DAVID BAKER

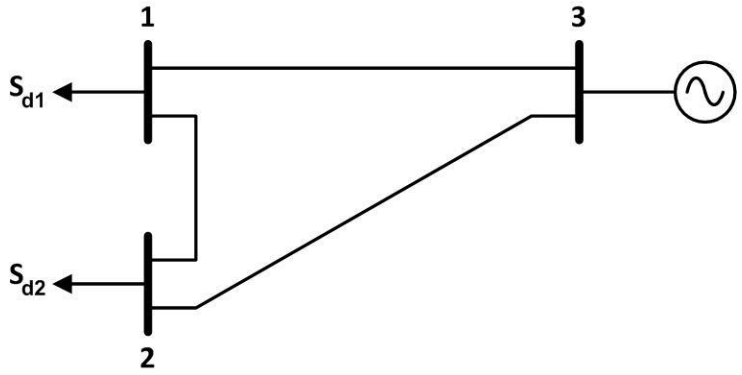
EENG 581: Power System Operation and Management

HW1: Power Flow

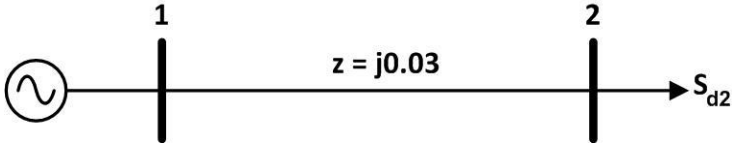
Learning Objectives:

- Develop the admittance matrix for a power transmission network.
- Formulate the power flow problem using various techniques such as Gauss/Gauss-Seidel and Newton Raphson.
- Write a code to iteratively solve the power flow problem.

1. Use Jacobi (Gauss) and Gauss-Seidel methods with flat start to find the voltages at buses 1 and 2 in the power system shown below. On a 50 MVA, 13.2kV line voltage base, power injections at the buses are assumed to be $S_1 = -0.4 + j0.2$ and $S_2 = -0.3 + j0.3$, and the line impedances are $z_{12} = 0.01 + j0.01$, $z_{13} = 0.02 + j0.02$, and $z_{23} = 0.03 + j0.03$ (all in per-unit). Bus 3 is the slack bus and maintains its voltage at $1\angle 0$ p.u. Ignore the sunt capacitances of lines. Calculate voltages assuming the error tolerance of the voltage magnitude is 10^{-6}.



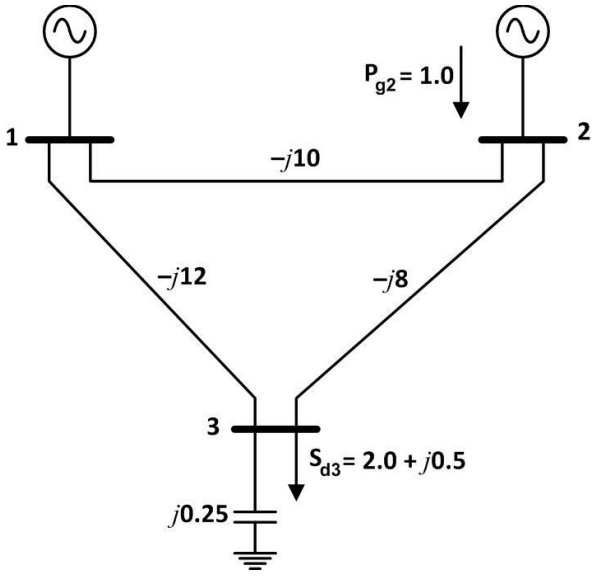
2. Apply the Newton-Raphson method to find the voltage at bus 2, assuming the load at the bus is $0.5 + j0.3$. Bus 1 is the slack bus and maintains the voltage at $1.0\angle 0$ p.u. All the values given in the diagram are in per-unit. Perform three iterations and report the results.



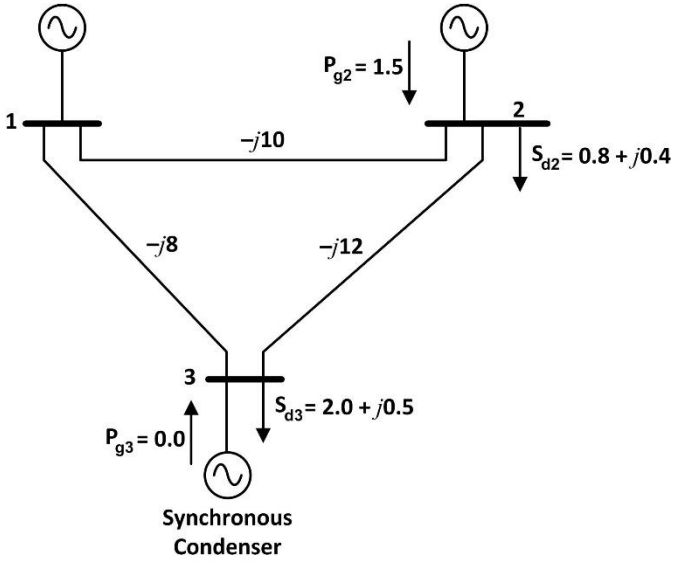
Provide a table in the form:

Iteration	Angle for bus 2 (in radians) Magnitude for bus 1 (in per unit)
1	
2	
3	

3. Consider the three bus electric power system shown below. Bus 1 is the slack bus, bus 2 is a PV bus, and bus 3 is a PQ bus. Voltages at buses 1 and 2 are maintained at $1.0\angle 0$ and 0.98 p.u, respectively. Explicitly write the iterative equations for the load flow problem using the fast-decoupled power flow technique. Perform three iterations of the algorithm.



