Programming Assignment Report:

Signal Flow Graphs & Routh Stability Criterion

Contributors:

Name\ عبد الرحمن محمد أحمد أحمد نصر
ID\22010887

Name\ عبدالرحمن اسماعيل محمد حسن
ID\ 22010866

Name\ لجين سامح عبد المنعم عبد الجواد
ID\ 22011054

Name\محمد مصطفى سيد محمد محمد علي
ID\22011170

Name\ بدر السيد جلال

ID\ 22010664

Name\ نور خالد محمد
ID\ 22011319

1.Problem Statement:

- **Part 1**: Analyze a signal flow graph (SFG) to compute:
  - Forward paths, loops, and non-touching loops.
  - Determinants and the overall transfer function using Mason's Gain Formula.
- **Part 2**: Determine system stability using the Routh-Hurwitz criterion for a given characteristic equation.

2.Main Features:

- **Signal Flow Graph Solver:**
  - Parses SFG input (nodes, edges, gains) using **SignalFlowGraphSolverSchema.**
  - Computes paths/loops using graph traversal (implied by **networkx**).
  - Calculates determinants and transfer function symbolically (**sympy**).
- **Routh Stability Analyzer:**
  - Parses polynomial input (s^5 + s^4 + 10s^3).
  - Constructs Routh array and checks for sign changes.
  - Handles edge cases.
- **Additional Features:**
  - Input validation by **marshmallow**.
  - Symbolic math for precise calculations (**sympy**).

3.Data Structures:

- **Signal Flow Graph:**
  - Represented as a directed multigraph (**networkx.MultiDiGraph**).
  - Nodes: Strings ("A", "B").
  - Edges: Tuples (source, target, gain) with **sympy**-parsed gains.
- **Paths/Loops:**
  - **Path** class stores nodes/edges and computes gains.
  - **ForwardPath** extends **Path** with determinant tracking.
- **Routh Table:**
  - 2D NumPy array for coefficients.
  - Auxiliary polynomial handling for zero rows.

4.Main Modules:

1. Signal Flow Graph (SFG) Solver Module

- **Files**: sfg.py, marshaller.py, path.py, classes.py (implied).
- **Responsibilities:**
  - **Input Validation**: Uses **marshmallow** (**SignalFlowGraphSolverSchema**) to validate JSON input for nodes, edges, and gains.
  - **Graph Representation**: Constructs a directed multigraph (**networkx.MultiDiGraph**) from validated input.
  - **Path/Loop Analysis**:
    - **Path** and **ForwardPath** classes store nodes/edges and compute gains.
    - Algorithms (DFS/BFS) traverse the graph to find forward paths and loops.
  - **Mason's Gain Formula**: Computes determinants (Δ, Δ*k*) and transfer function symbolically (**sympy**).

2. Routh Stability Analyzer Module

- **Files**: solver.py.
- **Responsibilities**:
  - **Polynomial Parsing**: Converts input strings to sympy expressions.
  - **Routh Table Construction**:
    - Builds the Routh array from polynomial coefficients.
    - Handles edge cases.
  - **Stability Check**: Counts sign changes in the first column of the Routh table to determine stability.

3. Data Marshalling Module

- **Files**: marshaller.py.
- **Responsibilities**:

- **Input/Output Standardization**:
  - **marshall_input:** Validates and converts raw input (JSON) into a graph object.
  - **marshall_output:** Formats results (paths, determinants, etc.) for display/storage.

4. Core Utilities

- **Dependencies**:
  - **sympy**: Symbolic math for determinants and transfer functions.(v1.13.1)
  - **networkx**: Graph traversal and cycle detection.
  - **numpy**: Numerical operations for Routh array.

5.Algorithms:

Part 1: Signal Flow Graph

1. **Graph Traversal**:
   - **Forward Paths**: DFS/BFS from input to output node.
   - **Loops**: Cycle detection in **networkx**.
2. **Mason's Gain Formula**:
   - $\Delta = 1 - \sum L_i + \sum L_i L_j - \sum L_i L_j L_k + \ldots$
   - Δk: Determinant for the k-th forward path.
3. **Non-Touching Loops:**
   - Combinations of loops with no common nodes (combinatorial search).

