Control Systems

Programming Assignment

https://github.com/MostafaElKaranshawy/Signal-Flow-Graph

Karene Antoine Nassif	21010969
Rowan Gamal Ahmed Elkhouly	21010539
Yomna Yasser sobhy Zaki	21011566
Mustafa Mohamed Elkaranshawy	21011375
Youssef Mahmoud Mohamed	21011630
Amir Ragaie Soliman	21010300

Part One : Signal Flow Graph

1) Problem Statement :

Creating a program for Signal flow graph representation of a given system given the total number of nodes and the numeric values of the branches gains.
A web based program using VueJS for frontend and Java springboot for backend.

2) Main Features Of The Program.

- Simple and user friendly graphical interface.
- Visualization of the signal flow graph using nodes and edges.
- Calculating the overall transfer function of the given system.
- Listing all the forward paths along with their gains.
- Listing all loops in the system along with their gains.
- Listing all non-touching loops in the system along with their gains.
- Calculating the values of Δ , $\Delta_1, \Delta_2 \ldots \Delta_n$ in the system.

3) Data Structures Used

Implementing classes that represent

I. Paths
- List of Integers that represent gain of each path of the forward paths.
- List of Lists such that each list contains a forward path(nodes) from the start node to the end node.

II. Single Loop
- A class to represent each loop, contains the loop id , the loop gain and the loop path.

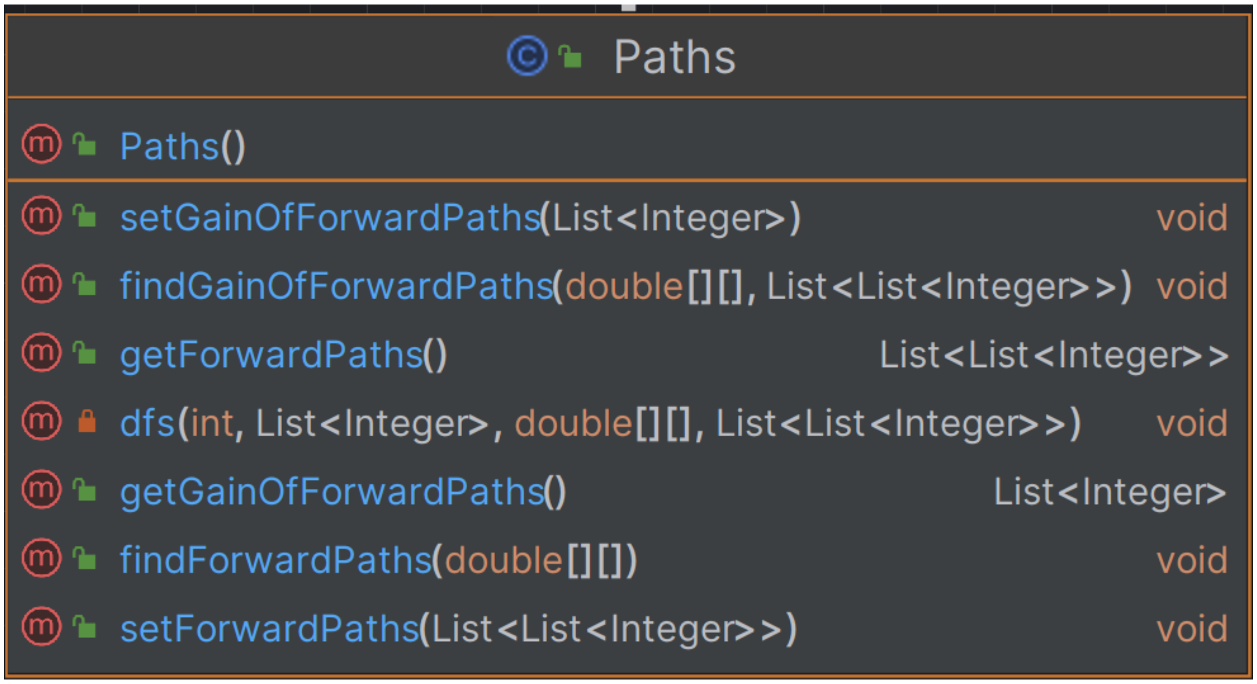
III. Loops
- A 2-d array of doubles (adjacency matrix) contains the graph representation of the system
- List of Single loops that contains all loops in the system.
- List of Sets such that each set represents a pair of 2 non-touching loops.
- HashSet used to set the path of each loop to prevent having the same loop's path but starting from another node.

IV. Deltas
- Arraylist of doubles to calculate $\Delta_1, \Delta_2 \ldots \Delta_n$.

V. Service (Wrapper Class)
- Includes ArrayList of forwardPaths
- ArrayList of gainofForwardPaths
- ArrayList of Loops
- ArrayList of Non-touching Loops
- ArrayList of Deltas
- Main Delta
- Transfer Function