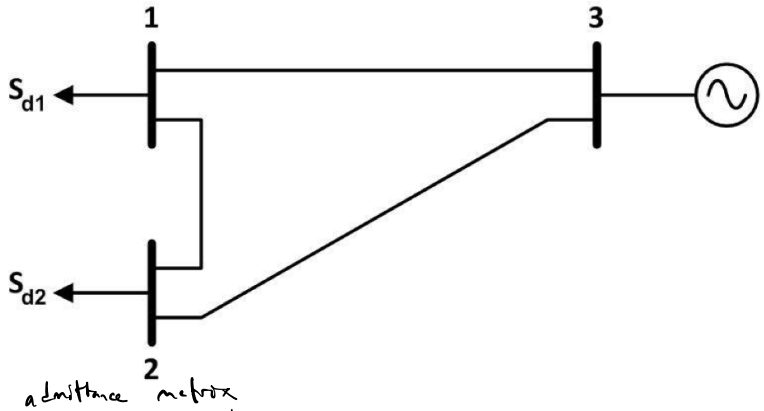
4. Consider the 3-bus power system illustrated below (all admittances are in per-unit). Buses 1 and 2 are equipped with generating units which control their corresponding voltages at $1.02\angle 0$ and 1.01, respectively. At bus 3 there is a synchronous condenser that generates reactive power to control the voltage magnitude and regulate it at 1.0 per-unit. Hence, you can consider this bus as a PV bus with zero active power injection. (*a*) formulate the load flow problem using the Newton-Raphson method and perform 2 iterations, then calculate the reactive power of the synchronous condenser, (*b*) repeat using fast-decoupled method, and compare the results. Consider an error mismatch of 10^{-3} for powers.



Clearly show the following answers:

- A table listing the state variables for 2 iterations using Newton-Raphson method
- A table listing the state variables for 2 iterations using Fast-Decoupled method

1. Use Jacobi (Gauss) and Gauss-Seidel methods with flat start to find the voltages at buses 1 and 2 in the power system shown below. On a 50 MVA, 13.2kV line voltage base, power injections at the buses are assumed to be $S_1 = -0.4 + j0.2$ and $S_2 = -0.3 + j0.3$, and the line impedances are $z_{12} = 0.01 + j0.01$, $z_{13} = 0.02 + j0.02$, and $z_{23} = 0.03 + j0.03$ (all in per-unit). Bus 3 is the slack bus and maintains its voltage at $1\angle 0$ p.u. Ignore the sunt capacitances of lines. Calculate voltages assuming the error tolerance of the voltage magnitude is 10^{-6}.



First let's find the admittance matrix

$$y_{12} = \frac{1}{0.01 + j0.01}\left(\frac{0.01 - j0.01}{0.01 - j0.01}\right) = \frac{0.01 - j0.01}{.0001 + 0.0001} = 50 - j50$$

$$y_{13} = \frac{1}{0.02 + j0.02} = 25 - j25$$

$$y_{23} = \frac{1}{0.03 + j0.03} = 16.67 - j16.67$$

$$Y = \begin{bmatrix} 75 - j75 & -50 + j50 & -25 + j25 \\ -50 + j50 & 66.67 - j66.67 & -16.67 + j16.67 \\ -25 + j25 & -16.67 + j16.67 & 41.67 - j41.67 \end{bmatrix}$$

Gauss Solution

$$\tilde{V}_k^{\nu+1} = \frac{1}{Y_{kk}} \times \left(\frac{S_k^*}{(\tilde{V}_k^*)^\nu} - \sum_{\substack{j=1 \\ j \neq k}}^{N} Y_{kj} \cdot \tilde{V}_j^\nu \right)$$

$$\begin{bmatrix} V \\ \delta \end{bmatrix} =$$

```
[(0.9976425361127977, -0.013778470876628188),
 (0.9983603853561321, -0.014746399065177274)]
```

(I realize the convention is $\begin{bmatrix} \delta \\ V \end{bmatrix}$)

Gauss-Seidel Solution

$$\tilde{V}_k^{\nu+1} = \frac{1}{Y_{kk}} \times \left(\frac{S_k^*}{(\tilde{V}_k^*)^\nu} - \sum_{j=1}^{k-1} Y_{kj} \cdot \tilde{V}_j^{\nu+1} - \sum_{j=k+1}^{N} Y_{kj} \cdot \tilde{V}_j^\nu \right)$$

V_1 δ_1

```
(0.9976294692328347, -0.01393812329009965)
(0.9983280454368093, -0.014961074673454616)
```

