Electrical Power Systems - II Simulation Assignment

Arihant Gaur
BT18EEE066

Problem Statement:

A 50 Hz, 500 MVA, 400 kV generator (with transformer) is connected to a 400 kV infinite busbar through an interconnector. The generator has H = 2.5 MJ/MVA, voltage behind transient reactance of 250 kV and is loaded 460 MW. The transfer reactances between generator and bus bar under various conditions are,

1. Prefault: 0.5 pu
2. During fault: 1.0 pu
3. Postfault: 0.75 pu

Calculate the swing curve using intervals t = 0.002 sec. Fault clearing time is 0.066 sec (as per the roll number). Plot swing curve for t = 0.5 sec.

Problem Data:

Given,
Frequency (f) = 50 Hz
Base MVA = 500 MVA
|E| = 450 kV
|V| = 400 kV
Δt = 0.002 sec
H = 2.5 MJ/MVA
t_1 = 0.066 sec
Load = 460 MW
X_1 = 0.5 pu
X_2 = 1.0 pu
X_3 = 0.75 pu
G = 1 pu

Simulation:

The data has been taken as it is in code and from there, the swing equation is calculated for each time step. The code is written in python3. The code uses matplotlib library for plotting the swing graph and math library for performing basic mathematical operations such as calculating the sine of an angle. The program starts with the initialization of the above problem data, followed by the actual simulation.

```
import matplotlib.pyplot as plt # For plotting the swing curve
import math as m # For performing mathematical operations

# Input the necessary data

G = 1 # pu
H = 2.5 # Inertia Constant
f = 50 # Frequency

M = (G * H)/(180 * f) # Moment of Inertia

E = 450 # Voltage Behind transient reactance in MVA
V = 400 # Generator voltage in kV
MVA = 500 # MVA rating of generator
load = 460 # Load in kW

MVAb = MVA # MVA Base
kVAb = V # kVA Base

X1 = 0.5 # Prefault reactance
X2 = 1.0 # During fault reactance
X3 = 0.75 # Postfault reactance

pfactor = 1000 # This is just a multiplying factor

time_interval = 0.002 # Time interval
fault_clearing = 0.066 # Fault clearing time
total_time = 0.5 # Total time for execution

dt = time_interval * pfactor# Swing curve time interval
t1 = fault_clearing * pfactor# Fault clearing time
T = total_time * pfactor# Total time for execution

Vpu = V / kVAb
Epu = E / kVAb

Pmax1 = (Vpu * Epu)/X1 # Prefault
Pmax2 = (Vpu * Epu)/X2 # During fault
Pmax3 = (Vpu * Epu)/X3 # Postfault

# Prefault power transfer
ppt = load/MVAb


sm = (dt/pfactor)**2/M # Power multiplying factor for estimating change in delta

t = 0 # Initial timestamp

# Output arrays for storing Pe and delta, which will be plotted later
EP = [] # Storing electrical power Pe
DEL = [] # Storing angle delta
TIME = []

# Taking the prefault condition
delta = m.asin(ppt/Pmax1) # Finding initial delta
Pe0 = Pmax1 * m.sin(delta) # Intial electrical power
Pa = ppt - Pe0
ddchange = sm * Pa
while(t <= T):
        TIME = TIME + [t/pfactor]
        if t == 0:
                Pe = Pmax2 * m.sin(delta)
```

