Tarea 2

Eli-246

Integrantes: - Roy Quinchaleo
- Kevin Tarqui
Profesor: - Guillermo Huerta Soto

Fecha de entrega: 5 de julio de 2024

Índice de Contenidos

1. Introducción **1**
1.1. Objetivo del Informe . . . 1

2. Desarrollo **2**
2.1. Análisis . . . 2
2.2. Demostración Matemática . . . 2
2.2.1. Método Iterativo de Newton-Raphson . . . 3
2.2.2. Método de Newton-Raphson en Varias Variables . . . 3
2.2.3. Ejemplo . . . 4
2.2.4. Diagrama de Flujo del Algoritmo . . . 7
2.2.5. Algoritmo Python para Newton-Raphson . . . 8
2.2.6. Flujo de Potencia del Circuito . . . 11
2.2.7. Comparación de Resultados . . . 12
2.2.8. Función de Error y Escenarios de Tolerancia . . . 14

3. Consideraciones para Implementación de Modelos en Python **16**
3.1. Importación de Librerías . . . 16
3.2. Modelo de Líneas y Transformadores . . . 16
3.3. Configuración de la Barra Slack . . . 16
3.4. Porcentajes de las Cargas . . . 16

4. Conclusiones **17**

Índice de Figuras

1.1. Sistema bajo estudio . . . 1
2.1. Sistema bajo estudio . . . 2
2.2. Ejemplo . . . 4
2.3. Diagrama de Flujo . . . 7

Índice de Tablas

2.1. Resultados de la Iteración 1 . . . 12
2.2. Resultados de la Iteración 2 . . . 12
2.3. Resultados de la Iteración 3 . . . 12
2.4. Resultados pandapower . . . 13
2.5. Resultados finales del método de Newton-Raphson . . . 13
2.6. Resultados de referencia (Pandapower) . . . 14
2.7. Resultados finales del método de Newton-Raphson . . . 14
2.8. Resultados con tolerancia 0.0001 (Converged in 20 iterations) . . . 14
2.9. Resultados con tolerancia 1e-06 (Converged in 20 iterations) . . . 15
2.10. Resultados con tolerancia 1e-08 (Converged in 20 iterations) . . . 15

1. Introducción

1.1. Objetivo del Informe

En el contexto del estudio del flujo de potencia, se ha propuesto realizar el análisis y simulación del comportamiento de un sistema eléctrico utilizando técnicas avanzadas de programación y simulación. Para ello, se empleará un circuito específico (Figura 1.1), mismo circuito de la Tarea 1, y se utilizará Python con la biblioteca pandapower.

El objetivo principal del informe es comprender cómo las variaciones en la topología del sistema y en los consumos influyen en las métricas de las barras, en otras palabras, la alteraciones que presentan las barras en sus valores. Esto se logrará mediante la implementación del método iterativo de Newton-Raphson para resolver el flujo de potencia. A lo largo del presente informe se abordará, también, la construcción del Jacobiano y el desarrollo de algoritmos para la obtención de los resultados.

Además, se comparará lo obtenido por el método de Newton-Raphson con los valores provistos por pandapower y sus herramientas, con el fin de determinar el error asociado a la utilización del método.