V_2 δ_2

Gauss

```python
import numpy as np

y_12 = 1 / (0.01 + 0.01j)
y_13 = 1 / (0.02 + 0.02j)
y_23 = 1 / (0.03 + 0.03j)


# polar form to rectangular
def P2R(A, phi):
    return A * (np.cos(phi) + np.sin(phi) * 1j)


# rectangular to polar
def R2P(x):
    return abs(x), np.angle(x)


Y = np.matrix(
    [
        [y_12 + y_13, -y_12, -y_13],
        [-y_12, y_12 + y_23, -y_23],
        [-y_13, -y_23, y_13 + y_23],
    ]
)

# complex powers at buses 1 and 2
S = [-0.4 + 0.2j, -0.3 + 0.3j]


def check_symmetric(a, rtol=1e-05, atol=1e-08):
    return np.allclose(a, a.T, rtol=rtol, atol=atol)


if not check_symmetric(Y):
    raise Exception("not symmetric, something is wrong with your admittance matrix")


# buses 1 and 2
pq_buses = [0, 1]

all_buses = [0, 1, 2]


# voltage and angle, flat start
V = [[(1, 0), (1, 0)]]

i = 0


TOLERANCE = 10e-6


def has_converged(voltage_list):
    cur = voltage_list[-1]
    prev = (
        voltage_list[-2] if len(voltage_list) > 1 else [(0, 0), (0, 0)]
    )  # dummy value if starting out

    for k in pq_buses:
        if np.abs(cur[k][0] - prev[k][0]) > TOLERANCE:
            return False

    return True


while not has_converged(V) and i < 1000:
    i += 1
    new_v = [(0, 0), (0, 0)]  # initialize new voltages with dummy values
    for k in pq_buses:
        sum_term = 0
        for j in all_buses:
            if k != j:
                # slack bus when j == 2
                V_j = P2R(1, 0) if j == 2 else P2R(*V[i - 1][j])
                sum_term += Y[k, j] * V_j

        new_v[k] = (1 / Y[k, k]) * ((np.conjugate(S[k]) / P2R(*V[i - 1][k])) -
sum_term)
    new_v_polar = [R2P(vv) for vv in new_v]
    V.append(new_v_polar)


print(f"took {i} iterations")
V[-1]

```

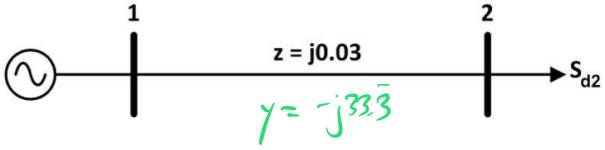
```
took 15 iterations

[(0.9976313146636302, -0.014018557035269214),
 (0.9983353371644513, -0.014997561323291783)]
```

Gauss-Seidel

```python
import numpy as np

y_12 = 1 / (0.01 + 0.01j)
y_13 = 1 / (0.02 + 0.02j)
y_23 = 1 / (0.03 + 0.03j)


# polar form to rectangular
def P2R(A, phi):
    return A * (np.cos(phi) + np.sin(phi) * 1j)


# rectangular to polar
def R2P(x):
    return abs(x), np.angle(x)


Y = np.matrix(
    [
        [y_12 + y_13, -y_12, -y_13],
        [-y_12, y_12 + y_23, -y_23],
        [-y_13, -y_23, y_13 + y_23],
    ]
)

# complex powers at buses 1 and 2
S = [-0.4 + 0.2j, -0.3 + 0.3j]


def check_symmetric(a, rtol=1e-05, atol=1e-08):
    return np.allclose(a, a.T, rtol=rtol, atol=atol)


if not check_symmetric(Y):
    raise Exception("not symmetric, something is wrong with your admittance matrix")


# buses 1 and 2
pq_buses = [0, 1]

all_buses = [0, 1, 2]


# voltage and angle, flat start
# IMPORTANT! I changed how I am storing the voltage list compared to Gauss method.
# Before my V looked like this:  V = [[(1, 0), (1, 0)]]
V = ([(1, 0)], [(1, 0)])

i = 0


TOLERANCE = 10e-6


def has_converged(voltage_list):
    for k in pq_buses:
        cur = voltage_list[k][-1]
        prev = (
            voltage_list[k][-2] if len(voltage_list[k]) > 1 else (0, 0)
        )  # dummy value if starting out

        print("here", cur[0], prev[0])
        print("outside the tolerance:", np.abs(cur[0] - prev[0]) > TOLERANCE)
        if np.abs(cur[0] - prev[0]) > TOLERANCE:
            return False

    return True


while not has_converged(V) and i < 1000:
    i += 1
    # new_v = [(0, 0), (0, 0)]  # initialize new voltages with dummy values
    for k in pq_buses:
        sum_term = 0
        for j in all_buses:
            if k != j:
                # slack bus when j == 2
                V_j_polar = ((1, 0)) if j == 2 else V[j][-1]
                V_j = P2R(*V_j_polar)
                sum_term += Y[k, j] * V_j

        new_v_rect = (1 / Y[k, k]) * ((np.conjugate(S[k]) / P2R(*V[k][-1])) -
sum_term)new_v_polar = R2P(new_v_rect)
        V[k].append(new_v_polar)
        # print("down here", k, new_v[k])


print(f"took {i} iterations")
display(V[0][-1], V[1][-1])
```

```
took 7 iterations

(0.9976294692328347, -0.01393812329009965)

(0.9983280454368093, -0.014961074673454616)
```

2. Apply the Newton-Raphson method to find the voltage at bus 2, assuming the load at the bus is 0.5 + *j*0.3. Bus 1 is the slack bus and maintains the voltage at 1.0∠0 p.u. All the values given in the diagram are in per-unit. Perform three iterations and report the results.



