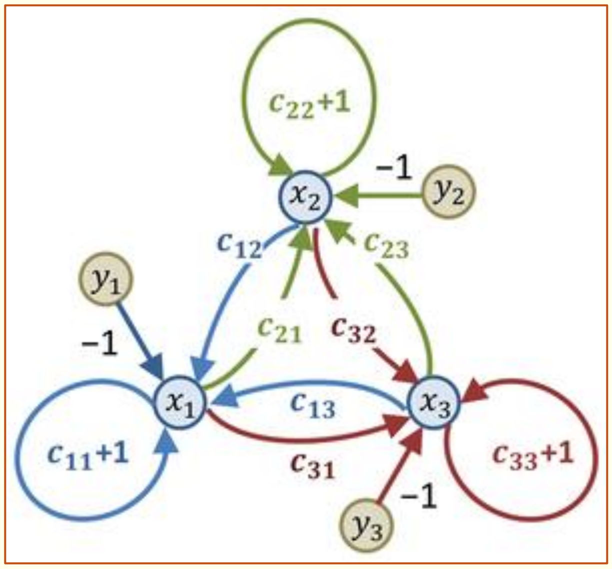


CONTROL SYSTEMS

Programming Assignment

TEAM MEMBERS

Ahmed Mustafa Elmorsi Amer	21010189
Ahmed Ayman Ahmed Abdalla	21010048
Ahmed Youssef Sobhy Elgoerany	21010217
Ebrahim Alaa El-din Ebrahim	21010017
Abdallah Adel Ahmed Mohamed	21010808
Moustafa Esam El-Sayed Amer	21011364

Code

GitHub Repository Link

Contents

Signal Flow Graph

Problem Statement

Given

Signal flow graph representation of the system. Under the assumptions that the number of nodes and branches are given, besides all nodes must be connected.

Assumptions

- There must be exactly one input node and exactly one output node in the signal flow graph.
- Multi-edges, representing multiple edges with the same direction between the same pair of nodes, are simplified into a single edge with the summation of all the multi-edges' gains when performing calculations.
- The input graph must be connected.
- Weights of the branches must be a number (floating or integer).

Required

1- Graphical interface.
2- Draw the signal flow graph showing nodes, branches, gains, …
3- Listing all forward paths, individual loops, all combination of n non-touching loops.
4- The values of Δ, Δ_1 , …, Δ_m where m is number of forward paths.
5- Overall system transfer function.

Main Features

This Signal Flow Graph (SFG) program is designed to analyze and calculate various properties of a signal flow graph. It provides functionalities to find loops, forward paths, calculate gains, and determine the overall transfer function of the system represented by the graph.

Its interactive graphical user interface (GUI) provides a platform to visualize and solve signal flow graphs. It enables users to dynamically create, modify, and freely move the graph, facilitating easy analysis of signal flow graphs.

- **Graphical Representation:** Visualize signal flow graphs using an interactive graph visualization tool.
- **Add Nodes:** Dynamically add nodes with any unique name.

- **Add Edges:** Dynamically add edges (branches) with gain values.
- **Delete Nodes:** Remove all selected nodes from the graph along with their associated edges.
- **Solver:** Solve the drawn signal flow graph to obtain:
    - **Loop Analysis:**
        - Identify loops within the graph.
        - Calculate gains associated with each loop.
    - **Forward Path Analysis:**
        - Determine all possible forward paths.
        - Compute gains for each forward path.
        - Compute delta for each forward path. (Δ1, Δ2, …., Δm) where m is the number of forward paths.
    - **Non-Touching Loop Analysis:**
        - Identify sets of all N combinations of non-touching loops.
        - Determine their respective gains.
    - **Input and Output Nodes:**
        - Find the input and output nodes of the system, computed through calculating indegrees and outdegrees of the graph nodes.
    - **Overall Delta Value (Δ):**
        - Calculate the overall delta (Δ) of the system.
    - **Overall Transfer Function:**
        - Calculate the overall transfer function of the system.

Data Structures & Algorithms

Part 1 is implemented using Python.

Data Structures

- **Graph Representation:** Signal flow graphs are represented using **adjacency lists/dictionaries**, where nodes are keys, and their outgoing edges are stored as lists of tuples containing target nodes and edge weights.

- **Path and Loop Storage:** Forward paths, loops, and non-touching loops are stored as **lists** of node sequences or sets of node sets, allowing for efficient traversal and analysis.
- **Dash Module** Used for the GUI part of the code.

Algorithms

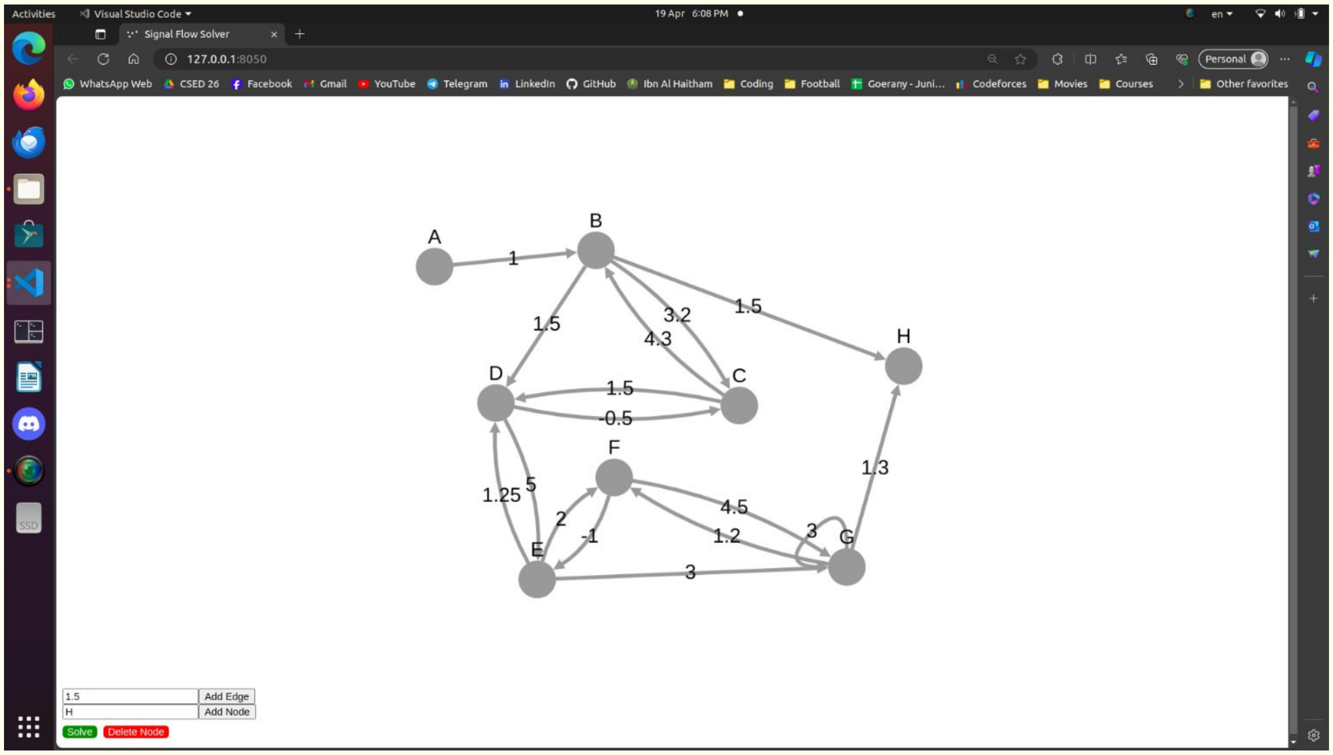
- **Depth First Search (DFS)** Used not only to find loops in the graph but also to discover forward paths. DFS is applied recursively to traverse the graph, identifying cycles as loops, and recording distinct paths from the input to the output nodes.
- **Combinations** Utilized to find all combinations of non-touching loops.
- **Calculation of Gains & Delta** Algorithms are implemented to calculate loop gains, forward path gains, and overall delta of the system.
- **Overall Transfer Function** Mason's Gain Formula for calculating the overall transfer function of the system is implemented.

