Lecture 14 — Systems of NLEs

* AMA / Quiz / Prayer

I. Methods for Systems of NLEs

* Solving a multi-variable system of NLEs is very similar to solving a single NLE. We will still used fixed point methods, but we need to adapt for multiple equations & unknowns.

* The problem

  - Now, we have multiple equations & unknowns

$$\left.\begin{aligned} f_0(x_0, x_1, \dots, x_{n-1}) &= 0 \\ f_1(x_0, x_1, \dots, x_{n-1}) &= 0 \\ &\vdots \\ f_{n-1}(x_0, x_1, \dots, x_{n-1}) &= 0 \end{aligned}\right\} \text{standard / residual form}$$

  - we can write this more compactly in vector notation:

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

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

* Fixed Point Methods

  - Again, just like in single NLEs, the form is:

$$\underline{x}^{(k+1)} = \underline{g}(\underline{x}^{(k)}) \Rightarrow \begin{bmatrix} x_0^{(k+1)} \\ x_1^{(k+1)} \\ \vdots \\ x_{n-1}^{(k+1)} \end{bmatrix} = \begin{bmatrix} g_0(x_0^{(k)}, x_1^{(k)}, \ldots, x_{n-1}^{(k)}) \\ g_1(x_0^{(k)}, x_1^{(k)}, \ldots, x_{n-1}^{(k)}) \\ \vdots \\ g_{n-1}(x_0^{(k)}, x_1^{(k)}, \ldots, x_{n-1}^{(k)}) \end{bmatrix}$$

A. Picard's Method

$$\underline{g}(\underline{x}) = \underline{x} + \underline{f}(\underline{x}), \quad \text{so}$$

$$\boxed{\underline{x}^{(k+1)} = \underline{x}^{(k)} + f(\underline{x}^{(k)})}$$

- Just like a single NLE, Picard's method is easy, but doesn't work very well.

B. Newton's method

- Remember for a single equation Newton's method was:

$$g(x) = x - \frac{f(x)}{f'(x)}$$

- Analogously, for multiple equations

$$\underline{g}(\underline{x}) = \underline{x} - \underline{\underline{J}}(\underline{x})^{-1} \cdot \underline{f}(\underline{x})$$

inverse of the matrix of all of the derivatives. (Jacobian)

$$\underline{\underline{J}}(\underline{x}) = \begin{bmatrix} \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial x_1} & \cdots & \frac{\partial f_0}{\partial x_{n-1}} \\ \frac{\partial f_1}{\partial x_0} & \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_{n-1}} \\ \vdots & & & \\ \frac{\partial f_{n-1}}{\partial x_0} & \frac{\partial f_{n-1}}{\partial x_1} & \cdots & \frac{\partial f_{n-1}}{\partial x_{n-1}} \end{bmatrix} = \left[\frac{\partial f_i}{\partial x_j} \right]$$

rows

columns

(*) See supplemental notes on derivation if interested.

- If you haven't yet met the partial derivative, it is just like the derivative you already know.
  - ↳ (*) See supplemental notes on partial derivatives for more details.

- Solving for $\underline{x}^{(k+1)}$?

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} - \underline{\underline{J}}^{-1}(\underline{x}^{(k)}) \cdot \underline{f}(\underline{x}^{(k)})$$

$$\underbrace{\underline{x}^{(k+1)} - \underline{x}^{(k)}}_{\underline{\delta}} = \underline{\underline{J}}^{-1}(\underline{x}^{(k)}) \cdot (-\underline{f}(\underline{x}^{(k)}))$$

multiply both sides by $\underline{\underline{J}}$

$$\boxed{\begin{aligned} \underline{\underline{J}}(\underline{x}^{(k)}) \cdot \underline{\delta} &= -\underline{f}(\underline{x}^{(k)}) \\ x^{(k+1)} &= \underline{x}^{(k)} + \underline{\delta} \end{aligned}}$$

- evaluate f
- evaluate J
- solve linear system : $\underline{\underline{J}} \cdot \underline{\delta} = -\underline{f}$
- update x

II. Example : 2 Eq., 2 unknowns

* Write Picard & Newton's method for :

$$z = e^{-y}$$
$$y = 3z - z^2$$

(1) Convert to standard / residual form :

$$x_0 = y \qquad x_1 = z \quad \Rightarrow \quad x_1 = e^{-x_0}$$

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

$$\Rightarrow \boxed{\begin{aligned} -x_1 + e^{-x_0} &= 0 \\ x_0 - 3x_1 + x_1^2 &= 0 \end{aligned}} \quad \begin{aligned} f_0(x_0, x_1) &= 0 \\ f_1(x_0, x_1) &= 0 \end{aligned}$$