Part 2: Routh-Hurwitz Criterion

1. **Polynomial Parsing:**
   - Convert input string to **sympy** expression.
2. **Routh Array Construction:**

- Fill rows recursively:

$$r_{i,j} = (r_{i-1,0} * r_{i-2,j+1} - r_{i-2,0} * r_{i-1,j+1}) / (r_{i-1,0})$$

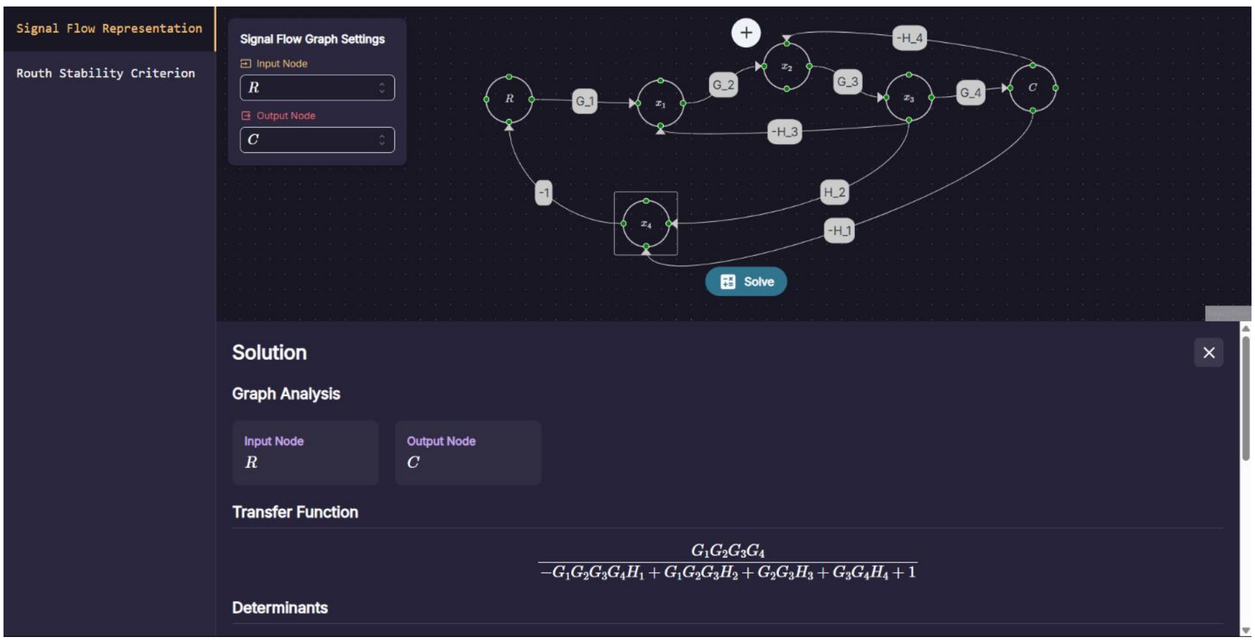
- Handle zero rows with auxiliary polynomials.

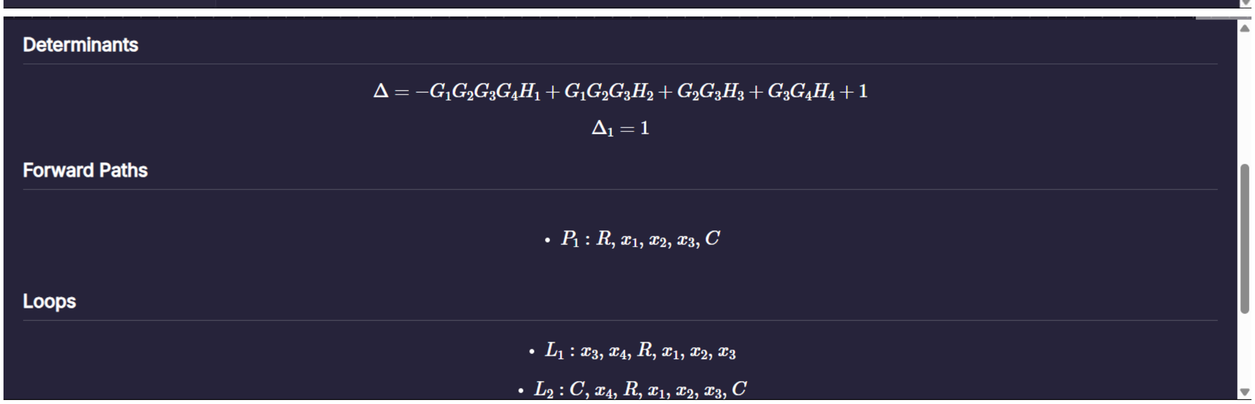
3. **Stability Check:**

   - Count sign changes in the first column (unstable if > 0).

