solution will *converge*. Otherwise, the relaxation method *diverges*. This informs of why rearranging the equation into the functional inverse $x = f^{-1}(x)$ results in convergence, whose derivative is the inverse of $f'(x)$'

Rate of convergence of the relaxation method

Equation (7) tells us that the distance to the solution shrinks by a factor of $|f'(x^*)|$ every iteration, which is an exponential decay. We can estimate how quickly we are converging to the true solution x^*. Let's define the error as ϵ with $x^* = x + \epsilon$. Let ϵ' equal the error on the next iteration so that $x^* = x' + \epsilon'$. We can take Equation (7) and see that the refined error is

$$\epsilon' = \epsilon f'(x^*), \tag{8}$$

resulting in

$$x^* = x + \epsilon = x + \frac{\epsilon'}{f'(x^*)}. \tag{9}$$

From this expression, we can use $x^* = x' + \epsilon'$ and solve for the new error

$$\epsilon' = \frac{x - x'}{1 - 1/f'(x^*)} \simeq \frac{x - x'}{1 - 1/f'(x)} \tag{10}$$

If the functional form $f(x)$ is known, then this procedure is straightforward. In many computational problems, there is no function, and we are dealing with pure data from which we cannot calculate the derivative analytically.

However, we can take the following approach to estimate the error convergence rate. Suppose that we have three successive estimates of x, which we denote x, x', and x''. We can take the last equation to calculate the error on x'', which is

$$\epsilon'' \simeq \frac{x' - x''}{1 - 1/f'(x)}. \tag{11}$$

We can approximate the derivative as

$$f'(x) \simeq \frac{f(x) - f(x')}{x - x'}, \tag{12}$$

and by definition $x' = f(x)$ and $x'' = f(x')$, so that

$$f'(x) \simeq \frac{x' - x''}{x - x'}, \tag{13}$$

which we can use in the error estimate (Equation 11),

$$\epsilon'' = \frac{x' - x''}{1 - (x - x')/(x' - x'')} \tag{14}$$

Example 6.3: Ferromagnetism

In the mean-field theory of ferromagnetism, the magnetization strength M depends on the temperature T according to

$$M = \mu \tanh \frac{JM}{k_{\mathrm{B}}T}, \tag{15}$$

where μ is the magnetic moment, J is a coupling constant, and k_{B} is Boltzmann's constant. We can simplify the expression by defining $m = M/\mu$ and $C = \mu J/k_{\mathrm{B}}$, resulting in

$$m = \tanh \frac{Cm}{T} \tag{16}$$

It is obvious that there exists a solution at $m = 0$ (unmagnetized matter), but does a solution exist at $m \neq 0$ at some temperature T? We can use the *relaxation method* to find the magnetization as a function of T. First, we can calculate the error as

$$\epsilon' = \frac{m - m'}{1 - T\cosh^2(m/T)} \tag{17}$$

to compare against when deciding when to stop the iterative process. Here is the program that finds the solutions and plots them as a function of temperature.

```
#%matplotlib inline
from math import tanh,cosh
import numpy as np
import matplotlib.pyplot as plt

# Constants
Tmax = 2.0
points = 1000
accuracy = 1e-6

# Set up lists for plotting
y = []
temp = np.linspace(0.01,Tmax,points)
```

```
# Temperature loop
for T in temp:
    m1 = 1.0
    error = 1.0

    # Loop until error is small enough
    while error > accuracy:
        m1,m2 = tanh(m1/T),m1
        error = abs((m1-m2)/(1-T*cosh(m2/T)**2))
    y.append(m1)

# Make the graph
plt.plot(temp,y)
plt.ylim(-0.1,1.1)
plt.xlabel("Temperature")
plt.ylabel("Magnetization")
plt.show()
```

