Load Frequency Control of Single Area Thermal Power Plant Using Type 1 Fuzzy Logic Controller Simulation

---

By Ali Bornaee Sup. by Dr. Shahsadeghi

Importing Libraries

We use python to simulate this system and show output of it using `scipy`, `numpy` and `matplotlib`. Then we tries to create a PID controller and fuzzy controller using `scikit-fuzzy` package.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from graphviz import Digraph
from IPython.display import Image, display
import skfuzzy as fuzz
import skfuzzy.control as ctrl

# === SYSTEM SETUP ===
Tg = 0.08
Tt = 0.3
Tp = 20.0
Kp_sys = 120
R = 2.4

t = np.linspace(0, 40, 1000)
dt = t[1] - t[0]
ΔPL = np.ones_like(t) * 0.01

u_min, u_max = -1.0, 1.0
```

Creating fuzzy controll system

We create and train the fuzzy control system based on what we've got from the article. We create the membership function and rulebase and try to tune the parameters of each.

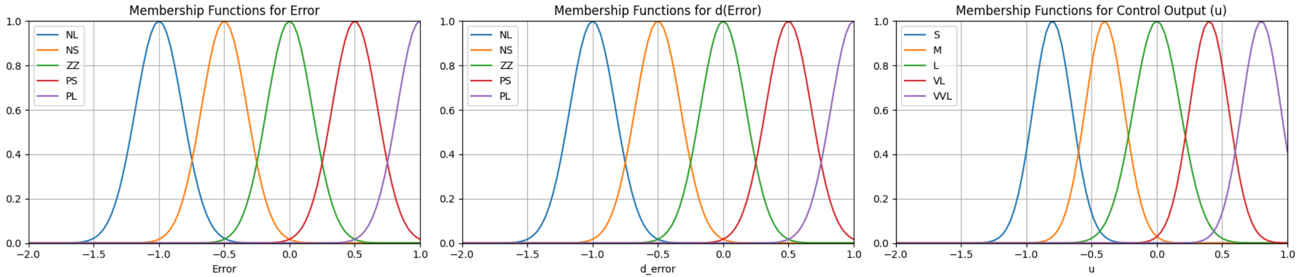
```
# === FUZZY CONTROLLER SETUP ===
linguistic_labels = ['NL', 'NS', 'ZZ', 'PS', 'PL']
output_labels = ['S', 'M', 'L', 'VL', 'VVL']

# Define fuzzy variables
x_error = np.linspace(-2, 1, 100)
x_d_error = np.linspace(-2, 1, 100)
x_u = np.linspace(-2, 1, 100)

error = ctrl.Antecedent(x_error, 'error')
d_error = ctrl.Antecedent(x_d_error, 'd_error')
u_out = ctrl.Consequent(x_u, 'u')

# Define custom Gaussian MFs
```

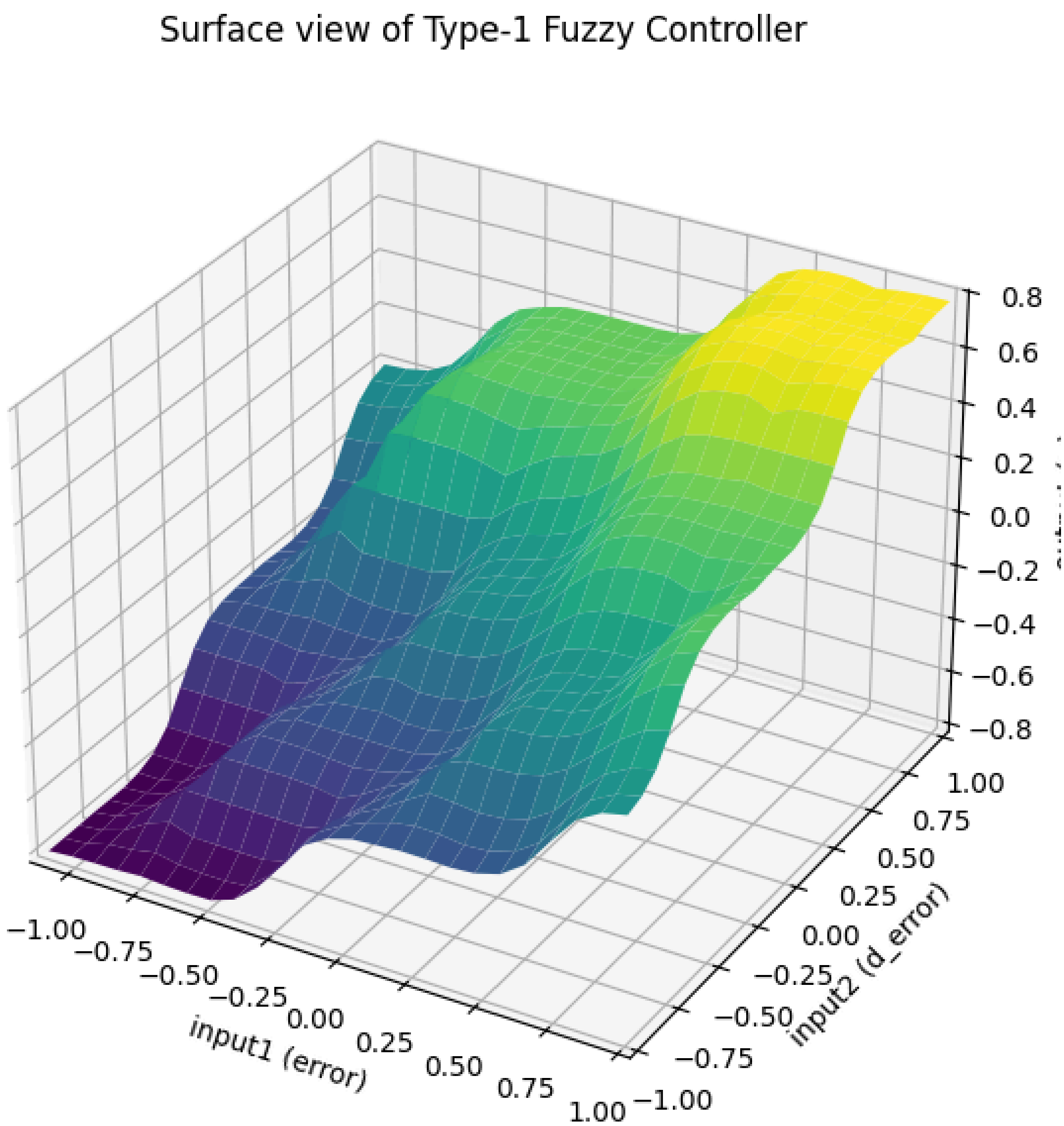
```python
error['NL'] = fuzz.gaussmf(x_error, -1.0, 0.18)
error['NS'] = fuzz.gaussmf(x_error, -0.5, 0.175)
error['ZZ'] = fuzz.gaussmf(x_error, 0.0, 0.175)
error['PS'] = fuzz.gaussmf(x_error, 0.5, 0.175)
error['PL'] = fuzz.gaussmf(x_error, 1.0, 0.175)

