Task 1: Modelling of Non Linear Systems

Part - 01
Modeling of non-linear Dynamical Systems

In mathematics and science, a non-linear system is a system in which the change of the output is not linearly proportional to the change of the input. Non-linear problems are of interest to engineers, biologists, mathematicians etc because most systems that occur in nature are inherently non-linear.

Non-linear dynamical systems that describe changes in variable over time may often appear chaotic, unpredictable or counter-intuitive in nature, contrasting with much simpler linear systems.

Typically the behavior of a non-linear system is described as a set of simultaneous equations in which the unknowns (or unknown functions in the case of differential equations) appear as variables of a polynomial of degree higher than one. Such a system is called a **non-linear system of equations.**

We will deal with dynamical systems that are modeled by a finite number of coupled first order ordinary differential equations

$$\dot{x}_1 = f_1(t, x_1, \ldots\ldots, x_n, u_1, \ldots\ldots, u_p)$$

$$\dot{x}_2 = f_2(t, x_1, \ldots\ldots, x_n, u_1, \ldots\ldots, u_p)$$

$$\cdot\,\cdot$$

$$\cdot\,\cdot$$

$$\dot{x}_n = f_n(t, x_1, \ldots\ldots, x_n, u_1, \ldots\ldots, u_p)$$

Here **$\dot{x}_1$, $\dot{x}_2$,…$\dot{x}_n$** denote the derivative of **x_1, x_2…x_n** respectively with respect to time variable t and **u_1, u_2,… u_p** etc are specified input variables. We call the variables **x_1, x_2, …x_n** the **state variables**.

State Variables are used to to represent the memory the dynamical system has of its past or the desired variable of interest. We usually use vector notation to write these equations in a compact form.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}, u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_p \end{bmatrix}, f(t,x,u) = \begin{bmatrix} f_1(t,x,u) \\ f_2(t,x,u) \\ \vdots \\ \vdots \\ f_n(t,x,u) \end{bmatrix}$$

We can rewrite the n first-order differential equations as one n-dimensional first-order vector differential equation

$$\dot{x} = f(t, x, u)$$

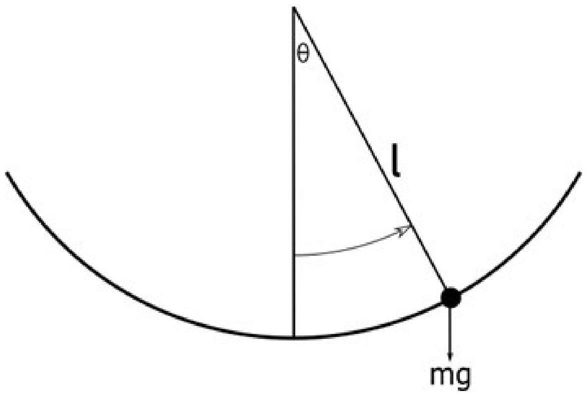
We call above equation as the **State Equation** of the system and refer to x as the **state** and u as the **input**.

Take the quiz - 1 on WHATSAPP PROJECT GROUP before moving forward.

--- END OF PART – 1 --

Task 1: Modelling of Non Linear Systems

Part – 2
Modeling of a Simple Pendulum



We will start with the mathematical modeling of a simple system.

Consider a simple pendulum shown in Fig , where l denotes the length of rod and m denotes the mass of the bob. Assume the rod is rigid and has zero mass. Let θ denote the angle subtended by the rod and the vertical axis through the pivot point.

The pendulum is free to swing in the vertical plane, The bob of the pendulum moves in a circle of radius l. To write the equations of motion, let us identify the forces acting on the bob.

1. Downward gravitational force **mg** where **g** is acceleration due to gravity.

2. Frictional force resisting motion which can be assumed to be proportional to the speed of the bob with a coefficient of friction **k**.

Using the Newton's second law of motion, the equation of motion for the bob in the tangential direction of motion can be written as

$$ml\ddot{\theta} = -mg\sin\theta - kl\dot{\theta}$$

To obtain the State Equation for the pendulum, the state variables can be assumed as

$$x_1 = \theta, x_2 = \dot{\theta}$$

The state equations for the pendulum model are:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -\frac{g}{l}\sin x_1 - \frac{k}{m}x_2$$

Task 1: Modelling of Non Linear Systems

It is possible to find the **equilibrium points** of this system by setting the **state derivatives** to zero and then solving for x1 and x2.

$$0 = x_2$$

$$0 = -\frac{g}{l}\sin x_1 - \frac{k}{m}x_2$$

The equilibrium points are located at (nπ,0) for n = ±1,±2,… . From the physical descriptions of the pendulum, it is clear that there are only two equilibrium positions (0,0) and (π,0). The rest of equilibrium points are just repetitions based on number of full swings of the pendulum.