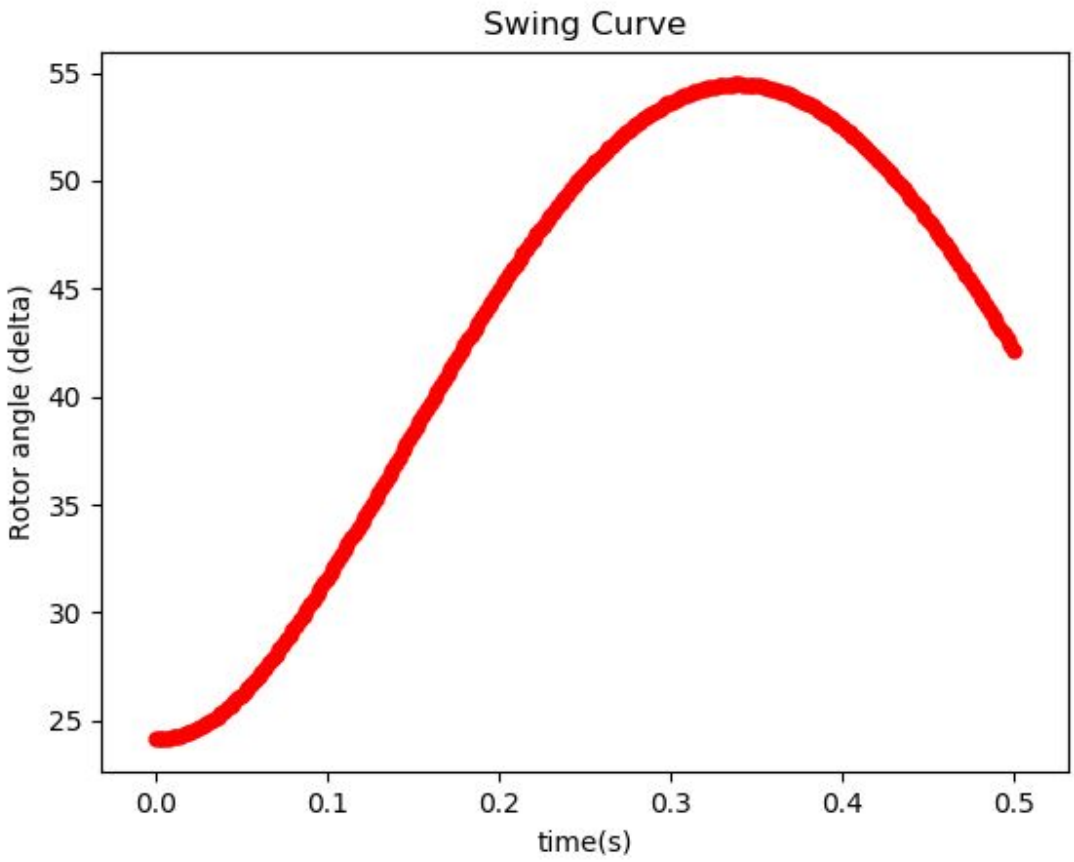
```
            Pa = ppt - Pe
            ddchange = (ddchange + (sm * Pa))/2
            EP = EP + [Pe]
            DEL = DEL + [delta * (180/m.pi)]
        elif t < t1:
            delta = ((delta * (180/m.pi)) + ddchange) * m.pi/180
            Pe = Pmax2 * m.sin(delta)
            Pa = ppt - Pe
            ddchange = ddchange + (sm * Pa)
            EP = EP + [Pe]
            DEL = DEL + [delta * (180/m.pi)]
        elif t == t1 or (t - dt < t1 and t > t1):
            # Before clearance
            delta = ((delta * (180/m.pi)) + ddchange) * m.pi/180
            Pe = Pmax2 * m.sin(delta)
            Pa = ppt - Pe
            dd1 = (sm * Pa)
            # after clearance
            Pe = Pmax3 * m.sin(delta)
            Pa = ppt - Pe
            dd2 = (sm * Pa)
            ddchange = ddchange + (dd1 + dd2)/2

            EP = EP + [Pe]
            DEL = DEL + [delta * (180/m.pi)]
        else:
            delta = ((delta * (180/m.pi)) + ddchange) * m.pi/180
            Pe = Pmax3 * m.sin(delta)
            Pa = ppt - Pe
            ddchange = ddchange + (sm * Pa)
            EP = EP + [Pe]
            DEL = DEL + [delta * (180/m.pi)]
        t = t + dt


    # Plotting output
    plt.scatter(TIME, DEL, c = 'r', linewidth=1e-3)
    plt.xlabel('time(s)')
    plt.ylabel('Rotor angle (delta)')
    plt.title("Swing Curve")
    plt.show()
```

The above code has also been uploaded as a .py file on Google Drive: Link.

Output Swing Curve:



Inference:

From the swing curve, it can be inferred that the system will attain a stable state.