Systems Lab: Systems of ODEs in MATLAB

In this lab, you will write your own ODE system solver for the Heun method (aka the Improved Euler method), and compare its results to those of `ode45`.

You will also learn how to save images in MATLAB.

Opening the m-file lab4.m in the MATLAB editor, step through each part using cell mode to see the results. Compare the output with the PDF, which was generated from this m-file.

There are four (4) exercises in this lab that are to be handed in on the due date of the lab. Write your solutions in a separate file, including appropriate descriptions in each step. Save the m-files and the pdf-file for Exercise 4 and submit them on Quercus.

Student Information

```
Student Name: Robert Purcaru

Student Number: 1007019842
```

Exercise 1

Objective: Write your own ODE system solver using the Heun/Improved Euler Method and compare it to `ode45`.

Details: Consider the system of 2 ODEs:

`x1'=f(t,x1,x2), x2'=g(t,x1,x2)`

This m-file should be a function which accepts as variables (t0,tN,x0,h), where t0 and tN are the start and end points of the interval on which to solve the ODE, h is the stepsize, and x0 is a vector for the initial condition of the system of ODEs `x(t0)=x0`. Name the function solvesystem_<UTORid>.m (Substitute your UTORid for UTORid). You may also want to pass the functions into the ODE the way `ode45` does (check MATLAB labs 2 and 3).

Your m-file should return a row vector of times and a matrix of approximate solution values (the first row has the approximation for `x1` and the second row has the approximation for `x2`).

Note: you will need to use a loop to do this exercise. You will also need to recall the Heun/Improved Euler algorithm learned in lectures.

Exercise 2

Objective: Compare Heun with an exact solution

Details: Consider the system of ODEs

`x1' = x1/2 - 2*x2, x2' = 5*x1 - x2`

with initial condition `x(0)=(1,1)`.

Use your method from Exercise 1 to approximate the solution from `t=0` to `t=4*pi` with step size `h=0.05`.

Compute the exact solution (by hand) and plot both phase portraits on the same figure for comparison.

Your submission should show the construction of the inline function, the use of your Heun's method to obtain the solution, a construction of the exact solution, and a plot showing both. In the comments, include the exact solution.

Label your axes and include a legend.