Hence this is how a simple physical system is modeled.

Stable and Unstable Equilibrium Points

A typical problem that arises while dealing with non-linear dynamical systems is to check if a system is stable or unstable at a given equilibrium point.

Equilibrium Point of a system is the point at which the state of the system doesn't change. The equilibrium points can be estimated by setting $\dot{x}_1$=0 and $\dot{x}_2$ = 0 and solving the given equations for x_1 and x_2.

Stable Equilibrium - If a system always returns to the equilibrium point after small perturbations.

Unstable Equilibrium - If a system moves away from equilibrium point after small perturbations.

Consider the following set of coupled equations as given below.

$$\dot{x}_1 = -x_1 + 2x_1^3 + x_2$$

$$\dot{x}_2 = -x_1 - x_2$$

We will discuss the steps in order to compute the stability of system.

1. **Calculate the equilibrium points of the system by setting the values of $\dot{x}_1 = \dot{x}_2 = 0$.**

$$0 = -x_1 + 2x_1^3 + x_2$$

$$0 = -x_1 - x_2$$

By solving the above two equations, the values of equilibrium point are : **(0,0) , (-1,1) & (1,-1)**

2.Linearize the set of equations by calculating the Jacobian.

Behavior of Non-linear systems is very hard to analyse. Linearization is a method which involves creating a linear approximation of a non-linear system that is valid in a small region around the operating point (in this case, the equilibrium points).

Task 1: Modelling of Non Linear Systems

In order to linearize a set of equations, we need to first calculate the Jacobian for a set of equations. You can read more about Jacobian here

Consider the following:

The Jacobian J for the set of equations can be calculated as:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 - 1 & 1 \\ -1 & -1 \end{bmatrix}$$

3.Substitute the value of the equilibrium points in the matrix J to find 3 matrices J_1, J_2 and J_3.

$$J_1 = \begin{bmatrix} 6(0)^2 - 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix}$$

$$J_2 = \begin{bmatrix} 6(1)^2 - 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ -1 & -1 \end{bmatrix}$$

$$J_3 = \begin{bmatrix} 6(-1)^2 - 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ -1 & -1 \end{bmatrix}$$

4.Calculate the eigenvalues of each of the matrices.

Eigen values of state matrices represent poles of a system. The poles decides the stability of a system and if the poles are on the negative half of the complex plane i.e. they have the negative real part then the system is stable and if the real part is positive the system is unstable. A system is marginally stable if it has simple poles (non repeated) on imaginary axis and unstable if it is repeated. Hence by intuition you can see that for simple pendulum case the pendulum in downward position is stable and in upright position it is unstable. Find the Jacobian around the two equilibrium points and verify the same.

The eigenvalues can be calculated by constructing the characteristic equation of the matrix and equating it to zero.

For equilibrium point (0,0) :-

$$|sI - J_1| = 0$$

$$\left| s\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix} \right| = 0$$

$$\begin{vmatrix} (s+1) & -1 \\ 1 & (s+1) \end{vmatrix} = 0$$

$$(s+1)^2 + 1 = 0$$

$$s = (-1+i), (-1-i)$$

Task 1: Modelling of Non Linear Systems

For equilibrium point (-1,1) and (1,-1):

$$|sI - J_2| = 0$$

$$\left| s\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 5 & 1 \\ -1 & -1 \end{bmatrix} \right| = 0$$

$$\begin{vmatrix} (s-5) & -1 \\ 1 & (s+1) \end{vmatrix} = 0$$

$$(s-5)(s+1)+1=0$$

$$s = (2+2\sqrt{2}),(2-2\sqrt{2})$$

5.Check the real part of the eigenvalues calculated for each equilibrium points.

For each Equilibrium point

- If all the eigenvalues have negative real part, the system is **Stable** at the given Equilibrium point.
- If even one of the eigenvalues has positive real part, the system is **Unstable** at the given Equilibrium point.

For Equilibrium Point (0,0), the eigenvalues are (-1-i) and (-1+i). Since both eigenvalues have negative real part, the system is **Stable**.

For Equilibrium Points (1,-1) and (-1,1), the eigenvalues are (2+2√2) and (2-2√2). Since one of the eigenvalues has a positive real part, the system is **Unstable**.

