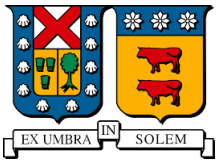


Informe Tarea 2 SEP A.

[G6A]

José Gutierréz

201904160-6

Vicente Marín

201804086-k

Profesor: Guillermo Huerta
Ayudantes: Anais Vattier, Pablo Vera
Asignatura: Análisis de Sistemas de Potencia

julio, 2024

Casa Central

Tabla de Contenidos

mathrsfs

1 Contexto

El sistema eléctrico en el mundo moderno muchas veces se pasa por alto, aunque este sea de vital importancia para el correcto funcionamiento de prácticamente todos los servicios, matrices productivas y comodidades con las que se cuenta en una sociedad moderna como la actual. Es por esto que el correcto funcionamiento del mismo se vuelve una tarea de gran relevancia y complejidad, siendo necesarias las herramientas de simulación y control para poder mantener a todas sus partes funcionando sin desperfectos.
Muchas veces, (dada la complejidad de los SEP) no se cuenta con métodos matemáticos directos para resolver las ecuaciones que describen un sistema cualquiera, por lo que, se acude al uso de métodos iterativos que entregan soluciones con un margen de error muy pequeño con respecto la realidad.

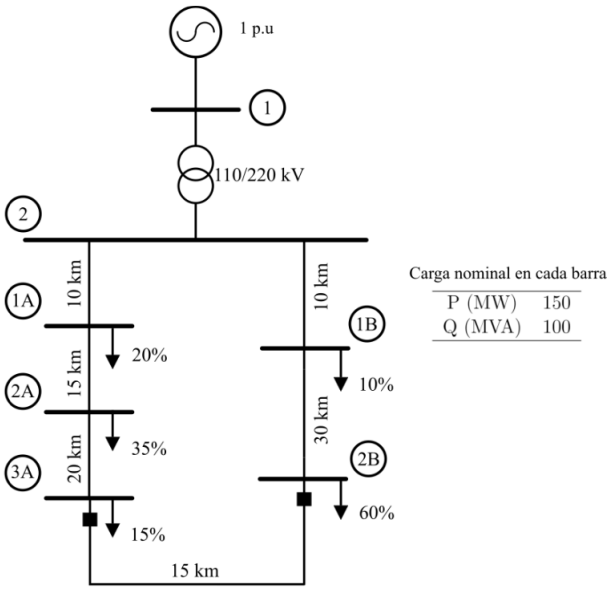


Figura 1: Figura del SEP a analizar

2 Resumen ejecutivo.

2.1 Arquitectura.

Como la naturaleza del proyecto presentado en la tarea no es sencilla, resulta útil contar con un esquema general de como se aborda el mismo, en la forma de un esquema o diagrama de flujo que permite abordar la tarea paso por paso y contar con un orden en el proceso de desarrollo.

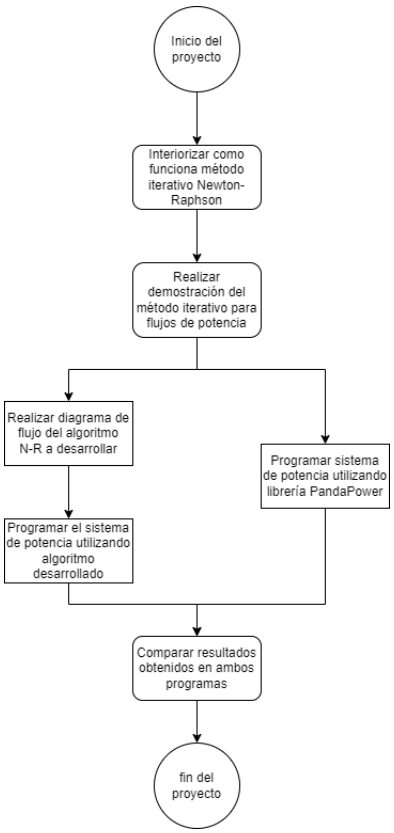


Figura 2: diagrama de flujo del proyecto

Introducción

El presente informe aborda la simulación y análisis de un Sistema Eléctrico de Potencia (SEP) utilizando herramientas de software gratuito, con énfasis en el método iterativo de Newton-Raphson y su implementación en Python mediante la librería PandaPower y un algoritmo desarrollado por los estudiantes.

Objetivo

El objetivo principal de esta tarea es demostrar la capacidad del método iterativo de Newton-Raphson para la resolución de los SEP, también contrastar distintos tipos de ejecución de este algoritmo mediante un código desarrollado en Python y la librería PandaPower.

Metodología

Plataforma de Desarrollo:

- **Lenguaje de Programación:** Python.
- **Librería:** PandaPower.
- **Entorno de Desarrollo:** Visual Studio Code y Jupyter Notebook.
- **Control de Versiones:** GitHub.