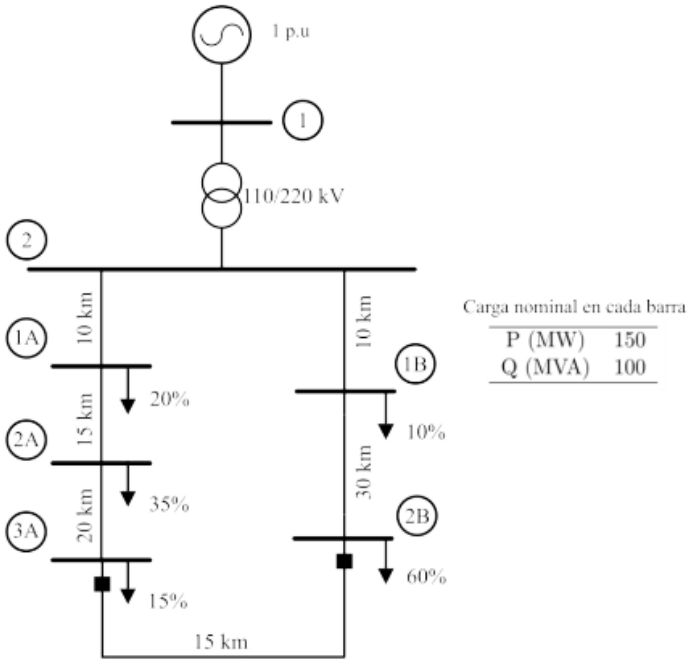

Figura 1.1: Sistema bajo estudio

2. Desarrollo

2.1. Análisis

2.2. Demostración Matemática

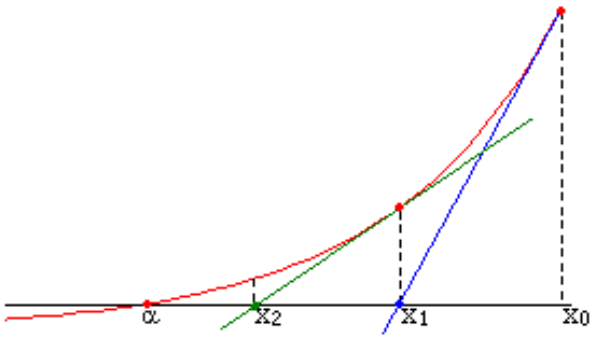

Figura 2.1: Sistema bajo estudio

Paso 1: Estimación Inicial ($\mathbf{x_0}$)

Se selecciona un punto inicial $\mathbf{x_0}$ cercano a la raíz buscada de la función $f(x)$. En este punto, el valor de la función es $y_0 = f(x_0)$.

Paso 2: Ecuación de la Tangente en $\mathbf{x_0}$

Se traza la tangente a la curva en el punto (x_0, y_0). La pendiente de esta tangente es $f'(x_0)$, la derivada de $f(x)$ en x_0.

La ecuación de la tangente es:

$$y = f(x_0) + f'(x_0)(x - x_0)$$

Paso 3: Punto de Intersección de la Tangente con el Eje x

Para encontrar el punto donde la tangente cruza el eje x, se resuelve la ecuación de la tangente igualando y a 0:

$$0 = f(x_0) + f'(x_0)(x_1 - x_0)$$

Al despejar x_1, se obtiene:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Paso 4: Repetición del Proceso

El proceso se repite utilizando x_1 como el nuevo punto inicial:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Este ciclo continúa hasta que x_n converja a la raíz deseada con una precisión aceptable.

2.2.1. Método Iterativo de Newton-Raphson

Demostración matemática del método y explicación detallada del Jacobiano de la función.
Por ecuaciones de flujo de potencia

$$S^* = diag(V^*)Y_bV = V^* \cdot (Y_BV) \tag{2.1}$$

Se recurre a metodos iterativos para encontrar solucion debido a su forma de trabajo en variables complejas con sistema no lineal de ecuaciones.

Basado en aproximaciones de series de Taylor, Newton-Raphsonn,tanto para forma unidimensional y multidimensional, El Jacobiano presente en la forma multidimensional contiene las derivadas parciales de las componentes F respecto a las variables x.

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Sistema

$$x_{k+1} = x_n - J(x_k)^{-1}F(x_k) \tag{2.2}$$

Problema flujo de potencia

$$S_i^* : \begin{cases} P_i = V_i \sum_{j=1}^{N} V_i(G_{ij}cos\theta_{ij} + B_{ij}sin\theta_{ij}) \\ Q_i = V_i \sum_{j=1}^{N} V_i(G_{ij}sin\theta_{ij} - B_{ij}cos\theta_{ij}) \end{cases} \tag{2.3}$$

Se definirá el método Newton-Raphson, en especifico, en Varias Variables para su utilización, esto debido a la complejidad del sistema y sus múltiples variables asociadas.

2.2.2. Método de Newton-Raphson en Varias Variables

El método de Newton-Raphson se generaliza de la siguiente forma para Varias Variables:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}^{-1}(\mathbf{x}_n)\mathbf{F}(\mathbf{x}_n) \tag{2.4}$$

Donde:

$$\mathbf{x} = \begin{pmatrix} \delta_1 \\ \vdots \\ \delta_{NPQ} \\ \vdots \\ \delta_{NPQ+NPV} \\ V_1 \\ \vdots \\ V_{NPQ} \end{pmatrix} = \begin{bmatrix} \delta \\ - \\ V \end{bmatrix} \tag{2.5}$$