$y = -j33.\bar{3}$

Provide a table in the form:

Iteration	Angle for bus 2 (in radians) Magnitude for bus 1 (in per unit)
1	
2	
3	

$$Y = \begin{bmatrix} -j33.\bar{3} & j33.\bar{3} \\ j33.\bar{3} & -j33.\bar{3} \end{bmatrix} = G + jB$$

Only concerned w/ bus 2.

$$P_k = f_{P,k}(V,\delta) = V_k \sum_{j=1}^{n} V_j \cdot (G_{kj}\cos\delta_{kj} + B_{kj}\sin\delta_{kj})$$

$$Q_k = f_{Q,k}(V,\delta) = V_k \sum_{j=1}^{n} V_j \cdot (G_{kj}\sin\delta_{kj} - B_{kj}\cos\delta_{kj})$$

$$\begin{bmatrix} \delta^{\nu+1} \\ V^{\nu+1} \end{bmatrix} = \begin{bmatrix} \delta^{\nu} \\ V^{\nu} \end{bmatrix} + \begin{pmatrix} \frac{\partial f_P}{\partial \delta} & \frac{\partial f_P}{\partial V} \\ \frac{\partial f_Q}{\partial \delta} & \frac{\partial f_Q}{\partial V} \end{pmatrix}^{-1}_{\substack{\delta^{\nu} \\ V^{\nu}}} \cdot \begin{bmatrix} P - f_P(\delta^{\nu}, V^{\nu}) \\ Q - f_Q(\delta^{\nu}, V^{\nu}) \end{bmatrix}$$

$\sin\delta_{22} = 0$, $\cos\delta_{22} = 1$, $\delta_1 = 0$, $V_1 = 0$

$P_2 = -0.5$, $Q_2 = -0.3$

$$f_{P2}(V,\delta) = V_2 V_1 B_{21}\sin\delta_{21} = 33.\bar{3}\,V_2\sin\delta_2$$

$$f_{Q2}(V,\delta) = -V_2 V_1 B_{21}\cos\delta_{21} - V_2^2 B_{22} = -33.\bar{3}\,V_2\cos\delta_2 + 33.\bar{3}\,V_2^2$$

Jacobian

$$J = \begin{bmatrix} 33.\bar{3}V_2\cos\delta_2 & 33.\bar{3}\sin\delta_2 \\ 33.\bar{3}V_2\sin\delta_2 & -33.\bar{3}\cos\delta_2 + 66.\bar{6}V_2 \end{bmatrix}$$

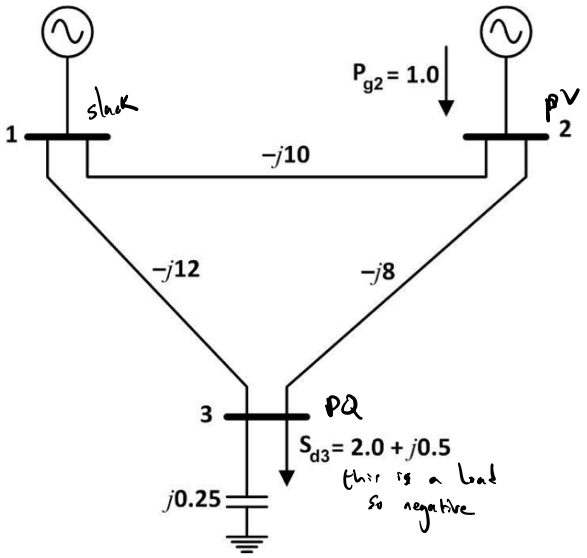
then

$$\begin{bmatrix} \delta_2^{\nu+1} \\ V_2^{\nu+1} \end{bmatrix} = \begin{bmatrix} \delta_2^{\nu} \\ V_2^{\nu} \end{bmatrix} + J^{-1}\Big|_{\substack{\delta^{\nu} \\ V^{\nu}}} \begin{bmatrix} -0.5 - 33.\bar{3}V_2\sin\delta_2 \\ -0.3 + 33.\bar{3}V_2\cos\delta_2 - 33.\bar{3}V_2^2 \end{bmatrix}$$

	iteration	angle for bus 2 [radians]	magnitude for bus 2 [pu]
0	1	-0.01500	0.991000
1	2	-0.01514	0.990798
2	3	-0.01514	0.990802

```python
import numpy as np
import pandas as pd


# X = [[delta, v]]
# flat start
X = [np.array([0, 1])]


B_21 = 33.33333333
B_22 = -33.33333333


def jacobian_inverse(x):
    delta = x[0]
    v = x[1]
    J = np.matrix(
        [
            [B_21 * v * np.cos(delta), B_21 * np.sin(delta)],
            [B_21 * v * np.sin(delta), B_22 * np.cos(delta) - 2 * B_22 *
v**2],  ]
    )

    return np.linalg.inv(J)


for i in [1, 2, 3]:
    delta_cur = X[-1][0]
    v_cur = X[-1][1]
    mismatch = np.array(
        [
            -0.5 - (B_21 * v_cur * np.sin(delta_cur)),
            -0.3 - (-B_21 * v_cur * np.cos(delta_cur) - B_22 * v_cur**2),
        ]
    )
    new_X = X[-1] + jacobian_inverse(X[-1]).dot(mismatch)

    X.append(np.ravel(new_X))


df = pd.DataFrame(
    {
        "iteration": [1, 2, 3],
        "angle for bus 2 [radians]": [xx[0] for xx in X[1:]],
        "magnitude for bus 2 [pu]": [xx[1] for xx in X[1:]],
    }
)

df
```