Implementación del Algoritmo de Newton-Raphson:

- Desarrollar un algoritmo en Python para resolver el SEP mediante el método iterativo de Newton-Raphson.
- Definir las ecuaciones de balance de potencias y su derivada (Jacobiano) para la iteración del algoritmo.

Simulación en PandaPower:

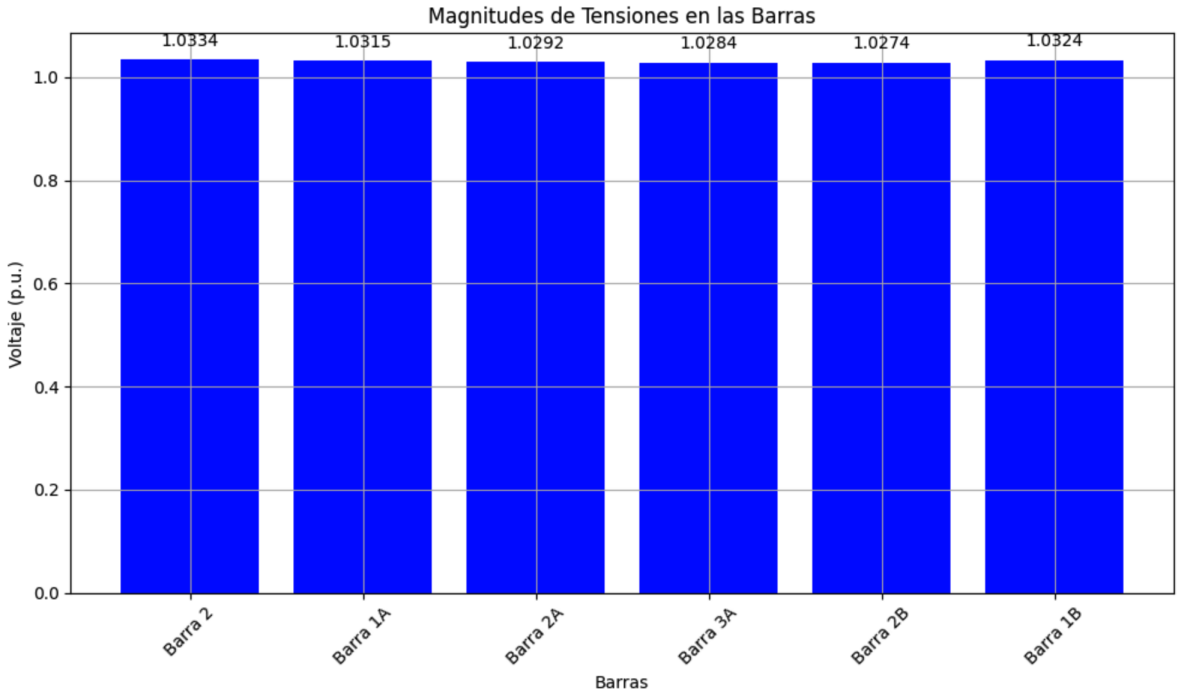
- Configurar y ejecutar la simulación del SEP en PandaPower, especificando las condiciones iniciales, parámetros del sistema y criterios de convergencia.

Resultados

Resultados de PandaPower:

- Se presentan los valores de tensiones y ángulos de fase en cada barra del SEP obtenidos mediante PandaPower.

N° Barra	Tensión [pu]	Ángulo [°]
2	1.0334	-14.28
1A	1.0315	-14.49
2A	1.0292	-14.73
3A	1.0284	-14.86
2B	1.0274	14.87
1B	1.0324	-14.47



s

Figura 3: Tensiones de las barras del sistema.

Resultados del Algoritmo de Newton-Raphson:

- Se muestran los resultados del flujo de carga calculados por el algoritmo de Newton-Raphson implementado en Python.

N° Barra	Tensión [pu]	Ángulo [°]
2	1.0022	0.56
1A	1.0169	-11.29
2A	1.0374	-13.18
3A	1.0175	-10.35
2B	1.0218	-15.26
1B	1.0231	-10.02