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} \Delta P_1 \\ \vdots \\ \Delta P_{NPQ} \\ \vdots \\ \Delta P_{NPQ+NPV} \\ \Delta Q_1 \\ \vdots \\ \Delta Q_{NPQ} \end{pmatrix} = \begin{bmatrix} \Delta P \\ - \\ \Delta Q \end{bmatrix} \tag{2.6}$$

Para el calculo del Jacobino se hará uso de las ecuaciones vistas en clases.

$$\mathbf{J} = \begin{bmatrix} H & N \\ M & L \end{bmatrix} \tag{2.7}$$

$$H_{ij} = \begin{cases} -V_i V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}), & \text{si } i \neq j \\ Q_i + B_{ii} V_i^2, & \text{si } i = j \end{cases} \tag{2.8}$$

$$N_{ij} = \begin{cases} -V_i (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}), & \text{si } i \neq j \\ -P_i/V_i - V_{ii} G_{ii}, & \text{si } i = j \end{cases} \tag{2.9}$$

$$M_{ij} = \begin{cases} V_i V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}), & \text{si } i \neq j \\ -P_i + G_{ii} V_i^2, & \text{si } i = j \end{cases} \tag{2.10}$$

$$L_{ij} = \begin{cases} -V_i (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}), & \text{si } i \neq j \\ -Q_i/V_i + B_{ii} V_i, & \text{si } i = j \end{cases} \tag{2.11}$$

Para la demostración del método se usará un sistema simple.

2.2.3. Ejemplo

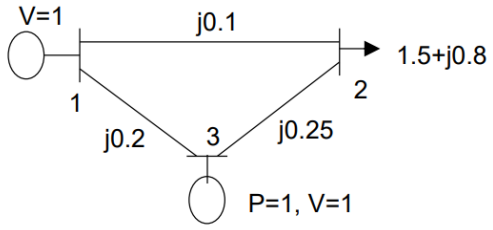

Figura 2.2: Ejemplo

Parámetros del Sistema:

1. **Barra 1: Slack**
   - $\delta = 0\,^\circ$
   - $V = 1\,\text{pu}$
2. **Barra 2: PQ**
   - $1{,}5 + j0{,}8\,\text{pu}$
3. **Barra 3: PV**
   - $P = 1\,\text{pu}$
   - $V = 1\,\text{pu}$

Al observar el sistema se aprecia una barra Slack que no entrega ecuaciones ni incógnitas, una PQ que añade 2 ecuaciones y 2 incógnitas (δ2, V2),finalmente una barra Pv que adiciona una ecuación y una incógnita (V3).

1. **Entre Barra 1 y Barra 2:**

$$Y_{12} = \frac{1}{j0{,}1} = -j10$$

2. **Entre Barra 1 y Barra 3:**

$$Y_{13} = \frac{1}{j0{,}2} = -j5$$

3. **Entre Barra 3 y Barra 2:**

$$Y_{23} = \frac{1}{j0{,}25} = -j4$$

Matriz de admitancia del sistema:

$$\mathbf{Y} = \begin{bmatrix} -j15 & -j10 & -j5 \\ -j10 & -j14 & -j4 \\ -j5 & -j4 & -j9 \end{bmatrix} \tag{2.12}$$

Se obtiene H N M L:

$$\mathbf{H} = \begin{bmatrix} -14{,}8 & 4 \\ 4 & -9 \end{bmatrix} \tag{2.13}$$

$$\mathbf{N} = \begin{bmatrix} -10 & -4 \\ -4 & -1 \end{bmatrix} \tag{2.14}$$

$$\mathbf{M} = \begin{bmatrix} 10 & 4 \\ 4 & 1 \end{bmatrix} \tag{2.15}$$

$$\mathbf{L} = \begin{bmatrix} 14 & 4 \\ 4 & -10 \end{bmatrix} \tag{2.16}$$

Jacobiano:

$$\mathbf{J} = \begin{bmatrix} -14{,}8 & -10 & -4 \\ 10 & 14 & 4 \\ 4 & 4 & -10 \end{bmatrix} \tag{2.17}$$

Jacobiano inverso:

$$\mathbf{J}^{-1} = \begin{bmatrix} -0{,}12862797 & -0{,}09564644 & 0{,}01319261 \\ 0{,}09564644 & 0{,}13522427 & 0{,}01583113 \\ -0{,}01319261 & 0{,}01583113 & -0{,}0883905 \end{bmatrix} \tag{2.18}$$