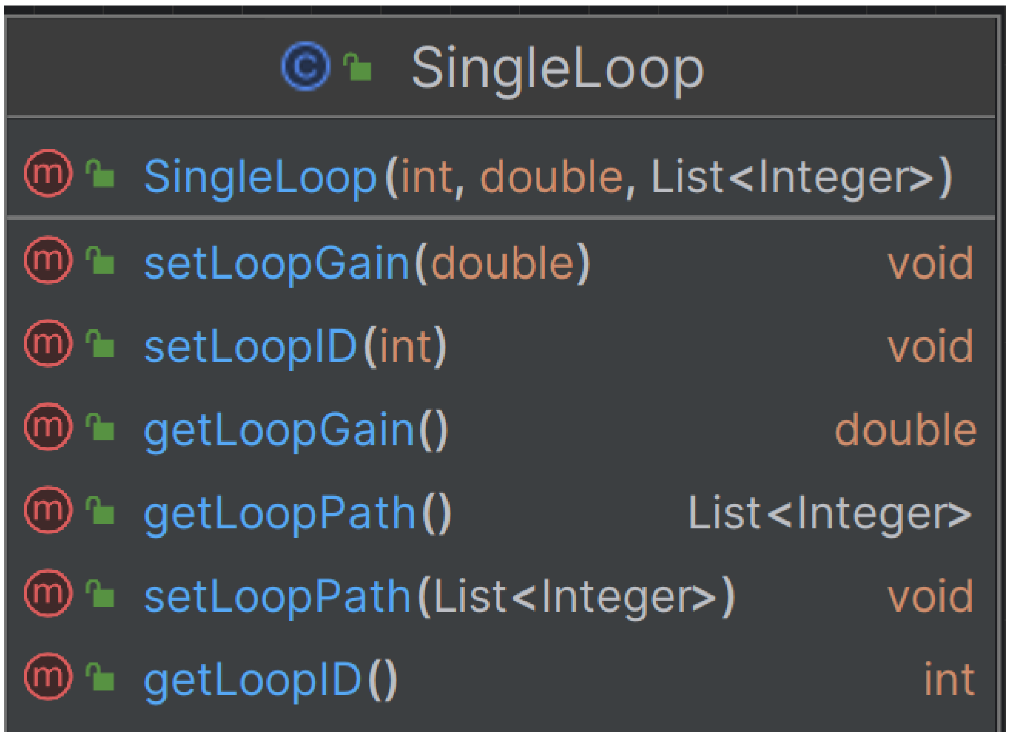
4) Main Modules

I. Paths



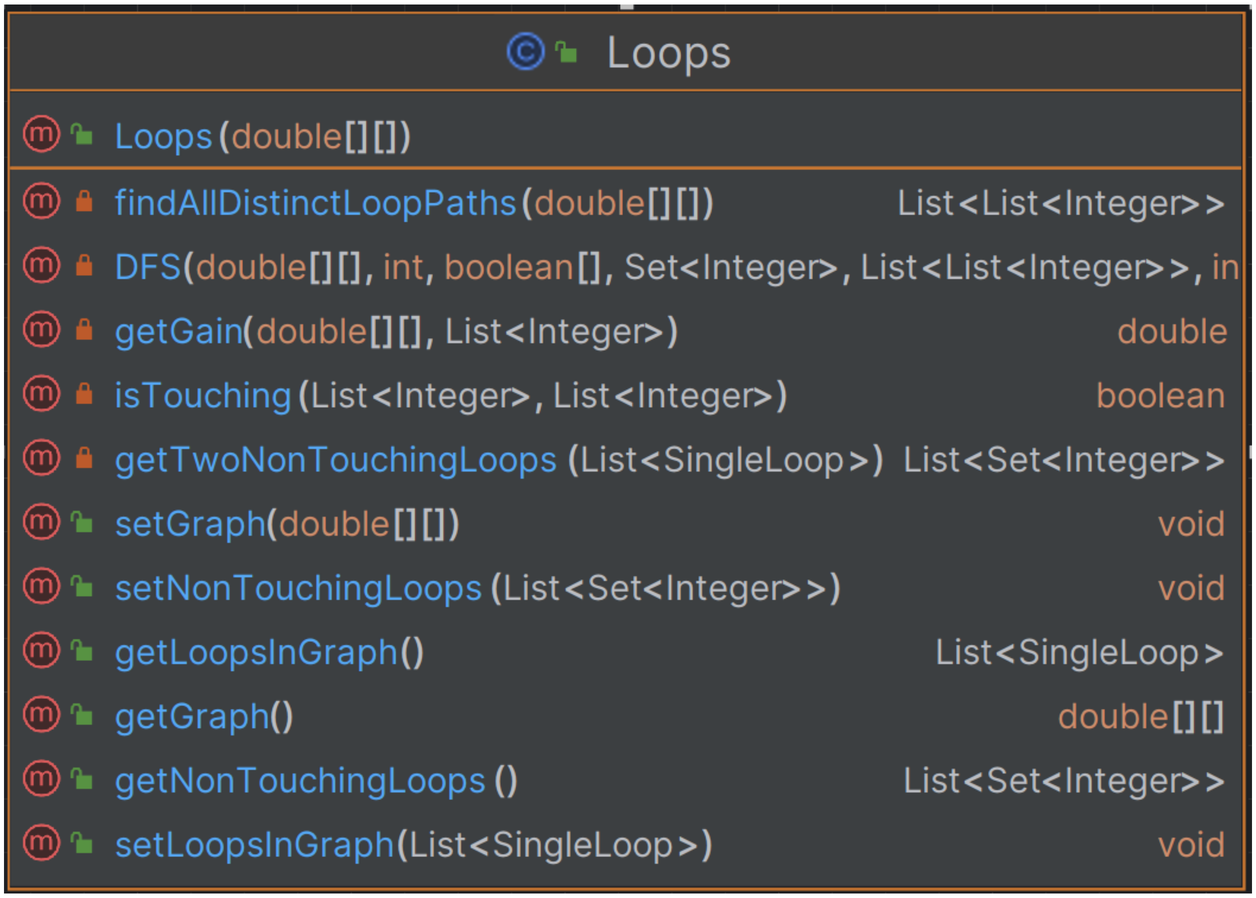
```
Paths

Paths()

setGainOfForwardPaths(List<Integer>)                      void
findGainOfForwardPaths(double[][], List<List<Integer>>)   void
getForwardPaths()                            List<List<Integer>>
dfs(int, List<Integer>, double[][], List<List<Integer>>)  void
getGainOfForwardPaths()                          List<Integer>
findForwardPaths(double[][])                              void
setForwardPaths(List<List<Integer>>)                      void
```

II. Single Loop



```
SingleLoop

SingleLoop(int, double, List<Integer>)

setLoopGain(double)                  void
setLoopID(int)                       void
getLoopGain()                      double
getLoopPath()               List<Integer>
setLoopPath(List<Integer>)           void
getLoopID()                           int
```

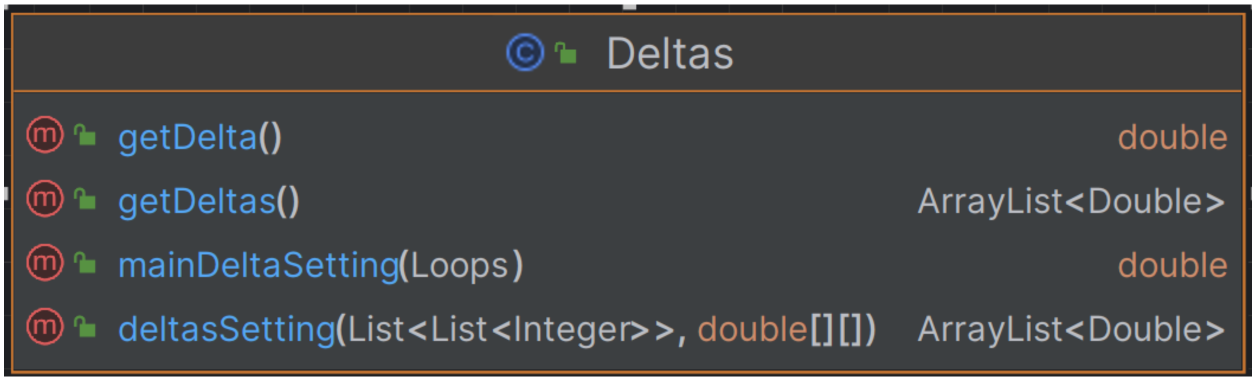
III. Loops



Loops

Loops(double[][])		
findAllDistinctLoopPaths(double[][])	List<List<Integer>>
DFS(double[][], int, boolean[], Set<Integer>, List<List<Integer>>, in	
getGain(double[][], List<Integer>)	double
isTouching(List<Integer>, List<Integer>)	boolean
getTwoNonTouchingLoops(List<SingleLoop>)	List<Set<Integer>>
setGraph(double[][])	void
setNonTouchingLoops(List<Set<Integer>>)	void
getLoopsInGraph()	List<SingleLoop>
getGraph()	double[][]
getNonTouchingLoops()	List<Set<Integer>>
setLoopsInGraph(List<SingleLoop>)	void