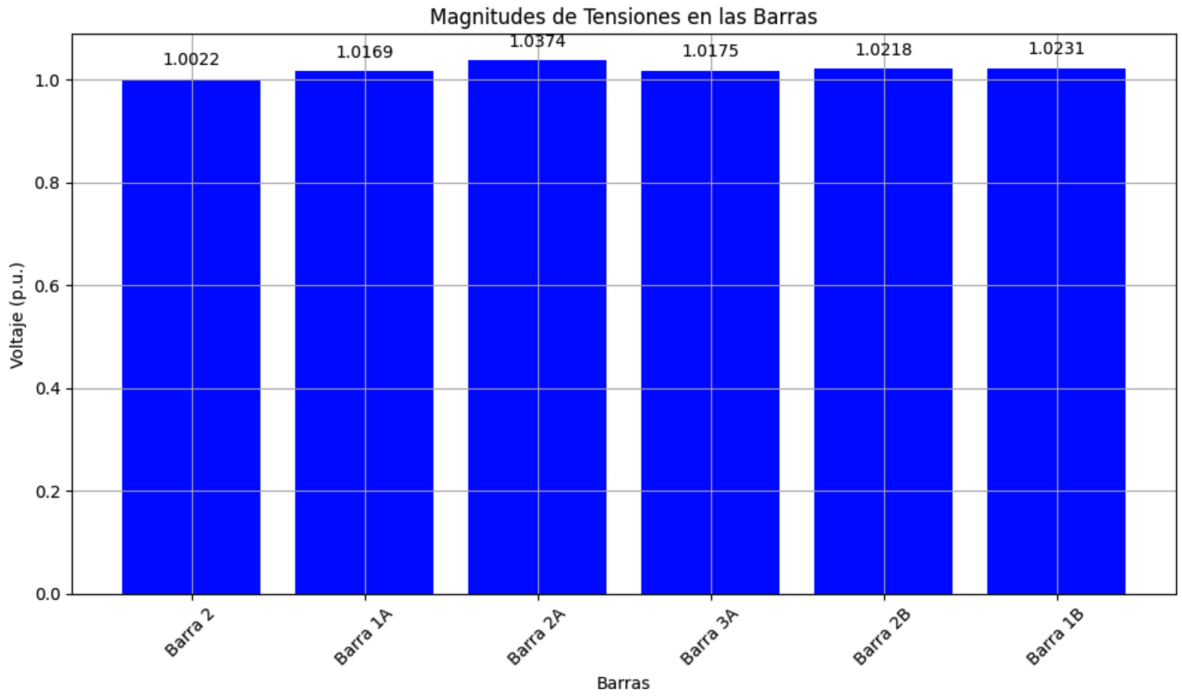


Figura 4: Tensiones de las barras del sistema.

Conclusiones

- **Eficiencia y Precisión:** Ambos métodos demostraron ser efectivos para resolver el problema de flujo de carga en SEPs, con diferencias mínimas en los resultados.
- **Ventajas de PandaPower:** La simplicidad y rapidez de configuración y ejecución hacen de PandaPower una herramienta valiosa para simulaciones rápidas.
- **Contribuciones del Algoritmo Propio:** El desarrollo del algoritmo de Newton-Raphson permite una mayor comprensión del proceso iterativo y ofrece flexibilidad para ajustes específicos.

3 Plataforma de desarrollo y programación del proyecto.

Como no se cuenta con un software de simulación de pago se decide utilizar Python, lenguaje de programación gratuito que cuenta con la librería PandaPower, la cual cuanta con todas las herramientas necesarias para simular los SEP en cuestión.
Ya que el proyecto se lleva a cabo en grupo, es importante contar con una plataforma de desarrollo, editor de código y entorno de programación que hagan compatible el trabajo en equipo, reduciendo la probabilidad de que los integrantes se estorben mutuamente o induzcan errores en el código de otros. Dado esto se decide utilizar GitHub como plataforma de desarrollo, ya que cuenta con mecanismos de control de versiones y almacenamiento en un repositorio en la nube, lo que hace más sencilla la comunicación entre las partes sin necesidad de estar enviando mensajes directos cada vez que se modifica algo. En cuanto al editor de código y entorno de programación, se escoge Visual Studio Code y JupyterNotebook respectivamente, los cuales pueden ser sincronizados con el repositorio en Github, por lo que traer los archivos desde la nube a nuestro entorno de programación se vuelve una tarea sencilla, sin necesidad de descargar o subir manualmente los archivos cada vez que efectuemos un cambio. También permite comparar los cambios en los archivos/códigos a través de distintas versiones, así todos los integrantes pueden ver y estar al tanto de como cambia el código a lo largo del tiempo.

3.1 PandaPower

Es importante comprender como funciona el programa a utilizar, así se puede evaluar a priori si las simulaciones desarrolladas pueden ser un fiel representante del SEP que se busca simular. Ahora, como breve introducción al análisis se definen dos tipos de modelos para representar un SEP:
Librería PandaPower, trabaja con un modelo de parámetros concentrados, en concreto, con el modelo pi línea media ya revisado en el curso. El método iterativo estándar que ocupa PandaPower corresponde al método Newton-Raphson, por lo que basta con definir la cantidad máxima de iteraciones y la tolerancia para ocupar esta herramienta.

4 Demostración método iterativo de Newton-Raphson

El método iterativo de Newton-Raphson se usa para encontrar las raíces de una función no lineal mediante la siguiente relación matemática:

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

Donde se busca que finalmente f_x sea cercano a 0, superando un umbral de tolerancia ϵ tal que:

$$x_{n+1} - x_n < \epsilon \tag{2}$$

Ahora, en el caso particular de los sistemas de potencia, la variable x_n corresponde al valor de las tensiones con su respectivo ángulo de fase en cada barra, mientras que $f(x_n)$ es el balance de potencias dentro del sistema, el cual relaciona las variables de tensión de las distintas barras del sistema por medio de la matriz de admitancia del sistema y las potencias conocidas del mismo. Es por esto que se busca un $f(x_n) = 0$, porque no puede existir un desbalance de potencias en el sistema.

