Alexandria University

Faculty of Engineering

CSED 2025

Control Project Report

Submitted to

Prof. Dr. Prof. Ibrahim Abdelsalam

Eng. Omar Ahmed Wasfy

Submitted by

Ehab Yasser Mahmoud	**20010312**
Islam Yasser Mahmoud	**20010312**
Rowaina Abd-Elnasser Mohamed	**20010605**
Marwan Yasser Mahmoud	**20011870**
Mkario Michel Azer	**20011982**

Faculty of engineering Alex. University

May 2023

Table of Contents

Table of Figures

Problem Statement:

Part 1:

A Routh stability criterion to determine the stability of the systems. The system will be on the form of a polynomial equation, like s^5 + s^4 + 10s^3 + 72s^2 + 152s + 240, assuming that all the coefficients of s^0 to s^n are given. If the system is unstable, this will be printed in addition to the list of the number and values of poles in the right-hand side (RHS) of the s-plane will be printed in addition to the Routh table. And the system is stable, it will that in addition to the Routh table only.

Part 2:

A signal flow graph representation of a system besides:

- A graphical interface for the signal flow graph.
- Drawing the signal flow graph, including nodes, branches, and gains.
- Listing all forward paths, individual loops, and all combinations of n non-touching loops.
- Calculating the values of delta (Δ), delta1 (Δ1), and up to delta m (Δm), where m is the number of forward paths.
- Determining the overall system transfer function.

Features of the program:

Part 1:

Main Features:

- The input take operation is simple and easy just asking for the order of the system and the coefficients.
- The program checks if the signs of the coefficients are not the same in the beginning:
    1. **If so**, the program will state and print that the system isn't stable besides the RHS poles and their number.
    2. **If not,** the program will make a Routh table to know the state of the system whether it is stable or unstable and if the system is:
        - **Stable:** the program will print that it is Stable besides the Routh table.
        - **Unstable:** the program will print that it is unstable besides the RHS poles and their number and the Routh table.

Additional Features:

- **Checking the signs** of the coefficients in the beginning to know if the system is not stable from the beginning to save time.
- **Plotting all the poles** of the system to double check that the solution is right.
- The program determines whether the system is stable, Unstable or **Critically stable.**
- The system will be unstable if there are one of these 2 reasons:
    1. There are poles in the RHS. (If so, the program will print the system is Unstable)
    2. **There are repeated roots on the imaginary axis too.** (If so, **the system will** print the system is unstable and **will print the reason why it is unstable** as there are repeated roots on the imaginary axis)

Part 2:

Main Features:

- GUI Application.
- Drawing signal flow graph showing nodes, branches, and gains.

- Listing all forward paths, individual loops, and all combinations of n non-touching loops.
- Showing the values of Δ, Δ1, Δ1, …, Δm where m is number of forward paths.
- Overall system transfer function.

Additional Features:

- GUI Application using
    1. **PyQt5** for the main screen.
    2. **Tkinter** for graph window and the associated dialog box.
    3. Using **Canvas** library for graph drawing.
- Calculations can be performed both with **symbols** and **numbers.**
- Users can **choose the number of nodes** and **modify the input edges.**
- In the graph, on adding weight to nodes that already have weight, the added weight is added to the previous weight.
- The program has an **exe version.**
- Scroll area where the output is displayed is resizable (can be zoomed in and out).
- When creating an edge between two nodes that another edge already exists between, the value of the gain input to the new edge will be automatically added to the gain of the first edge.
- Error avoidance two cases: the number of nodes of source or destination is larger than the total number of nodes and when there is no forward path exists.

Used data structures:

Part 1:

- Normal **1D and 2D arrays (list in python)**

Part 2:

- **Graph:** Implemented as an adjacency list. The adjacency list is represented as a dict of nodes, inside of which is a dict of adjacent nodes, inside of which is the gain of each edge.
- **Python List** which is used everywhere in the algorithms.

Main Modules:

Part 1:

Part 2:

- **GUI:**
    1. **MainWindow.py** file for the main screen window.
    2. **Grapher.py** file for the Graph window.
- **Algorithms:**
    1. **Solver.py** file for solving the problem using Mason formula.

Algorithms used:

Part 1:

Check stability function:

Takes the coeff and check generates a Routh table to check the stability of the system printing the table and the state of the system in addition to plotting the poles.