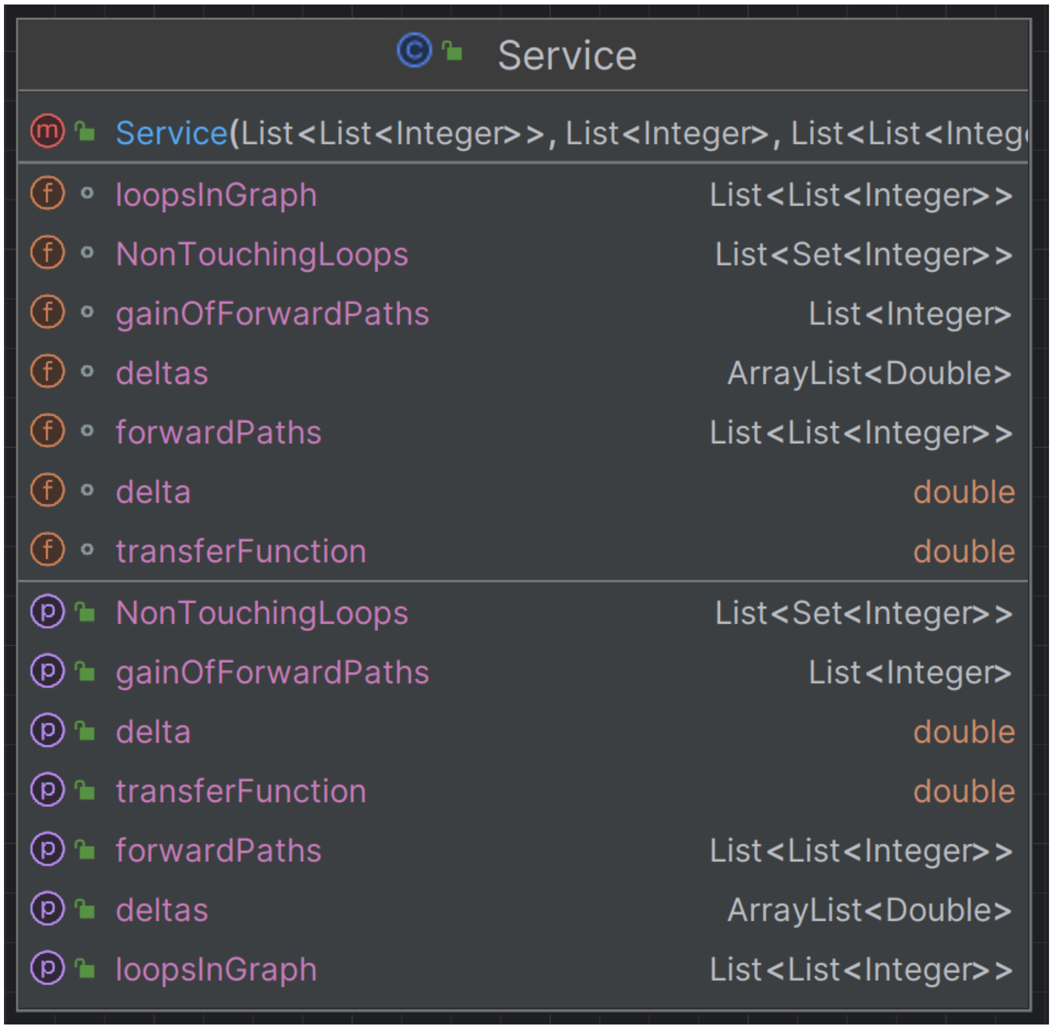
IV. Deltas



Deltas

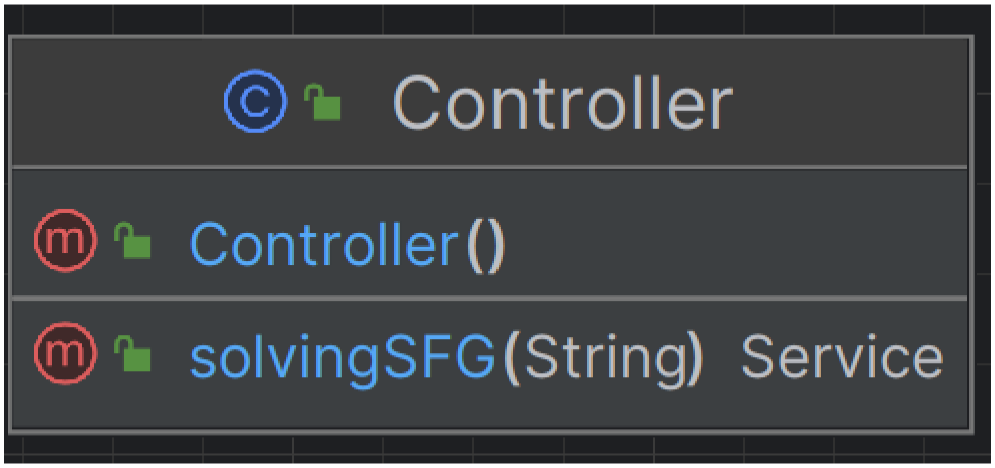
getDelta()	double
getDeltas()	ArrayList<Double>
mainDeltaSetting(Loops)	double
deltasSetting(List<List<Integer>>, double[][])	ArrayList<Double>

V. Service Wrapper Class



```
Service
Service(List<List<Integer>>, List<Integer>, List<List<Integ
loopsInGraph                List<List<Integer>>
NonTouchingLoops            List<Set<Integer>>
gainOfForwardPaths          List<Integer>
deltas                      ArrayList<Double>
forwardPaths                List<List<Integer>>
delta                       double
transferFunction            double
NonTouchingLoops            List<Set<Integer>>
gainOfForwardPaths          List<Integer>
delta                       double
transferFunction            double
forwardPaths                List<List<Integer>>
deltas                      ArrayList<Double>
loopsInGraph                List<List<Integer>>
```

VI. Controller



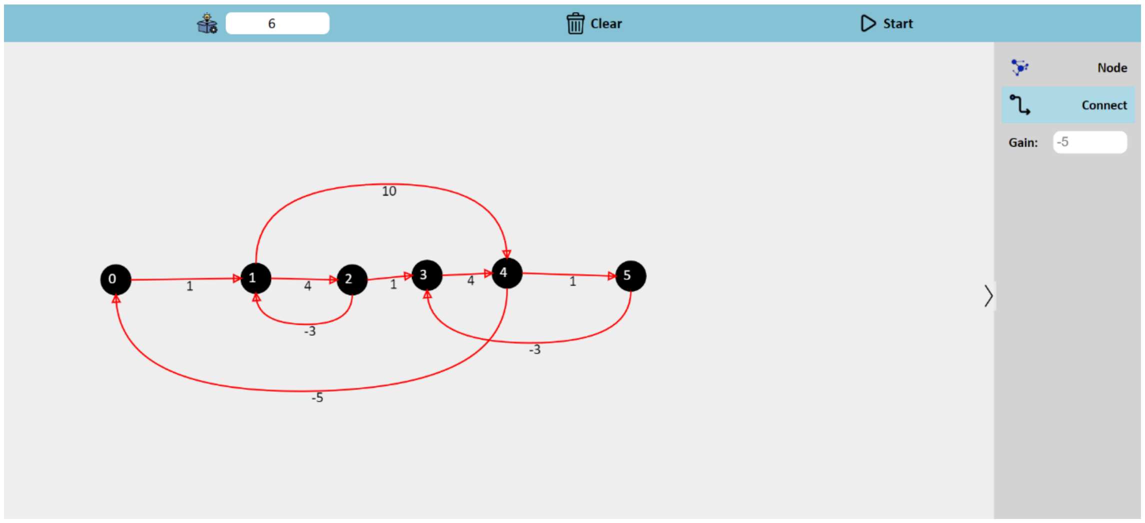
```
Controller
Controller()
solvingSFG(String)  Service
```

5) Algorithms Used

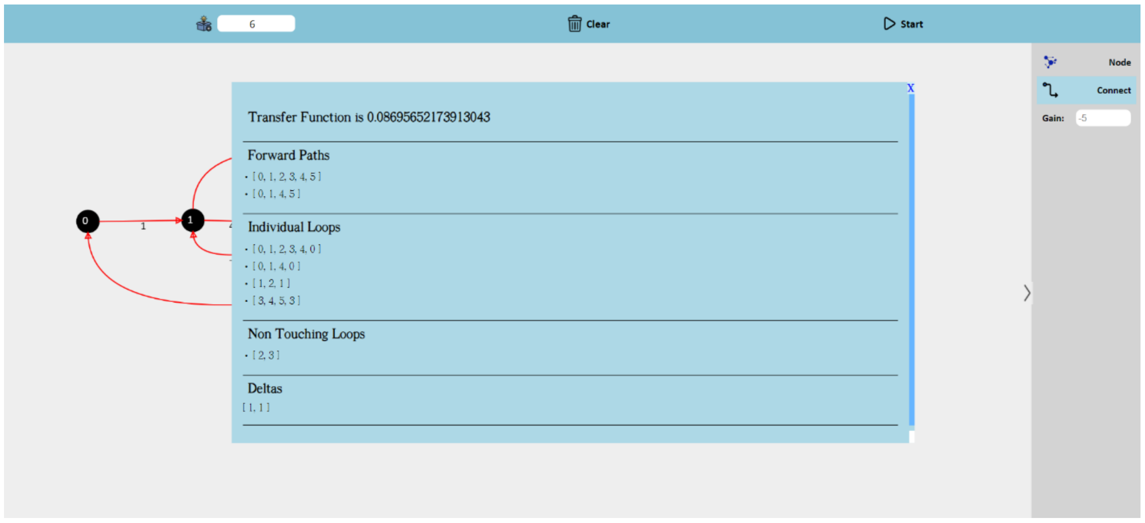
- Modified DFS Algorithm to calculate the forward paths and the loops' paths.
- Mason's Rule :)

6) Sample Runs

- **Sample run 1:**



- Sample run1 output:



Transfer Function is 0.08695652173913043

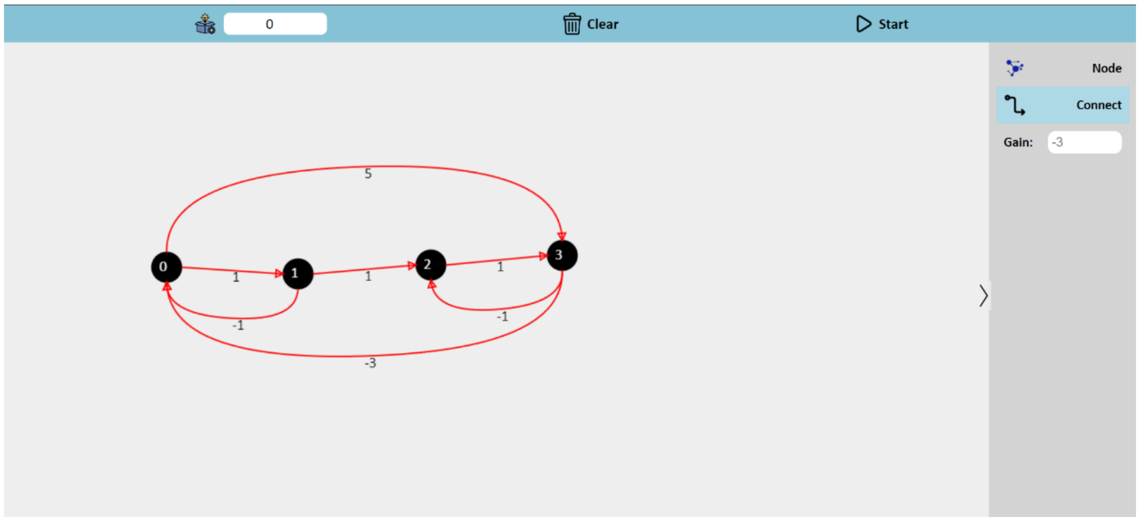
Forward Paths
- [0, 1, 2, 3, 4, 5]
- [0, 1, 4, 5]

Individual Loops
- [0, 1, 2, 3, 4, 0]
- [0, 1, 4, 0]
- [1, 2, 1]
- [3, 4, 5, 3]

