Lunar Scout - Task 0B - Practical

Hyperactive Leader Aug 2023

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 **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 **Task0B.zip** file to get all the required files for this task. In the **Task_0B** folder, you will find the following scripts.

1. Main_File.m
2. Function_File.m
3. Test_Suite.m
4. task0b.pyc

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

Note:

- First activate your conda environment using the following command:

```
conda activate LS_<Team_ID>
```

- For **Ubuntu 20.04 & 22.04** open Octave in the conda environment using the following command :

```
/usr/local/OCTAVE/6.4.0/bin/octave --g
```

- For **Windows** open Octave in the conda environment using the following command in the cmd:

```
C:\Octave\Octave-5.2.0.0\octave.vbs
```

- For **mac OS** open Octave in the conda environment using the following command in the terminal:

```
/Applications/Octave-5.2.0.app/Content
```

---

1. Main_File.m

```
1. 1;
2. Function_File;
3. pkg load symbolic                # Load t
4. syms x1 x2                       # Define
5. x1_dot = -x1 + 2*x1^3 + x2;      # Write
6. x2_dot = -x1 - x2;               # YOU CA
7. [x_1 x_2  jacobians  eigen_values  stab
```

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.**

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_

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:

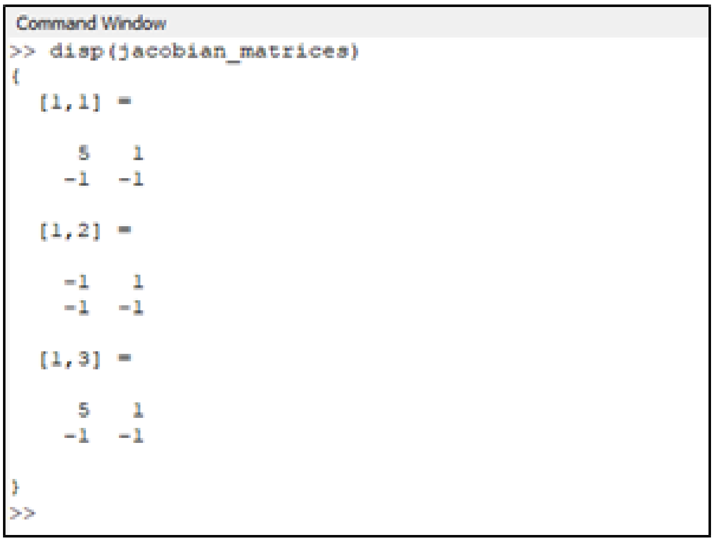
Topics
My Posts
More
Categories
Lunar Scout (LS)
Guidelines
ChatbotDocs
All categories
Tags
task-0
task-1
task-2
other
task-4
All tags
Skip to main content

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

```
find_jacobian_matrices()

1. function jacobian_matrices = find_jacob

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:



```
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] =  ch
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
```

Skip to main content

```
            stability{k} = "Stable";
        else
            fprintf("The system is unstabl
            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.
- 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)

      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:

```
Command Window
>> 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_value
        pkg load symbolic;
        syms x1 x2;
        [x_1, x_2] = find_equilibrium_
        jacobians = find_jacobian_matr
        [eigen_values stability] = che
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**.
- You are not allowed to make any changes to this function. You need to run it as it is.

Skip to main content

- 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:

Skip to main content

```
>> 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, your **Function_File.m** file is ready to be evaluated

4. test0b.pyc (Submitting task0b)

1. Open new Terminal (on Ubuntu OS or MacOS) or Anaconda Prompt (on Windows OS) and navigate to the **Task0B** folder.

2. Activate your conda environment with the command

```
conda activate LS_<team_id>
```

   Example: conda activate LS_9999

```
# To activate this environment, use
#
#     $ conda activate LS_9999
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Figure 1: conda activate in terminal

3. Run the **task0B.pyc** by running command

```
python task0b.pyc
```

Skip to main content

4. When asked, you have to enter your Team ID, such as 9999.

5. It will ask you to drag and drop a file named “octave-cli” as shown below in animated figure:

   - You can find the **“octave-cli”** in the following location based on your OS

     - **Windows**

       `C:\Octave\Octave-5.2.0.0\mingw64\bin`

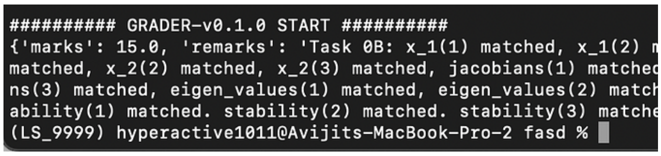
     - **Ubuntu(20.04 & 22.04)**

       `/usr/local/OCTAVE/6.4.0/bin`

     - **macOS**

       `/Applications/Octave-5.2.0.app/Contents/Resources/usr/Cellar/octave-octave-app@5.2.0/5.2.0/bin/`

- If you get the following outcome on your terminal, you are ready for submission :

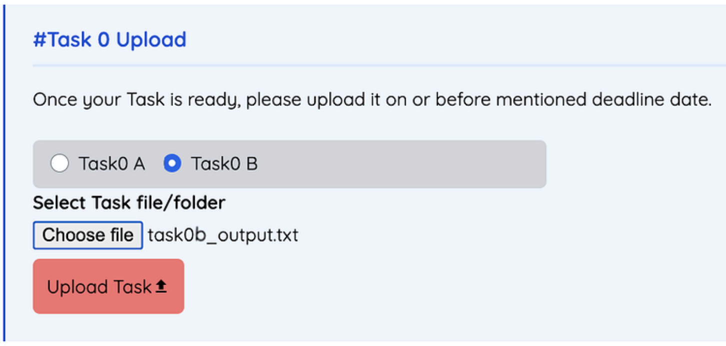


- For successful completion of **Task 0B**, upload the **`task0b_output.txt`** file on the portal. (**Sign in to eyantra**)

- Click on this link: **https://portal.e-yantra.org/task_task0** . In the **Task 0 Upload** section, click on **Task0 B** bullet and select **Choose file** button to upload the **`task0b_output.txt`** file. From the dialogue box, select the file and click **Open**.

- You shall see the file name **task0b_output.txt** in text-box besides the **Choose file** button as in figure below. Click on **Upload Task** button to submit the file.

Skip to main content



That's it !! Task 0B is complete!!

Congrats!!

HOME>>

Unlisted on Aug 31, 2023

Closed on Sep 5, 2023

Related Topics

Topic	Replies	Views	Activity
LS_2227 Failing to achieve Open loop oscillation task-1	3	250	Sep 2023
Lunar Scout: Task 1	3	9.6k	Sep 2023
Questionarie question 2	5	86	Dec 2023
Lunar Scout: Task 2	3	3.8k	Sep 2023

Skip to main content

Topic	Replies	Views	Activity
☑ **Doubt in Task 2A LS_3083** task-2	1	76	Oct 2023