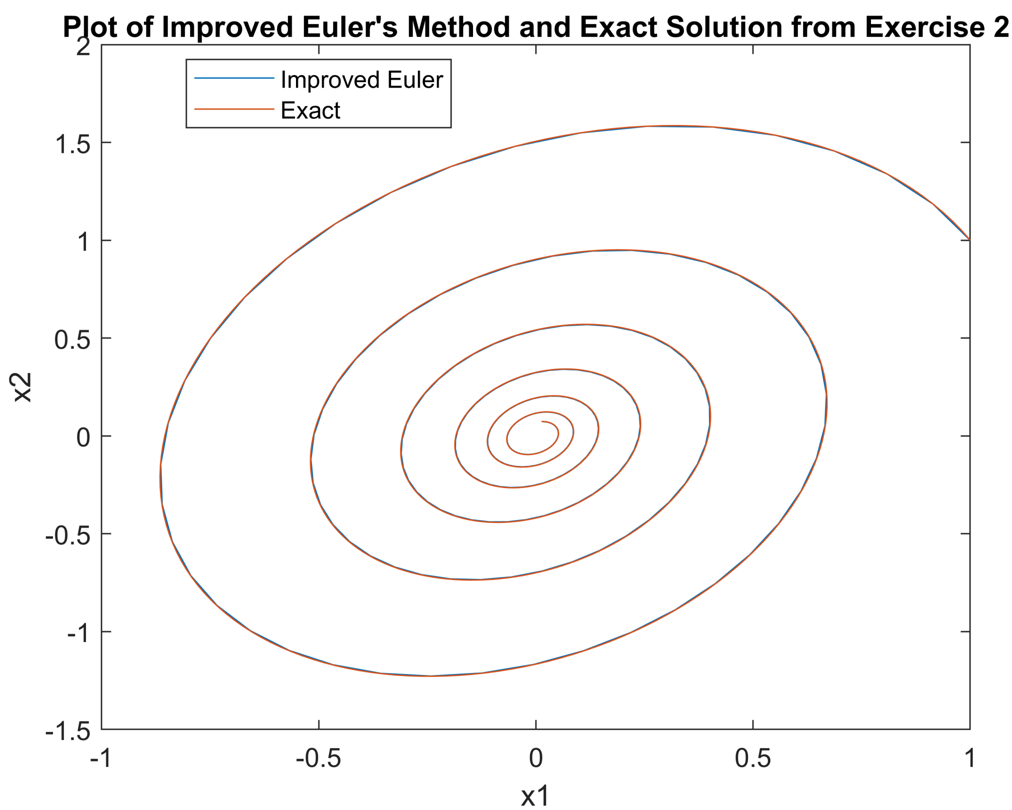
```
f = @(t, x1, x2) x1./2 - 2.*x2;
g = @(t, x1, x2) 5.*x1 - x2;

x0 = 2:1;
x0(1,1) = 1;
x0(2,1) = 1;

[t, y] = solvesystem_(f, g, 0, 4*pi, x0, 0.05);

t_exact = linspace(0, 4*pi, 1000);
x1_exact = exp(-t_exact/4).*(3/20*cos(sqrt(151)*t_exact/4)-(sqrt(151)/20)*sin(sqrt(151)*t_exact
x2_exact = exp(-t_exact/4).*(cos(sqrt(151)*t_exact/4)+(17/sqrt(151))*(sin(sqrt(151)*t_exact/4)

plot(y(1,:),y(2,:), x1_exact, x2_exact);
ylabel('x2');
xlabel('x1');
title("Plot of Improved Euler's Method and Exact Solution from Exercise 2");
legend("Improved Euler","Exact",'Location','Best');
```



Exercise 3

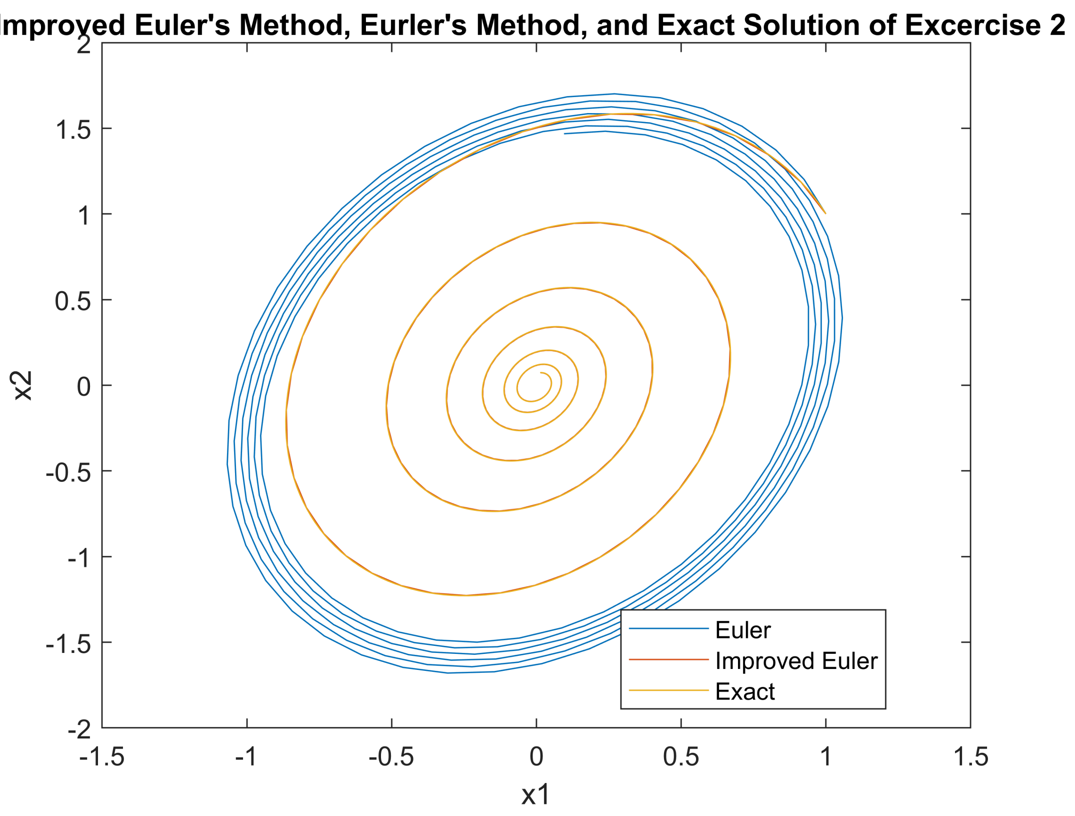
Objective: Compare your method with Euler's Method (from `iode`).