Relaxation method for two or more variables

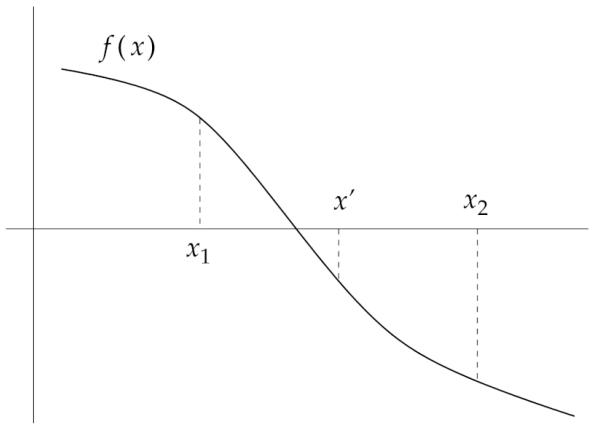
The relaxation method easily extends to two variables and more. Suppose that we have two equations to solve for two variables: (x, y). We first rearrange them into the form: $x = f(x, y)$ and $y = g(x, y)$ and pick an initial guess for both x and y and iterate on the process. More generally, we will be in essence solving N equations with N variables,

$$\begin{aligned} x_1 &= f_1(x_1, \ldots, x_N) \\ &\vdots \\ x_N &= f_N(x_1, \ldots, x_N), \end{aligned}$$

and we can use the same methods we learned earlier in the chapter to solve for each iteration in the relaxation method.

Binary Search

The relaxation method is reasonably fast and easy to implement, but it does have its downsides. The *binary search* (also known as the bisection method) is an alternative and more robust method. First, we only need to rearrange the formula in the form: $f(x) = 0$



that is possible in any case and then search
for roots.

After rearranging the equation, we need to select two bounds, x_1 and x_2, to search for the root. If the function is positive at one point $f(x_1)$ and it is negative at the other point $f(x_2)$, then the root *must* lie between the two points. We then select the midpoint as a new evaluation point $x' = (x_1 + x_2)/2$ and calculate $f(x')$. This procedure is shown in the Figure above. Then we check whether the sign changed on the left or right of the midpoint, essentially halving the search length. We repeat this process until the root is found to some user-defined accuracy. Even in a bad scenario where the initial range is 10^{10} and we desire an accuracy of 10^{-10}, we can converge toward the root in under 100 iterations: $N = \log_2[(x_1 - x_2)/\epsilon] \simeq 66.4$.

However, there are downsides to the binary search. First, we have to pick the initial bounds. If the root doesn't exist between them, the method will never find it. Second if the function is even and the function has the same sign at both limits, then the binary search may not find the root. For example, think about the roots of a parabola with a minima at $y = 0$ or two periods of a sine function. Last, it cannot find multiple roots.

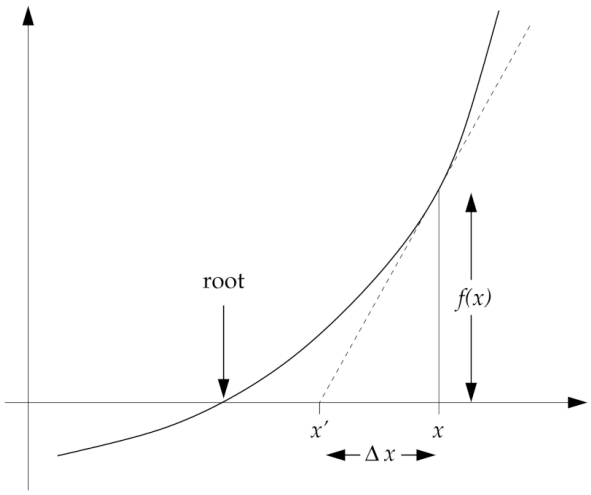
To avoid these problems, here are a couple of guidelines when using the binary search.

1. If you know some facts of the function that can help bracket the root, then this is usually the best place to start. For instance, if you know of specific special values where the function must be positive or negative, perhaps for physical reasons, make use of those points.
2. If you are looking for a root near a particular value of x, start with values closely spaced around that value and check to see if they bracket the root. If not, double their distance from x in either direction and check again. Keep doubling until you find values that bracket the root.

Newton's Method

The relaxation method and binary search are both useful and fast, but they have their unique shortcomings. A third common method to solve non-linear equations is called *Newton's method* that addresses some of the problems in the two previous ones but still has its own problems. Note that it be only be used for problems with known functions and their derivatives.

