Computación científica LF-214

Profesor: Julio C. Marín

Departamento de Meteorología, Universidad de Valparaíso

Primer Semestre

Temas de la clase

- Ecuaciones no-lineales.
- Método de relajación
- Solución usando función fsolve de Python

Resolución numérica de ecuaciones no lineales

- Muchas ecuaciones en física son no-lineales
- Resolver una ecuación no-lineal es mucho más difícil que resolver una ecuación lineal
- La complejidad aumenta si es necesario resolver un sistema de ecuaciones no-lineales
- Se han desarrollado varios métodos para resolver sistema de ecuaciones no lineales

- Soluciones o raíces de ecuación pueden ser reales o complejas.
- Raíces complejas casi no se calculan por no tener significado físico.

Ejemplo 1

$$\sin x - x = 0$$

tiene 1 solución: $x = 0$

- Soluciones o raíces de ecuación pueden ser reales o complejas.
- Raíces complejas casi no se calculan por no tener significado físico.

Ejemplo 1

$$\sin x - x = 0$$

tiene 1 solución: $x = 0$

Ejemplo 2

$$\tan x - x = 0$$

Tiene un número infinito de soluciones:
$(x = 0, \pm 4.493, \pm 7.725, \ldots)$

Método de relajación

- Suponiendo queremos resolver una ecuación no-lineal:

$$x = 2 - e^{-x}$$

- No existe método analítico para resolver ecuación, necesitamos métodos numéricos
- Método de iteración es método sencillo de obtener solución
- Suponiendo valor inicial $x = 1$

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

Método de relajación

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

- Repitiendo el proceso:

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

- Con suerte valor convergerá a valor fijo de ecuación.

Método de relajación

- Código sencillo para hacer este cálculo:

Método de relajación

- Código sencillo para hacer este cálculo:

```
from math import exp
x = 1.0
for ii in range(10):
    x = 2 - exp(-x)
    print (x)
```

- Resultado converge a valor 1.84140564533

- Código sencillo para hacer este cálculo:

```
from math import exp
x = 1.0
for ii in range(10):
    x = 2 - exp(-x)
    print (x)
```

- Resultado converge a valor 1.84140564533

Cómo modificar código para que calcule solución y automáticamente pare cuando se cumpla cierta condición de exactitud?

Método de relajación

```
import numpy as np
ii = 1
x1 = 2.0; x0 = 0.1
Er = abs(x1 - x0)
while (Er > 10**-5):
    x0 = x1
    x1 = 2 - np.exp(-x0)
    Er = abs(x1 - x0)
    ii += 1
    print("Solución x1 = ", x1)
    print("Diferencia entre iteraciones = ", Er)
    print("Iteración No. ", ii)
    print()
```

- Si código converge a valor fijo $\Rightarrow$ este será solución de: $x = 2 - e^{-x}$
- Este método se llama "Método de Relajación"
- Problemas: Ecuación que se resuelve debe estar en la forma $x = f(x)$
- Si no está en esa forma, hay que reordenarla a esa forma

Ejercicio

- Considere la ec. $x = 1 - e^{-cx}$, donde c es una cte y x es desconocida. Escriba un programa que resuelva la ecuación usando el método de relajación para $c = 2$. Calcule la solución con una exactitud de 10^{-6}.

- Vimos forma sencilla de calcular ecuación no lineal: Método de relajación
- En el libro (Newman) y otros libros se describen otros métodos:
  - Búsqueda binaria
  - Método de la secante
  - Método de Newton

Python tiene implementado función fsolve en paquete scipy!!

Función fsolve de scipy

- Pasos para resolver ecuación no-lineal con fsolve:
  - Poner ecuación en la forma f(x) = 0. Conocemos la función y queremos encontrar el valor de x que es solución de $f(x) = 0$
  - Definimos $f(x)$
  - Damos un valor inicial para x
  - Importamos la librería fsolve dentro de scipy.optimize
  - Usamos x1 = fsolve(f, x0)

Función fsolve de scipy

- Ejemplo 1:
- Resolver ecuación no-lineal:

$$x = 2 - e^{-x}$$

Función fsolve de scipy

- Ejemplo 1:
- Resolver ecuación no-lineal:

$$x = 2 - e^{-x}$$

- from scipy.optimize import fsolve
- def f(x):
-     return 2 - np.exp(-x) - x
- sol = fsolve(f, 0.5) # fsolve recibe función y estimación inicial
- print("Solución de ecuación = ", sol[0])
- print("Valor de f(x) = ", f(sol))

Función fsolve de scipy

- Ejemplo 2:
- Resolver ecuación no-lineal:

$$x = e^{1-x^2}$$

Función fsolve de scipy

- Ejemplo 2:
- Resolver ecuación no-lineal:

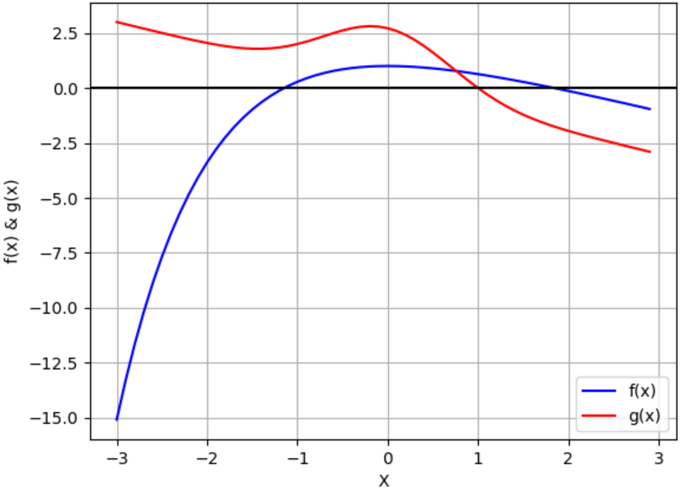
$$x = e^{1-x^2}$$

- import numpy as np
- from scipy.optimize import fsolve
- def f(x):
-     return x - np.exp(1 - x**2)
- sol = fsolve(f, 0.2) # fsolve recibe función y estimación inicial
- print("Solución de ecuación = ", sol[0])
- print("Valor de f(x) = ", f(sol))

Cómo escoger valor inicial?