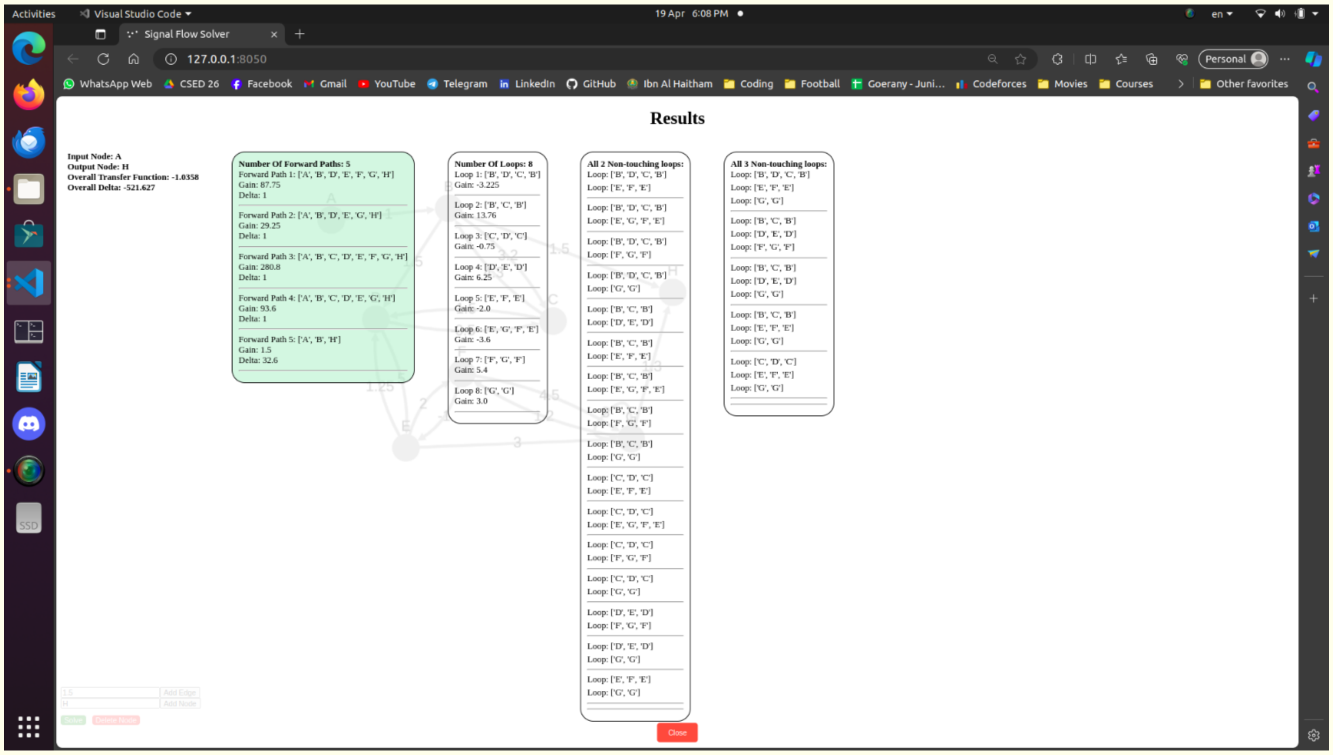
Main Modules

- **SignalFlowGraph Class**
  This class encapsulates all functionalities related to analyzing the signal flow graph. It includes methods for finding loops, forward paths, calculating gains, determining the overall transfer function and more.
- **Dash Interactive GUI Module**
  This module includes the GUI display functions and their implementation, utilizing python Dash Module, which is a Python framework for building web applications. It is built on top of Flask, Plotly. js, React and React Js. It enables you to build dashboards using pure Python.
  This module also utilizes Dash Cytoscape , which is a graph visualization component for creating easily customizable, high-performance, interactive, and web-based networks. It extends and renders Cytoscape.

Sample Runs

- Test Case #1:





Results

Input Node: A
Output Node: H
Overall Transfer Function: -1.0358
Overall Delta: -521.627

Number Of Forward Paths: 5
Forward Path 1: ['A', 'B', 'D', 'E', 'F', 'G', 'H']
Gain: 87.75
Delta: 1

Forward Path 2: ['A', 'B', 'D', 'E', 'G', 'H']
Gain: 29.25
Delta: 1

Forward Path 3: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
Gain: 280.8
Delta: 1

Forward Path 4: ['A', 'B', 'C', 'D', 'E', 'G', 'H']
Gain: 93.6
Delta: 1

Forward Path 5: ['A', 'B', 'H']
Gain: 1.5
Delta: 32.6

Number Of Loops: 8
Loop 1: ['B', 'D', 'C', 'B']
Gain: -3.225

Loop 2: ['B', 'C', 'B']
Gain: 13.76

Loop 3: ['C', 'D', 'C']
Gain: -0.75

Loop 4: ['D', 'E', 'D']
Gain: 6.25

Loop 5: ['E', 'F', 'E']
Gain: -2.0

Loop 6: ['E', 'G', 'F', 'E']
Gain: -3.6

Loop 7: ['F', 'G', 'F']
Gain: 5.4

Loop 8: ['G', 'G']
Gain: 3.0

All 2 Non-touching loops:
Loop: ['B', 'D', 'C', 'B']
Loop: ['E', 'F', 'E']

Loop: ['B', 'D', 'C', 'B']
Loop: ['E', 'G', 'F', 'E']

Loop: ['B', 'D', 'C', 'B']
Loop: ['F', 'G', 'F']

Loop: ['B', 'D', 'C', 'B']
Loop: ['G', 'G']

Loop: ['B', 'C', 'B']
Loop: ['D', 'E', 'D']

Loop: ['B', 'C', 'B']
Loop: ['E', 'F', 'E']

Loop: ['B', 'C', 'B']
Loop: ['E', 'G', 'F', 'E']

Loop: ['B', 'C', 'B']
Loop: ['F', 'G', 'F']

Loop: ['B', 'C', 'B']
Loop: ['G', 'G']

Loop: ['C', 'D', 'C']
Loop: ['E', 'F', 'E']

Loop: ['C', 'D', 'C']
Loop: ['E', 'G', 'F', 'E']

Loop: ['C', 'D', 'C']
Loop: ['F', 'G', 'F']

Loop: ['C', 'D', 'C']
Loop: ['G', 'G']

Loop: ['D', 'E', 'D']
Loop: ['F', 'G', 'F']

Loop: ['D', 'E', 'D']
Loop: ['G', 'G']

Loop: ['E', 'F', 'E']
Loop: ['G', 'G']

All 3 Non-touching loops:
Loop: ['B', 'D', 'C', 'B']
Loop: ['E', 'F', 'E']
Loop: ['G', 'G']

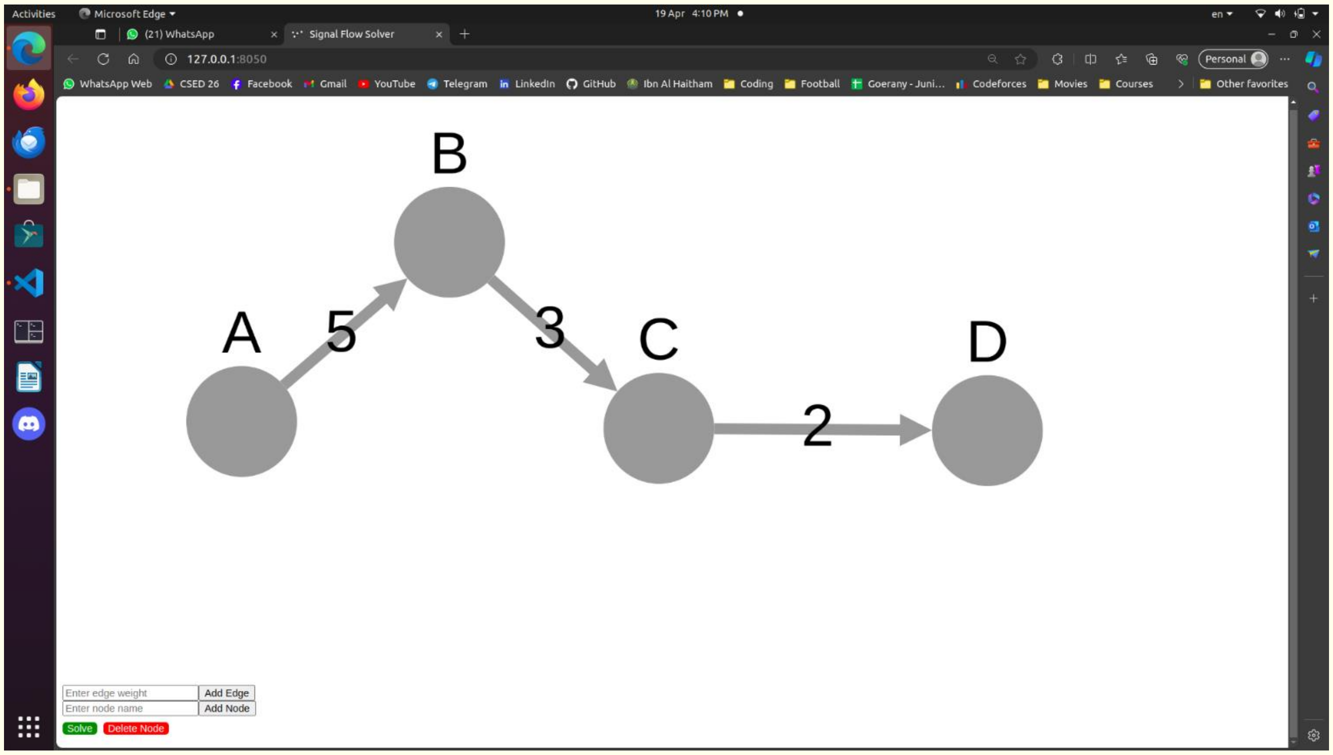
Loop: ['B', 'C', 'B']
Loop: ['D', 'E', 'D']
Loop: ['F', 'G', 'F']