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

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

(2) write in the form $\underline{x} = \underline{g}(\underline{x})$

(Picard's Method)

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} + \underline{f}(\underline{x}^{(k)})$$

$$\begin{bmatrix} x_0^{(k+1)} \\ x_1^{(k+1)} \end{bmatrix} = \begin{bmatrix} x_0^{(k)} \\ x_1^{(k)} \end{bmatrix} + \begin{bmatrix} -x_1^{(k)} + e^{-x_0^{(k)}} \\ x_0^{(k)} - 3x_1^{(k)} + (x_1^{(k)})^2 \end{bmatrix}$$

$$\boxed{\begin{aligned} x_0^{(k+1)} &= x_0^{(k)} - x_1^{(k)} + e^{-x_0^{(k)}} \\ x_1^{(k+1)} &= x_0^{(k)} - 2x_1^{(k)} + (x_1^{(k)})^2 \end{aligned}}$$

(Newton's Method)

$$\underline{\underline{J}}(\underline{x}^{(k)}) \cdot \underline{\delta} = -\underline{f}(\underline{x}^{(k)})$$

$\uparrow$

need derivatives (4 of them!)

$$\underline{\underline{J}} = \begin{bmatrix} \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial x_1} \\ \frac{\partial f_1}{\partial x_0} & \frac{\partial f_1}{\partial x_1} \end{bmatrix}$$

$$\frac{\partial f_0}{\partial x_0} = \frac{\partial}{\partial x_0}\left(-x_1 + e^{-x_0}\right) = -e^{-x_0}$$

$$\frac{\partial f_0}{\partial x_1} = \frac{\partial}{\partial x_1}\left(-x_1 + e^{-x_0}\right) = -1$$

$$\frac{\partial f_1}{\partial x_0} = \frac{\partial}{\partial x_0}\left(x_0 - 3x_1 + x_1^2\right) = 1$$

$$\frac{\partial f_1}{\partial x_1} = \frac{\partial}{\partial x_1}\left(x_0 - 3x_1 + x_1^2\right) = 2x_1 - 3$$

$$\begin{bmatrix} -e^{-x_0^{(k)}} & -1 \\ 1 & 2x_1^{(k)} - 3 \end{bmatrix} \cdot \begin{bmatrix} \delta_0 \\ \delta_1 \end{bmatrix} = - \begin{bmatrix} -x_1^{(k)} + e^{-x_0^{(k)}} \\ x_0^{(k)} - 3x_1^{(k)} + (x_1^{(k)})^2 \end{bmatrix}$$