En los SEP existen múltiples barras, lo que significa que hay una multitud de variables (tensiones y ángulos) distintas a calcular, por lo que resulta conveniente llevar el algoritmo a una forma matricial que tendrá la siguiente forma:

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - \frac{\boldsymbol{F(x_n)}}{\boldsymbol{F'(x_n)}} \tag{3}$$

Donde x_n se define con la siguiente estructura:

$$\boldsymbol{x_n} = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \\ V_1 \\ \vdots \\ V_n \end{bmatrix}$$

Y $F(x_n)$ se define como una matriz que representa el balance de potencias de cada barra, comparando los valores conocidos (de inyección o consumo) con los calculados.

$$\boldsymbol{F(x_n)} = \begin{bmatrix} \Delta P_1 \\ \vdots \\ \Delta P_n \\ \Delta Q_1 \\ \vdots \\ \Delta Q_n \end{bmatrix}$$

$$\Delta P_i = P_{inyectada} - P_{calculada}$$
$$\Delta Q_i = Q_{inyectada} - Q_{calculada}$$

Para conocer el valor de las potencias calculadas se sabe que:

$$S_B = V_B I_B^* \tag{4}$$
$$I_B = Y_B * V_B \tag{5}$$

Considerando $\theta_{ij} = \delta_i - \delta_j$:

$$I_i = \sum_{j=1}^{n} (G_{ij} + jB_{ij}\sin(\theta_{ij})) \cdot (\cos(\theta_{ij}) + j\sin(\theta_{ij})) \cdot V_j \tag{6}$$

Reemplazando corriente en ecuación de potencia y separando en parte real e imaginaria:

$$P_i = V_i \sum_{j=1}^{n} (G_{ii}\cos\theta_i j + B_{ij}\sin\theta_{ij}) \cdot V_j \tag{7}$$
$$Q_i = V_i \sum_{j=1}^{n} (G_{ii}\cos\theta_i j - B_{ij}\sin\theta_{ij}) \cdot V_j \tag{8}$$

Por lo que la matriz F_x está compuesta por la diferencia entre una constante y una expresión dependiente de las variables de estado. Ahora, la derivada de dicha expresión será una matriz de dimensiones $F^{'}(x_n) = [N_{F(x_n)} x N_{x_n}]$, dado que se tiene que derivar en cada componente de $F(x)$ en x_n. Resultando:

$$\boldsymbol{F^{'}(x_n)} = \begin{bmatrix} \frac{\partial \Delta P_1}{\partial \delta_1} & \cdots & \frac{\partial \Delta P_1}{\partial \delta_n} & \frac{\partial \Delta P_1}{\partial V_1} & \cdots & \frac{\partial \Delta P_1}{\partial V_n} \\ \vdots & & & & & \\ \frac{\partial \Delta P_n}{\partial \delta_1} & \cdots & \frac{\partial \Delta P_n}{\partial \delta_n} & \frac{\partial \Delta P_n}{\partial V_1} & \cdots & \frac{\partial \Delta P_n}{\partial V_n} \\ \frac{\partial \Delta Q_1}{\partial \delta_1} & \cdots & \frac{\partial \Delta Q_1}{\partial \delta_n} & \frac{\partial \Delta Q_1}{\partial V_1} & \cdots & \frac{\partial \Delta Q_1}{\partial V_n} \\ \vdots & & & & & \\ \frac{\partial \Delta Q_n}{\partial \delta_1} & \cdots & \frac{\partial \Delta Q_n}{\partial \delta_n} & \frac{\partial \Delta Q_n}{\partial V_1} & \cdots & \frac{\partial \Delta Q_n}{\partial V_n} \end{bmatrix} = \boldsymbol{J(x_n)} \tag{9}$$

A esta expresión es a la que denominamos Jacobiano. Se logran identificar 4 submatrices dentro del jacobiano, que se puede expresar como:

$$\boldsymbol{J(x_n)} = \begin{bmatrix} \frac{\partial \Delta P}{\partial \delta} & \frac{\partial \Delta P}{\partial V} \\ \frac{\partial \Delta Q}{\partial \delta} & \frac{\partial \Delta Q}{\partial V} \end{bmatrix} \tag{10}$$

Se aprecia que de derivar ΔP y ΔQ solo quedan los términos P_i y Q_i, dado que el otro término es constante. Ahora, el resultado de derivar (7) y (8) para todos los ángulos y tensiones arroja el siguiente resultado:

$$\frac{\partial P_i}{\partial \theta_j} = -V_i(G_{ii}\sin\theta_i j - B_{ij}\cos\theta_{ij})V_j \tag{11}$$
$$\frac{\partial Q_i}{\partial \theta_j} = V_i(G_{ii}\cos\theta_i j + B_{ij}\sin\theta_{ij})V_j \tag{12}$$
$$\frac{\partial P_i}{\partial V_j} = -V_i(G_{ii}\cos\theta_i j + B_{ij}\sin\theta_{ij}) \tag{13}$$
$$\frac{\partial Q_i}{\partial V_j} = -V_i(G_{ii}\sin\theta_i j - B_{ij}\cos\theta_{ij}) \tag{14}$$

Notar que la sumatoria desaparece debido a la derivación respecto a solo una variable dependiendo de j, por lo que todos los otros términos se hacen 0.