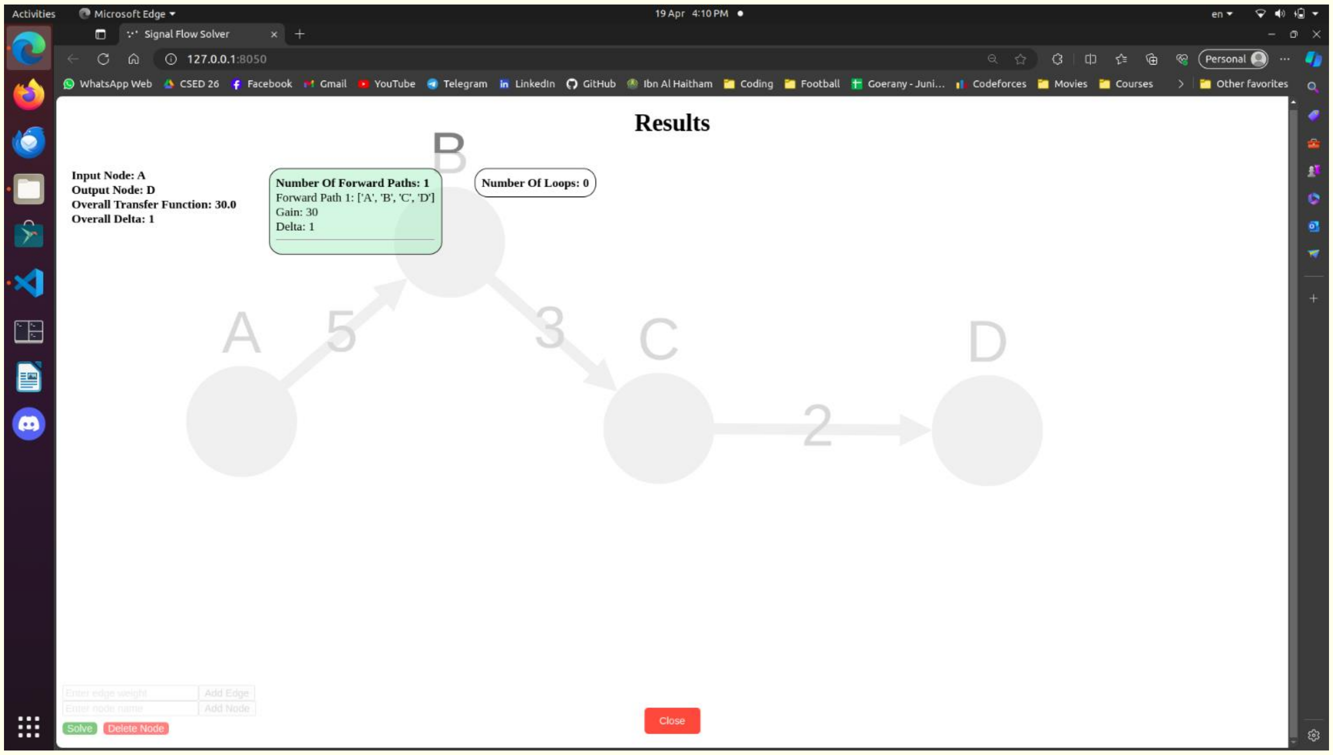
Loop: ['B', 'C', 'B']
Loop: ['D', 'E', 'D']
Loop: ['G', 'G']

Loop: ['B', 'C', 'B']
Loop: ['E', 'F', 'E']
Loop: ['G', 'G']

Loop: ['C', 'D', 'C']
Loop: ['E', 'F', 'E']
Loop: ['G', 'G']

- Test Case #2:



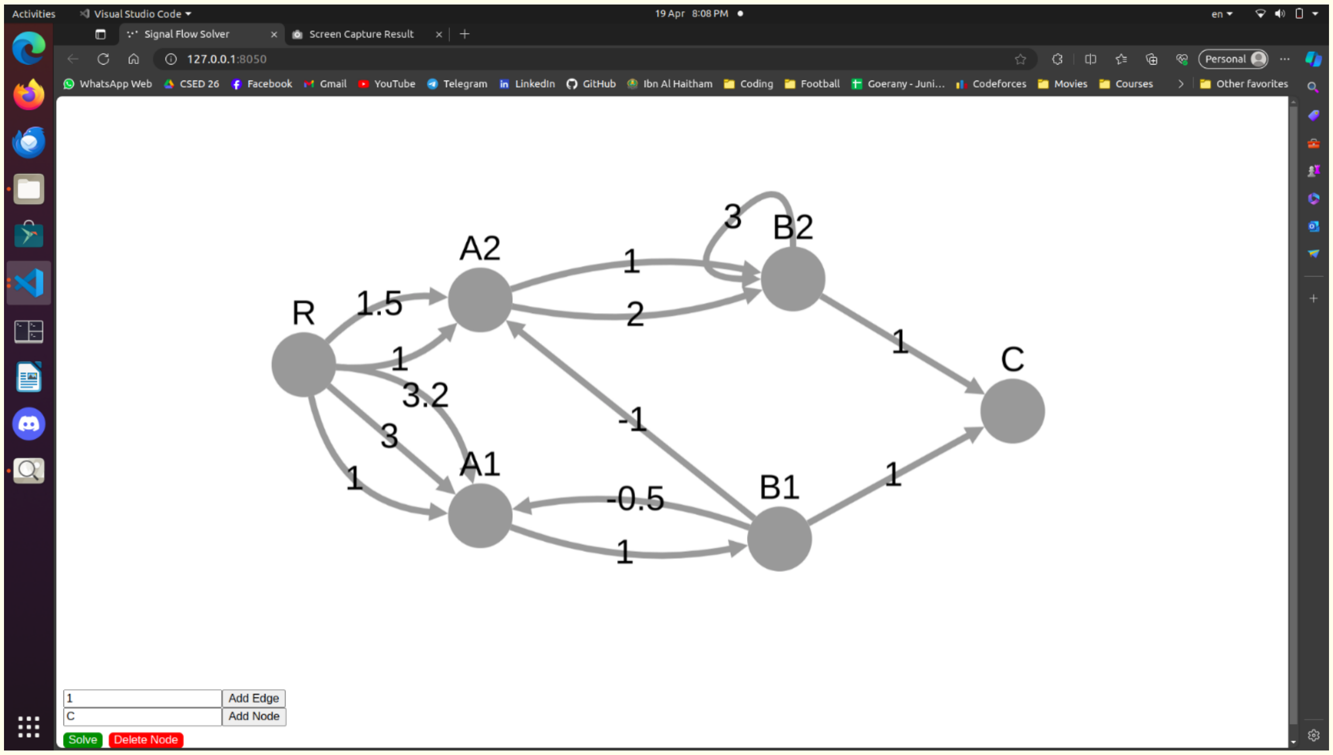

Results

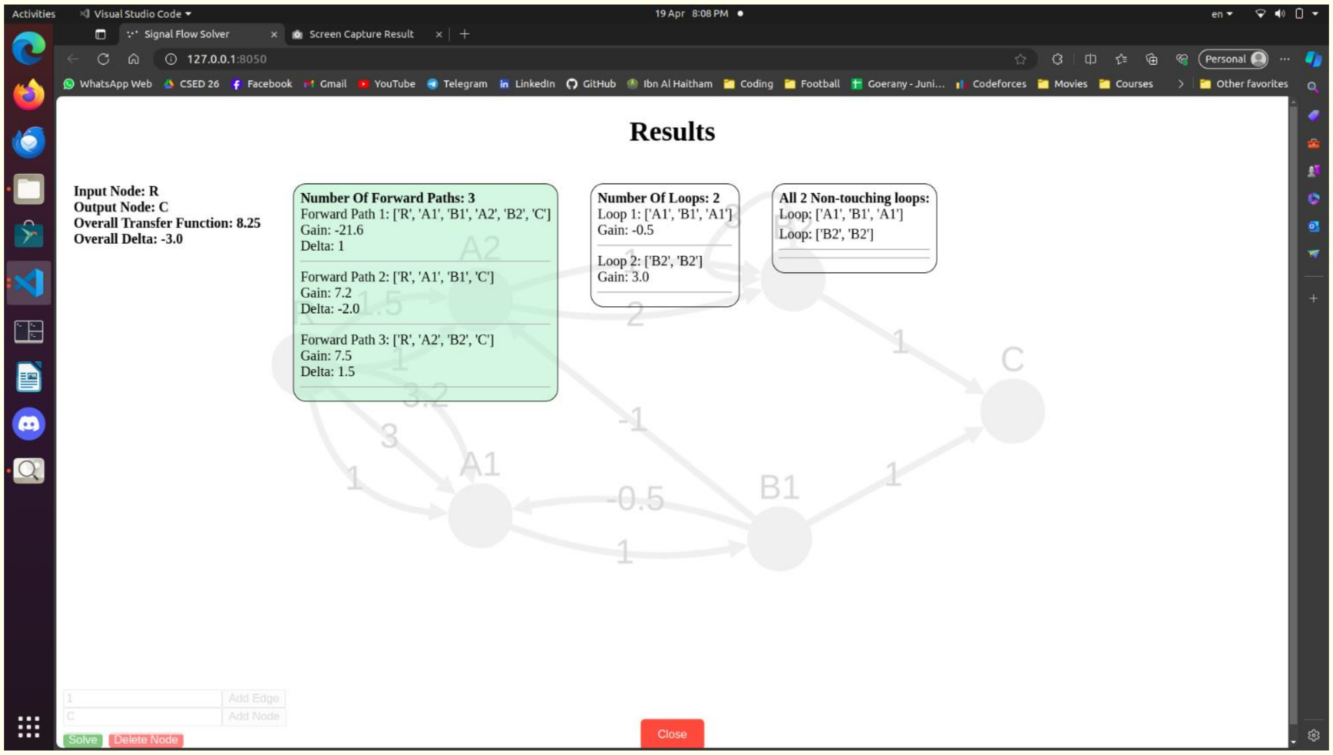
Input Node: A
Output Node: D
Overall Transfer Function: 30.0
Overall Delta: 1

Number Of Forward Paths: 1
Forward Path 1: ['A', 'B', 'C', 'D']
Gain: 30
Delta: 1

Number Of Loops: 0

- Test Case #3: (Illustrating how we handle multi-edges)





Results

Input Node: R
Output Node: C
Overall Transfer Function: 8.25
Overall Delta: -3.0

Number Of Forward Paths: 3
Forward Path 1: ['R', 'A1', 'B1', 'A2', 'B2', 'C']
Gain: -21.6
Delta: 1

Forward Path 2: ['R', 'A1', 'B1', 'C']
Gain: 7.2
Delta: -2.0

Forward Path 3: ['R', 'A2', 'B2', 'C']
Gain: 7.5
Delta: 1.5

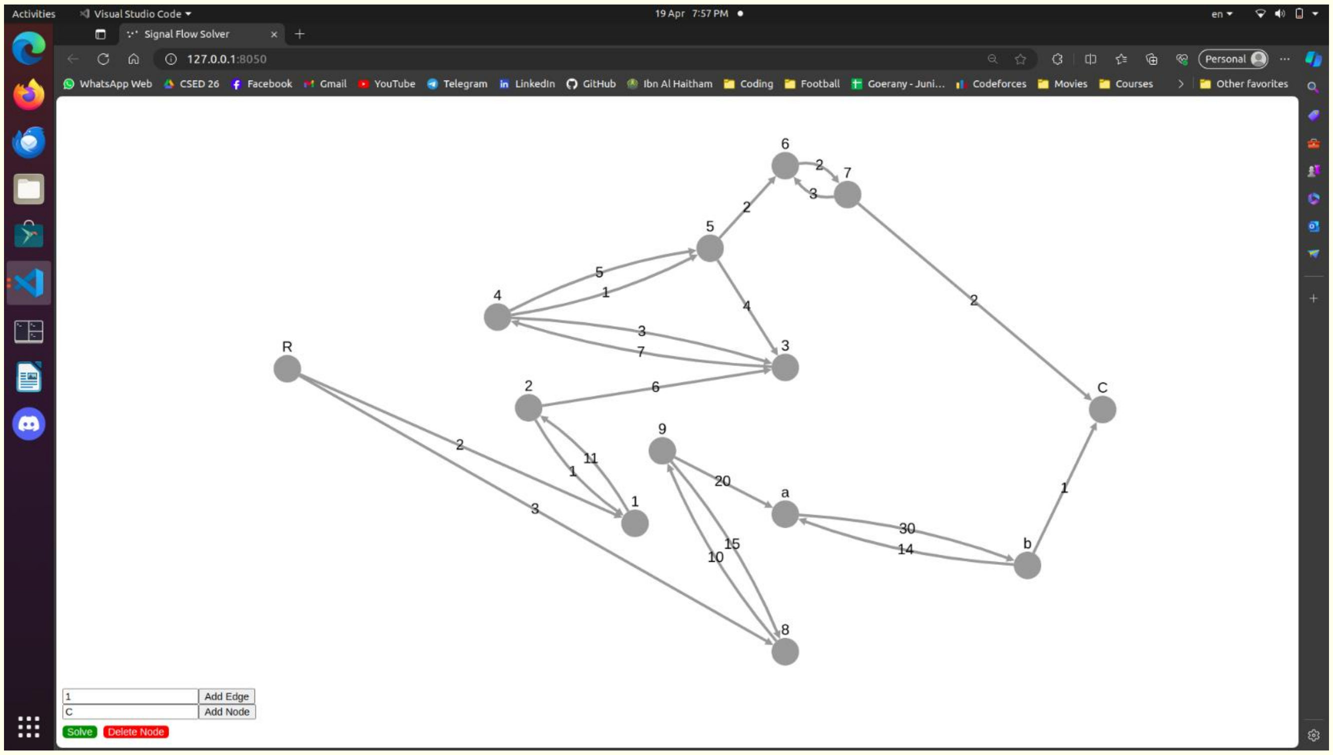
Number Of Loops: 2
Loop 1: ['A1', 'B1', 'A1']
Gain: -0.5

