Métodos Numéricos (0408)

Trabajo Práctico N°2: Matriz de Admitancia y Resolución de Sistemas Lineales

Integrantes:

- Basso Joaquin.
- Fernandez Matias Nicolas.
- Firmapaz Javier.

Profesores:

- Magnago Fernando.

Introducción:

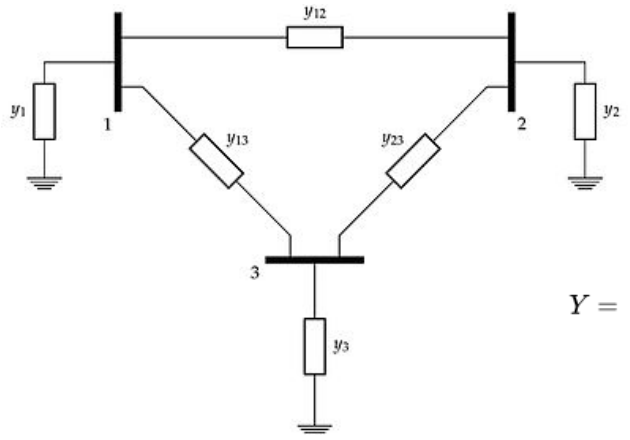
En este trabajo se presenta el concepto de la matriz de admitancia, su aplicación en sistemas eléctricos. Además, se crea un programa que recibe datos de una red eléctrica para formar la matriz de admitancia y resolver un sistema lineal.

Desarrollo:

La matriz de admitancia es una herramienta de análisis de redes que relaciona las inyecciones de corriente a una barra con los voltajes de barra. Esta es una matriz cuadrada, simétrica y por lo general rala, de dimensiones NxN, donde N es la cantidad de nodos del sistema. La matriz Y es uno de los requisitos de datos necesarios para formular un estudio de flujo de potencia. Para construir la matriz de admitancia, se procede como indican las siguientes ecuaciones:

$$Y_{ij} = \begin{cases} y_i + \sum_{\substack{k=1,2,\ldots,N \\ k \neq i}} y_{ik}, & \text{if} \quad i = j \\ -y_{ij}, & \text{if} \quad i \neq j \end{cases} \qquad Y = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1n} \\ Y_{21} & Y_{22} & \cdots & Y_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ Y_{n1} & Y_{n2} & \cdots & Y_{nn} \end{bmatrix}$$

El elemento Yii que se encuentra en la diagonal principal, es la suma de todas las admitancias que conectan al nodo i, los demás elementos serán el negativo de la admitancia de línea que corresponde a su índice en la matriz. Como ejemplo se ilustra un sistema de tres nodos:



La matriz asociada asociada a esta configuración es:

$$Y = \begin{pmatrix} y_1 + y_{12} + y_{13} & -y_{12} & -y_{13} \\ -y_{12} & y_2 + y_{12} + y_{23} & -y_{23} \\ -y_{13} & -y_{23} & y_3 + y_{13} + y_{23} \end{pmatrix}$$

Se procede a presentar el código escrito en Python que permite construir la matriz de admitancia a partir de una hoja de datos.

Se trabaja con el módulo numpy para simplificar la tarea de parsear los archivos csv, ya que este módulo tiene el código necesario. Se ingresan los datos de la red.

```
import numpy as np

#Carga de archivos csv
matrixL = np.matrix(np.genfromtxt('datos_de_linea.csv', delimiter=';'))
matrixN = np.matrix(np.genfromtxt('datos_de_nodos.csv', delimiter=';'))
```

Se declara la variable que define la cantidad de decimales, y la segunda variable sirve para almacenar el número de bus.

```
outputAccuracy = 5 #Precision de decimales
busNumer = int(np.amax(matrixL)) #Carga el numero de nodos
```

En el siguiente segmento de código crea variables para la capacitancia parásita entre nodos y la admitancia de línea. El formato de estas variables es JSON.

```
capPar = {}
nAdmitance = {}
for i in range(1, busNumer +1):
    j=np.where(matrixL==(i))
    capVal = 0;
    nodeAdmitance = complex(0, 0)

    for k in j[0]:
        capVal += matrixL[k,4]
        nodeAdmitance += 1/complex(matrixL[k,2], matrixL[k,3])

    capPar[str(i)] = complex(0, capVal)
    nAdmitance[str(i)] = nodeAdmitance
```

En este bloque se crea la variable de admitancia entre nodos

```
admitance = {}
for i in range(matrixL.shape[0]):
    key = str(min(int(matrixL[i,0]), int(matrixL[i,1]))) + str(max(int(matrixL[i,0]), int(matrixL[i,1])))
    admitance[key] = 1/complex(matrixL[i,2], matrixL[i,3])
```