Take the quiz - 2 on WHATSAPP PROJECT GROUP before moving forward:

------------------------------------ END OF PART – 2 --

Task 1: Modelling of Non Linear Systems

Part – 3

PRACTICAL (AB AAYEGA MAZA)

Task Instructions

In this task, we will explain how to solve a system of non-linear equations to find the equilibrium points and then check the stability of the system at the given equilibrium points using Octave. The theory of this has already been explained in previous sections. Please make sure you have read and thoroughly understood that document.

If you are new to Octave we would advise you to go through this youtube link. Octave syntax is very easy to understand. If you have prior experience with MATLAB, Python (or pretty much any programming language), then it shouldn't be hard to grasp.

Download and extract Task1.zip ,file to get all the required files for this task. In the **Task1** folder, you will find the following scripts.

1.Main_File.m

2.Function_File.m

3.Test_Suite.m

We will examine the following files one-by-one.

1. **Main_File.m**

```
1. 1;
2. Function_File;
3. pkg load symbolic              # Load the octave symbolic library
4. syms x1 x2                     # Define symbolic variables x1 and x1
5. x1_dot = -x1 + 2*x1^3 + x2;    # Write the expressions for x1_dot and x2_dot
6. x2_dot = -x1 - x2;             # YOU CAN MODIFY THESE 2 LINES TO TEST OUT OTHER EQUATIONS
7. [x_1 x_2  jacobians  eigen_values  stability] = main_function (x1_dot, x2_dot);
```

Lets now look at the main file line by line:
Line 3 - It loads the symbolic library for Octave. When we want to define equations containing variables like x, y, z etc, we need to use the symbolic library.
Line 4 - Declares 2 symbolic variables x1 and x2 (to denote x_1 and x_2)
Line 5-6 - Defines a coupled set of non-linear equations using x1 and x2 (x1_dot and x2_dot are used to denote $\dot{x}_1$ and $\dot{x}_2$).
Line 7 - It is a function call to a function named main_function(). main_function() is defined in Function_File.m and takes 2 arguments (x1_dot and x2_dot). We will explain this function later.
Main_File can be used to test your code for different sets of equations on your side. You can change the x1_dot and x2_dot values to specify different equations for your code to test. **Please do not modify any other line in this script.**

Task 1: Modelling of Non Linear Systems

2. **Function_File.m (This is where the magic happens!!)**
There are 4 functions defined in this script. You are required to complete 3 of them in order to complete the task successfully.

```
find_equilibrium_points()

1. function [x_1, x_2] = find_equilibrium_points(x1_dot, x2_dot)

2. x1_dot == 0;

3. x2_dot == 0;

######################ADD YOUR CODE HERE##########################

##################################################################
4. x_1=double(x_1);

5. x_2=double(x_2);

6. endfunction
```

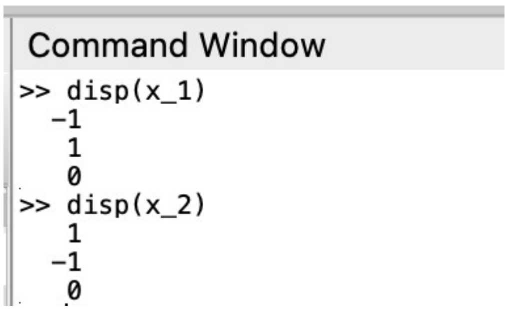
Lets now look at the function line by line:
This function is pretty straightforward
find_equilibrium_points() takes **x1_dot** and **x2_dot** as arguments.

Line 2 and 3 equate the expressions specified by **x1_dot** and **x2_dot** equal to zero.

The function returns the set of equilibrium points for **x1_dot = 0** and **x2_dot = 0**. This set is stored in the array **[x_1, x_2]**.

Please complete the code in this function so that it calculates the equilibrium points for the set of equations and stores it in the variable **[x_1, x_2].**

When you display **[x_1, x_2]** (using disp() function in octave), the structure should be as shown:



```
Command Window
>> disp(x_1)
  -1
   1
   0
>> disp(x_2)
   1
  -1
   0
```

Task 1: Modelling of Non Linear Systems

```
find_jacobian_matrices()

1. function jacobian_matrices = find_jacobian_matrices(x_1, x_2, x1_dot, x2_dot)

2. syms x1 x2;

3. solutions = [x_1, x_2];

4. jacobian_matrices = {};

#####################   ADD YOUR CODE HERE  #####################

#################################################################

5. endfunction
```

•This function takes the **equilibrium points (x_1, x_2)** and **x1_dot** and **x2_dot** as arguments.