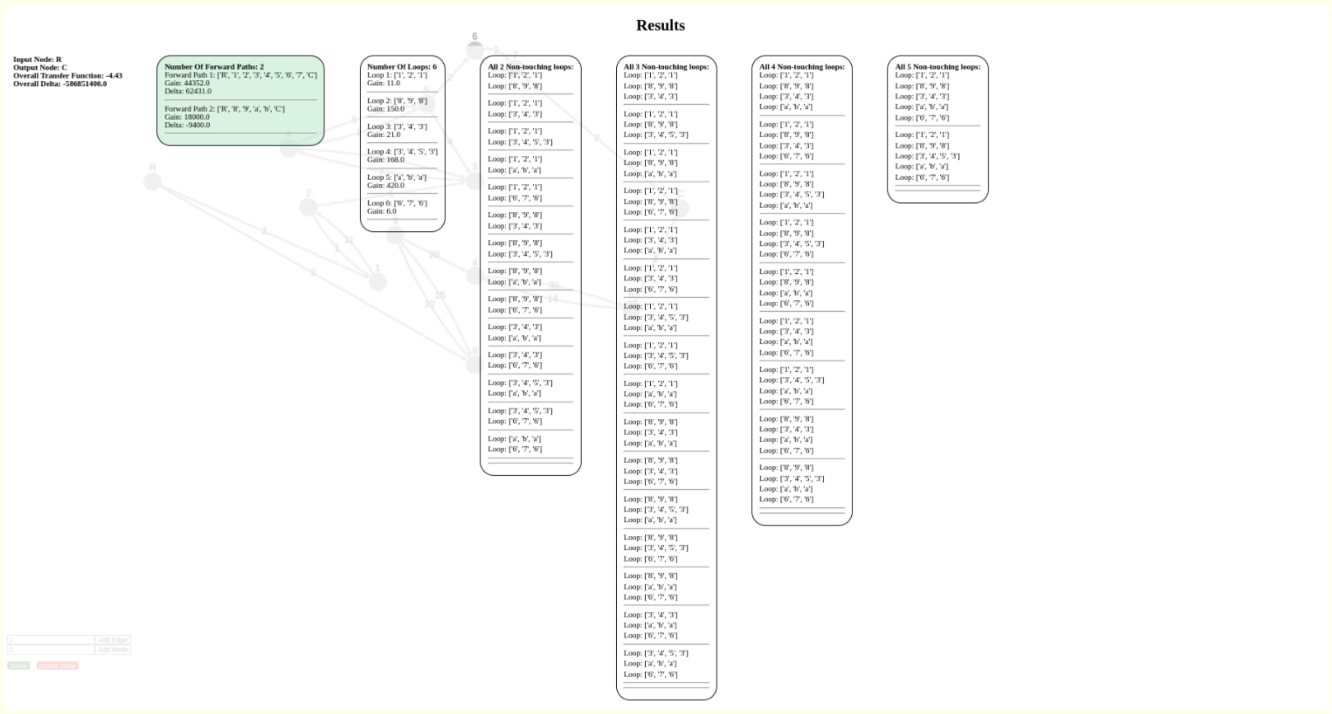
Loop 2: ['B2', 'B2']
Gain: 3.0

All 2 Non-touching loops:
Loop: ['A1', 'B1', 'A1']
Loop: ['B2', 'B2']

Close

- Test Case #4:





Results

Input Node: R
Output Node: C
Overall Transfer Function: -4.43
Overall Delta: -586851400.0

Number Of Forward Paths: 2
Forward Path 1: ['R', '1', '2', '3', '4', '5', '6', '7', 'C']
Gain: 44352.0
Delta: 62431.0

Forward Path 2: ['R', '8', '9', 'a', 'b', 'C']
Gain: 18000.0
Delta: -9400.0

Number Of Loops: 6
Loop 1: ['1', '2', '1']
Gain: 11.0

Loop 2: ['8', '9', '8']
Gain: 150.0

Loop 3: ['3', '4', '3']
Gain: 21.0

Loop 4: ['3', '4', '5', '3']
Gain: 168.0

Loop 5: ['a', 'b', 'a']
Gain: 420.0

Loop 6: ['6', '7', '6']
Gain: 6.0

All 2 Non-touching loops:
Loop: ['1', '2', '1']
Loop: ['8', '9', '8']

Loop: ['1', '2', '1']
Loop: ['3', '4', '3']

Loop: ['1', '2', '1']
Loop: ['3', '4', '5', '3']

Loop: ['1', '2', '1']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '3']

Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']

Loop: ['8', '9', '8']
Loop: ['a', 'b', 'a']

Loop: ['8', '9', '8']
Loop: ['6', '7', '6']

Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

Loop: ['3', '4', '3']
Loop: ['6', '7', '6']

Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']

Loop: ['3', '4', '5', '3']
Loop: ['6', '7', '6']

Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

All 3 Non-touching loops:
Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['3', '4', '3']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['3', '4', '5', '3']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']

Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

All 4 Non-touching loops:
Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

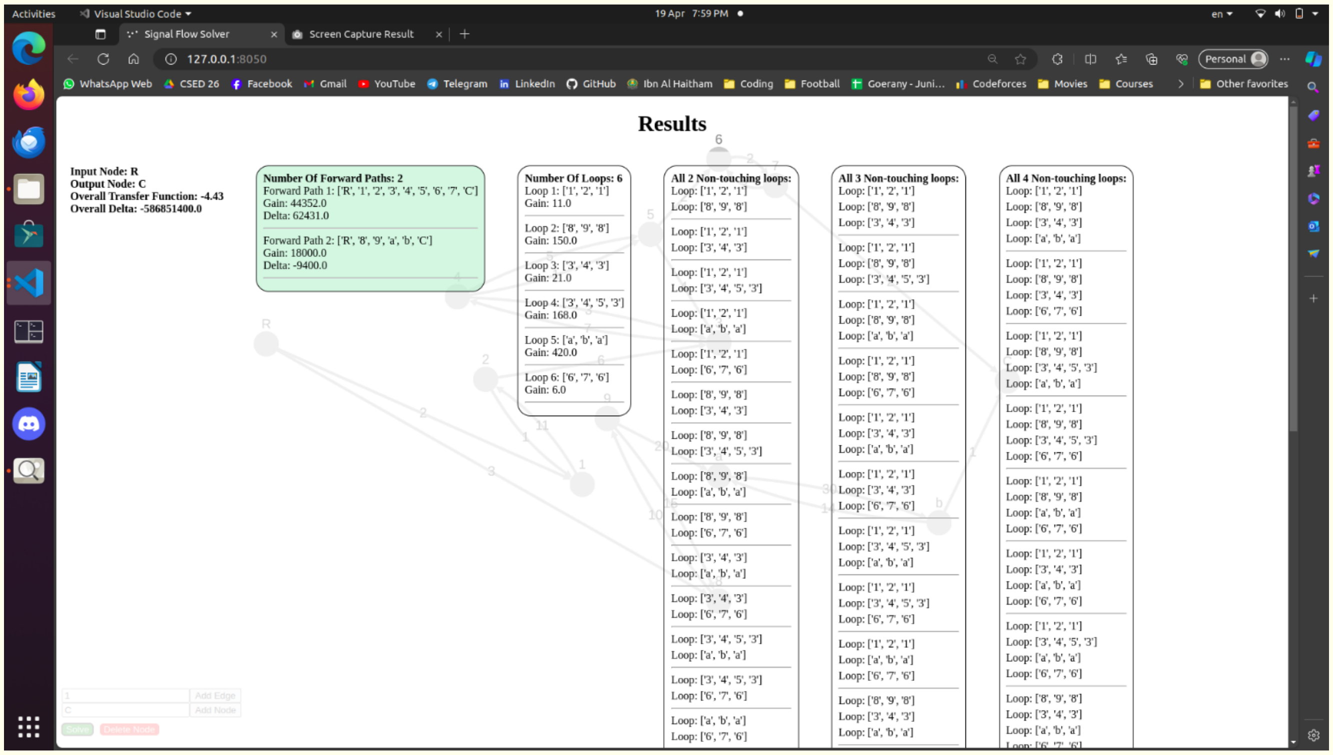
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

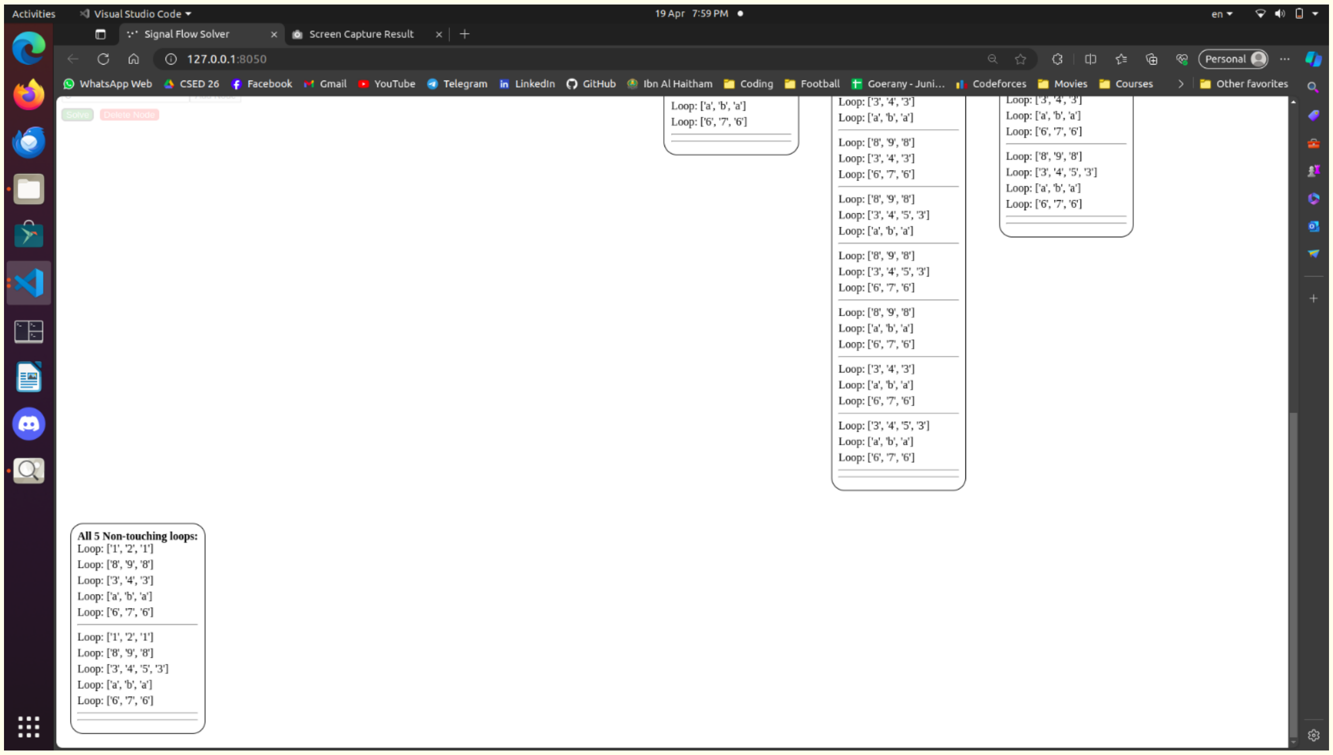
All 5 Non-touching loops:
Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