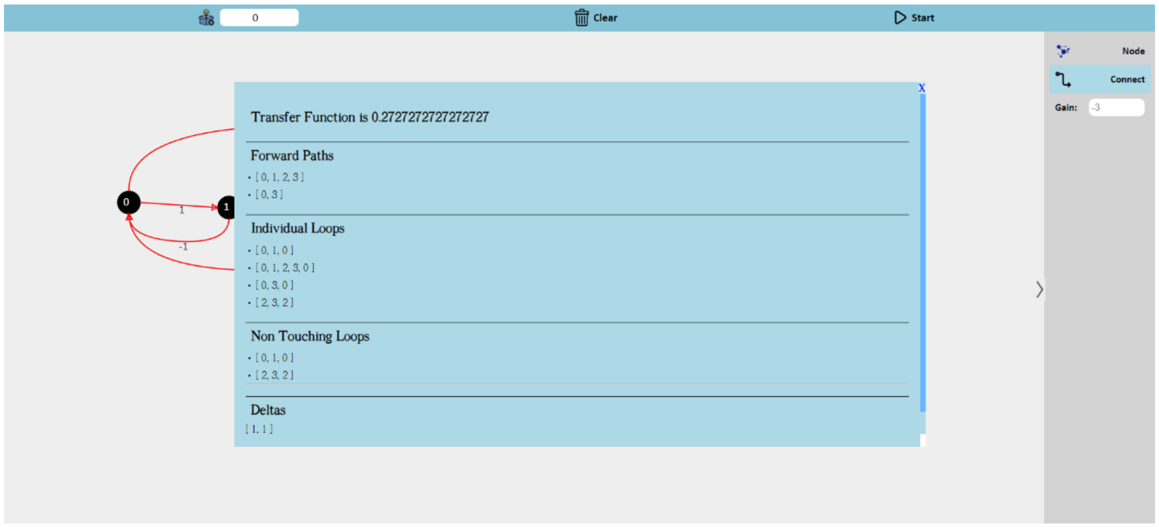
Non Touching Loops
- [2, 3]

Deltas
[1, 1]

- **Sample run 2:**



- Sample run 2 outputs:



Transfer Function is 0.2727272727272727

Forward Paths

- [0, 1, 2, 3]
- [0, 3]

Individual Loops

- [0, 1, 0]
- [0, 1, 2, 3, 0]
- [0, 3, 0]
- [2, 3, 2]

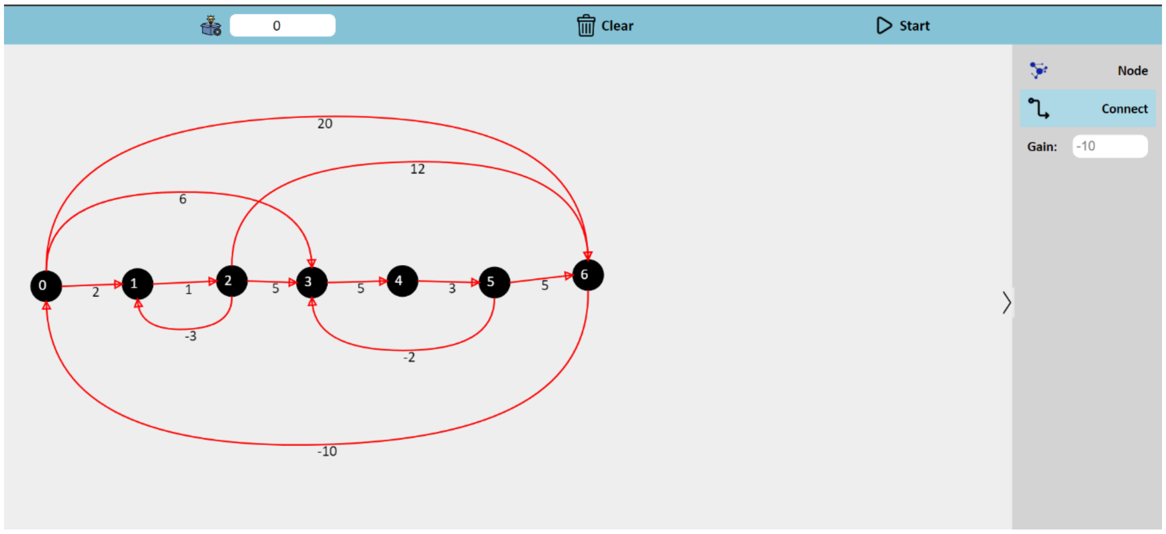
Non Touching Loops

- [0, 1, 0]
- [2, 3, 2]

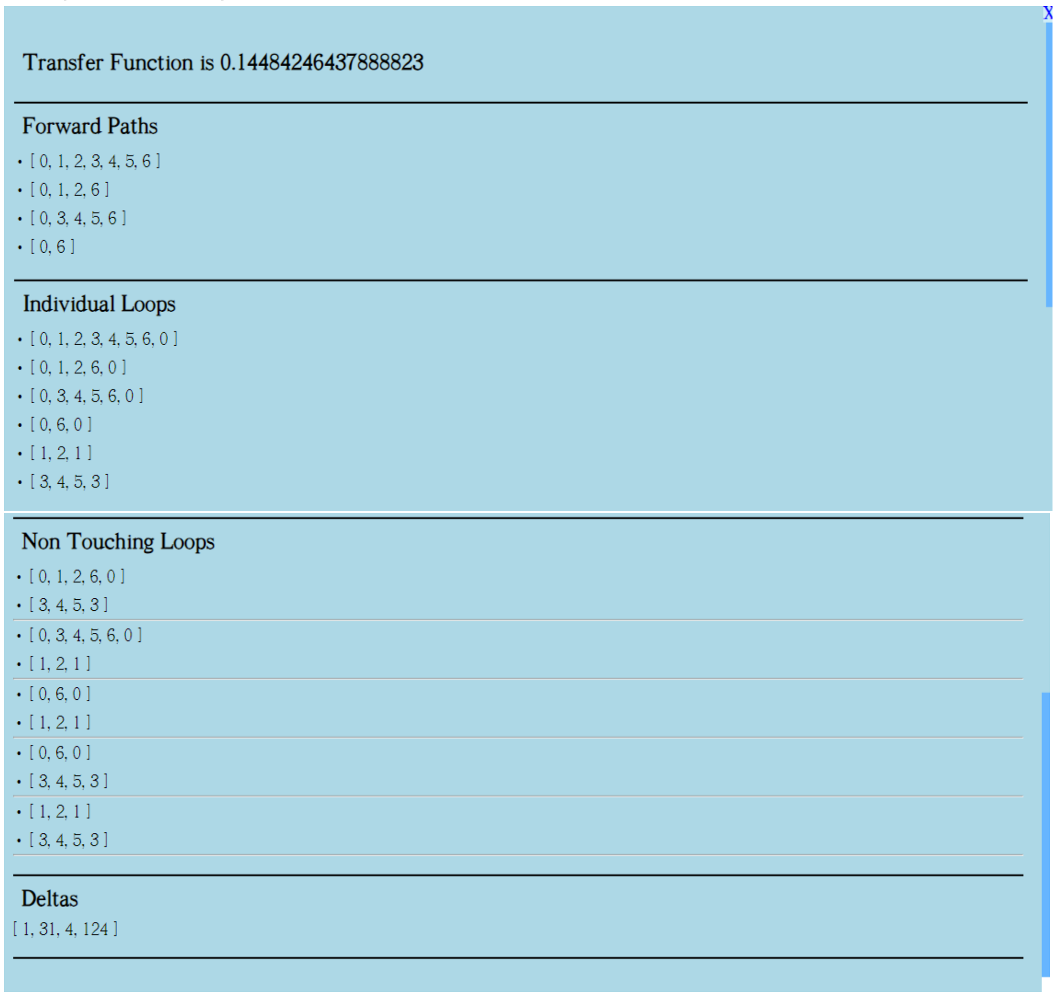
Deltas

[1, 1]

- **Sample run 3:**



- Sample run 3 outputs:



Transfer Function is 0.14484246437888823

Forward Paths

- [0, 1, 2, 3, 4, 5, 6]
- [0, 1, 2, 6]
- [0, 3, 4, 5, 6]
- [0, 6]

Individual Loops

- [0, 1, 2, 3, 4, 5, 6, 0]
- [0, 1, 2, 6, 0]
- [0, 3, 4, 5, 6, 0]
- [0, 6, 0]
- [1, 2, 1]
- [3, 4, 5, 3]

Non Touching Loops

- [0, 1, 2, 6, 0]
- [3, 4, 5, 3]

- [0, 3, 4, 5, 6, 0]
- [1, 2, 1]

- [0, 6, 0]
- [1, 2, 1]

- [0, 6, 0]
- [3, 4, 5, 3]

- [1, 2, 1]
- [3, 4, 5, 3]

Deltas

[1, 31, 4, 124]

7) <u>Sample User Guide</u>

- **Code running:**
  - Open code source in the IDE.
  - Run command **npm run serve** in the project terminal.
  - Open the project running local host.
- **Application:**
  - Enter the total number of nodes in the problem at its field in the main bar.
  - Set the nodes position by clicking the nodes button in the sidebar, after enabling it, you can click at any empty place in the board to put the node.
  - After setting all nodes click the connect button to start adding branches between nodes.
  - Enter the gain of each branch in its field in the sidebar then choose any two nodes to add this branch with the entered gain in it.
  - Make sure that feedback branch gain must be negative, and the forward branch gain must be positive to get a right transfer function.
  - To change the value of the branch gain, make another connect between the same two nodes with different value, if you need to remove the branch make the new value equal to 0.
  - After adding all nodes and branches press start to get your problem solution.
  - The solution is displayed as following:
    - Transfer Function Gain.
    - Forward Paths nodes.
    - Individual Loops nodes
    - Non touching loops nodes separated as a group by a line.
    - Deltas values.
  - Now You can close the Output window and clear the graph, so you can add a new problem to solve.

