Control Systems Basics

Lab 2

Signal Flow Graph

Students :

Ahmed Mohamed Hazem Mohamed	18015030
Ahmed Mohamed Abd-El Monem Mohamed	18010225
Ali Ahmed Ibrahim Fahmy	18011064
Fares Waheed Abd-El Hakim Abdallah	18011224
Mohamed Ibrahim El-Sayed Shaker	18011333
Mohamed Mofreh Abd-El Monem El Gazzar	18011626
Yousef Mohamed Huessien Ragab	18015031
Youssef Hany Fathy Shamsia	18015025

How TO RUN :

- Run the Vue.js Applications using VS codes.
- open the “controllab2” folder in VS codes.
- Make sure you have the extension “vuetify-vscode ” from VS market.
- Make sure you install yarn.
- Make sure you install node.js.
- Open terminal and write “yarn run serve” to run the application.
- The program will run at : http://localhost:8080/

1) Problem Statement:

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

2) Main Features of the program :

- Graphical user interface
- Draw the signal flow graph illustrating nodes, paths and gains
- Listing all forward paths, individual loops, and non-touching loops
- Listing the values of all deltas for forward paths
- Calculate overall system transfer function
- The user can modify an existing branch’s gain by adding a new gain with the paths nodes.

3) Data Structure:

- **We use graph data structure representing it by a list of nodes, each node consisting of edges. Each edge consists of 2 fields toNode & gain .**

4) Main modules:

```
/**
     * Getting all forward paths from node r to node c.
     * @param {*int} r index of input node.
     */
    find_all_forward_paths(r)
```

```
/**
     * Depth-First-Search for getting forward paths.
     * @param {*int} r index of input node.
     * @param {*forwardPath} path forward_path to be constructed.
     */
    dfs(r,path)
```

```
  /**
     * Method to get intersection between arrays.
     * @param {*LIST} a1 first list.
     * @param {*LIST} a2 second list.
     */
    getArraysIntersection(a1,a2)
```

```
/*
    This method adds the indices of all combinations of N non-touching
loops to the array nontouching_loops.
    The indices match those in the loops array of the graph.
    */
    find_all_nontouching_loops()
/**
     * Getting all loops in the graph
     */
    find_all_loops()
/**
     * Depth-First-Search for getting all loops
     */
    dfsLoops()
 /**
     * Depth-First-Search modified for getting a loop
     * @param {*int} r index of input node.
     * @param {*forwardPathLoop} path forward_path to be constructed.
     * @param {*int} f index of node that has a backwards edge towards
start of loop
     * @param {*int} start index of start of loop
     */
```

```
    dfsModified(r,path,f,start)
```

```
/**
     *
     * @returns Total delta of the system
     */
     delta()
/**
     *
     * @returns delta 1,2,..n for corresponding forward paths
     */
     deltas()
/**
     *
     * @returns Overall transfer function of the system
     */
     transf_fun()
```

5) Algorithms used:

Getting Forward Paths:

```
find_all_forward_paths(r){
       // set the visited array to false initially.
      visited = new Array(nodes.length).fill(false);
      path = new forwardPath();
      dfs(r,path);

  }

 dfs(r,path){
    if(r == c){  // c is the output node
      path.nodes.push(r);
       p = new forwardPath() ;
      p = JSON.parse(JSON.stringify(path)); // deep copy
      forwardPaths.push(p);
```

```
    return ; // end of the forward path.
   }
   this.visited[r] = true;
   path.nodes.push(r);
   for( i =0;i<this.nodes[r].edges.length;i++){
          e = this.nodes[r].edges[i] ;
         if(! visited[e.toNode]){
           p = new forwardPath() ;
           p = JSON.parse(JSON.stringify(path));
           p.gain = p.gain * e.gain;
           dfs(e.toNode,p);
                    }
            }
  visited[r] = false ;
  path.nodes.pop();
  return ;
  }
```