The last picture but in higher quality (divided into two screenshots)



```
Results

Input Node: R
Output Node: C
Overall Transfer Function: -4.43
Overall Delta: -586851400.0

Number Of Forward Paths: 2
Forward Path 1: ['R', '1', '2', '3', '4', '5', '6', '7', 'C']
Gain: 44352.0
Delta: 62431.0

Forward Path 2: ['R', '8', '9', 'a', 'b', 'C']
Gain: 18000.0
Delta: -9400.0

Number Of Loops: 6
Loop 1: ['1', '2', '1']
Gain: 11.0

Loop 2: ['8', '9', '8']
Gain: 150.0

Loop 3: ['3', '4', '3']
Gain: 21.0

Loop 4: ['3', '4', '5', '3']
Gain: 168.0

Loop 5: ['a', 'b', 'a']
Gain: 420.0

Loop 6: ['6', '7', '6']
Gain: 6.0

All 2 Non-touching loops:
Loop: ['1', '2', '1']
Loop: ['8', '9', '8']

Loop: ['1', '2', '1']
Loop: ['3', '4', '3']

Loop: ['1', '2', '1']
Loop: ['3', '4', '5', '3']

Loop: ['1', '2', '1']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '3']

Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']

Loop: ['8', '9', '8']
Loop: ['a', 'b', 'a']

Loop: ['8', '9', '8']
Loop: ['6', '7', '6']

Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

Loop: ['3', '4', '3']
Loop: ['6', '7', '6']

Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']

Loop: ['3', '4', '5', '3']
Loop: ['6', '7', '6']

Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

All 3 Non-touching loops:
Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['3', '4', '3']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['3', '4', '5', '3']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

All 4 Non-touching loops:
Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']
```



```
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']

Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']

Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

All 5 Non-touching loops:
Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']

Loop: ['1', '2', '1']
Loop: ['8', '9', '8']
Loop: ['3', '4', '5', '3']
Loop: ['a', 'b', 'a']
Loop: ['6', '7', '6']
```

Simple User Guide

- **Installation**
  1. Clone this repository to your local machine:

```
git clone https://github.com/ahmedyoussefg/SignalFlowGraph-And-Routh.git
```

  2. Navigate to the project directory:

```
cd SignalFlowGraph-And-Routh
```

  3. Install the required dependencies:

```
pip install -r requirements.txt
```

  4. Run the application:

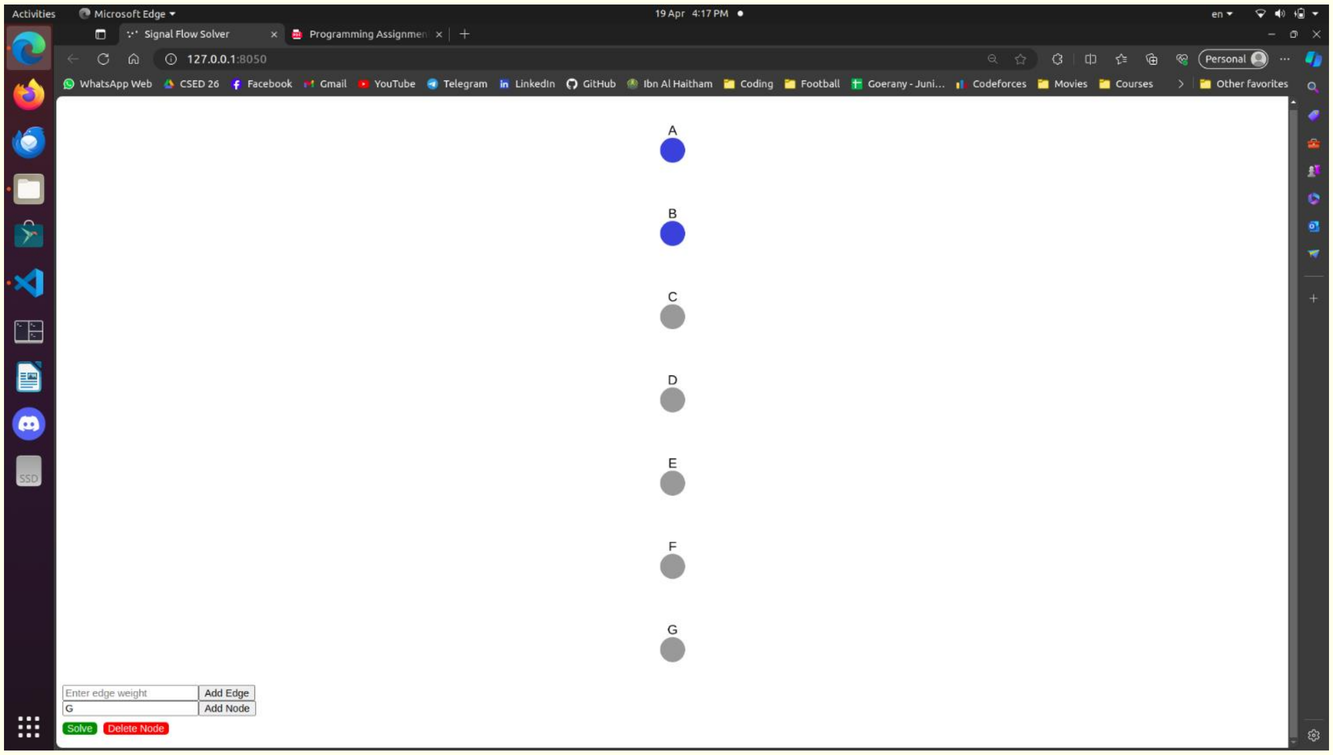
```
python3 dash_interactive_gui.py
```

  5. Access the application GUI by opening a web browser and navigating to *http://127.0.0.1:8050*.

- **How To Use**
  - **User Guide Video Link:** **https://drive.google.com/file/d/1YHF4xYRUhUy3GThPO1i_8fbL0PY5LGIC/view?usp=drive_link**

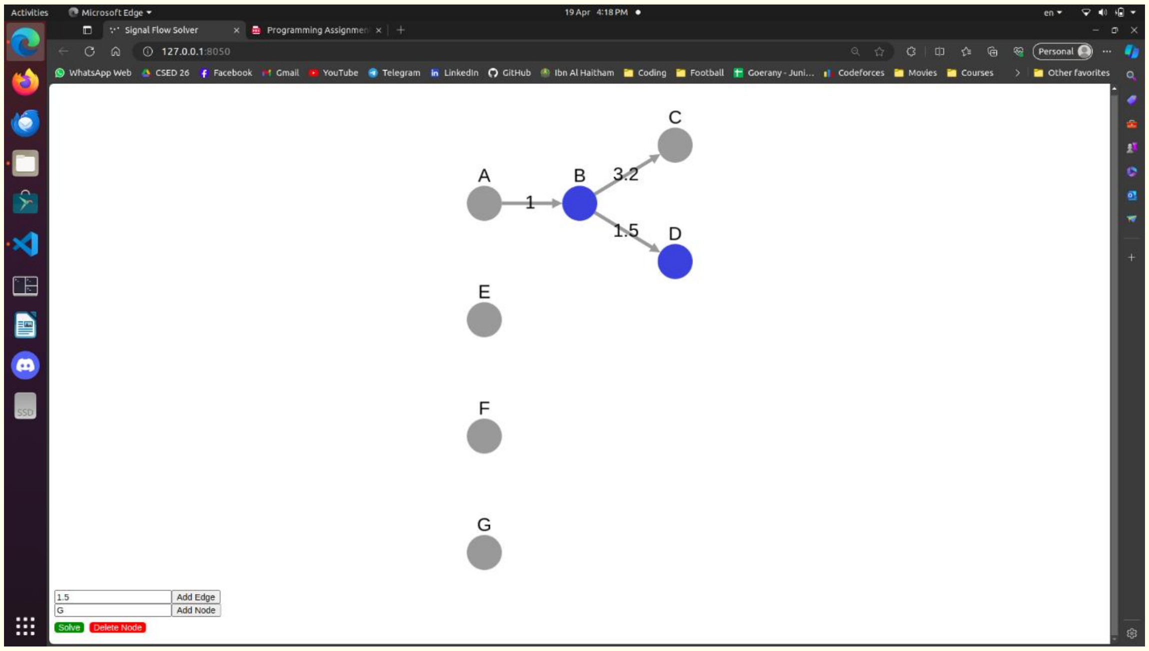
  - Selecting Nodes:

- Hold CTRL key while clicking on the nodes to select multiple nodes, press anywhere else to unselect all.
- You can also drag and move the nodes and the graph as much as you like.



- Adding Nodes and Edges:

- To add a node, enter the name of the node in the "Enter node name" field and click the "Add Node" button.
- To add an edge between two nodes, Enter the weight of the edge in the "Enter edge weight" field, then select the source node and while holding the CTRL key select the target node, then click the "Add Edge" button.

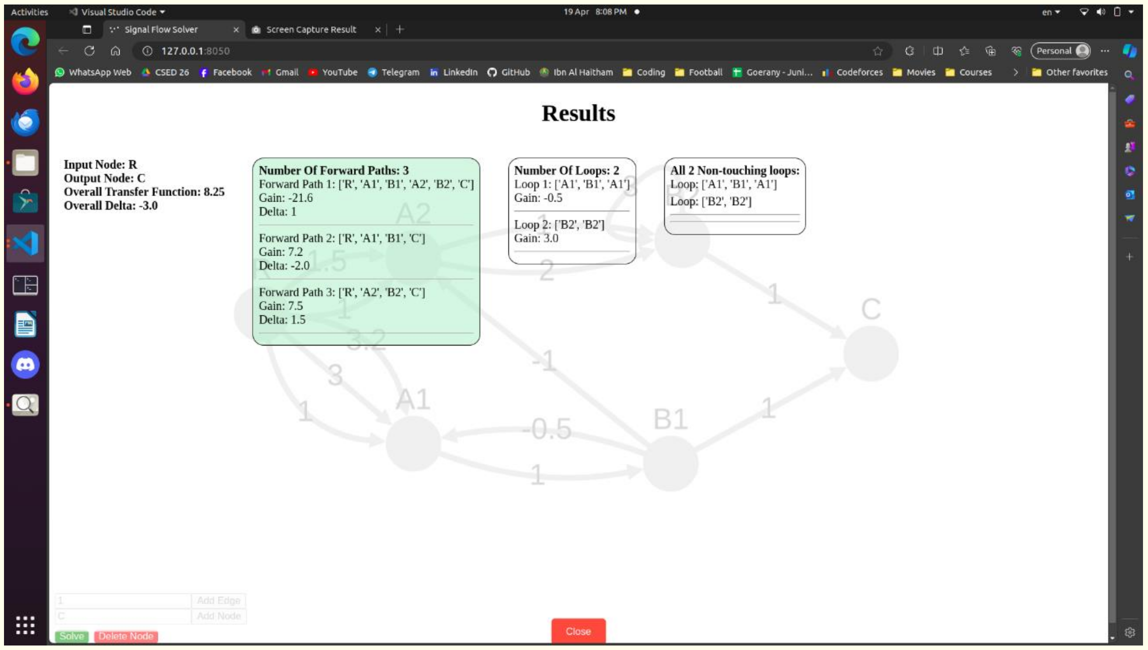


- Deleting Nodes:

- Select the node(s) you want to delete by first selecting them, then click the "Delete Node" button to remove the selected node(s) along with their associated edges.

- Solving the Graph:

- Once you have constructed the signal flow graph, click the "Solve" button to obtain the analysis results.
- The results will be displayed in the overlay container.



Results

Input Node: R
Output Node: C
Overall Transfer Function: 8.25
Overall Delta: -3.0

Number Of Forward Paths: 3
Forward Path 1: ['R', 'A1', 'B1', 'A2', 'B2', 'C']
Gain: -21.6
Delta: 1

Forward Path 2: ['R', 'A1', 'B1', 'C']
Gain: 7.2
Delta: -2.0

Forward Path 3: ['R', 'A2', 'B2', 'C']
Gain: 7.5
Delta: 1.5

Number Of Loops: 2
Loop 1: ['A1', 'B1', 'A1']
Gain: -0.5

Loop 2: ['B2', 'B2']
Gain: 3.0

All 2 Non-touching loops:
Loop: ['A1', 'B1', 'A1']
Loop: ['B2', 'B2']

Routh-Herwitz Criterion

Problem Statement

Given

Characteristic equation of the system. Assuming that all the coefficients of s^0 to s^n

are given, however if s^0 does not exist it is still tested for stability.

Input format s^5 + s^4 + 10*s^3 + 72*s^2 + 152*s + 240

Assumptions

- Input should follow the previous format, however the power can be represented either as ** or ^.
- If the coefficients alter their signs, the system will be unstable and will be checked for the number of roots in the positive side.
- If some coefficients do not exist, the system will not be checked and no further routh array will be constructed.

Required

1- Using Routh criterion, state if the system is stable or not.
2- If the system is not stable, list the number and values of poles in the RHS of the s-plane.

Main Features

The Routh_Criterion is designed to check for the stability of a system given its characteristic equation, based upon the Routh-Herwitz matrix and criterion for a stable system – that none of the roots of the characteristic equation should have a positive real part.

Given an equation of any order of powers the program checks for its stability and output the results whether the system would be stable, marginally stable, or unstable with several positive real roots.

Data Structures & Algorithms

Part 2 is implemented using Python3.

Data Structures

- **Sympy Module** Used to parse equations from user input, extract the coefficients and solve for values of coefficients.
- **2D List** Used to represent the Routh Array.

Algorithms

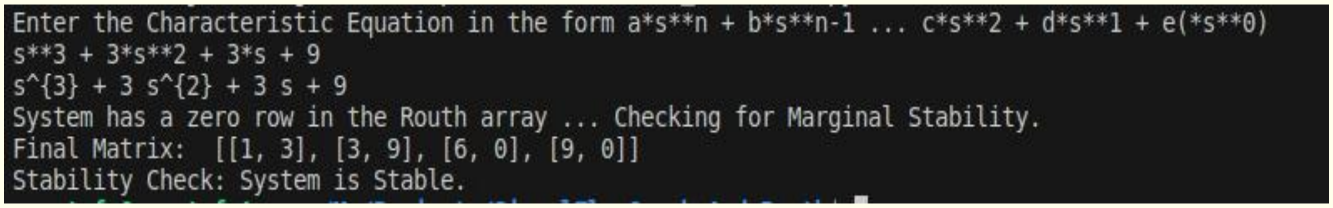
- **Sympy Solve Methods** Used to calculate the values of the positive roots, that cause the system to be unstable.
- **Routh-Herwitz Algorithm** Used to calculate the values of the Routh Array.

Main Modules

- **Routh_Criterion Module**
  This Module contains the implementation of the routh-herwitz stability criterion functions, including functions to parse the user input into a **sympy** equation, extracting coefficients, calculating Routh matrix, and handling special cases, whether a 0 element in the 1st column or a whole **Odd-Ordered** 0_s row.