3. Consider the three bus electric power system shown below. Bus 1 is the slack bus, bus 2 is a PV bus, and bus 3 is a PQ bus. Voltages at buses 1 and 2 are maintained at $1.0\angle 0$ and 0.98 p.u, respectively. Explicitly write the iterative equations for the load flow problem using the fast-decoupled power flow technique. Perform three iterations of the algorithm.



$$Y = \begin{bmatrix} -j22 & j10 & j12 \\ j10 & -j18 & j8 \\ j12 & j8 & -j19.75 \end{bmatrix} \Rightarrow B = \begin{bmatrix} -22 & 10 & 12 \\ 10 & -18 & 8 \\ 12 & 8 & -19.75 \end{bmatrix}$$

$$\left[\delta^{\nu+1}\right] = \left[\delta^{\nu}\right] + B_P^{-1} \cdot \mathrm{diag}(V^{\nu})^{-1} \cdot \left[P - f_P(\delta^{\nu}, V^{\nu})\right]$$

$$\left[V^{\nu+1}\right] = \left[V^{\nu}\right] + B_Q^{-1} \cdot \mathrm{diag}(V^{\nu})^{-1} \cdot \left[Q - f_Q(\delta^{\nu}, V^{\nu})\right]$$

$$P_k = f_{P,k}(V,\delta) = V_k \sum_{j=1}^{n} V_j \cdot (G_{kj}\cos\delta_{kj} + B_{kj}\sin\delta_{kj})$$

$$Q_k = f_{Q,k}(V,\delta) = V_k \sum_{j=1}^{n} V_j \cdot (G_{kj}\sin\delta_{kj} - B_{kj}\cos\delta_{kj})$$

B_P = –Im[Y] when we take out the row/column of Y corresponding to the slack bus,
B_Q = –Im[Y] when we take out the rows/columns of Y corresponding to PV/slack buses.

$$f_{P2}(\delta, V) = V_2\left(V_1 B_{21}\sin\delta_{21} + V_3 B_{23}\sin\delta_{23}\right)$$
$$= 0.98\left(B_{21}\sin\delta_2 + V_3 B_{23}\sin\delta_{23}\right)$$

$$f_{Q2}(\delta, V) = V_2\left(-V_1 B_{21}\cos\delta_{21} - V_2 B_{22} - V_3 B_{23}\cos\delta_{23}\right)$$
$$= -0.98\left(B_{21}\delta_2 + 0.98 B_{22} + V_3 B_{23}\cos\delta_{23}\right)$$

$$f_{P3}(\delta, V) = V_3\left(B_{31}\sin\delta_2 + V_2 B_{32}\sin\delta_{32}\right)$$

$$f_{Q3}(\delta, V) = -V_3\left(B_{31}\cos\delta_3 + B_{32}V_2\cos\delta_{32} + B_{33}V_3\right)$$

$$B_P = \begin{bmatrix} 18 & -8 \\ -8 & 19.75 \end{bmatrix}$$

$$B_Q = \begin{bmatrix} 19.75 \end{bmatrix}$$

so we have

$$\begin{bmatrix} \delta_2 \\ \delta_3 \end{bmatrix}^{\nu+1} = \begin{bmatrix} \delta_2 \\ \delta_3 \end{bmatrix}^{\nu} + \begin{bmatrix} 18 & -8 \\ -8 & 19.75 \end{bmatrix}^{-1} \begin{bmatrix} 0.98 & 0 \\ 0 & V_3^{\nu} \end{bmatrix}^{-1} \begin{bmatrix} 1 - \left(0.98\left(B_{21}\sin\delta_2^{\nu} + V_3^{\nu} B_{23}\sin\delta_{23}^{\nu}\right)\right) \\ -2 - V_3^{\nu}\left(B_{31}\sin\delta_2^{\nu} + 0.98 B_{32}\sin\delta_{32}^{\nu}\right) \end{bmatrix}$$

we discard this and

$$\begin{bmatrix} V_2 \\ V_3 \end{bmatrix}^{\nu+1} = \begin{bmatrix} 0.98 \\ V_3 \end{bmatrix}^{\nu} + \begin{bmatrix} 19.75 \end{bmatrix}^{-1} \begin{bmatrix} 0.98 & 0 \\ 0 & V_3^{\nu} \end{bmatrix}^{-1} \begin{bmatrix} Q_2 + 0.98\left(B_{21}\delta_{21} + 0.98 B_{22} + V_3 B_{23}\cos\delta_{23}\right) \\ -0.5 + V_3\left(B_{31}\cos\delta_3 + B_{32}V_2\cos\delta_{32} + B_{33}V_3\right) \end{bmatrix}$$

Actually might be

$$\begin{bmatrix} V_3 \end{bmatrix}^{\nu+1} = \begin{bmatrix} V_3 \end{bmatrix}^{\nu} + \begin{bmatrix} 19.75 \end{bmatrix}^{-1} \begin{bmatrix} V_3^{\nu} \end{bmatrix}^{-1} \left[-0.5 + V_3^{\nu}\left(B_{31}\cos\delta_3^{\nu} + B_{32}V_2\cos\delta_{32}^{\nu} + B_{33}V_3^{\nu}\right)\right]$$

