Guía de laboratorio 6: Resolución de Sistemas de Ecuaciones No Lineales (Métodos Multivariables: Punto Fijo y Newton-Raphson modificado).

1. Introducción

Los problemas que modelan fenómenos físicos, químicos o de ingeniería frecuentemente conducen a sistemas de ecuaciones no lineales. Para resolverlos numéricamente se usan métodos iterativos. En esta práctica veremos dos técnicas importantes:

- **Método de Punto Fijo Multivariable**: transforma el sistema $F(x) = 0$ en $x = G(x)$ y aplica la iteración $x^{(k+1)} = G(x^{(k)})$.
- **Newton–Raphson modificado**: variante de Newton en la que el jacobiano se evalúa en la aproximación inicial y se mantiene (jacobiano "congelado"), reduciendo el coste de evaluación del jacobiano en cada iteración.

En esta práctica de laboratorio se abordará la resolución de diversas funciones no lineales de dos y tres *variables* utilizando ambos métodos, con el fin de comparar su eficiencia, precisión y número de iteraciones necesarias. Además, se implementarán en Python para observar la evolución de las aproximaciones y analizar los resultados obtenidos.

2. Objetivos

- Entender la formulación de punto fijo multivariable y condiciones intuitivas de convergencia (contracción).
- Implementar y comparar el comportamiento del método de punto fijo con el de Newton–Raphson modificado.
- Practicar la implementación en Python (vectores, jacobiano, normas, control de convergencia).
- Analizar convergencia numérica, número de iteraciones y sensibilidad a la elección de la transformación G o al jacobiano congelado.

3. Materiales y Recursos

- Computadora personal o laboratorio de cómputo.
- Python 3.10+ instalado (preferible en Anaconda).
- Librería *math*, *numpy*, *pandas* (pip install *nombre librería*).
- Editor de código (Sublime Text, Visual Studio Code o Jupyter Notebook).
- Opcional (crear un ambiente virtual)

4. Fundamentación Teórica

Es un problema central en el análisis numérico y en aplicaciones de la ingeniería, la física y la economía. Como muchas funciones no admiten soluciones exactas, se utilizan *métodos iterativos* que generan aproximaciones sucesivas a la raíz buscada.

En esta práctica se abordará dos de los métodos más empleados: el *Método del punto fijo multivariable y Newton-Raphson modificado (jacobiano congelado).*

Método de Punto fijo Multivariable

Dado $x \in \mathbb{R}^n$ y una función $G : \mathbb{R}^n \to \mathbb{R}^n$, la iteración es

$$x^{(k+1)} = G\big(x^{(k)}\big).$$

Convergencia típica si G es contráctil en una vecindad del punto fijo: $\|DG(x)\| < 1$ en alguna norma. Es simple, pero depende fuertemente de la elección de G.

Método de Newton-Raphson modificado (Jacobiano congelado)

Partimos de $F(x) = 0$. En Newton estándar

$$x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)})F(x^{(k)}).$$

En la versión modificada calculamos $J_0 = J\,(x^{(0)})$ y reutilizamos J_0 en todas las iteraciones:

$$x^{(k+1)} = x^{(k)} - J_0^{-1}F(x^{(k)}).$$

Ventaja: menos evaluaciones del Jacobiano / menos factorizaciones si J_0 se factoriza una vez. Inconveniente: pérdida de cuadrática; convergencia solo si la función se comporta bien respecto a J_0

4. Ejercicios y Desarrollo

Se plantearon ocho ejercicios correspondientes a funciones no lineales de dos variables, se ha definido la solución en código Python utilizando funciones, cuatro para Punto Fijo y cuatro para Newton-Raphson modificado, se imprime los 5 últimos registros de las iteraciones dadas, se indica también el número de iteraciones para dar con el resultado. Además, se proponen 4 ejercicios para su codificación, finalmente se establece las conclusiones.

Punto Fijo (4 ejercicios)

PF1 — sistema 2×2

- $x = \cos(y)$
- $y = \sin(x)$
- $x^{(0)} = (0.5, 0.5)$

Indicar resultado.

PF2 — sistema 2×2 (simétrico con raíces cuadradas)

- $x = \sqrt{\frac{1+y}{2}}$
- $y = \sqrt{\frac{1+x}{2}}$
- $x^{(0)} = (0.5, 0.5)$

Indicar resultado.

PF3 — sistema 2×2 (sin/cos escalado)

- $x = \frac{\sin(y)+1}{2}$
- $y = \frac{\cos(x)+1}{2}$
- $x^{(0)} = (0.3, 0.3)$

Indicar resultado.

PF4 — sistema 3×3

- $x = \dfrac{\cos(yz) + 1}{2}$
- $y = \dfrac{\sin(x) + 1}{3}$
- $z = \dfrac{xy + 1}{4}$
- $x^{(0)} = (0.2, 0.2, 0.2)$

Indicar resultado.