6.Sample Runs:

Signal Flow Representation:





Loops

- $L_1 : x_3, x_4, R, x_1, x_2, x_3$
- $L_2 : C, x_4, R, x_1, x_2, x_3, C$
- $L_3 : C, x_2, x_3, C$
- $L_4 : x_3, x_1, x_2, x_3$

Non Touching Loops

Signal Flow Representation

Routh Stability Criterion

Signal Flow Graph Settings

Input Node

R

Output Node

C

Solve

Solution

Graph Analysis

Input Node
R

Output Node
C

Transfer Function

$$\frac{G_1G_2G_3G_4}{G_1G_2G_3G_4H_6 + G_1G_2G_3H_1H_5 + G_1G_3H_1H_3 + G_1G_4H_1H_4 + G_1H_1 + G_2G_3H_5 + G_2G_4H_2H_4 + G_2H_2 + G_3H_3 + G_4H_4 + 1}$$

Determinants

Determinants

$$\Delta = G_1G_2G_3G_4H_6 + G_1G_2G_3H_1H_5 + G_1G_3H_1H_3 + G_1G_4H_1H_4 + G_1H_1 + G_2G_3H_5 + G_2G_4H_2H_4 + G_2H_2 + G_3H_3 + G_4H_4 + 1$$

$$\Delta_1 = 1$$

Forward Paths

- $P_1 : R, x_1, x_2, x_3, x_4, x_5, C$

Loops

- $L_1 : C, x_5, C$
- $L_2 : x_3, x_4, x_2, x_3$

Loops

- $L_1 : C, x_5, C$
- $L_2 : x_3, x_4, x_2, x_3$
- $L_3 : x_2, x_1, x_2$
- $L_4 : C, R, x_1, x_2, x_3, x_4, x_5, C$
- $L_5 : x_3, x_4, x_5, x_3$
- $L_6 : x_4, x_5, x_4$