En este apartado se trabaja con la segunda matriz para la capacitancia shunt

```
shuntCap = {}
for i in range(matrixN.shape[0]):
    shuntCap[str(int(matrixN[i,0]))] = complex(0, matrixN[i,1])
```

Con los datos de admitancias ya listos se crea la matriz de admitancia, la cual se va almacenando en un array de valores complejos, y despues se agrega otro

```
#Creacion de la matriz
ybus = []
for i in range(1, busNumer+1):
    xbus = []
    for j in range(1, busNumer+1):
        if i==j:
          val = 0;

          key = str(i)
          if key in nAdmitance:
            val = nAdmitance[key]

          if key in capPar:
            val += capPar[key]

          if key in shuntCap:
            val += shuntCap[key]
          xbus.append(round(val.real, outputAccuracy) + round(val.imag, outputAccuracy) * 1j)
        else:
          key = str(min(i, j)) + str(max(i, j))
          if key in admitance:
            adm = -admitance[key]
            xbus.append(round(adm.real, outputAccuracy) + round(adm.imag, outputAccuracy) * 1j)
          else:
            xbus.append(0)
    ybus.append(xbus)
```

Con la matriz ya construida, se convierte el array a un string para luego formatear los datos y adjuntarlos a un archivo csv

```
#Formatea la matriz al estandar de csv
formattedMatrix = str(ybus)
formattedMatrix = formattedMatrix.replace("], [", "\n")
formattedMatrix = formattedMatrix.replace("[[", "")
formattedMatrix = formattedMatrix.replace("]]", "")
formattedMatrix = formattedMatrix.replace(",", ";")

#Escribe el archivo con el resultado
fw = open('matrix.csv','w')
fw.writelines(formattedMatrix)
fw.close()
```

El resultado se presenta en forma de tabla para facilitar la lectura:

(6.02503-19.44977j)	(-4.99913+15.26309j)	0	0	(-1.0259+4.23498j)	0	0	0	0	0	0	0	0	0
(-4.99913+15.26309j)	(9.52132-30.26802j)	(-1.13502+4.78186j)	(-1.68603+5.11584j)	(-1.70114+5.19393j)	0	0	0	0	0	0	0	0	0
0	(-1.13502+4.78186j)	(3.12099-9.81468j)	(-1.98598+5.06882j)	0	0	0	0	0	0	0	0	0	0
0	(-1.68603+5.11584j)	(-1.98598+5.06882j)	(10.51299-38.29483j)	(-6.84098+21.57855j)	0	4.78194j	0	1.79798j	0	0	0	0	0
(-1.0259+4.23498j)	(-1.70114+5.19393j)	0	(-6.84098+21.57855j)	(9.56802-34.9301j)	3.96794j	0	0	0	0	0	0	0	0
0	0	0	0	3.96794j	(6.57992-17.34073j)	0	0	0	0	(-1.95503+4.09407j)	(-1.52597+3.17596j)	(-3.09893+6.10276j)	0
0	0	0	4.78194j	0	0	-19.54901j	5.67698j	9.09008j	0	0	0	0	0
0	0	0	0	0	0	5.67698j	-5.67698j	0	0	0	0	0	0
0	0	0	1.79798j	0	0	9.09008j	0	(5.32606-24.09251	(-3.90205+10.36539j)	0	0	0	(-1.42401+3.02905j)
0	0	0	0	0	0	0	0	(-3.90205+10.3653	(5.78293-14.76834j)	(-1.88088+4.40294j)	0	0	0
0	0	0	0	0	(-1.95503+4.09407j)	0	0	0	(-1.88088+4.40294j)	(3.83591-8.49702j)	0	0	0
0	0	0	0	0	(-1.52597+3.17596j)	0	0	0	0	0	(4.01499-5.42794j)	(-2.48902+2.25197j)	0
0	0	0	0	0	(-3.09893+6.10276j)	0	0	0	0	0	(-2.48902+2.25197j)	(6.72495-10.66969j)	(-1.13699+2.31496j)
0	0	0	0	0	0	0	0	(-1.42401+3.02905	0	0	0	(-1.13699+2.31496j)	(2.561-5.34401j)

Teniendo la matriz de admitancia se procede a utilizarla para resolver un sistema de ecuaciones lineales de la forma YV=I implementado en Python:

$$\begin{pmatrix} Y_{11} & Y_{12} & \cdots & Y_{1N} \\ Y_{21} & Y_{22} & \cdots & Y_{2N} \\ \vdots & \ddots & \cdots & \vdots \\ Y_{NN} & Y_{N2} & \cdots & Y_{NN} \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_N \end{pmatrix} = \begin{pmatrix} I_1 \\ I_2 \\ \vdots \\ I_N \end{pmatrix}$$