Variables primera iteración:

$$\mathbf{Xo} = \begin{bmatrix} \theta_2 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \tag{2.19}$$

$$\mathbf{F(Xo)} = \begin{bmatrix} P_2^{\text{calc}} - P_2^{\text{spec}} \\ Q_2^{\text{calc}} - Q_2^{\text{spec}} \\ Q_3^{\text{calc}} - Q_3^{\text{spec}} \end{bmatrix} = \begin{bmatrix} -1{,}5 \\ -0{,}8 \\ 0 \end{bmatrix} \tag{2.20}$$

$$\mathbf{x_1} = \mathbf{x_0} + \Delta\mathbf{x_0} \tag{2.21}$$

$$\mathbf{x_1} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -0{,}12862797 & -0{,}09564644 & 0{,}01319261 \\ 0{,}09564644 & 0{,}13522427 & 0{,}01583113 \\ -0{,}01319261 & 0{,}01583113 & -0{,}0883905 \end{bmatrix} \begin{bmatrix} -1{,}5 \\ -0{,}8 \\ 0 \end{bmatrix} \tag{2.22}$$

$$\mathbf{x_1} = \begin{bmatrix} -0{,}2694591 \\ 1{,}25164908 \\ 0{,}99287599 \end{bmatrix} \tag{2.23}$$

De esta forma se realiza la primera iteración del método, obteniéndose las variables del sistema.

2.2.4. Diagrama de Flujo del Algoritmo

Presentar el diagrama de flujo y explicar cada paso del algoritmo.