If we have both functional forms, then Newton's method converges to the root much faster than the previous two methods. First, let us start with a single guess x for the root. We can use the slope $f'(x)$ at that point to extrapolate to $y = 0$ (see the Figure) and make another guess x', which will usually be better than the initial guess. From the Figure, we



can see that the new guess is

$$x' = x - \Delta x = x - \frac{f(x)}{f'(x)}, \tag{18}$$

where we have used the slope $f'(x) = f(x)/\Delta x$. We iterate on this method until Δx becomes smaller than some tolerance.

To determine how many iterations are required to attain some accuracy, we can use our old friend: a Taylor expansion. Let x^* be the true solution to the equation, giving the expansion around x

$$f(x^*) = f(x) + (x^* - x)f'(x) + \tfrac{1}{2}(x^* - x)^2 f''(x) + \ldots \tag{19}$$

Because x^* is a root, then $f(x^*) = 0$, so the left-hand side vanishes. We can divide throughout by $f'(x)$ and solve for x^*, finding

$$x^* = \left[x - \frac{f(x)}{f'(x)}\right] - \tfrac{1}{2}(x^* - x)^2 \frac{f''(x)}{f'(x)} + \ldots \tag{20}$$

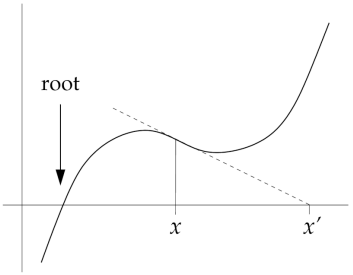
We can see from Equation (18) that the first term is exactly our new estimate, so

$$x^* = x' - \tfrac{1}{2}(x^* - x)^2 \frac{f''(x)}{f'(x)} + \ldots \tag{21}$$

which says that the error quickly decreases with every iteration as

$$e' = \left[\frac{-f''(x)}{2f'(x)}\right] \epsilon^2 \tag{22}$$

Newton's method has two main disadvantages. First as mentioned before, the derivative must be known, which isn't always the case. A more serious problem is that Newton's method can *diverge* when either the slope is very small or the slope "points" in the opposite direction of root, as the cubic function shown to the right.



The secant method

In many cases, we want to find a solution to a problem without a functional form. The *secant method* is similar to Newton's method but the derivative is estimated numerically, using the techniques at the end of Chapter 5. Here we choose two points x_1 and x_2, just like the binary method, but they don't have to bound the solution. We will use the slope of the function between these two points as an estimate of $f'(x)$ in Newton's method,

$$f'(x_2) \simeq \frac{f(x_2) - f(x_1)}{x_2 - x_1} \tag{23}$$

that is used in Newton's method to find the new estimate,

$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)} \tag{24}$$

The secant method converges just as fast as Newton's method but has the same problems.

Newton's method for two or more variables

Unlike the binary search, Newton's method can be easily extended to multiple variables. Like we've seen in the relaxation method, we can write a system of N equations and N unknowns,

$$\begin{aligned} f_1(x_1, \ldots, x_N) &= 0 \\ &\vdots \\ f_N(x_1, \ldots, x_N) &= 0. \end{aligned}$$

Let's denote their solutions as x_i^*, so we can make a Taylor equation around our guess in both summation and vector notation,

$$f_i(x_1^*, \ldots, x_N^*) = f_i(x_1, \ldots, x_N) + \sum_j (x_j^* - x_j)\frac{\partial f_i}{\partial x_j} + \ldots \tag{25}$$

$$\mathbf{f}(\mathbf{x}^*) = \mathbf{f}(\mathbf{x}) + \nabla \mathbf{f} \cdot (\mathbf{x}^* - \mathbf{x}) + \ldots, \tag{26}$$

where $\nabla \mathbf{f}$ is the $N \times N$ matrix with the partial derivatives. Because $\mathbf{f}(\mathbf{x}^*)$ are the roots, the left-hand side of the equations vanish, leaving

$$\nabla \mathbf{f} \cdot \Delta \mathbf{x} = \mathbf{f}(\mathbf{x}). \tag{27}$$

This is just an extension of the one variable case, but now a system of simultaneous equations in the form $\mathbf{A}\mathbf{x} = \mathbf{v}$ must be solved. After solving them, we obtain a new solution $\mathbf{x}' = \mathbf{x} - \Delta \mathbf{x}$. This method can also be used with the secant method but where the derivatives are calculated numerically.