# PROCESS INSTRUMENTATION ASSIGNMENT 1

**1.** One time constant is the amount of time to get to $1 - e^{-1} = 0.632$ or 63.2% of the way to steady state from 0 to 1. It comes from the analytic solution of the first order differential equation $\tau \frac{dy}{dt} + y = u$ with $u = 1$.

$$y(t) = \left(1 - e^{-t/\tau}\right)$$

What is the value of $y(t)$ at two time constants $t = 2\tau$?

```python
#Qn1
def response_at_normalized_time(normalized_time):
  """
  This function calculates the response (y) of a first-order system
  at a specified normalized time constant (t/tau).

  Args:
      normalized_time: The time value normalized by the time constant (tau).

  Returns:
      The response (y) of the system at the given normalized time.
  """
  # Define the time constant
  time_constant = 1

  # Calculate response
  response = 1 - np.exp(-normalized_time)
  return response

# Calculate response at t = 2*tau (normalized time = 2)
normalized_time = 2
system_response = response_at_normalized_time(normalized_time)

# Print the result
print("The system response at t =", normalized_time, "tau is:", system_response)
```

```
The system response at t = 2 tau is: 0.8646647167633873
```

Use this information to answer questions 2-4. A first-order linear system with time delay is a common empirical description of many stable dynamic processes. The equation

$$\tau_p \frac{dy(t)}{dt} = -y(t) + K_p u\left(t - \theta_p\right)$$

has variables *y(t)* and *u(t)* and three unknown parameters.

$$K_p \quad = \text{Process gain}$$

$$\tau_p \quad = \text{Process time constant}$$

$$\theta_p \quad = \text{Process dead time}$$

**2.** Determine the value of $K_p$ that best fits the step response data.

**3.** Determine the value of $\theta_p$ that best fits the step response data.

**4.** Determine the value of $\tau_p$ that best fits the step response data.

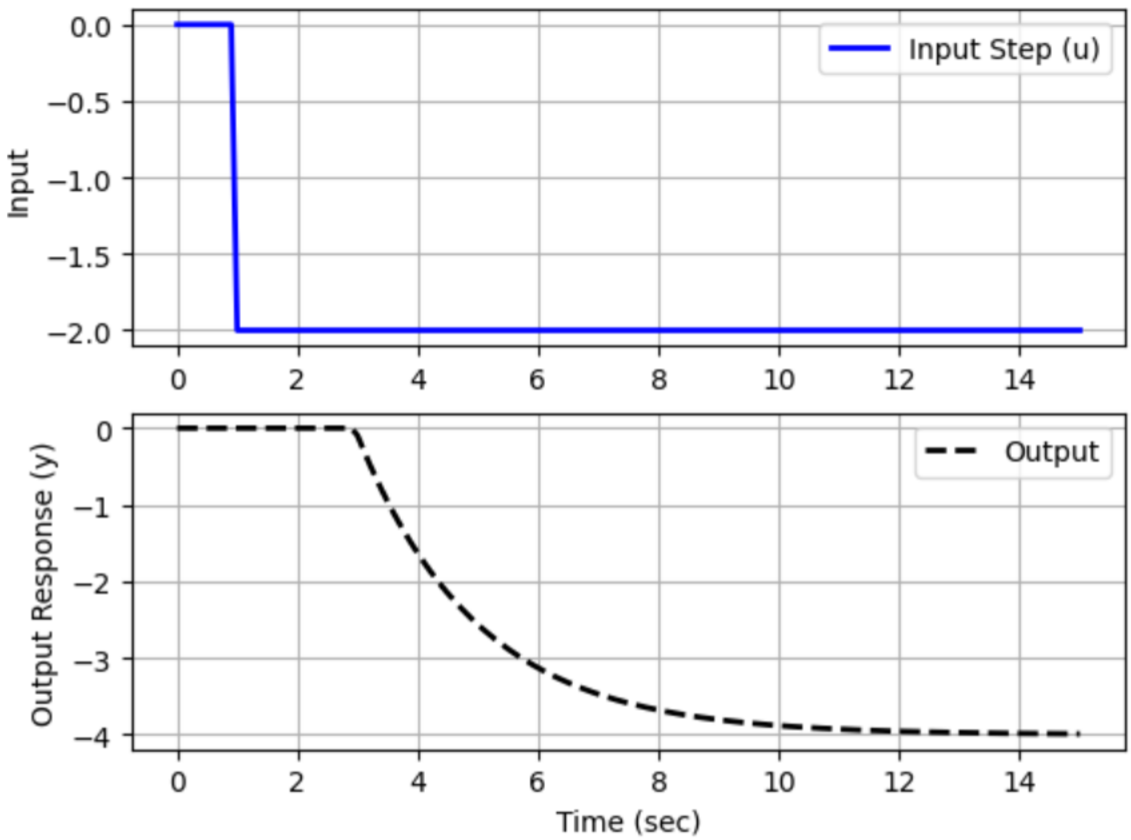
```python
# Specify simulation parameters
num_steps = 150
time_points = np.linspace(0, 15, num_steps + 1)
time_step = time_points[1] - time_points[0]

# Define input signal
input_signal = np.zeros(num_steps + 1)
input_signal[10:] = -2.0

# Create linear interpolation function for input
input_func = interp1d(time_points, input_signal)

# Simulate FOPDT model
def simulate_fopdt(gain, time_constant, dead_time):
  # Initialize output storage
  model_output = np.zeros(num_steps + 1)
  # Set initial condition
  model_output[0] = 0

  # Loop through time steps
  for i in range(1, num_steps + 1):
    time_span = [time_step * (i - 1), time_step * i]
    # Solve differential equation for the current step
    state_solution = odeint(fopdt_system, [model_output[i - 1]], time_span, args=(input_func, gain, time_constant, dead_time))
    model_output[i] = state_solution[1][0]  # Access final state value
  return model_output
```




*-Chandramita Santra*

*121CH0950*