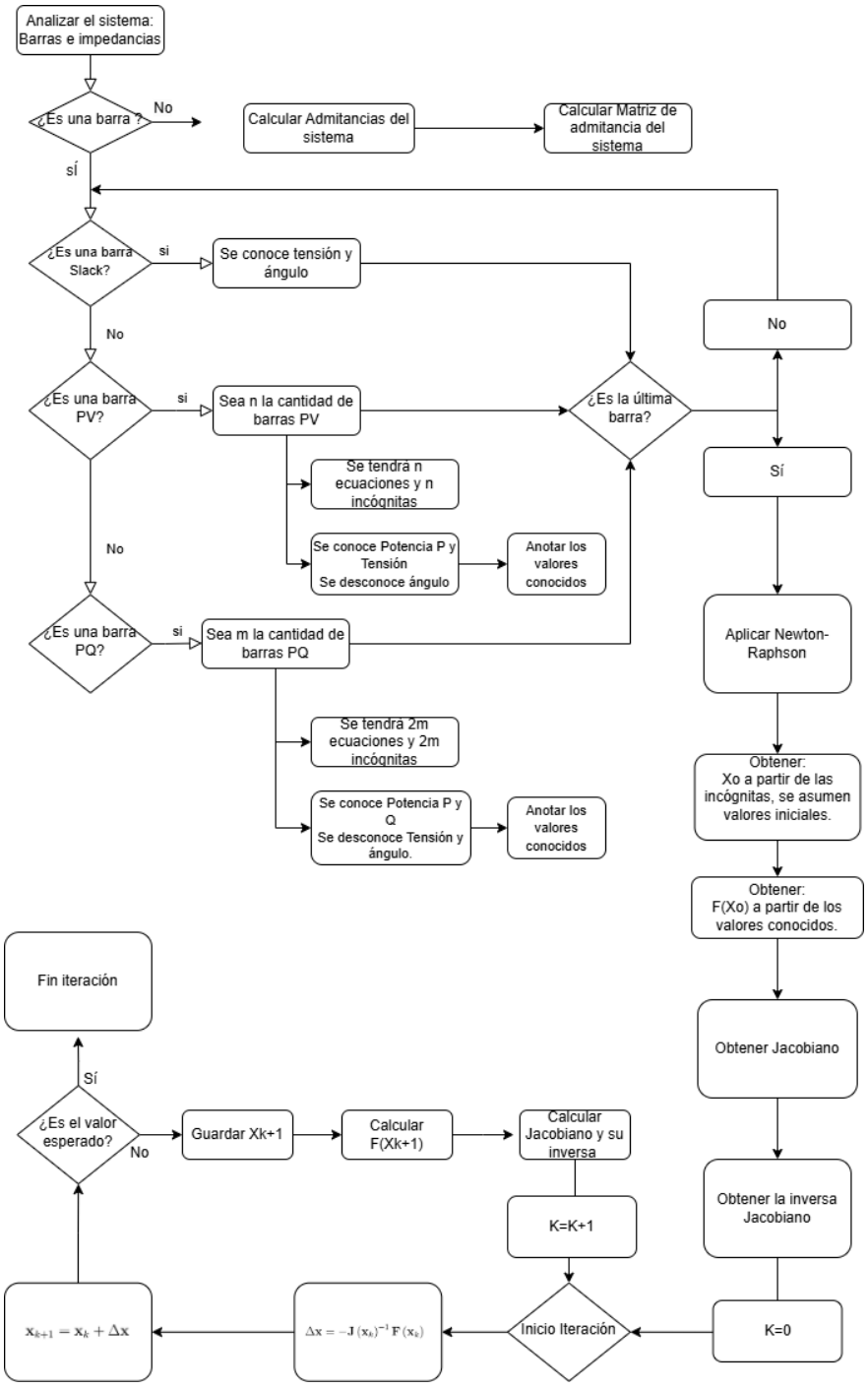

Figura 2.3: Diagrama de Flujo

2.2.5. Algoritmo Python para Newton-Raphson

Definición de Variables

```
# Matriz Ybus
Y_bus = net._ppc['internal']['Ybus'].todense()

# Se definen los valores iniciales para las tensiones y ángulos desconocidos.
V = np.ones(len(net.bus))  # Tensiones en 1.0 p.u.
theta = np.zeros(len(net.bus))  # Ángulos en 0 radianes.

# Se definen las potencias especificadas (P, Q) en pu.
S_base = 100  # Base de potencia en MVA
P_variable = np.zeros(len(net.bus))  # Se empieza con ceros
Q_variable = np.zeros(len(net.bus))  # Se empieza con ceros

# Se configuran las potencias especificas.
for load in net.load.itertuples():
    P_variable[load.bus] -= load.p_mw / S_base  # Convertir a p.u.
    Q_variable[load.bus] -= load.q_mvar / S_base  # Convertir a p.u.

# Se definen las barras PV y PQ; se excluye barra slack.
slack_bus = int(net.ext_grid.bus.values[0])
pv_buses = [bus for bus in net.gen.bus if bus != slack_bus]
pq_buses = [bus for bus in net.load.bus if bus != slack_bus]
```

Desbalance de Potencia

```
# Función para calcular el desbalance de potencia
def calc_power_mismatch(Y_bus, V, theta, P_spec, Q_spec, pv_buses, pq_buses):
    # Número de barras
    n_buses = len(V)

    # Potencias activa (P) y reactiva (Q)
    P = np.zeros(n_buses)
    Q = np.zeros(n_buses)

    for i in range(n_buses):
        for j in range(n_buses):
            P[i] += V[i] * V[j] * (Y_bus[i,j].real * np.cos(theta[i] - theta[j]) + Y_bus[i,j].imag * np.sin(theta[i] -
    ↪ theta[j]))
            Q[i] += V[i] * V[j] * (Y_bus[i,j].real * np.sin(theta[i] - theta[j]) - Y_bus[i,j].imag * np.cos(theta[i] -
    ↪ theta[j]))

    # Delta P y Delta Q
    dP = P_variable - P
    dQ = Q_variable - Q

    # Construcción de vector.
    mismatch = []

    # Para las barras PV, sólo se incluye P.
    for i in pv_buses:
        mismatch.append(dP[i])

```

```
    # Para las barras PQ, se incluye el P y Q.
    for i in pq_buses:
        mismatch.append(dP[i])
        mismatch.append(dQ[i])

    return np.array(mismatch)
```

