Optimization of PID Controller using Nelder Mead

(Optimization of Settling Time)

By

Rushil Gupta

(BTECH/10031/21)

Report Submitted To

Dr. Sushma Kamlu
(MO-2023)

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

BIRLA INSTITUTE OF TECHNOLOGY

MESRA, RANCHI - 835215

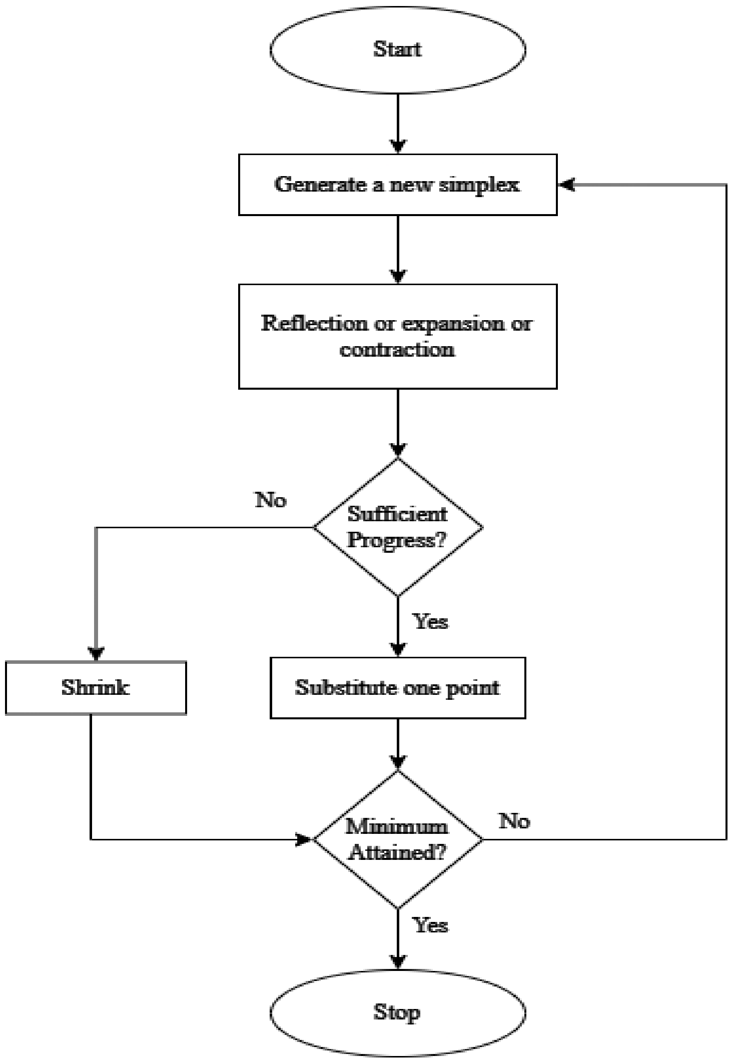
Problem Statement

Optimize PID controller parameters using Nelder-Mead in Python to minimize Settling Time (t_s), enhancing control system performance through precise and efficient tuning.

Theory

The Nelder-Mead optimization technique is a versatile algorithm employed to optimize PID controller parameters, specifically targeting the reduction of Settling Time (t_s) in control systems. Unlike gradient-based methods, Nelder-Mead operates without requiring gradient information, making it applicable to complex, non-linear systems. The algorithm iteratively adjusts proportional, integral, and derivative parameters, seeking the optimal combination that minimizes settling time. This approach enhances control system performance by mitigating the extent to which the system response surpasses its steady-state value. The Nelder-Mead technique is prized for its effectiveness in navigating parameter space, contributing to precise tuning and improved stability without the need for explicit mathematical derivatives.