•It computes the jacobian matrix for **x1_dot** and **x2_dot** (It should be a 2x2 symbolic array).

•It then substitutes calculated values of **x1** and **x2** for each of the equilibrium points. This function returns a variable called **jacobian_matrices**. It is a cell array in which each element is a 2x2 J matrix calculated for each of the corresponding equilibrium points.

•You are not allowed to change any code already written in this function. You are required to add your own code in the space provided.

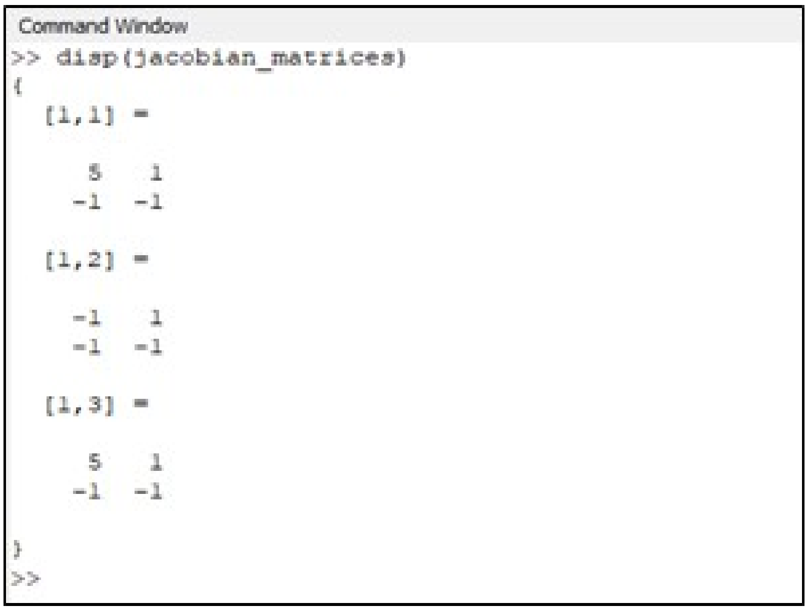
•Line 3 - solution x_1 & x_2 are combined in one array **solutions**

•Line 4 - empty cell array **jacobian_matrices** is initialized.

•You are required to add code which does the following :

- Computes the jacobian of **x1_dot** and **x2_dot**.
- For each of the equilibrium points, substitute the calculated values of x1 and x2 in the jacobian and form a 2x2 matrix.
- Store that jacobian matrix as an element of the cell array **jacobian_matrices**.
- When you display the jacobian matrices cell array (using disp() function in octave), the cell array structure should be similar to the following:

Task 1: Modelling of Non Linear Systems



```
Command Window
>> disp(jacobian_matrices)
{
  [1,1] =

     5   1
    -1  -1

  [1,2] =

    -1   1
    -1  -1

  [1,3] =

     5   1
    -1  -1

}
>>
```

```
check_eigen_values()

function [eigen_values stability] =  check_eigen_values(x_1, x_2, jacobian_matrices)

stability = {};

eigen_values = {};

for k = 1:length(jacobian_matrices)

     matrix = jacobian_matrices{k};

     flag = 1;

#####################   ADD YOUR CODE HERE  #####################

#################################################################

     if flag == 1

         fprintf("The system is stable for equilibrium point(%d, %d)\n",double(eqbm_pts{k}.x1),double(eqbm_pts{k}.x2));

         stability{k} = "Stable";

     else

         fprintf("The system is unstable for equilibrium point (%d, %d) \n",double(eqbm_pts{k}.x1),double(eqbm_pts{k}.x2));
         stability{k} = "Unstable";

     endif

endfor

endfunction
```

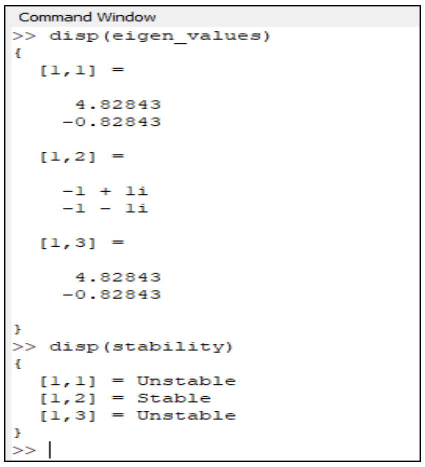
•This function takes the **x_1, x_2** and **jacobian_matrices** as input. For each jacobian matrix stored in **jacobian_matrices**, the eigenvalues of matrix are calculated and stored in the cell array **eigen_values**. Subsequently the eigenvalues are checked in this function. If for any jacobian matrix the eigenvalues have positive real part, the system is unstable at the corresponding equilibrium point. If all eigenvalues have negative real part, the system is stable at the corresponding equilibrium point.