- import matplotlib.pyplot as plt
- import numpy as np
- def f(x):
-     return 2 - np.exp(-x) - x
- def g(x):
-     return np.log(x) + x**2 - 1
- x = np.arange(-4,4,0.1)
- plt.clf()
- plt.plot(x, f(x), '-b', x, g(x), '-r')
- plt.ylabel('f(x) & g(x)'); plt.xlabel('X')
- plt.axhline(0, -4, 4, color='k')
- plt.legend(('f(x)', 'g(x)'), loc=2)
- plt.grid()

Cómo escoger valor inicial?



- Método gráfico tiene valor práctico limitado. No son muy precisos.
- Se usan para obtener valor estimado de raíz para usar en métodos más exactos.
- Son útiles para entender las propiedades de las funciones y anticipar los problemas de los métodos numéricos.

Función fsolve de scipy

- Ejemplo 3:
- Encontrar todas las soluciones de ecuación no-lineal:

$$x = 2 - e^{-x}$$

Función fsolve de scipy

- Ejemplo 3:
- Encontrar todas las soluciones de ecuación no-lineal:

$$x = 2 - e^{-x}$$

- import numpy as np
- from scipy.optimize import fsolve
- def f(x):
-     return x - 2 + np.exp(-x)
- sol = fsolve(f, np.array([-5, 5]))
- print("Soluciones de ecuación = ", sol)

Sistema de ecuaciones no-lineales

- Supongamos que tenemos dos ecuaciones con 2 incógnitas:
- $h(y, z) = 0$ y $g(y, z) = 0$ tal que:

$$h(y, z) = y + 2z$$

$$g(y, z) = \sin(y)/z$$

- Podemos usar función fsolve para calcular este sistema de ecuaciones?
- Se puede usar para sistemas de 4, 5, o más ecuaciones?

Sistema de ecuaciones no-lineales

Aproximación general:

- En lugar de resolver $f(x) = 0$, se resuelve $\vec{f}(\vec{x}) = 0$
- Donde $\vec{x}$ es un vector de incógnitas y $\vec{f}$ es un vector de funciones
- Podemos escribir ejemplo anterior como:

$$f_0(x_0, x_1) = x_0 + 2x_1,$$
$$f_1(x_0, x_1) = \sin(x_0)/x_1,$$

- En forma matricial:

$$\begin{bmatrix} f_0(x_0, x_1) \\ f_1(x_0, x_1) \end{bmatrix} = \begin{bmatrix} x_0 + 2x_1 \\ \sin(x_0)/x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Sistema de ecuaciones no-lineales

- En forma matricial:

$$\begin{bmatrix} f_0(x_0, x_1) \\ f_1(x_0, x_1) \end{bmatrix} = \begin{bmatrix} x_0 + 2x_1 \\ \sin(x_0)/x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- o como:

$$\vec{f}(\vec{x}) = \begin{bmatrix} x_0 + 2x_1 \\ \sin(x_0)/x_1 \end{bmatrix} = \vec{0},$$

- Se resuelve sistema como antes:
- Se define función vectorial que toma arreglo de incógnitas $\vec{x}$ y entrega arreglo de valores de $\vec{f}$
- Se entrega arreglo de valores iniciales a función fsolve

Sistema de ecuaciones no-lineales

$$h(y, z) = y + 2z$$

$$g(y, z) = \sin(y)/z$$

```
import numpy as np
def hg(yz):
    y = yz[0]
    z = yz[1]
    h = y + 2*z
    g = np.sin(y)/z

    return np.array([h, g])
yz0 = np.array([1, 2])
yz = fsolve(hg, yz0)
print(yz)
```

Sistema de ecuaciones no-lineales

Otra forma de programarlo

$$h(y, z) = y + 2z$$

$$g(y, z) = \sin(y)/z$$

```
import numpy as np
def f(x):

    f0 = x[0] + 2*x[1]
    f1 = np.sin(x[0])/x[1]

    return np.array([f0, f1])
x0 = np.array([1, 2])
yz = fsolve(f, x0)
print(yz)
```

Ejercicio - Sistema de ecuaciones no-lineales

Resuelva el siguiente sistema de ecuaciones con 3 incógnitas:

$$x^2 + y^2 = 1$$

$$xy + yz = -1.1$$

$$y^2 + z^2 = 2$$

Ejercicio - Sistema de ecuaciones no-lineales

Resuelva el siguiente sistema de ecuaciones con 3 incógnitas:

$$x^2 + y^2 = 1$$

$$xy + yz = -1.1$$

$$y^2 + z^2 = 2$$

- import numpy as np
- def f(x):
-     f0 = x[0]**2 + x[1]**2 - 1
-     f1 = x[0]*x[1] + x[1]*x[2] + 1.1
-     f2 = x[1]**2 + x[2]**2 - 2
-     return np.array([f0, f1, f2])
- xyz0 = np.array([1, 2, 3])
- xyz = fsolve(f, xyz0)
- print(xyz)