Cuando se analizan las diagonales las 4 submatrices, la sumatoria no desaparece, pues δ_i se encuentra en todos los θ_{ij}, resultando para cada derivada parcial presentada anteriormente:

$$\frac{\partial P_i}{\partial \theta_i} = -V_i \sum_{j=1}^{n} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) - V_i^2 B_{ii} \tag{15}$$

$$\frac{\partial Q_i}{\partial \theta_i} = -V_i \sum_{j=1}^{n} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) + V_i^2 G_{ii} \tag{16}$$

$$\frac{\partial P_i}{\partial V_i} = 2V_i G_{ii} + \sum_{j=1}^{n} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) \tag{17}$$

$$\frac{\partial Q_i}{\partial V_i} = -2V_i B_{ii} - \sum_{j=1}^{n} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) \tag{18}$$

En donde, cada término que no se encuentra dentro de la sumatoria es el resultado del caso particular j=i. Si separamos el caso particular de j=i y $j \neq i$ se obtienen las ecuaciones finales simplificadas:
Para $i \neq j$:

$$\frac{\partial P_i}{\partial \theta_i} = -V_i V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) \tag{19}$$

$$\frac{\partial Q_i}{\partial \theta_i} = V_i V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) \tag{20}$$

$$\frac{\partial P_i}{\partial V_i} = -V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) \tag{21}$$

$$\frac{\partial Q_i}{\partial V_i} = -V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) \tag{22}$$

Para i=j:

$$\frac{\partial P_i}{\partial \theta_i} = Q_i + B_{ii} V_i^2 \tag{23}$$

$$\frac{\partial Q_i}{\partial \theta_i} = -P_i + G_{ii} V_i^2 \tag{24}$$

$$\frac{\partial P_i}{\partial V_i} = -\frac{P_i}{V_i} - G_{ii} V_i \tag{25}$$

$$\frac{\partial Q_i}{\partial V_i} = -\frac{Q_i}{V_i} + B_{ii} V_i \tag{26}$$

Por lo que ahora, podemos pasar el algoritmo de su expresión unidimensional, a una expresión multidimensional de la forma:

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - \frac{\boldsymbol{F(x_n)}}{\boldsymbol{F'(x_n)}} = \boldsymbol{x_n} - \boldsymbol{J(x_n)^{-1} F(x_n)} \tag{27}$$

Finalmente, se encuentra demostrada la expresión matemática del algoritmo N-R aplicado a un problema de flujos de potencia.

4.1 Algoritmo para resolver un SEP mediante Newton-Raphson

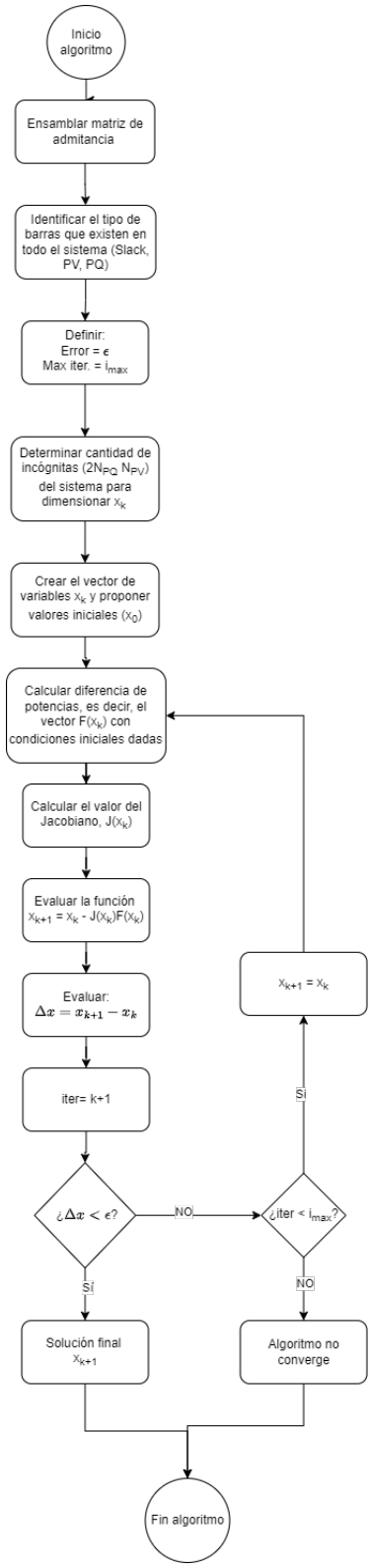


Figura 5: Diagrama de flujo del algoritmo

5 Creación del algoritmo N-R mediante Python

Se requería simular el sistema de potencia por medio del método iterativo Newton-Raphson. Cabe destacar, que el sistema descrito por la figura, se había simulado antes con la librería PandaPower. Parte de esa simulación, fue usada para hallar la matriz de admitancia del sistema, para así dar inicio al algoritmo solicitado en la tarea. Se adjunta parte del código:

```
pp.runpp(net)
Ybus = net._ppc['internal']['Ybus']

# Convertir Ybus a un DataFrame de pandas para una mejor visualizacio n
Ybus_df = pd.DataFrame(data=Ybus.todense())
print("Matriz Ybus (7x7):")
print(Ybus_df)
```