Newton–Raphson modificado (ejercicios)

NRM1 — 2×2 (círculo + exponencial)

- $f_1(x, y) = x^2 + y^2 - 4$
- $f_2(x, y) = e^x + y - 1$
- $x^{(0)} = (1.0, 1.0)$

Indicar resultado.

NRM2 — 3×3

- $f_1 = x + y + z - 3$
- $f_2 = x^2 + y^2 + z^2 - 5$
- $f_3 = e^x + y - z - 1$
- $x^{(0)} = (0.5, 1.0, 1.5)$

Indicar resultado.

NRM3 — 2×2 (cuadráticas simétricas)

- $f_1 = x^2 - y - 1$
- $f_2 = y^2 - x - 1$
- $x^{(0)} = (1.5, 1.5)$

Indicar resultado.

NRM4 — 2×2 (sin/cos mix)

- $f_1 = \sin(x) + y - 1$
- $f_2 = x + \cos(y) - 0.5$
- $x^{(0)} = (0.5, 0.5)$

Indicar resultado.

```
"""
Laboratorio Practica 6: Métodos Multivariables (Punto Fijo y Newton - Raphson
modificado)

Incluye:
- 4 ejercicios con Punto Fijo multivariable
- 4 ejercicios con Newton - Raphson modificado
"""

import numpy as np
import pandas as pd
from math import sin, cos, sqrt, exp

# ----------------------------------------------------------------
# MODELAMIENTO COMPUTACIONAL PARA INGENIERIA - Métodos numéricos
# ----------------------------------------------------------------

def punto_fijo_multivariable(g, x0, tol=1e-8, maxiter=100):
    """
    Método de punto fijo multivariable: x_{k+1} = g(x_k)
    g: función que recibe vector y devuelve vector (numpy array)
    x0: vector inicial
    """
    x = x0.astype(float).copy()
    history = []
    for k in range(1, maxiter+1):
        x_new = np.asarray(g(x))
        err = np.linalg.norm(x_new - x, ord=np.inf)
        history.append((k, x.copy(), x_new.copy(), err))
        x = x_new
        if err < tol:
            break
    return x, err, k, history

def newton_modificado(f, J, x0, tol=1e-8, maxiter=50):
    """
    Newton-Raphson modificado (Jacobiano congelado en x0):
    - Evalúa J0 = J(x0) una vez y lo usa para todas las iteraciones.
    - Cada iteración resuelve J0 * delta = -f(x_k)
    """
    x = x0.astype(float).copy()
    J0 = np.asarray(J(x0))
    history = []
    try:
        for k in range(1, maxiter+1):
            fx = np.asarray(f(x))
            delta = np.linalg.solve(J0, -fx)
            x_new = x + delta
```

```
            err = np.linalg.norm(delta, ord=np.inf)
            resnorm = np.linalg.norm(fx, ord=2)
            history.append((k, x.copy(), delta.copy(), err, resnorm))
            x = x_new
            if err < tol and resnorm < tol:
                break
        return x, err, k, history
    except Exception as e:
        return None, str(e), 0, history

def mostrar_historial_puntofijo(history, names=None):
    rows = []
    for k, x_old, x_new, err in history:
        row = {"iter": k}
        if names is None:
            for i, val in enumerate(x_new):
                row[f"x{i}"] = val
        else:
            for i, name in enumerate(names):
                row[name] = x_new[i]
        row["||x_{k+1}-x_k||_inf"] = err
        rows.append(row)
    return pd.DataFrame(rows)

def mostrar_historial_newton(history, names=None):
    rows = []
    for k, x_old, delta, err, resnorm in history:
        row = {"iter": k}
        if names is None:
            for i, val in enumerate(x_old):
                row[f"x{i}"] = val
        else:
            for i, name in enumerate(names):
                row[name] = x_old[i]
        for i, d in enumerate(delta):
            row[f"delta{i}"] = d
        row["||delta||_inf"] = err
        row["||f(x)||_2"] = resnorm
        rows.append(row)
    return pd.DataFrame(rows)

# ---------------------------
# EJERCICIOS DE PUNTO FIJO
# ---------------------------

# Ejercicio PF1: x = cos(y), y = sin(x)
def g1(v):
    x, y = v
```

```
    return np.array([cos(y), sin(x)])
x0_1 = np.array([0.5, 0.5])

# Ejercicio PF2: x = sqrt((1+y)/2), y = sqrt((1+x)/2)
def g2(v):
    x, y = v
    return np.array([sqrt((1+y)/2), sqrt((1+x)/2)])
x0_2 = np.array([0.5, 0.5])

# Ejercicio PF3: x = (sin(y)+1)/2, y = (cos(x)+1)/2
def g3(v):
    x, y = v
    return np.array([(sin(y)+1)/2, (cos(x)+1)/2])
x0_3 = np.array([0.3, 0.3])

# Ejercicio PF4: sistema 3 variables
def g4(v):
    x, y, z = v
    return np.array([(cos(y*z)+1)/2, (sin(x)+1)/3, (x*y + 1)/4])
x0_4 = np.array([0.2, 0.2, 0.2])

# ---------------------------
# EJERCICIOS NEWTON-RAPHSON MODIFICADO
# ---------------------------

# Ejercicio NRM1: f1 = x^2+y^2-4, f2 = exp(x)+y-1
def f_nr1(v):
    x, y = v
    return np.array([x**2 + y**2 - 4, exp(x) + y - 1])
def J_nr1(v):
    x, y = v
    return np.array([[2*x, 2*y],[exp(x), 1.0]])
x0_nr1 = np.array([1.0, 1.0])

# Ejercicio NRM2: sistema 3 variables
def f_nr2(v):
    x, y, z = v
    return np.array([x + y + z - 3, x**2 + y**2 + z**2 - 5, exp(x) + y - z -
1])
def J_nr2(v):
    x, y, z = v
    return np.array([[1,1,1],[2*x,2*y,2*z],[exp(x),1,-1]])
x0_nr2 = np.array([0.5, 1.0, 1.5])

# Ejercicio NRM3: f1 = x^2 - y -1, f2 = y^2 - x -1
def f_nr3(v):
    x, y = v
    return np.array([x**2 - y - 1, y**2 - x - 1])
```

```
def J_nr3(v):
    x, y = v
    return np.array([[2*x, -1],[-1, 2*y]])
x0_nr3 = np.array([1.5, 1.5])

# Ejercicio NRM4: f1 = sin(x)+y-1, f2 = x+cos(y)-0.5
def f_nr4(v):
    x, y = v
    return np.array([sin(x) + y - 1, x + cos(y) - 0.5])
def J_nr4(v):
    x, y = v
    return np.array([[cos(x), 1], [1, -sin(y)]])
x0_nr4 = np.array([0.5, 0.5])

# ---------------------------
# MAIN (PRINCIPAL): resolver todos los ejercicios
# ---------------------------

if __name__ == "__main__":

    print("===== PUNTO FIJO =====")
    for i, (g, x0, name) in enumerate([
        (g1,x0_1,"PF1: cos/sen"),
        (g2,x0_2,"PF2: sqrt-symmetric"),
        (g3,x0_3,"PF3: sen/cos scaled"),
        (g4,x0_4,"PF4: 3-variable")]):

        sol, err, iters, hist = punto_fijo_multivariable(g, x0, tol=1e-10,
maxiter=500)
        df = mostrar_historial_puntofijo(hist)
        print(f"{name}: solución = {sol}, iteraciones = {iters}, error final =
{err}")
        print(df.tail(), "\n")

    print("===== NEWTON-RAPHSON MODIFICADO =====")
    for i, (f, J, x0, name) in enumerate([
        (f_nr1,J_nr1,x0_nr1,"NRM1: circle+exp"),
        (f_nr2,J_nr2,x0_nr2,"NRM2: 3-var"),
        (f_nr3,J_nr3,x0_nr3,"NRM3: symmetric quad"),
        (f_nr4,J_nr4,x0_nr4,"NRM4: sin/cos mix")]):

        sol, err, iters, hist = newton_modificado(f, J, x0, tol=1e-10,
maxiter=50)
        if sol is None:
            print(f"{name}: error -> {err}")
        else:
            df = mostrar_historial_newton(hist)
```

```
            print(f"{name}: solución = {sol}, iteraciones = {iters}, error
final = {err}")
            print(df.tail(), "\n")
```

