Problem 1 (33%)

In this problem, you will be working with a discrete-time system that has 2 inputs and 2 outputs. Your task is to implement the provided code and perform various operations on the system.

Instructions:

1. Load the data:
   - Use the provided code snippet to load the data from the file `testSys.mat`. This file contains the necessary information for the system.(5%)
2. Calculate and plot the Hankel singular values:
   - Use the loaded data to compute the Hankel singular values of the system.
   - Plot the Hankel singular values using a logarithmic scale and mark the singular value corresponding to a reduced model order of `r = 10` with a blue circle.(6%)
3. Choose a reduced model order:
   - Find the minimum reduced model order `r` so that the reduced model can capture over 80% of the input-output energy.(10%)
4. Implement exact balanced truncation:
   - Use the provided system and the chosen reduced model order `r` to perform exact balanced truncation.
   - Obtain the reduced model using the balanced truncation technique.(6%)
5. Plot the step responses:
   - Plot the step response of both the full model and the reduced model on the same graph.
   - Set the final time `Tend` to 50 for the step response plot.(6%)

Problem: GA for controller design (33%)

A unity negative feedback control system has loop transfer function

$$L(s) = G_c(s)G(s) = G_c\frac{1}{(s+6)^2}.$$

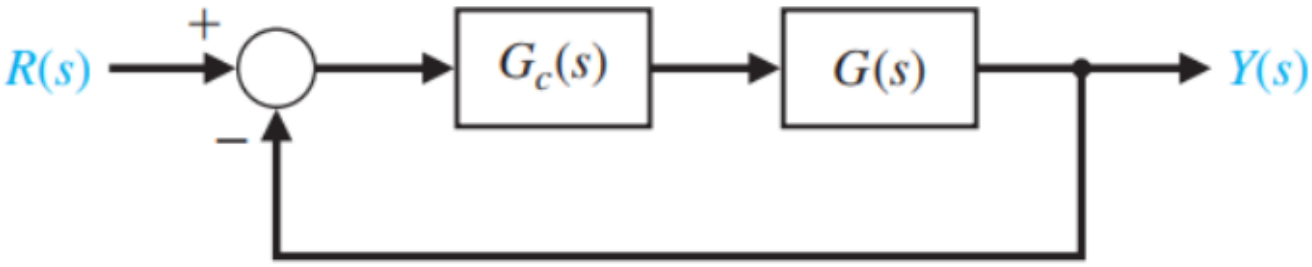

Figure 1. A feedback control system with compensation.

Design a **phase-lag** compensator using GA to meet the following specs:

- steady-state error to a step input to be 5%;
- phase margin to be $44^\circ \leq P.M. \leq 46^\circ$.
  You need to implement the following:

1. Training part using GA. (13%)
2. Evaluation using the designed compensator. In the evaluation, you need to justify that your compensator can meet the specs. (20%)

Problem: SARSA Algorithm for Grid World (34%)

You are given a grid world represented by a 5×5 grid. The agent starts at the top-left corner (cell [0, 0]) and needs to reach the goal at the top-right corner (cell [4, 0]). The agent can move in four directions: up, down, left, and right. However, some cells in the grid are blocked and cannot be traversed.
Specifics:

- The grid world layout is as follows:

```
---------------------
| S |   |   | X | G |
---------------------
|   | X |   | X |   |
---------------------
|   |   |   |   |   |
---------------------
| X |   | X |   |   |
---------------------
|   |   | X |   |   |
---------------------
```

- "S" represents the starting cell at [0, 0].
- "G" represents the goal cell at [4, 0].
- "X" represents blocked cells that cannot be traversed.

You need to use Python or MATLAB to implement the following:

Python

PYTHON

```python
# GridWorld class: This class represents the grid world environment.
# It should have the following methods:
class GridWorld:
#10%
    def __init__(self):
        # Initializes the grid world by defining the grid layout,
        # including blocked cells and the goal cell.
        pass

    def get_actions(self, state):
        # Returns the available actions in a given state.
        pass

    def is_goal(self, state):
        # Checks if the given state is the goal state.
        pass

    def is_blocked(self, state):
        # Checks if the given state is a blocked cell.
        pass

# SarsaAgent class: This class represents an agent using the SARSA
algorithm.
# It should have the following methods:
class SarsaAgent:
#10%
    def __init__(self, epsilon, alpha, gamma):
        # Initializes the agent with the given epsilon (exploration
rate),
        # alpha (learning rate), and gamma (discount factor).
        pass

    def select_action(self, state):
        # Selects an action for the given state based on the SARSA
algorithm.
        # Returns the selected action.
        pass

    def update_q_table(self, state, action, reward, next_state,
```

```
next_action):
        # Updates the agent's Q-table based on the SARSA update rule
        # using the received reward, next state, and next action.
        pass

    def reset(self):
        # Resets the agent's internal state at the beginning of each
episode.
        pass

# main() function: This function should simulate the grid world
problem
# using the SARSA algorithm and output the optimal path taken by the
agent.
def main():
#7%
    # Simulate the grid world problem using the SARSA algorithm
    # and output the optimal path taken by the agent.
    pass

# evaluate() function: This function should import the stored data
and display the learning curve
# by plotting a line graph or using a suitable visualization method,
with the
# episode number on the x-axis and the total accumulated rewards on
the y-axis.
def evaluate():
#7%
    # Import the stored data and display the learning curve.
    pass
```

MATLAB

MATLAB

```
% GridWorld class: This class represents the grid world environment.
% It should have the following methods:
classdef GridWorld
%10%
    methods
        function obj = GridWorld()
            % Initializes the grid world by defining the grid layout,
            % including blocked cells and the goal cell.
        end

        function actions = get_actions(obj, state)
            % Returns the available actions in a given state.
        end

        function is_goal_state = is_goal(obj, state)
            % Checks if the given state is the goal state.
        end

        function is_blocked_cell = is_blocked(obj, state)
            % Checks if the given state is a blocked cell.
        end
    end
end

% SarsaAgent class: This class represents an agent using the SARSA
algorithm.
% It should have the following methods:
classdef SarsaAgent
%10%
    methods
        function obj = SarsaAgent(epsilon, alpha, gamma)
            % Initializes the agent with the given epsilon
(exploration rate),
            % alpha (learning rate), and gamma (discount factor).
        end

        function action = select_action(obj, state)
            % Selects an action for the given state based on the
SARSA algorithm.
```

```
            % Returns the selected action.
        end

        function update_q_table(obj, state, action, reward,
next_state, next_action)
            % Updates the agent's Q-table based on the SARSA update
rule
            % using the received reward, next state, and next action.
        end

        function reset(obj)
            % Resets the agent's internal state at the beginning of
each episode.
        end
    end
end

% main() function: This function should simulate the grid world
problem
% using the SARSA algorithm and output the optimal path taken by the
agent.
function main()
%7%
    % Implementation of the grid world problem using the SARSA
algorithm.
end

% evaluate() function: This function should import the stored data
and display the learning curve
% by plotting a line graph or using a suitable visualization method,
with the
% episode number on the x-axis and the total accumulated rewards on
the y-axis.
function evaluate()
%7%
    % Import the stored data and display the learning curve.
end
```

Use the following settings:

- The agent receives a reward of +10 when reaching the goal state and a reward of -1 for each step taken.
- The agent should explore the grid world using the epsilon-greedy strategy, where epsilon is the exploration rate.
- The agent's Q-table should be initialized with zeros and updated based on the SARSA update rule:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

  where s is the current state, a is the current action, r is the received reward, s' is the next state, a' is the next action, α is the learning rate, and γ is the discount factor.
- The agent should continue learning until it reaches the goal state or a maximum number of steps is reached.

Your task is to implement the `GridWorld` class, `SarsaAgent` class, and the `main()` function to solve the grid world problem using the SARSA algorithm. After solving the grid world problem, using `evaluate` function to present your training result.

Notes:

1. Ensure Code Readability: Pay attention to the readability of your code. Avoid writing complex, convoluted code that sacrifices clarity for brevity. Instead, strive for clear and understandable code by using meaningful variable and function names, adding comments where necessary, and structuring your code logically.

   PYTHON

   ```python
   # You should avoid coding in the style shown as below
   result = [x + y for x in range(1, 5) for y in range(6, 10) if (x
   + y) % 2 == 0]
   ```

2. Readability Impact on Score: Note that the readability of your code will have an impact on the score you receive for the assignment. Clear and well-organized code will be easier to understand and evaluate, contributing to a higher score.