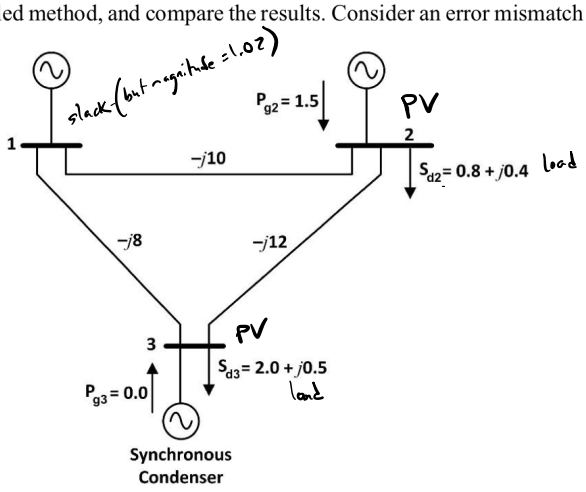
(code and answer on next page)

It took 14 iterations to converge (with tolerance 10e-6 so I ran more than 3 iterations

iteration	angle for bus 2 [radians]	angle for bus 3 [radians]	magnitude for bus 3[pu]
1	0.014247	-0.095495	0.979241
2	-0.021580	-0.178600	0.973548
3	-0.035969	-0.213260	0.964004
4	-0.041769	-0.228797	0.958309
5	-0.044441	-0.236369	0.955383
6	-0.045762	-0.240184	0.953875
7	-0.046428	-0.242129	0.953096
8	-0.046768	-0.243125	0.952694
9	-0.046942	-0.243638	0.952486
10	-0.047032	-0.243902	0.952379
11	-0.047078	-0.244038	0.952324
12	-0.047102	-0.244108	0.952295
13	-0.047114	-0.244145	0.952281
14	-0.047120	-0.244163	0.952273

```python
import numpy as np
import pandas as pd

TOLERANCE = 10e-6

np.linalg.inv(np.matrix([[18, -8], [-8, 19.75]]))
np.linalg.inv(np.matrix([[19.75]]))

B_mat = np.matrix([[-22, 10, 12], [10, -18, 8], [12, 8, -19.75]])
B_P = np.matrix([[18, -8], [-8, 19.75]])
B_Q = np.matrix([[19.75]])

B_P_inv = np.linalg.inv(B_P)
B_Q_inv = np.linalg.inv(B_Q)


# converts actual index to zero-based
def B(row, col):
    return B_mat[row - 1, col - 1]


V_2 = 0.98
P_2 = 1
P_3 = -2
Q_3 = -0.5


def has_converged(voltage_list):
    cur = Vs[-1]
    prev = Vs[-2] if len(Vs) > 1 else 0  # dummy value at start

    if np.abs(cur - prev) > TOLERANCE:
        return False

    return True


# flat start
# [[delta2, delta3], ...]
deltas = [np.array([0, 0])]
# [[V_3] ...]
Vs = [np.array([1])]


i = 0
while not has_converged(Vs) and i < 1000:
    i += 1
    delta_2 = deltas[-1][0]
    delta_3 = deltas[-1][1]
    v_3 = Vs[-1][0]

    # delta calculations
    matrix_A1 = B_P_inv.dot(np.linalg.inv(np.diag([V_2, v_3])))

    mismatch_P2 = P_2 - (
        V_2 * (B(2, 1) * np.sin(delta_2) + v_3 * B(2, 3) * np.sin(delta_2 -
delta_3))
    mismatch_P3 = P_3 - (
        v_3 * (B(3, 1) * np.sin(delta_2) + V_2 * B(3, 2) * np.sin(delta_3 -
delta_2))

    mismatch_P = np.matrix([[mismatch_P2], [mismatch_P3]])
    deltas_new = np.add(deltas[-1].reshape(2, 1), matrix_A1.dot(mismatch_P))

    # voltage calculations
    matrix_A2 = B_Q_inv.dot(np.linalg.inv(np.diag([v_3])))
    mismatch_Q3 = Q_3 - (
        -v_3
        * (
            B(3, 1) * np.cos(delta_3)
            + B(3, 2) * V_2 * np.cos(delta_3 - delta_2)
            + B(3, 3) * v_3
        )
    )

    mismatch_Q = np.matrix([[mismatch_Q3]])

    v_new = np.add(Vs[-1].reshape(1, 1), matrix_A2.dot(mismatch_Q))

    deltas.append(np.ravel(deltas_new))
    Vs.append(np.ravel(v_new))


df = pd.DataFrame(
    {
        "iteration": np.arange(1, i + 1),
        "angle for bus 2 [radians]": [xx[0] for xx in deltas[1:]],
        "angle for bus 3 [radians]": [xx[1] for xx in deltas[1:]],
        "magnitude for bus 3[pu]": [xx[0] for xx in Vs[1:]],
    }
)

df

```

4. Consider the 3-bus power system illustrated below (all admittances are in per-unit). Buses 1 and 2 are equipped with generating units which control their corresponding voltages at $1.02\angle 0$ and 1.01, respectively. At bus 3 there is a synchronous condenser that generates reactive power to control the voltage magnitude and regulate it at 1.0 per-unit. Hence, you can consider this bus as a PV bus with zero active power injection. (*a*) formulate the load flow problem using the Newton-Raphson method and perform 2 iterations, then calculate the reactive power of the synchronous condenser, (*b*) repeat using fast-decoupled method, and compare the results. Consider an error mismatch of 10^{-3} for powers.