NOTA: Complete la codificación en los ejercicios

Ejercicio 9 – Punto Fijo de 3 variables

Transformación $x = G(x)$ (usar la iteración $x^{(k+1)} = G(x^{(k)})$):

$$\begin{cases} x = \dfrac{\cos(yz)+1}{3}, \\ y = \dfrac{\sin(x)+1}{4}, \\ z = \dfrac{x\,y+1}{5}. \end{cases}$$

- **Vector inicial sugerido:** $x^{(0)} = (0.2,\ 0.2,\ 0.2)$.
- **Tolerancia:** $\mathrm{tol} = 10^{-10}$, máximo iteraciones 500.

Mostrar resultado y tabla de iteraciones

Ejercicio 10 – Punto Fijo de 3 variables

Transformación G simétrica (apto para variables no negativas):

$$\begin{cases} x = \sqrt{\dfrac{1+y+z}{4}}, \\ y = \sqrt{\dfrac{1+x+z}{4}}, \\ z = \sqrt{\dfrac{1+x+y}{4}}. \end{cases}$$

- **Vector inicial sugerido:** $x^{(0)} = (0.5,\ 0.5,\ 0.5)$.
- **Tolerancia:** 10^{-10}, máximo 500 iter.

Mostrar resultado y tabla de iteraciones

5. Conclusiones (Máximo en 10 líneas)