<u>Part 2: Routh Criterion</u>

<u>A-Problem Statement:</u>

For a given characteristic equation, Use Routh criteria to state if the system is stable or not and if the system is not stable, list the number and values of poles in the RHS of the s-plane.

<u>B-Main Features:</u>

-The program takes the degree of the characteristic equation (n) and the coefficient of each variable in the equation from (S^0 to S^n) and then uses the Routh-Hurwitz Stability

Criterion to check the stability of the system by checking the number of sign changes in the first column in routh table.

-prints the routh table.

-tells the user if the system is stable or not.

-if the system is unstable, it provides the place of the roots in the RHS of S-Plane which causes the unstability.

-it handles the special case of having an entry in the first column of the routh table with zero value by replacing the zero with a very small value (EPSILON = 10^{-9}) and continues the calculations.

-it handles the special case of having a row full of zeros in the routh table by replacing that row by the derivative of the auxiliary equation and check if there are duplicate roots on the jw-axis that will lead to the system being unstable.

C-Data Structures:

One-dimensional arrays and two-dimensional arrays are used to represent routhTable, coefficient Array, and other parts of code.

D-Main modules:

The program consists of a single module and multiple methods, each on for a particular purpose (some methods are further illustrated and other assumed to be clear from the names of the methods):

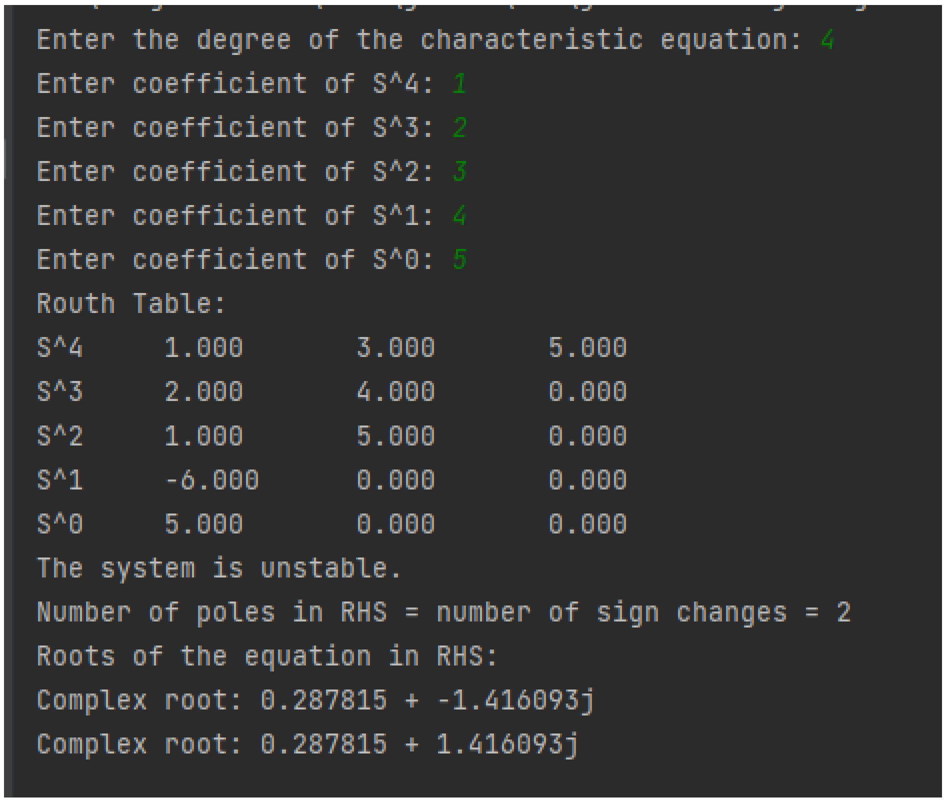
- checkZeroRow
- replaceZeroRow => used to replace the zero row with the derivative of the auxiliary equation
- printRouthTable => used to print at any point of code and stops printing if a zero row is encountered
- checkStability => decides if the system is stable or not based on the number of sign changes in the first column of routh table or if there are duplicate roots in the RHS of S-Plane
- createRouthTable => calculate entries in the routh table
- getAuxiliaryCoefficients => get coefficients of the auxiliary array
- checkDuplicateRoots => check the existence of multiple roots on the jw-axis
- findRootsInRHS
- main => takes input from the user regarding the degree of characteristic equation and its coefficients and calls other methods to perform the required functionality

D-Algorithms used:

Routh-Hurwitz Stability Criterion.

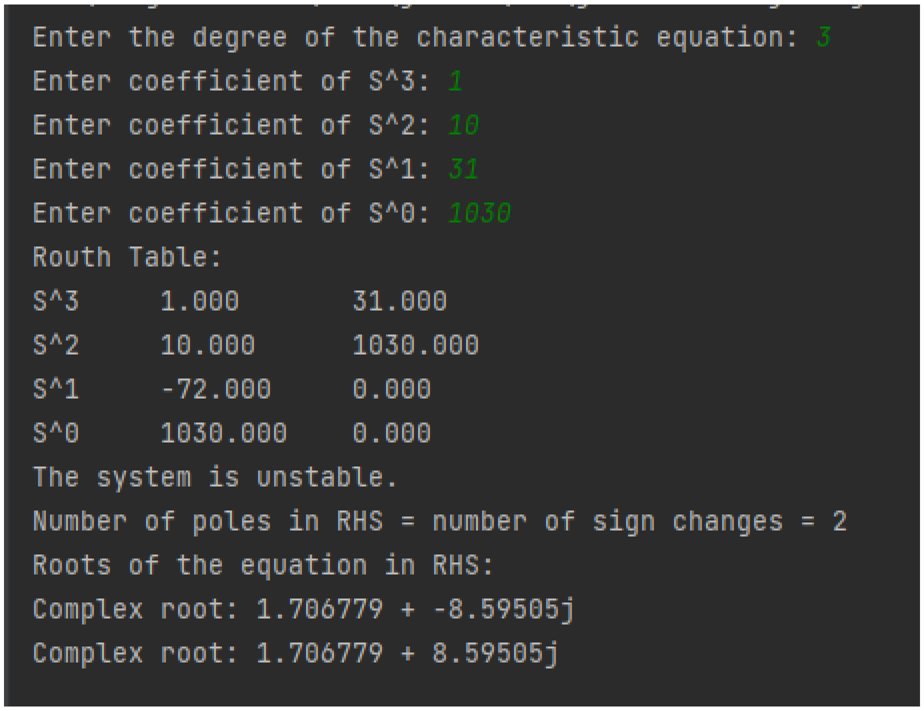
E-Sample Runs:

Test 1:



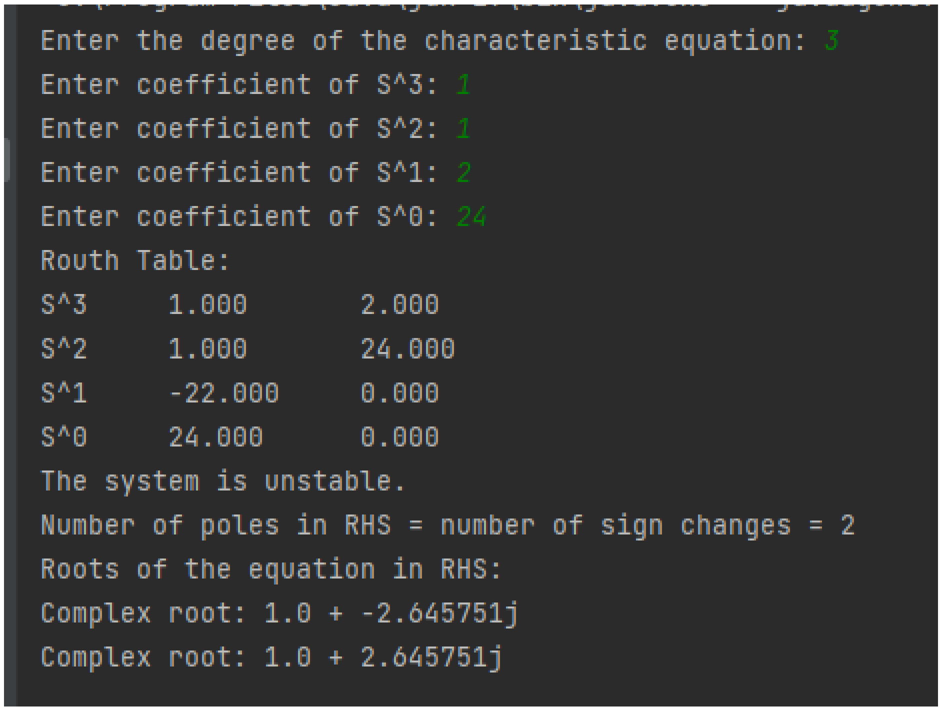
```
Enter the degree of the characteristic equation: 4
Enter coefficient of S^4: 1
Enter coefficient of S^3: 2
Enter coefficient of S^2: 3
Enter coefficient of S^1: 4
Enter coefficient of S^0: 5
Routh Table:
S^4     1.000       3.000       5.000
S^3     2.000       4.000       0.000
S^2     1.000       5.000       0.000
S^1     -6.000      0.000       0.000
S^0     5.000       0.000       0.000
The system is unstable.
Number of poles in RHS = number of sign changes = 2
Roots of the equation in RHS:
Complex root: 0.287815 + -1.416093j
Complex root: 0.287815 + 1.416093j
```