Detecting Loops:

```
  find_all_loops(){

    visitedArray = new Array(nodesArray.length).fill(false)

    dfsLoops()

 }

  dfsLoops(){

     lastLoopNodeIndex = null

     for k = 0 to k < nodesArray.length{

       startingNodeIndex = k

        for i = k to I < nodesArray.length {
```

```
            currentNode = nodesArray [i]
            for j = 0 to j < currentNode.edgesArray.length {
               cycle = new loop()
               lastLoopNodeIndex = null
               destination = currentNode. edgesArray [j].toNode
               if (destination == startingNodeIndex){
                  lastLoopNodeIndex = i
               }
               if (lastLoopNodeIndex != null){
                  dfsModified(startingNodeIndex,
cycle,lastLoopNodeIndex,startingNodeIndex)
               }
            }
         }
visitedArray [k] = true
       }
     return
     }


   dfsModified(r,path,f,start){
      if(r == f){
        path. nodesArray.push(r)
        path. nodesArray.push(start)
```

```
  p = new forwardPath()
  p = JSON.parse(JSON.stringify(path))
  for i =0 to i< nodesArray [r]. edgesArray.length{
    currentEdge = this. nodesArray [r]. edgesArray [i]
    if (currentEdge.toNode == start){
        p.gain = p.gain * currentEdge.gain
        break
    }
}
  loopsArray.push(p)
  return
}
visitedArray [r] = true
path. nodesArray.push(r)
for i = 0 to i<this. nodesArray [r]. edgesArray.length{
        e = this. nodesArray [r]. edgesArray [i]
        if(!visitedArray  [e.toNode]){
          p = new forwardPath()
          p = JSON.parse(JSON.stringify(path))
          p.gain = p.gain * e.gain
          dfsModified(e.toNode,p,f,start)
      }
```

```
 }
visitedArray  [r] = false
path. nodesArray.pop()
return
}
```

Determining Non-Touching Loops:

```
getArraysIntersection(a1,a2)
      {return  a1.filter(function(n) { return a2.indexOf(n) !== -1;})


non touching loops()

      for i=2 to i<len
      for j=0 to j<len-1

          intersection=list[j].nodes
          concatHolder=intersection
          counter=1
          innerRes=[]
```

```
            for k=j+1 to k<len

          intersection
=this.getArraysIntersection(concatHolder,list[k].nodes)

              if(intersection.length==0)

               if(!innerRes.includes(j))

                    innerRes.push(j);

               innerRes.push(k);

               counter++

               concatHolder=concatHolder.concat(list[k].nodes)

              if(counter==i)

               result.push(innerRes);

               intersection=list[j].nodes;

                   reset innerRes [] , concatHolder ,counter to 1

     this.nonTouchingLoops=result;
```