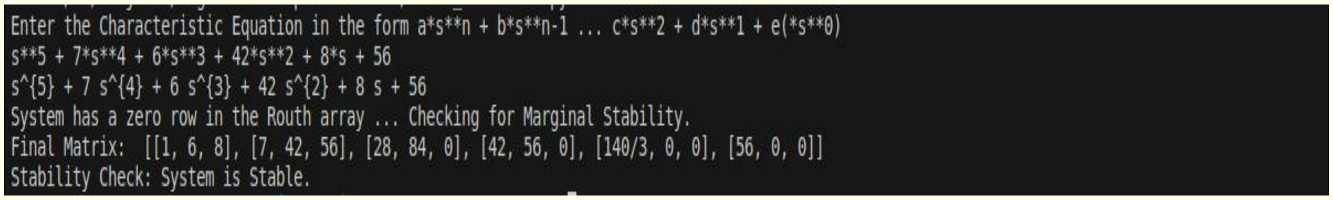
Sample Runs

- Test Case #1
  Stable System with a zero row, i.e **Two identical roots around imaginary plane.**



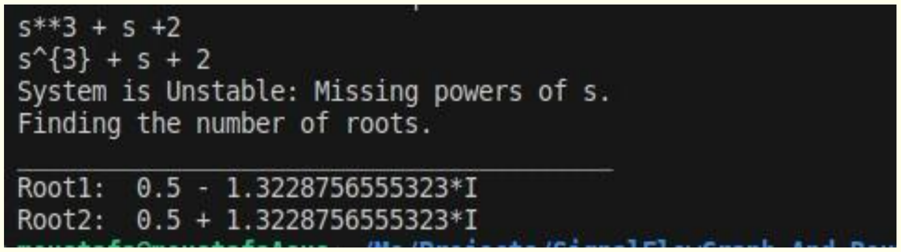
```
Enter the Characteristic Equation in the form a*s**n + b*s**n-1 ... c*s**2 + d*s**1 + e(*s**0)
s**3 + 3*s**2 + 3*s + 9
s^{3} + 3 s^{2} + 3 s + 9
System has a zero row in the Routh array ... Checking for Marginal Stability.
Final Matrix:  [[1, 3], [3, 9], [6, 0], [9, 0]]
Stability Check: System is Stable.
```

- Test Case #2
  Stable System with a zero row, i.e **Two identical roots around imaginary plane.**



```
Enter the Characteristic Equation in the form a*s**n + b*s**n-1 ... c*s**2 + d*s**1 + e(*s**0)
s**5 + 7*s**4 + 6*s**3 + 42*s**2 + 8*s + 56
s^{5} + 7 s^{4} + 6 s^{3} + 42 s^{2} + 8 s + 56
System has a zero row in the Routh array ... Checking for Marginal Stability.
Final Matrix:  [[1, 6, 8], [7, 42, 56], [28, 84, 0], [42, 56, 0], [140/3, 0, 0], [56, 0, 0]]
Stability Check: System is Stable.
```

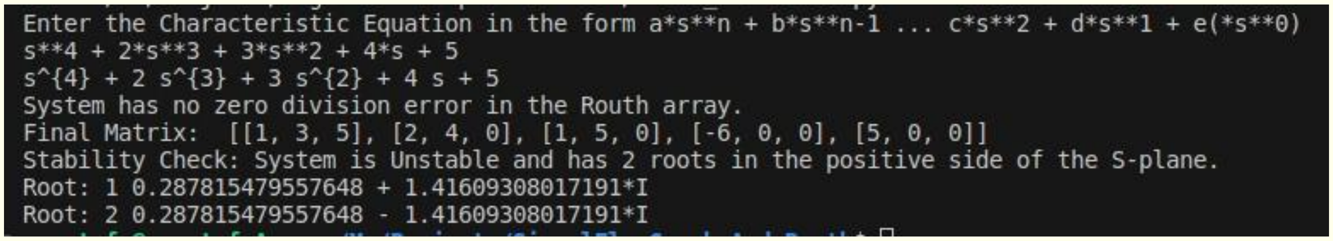
- Test Case #3
  Unstable System due to missing powers of S



```
s**3 + s +2
s^{3} + s + 2
System is Unstable: Missing powers of s.
Finding the number of roots.
________________________________________
Root1:  0.5 - 1.3228756555323*I
Root2:  0.5 + 1.3228756555323*I
```

- Test Case #4

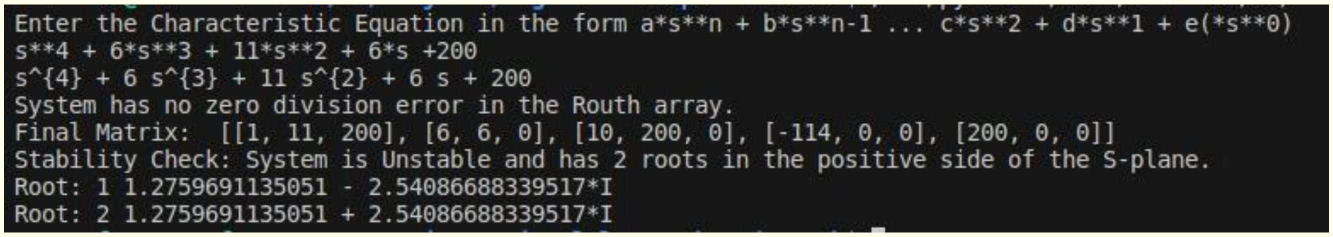
  Unstable System with 2 imaginary roots having positive real parts.



```
Enter the Characteristic Equation in the form a*s**n + b*s**n-1 ... c*s**2 + d*s**1 + e(*s**0)
s**4 + 2*s**3 + 3*s**2 + 4*s + 5
s^{4} + 2 s^{3} + 3 s^{2} + 4 s + 5
System has no zero division error in the Routh array.
Final Matrix:  [[1, 3, 5], [2, 4, 0], [1, 5, 0], [-6, 0, 0], [5, 0, 0]]
Stability Check: System is Unstable and has 2 roots in the positive side of the S-plane.
Root: 1 0.287815479557648 + 1.41609308017191*I
Root: 2 0.287815479557648 - 1.41609308017191*I
```

- Test Case #5

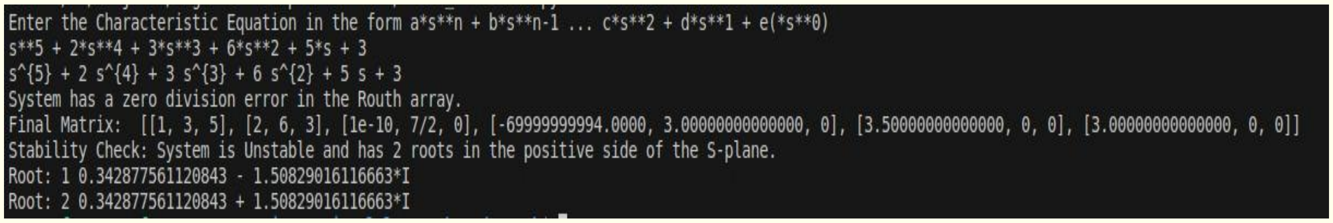
  Unstable System with 2 imaginary roots having positive real parts.



```
Enter the Characteristic Equation in the form a*s**n + b*s**n-1 ... c*s**2 + d*s**1 + e(*s**0)
s**4 + 6*s**3 + 11*s**2 + 6*s +200
s^{4} + 6 s^{3} + 11 s^{2} + 6 s + 200
System has no zero division error in the Routh array.
Final Matrix:  [[1, 11, 200], [6, 6, 0], [10, 200, 0], [-114, 0, 0], [200, 0, 0]]
Stability Check: System is Unstable and has 2 roots in the positive side of the S-plane.
Root: 1 1.2759691135051 - 2.54086688339517*I
Root: 2 1.2759691135051 + 2.54086688339517*I
```

- Test Case #6

  Unstable System with 2 imaginary roots having positive real parts.



```
Enter the Characteristic Equation in the form a*s**n + b*s**n-1 ... c*s**2 + d*s**1 + e(*s**0)
s**5 + 2*s**4 + 3*s**3 + 6*s**2 + 5*s + 3
s^{5} + 2 s^{4} + 3 s^{3} + 6 s^{2} + 5 s + 3
System has a zero division error in the Routh array.
Final Matrix:  [[1, 3, 5], [2, 6, 3], [1e-10, 7/2, 0], [-69999999994.0000, 3.00000000000000, 0], [3.50000000000000, 0, 0], [3.00000000000000, 0, 0]]
Stability Check: System is Unstable and has 2 roots in the positive side of the S-plane.
Root: 1 0.342877561120843 - 1.50829016116663*I
Root: 2 0.342877561120843 + 1.50829016116663*I
```

Simple User Guide

- Run the Python File Routh_Criterion.py – **After Installation in part 1**.
- **Input format** s^5 + s^4 + 10*s^3 + 72*s^2 + 152*s + 240
- Input should follow the previous format, however the power can be represented either as ** or ^.