d_error['NL'] = fuzz.gaussmf(x_d_error, -1.0, 0.175)
d_error['NS'] = fuzz.gaussmf(x_d_error, -0.5, 0.175)
d_error['ZZ'] = fuzz.gaussmf(x_d_error, 0.0, 0.175)
d_error['PS'] = fuzz.gaussmf(x_d_error, 0.5, 0.175)
d_error['PL'] = fuzz.gaussmf(x_d_error, 1.0, 0.175)

u_out['S']   = fuzz.gaussmf(x_u, -0.8, 0.15)
u_out['M']   = fuzz.gaussmf(x_u, -0.4, 0.15)
u_out['L']   = fuzz.gaussmf(x_u,  0.0, 0.18)
u_out['VL']  = fuzz.gaussmf(x_u,  0.4, 0.15)
u_out['VVL'] = fuzz.gaussmf(x_u,  0.8, 0.15)

rules = [
    ctrl.Rule(error['NL'] & d_error['NL'], u_out['S']),
    ctrl.Rule(error['NL'] & d_error['NS'], u_out['S']),
    ctrl.Rule(error['NL'] & d_error['ZZ'], u_out['M']),
    ctrl.Rule(error['NL'] & d_error['PS'], u_out['M']),
    ctrl.Rule(error['NL'] & d_error['PL'], u_out['L']),
    ctrl.Rule(error['NS'] & d_error['NL'], u_out['S']),
    ctrl.Rule(error['NS'] & d_error['NS'], u_out['M']),
    ctrl.Rule(error['NS'] & d_error['ZZ'], u_out['M']),
    ctrl.Rule(error['NS'] & d_error['PS'], u_out['VL']),
    ctrl.Rule(error['NS'] & d_error['PL'], u_out['VL']),
    ctrl.Rule(error['ZZ'] & d_error['NL'], u_out['M']),
    ctrl.Rule(error['ZZ'] & d_error['NS'], u_out['M']),
    ctrl.Rule(error['ZZ'] & d_error['ZZ'], u_out['L']),
    ctrl.Rule(error['ZZ'] & d_error['PS'], u_out['VL']),
    ctrl.Rule(error['ZZ'] & d_error['PL'], u_out['VL']),
    ctrl.Rule(error['PS'] & d_error['NL'], u_out['M']),
    ctrl.Rule(error['PS'] & d_error['NS'], u_out['L']),
    ctrl.Rule(error['PS'] & d_error['ZZ'], u_out['VL']),
    ctrl.Rule(error['PS'] & d_error['PS'], u_out['VVL']),
    ctrl.Rule(error['PS'] & d_error['PL'], u_out['VVL']),
    ctrl.Rule(error['PL'] & d_error['NL'], u_out['L']),
    ctrl.Rule(error['PL'] & d_error['NS'], u_out['VL']),
    ctrl.Rule(error['PL'] & d_error['ZZ'], u_out['VL']),
    ctrl.Rule(error['PL'] & d_error['PS'], u_out['VVL']),
    ctrl.Rule(error['PL'] & d_error['PL'], u_out['VVL']),
]

fuzzy_ctrl = ctrl.ControlSystem(rules)
fuzzy_sim = ctrl.ControlSystemSimulation(fuzzy_ctrl)