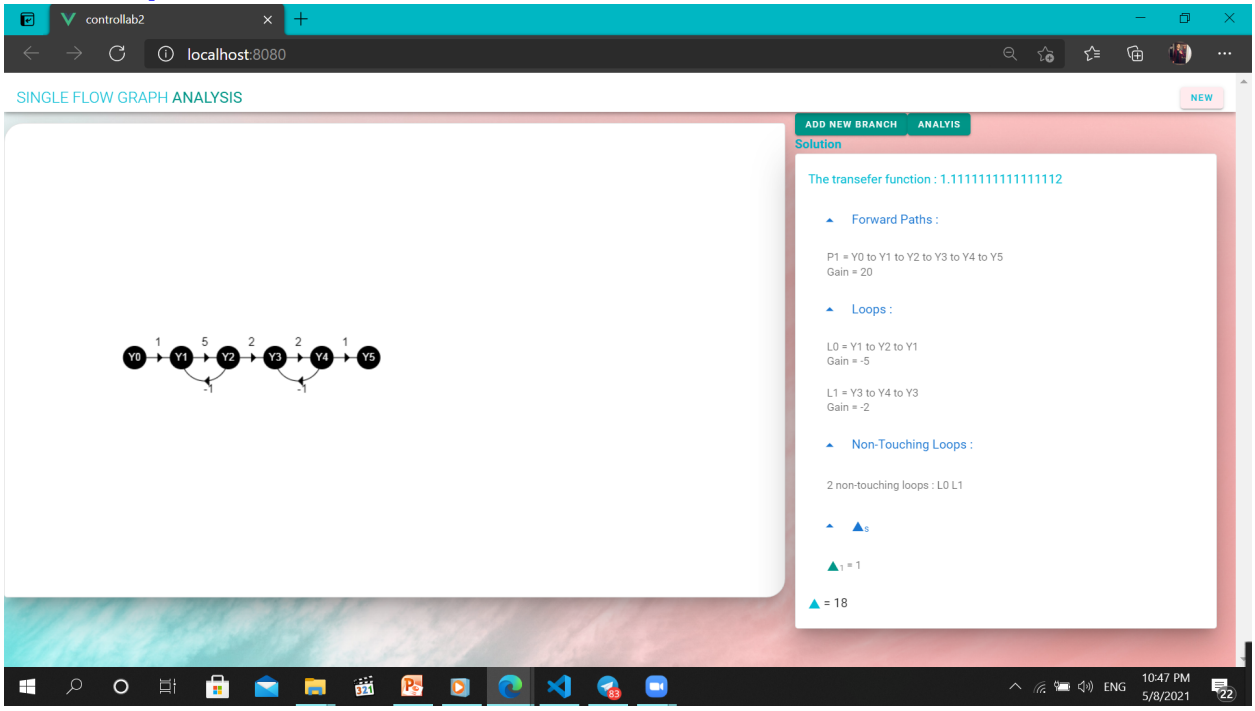
Deltas & transfer function:

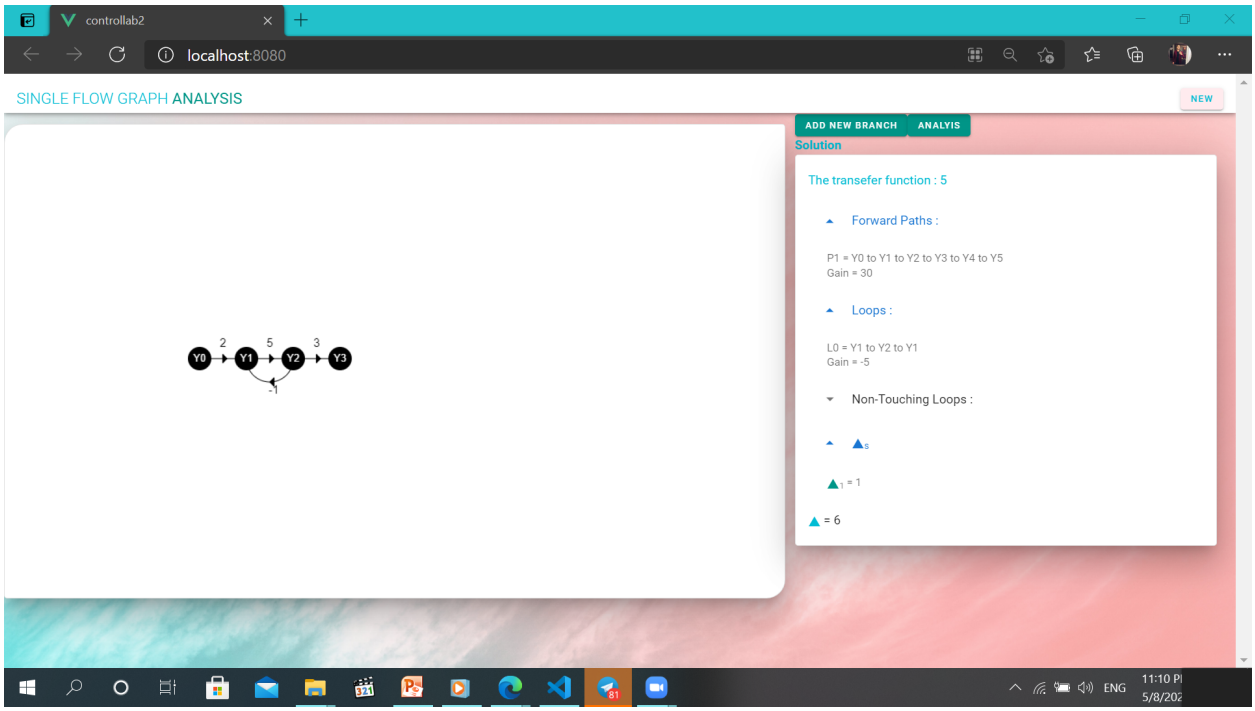
```
transf_fun() {
       t_f = 0.0;
      ds=deltas();
       d = delta();
    for ( i = 0; i <  forwardPaths.length; i++) {
     t_f +=   forwardPaths[i].gain * ds[i];
    }
    return t_f/d ; }
 not_mutual(a1, a2) {
    return this.getArraysIntersection(a1, a2).length == 0;
  }
```