Details: Use `iode` to plot the solution for the same problem with the same step size as on Exercise 2.

```
f_3 = @(t,x) [x(1)./2 - 2.*x(2); 5.*x(1) - x(2)];

t_iode = linspace(0, 4*pi, 4*pi/0.05);
y_iode = euler(f_3, x0, t_iode);

% Plotting and labeling the results
plot(y_iode(1,:),y_iode(2,:),y(1,:),y(2,:),x1_exact,x2_exact);
xlabel('x1');
ylabel('x2');
title("Plot of Improved Euler's Method, Eurler's Method, and Exact Solution of Excercise 2 Syst
legend("Euler", "Improved Euler", 'Exact','Location','Best');
```



Compare your solution on exercise 2, the exact solution from exercise 2 and the approximation using Euler's method. Plot the solution for Euler's method and make note of any differences.

The Euler's method is very far off the exact and the improved Euler's graphs. The Euler's method graph spirals in a lot slower than the other two (still spiraling IN however), completeing 6 revolutions while the other two complete more than 1. This is probably a result of the euler's method being influenced by nearby solutions as the approximation misses the exact solution and following paths that are further out of the spiral than the actual solution.

Saving Images in MATLAB

To do the following exercises, you will need to know how to output graphics from MATLAB. Create a folder on your Desktop (or elsewhere) to contain the files generated by these exercises. Make this folder the "Current Folder" in the left side of the main MATLAB window. This will ensure that the files output by MATLAB end up in the folder you created.

To save an image of a phase portrait, use the following steps:

1. Get the phase portrait looking the way you want in the `iode` window.

2. Leaving `iode` open, switch to the main MATLAB window.

3. Type the command `print -dpng -r300 'filename.png'` in the command window.

This command will create a PNG graphic called `filename.png` in the current folder. The `-dpng` option tells MATLAB to output the graphic in PNG format; MATLAB also allows output in other formats, such as BMP, EPS, PNG and SVG. The `-r300` option tells MATLAB to set the resolution at 300 dots per inch and can be adjusted if you wish.

Exercise 4

Objective: Analyze phase portraits.

Details: Compile the results of the following exercises into a single document (e.g. using a word processor) and export it to `PDF` for submission on Quercus.

For each of the first-order systems of ODEs 4.1 to 4.10 below, do the following exercises:

(a) Generate a phase portrait for the system (centre the graph on the equilibrium point at (0,0)). Include a few trajectories.

(b) Classify the equilibrium on asymptotic stability, and behaviour (sink, source, saddle-point, spiral, center, proper node, improper node) - check table 3.5.1 and figure 3.5.7. Classify also as for clockwise or counterclockwise movement, when relevant.

(c) Compute the eigenvalues of the matrix (you do not need to show your calculations). Using the eigenvalues you computed, justify part (b).