Cálculo del Jacobiano

```
# Función para calcular la matriz Jacobiana
def calc_jacobian(Y_bus, V, theta, P, Q, pv_buses, pq_buses):
    n_buses = len(V)
    H = np.zeros((n_buses, n_buses))
    N = np.zeros((n_buses, n_buses))
    M = np.zeros((n_buses, n_buses))
    L = np.zeros((n_buses, n_buses))

    for i in range(n_buses):
        for j in range(n_buses):
            if i != j:
                H[i, j] = -V[i] * V[j] * (Y_bus[i, j].real * np.sin(theta[i] - theta[j]) - Y_bus[i, j].imag * np.cos(theta[
    ↪ i] - theta[j]))
                N[i, j] = -V[i] * (Y_bus[i, j].real * np.cos(theta[i] - theta[j]) + Y_bus[i, j].imag * np.sin(theta[i] -
    ↪ theta[j]))
                M[i, j] = V[i] * V[j] * (Y_bus[i, j].real * np.cos(theta[i] - theta[j]) + Y_bus[i, j].imag * np.sin(theta
    ↪ [i] - theta[j]))
                L[i, j] = -V[i] * (Y_bus[i, j].real * np.sin(theta[i] - theta[j]) - Y_bus[i, j].imag * np.cos(theta[i] -
    ↪ theta[j]))
            else:
                H[i, i] = Q[i] + V[i]**2 * Y_bus[i, i].imag
                N[i, i] = -P[i] / V[i] - V[i] * Y_bus[i, i].real
                M[i, i] = -P[i] + V[i]**2 * Y_bus[i, i].real
                L[i, i] = -Q[i] / V[i] + V[i] * Y_bus[i, i].imag

    # Se excluyendo la barra slack
    pv_pq_buses = pv_buses + pq_buses
    H = H[np.ix_(pv_pq_buses, pv_pq_buses)]
    N = N[np.ix_(pv_pq_buses, pq_buses)]
    M = M[np.ix_(pq_buses, pv_pq_buses)]
    L = L[np.ix_(pq_buses, pq_buses)]

    return H, N, M, L
```

Iteraciones de Newton-Raphson

```
# Iteraciones de Newton-Raphson
max_iter = 3
tol = 1e-6

for k in range(max_iter):
    P = net.res_bus.p_mw.values / S_base
    Q = net.res_bus.q_mvar.values / S_base

    F_xk = calc_power_mismatch(Y_bus, V, theta, P_spec, Q_spec, pv_buses, pq_buses)
```

```
10
11    if np.linalg.norm(F_xk) < tol:
12        print(f'Converged in {k} iterations')
13        break
14
15    H, N, M, L = calc_jacobian(Y_bus, V, theta, P, Q, pv_buses, pq_buses)
16
17    J_top = np.hstack((H, N))
18    J_bottom = np.hstack((M, L))
19    J = np.vstack((J_top, J_bottom))
20
21    J_inv = np.linalg.inv(J)
22    delta_x = -np.dot(J_inv, F_xk)
23
24    # Se Actualizan tensiones y ángulos
25    delta_theta = delta_x[:len(pv_buses) + len(pq_buses)]
26    delta_V = delta_x[len(pv_buses) + len(pq_buses):]
27
28    for i, bus in enumerate(pv_buses + pq_buses):
29        theta[bus] += delta_theta[i]
30
31    for i, bus in enumerate(pq_buses):
32        V[bus] += delta_V[i]
33
34    # Convertir los ángulos a grados
35    V_pu = V
36    theta_deg = np.degrees(theta)
37
38    # Imprimir las tensiones y ángulos
39    print(f'Iteration {k + 1}:')
40    print('Theta (grados):', theta_deg)
41    print('V (pu):', V_pu)
```

$$\text{Theta} = \begin{bmatrix} 0{,}0 \\ 10{,}98417496 \\ 11{,}11745259 \\ 11{,}26656624 \\ 11{,}12059448 \\ 11{,}37220741 \\ 11{,}0816659 \end{bmatrix} \tag{2.24}$$

$$\text{V (pu)} = \begin{bmatrix} 1. \\ 1. \\ 0{,}9999938 \\ 1{,}0015139 \\ 1{,}0094908 \\ 1{,}0107671 \\ 1{,}0039435 \end{bmatrix} \tag{2.25}$$

2.2.6. Flujo de Potencia del Circuito

Explicación del cálculo del flujo de potencia en el circuito de la Figura 1 y presentación de los resultados obtenidos.

Código para el Flujo de Potencia

```
# Ejecutar el flujo de potencia
pp.runpp(net)

# Resultados
print("Resultados del flujo de potencia:")
print("Tensiones de las barras (pu):")
print(net.res_bus[["vm_pu", "va_degree"]])

print("\nCargas de las barras:")
print(net.res_load[["p_mw", "q_mvar"]])

print("\nGeneración:")
print(net.res_ext_grid[["p_mw", "q_mvar"]])

print("\nFlujos de potencia de las líneas:")
print(net.res_line[["p_from_mw", "q_from_mvar", "p_to_mw", "q_to_mvar"]])
```