Test 2:



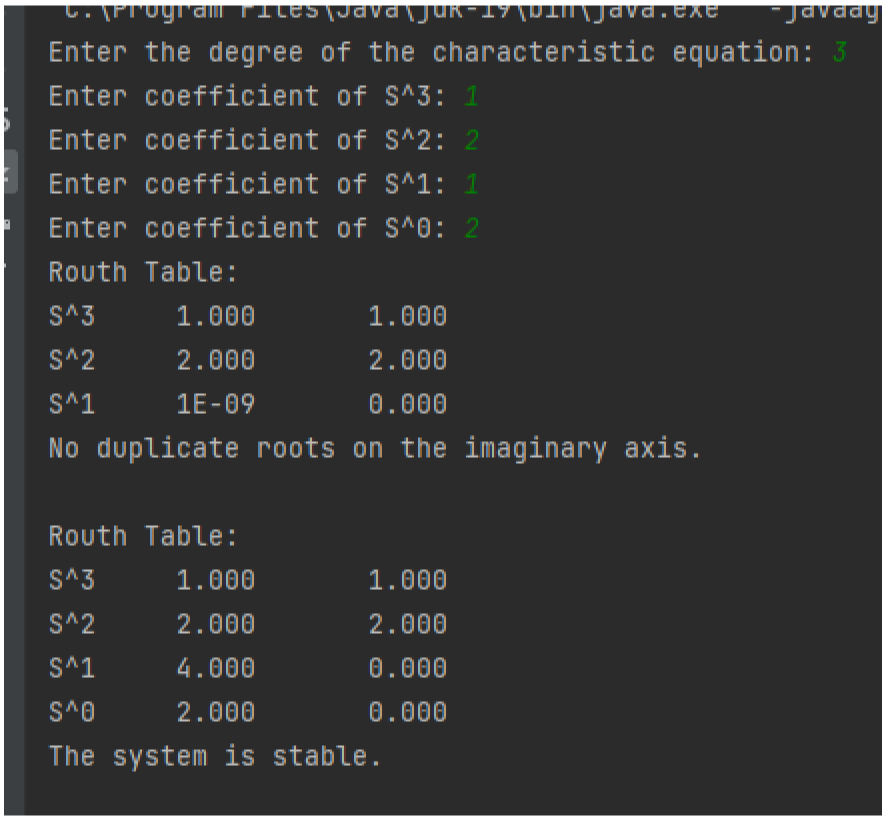
```
Enter the degree of the characteristic equation: 3
Enter coefficient of S^3: 1
Enter coefficient of S^2: 10
Enter coefficient of S^1: 31
Enter coefficient of S^0: 1030
Routh Table:
S^3     1.000       31.000
S^2     10.000      1030.000
S^1     -72.000     0.000
S^0     1030.000    0.000
The system is unstable.
Number of poles in RHS = number of sign changes = 2
Roots of the equation in RHS:
Complex root: 1.706779 + -8.59505j
Complex root: 1.706779 + 8.59505j
```

Test 3:



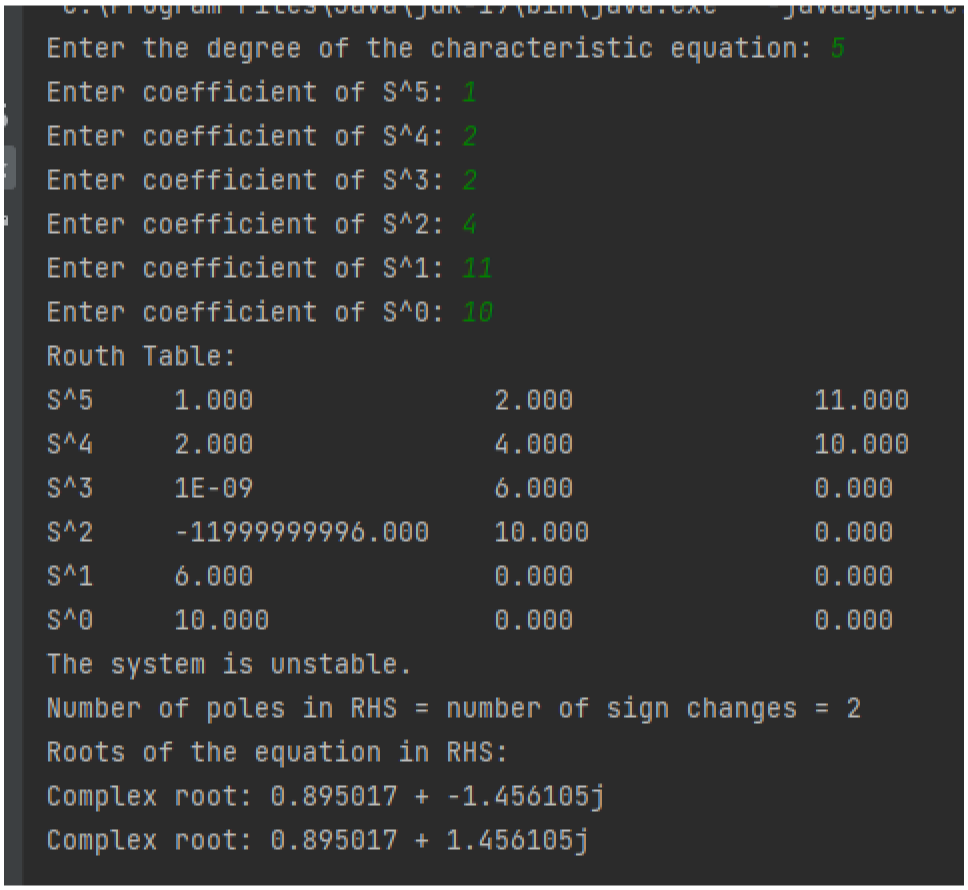
```
Enter the degree of the characteristic equation: 3
Enter coefficient of S^3: 1
Enter coefficient of S^2: 1
Enter coefficient of S^1: 2
Enter coefficient of S^0: 24
Routh Table:
S^3     1.000       2.000
S^2     1.000       24.000
S^1     -22.000     0.000
S^0     24.000      0.000
The system is unstable.
Number of poles in RHS = number of sign changes = 2
Roots of the equation in RHS:
Complex root: 1.0 + -2.645751j
Complex root: 1.0 + 2.645751j
```

Test 4:



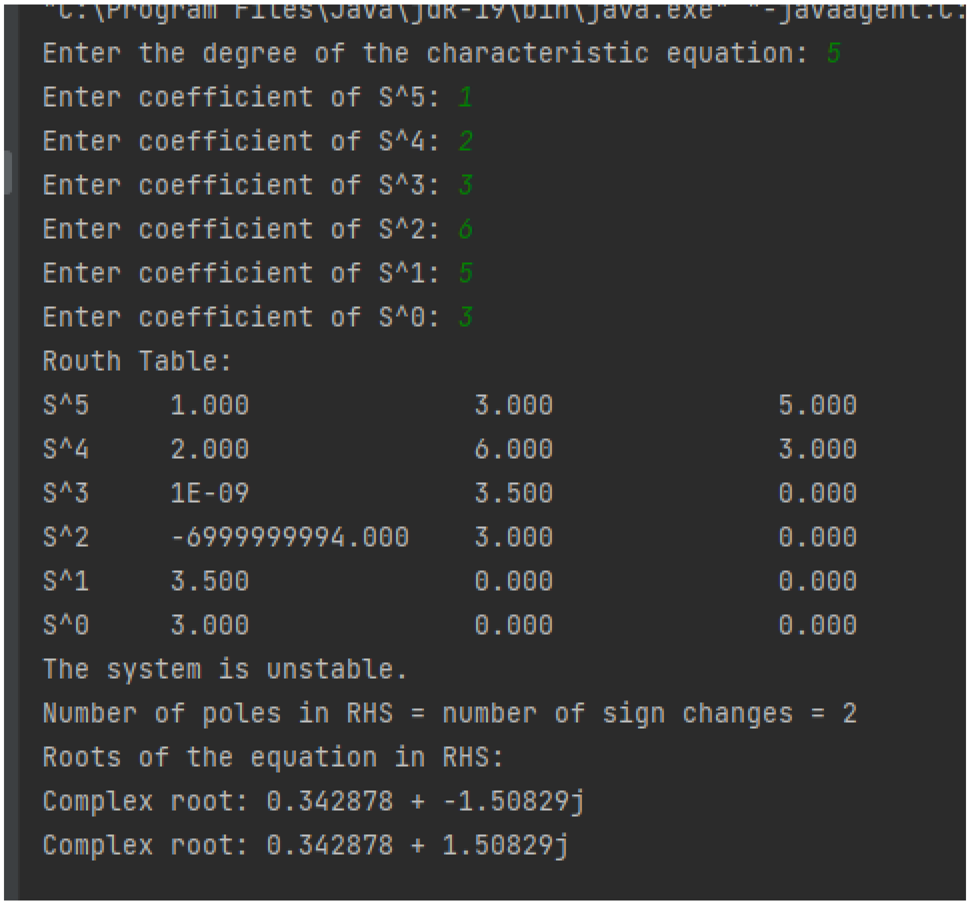
```
 C:\Program Files\Java\jdk-19\bin\java.exe  -javaag
Enter the degree of the characteristic equation: 3
Enter coefficient of S^3: 1
Enter coefficient of S^2: 2
Enter coefficient of S^1: 1
Enter coefficient of S^0: 2
Routh Table:
S^3     1.000       1.000
S^2     2.000       2.000
S^1     1E-09       0.000
No duplicate roots on the imaginary axis.

Routh Table:
S^3     1.000       1.000
S^2     2.000       2.000
S^1     4.000       0.000
S^0     2.000       0.000
The system is stable.
```

Test 5:



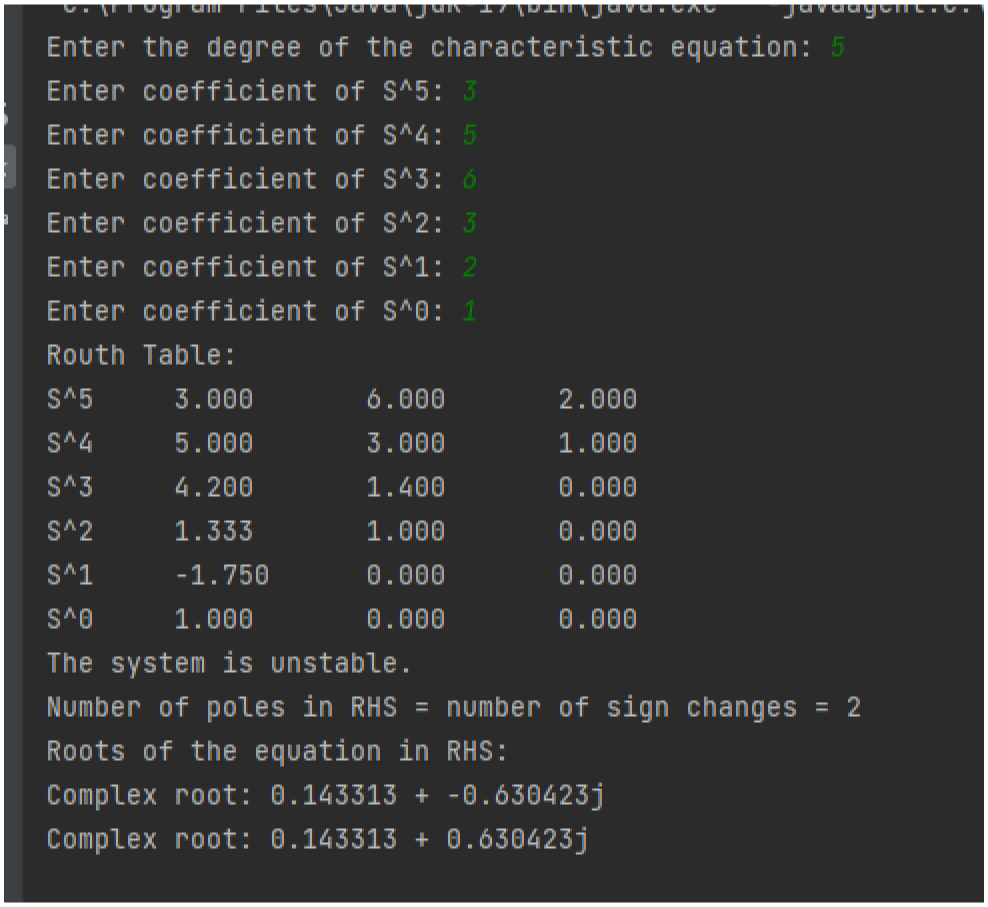
```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C
Enter the degree of the characteristic equation: 5
Enter coefficient of S^5: 1
Enter coefficient of S^4: 2
Enter coefficient of S^3: 2
Enter coefficient of S^2: 4
Enter coefficient of S^1: 11
Enter coefficient of S^0: 10
Routh Table:
S^5     1.000               2.000               11.000
S^4     2.000               4.000               10.000
S^3     1E-09               6.000               0.000
S^2     -11999999996.000    10.000              0.000
S^1     6.000               0.000               0.000
S^0     10.000              0.000               0.000
The system is unstable.
Number of poles in RHS = number of sign changes = 2
Roots of the equation in RHS:
Complex root: 0.895017 + -1.456105j
Complex root: 0.895017 + 1.456105j
```

Test 6:



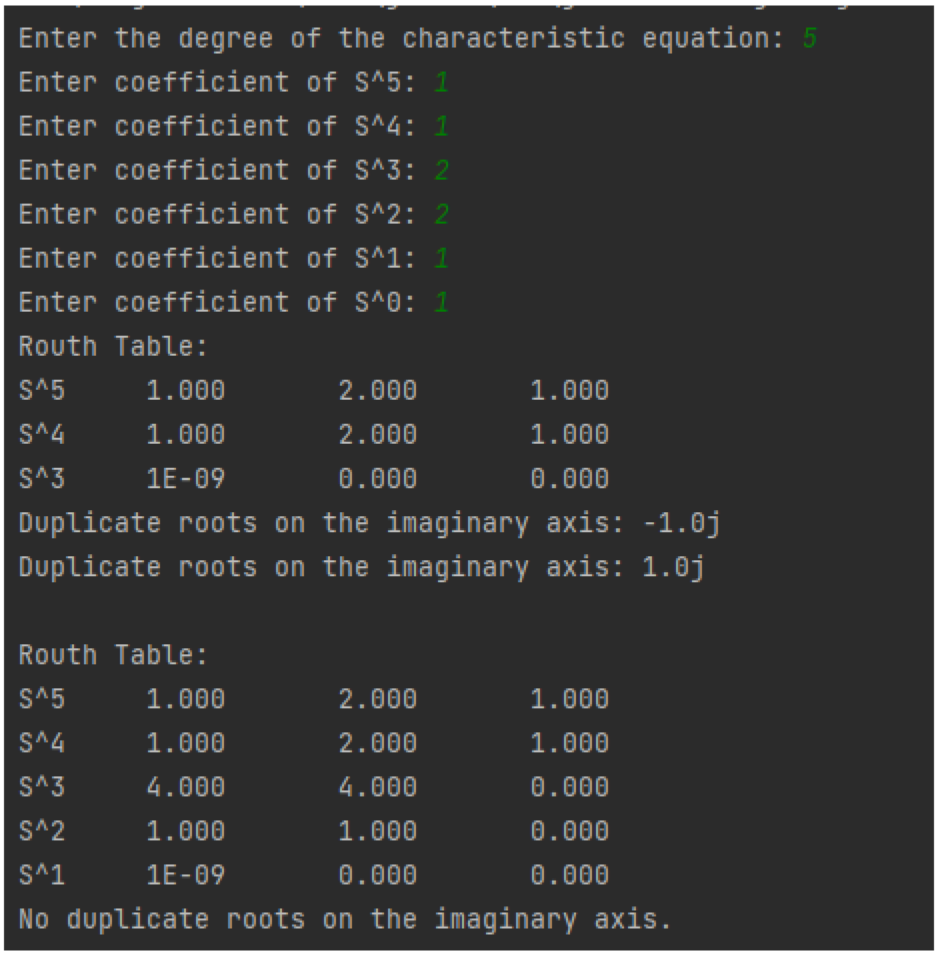
```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:
Enter the degree of the characteristic equation: 5
Enter coefficient of S^5: 1
Enter coefficient of S^4: 2
Enter coefficient of S^3: 3
Enter coefficient of S^2: 6
Enter coefficient of S^1: 5
Enter coefficient of S^0: 3
Routh Table:
S^5     1.000              3.000              5.000
S^4     2.000              6.000              3.000
S^3     1E-09              3.500              0.000
S^2     -6999999994.000    3.000              0.000
S^1     3.500              0.000              0.000
S^0     3.000              0.000              0.000
The system is unstable.
Number of poles in RHS = number of sign changes = 2
Roots of the equation in RHS:
Complex root: 0.342878 + -1.50829j
Complex root: 0.342878 + 1.50829j
```