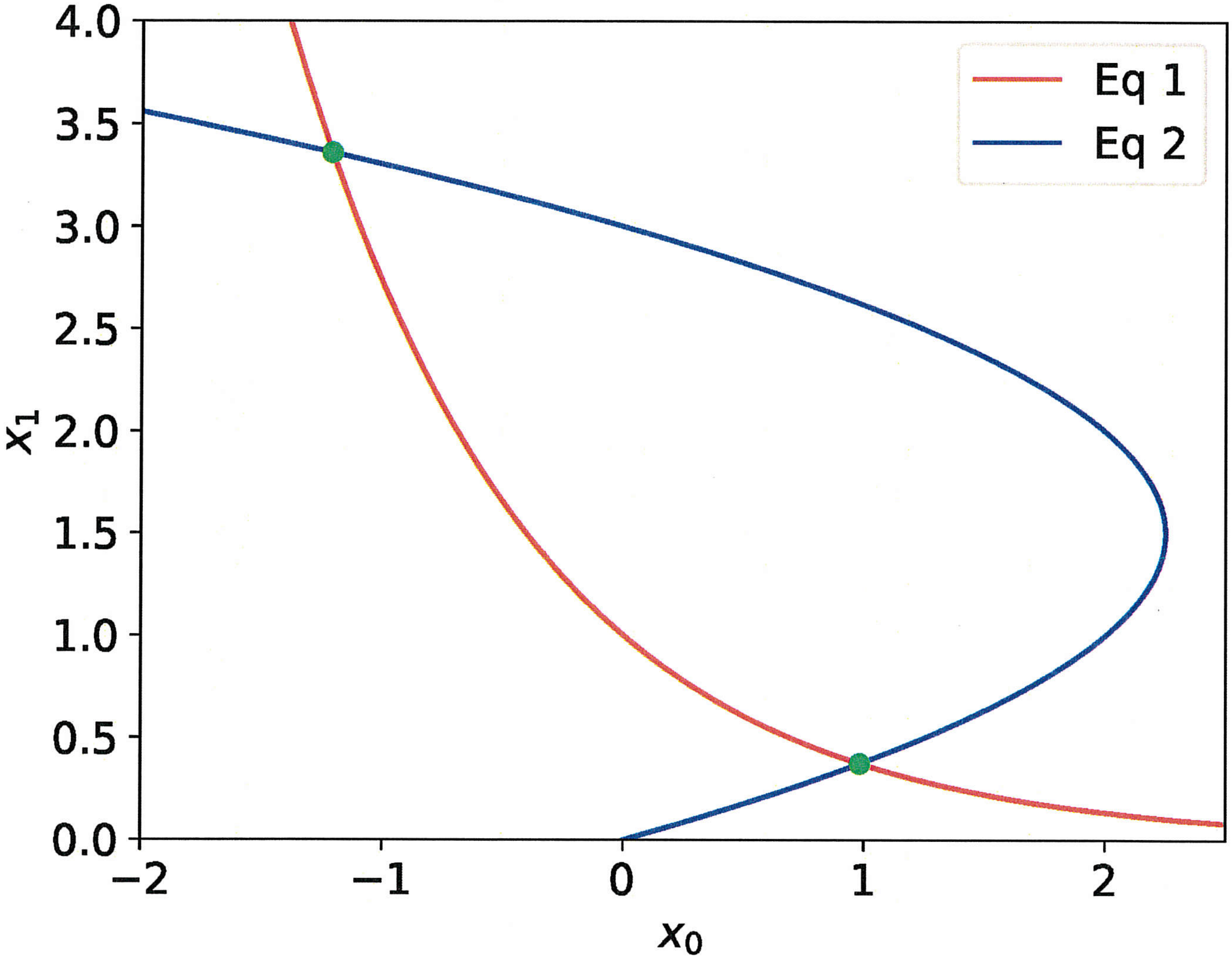
$$\begin{bmatrix} x_0^{(k+1)} \\ x_1^{(k+1)} \end{bmatrix} = \begin{bmatrix} x_0^{(k)} + \delta_0 \\ x_1^{(k+1)} + \delta_1 \end{bmatrix}$$

Activity

* Go through the code for Picard & Newton's method that I wrote.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np

def f(x):
    x0 = x[0]
    x1 = x[1]
    f0 = np.exp(-x0) - x1
    f1 = x0 + x1**2 - 3*x1
    return np.array([f0, f1])

def J(x):
    x0 = x[0]
    x1 = x[1]
    df0_dx0 = -np.exp(-x0)
    df0_dx1 = -1
    df1_dx0 = 1
    df1_dx1 = 2*x1 - 3
    return np.array([[df0_dx0, df0_dx1],
                     [df1_dx0, df1_dx1]])

tol = 1e-12
k_max = 50

k=0
x = np.array([0., 0.]) # guess1
#x = np.array([-1., 3.]) # guess2
res = np.linalg.norm(f(x))

print('%4s%12s%12s%16s'%('k', 'x0', 'x1', 'residual'))
print('-'*44)

# Picard's method
while (res > tol):

    print('%4d%12f%12f%16e'%(k, x[0], x[1], res))

    delta = np.linalg.solve(J(x), -f(x))
    x = x + delta
    k+=1
    res = np.linalg.norm(f(x))

    if (k >= k_max):
        print("Newton's method did not converge")
        break

print('%4d%12f%12f%16e'%(k, x[0], x[1], res))
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np

def f(x):
    x0 = x[0]
    x1 = x[1]
    f0 = np.exp(-x0) - x1
    f1 = x0 + x1**2 - 3*x1
    # switch f0 & f1 and it will not converge
    return np.array([f0, f1])

tol = 1e-12
k_max = 50

k=0
x = np.array([0., 0.]) # guess

res = np.linalg.norm(f(x))

print('%4s%12s%12s%16s'%('k', 'x0', 'x1', 'residual'))
print('-'*44)

# Picard's method
while (res > tol):

    print('%4d%12f%12f%16e'%(k, x[0], x[1], res))

    x = x + f(x)
    k+=1
    res = np.linalg.norm(f(x))