Listing 1: Hallar matriz de admitencia desde el problema resuelto por PandaPower.

Posteriormente se empiezan a definir los elementos que tiene el método de Newton Raphson, ya sea, la función, la matriz jacobiana, vector de estado, etc. Un ejemplo en código sería:

```
def F(x):
    V = x[:len(x)//2]
    theta = x[len(x)//2:]

    P = np.zeros(len(V))
    Q = np.zeros(len(V))

    for i in range(len(V)):
        for j in range(len(V)):
            G = Ybus[i, j].real
            B = Ybus[i, j].imag
            P[i] += V[i] * V[j] * (G * np.cos(theta[i] - theta[j]) + B * np.sin(theta
    [i] - theta[j]))
            Q[i] += V[i] * V[j] * (G * np.sin(theta[i] - theta[j]) - B * np.cos(theta
    [i] - theta[j]))

    P_spec = net.res_bus.p_mw.values / net.sn_mva
    Q_spec = net.res_bus.q_mvar.values / net.sn_mva

    # Excluir barra slack de las ecuaciones
    P[0] = 0
    Q[0] = 0
    P_spec[0] = 0
    Q_spec[0] = 0

    return np.concatenate([P - P_spec, Q - Q_spec])
```

Listing 2: Función de Newton-Raphson

6 Resolución del problema

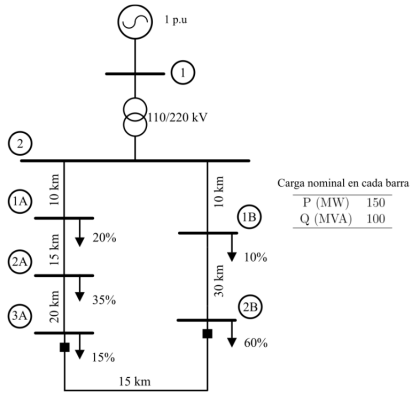


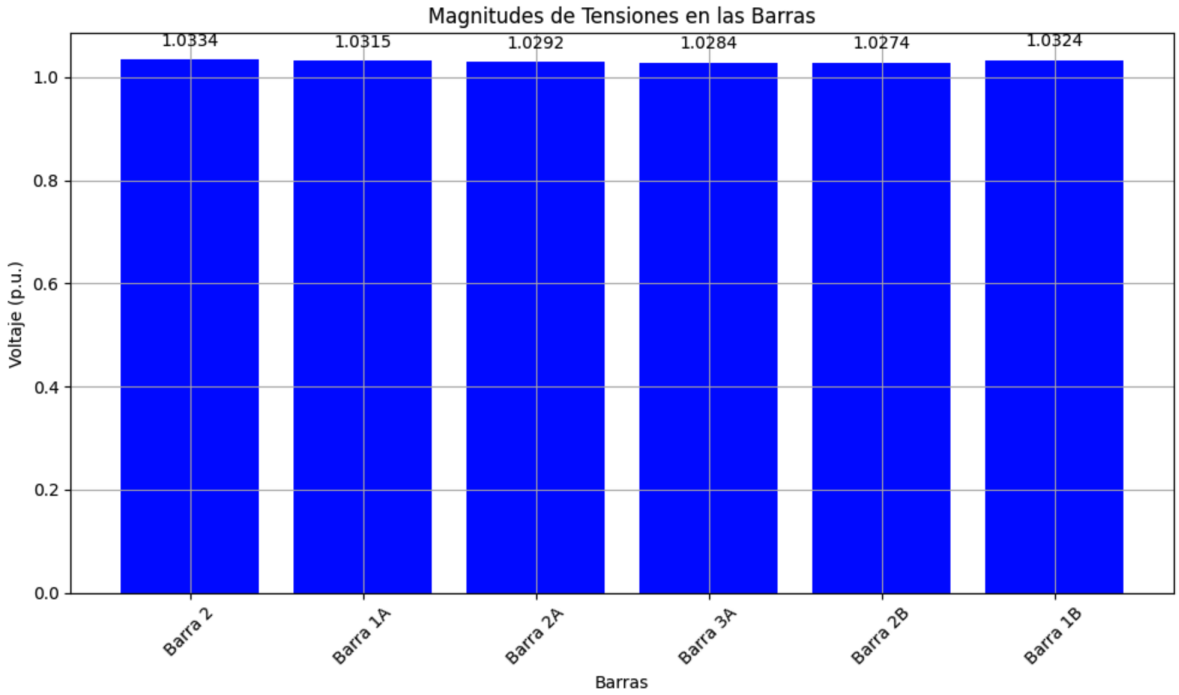
Figura 6: Figura correspondiente SEP.

