Lecture 14 - NLEs with Scipy.

* AMA / Quiz / Prayer.

I. Systems of Nonlinear Equations

* Very often we will not only have a single nonlinear equation, but we will instead have a system of coupled nonlinear equations. We need to update what we learned last time for multiple equations and multiple unknowns.

Example: 2 Equations / 2 unknowns

(1) $z = e^{-y}$

(2) $y = 3z - z^2$

- Lets rename $y \rightarrow x_0$, $z \rightarrow x_1$

$$x_1 = e^{-x_0}$$

$$x_0 = 3x_1 - x_1^2$$

- Put in standard (i.e. residual) form

$$x_1 - e^{-x_0} = 0$$

$$x_0 - 3x_1 + x_2^2 = 0$$

$f_0(x_0, x_1) = 0$, $f(x_0, x_1) = -e^{x_0} + x_1$

$f_1(x_0, x_1) = 0$, $f_1(x_0, x_1) = x_0 - 3x_1 + x_2^2$

- we can rewrite this in vector form

$$\underline{f}(\underline{x}) = \underline{0} \qquad \underline{f} = \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} \quad \underline{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \quad \underline{0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

* with more equations and unknowns, we can write this in general as:

$$f_0(x_0, x_1, \ldots, x_{n-1}) = 0$$
$$f_1(x_0, x_1, \ldots, x_{n-1}) = 0$$
$$\vdots$$
$$f_{n-1}(x_0, x_1, \ldots, x_{n-1}) = 0$$

or

$$\underline{f}(\underline{x}) = \underline{0}$$ ← standard / residual form for systems of NLEs.

$$\underline{f} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{bmatrix} \quad \underline{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \quad \underline{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

* Important: we will need to write our functions as vector equations in Python too.

x is an array

```
def f(x):
    f0 = - np.exp(x[0]) + x[1]
    f1 = x[0] - 3*x[1] + x[1]**2
    return np.array([f0, f1])
```

f is an array

$\boxed{\text{Activity}}$ write the following in vector notation and write pseudo-code defining a vector function for the system:

$$\cos(a^2 + b^2) = 2$$

$$\ln(3b) = -a^2$$

A solution:

$$\left.\begin{aligned} f_0 &= \cos(x_0^2 + x_1^2) - 2 \\ f_1 &= \ln(3x_1) + x_0^2 \end{aligned}\right\} \text{vector format}$$

pseudo-code:

```
def f(x):
    f0 = np.cos(x[0]**2
              + x[1]**2) - 2
    f1 = np.log(3*x[1]) + x[0]**2
    return np.array([f0, f1])
```

II. SciPy Root Finding.

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

* one imports this library (module) via the command

  import scipy

* Scipy actually contains much more than non-linear solvers. It has many useful sub-modules. One is for linear algebra, "linalg". The relevant one for today is "optimize".

* Scipy.optimize contains root finding and optimization methods. Why these are related will be clear after the next lecture. I typically import this module using:

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

* The function for solving <u>both</u> single NLEs and systems of NLEs is called "root":

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

module name — function — The function returns a "dictionary" with lots of info. ".x" gives you the answer only.

comments about root

- f needs to be a <u>function</u>, defined using <u>def</u>. Simply assigning an array won't work.

Good:

```
def f(x):
    return x**2
```

Bad:

```
f = x**2
```

- when solving a <u>system</u>, you must define a <u>vector function</u> with a vector input.

Good:

```
def f(x):
    f0 = x[0]*x[1]
    f1 = x[1]**2
    return np.array([f0, f1])
```

Bad: def f(x,y): ← not an array input

```
def f(x,y):
    return x * y, y ** 2
```

↑ returns a tuple, not a numpy array.

Activity

* Several Detailed examples using Scipy.optimize.root