Resultados del Flujo de Potencia

$$\text{Tensiones (pu), Theta} = \begin{bmatrix} 1{,}000000 & 0{,}000000 \\ 1{,}000000 & -14{,}725125 \\ 0{,}997821 & -14{,}948784 \\ 0{,}995318 & -15{,}195704 \\ 0{,}994288 & -15{,}335016 \\ 0{,}993342 & -15{,}344773 \\ 0{,}998769 & -14{,}924710 \end{bmatrix} \tag{2.26}$$

$$\text{Cargas (MW, MVar)} = \begin{bmatrix} 30{,}0 & 20{,}0 \\ 52{,}5 & 35{,}0 \\ 22{,}5 & 15{,}0 \\ 90{,}0 & 60{,}0 \\ 15{,}0 & 10{,}0 \end{bmatrix} \tag{2.27}$$

$$\text{Generación externa (MW, MVar)} = \begin{bmatrix} 212{,}387223 & 22{,}789148 \end{bmatrix} \tag{2.28}$$

$$\text{Flujos de potencia (MW, MVar)} = \begin{bmatrix} 116{,}841700 & -15{,}814173 & -116{,}561640 & -2{,}709788 \\ 86{,}561640 & -17{,}290212 & -86{,}330443 & -10{,}660226 \\ 33{,}830443 & -24{,}339774 & -33{,}781927 & -13{,}203004 \\ 11{,}281927 & -1{,}796996 & -11{,}273285 & -26{,}347773 \\ -78{,}726715 & -33{,}652227 & 79{,}114051 & -22{,}308850 \\ 94{,}305408 & -30{,}990583 & -94{,}114051 & 12{,}308850 \end{bmatrix} \tag{2.29}$$

2.2.7. Comparación de Resultados

Comparar los resultados obtenidos con el algoritmo propio y las funciones de la librería, analizando la cantidad de iteraciones necesarias en cada caso.

Iteración 1

Tabla 2.1: Resultados de la Iteración 1

Barra	**Θ (grados)**	**V (pu)**
1	0.0000	1.0000
2	4.5187	1.0000
1a	4.5188	1.0025
2a	4.5747	1.0046
3a	4.5519	1.0064
2b	4.6055	1.0058
1b	4.5098	1.0024

Iteración 2

Tabla 2.2: Resultados de la Iteración 2

Barra	**Θ (grados)**	**V (pu)**
1	0.0000	1.0000
2	3.7472	1.0000
1a	3.7443	1.0018
2a	3.7836	1.0034
3a	3.6875	1.0074
2b	3.8358	1.0071
1b	3.7353	1.0026

Iteración 3

Tabla 2.3: Resultados de la Iteración 3

Barra	**Θ (grados)**	**V (pu)**
1	0.0000	1.0000
2	10.9842	1.0000
1a	11.1175	0.9999
2a	11.2666	1.0015
3a	11.1206	1.0095
2b	11.3722	1.0108
1b	11.0817	1.0039

Resultados pandapower

Tabla 2.4: Resultados pandapower

Barra	**Θ (grados)**	**V (pu)**
1	0.0000	1.0000
2	-14.7251	1.0000
1a	-14.9488	0.9978
2a	-15.1957	0.9953
3a	-15.3350	0.9943
2b	-15.3448	0.9933
1b	-14.9247	0.9988

Resultados finales del método de Newton-Raphson

Tabla 2.5: Resultados finales del método de Newton-Raphson

Barra	**Θ (grados)**	**V (pu)**
1	0.0000	1.0000
2	10.9842	1.0000
1a	11.1175	0.9999
2a	11.2666	1.0015
3a	11.1206	1.0095
2b	11.3722	1.0108
1b	11.0817	1.0039

2.2.8. Función de Error y Escenarios de Tolerancia

Propuesta de una función de error, comparación de escenarios con distintas tolerancias y comentarios sobre los resultados obtenidos.

Resultados de Referencia (Pandapower)

Tabla 2.6: Resultados de referencia (Pandapower)

Bus	Theta (grados)	V (pu)
bus 1	0.0000	1.0000
bus 2	-14.7251	1.0000
bus 1a	-14.9488	0.9978
bus 2a	-15.1957	0.9953
bus 3a	-15.3350	0.9943
bus 2b	-15.3448	0.9933
bus 1b	-14.9247	0.9988

Resultados Finales del Método de Newton-Raphson

Tabla 2.7: Resultados finales del método de Newton-Raphson

Bus	Theta (grados)	V (pu)
bus 1	0.0000	1.0000
bus 2	10.9842	1.0000
bus 1a	11.1175	1.0000
bus 2a	11.2666	1.0015
bus 3a	11.1206	1.0095
bus 2b	11.3722	1.0108
bus 1b	11.0817	1.0039