Part 2:

Main Methods and Functions used:

- **find_forwardPaths method in MasonSolver class:**

The __find_forwardPaths function is responsible for finding all possible forward paths between the given start and end nodes. The function is implemented using a recursive depth-first search algorithm.

The algorithm starts by initializing an empty list path and appending the start_node to it. It then checks if the start_node is the same as the end_node. If this is the case, it returns a list containing a single ForwardPath object with the current path and a gain of an empty string. If the start_node is not the same as the end_node, the function checks if the start_node is in the adjacency list. If it is not, it returns an empty list.

If the start_node is in the adjacency list, the function loops through all of the neighbors of the start_node. For each neighbor, the function checks if the neighbor is already in the path. If it is not, it calls the __find_forwardPaths function recursively with the neighbor as the new start_node, the end_node and the current path.

The gain for each new path is calculated by multiplying the gain of the current path with the weight of the edge between the start_node and the neighbor. This is achieved by appending the edge weight to the current path gain. Once all possible forward paths have been found, the function returns a list of ForwardPath objects containing the path and the gain for each path.

- **find_loops method in MasonSolver class:**

The __find_loops function is responsible for finding all possible loops in the given signal flow graph. The function is implemented using a modified depth-first search algorithm. The algorithm starts by initializing an empty list loops to store all the loops that are found. It also initializes an empty set visited to keep track of nodes that have already been visited during the search. Finally, it initializes an empty list path to store the current path being explored. The algorithm then loops through all the nodes in the adjacency list. For each node, it calls the getLoopsHelper function, which is a recursive helper function that performs the actual depth-first search. The getLoopsHelper function takes the current node, the visited set, the start node of the current path, the current path, and the loops list as arguments. The getLoopsHelper function first checks if the current node has already been visited. If it has, it means that a loop has been found, so the function creates a new Loop object with the current path and appends it to the loops list. If the current node has not been visited, the function adds it to the visited set and loops through all of its neighbors. For each neighbor, the function checks if the neighbor is the same as the start node of the current path. If it is, it means that a loop has been found, so the function creates a new Loop object with the current path and appends it to the loops list. If the neighbor is not the start node, the function calls itself recursively with the neighbor as the new current node, the visited set, the start node of the current path, the current path with the neighbor appended to it, and the loops list. Once all possible loops have been found, the function filters the loops list to remove any duplicates or loops.

- **generate_nonTouchingLoops method in MasonSolver class:**

The __generate_nonTouchingLoops() method is used to generate all non-touching loops in a given graph. It takes a list of loops as input and returns a list of non-touching loops. Non-touching loops are defined as loops that do not share any nodes with each other. The method first creates a list of all possible combinations of loops. It then iterates through each combination and checks if the loops are non-touching. To check if two loops are non-touching, it compares the nodes in each loop and checks if they have any nodes in common. If there are no common nodes, the two loops are non-touching and are added to the list of non-touching loops. The method then repeats this process for all combinations of loops, ensuring that all non-touching loops are generated. Finally, the list of non-touching loops is returned. Overall, the __generate_nonTouchingLoops() method uses a combination of loop iteration and comparison to generate all non-touching loops in a given graph.

Additional Libraries:

Part 1:

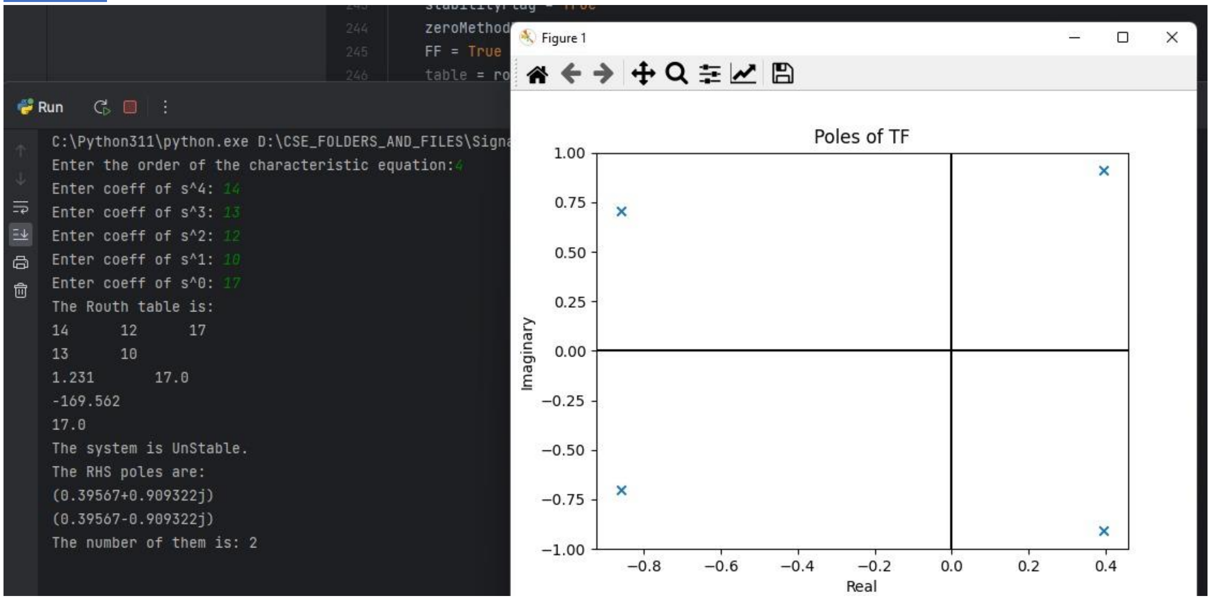
- **Numpy:** used to get the roots of the equation.
- **Sumpy:** used to evaluates limits.
- **Matplotlib.pyplot:** used to plot the poles of the equation.