Task 1: Modelling of Non Linear Systems

•Two empty cell arrays **stability** and **eigen_values** are defined. A for-loop is iterated through the length of **jacobian_matrices**. Within the for-loop, flag = 1 is initialized. You are required to write code which does the following:

- Find out the eigenvalues for the current jacobian matrix (value stored in matrix)
- Check real part of all eigenvalues of the matrix. If all eigenvalues have negative real part, then flag is set equal to 1.
- If even one eigenvalue is positive (greater than zero), the flag is set to 0.
- Store the eigenvalues calculated in the cell array **eigen_values**.

•Based on value of flag, the stability of system is reported in the if-else statement.
•When you display the **eigen_values** and **stability** cell arrays (using disp() in octave) the output should be similar to the following:



```
>> disp(eigen_values)
{
  [1,1] =

     4.82843
    -0.82843

  [1,2] =

    -1 + 1i
    -1 - 1i

  [1,3] =

     4.82843
    -0.82843

}
>> disp(stability)
{
  [1,1] = Unstable
  [1,2] = Stable
  [1,3] = Unstable
}
>> |
```

```
main_function()

function [x_1 x_2 jacobians eigen_values stability] = main_function(x1_dot, x2_dot)

            pkg load symbolic;

            syms x1 x2;

            [x_1, x_2] = find_equilibrium_points(x1_dot, x2_dot);

            jacobians = find_jacobian_matrices(x_1, x_2, x1_dot, x2_dot);

            [eigen_values stability] = check_eigen_values (x_1, x_2, jacobians);

endfunction
```

•This function puts together all the pieces.

•It takes x1_dot and x2_dot as argument. First the equilibrium points are calculated. For each equilibrium point, the jacobian matrix is calculated. Then the stability for each equilibrium point is determined by computing the eigen_values and checking the real parts of eigen values.

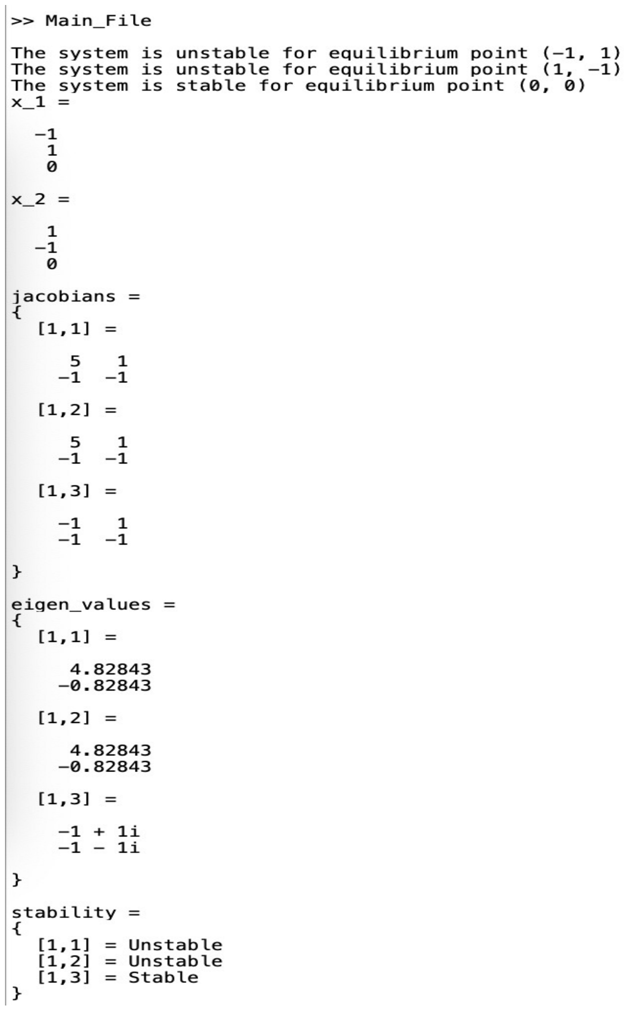
•This equation returns **x_1, x_2**, **jacobians**, **eigen_values**, **stability**.

