Lecture 14 - NLEs with Scipy

I. Systems of NLEs

II. Scipy Root Finding

I. Systems of NLEs

* system: multiple EQs, multiple unk. → coupled

Example:

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

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

* standard form (residual form)
  - single NLE: $f(x) = 0$

$$f_0 = z - e^{-y} = 0$$

$$f_1 = y - 3z + z^2 = 0$$

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

$$\underline{x} = \begin{bmatrix} x_0 = y \\ x_1 = z \end{bmatrix}$$

$$f_0 = x_1 - e^{-x_0}$$

$$f_1 = x_0 - 3x_1 + x_1^2$$

vector notation

$$\underline{f}(\underline{x}) = \underline{0} \qquad \underline{0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

* For the general case: ← n EQ's, n unk.

$$f_0(x_0, x_1, \dots, x_{n-1}) = 0$$

$$f_1(x_0, x_1, \dots, x_{n-1}) = 0$$

$$\vdots$$

$$f_{n-1}(x_0, x_1, \dots, x_{n-1}) = 0$$

$$\underline{f}(\underline{x}) = \underline{0}$$

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

* Important: our functions in Python need to be vector EQ's too!
(np.array)

Pseudocode

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

array

f(x0, x1) ← ind. values

f is an array too

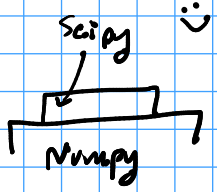
return f0, f1 ← not a tuple

Activity write the following in vector notation & write P.C. to define a vector function

$\cos(a^2+b^2) = 2$ } soln in online notes

$\ln(36) = -a^2$
(36 ↑ b)

II. Scipy Root Finding



* Numpy → linear solver "linalg"
* scipy → new module (sister/friend of Numpy)
  ↑ Scientific ← Python

* import:

```
import scipy
```

submodules
linalg – linear algebra (eigenvectors!)
optimize – optimization root finding

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

* Function for solving both single & systems of NLEs is called "root"

* Syntax:

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

- f: function to solve
- opt: module name
- root: root finding function
- x_guess: initial guess
- .x: opt.root returns a dict. .x is solution part of dict.

* comments:

- f needs to be a function (programming sense)

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

Bad: ✗
```
f = x ** 2
```

- when solving a system of NLEs, you must define a vector function

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

Bad: ✗
```
def f(x,y):        # x not an array
    return x*y, y**2   # not an array
```

Activity: Examples of solving NLEs using opt.root.