Test 7:

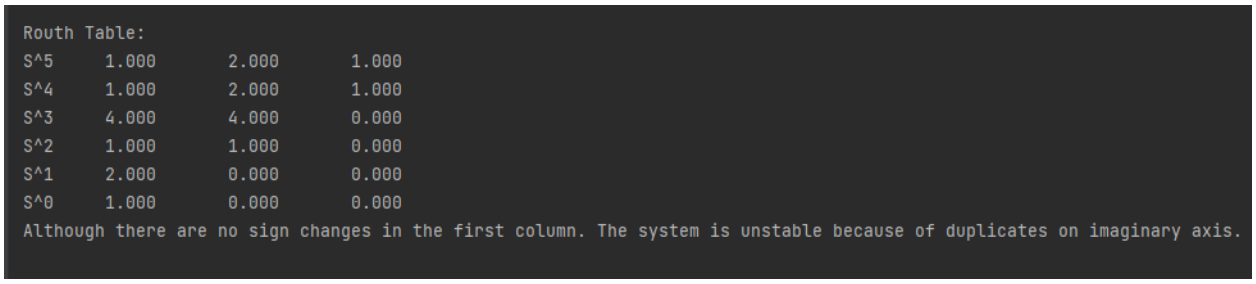


```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:
Enter the degree of the characteristic equation: 5
Enter coefficient of S^5: 3
Enter coefficient of S^4: 5
Enter coefficient of S^3: 6
Enter coefficient of S^2: 3
Enter coefficient of S^1: 2
Enter coefficient of S^0: 1
Routh Table:
S^5     3.000       6.000       2.000
S^4     5.000       3.000       1.000
S^3     4.200       1.400       0.000
S^2     1.333       1.000       0.000
S^1     -1.750      0.000       0.000
S^0     1.000       0.000       0.000
The system is unstable.
Number of poles in RHS = number of sign changes = 2
Roots of the equation in RHS:
Complex root: 0.143313 + -0.630423j
Complex root: 0.143313 + 0.630423j
```

Test 8:

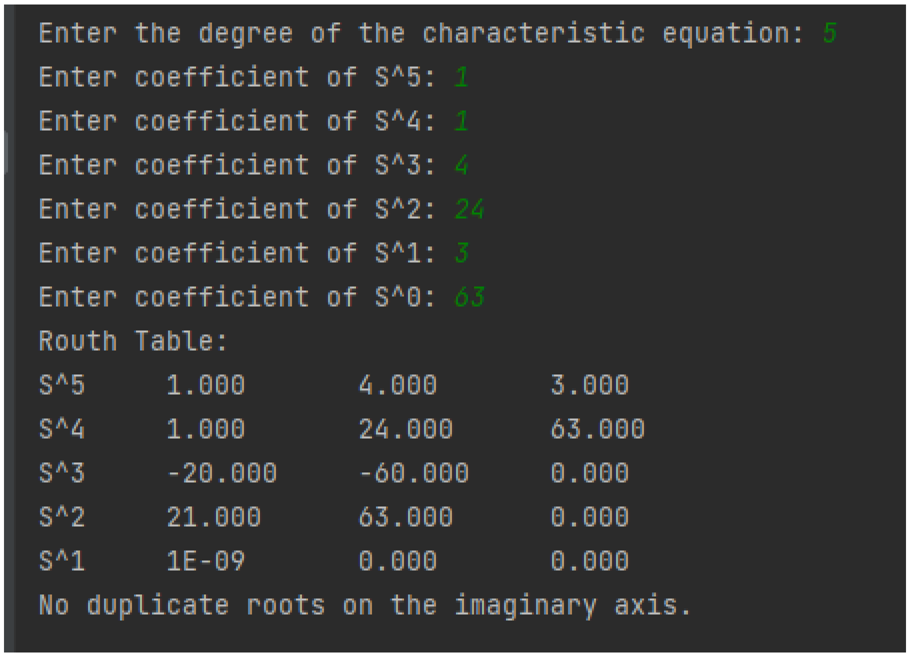


```
Enter the degree of the characteristic equation: 5
Enter coefficient of S^5: 1
Enter coefficient of S^4: 1
Enter coefficient of S^3: 2
Enter coefficient of S^2: 2
Enter coefficient of S^1: 1
Enter coefficient of S^0: 1
Routh Table:
S^5     1.000       2.000       1.000
S^4     1.000       2.000       1.000
S^3     1E-09       0.000       0.000
Duplicate roots on the imaginary axis: -1.0j
Duplicate roots on the imaginary axis: 1.0j

Routh Table:
S^5     1.000       2.000       1.000
S^4     1.000       2.000       1.000
S^3     4.000       4.000       0.000
S^2     1.000       1.000       0.000
S^1     1E-09       0.000       0.000
No duplicate roots on the imaginary axis.
```

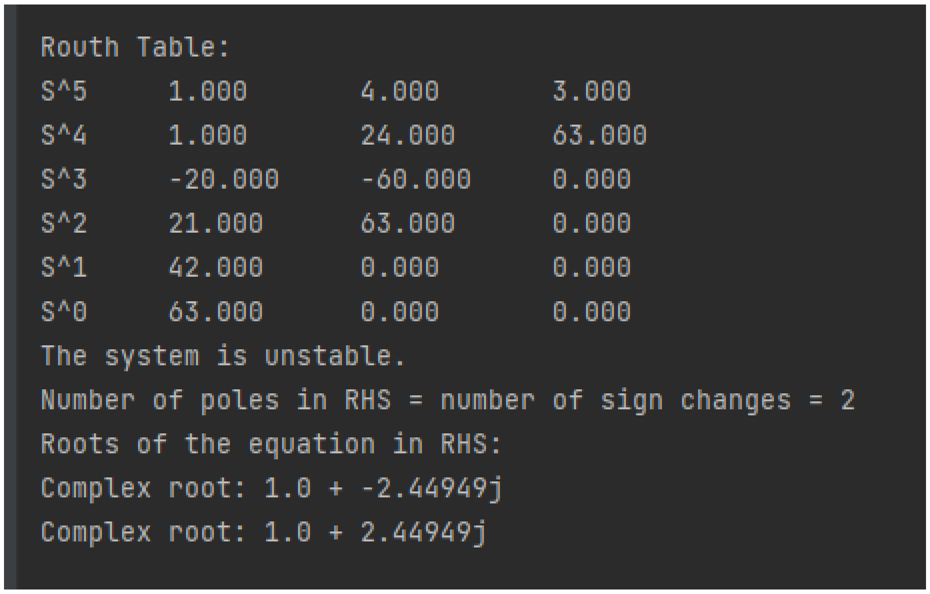


```
Routh Table:
S^5     1.000       2.000       1.000
S^4     1.000       2.000       1.000
S^3     4.000       4.000       0.000
S^2     1.000       1.000       0.000
S^1     2.000       0.000       0.000
S^0     1.000       0.000       0.000
Although there are no sign changes in the first column. The system is unstable because of duplicates on imaginary axis.
```

Test 9:



```
Enter the degree of the characteristic equation: 5
Enter coefficient of S^5: 1
Enter coefficient of S^4: 1
Enter coefficient of S^3: 4
Enter coefficient of S^2: 24
Enter coefficient of S^1: 3
Enter coefficient of S^0: 63
Routh Table:
S^5     1.000       4.000       3.000
S^4     1.000       24.000      63.000
S^3     -20.000     -60.000     0.000
S^2     21.000      63.000      0.000
S^1     1E-09       0.000       0.000
No duplicate roots on the imaginary axis.
```



```
Routh Table:
S^5     1.000       4.000       3.000
S^4     1.000       24.000      63.000
S^3     -20.000     -60.000     0.000
S^2     21.000      63.000      0.000
S^1     42.000      0.000       0.000
S^0     63.000      0.000       0.000
The system is unstable.
Number of poles in RHS = number of sign changes = 2
Roots of the equation in RHS:
Complex root: 1.0 + -2.44949j
Complex root: 1.0 + 2.44949j
```

F-User Guide:

The user starts by entering the degree of the equation that he wants to test its stability, then enters the coefficients of the variables in the equation one after the other starting from the coefficient of the highest degree variable (S^n) to the coefficient of the lowest degree variable (S^0).