    if (k >= k_max):
        print("Picard's method did not converge")
        break

print('%4d%12f%12f%16e'%(k, x[0], x[1], res))
```



Eq 1
Eq 2
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2
−1
0
1
2
x_1
x_0

Lecture 14 - Supplemental Notes

I. Partial Derivatives

* A partial derivative is a derivative with respect to one variable of a multi-variate function

$f(x_0, x_1, x_2)$

function of 3 variables : x_0, x_1, x_2

$\frac{\partial f}{\partial x_0}$ ← partial derivative with respect to x_0, with x_1 & x_2 held constant.

* Formal definition

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_0, x_1, \ldots x_i + h, \ldots, x_{n-1}) - f(x_0, x_1, \ldots, x_i, \ldots, x_n)}{h}$$

compare to total derivative

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

* notation :

- we use a curly 'D' : ∂ for a partial derivative.

  Note that this is not a Greek delta : δ

$\frac{\partial f}{\partial x_0}$ — partial derivative

$\frac{df}{dx}$ — single variable / total derivative

$\frac{\delta f}{\delta x}$ — variational derivative (you will probably never use this.)

* You evaluate partial derivatives exactly the same way as "regular" derivatives except you treat all of the other x's as constants.

Examples:

y is const. so $\frac{\partial (y^2)}{\partial x} = 0$

(polynomial) $\frac{\partial}{\partial x}(x^2+y^2) = 2x$

$\frac{\partial}{\partial x}(x^2y^2) = 2xy^2$ ← y is const so it stays.

(trig) $\frac{\partial}{\partial y}(\sin(xy)) = x\cos(xy)$

comes from chain rule

$\frac{\partial}{\partial y}(\cos(x^2+y^2)) = -\sin(x^2+y^2)\cdot 2y$

(logarithm/exponential) $\frac{\partial}{\partial x}(x \ln y) = \ln y$

$\frac{\partial}{\partial y}(x\ln y) = \frac{x}{y}$

$\frac{\partial}{\partial x}(ye^x) = ye^x$

$\frac{\partial}{\partial y}(ye^x) = e^x$

* See your multivariable calculus text for more details

II. Derivation of multivariate Newton's method.

$\underline{f}(\underline{x}) = \underline{0}$ is our standard form

* Expand $\underline{f}(\underline{x})$ in a Taylor Series about point $\underline{x}^{(k)}$ "Higher Order Terms"

$$f_0(\underline{x}) = f_0(\underline{x}^k) + \left.\frac{\partial f_0}{\partial x_0}\right|_{\underline{x}^{(k)}} (x_0 - x_0^{(k)}) + \dots + \left.\frac{\partial f_0}{\partial x_{n-1}}\right|_{\underline{x}^{(k)}} (x_{n-1} - x_{n-1}^{(k)}) + \text{H.O.T}$$

$$\vdots$$

$$f_{n-1}(\underline{x}) = f_{n-1}(\underline{x}^k) + \underbrace{\left.\frac{\partial f_{n-1}}{\partial x_0}\right|_{\underline{x}^k} (x_0 - x_0^{(k)}) + \dots + \left.\frac{\partial f_{n-1}}{\partial x_{n-1}}\right|_{\underline{x}^k} (x_{n-1} - x_{n-1}^{(k)})}_{\text{all the linear terms}} + \text{H.O.T}$$

* In vector notation

$$\underline{f}(\underline{x}) = \underline{f}(\underline{x}^{(k)}) + \underbrace{\underline{\underline{J}}(\underline{x}^{(k)})\cdot(\underline{x} - \underline{x}^{(k)})}_{\text{linear terms.}} + \text{H.O.T.}$$

$$\underline{\underline{J}} = \begin{bmatrix} \frac{\partial f_0}{\partial x_0} & \cdots & \frac{\partial f_0}{\partial x_{n-1}} \\ \vdots & & \vdots \\ \frac{\partial f_{n-1}}{\partial x_0} & \cdots & \frac{\partial f_{n-1}}{\partial x_{n-1}} \end{bmatrix} \quad \text{Jacobian matrix}$$

$$\underline{\underline{J}} = \left[\frac{\partial f_i}{\partial x_j} \right]$$

* Now, let $\underline{x} = \underline{x}^*$, a root and let $\underline{x}^k$ be close to the root so that:

$\underline{f}(\underline{x}^*) = \underline{0}$, and

$(\underline{x}^* - \underline{x}^k)$ is small so we can neglect H.O.T of size $(\underline{x}^* - \underline{x}^k)\cdot(\underline{x}^* - \underline{x}^k)$

* Using this idea we get:

$$0 = \underline{f}(\underline{x}^{k}) + \underline{J}(\underline{x}^{k}) \cdot (\underline{x}^{*} - x^{(k)})$$

let this be $x^{(k+1)}$, our next guess

$$J(x^{(k)}) \cdot (\underbrace{x^{(k+1)} - x^{(k)}}_{\underline{\delta}}) = -f(\underline{x}^{(k)})$$

$$\boxed{\begin{aligned} J(x^{(k)}) \cdot \underline{\delta} &= -f(\underline{x}^{(k)}) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \underline{\delta} \end{aligned}}$$