Task 1: Modelling of Non Linear Systems

•You are not allowed to make any changes to this function. You need to run it as it is.

•After you have modified the functions explained above as instructed. You need to test your solution.

•To test your script, you need to run Main_File.m in octave. If your solution is correct you will see the following octave prompt:



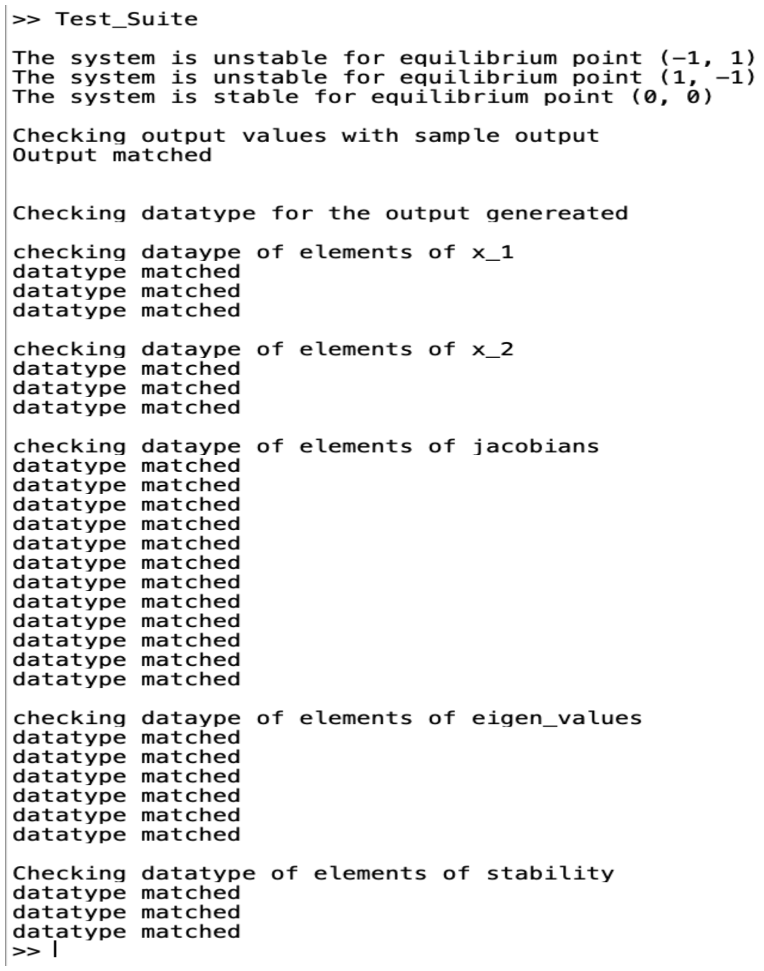
```
>> Main_File

The system is unstable for equilibrium point (-1, 1)
The system is unstable for equilibrium point (1, -1)
The system is stable for equilibrium point (0, 0)
x_1 =

  -1
   1
   0

x_2 =

   1
  -1
   0

jacobians =
{
  [1,1] =

     5   1
    -1  -1

  [1,2] =

     5   1
    -1  -1

  [1,3] =

    -1   1
    -1  -1

}

eigen_values =
{
  [1,1] =

     4.82843
    -0.82843

  [1,2] =

     4.82843
    -0.82843

  [1,3] =

    -1 + 1i
    -1 - 1i

}

stability =
{
  [1,1] = Unstable
  [1,2] = Unstable
  [1,3] = Stable
}
```

3. Test_Suite.m (Testing your solution)

•Test_Suite.m is used for testing your solution. You are not allowed to make any changes in this script.

•The Test_Suite script has a set of non-linear equations. If your code runs successfully for the given equations then the output should be as follows:

Task 1: Modelling of Non Linear Systems



```
>> Test_Suite

The system is unstable for equilibrium point (-1, 1)
The system is unstable for equilibrium point (1, -1)
The system is stable for equilibrium point (0, 0)

Checking output values with sample output
Output matched


Checking datatype for the output genereated

checking dataype of elements of x_1
datatype matched
datatype matched
datatype matched

checking dataype of elements of x_2
datatype matched
datatype matched
datatype matched

checking dataype of elements of jacobians
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched

checking dataype of elements of eigen_values
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched
datatype matched

Checking datatype of elements of stability
datatype matched
datatype matched
datatype matched
>> |
```

• If your Test_Suite runs successfully, Congratulations your task is completed.

--- ENJOY --

-- (BCZ IN NEXT TASK, U WILL NOT) ---