Part 2:

- **Sympy:** Used in simplifying symbolic and numerical expressions when evaluating the ∆s for the system and each forward path, the gain of each forward path and loop, and the overall transfer function.

Sample runs:

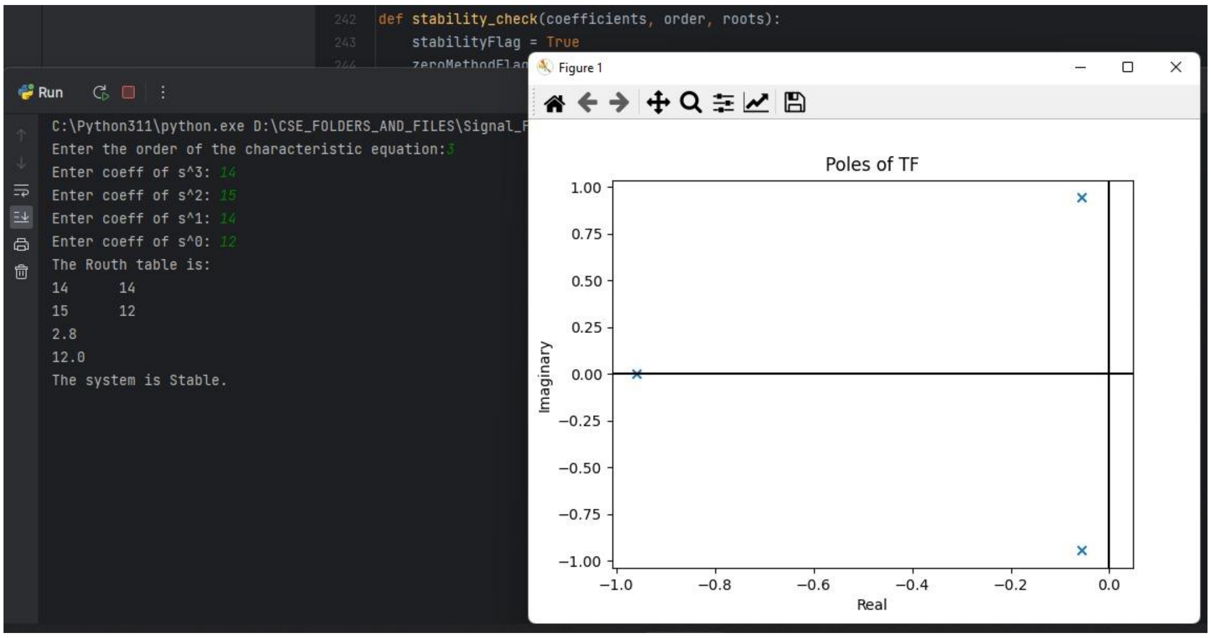
Part 1:



```
zeroMethod
FF = True
table = ro
```

```
C:\Python311\python.exe D:\CSE_FOLDERS_AND_FILES\Signa
Enter the order of the characteristic equation:4
Enter coeff of s^4: 14
Enter coeff of s^3: 13
Enter coeff of s^2: 12
Enter coeff of s^1: 10
Enter coeff of s^0: 17
The Routh table is:
14      12      17
13      10
1.231      17.0
-169.562
17.0
The system is UnStable.
The RHS poles are:
(0.39567+0.909322j)
(0.39567-0.909322j)
The number of them is: 2
```