Trabajando con el formato de arrays de numpy, se importa la matriz de admitancia y se toma sus dimensiones:

```
#Importo la matriz de admitancia
Y = np.genfromtxt('matrix.csv', delimiter=';', dtype = "complex_")
n = len(Y)
```

Como datos iniciales, se crea un vector I de valores complejos aleatorios y se propone un vector solución V de ceros.

```
#Creo un vector I que toma valores complejos aleatorios entre 0 y 1
Ix = np.array((np.random.rand(n)), dtype = "complex_")
Iy = np.array(1j*(np.random.rand(n)), dtype = "complex_")
I = np.sum([Ix, Iy], axis=0, dtype = "complex_")
print(I)
#Propongo un vector V de ceros como solución
V = np.zeros(n, dtype = "complex_")
print(V)
```

Con estos datos de entrada se itera la solución 15 veces en la función seidel que realiza el método iterativo de Gauss-Seidel, para lograr una buena aproximación de la solución del sistema:

```
#Realizo 15 iteraciones de aproximacion
for i in range(0, 15):
    V = seidel(Y, V, I)
    #Imprime la solución aproximada cada iteracion
    print(V)
```

El algoritmo de la función que aplica el método se muestra a continuación:

```
def seidel(Y, V ,I):
    #Encuentro la longitud de 'Y'
    n = len(Y)
    #Itero n veces para calcular V1, V2, ... ,Vn
    for j in range(0, n):
        #Almaceno I[j]
        d = I[j]

        #Y calculo V1i, V2i, ... ,Vni
        for i in range(0, n):
            if(j != i):
                d-=Y[j][i] * V[i]
        #Actualizo el valor de solucion
        V[j] = d / Y[j][j]
    #Devuelvo el nuevo valor
    return V
```

Su procedimiento se resume con la siguiente fórmula de sustitución progresiva:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

La matriz de admitancia nunca posee elementos en la diagonal iguales a cero, por lo que nunca se va a dividir por cero en la anterior fórmula y siempre se llega a una convergencia. A continuación se muestra un ejemplo:

Dado el vector I:

```
[0.78591571+0.81661436j 0.95815517+0.37404691j 0.20942123+0.23789476j
 0.45920384+0.08546339j 0.46917666+0.3396896j  0.82710364+0.97098495j
 0.04164101+0.56020384j 0.3265075 +0.1113166j  0.15522537+0.73134282j
 0.23504224+0.42199741j 0.24970612+0.98859179j 0.49878104+0.29462607j
 0.28992412+0.85959181j 0.00348913+0.84180304j]
```

La solución encontrada con 15 iteraciones es:

```
[-0.46532639+0.84338397j -0.46937231+0.84915798j -0.48414595+0.83676006j
 -0.51646778+0.83202061j -0.53245179+0.87376735j -0.81922278+1.02600088j
 -0.6936289 +0.80453433j -0.71323732+0.86204863j -0.78976078+0.85041011j
 -0.83328825+0.91036145j -0.91327983+1.03481354j -0.82383243+1.11068526j
 -0.89315207+1.09081728j -0.96361454+1.01652259j]
```

Conclusión:

Los códigos presentados durante este informe, a pesar de que cumplen de manera correcta su objetivo, no son aplicables a sistemas reales en producción de gran escala por su ineficiencia en términos de operaciones. Se podrían mejorar implementando un uso de bajo nivel para optimizar el uso del CPU, o se podría usar la GPU para un mejor rendimiento en el caso de las operaciones aritméticas/lógicas, módulos como el presentado numpy logran hacer. Con respecto al algoritmo de resolución del sistema lineal, se optó por utilizar el método Gauss-Seidel porque es más simple de implementar, desde un punto de vista del programa, se entiende más simple para cualquier persona que tenga que trabajar o hacerle mantenimiento al código, posee una mayor rapidez de convergencia comparado con el método de Jacobi.

El método de Gauss-Seidel se podría mejorar conociendo que la matriz de admitancia es rala, negando operaciones con los ceros de la matriz, otra forma de mejorar este método es aplicando un promedio ponderado de los resultados de las iteraciones anteriores y actuales, donde se ve afectado por un factor ponderado que acelera la convergencia al amortiguar las oscilaciones a la solución. Este factor se hace notorio en sistemas que se resuelven de manera repetida como el caso de la matriz de admitancia en un sistema eléctrico de potencia.

El método de Gauss-Seidel no garantiza la convergencia si no es una matriz diagonal dominante.