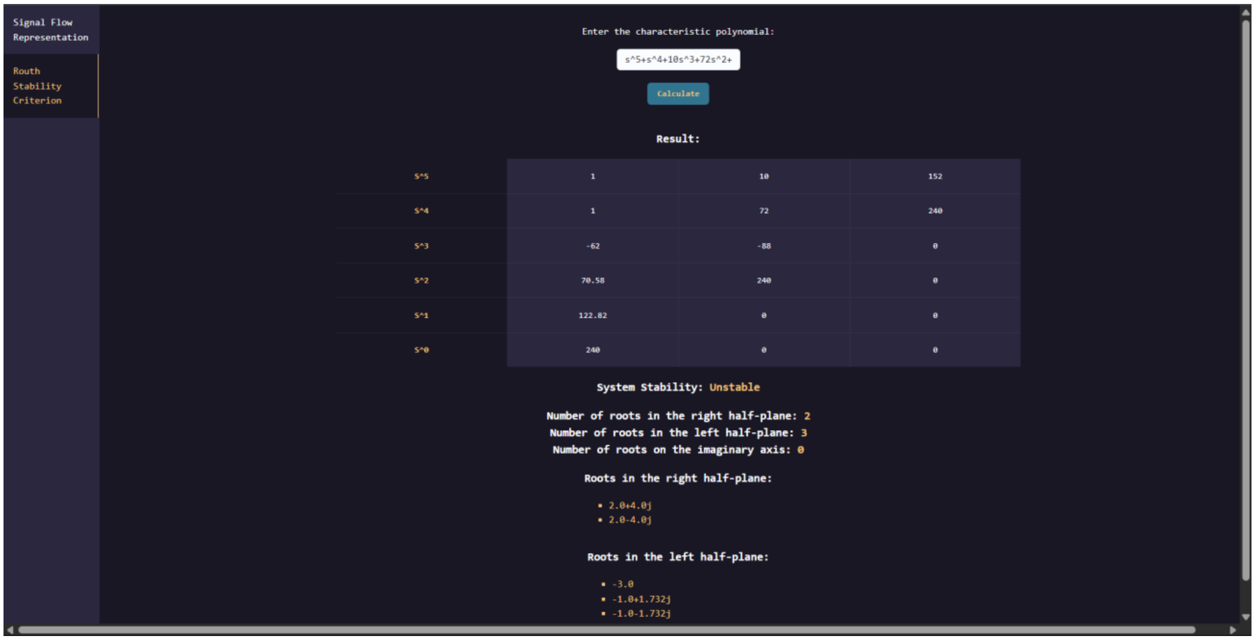
Non Touching Loops

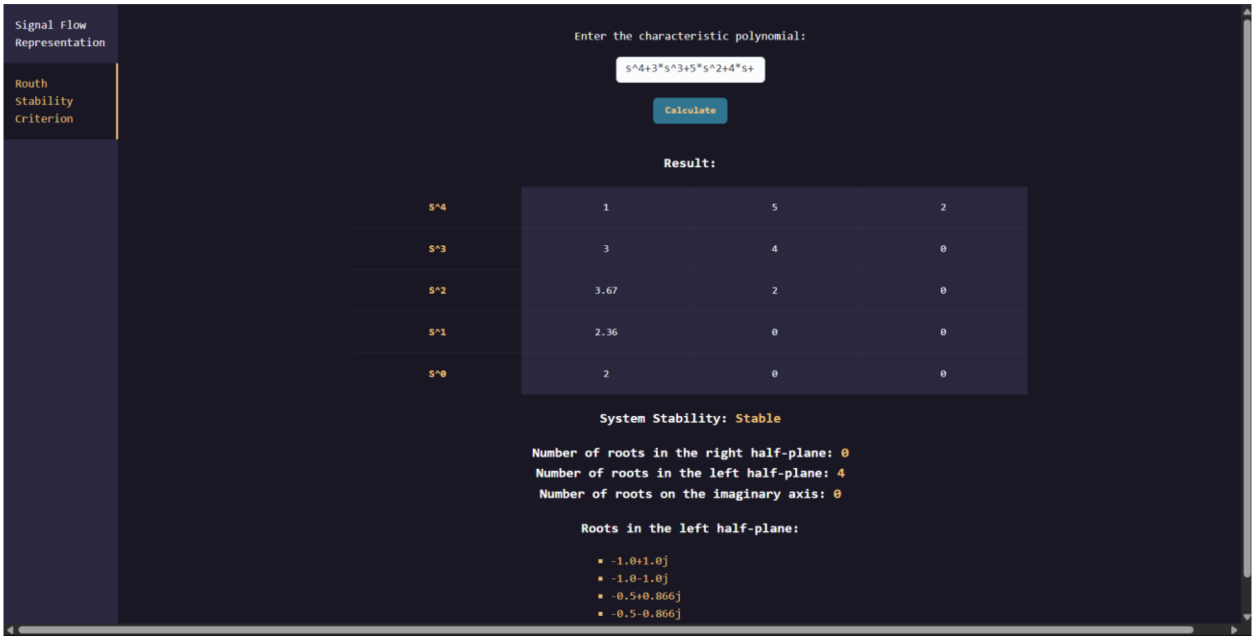
- Combinations of 2 non-touching loops:

Non Touching Loops

- Combinations of 2 non-touching loops:
  - $(C, x_5, C), (x_3, x_4, x_2, x_3)$
  - $(C, x_5, C), (x_2, x_1, x_2)$
  - $(x_2, x_1, x_2), (x_3, x_4, x_5, x_3)$
  - $(x_2, x_1, x_2), (x_4, x_5, x_4)$

Routh Stability Criterion:





7.User Guide:

1. **Installation**:
   - Run “pip install flask flask_cors networkx sympy numpy marshmallow”.
   - Note: make sure the version of **sympy** is **1.13.1**.

2. **Running the Program:**
   - **Run frontend:**

     While in the “react-frontend” folder:

     1. Run “npm install” in the terminal.
     2. Run “npm run dev” to run the front end.

   - **Run backend:**

     While in the “flask-backend” folder:

     1. Run “python app.py” in the terminal to run the back end.

        Note: make sure port 5000 is free on the machine.

3. **Input Format:**
   - **SFG:** the input is in the form of graph, so the user draws the signal flow graph using the (+) sign on top and connects using the small colored dots.
   - **Polynomial:** String with **^** or ****** for exponential relations (example: s^5 + 10s^3).

Repo: GitHub - Abdelrahman-Nasr6161/SignalFlowChartSolver