Figure 0-1: Routh Test case 1



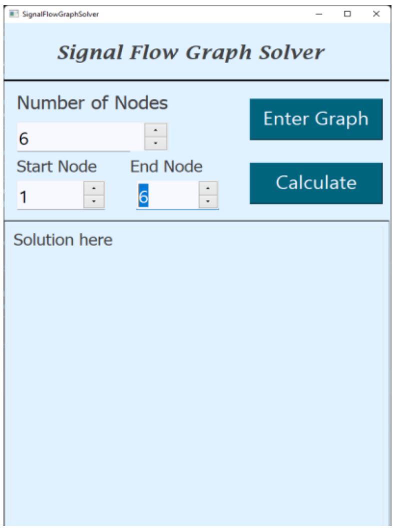
```
def stability_check(coefficients, order, roots):
    stabilityFlag = True
    zeroMethodFlag
```

```
C:\Python311\python.exe D:\CSE_FOLDERS_AND_FILES\Signal_F
Enter the order of the characteristic equation:3
Enter coeff of s^3: 14
Enter coeff of s^2: 15
Enter coeff of s^1: 14
Enter coeff of s^0: 12
The Routh table is:
14      14
15      12
2.8
12.0
The system is Stable.
```

Figure 0-2: Routh Test case 2

Part 2:

- User chooses the number of nodes to be inserted, start node, and the end node, then clicks on calculate.
- The graph window then appears and edges and weights are inserted.
- Close the graph window then click on the calculate button to solve the signal flow graph.



SignalFlowGraphSolver

Signal Flow Graph Solver

Number of Nodes: 6

Start Node: 1 End Node: 6

Solution here

Figure 0-4: Initial Screen

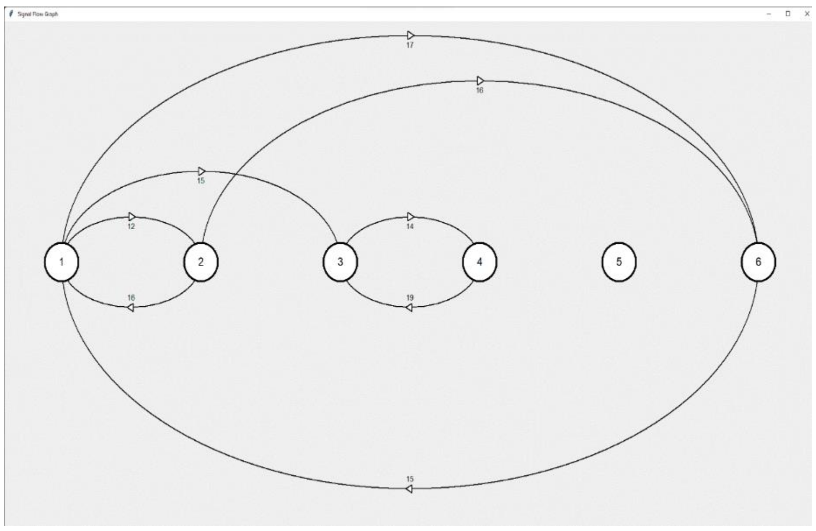
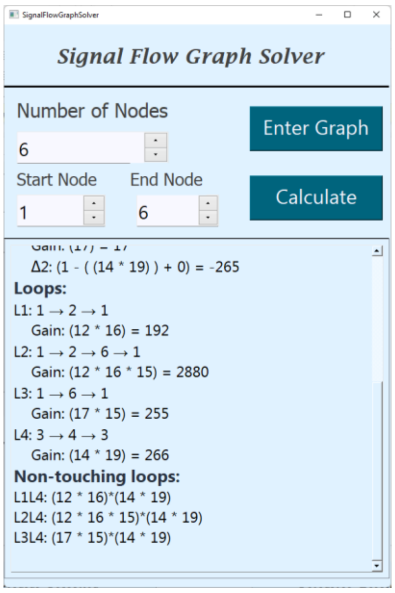
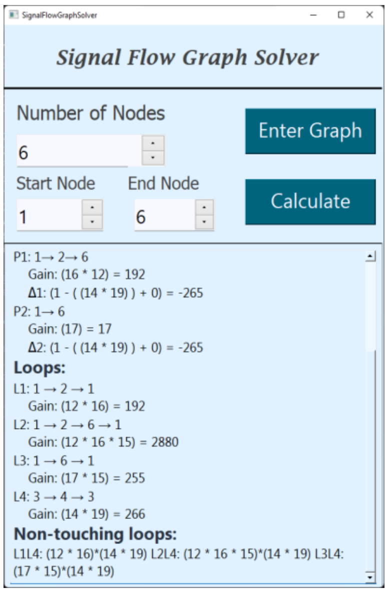