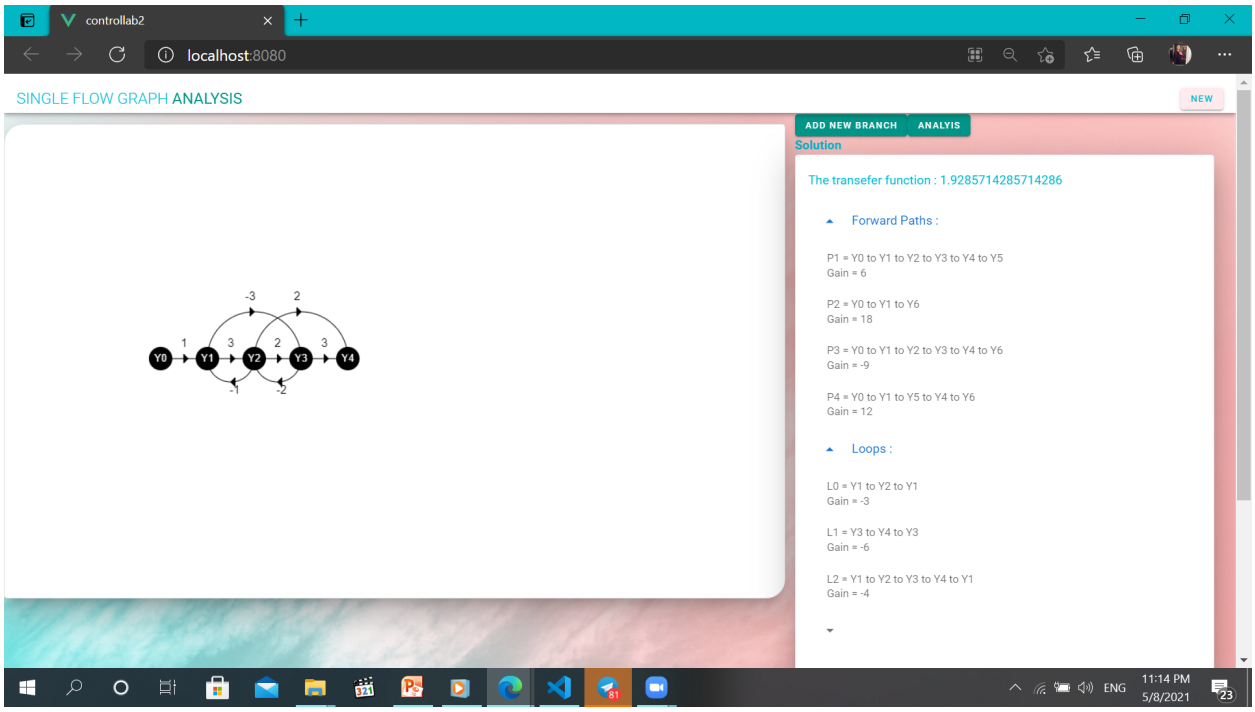
```
delta() {
      delta = 1;
    for ( z = 0; z < loops.length; z++) {
      delta -=  loops[z].gain;
    }
    for ( i = 0; i <  nonTouchingLoops.length; i++) {
      m = 1;
      for ( j = 0; j <  nonTouchingLoops[i].length; j++) {
        m *=  loops[  nonTouchingLoops[i][j] ].gain;
      }
      m *= Math.pow(-1,  nonTouchingLoops[i].length);
      delta += m;
    }
    return delta;
  }
deltas() {
      deltas = [];
   deltas.length = forwardPaths.length;
    for ( i = 0; i < deltas.length; i++) {
      deltas[i] = 1;
      for ( j = 0; j < loops.length; j++) {
        if (not_mutual( forwardPaths[i].nodes,  loops[j].nodes)) deltas[i] -=
loops[j].gain;
      }
      if (deltas[i] == 1) continue;
      for (var k = 0; k <  nonTouchingLoops.length; k++) {
        var m = 1;
        for (var f = 0; f <  nonTouchingLoops[k].length; f++) {
          var flag = false;
          if (! not_mutual( forwardPaths[i].nodes, loops[
nonTouchingLoops[k][f]].nodes)) {
            flag = true;
            break;
          }
          m *=  .loops[nonTouchingLoops[k][f]].gain;
        }
        m *= Math.pow(-1, nonTouchingLoops[k].length);
        if (!flag) deltas[i] += m;
      }
```