To avoid numerical error, you should use Runge-Kutta solver with a step size of `0.05`. Change the display parameters, if necessary, to best understand the phase portrait.

4.1. `dx/dt = [2 1; 1 3] x`

Asymptotic stability and behaviour: Unstable source

Rotation: N/A

Eigen Values: 1/2 * (5 + sqrt(5)), 1/2 * (5 - sqrt(5))

Justification: eigen values are both real, distinct, and > 0, so the general solution increases away from the equillibrium meaning solutions approach infinity as t approches infinity

4.2. `dx/dt = [-2 -1; -1 -3] x`

Asymptotic stability and behaviour: Stable sink

Rotation: N/A

Eigen Values: 1/2 * (-5 + sqrt(5)), 1/2 * (-5 - sqrt(5))

Justification: eigen values are both real, distinct, and < 0 so the general solution decreases toward the equillibrium meaning solutions approach the equilibrium (0,0) as t approches infinity

4.3. `dx/dt = [-4 -6; 3 5] x`

Asymptotic stability and behaviour: Unstable saddle point

Rotation: N/A

Eigen Values: 2, -1

Justification: Both eigen values are real and distinct. One is < 0 and the other > 0. This means that one eigen vector points toward the origin and the other away. Solutions follow the former toward the origin before following the latter away from the orign, toward infinity as t goes to infinity.

4.4. `dx/dt = [4 6; -3 -5] x`

Asymptotic stability and behaviour: Unstable saddle point

Rotation: N/A

Eigen Values: 1, -2

Justification: Both eigen values are real and distinct. One is < 0 and the other > 0. This means that one eigen vector points toward the origin and the other away. Solutions follow the former toward the origin before following the latter away from the orign, toward infinity as t goes to infinity. Difference between this one and 4.3 is the eigen vector that is followed toward the equillibrium then towards infinity

4.5. `dx/dt = [0 -1; 1 -1] x`

Asymptotic stability and behaviour: Stable inward spiral

Rotation: CCW

Eigen Values: -1/2(1 + i*sqrt(3)), -1/2 (1 - i*sqrt(3))

Justification: Both eigen values are complex with a negative real component, meaning that they spiral toward the equillibrium.

4.6. `dx/dt = [0 1; -1 1] x`

Asymptotic stability and behaviour: Unstable outward spiral

Rotation: CW

Eigen Values: 1/2(1 + i*sqrt(3)), 1/2 (1 - i*sqrt(3))

Justification: Both eigen values are complex with a poistive real component, meaning that they spiral away from the equllibrium in concentric spirals

4.7. `dx/dt = [2 8; -1 -2] x`

Asymptotic stability and behaviour: stable center

Rotation: CW

Eigen Values: 2i, -2i

Justification: Both eigen values are complex with a 0 real component, meaning that they neither approach nor receed from the origin. The direction of rotation is obtained by substituting values and checking.

4.8. `dx/dt = [-2 -8; 1 2] x`

Asymptotic stability and behaviour: Stable center

Rotation: CCW

Eigen Values: 2i, -2i

Justification: Both eigen values are complex with a 0 real component, meaning that they neither approach nor receed from the origin. The direction of rotation is obtained by substituting values and checking.

4.9. `dx/dt = [-8 5; -13 8] x`

Asymptotic stability and behaviour: Stable center

Rotation: CW

Eigen Values: i, -i

Justification: Both eigen values are complex with a 0 real component, meaning that they neither approach nor receed from the origin. The direction of rotation is obtained by substituting values and checking.

4.10. `dx/dt = [8 -5; 13 -8] x`

Asymptotic stability and behaviour: (sink, source, saddle-point, spiral, center, proper node, improper node)

Rotation: CCW

Eigen Values: i, -i

Justification: Both eigen values are complex with a 0 real component, meaning that they neither approach nor receed from the origin. The direction of rotation is obtained by substituting values and checking.