Figure 0-3: Graph Screen



```
6
1    6
    Gain: (17) = 17
    Δ2: (1 - ( (14 * 19) ) + 0) = -265
Loops:
L1: 1 → 2 → 1
    Gain: (12 * 16) = 192
L2: 1 → 2 → 6 → 1
    Gain: (12 * 16 * 15) = 2880
L3: 1 → 6 → 1
    Gain: (17 * 15) = 255
L4: 3 → 4 → 3
    Gain: (14 * 19) = 266
Non-touching loops:
L1L4: (12 * 16)*(14 * 19)
L2L4: (12 * 16 * 15)*(14 * 19)
L3L4: (17 * 15)*(14 * 19)
```

Figure 0-6: Outputs 2



```
6
1    6
P1: 1→ 2→ 6
    Gain: (16 * 12) = 192
    Δ1: (1 - ( (14 * 19) ) + 0) = -265
P2: 1→ 6
    Gain: (17) = 17
    Δ2: (1 - ( (14 * 19) ) + 0) = -265
Loops:
L1: 1 → 2 → 1
    Gain: (12 * 16) = 192
L2: 1 → 2 → 6 → 1
    Gain: (12 * 16 * 15) = 2880
L3: 1 → 6 → 1
    Gain: (17 * 15) = 255
L4: 3 → 4 → 3
    Gain: (14 * 19) = 266
Non-touching loops:
L1L4: (12 * 16)*(14 * 19) L2L4: (12 * 16 * 15)*(14 * 19) L3L4:
(17 * 15)*(14 * 19)
```

Figure 0-5: Outputs 1

Simple user guide:

Part 1:

- Open the Routh.py file.
- Press on run.
- Write the order of the equation on the console.
- Write the coefficient of each variable.

Part 2:

How to run:

There are 3 methods:

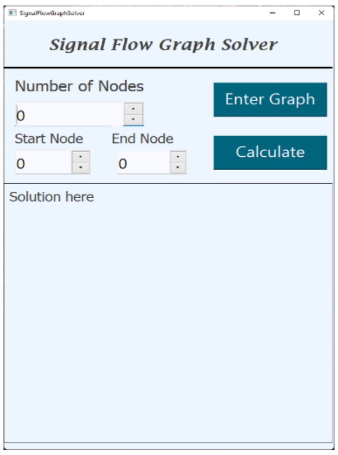
- Simply open the project in an IDE and run the MainWindow.py file.
- Go to the directory dist where you will find an executable version of the program.
- In the terminal, from the directory of the project type python MainWindow.py.

To form the exe file from the current code, type these commands:

- pip install --ignore-installed pyinstaller --user
- pyinstaller --onefile MainWindow.py
- pyinstaller MainWindow.spec

The last one is the one run to reform the already generated file on changing the code.

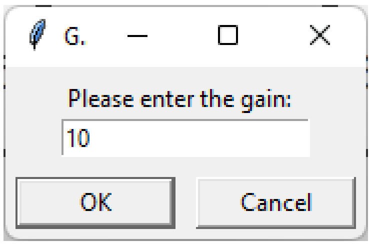
Program components:



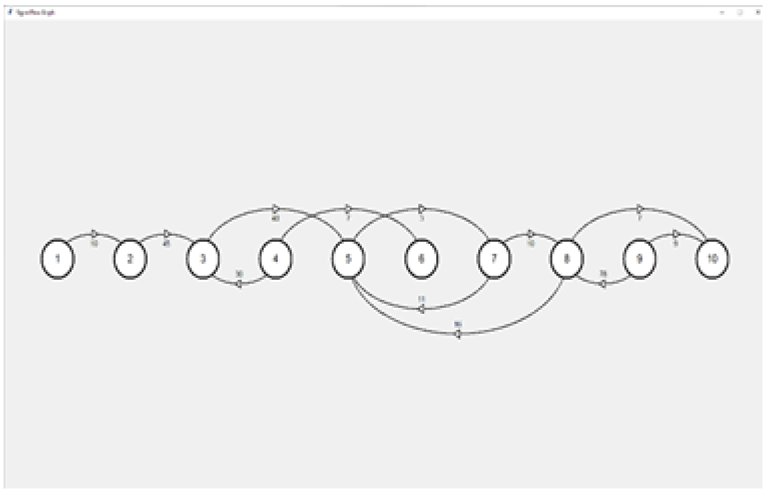
```
0
0    0
Solution here
```

As shown, the main window contains:

- three spin boxes:
  1. The upper specifies the number of nodes (Maximum = 12)
  1. The lower left for the start node.
  2. The lower right for the end node.
- Two buttons:
  1. Enter Graph: Opens the graph window.
  2. Calculate: Performs calculation on the formed graph and shows the solution.



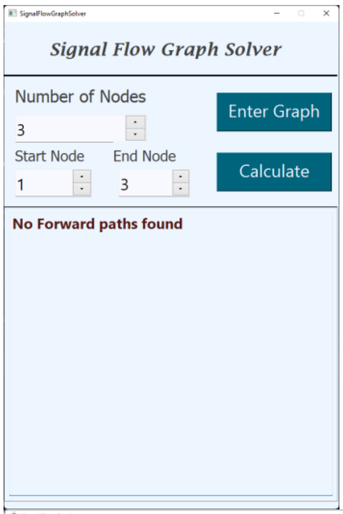
Please enter the gain:
10



In Graph window:

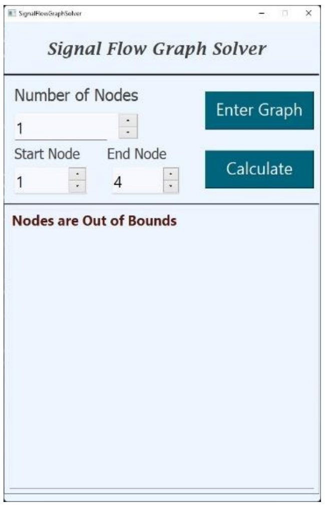
- Nodes generated automatically.
- To enter weight:
  1. Click on the from-node.
  2. Click on the to-node
  3. Dialog box appears to enter weight.

- On clicking on two nodes already connected, the entered weight is added to the existing weight.
- As mentioned, the errors are handled, and the following appears on clicking on Calculate button:



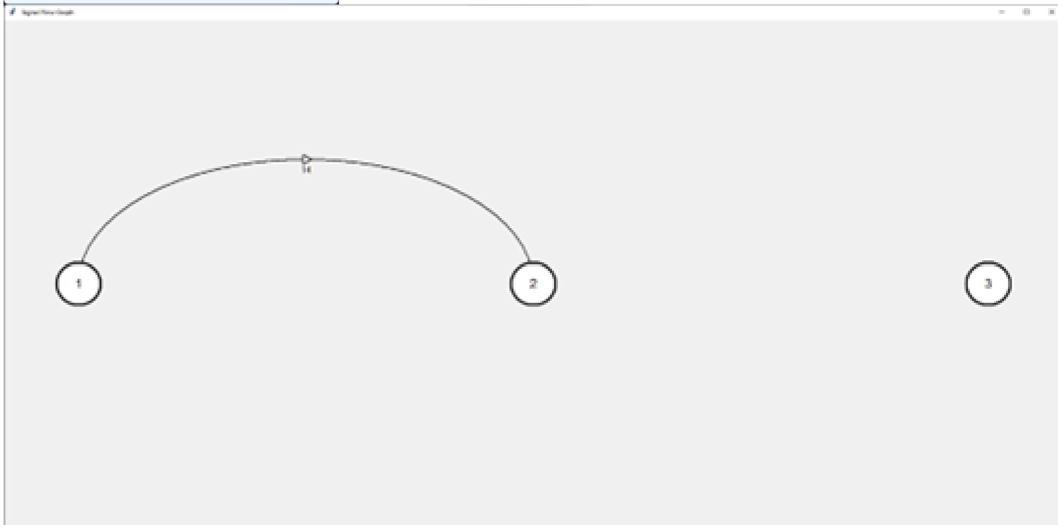
Number of Nodes: 3
Start Node: 1
End Node: 3

No Forward paths found



Number of Nodes: 1
Start Node: 1
End Node: 4

Nodes are Out of Bounds



Attachments:

- GitHub Repository containing the source code.
- Video Link