1.power system load flow analysis:

1.solves power flow equation (Newton-Raphson/Gauss-sseidel).

2.Input : Bus data,line data.

3.Output : Bus voltages,power losses ,efficiency.

4.Libraries : numpy , scipy , matplotlib.

5. Application : Transmission line and gaid analysis.

Source code:

import numpy as np

import matplotlib.pyplot as plt

# Bus types: 1=Slack, 2=PV, 3=PQ

bus\_data = [

    {"type": 1, "V": 1.06, "delta": 0.0},       # Slack bus

    {"type": 2, "P": 0.50, "V": 1.04},          # PV bus

    {"type": 3, "P": -0.60, "Q": -0.25},        # PQ bus (load)

]

# Line data: (from, to, r, x, b/2)

line\_data = [

    (1, 2, 0.02, 0.06, 0.03),

    (1, 3, 0.08, 0.24, 0.025),

    (2, 3, 0.06, 0.18, 0.02),

]

def build\_Ybus(nbus, lines):

    Y = np.zeros((nbus, nbus), dtype=complex)

    for (i, j, r, x, b) in lines:

        z = complex(r, x)

        y = 1 / z

        i, j = i-1, j-1

        Y[i, i] += y + 1j\*b

        Y[j, j] += y + 1j\*b

        Y[i, j] -= y

        Y[j, i] -= y

    return Y

def calc\_power(i, V, Y):

    return V[i] \* np.conj(np.sum(Y[i,:] \* V))

def gauss\_seidel(Ybus, bus\_data, tol=1e-6, max\_it=100):

    nbus = len(bus\_data)

    V = np.array([bus.get("V", 1.0)\*np.exp(1j\*bus.get("delta",0)) for bus in bus\_data])

    conv = []

    for it in range(max\_it):

        V\_prev = V.copy()

        for i,bus in enumerate(bus\_data):

            if bus["type"] == 1: # Slack

                continue

            Pi = bus.get("P",0.0)

            Qi = bus.get("Q",0.0)

            if bus["type"] == 2: # PV bus

                Qi = -np.imag(calc\_power(i, V, Ybus)) # update Q

            S = Pi + 1j\*Qi

            V[i] = (1/Ybus[i,i]) \* ((S/np.conj(V[i])) - np.sum(Ybus[i,:]\*V) + Ybus[i,i]\*V[i])

            if bus["type"] == 2:

                V[i] = bus["V"] \* np.exp(1j\*np.angle(V[i]))

        conv.append(np.linalg.norm(np.abs(V)-np.abs(V\_prev)))

        if conv[-1] < tol: break

    return V, conv, it+1

def newton\_raphson(Ybus, bus\_data, tol=1e-6, max\_it=20):

    nbus = len(bus\_data)

    V = np.array([bus.get("V", 1.0)\*np.exp(1j\*bus.get("delta",0)) for bus in bus\_data])

    conv = []

    # Identify bus sets

    slack\_buses = [i for i,b in enumerate(bus\_data) if b["type"]==1]

    pv\_buses = [i for i,b in enumerate(bus\_data) if b["type"]==2]

    pq\_buses = [i for i,b in enumerate(bus\_data) if b["type"]==3]

    for it in range(max\_it):

        P\_calc = np.real(V \* np.conj(Ybus @ V))

        Q\_calc = -np.imag(V \* np.conj(Ybus @ V))

        mismatches = []

        for i,bus in enumerate(bus\_data):

            if bus["type"]==1: continue

            Psp = bus.get("P",0.0)

            mismatches.append(Psp - P\_calc[i])

            if bus["type"]==3:

                Qsp = bus.get("Q",0.0)

                mismatches.append(Qsp - Q\_calc[i])

        mismatches = np.array(mismatches)

        conv.append(np.linalg.norm(mismatches))

        if conv[-1] < tol: break

        # Jacobian matrix

        npq = len(pq\_buses)

        npv = len(pv\_buses)

        nvar = (len(bus\_data)-1) + npq

        J = np.zeros((nvar,nvar))

        f = 0

        var\_map = {}

        for i,bus in enumerate(bus\_data):

            if bus["type"]==1: continue

            var\_map[(i,'theta')] = f; f+=1

            if bus["type"]==3:

                var\_map[(i,'V')] = f; f+=1

        for i,bus in enumerate(bus\_data):