Flow Chart

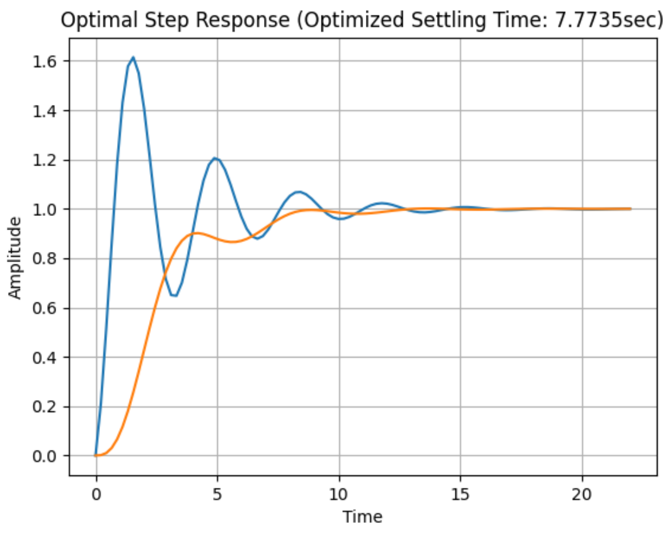


Python Code

```
from scipy.optimize import minimize

from scipy.signal import TransferFunction, step

import matplotlib.pyplot as plt


numerator = [float(i) for i in input("Enter TF Numerator: ").split()]

denominator = [float(i) for i in input("Enter TF Denominator: ").split()]

sys = TransferFunction(numerator, denominator)


def objective_function(params):

    kp, ki, kd = params

    tf = TransferFunction(numerator, [1, kp, ki, kd])

    t, y = step(tf)

    twoPercentage = 0.02 * y[-1]

    i = len(y) - 1

    while (y[-1] - twoPercentage) < y[i] < (y[-1] + twoPercentage):

        i -= 1

    return t[i]


initial_guess = [1.0, 1.0, 1.0]

result = minimize(objective_function, initial_guess, method="nelder-mead")

optimal_gains = result.x

optimal_tf = TransferFunction(numerator, [1, *optimal_gains])


_, y1 = step(sys)

t, y = step(optimal_tf)

factor = y[-1] / y1[-1]

y1 = [i / y1[-1] for i in y1]

y = [i / y[-1] for i in y]


twoPercentage1 = 0.02 * y1[-1]

i = len(y1) - 1

while i > 0 and (y1[-1] - twoPercentage1) < y1[i] < (y1[-1] + twoPercentage1):

    i -= 1
```

```
st1 = t[i]
print(f"Original Settling Time: {st1}sec")


twoPercentage = 0.02 * y[-1]
i = len(y) - 1
while i > 0 and (y[-1] - twoPercentage) < y[i] < (y[-1] + twoPercentage):
    i -= 1
st = t[i]
print(f"Optimized Settling Time: {st}sec")


plt.plot(t, y1)
plt.plot(t, y)
plt.title(f"Optimal Step Response (Optimized Settling Time: {st:.4f}sec)")
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
print("Optimal PID Controller Gains:", optimal_gains)
```

Graphs



🟦 : Original System

🟨 : Optimized System

Input:

Enter TF Numerator: *0.0123   1*

Enter TF Denominator: *0.0012345   0.012   1*

Output:

Original Settling Time: *11.99sec*

Optimized Settling Time: *7.77sec*

Optimal PID Controller Gains [k_p, k_i, k_d]: *[1.06004954 1.99354782 0.72983458]*

Conclusion

1. **Settling Time (t_s) Reduction:** Through the implementation of the Nelder-Mead Optimization Technique, a substantial decrease in Settling Time (t_s) was achieved, demonstrating its effectiveness in fine-tuning the control system.

2. **Maximum Peak Overshoot Improvement:** The application of the optimization technique also led to a notable reduction in Maximum Peak Overshoot, highlighting enhanced responsiveness and speed of the controlled system.

3. **Consistent Steady-State Response:** Despite the significant adjustments made to improve overshoot and settling time, the steady-state response remained consistent for both the optimized and initial systems.

These observations collectively highlight the Nelder-Mead Optimization Technique as a powerful tool for achieving specific performance improvements in control systems while maintaining stability in steady-state conditions.