Como bien se sabe, se requería resolver el sistema descrito por la figura por método de Newton Raphson, para posteriormente, tener margen de comparación respecto a la tarea 1, en donde se resolvió por pandapower.

6.1 Resultados entregados por Pandapower

Se adjuntan los resultados correspondientes al resolver el flujo de potencia por medio de la librería PandaPower.

N° Barra	Tensión [pu]	Ángulo [°]
2	1.0334	-14.28
1A	1.0315	-14.49
2A	1.0292	-14.73
3A	1.0284	-14.86
2B	1.0274	14.87
1B	1.0324	-14.47



s

Figura 7: Tensiones de las barras del sistema.

6.2 Resultados entregados por Newton Raphson

Se adjuntan los resultados correspondientes al resolver el flujo de potencia por medio del método de Newton Raphson.

N° Barra	Tensión [pu]	Ángulo [°]
2	1.0022	0.56
1A	1.0169	-11.29
2A	1.0374	-13.18
3A	1.0175	-10.35
2B	1.0218	-15.26
1B	1.0231	-10.02

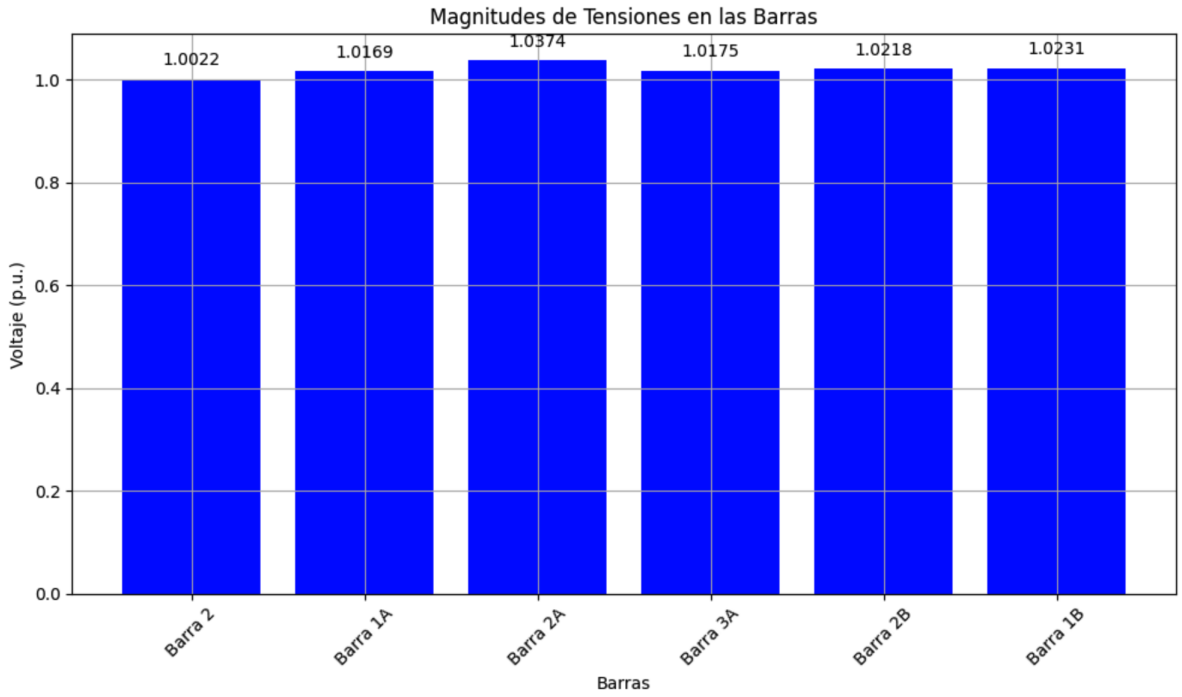


Figura 8: Tensiones de las barras del sistema.

6.3 Contraste de Resultados

es importante mencionar que para ambos métodos de resolución, se utilizó la misma matriz de admitancia, correspondiente a la calculada por la librería PandaPower, por lo que las diferencias entre los resultados obtenidos viene netamente del manejo interno e interpretación de los datos de cada algoritmo.

- Tensiones en las barras de carga: Se puede notar que las tensiones en las barras de carga son distintas en todas las barras. Por ejemplo: En la barra 1A la tensión es mayor en Pandapower que en Newton-Raphson, no obstante, la tensión es menor en la barra 2A utilizando PandaPower, respecto a Newton Raphson.