Clearly show the following answers:

- A table listing the state variables for 2 iterations using Newton-Raphson method
- A table listing the state variables for 2 iterations using Fast-Decoupled method

$$B = \begin{bmatrix} -18 & 10 & 8 \\ 10 & -22 & 12 \\ 8 & 12 & -20 \end{bmatrix}$$

$$Y = jB$$

$$P_k = f_{P,k}(V,\delta) = V_k \sum_{j=1}^{n} V_j \cdot (G_{kj}\cos\delta_{kj} + B_{kj}\sin\delta_{kj})$$

$$Q_k = f_{Q,k}(V,\delta) = V_k \sum_{j=1}^{n} V_j \cdot (G_{kj}\sin\delta_{kj} - B_{kj}\cos\delta_{kj})$$

$$\begin{bmatrix} \delta^{\nu+1} \\ V^{\nu+1} \end{bmatrix} = \begin{bmatrix} \delta^{\nu} \\ V^{\nu} \end{bmatrix} + \begin{pmatrix} \frac{\partial f_P}{\partial \delta} & \frac{\partial f_P}{\partial V} \\ \frac{\partial f_Q}{\partial \delta} & \frac{\partial f_Q}{\partial V} \end{pmatrix}^{-1}_{\substack{\delta^{\nu} \\ V^{\nu}}} \cdot \begin{bmatrix} P - f_P(\delta^{\nu}, V^{\nu}) \\ Q - f_Q(\delta^{\nu}, V^{\nu}) \end{bmatrix}$$

Knowns: V_1, V_2, V_3 → all voltages known

unknowns: δ_2, δ_3

$V_1 = 1.02$

$V_2 = 1.01$

$V_3 = 1.0$

4 equations but only 2 unknowns can ignore f_{Q2}, f_{Q3}

$$f_{P2}(V,\delta) = V_2\left(V_1 B_{21}\sin\delta_2 + V_3 B_{23}\sin\delta_{23}\right)$$

$$f_{P3}(V,\delta) = V_3\left(V_1 B_{31}\sin\delta_3 + V_2 B_{32}\sin\delta_{32}\right)$$

$P_2 = 1.5 - 0.8 = 0.7$

$P_3 = 0 - 2 = -2$

Newton Raphson

$$J = \begin{bmatrix} \frac{\partial f_{P2}}{\partial \delta_2} & \frac{\partial f_{P2}}{\partial \delta_3} \\ \frac{\partial f_{P3}}{\partial \delta_2} & \frac{\partial f_{P3}}{\partial \delta_3} \end{bmatrix} = \begin{bmatrix} V_2(V_1 B_{21}\cos\delta_2 + V_3 B_{23}\cos\delta_{23}) & -V_2 V_3 B_{23}\cos\delta_{23} \\ -V_2 V_3 B_{32}\cos\delta_{32} & V_3(V_1 B_{31}\cos\delta_3 + V_2 B_{32}\cos\delta_{32}) \end{bmatrix}$$

so formulation is

$$\begin{bmatrix} \delta_2 \\ \delta_3 \end{bmatrix}^{\nu+1} = \begin{bmatrix} \delta_2 \\ \delta_3 \end{bmatrix}^{\nu} + J^{-1}\begin{bmatrix} P_2 - f_{P2} \\ P_3 - f_{P3} \end{bmatrix}$$

code to follow

iteration	angle for bus 2 [radians]	angle for bus 3 [radians]
1	-0.032629	-0.118120
2	-0.032688	-0.118328

Q for sync. condenser...

$$f_{Q3}(V,\delta) = -V_3\left(V_1 B_{13}\cos\delta_3 + V_2 B_{23}\cos\delta_{32} + V_3 B_{33}\right)$$