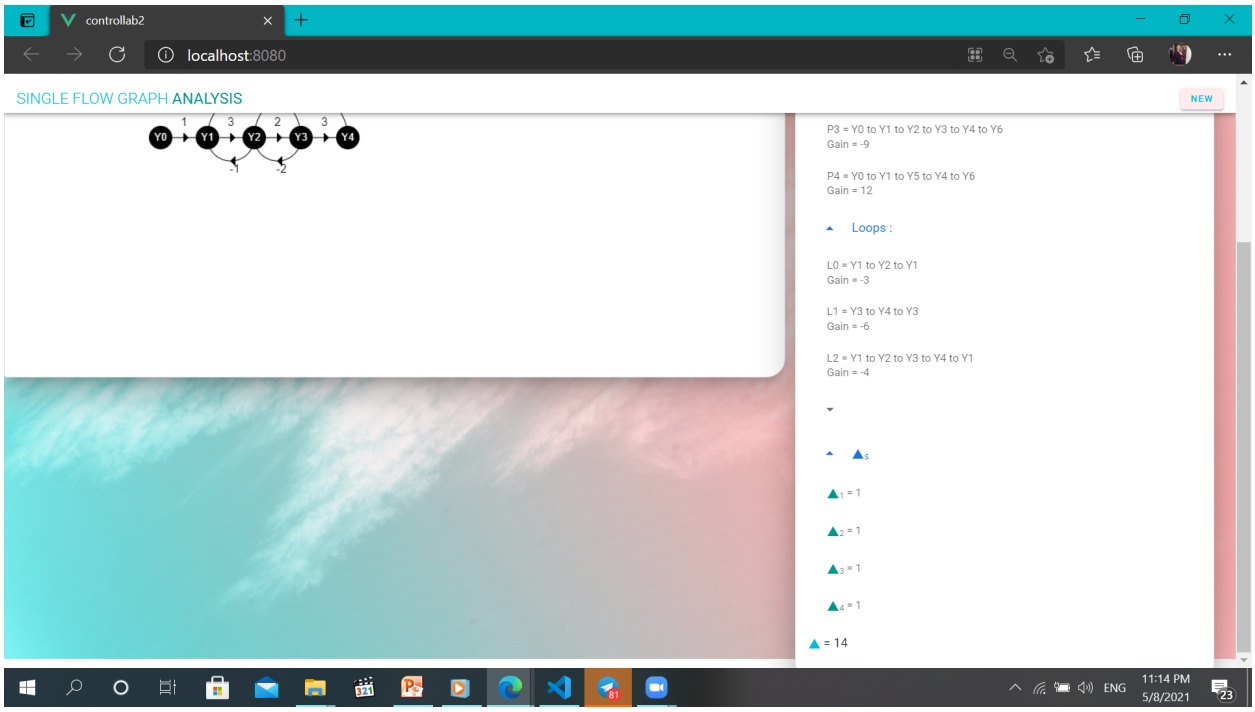
```
}
return deltas; }
```

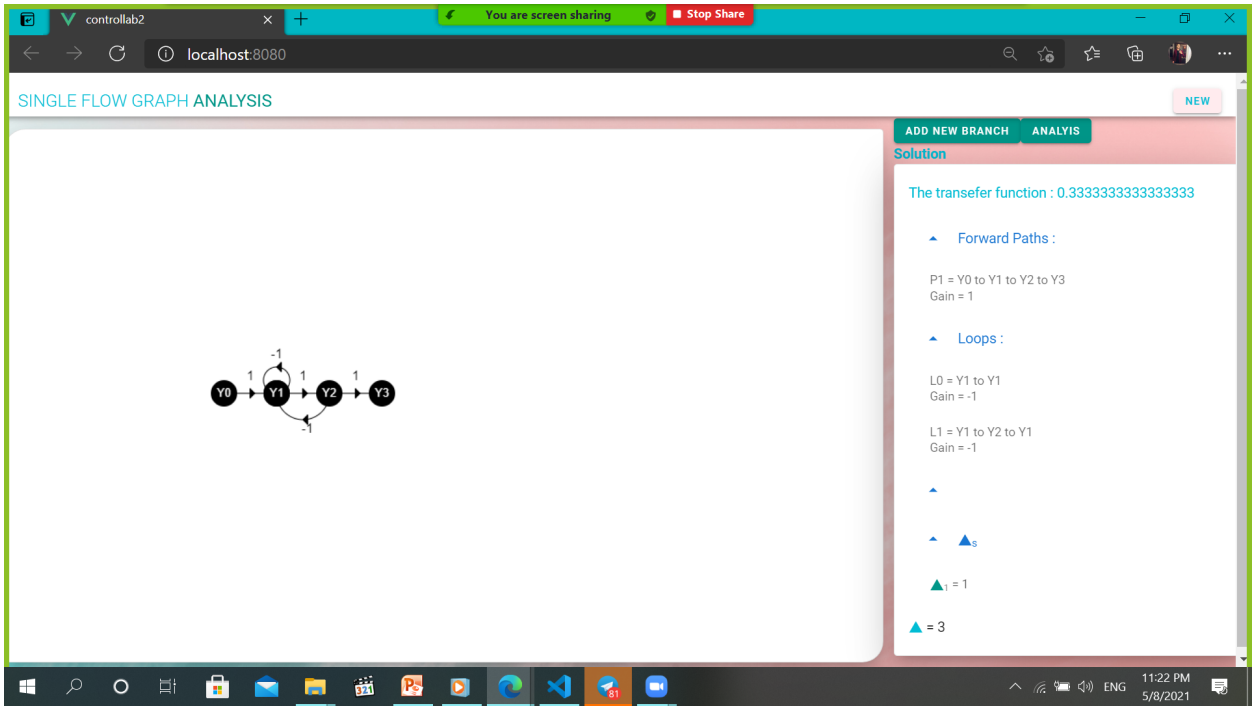
6) Sample runs:

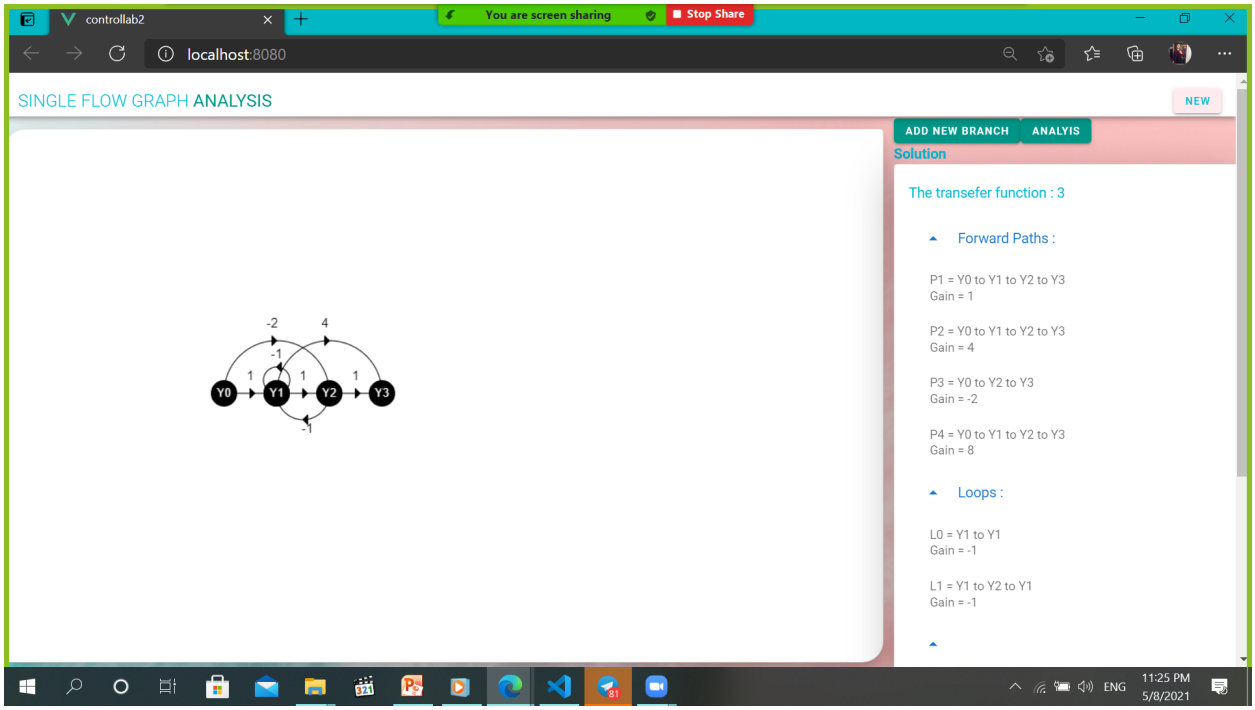


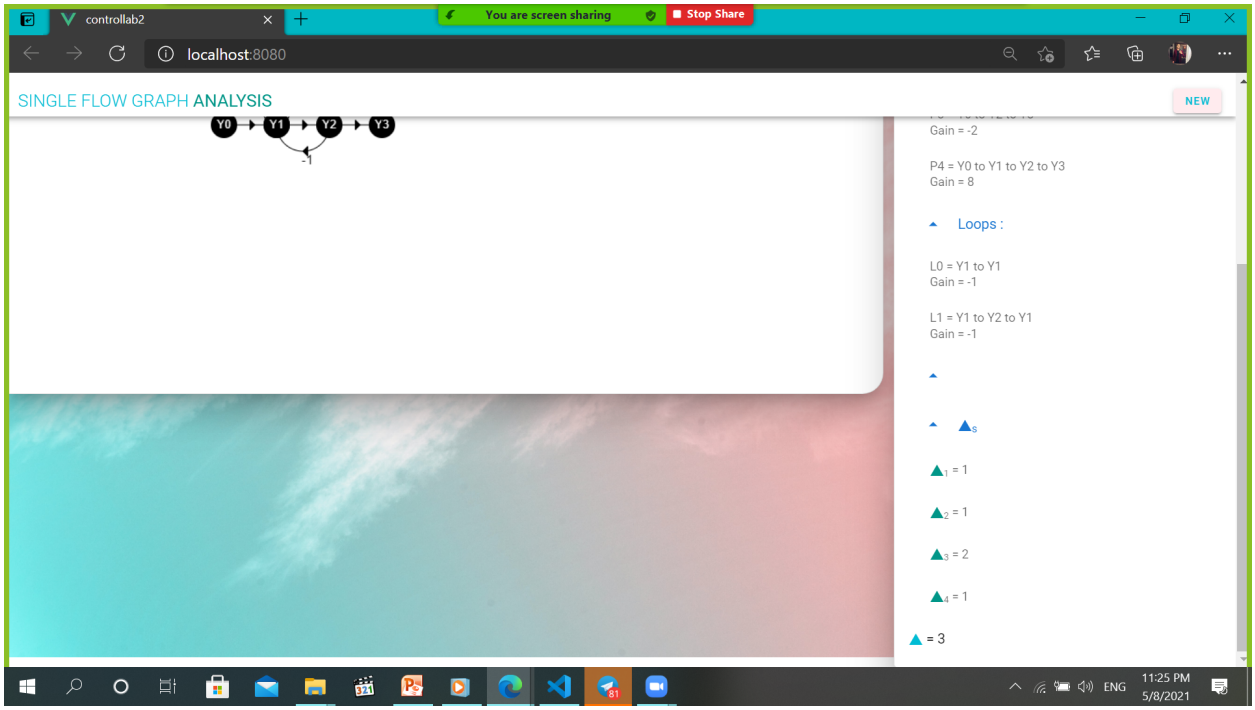



controllab2
localhost:8080
SINGLE FLOW GRAPH ANALYSIS
NEW
ADD NEW BRANCH
ANALYIS
Solution
The transefer function : 1.9285714285714286
Forward Paths :
P1 = Y0 to Y1 to Y2 to Y3 to Y4 to Y5
Gain = 6
P2 = Y0 to Y1 to Y6
Gain = 18
P3 = Y0 to Y1 to Y2 to Y3 to Y4 to Y6
Gain = -9
P4 = Y0 to Y1 to Y5 to Y4 to Y6
Gain = 12
Loops :
L0 = Y1 to Y2 to Y1
Gain = -3
L1 = Y3 to Y4 to Y3
Gain = -6
L2 = Y1 to Y2 to Y3 to Y4 to Y1
Gain = -4
Y0
Y1
Y2
Y3
Y4
1
3
2
3
-3
2
-1
-2
ENG
11:14 PM
5/8/2021


controllab2
localhost:8080
SINGLE FLOW GRAPH ANALYSIS
NEW
P3 = Y0 to Y1 to Y2 to Y3 to Y4 to Y6
Gain = -9
P4 = Y0 to Y1 to Y5 to Y4 to Y6
Gain = 12
Loops :
L0 = Y1 to Y2 to Y1
Gain = -3
L1 = Y3 to Y4 to Y3
Gain = -6
L2 = Y1 to Y2 to Y3 to Y4 to Y1
Gain = -4
Δ = 14
Y0
Y1
Y2
Y3
Y4
1
3
2
3
-1
-2
ENG
11:14 PM
5/8/2021


controllab2
You are screen sharing
Stop Share
localhost:8080
SINGLE FLOW GRAPH ANALYSIS
NEW
ADD NEW BRANCH
ANALYIS
Solution