- Ángulo de las tensiones en las barras de carga: Respecto a los ángulos, se puede notar que según el método PandaPower, los ángulos deberían tener un comportamiento casi constante, los ángulos presentan muy poca variación. Mientras que por el método de Newton-Raphson, este comportamiento no es tan claro, se nota una variación entre un ángulo y otro, pero no dejan de superar las dos cifras, lo que indica un que las magnitudes de los ángulos también cercanas entre los ángulos de las barras.

Teniendo en cuenta los puntos presentados anteriormente, se plantean diferentes hipótesis respecto al comportamiento.

- Modelos Generalizados en PandaPower: Se podría decir que una desventaja de las librerías, es que, se tiene como una especie de caja negra, es decir, al ser un algoritmo sofisticado, no deja en evidencia cálculos internos del sistema. Es por ello que se piensa, que los modelos generales de transformadores y líneas, podrían no ser los más precisos para los cálculos del problema. No obstante, se calculó la matriz de admitancia con las librerías para poder implementarlo en

Newton-Raphson, por lo que es necesario hablar sobre el número de iteraciones presentes en el problema.

- N° de iteraciones de ambos modelos: Es claro que el método por la librería, solamente tiene una iteración, mientras que el método de Newton Raphson, tiene un máximo de 100 iteraciones lo que no asegura la convergencia del problema. Si bien Newton-Raphson tiene un análisis mucho más profundo al tener muchas iteraciones, podría ser más costoso computacionalmente. Dependerá del tamaño del sistema, para este caso, fue evidente notar que Newton-Raphson se demoró más tiempo en hacer el cálculo de las variables. Es por ello, que se concluye que el método de Newton-Raphson fue más preciso para la resolución del problema, pero eso no significa que sea el mejor.

7 Conclusiones

En la actividad, se pudieron comprar dos herramientas poderosas para resolver un Sistema Eléctrico de Potencia. No obstante, dependerá del contexto resolver cuál es mejor que otro, dado que, ambos tienen sus ventajas y desventajas, las cuales se presentan a continuación:

7.1 PandaPower

Ventajas:

- Facilidad de Uso: Proporciona funciones predefinidas para la creación y análisis de redes eléctricas, lo que reduce significativamente el tiempo de desarrollo.
- Optimización: Utiliza algoritmos optimizados que mejoran la eficiencia computacional.
- Funcionalidades Avanzadas: Soporta análisis avanzados como análisis de contingencias, optimización y estudios de estabilidad.
- Documentación y Comunidad: Amplia documentación y una comunidad activa que puede ayudar a resolver problemas y dudas.

Desventajas:

- Sobrecarga de Abstracción: La capa adicional de abstracción puede introducir una sobrecarga computacional.
- Dependencia de Software: Requiere instalación y configuración de múltiples bibliotecas y dependencias.
- Flexibilidad Limitada: Menor control sobre los detalles del algoritmo, lo que puede ser una limitación para investigaciones específicas o implementaciones personalizadas.

7.2 Newton Raphson

Ventajas:

- Control Total: Ofrece un control total sobre cada paso del algoritmo, permitiendo optimizaciones específicas.
- Menor Sobrecarga: Puede ser más eficiente en términos de uso de recursos si se implementa adecuadamente.
- Aprendizaje Profundo: Implementarlo desde cero proporciona un entendimiento profundo del proceso y los problemas asociados.

Desventajas:

- Complejidad de Implementación: Requiere un entendimiento profundo de la teoría y puede ser complejo de implementar y depurar.
- Optimización Limitada: Sin optimizaciones avanzadas, puede ser menos eficiente que las implementaciones optimizadas en bibliotecas especializadas.
- escalabilidad: Para sistemas muy grandes, puede no ser tan eficiente ni rápido como las soluciones optimizadas disponibles en bibliotecas dedicadas.

7.3 Contraste

1. Tiempo de Desarrollo

- PandaPower: Significativamente menor debido a la disponibilidad de funciones predefinidas y documentación extensiva.
- Newton-Raphson: Mayor tiempo de desarrollo, ya que requiere implementación desde cero, incluyendo la formulación de las ecuaciones y la derivación del Jacobiano.

2. Eficiencia Computacional

- PandaPower: Generalmente más eficiente para usuarios debido a las optimizaciones internas. La sobrecarga adicional por la abstracción es compensada por estas optimizaciones.
- Newton-Raphson: La eficiencia depende de la implementación. Sin optimizaciones adecuadas, puede ser menos eficiente.

3. Flexibilidad y Control

- PandaPower: Ofrece menor flexibilidad en cuanto a la modificación de algoritmos internos. Es ideal para análisis estándar y avanzados sin necesidad de ajustes en los algoritmos base.
- Newton-Raphson: Ofrece mayor flexibilidad y control, permitiendo adaptaciones y optimizaciones específicas según las necesidades del estudio.

4. Facilidad de Uso

- PandaPower: Muy fácil de usar para ingenieros y analistas, con una interfaz intuitiva y soporte para múltiples funcionalidades avanzadas.
- Newton-Raphson: Requiere un mayor nivel de conocimiento técnico y matemático, así como habilidades de programación.