Resultados con Diferentes Tolerancias

Tolerancia 0.0001

Tabla 2.8: Resultados con tolerancia 0.0001 (Converged in 20 iterations)

Bus	Theta (grados)	V (pu)
bus 1	0.0000e+00	1.0000e+00
bus 2	5.2136e+27	1.0000e+00
bus 1a	3.2785e+04	-3.7782e+25
bus 2a	-4.2767e+05	-3.4003e+25
bus 3a	-3.7524e+04	-1.0017e+26
bus 2b	-4.4646e+04	-8.4780e+25
bus 1b	-4.0292e+04	-5.2256e+25

Tolerancia 1e-06

Tabla 2.9: Resultados con tolerancia 1e-06 (Converged in 20 iterations)

Bus	Theta (grados)	V (pu)
bus 1	0.0000e+00	1.0000e+00
bus 2	1.1442e+53	1.0000e+00
bus 1a	3.2084e+04	2.0916e+51
bus 2a	-5.8407e+05	6.1737e+50
bus 3a	-3.6274e+04	3.2139e+50
bus 2b	1.2588e+05	-1.2376e+51
bus 1b	-2.4294e+05	1.9005e+51

Tolerancia 1e-08

Tabla 2.10: Resultados con tolerancia 1e-08 (Converged in 20 iterations)

Bus	Theta (grados)	V (pu)
bus 1	0.0000e+00	1.0000e+00
bus 2	1.1987e+83	1.0000e+00
bus 1a	2.4275e+04	1.9640e+81
bus 2a	-5.6011e+05	-5.3851e+80
bus 3a	-2.3307e+04	-2.2626e+81
bus 2b	-2.0210e+06	-2.7090e+80
bus 1b	-9.8578e+05	1.1225e+81

3. Consideraciones para Implementación de Modelos en Python

3.1. Importación de Librerías

Listar las librerías utilizadas, incluyendo `pandapower`:

```
import pandapower as pp
import pandapower.plotting as plot
import matplotlib.pyplot as plt
import numpy as np
```

3.2. Modelo de Líneas y Transformadores

Descripción del modelo de líneas `N2XS(FL)2Y 1x185 RM/35 64/110 kV` y del transformador `100 MVA 220/110 kV`.

Características de las Líneas

Tipo de Conductor	c_nf_per_km	r_ohm_per_km	x_ohm_per_km
N2XS(FL)2Y 1x185 RM/35 64/110 kV	125.00	0.0990	0.156

Características del Transformador

Tipo de Transformador	i0_percent	pfe_kw	vkr_percent	sn_mva	vn_lv_kv
100 MVA 220/110 kV	0.0600	55.00	0.2600	100.00	110.0

3.3. Configuración de la Barra Slack

Para la implementación de la barra 1 se utilizó lo siguiente:

```
pp.create_ext_grid(net, b1, vm_pu=1.0, name="external grid", va_degree=0)
```

3.4. Porcentajes de las Cargas

Descripción de los porcentajes de las cargas en el sistema 2, tanto de P como de Q.

Cargas:

	1a	2a	3a	2b	1b
P	150×0.20	150×0.35	150×0.15	150×0.6	150×0.10
Q	100×0.20	100×0.35	100×0.15	100×0.6	100×0.10

4. Conclusiones

Para el análisis del flujo de potencia el método de Newton-Raphson ha permitido demostrar ser una técnica poderosa, eficiente y, salvando las proporciones con respecto a las herramientas entregadas por pandapower, precisa para la determinación de las variables pernitentes asociadas al problematico flujo de potencia de un sistema eléctrico. Sumado a lo mencionado, se comprendió cómo las variaciones en la topología del sistema y los cambios en los consumos afectan las métricas de las barras.

Referente al método iterativo de Newton-Raphson, se destaca la obtención del Jacobiano, dado que, para su obtención es necesario analizar bien el sistemado a trabajar, principalmente porque involucra todas las variables e incognitas de las barras. Por ese motivo, los algoritmos desarrolados en Python fueron sumamente importantes para la implementación correcta del método.

El proyecto demuestra la aplicación del método de Newton-Raphson en el análisis de flujo de potencia. Se observó que los resultados de tensiones y ángulos concuerdan correctamente hasta la iteración 3. Más allá de esta iteración, los resultados no convergen de manera adecuada, indicando posibles problemas en la estabilidad del método para este sistema específico.