# === SYSTEM COEFFICIENTS ===
a1 = Tp + Tt + Tg
a2 = Tp*Tt + Tp*Tg + Tt*Tg
a3 = Tp*Tt*Tg

# === SIMULATION WITH FUZZY CONTROLLER ===
y_fuzzy = np.zeros((len(t), 3))
output_fuzzy = np.zeros(len(t))
prev_error = 0.01
```

Visuallizing the fuzzy system

The plots bellow shows the membership function with surface graph of the system. It's tuned and ready to work.

```python
import matplotlib.pyplot as plt

plt.rcParams["axes.xmargin"] = 0
plt.rcParams["axes.ymargin"] = 0

# Plot all membership functions in one row
fig, axs = plt.subplots(1, 3, figsize=(18, 4))

# Plot for error
for label in error.terms:
    axs[0].plot(x_error, error[label].mf, label=label)
axs[0].set_title('Membership Functions for Error')
axs[0].set_xlabel('Error')
axs[0].legend()
axs[0].grid(True)

plt.tight_layout()

# Plot for d_error
for label in d_error.terms:
    axs[1].plot(x_d_error, d_error[label].mf, label=label)
axs[1].set_title('Membership Functions for d(Error)')
axs[1].set_xlabel('d_error')
axs[1].legend()
axs[1].grid(True)

# Plot for output u
for label in u_out.terms:
    axs[2].plot(x_u, u_out[label].mf, label=label)
axs[2].set_title('Membership Functions for Control Output (u)')
axs[2].set_xlabel('u')
axs[2].legend()
axs[2].grid(True)

plt.tight_layout()
plt.show()

# Plot fuzzy surface again
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='viridis')
ax.set_xlabel('input1 (error)')
ax.set_ylabel('input2 (d_error)')
ax.set_zlabel('output (u)')
ax.set_title('Surface view of Type-1 Fuzzy Controller')
plt.show()
```





Training the fuzzy system inside the block of plant

```python
# === SIMULATION WITH FUZZY CONTROLLER ===

for i in range(1, len(t)):
    error_val = -y_fuzzy[i-1, 0]
    derivative = (error_val - prev_error) / dt
    prev_error = error_val

    try:
        fuzzy_sim.input['error'] = float(np.clip(error_val, -1, 1))
        fuzzy_sim.input['d_error'] = float(np.clip(derivative, -1, 1))
        fuzzy_sim.compute()
        u = fuzzy_sim.output['u']
    except:
        u = 0.0

    u = np.clip(u, u_min, u_max)

    dddf = (Kp_sys * (u - ΔPL[i] / R) - a1 * y_fuzzy[i-1, 2] - a2 * y_fuzzy[i-1, 1]
    y_fuzzy[i, 0] = y_fuzzy[i-1, 0] + y_fuzzy[i-1, 1] * dt
    y_fuzzy[i, 1] = y_fuzzy[i-1, 1] + y_fuzzy[i-1, 2] * dt
    y_fuzzy[i, 2] = y_fuzzy[i-1, 2] + dddf * dt
    output_fuzzy[i] = y_fuzzy[i, 0]
```

Creating the PID controller

```
# === SIMULATION WITH PID CONTROLLER ===
Kp_pid, Ki_pid, Kd_pid = 0.4, 0.2, 0.1
integral = 0.0
prev_error = 0.0
y_pid = np.zeros((len(t), 3))
output_pid = np.zeros(len(t))

for i in range(1, len(t)):
    error = -y_pid[i-1, 0]
    integral += error * dt
    integral = np.clip(integral, -10, 10)
    derivative = (error - prev_error) / dt
    prev_error = error
    u = Kp_pid * error + Ki_pid * integral + Kd_pid * derivative
    u = np.clip(u, u_min, u_max)

    dddf = (Kp_sys * (u - ΔPL[i] / R) - a1 * y_pid[i-1, 2] - a2 * y_pid[i-1, 1] - a
    y_pid[i, 0] = y_pid[i-1, 0] + y_pid[i-1, 1] * dt
    y_pid[i, 1] = y_pid[i-1, 1] + y_pid[i-1, 2] * dt
    y_pid[i, 2] = y_pid[i-1, 2] + dddf * dt
    output_pid[i] = y_pid[i, 0]
```

Plotting the output of the system

```
# === PLOT COMPARISON ===
plt.figure(figsize=(10, 5))
plt.plot(t, output_pid, label='PID Controller', linestyle='--')
plt.plot(t, output_fuzzy, label='Fuzzy Controller', linestyle='-')
plt.title('Frequency Deviation Δf(t): PID vs Fuzzy Controller')
plt.xlabel('Time [s]')
plt.ylabel('Frequency Deviation [Hz]')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```