# Lecture 15 - Scipy Root Finding

* AMA / Quiz / Prayer

## I. Scipy's Root Finding Tools

* Just like Numpy has a linear solver, there is another library that implements Newton's method (and more complicated methods) for solving non-linear systems. This library is called Scipy (Scientific Python).

* One imports the library via the command

```
import scipy
```

* Just like Numpy has a submodule "linalg" for linear algebra functions, Scipy has the submodule "optimize" for non-linear equations.

  (why it is called optimize will be more clear after the next lecture.)

* I typically import it using:

```
import scipy.optimize as opt
```

* The function for solving both single non-linear equations and systems of NLEs is called root:

```
x_soln = opt.root(f, x_guess).x
```

root (solution)

function

guess value.

- comments:

  - The function f needs be defined as a function using def. simply assigning an array will not work.

    Good:
    ```
    def f(x):
        return x**2
    ```
    Bad.
    ```
    f = x**2
    ```

  - The function f can be a vector function and a function of multiple variables. This is how one solves a system of equations.

    The function must take a vector argument and return a vector, just like we did last time.

    Good:
    ```
    def f(x):
        x = x[0]
        y = x[1]
        EQ0 = x*y
        EQ1 = y**2
        return np.array([EQ0,EQ1])
    ```
    Bad:
    ```
    def f(x,y):
        return x*y, y**2
    ```

* I have an activity with several detailed examples for you to work through.

$\boxed{\text{Activity}}$ Using scipy.optimize.root.

## II. Solving Engineering NLEs

* Solving engineering NLEs present a number of additional issues to work with in addition to simply using "root."

* This will make more sense if we use an example:

pressure lost in a pipe while fluid flows →

$$\frac{\Delta P}{\rho g} = \frac{-v^2}{2g}\left(\frac{4Lf}{D}\right)$$

$$\frac{1}{\sqrt{f}} = 4\log_{10}\left(Re\sqrt{f}\right) - 0.4$$

← relation between how fast fluid goes and how much energy is lost.

$$Re = \frac{\rho v D}{\mu}$$ ← dimensionless fluid speed

Knowns: $\rho, g, \mu, v, L, \Delta P$ ← know speed & pressure loss

Unknowns: $Re, D, f$ ← solve for pipe diameter

### (1) Units

- Like we talked about @ the beginning of class, we need to be careful about units. Pick a consistent unit system (Eng / SI) or get rid of units by making equations dimensionless.

### (2) Converting to vector notation

- Converting to standard form & vector notation is less transparent. However the process is the same as in "Math-only" problems.

$$f_0 = 0 = \frac{\Delta P}{\rho g} + \frac{v^2}{2g}\left(\frac{4f}{x_0}\right) \qquad x_0 = D/L$$

↑ This has no units now!

$$f_1 = 0 = \frac{1}{\sqrt{x_1}} - 4\log_{10}(x_2\sqrt{x_1}) + 0.4 \qquad x_1 = f$$

$$x_2 = Re$$

$$f_2 = 0 = x_2 - \frac{\rho v L}{\mu} x_0$$

$$\underline{f}(\underline{x}) = \begin{bmatrix} \frac{\Delta P}{\rho g} + \frac{v^2}{2g}\left(\frac{4f}{x_0}\right) \\ \frac{1}{\sqrt{x_1}} - 4\log_{10}(x_2\sqrt{x_1}) + 0.4 \\ x_2 - \frac{\rho v L}{\mu} x_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

## (3) Getting root to converge

- To get it to converge we will need a good guess. Usually we get this from physical inuition. For example $f$ is usually about $10^{-3}$. Or we can use $f = \frac{16}{Re}$ as a guess (laminar flow in fluids). D is usually something in inches.

- Some equations have a hard time converging anyway! "Root" uses a numerical estimate of the Jacobian. If you provide an analytical Jacobian it can help a lot! You do this just like you did for Newton's method.

Jacobian function

```
x = opt.root(f, x_guess, jac=J).x

    def J(x):
        ⋮
        return np.array([[∂f0/∂x0 ... ∂f0/∂xn-1],
                              ⋮
                         [∂fn-1/∂x0 ... ∂fn-1/∂xn-1]])
```