$$f_{Q3} = Q_{condenser} - 0.5$$

$$Q_{condenser} = f_{Q3} + 0.5 = 0.3125 \text{ pu}$$

Newton Raphson

```python
import numpy as np
import pandas as pd


B_mat = np.matrix([[-18, 10, 8], [10, -22, 12], [8, 12, -20]])


# converts actual index to zero-based
def B(row, col):
    return B_mat[row - 1, col - 1]


# known variables
V_1 = 1.02
V_2 = 1.01
V_3 = 1.0
P_2 = 0.7
P_3 = -2


def jacobian_inverse(x):
    delta_2 = x[0]
    delta_3 = x[1]

    eq1 = V_2 * (
        V_1 * B(2, 1) * np.cos(delta_2) + V_3 * B(2, 3) * np.cos(delta_2 - delta_3)
    )
    eq2 = -V_2 * V_3 * B(2, 3) * np.cos(delta_2 - delta_3)
    eq3 = -V_2 * V_3 * B(3, 2) * np.cos(delta_3 - delta_2)
    eq4 = V_3 * (
        V_1 * B(3, 1) * np.cos(delta_3) + V_2 * B(3, 2) * np.cos(delta_3 - delta_2)
    )

    J = np.matrix(
        [
            [
                eq1,
                eq2,
            ],
            [
                eq3,
                eq4,
            ],
        ]
    )

    return np.linalg.inv(J)


# flat start
# [[delta2, delta3], ... ]
deltas = [np.array([0, 0])]

POWER_TOLERANCE = 10e-3

i = 0
while True:
    i += 1
    d2 = deltas[-1][0]
    d3 = deltas[-1][1]

    mismatch_P2 = P_2 - (
        V_2 * (V_1 * B(2, 1) * np.sin(d2) + V_3 * B(2, 3) * np.sin(d2 - d3))
    )
    mismatch_P3 = P_3 - (
        V_3 * (V_1 * B(3, 1) * np.sin(d3) + V_2 * B(3, 2) * np.sin(d3 - d2))
    )
    mismatch_P = np.matrix([[mismatch_P2], [mismatch_P3]])

    d = deltas[-1].reshape(2, 1)
    new_deltas = np.add(
        d,
        jacobian_inverse(deltas[-1]).dot(mismatch_P),
    )

    deltas.append(np.ravel(new_deltas))

    # convergence condition
    if np.abs(mismatch_P2) < POWER_TOLERANCE and np.abs(mismatch_P3) <
POWER_TOLERANCE:
        break


print(f"took {i} iterations to converge")
df = pd.DataFrame(
    {
        "iteration": np.arange(1, i + 1),
        "angle for bus 2 [radians]": [xx[0] for xx in deltas[1:]],
        "angle for bus 3 [radians]": [xx[1] for xx in deltas[1:]],
    }
)


d2 = deltas[-1][0]
d3 = deltas[-1][1]
Q_sync_condenser = 0.5 + -V_3 * (
    V_1 * B(1, 3) * np.cos(d3) + V_2 * B(2, 3) * np.cos(d3 - d2) + V_3 * B(3, 3)
)

print("Q_sync_condenser:", Q_sync_condenser)

df
```

```
took 2 iterations to converge
Q_sync_condenser: 0.321478375938554
```

	iteration	angle for bus 2 [radians]	angle for bus 3 [radians]
0	1	-0.032629	-0.118120
1	2	-0.032688	-0.118328

Now for fast-decoupled

$$B = \begin{bmatrix} -18 & 10 & 8 \\ 10 & -22 & 12 \\ 8 & 12 & -20 \end{bmatrix}$$

$$B_P = \begin{bmatrix} 22 & -12 \\ -12 & 20 \end{bmatrix}$$

$$\left[\delta^{\nu+1}\right] = \left[\delta^{\nu}\right] + B_P^{-1} \cdot \mathrm{diag}(V^{\nu})^{-1} \cdot \left[P - f_P(\delta^{\nu}, V^{\nu})\right]$$

$$\left[V^{\nu+1}\right] = \left[V^{\nu}\right] + B_Q^{-1} \cdot \mathrm{diag}(V^{\nu})^{-1} \cdot \left[Q - f_Q(\delta^{\nu}, V^{\nu})\right]$$

We don't need/have B_Q because all buses are slack or PV

	iteration	angle for bus 2 [radians]	angle for bus 3 [radians]
0	1	-0.034252	-0.120551
1	2	-0.032654	-0.118294
2	3	-0.032689	-0.118329

This time it took 3 iterations to converge, but converged to same values.

$Q_{condenser}$ again $= 0.3215_{pu}$

```python
import numpy as np
import pandas as pd


B_mat = np.matrix([[-18, 10, 8], [10, -22, 12], [8, 12, -20]])
B_P = np.matrix([[22, -12], [-12, 20]])
B_P_inv = np.linalg.inv(B_P)


# converts actual index to zero-based
def B(row, col):
    return B_mat[row - 1, col - 1]


# known variables
V_1 = 1.02
V_2 = 1.01
V_3 = 1.0
P_2 = 0.7
P_3 = -2

# flat start
# [[delta2, delta3], ...]
deltas = [np.array([0, 0])]

POWER_TOLERANCE = 10e-3

i = 0
while True:
    i += 1
    d2 = deltas[-1][0]
    d3 = deltas[-1][1]

    matrix_A = B_P_inv.dot(np.linalg.inv(np.diag([V_2, V_3])))

    mismatch_P2 = P_2 - (
        V_2 * (V_1 * B(2, 1) * np.sin(d2) + V_3 * B(2, 3) * np.sin(d2 - d3))
    )
    mismatch_P3 = P_3 - (
        V_3 * (V_1 * B(3, 1) * np.sin(d3) + V_2 * B(3, 2) * np.sin(d3 - d2))
    )
    mismatch_P = np.matrix([[mismatch_P2], [mismatch_P3]])

    d = deltas[-1].reshape(2, 1)
    new_deltas = np.add(d, matrix_A.dot(mismatch_P))

    deltas.append(np.ravel(new_deltas))

    # convergence condition
    if np.abs(mismatch_P2) < POWER_TOLERANCE and np.abs(mismatch_P3) <
POWER_TOLERANCE:


print(f"took {i} iterations to converge")
df = pd.DataFrame(
    {
        "iteration": np.arange(1, i + 1),
        "angle for bus 2 [radians]": [xx[0] for xx in deltas[1:]],
        "angle for bus 3 [radians]": [xx[1] for xx in deltas[1:]],
    }
)


d2 = deltas[-1][0]
d3 = deltas[-1][1]
Q_sync_condenser = 0.5 + -V_3 * (
    V_1 * B(1, 3) * np.cos(d3) + V_2 * B(2, 3) * np.cos(d3 - d2) + V_3 * B(3, 3)
)
print("Q_sync_condenser:", Q_sync_condenser)

df
```

```
took 3 iterations to converge
Q_sync_condenser: 0.3214788277576588
```