The transefer function : 0.3333333333333333
Forward Paths :
P1 = Y0 to Y1 to Y2 to Y3
Gain = 1
Loops :
L0 = Y1 to Y1
Gain = -1
L1 = Y1 to Y2 to Y1
Gain = -1
Y0
Y1
Y2
Y3
ENG
11:22 PM
5/8/2021


controllab2
You are screen sharing
Stop Share
localhost:8080
SINGLE FLOW GRAPH ANALYSIS
NEW
ADD NEW BRANCH
ANALYIS
Solution
The transefer function : 3
Forward Paths :
P1 = Y0 to Y1 to Y2 to Y3
Gain = 1
P2 = Y0 to Y1 to Y2 to Y3
Gain = 4
P3 = Y0 to Y2 to Y3
Gain = -2
P4 = Y0 to Y1 to Y2 to Y3
Gain = 8
Loops :
L0 = Y1 to Y1
Gain = -1
L1 = Y1 to Y2 to Y1
Gain = -1
Y0
Y1
Y2
Y3
ENG
11:25 PM
5/8/2021



7) Simple user guide

- **First user selects the number of nodes.**
- **To add a new branch click on “add new branch” , select From node & To node and write the gain of this branch then click add.**
- **After adding all the branches click on the “analysis” button to show all required information .**
- **Assumption : the right-most node is the output node and the left-most node is the input node.**
- **To start again click on the “new” button.**