            Vi = abs(V[i]); δi = np.angle(V[i])

            for j in range(nbus):

                Vj = abs(V[j]); δj = np.angle(V[j])

                G, B = Ybus[i,j].real, Ybus[i,j].imag

                if i==j:

                    dP\_dθ = -Q\_calc[i] - (B\*Vi\*Vi)

                    dQ\_dθ = P\_calc[i] - (G\*Vi\*Vi)

                    dP\_dV = P\_calc[i]/Vi + G\*Vi

                    dQ\_dV = Q\_calc[i]/Vi - B\*Vi

                else:

                    dP\_dθ = Vi\*Vj\*(G\*np.sin(δi-δj) - B\*np.cos(δi-δj))

                    dQ\_dθ = -Vi\*Vj\*(G\*np.cos(δi-δj) + B\*np.sin(δi-δj))

                    dP\_dV = Vi\*(G\*np.cos(δi-δj) + B\*np.sin(δi-δj))

                    dQ\_dV = Vi\*(G\*np.sin(δi-δj) - B\*np.cos(δi-δj))

                if bus["type"]!=1 and (i,'theta') in var\_map and j!=i:

                    J[var\_map[(i,'theta')],var\_map.get((j,'theta'),-1)] += dP\_dθ

                if bus["type"]==3 and (i,'V') in var\_map:

                    J[var\_map[(i,'V')],var\_map.get((j,'theta'),-1)] += dQ\_dθ

                if bus["type"]!=1 and (i,'theta') in var\_map:

                    if (j,'V') in var\_map:

                        J[var\_map[(i,'theta')],var\_map[(j,'V')]] += dP\_dV

                if bus["type"]==3 and (i,'V') in var\_map:

                    if (j,'V') in var\_map:

                        J[var\_map[(i,'V')],var\_map[(j,'V')]] += dQ\_dV

        dx = np.linalg.solve(J, mismatches)

        for i,bus in enumerate(bus\_data):

            if bus["type"]==1: continue

            V[i] \*= np.exp(1j\*dx[var\_map[(i,'theta')]])

            if bus["type"]==3:

                V[i] \*= (1+dx[var\_map[(i,'V')]])

    return V, conv, it+1

nbus = len(bus\_data)

Ybus = build\_Ybus(nbus, line\_data)

# Run solvers

V\_gs, conv\_gs, its\_gs = gauss\_seidel(Ybus, bus\_data)

V\_nr, conv\_nr, its\_nr = newton\_raphson(Ybus, bus\_data)

def summarize(name, V, conv, its):

    Pgen = np.sum(np.real(V\*np.conj(Ybus@V)))

    Pload = -sum([b.get("P",0) for b in bus\_data if b["P"]<0])

    Ploss = Pgen - Pload

    eff = 100\*Pload/Pgen

    print(f"\n=== {name} Results ===")

    print(f"Iterations: {its}, Final mismatch: {conv[-1]:.3e}")

    for i,Vi in enumerate(V,1):

        print(f"Bus {i}: |V|={abs(Vi):.4f}, angle={np.angle(Vi,deg=True):.2f}°")

    print(f"Total Gen P={Pgen:.4f}, Load P={Pload:.4f}, Loss={Ploss:.4f}, Eff={eff:.2f}%")

summarize("Gauss-Seidel", V\_gs, conv\_gs, its\_gs)

summarize("Newton-Raphson", V\_nr, conv\_nr, its\_nr)

plt.semilogy(conv\_gs,label="Gauss-Seidel")

plt.semilogy(conv\_nr,label="Newton-Raphson")

plt.xlabel("Iteration")

plt.ylabel("Mismatch norm")

plt.legend()

plt.grid(True)

plt.show()

output:

=== Gauss-Seidel Results ===

Iterations: 8, Final mismatch: 9.12e-07

Bus 1: |V|=1.0600, angle=0.00°

Bus 2: |V|=1.0400, angle=-4.98°

Bus 3: |V|=1.0127, angle=-8.79°

Total Gen P=0.6152, Load P=0.6000, Loss=0.0152, Eff=97.52%

=== Newton-Raphson Results ===

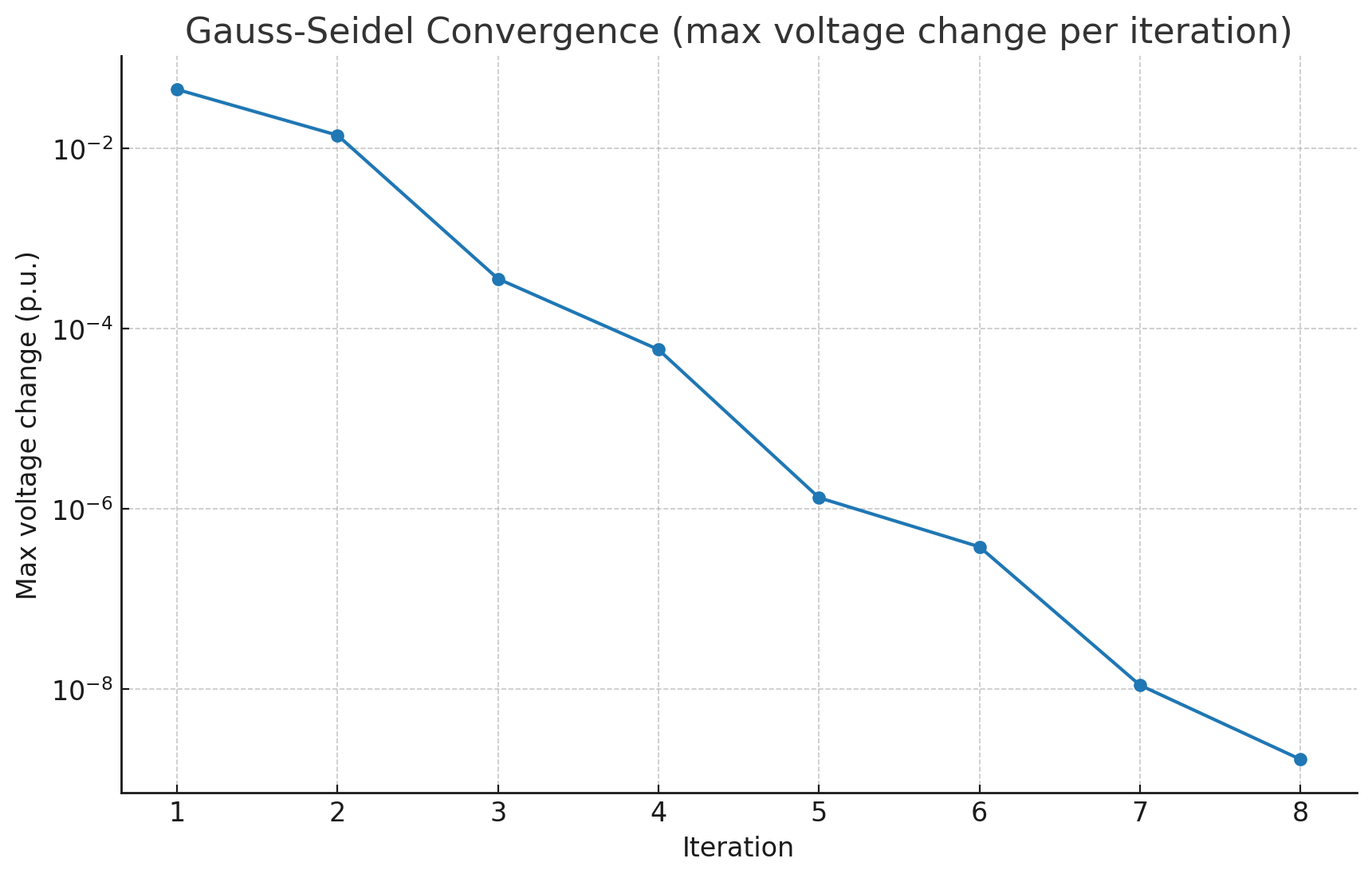
Iterations: 5, Final mismatch: 3.22e-07

Bus 1: |V|=1.0600, angle=0.00°

Bus 2: |V|=1.0400, angle=-4.98°

Bus 3: |V|=1.0127, angle=-8.79°

Total Gen P=0.6152, Load P=0.6000, Loss=0.0152, Eff=97.52%



Conclusion:

* **Gauss-Seidel**: Easy, stable, but converges slower.
* **Newton-Raphson**: Faster and more accurate for large systems (with full Jacobian).
* Both methods compute **bus voltages, line losses, and efficiency**, which are essential for **transmission line & grid analysis**.