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: Mustafa Khan

Student Number: 1006717037
```

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.

Answer: Comparison

While both Improved Euler Method (IEM) and ode45 take in a user generated function and use current and future slope values to make a step, IEM uses fixed step sizes while ode45 uses adaptive step sizes. Additionally, ode45 makes each step such that it is within a certain error tolerance while IEM doesn't do this.

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.

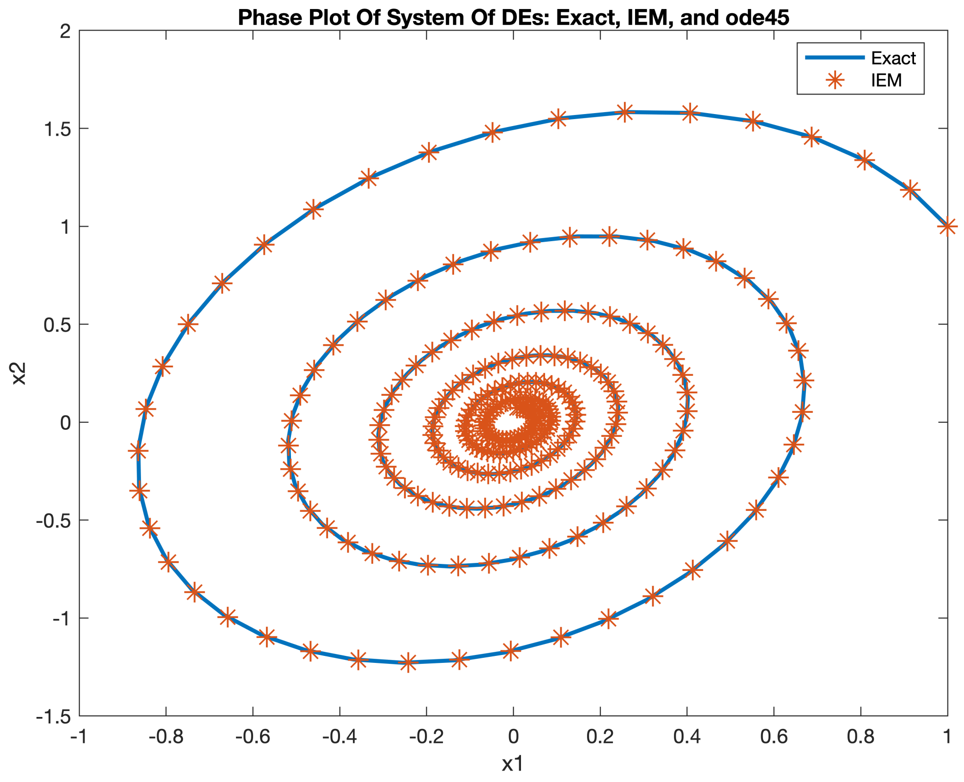
```
%Inline functions
f = @(t,x1,x2) x1/2 - 2*x2;
g = @(t,x1,x2) 5*x1 - x2;

t0 = 0;
tN = 4*pi;
h = 0.05;
x0 = [1, 1];

%IEM solver
[t, soln] = solvesystem_khanm382(f, g, t0, tN, x0, h);

%Exact solutions
f1exact = @(t) ((1./151).* exp(-t./4) .* ...
    (3.*sqrt(151).*sin(sqrt(151).*(t./4)) + ...
    151.*cos(sqrt(151).*(t./4)))) - ...
    ((8.*exp(-t./4).*sin(sqrt(151).*(t./4)))./sqrt(151));
f2exact = @(t) ((1./151).* exp(-t./4) .* ...
    (-3.*sqrt(151).*sin(sqrt(151).*(t./4)) + ...
    151.*cos(sqrt(151).*(t./4)))) + ...
    ((20.*exp(-t./4).*sin(sqrt(151).*(t./4)))./sqrt(151));

%Plot Of Systems Of DEs Using IEM
plot(f1exact(t), f2exact(t), soln(1, :), soln(2, :), '*', ...
    'MarkerSize',10, 'LineWidth', 2)
xlabel('x1');
ylabel('x2');
title('Phase Plot Of System Of DEs: Exact, IEM, and ode45');
legend('Exact', 'IEM', 'Location','Best');
```



Exercise 3

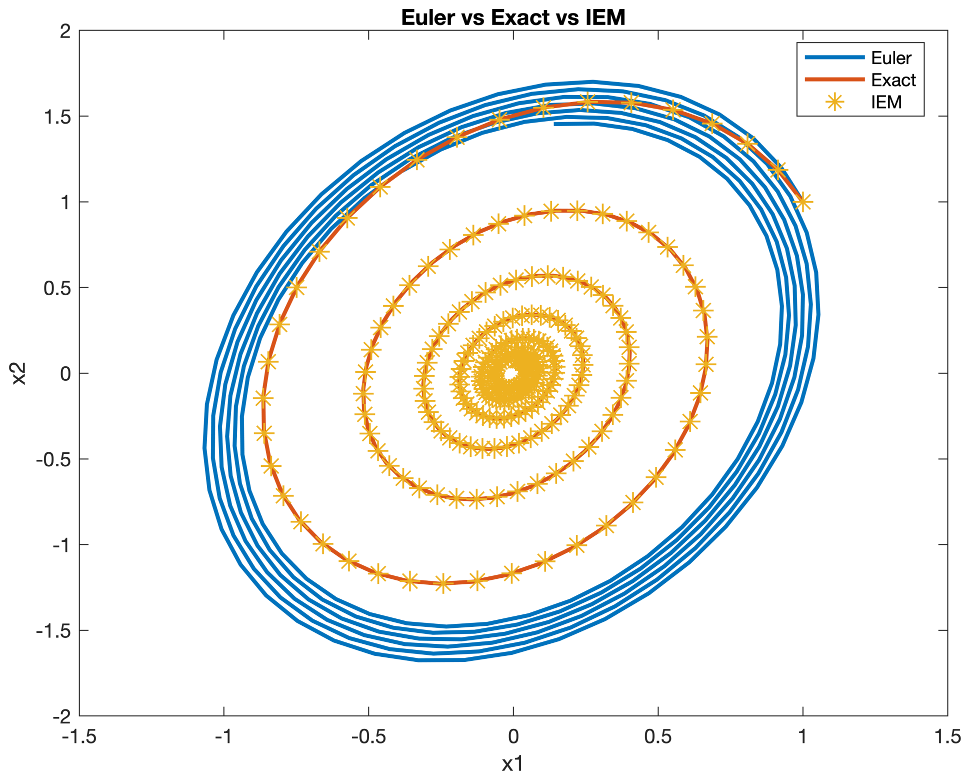
Objective: Compare your method with Euler's Method (from `iode`).

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

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.

```
j = @(t,x)[x(1)./2 - 2.*x(2); 5.*x(1) - x(2)];
x0 = [1;1];
h=0.05;
xc = euler(j, x0, t0:h:tN);

% Plot of Exact vs Euler (from iode) vs IEM
plot(xc(1, :), xc(2, :), f1exact(t), f2exact(t), ...
    soln(1, :), soln(2, :), '*', 'MarkerSize',10, 'LineWidth', 2)
xlabel('x1');
ylabel('x2');
title('Euler vs Exact vs IEM');
legend('Euler', 'Exact', 'IEM', 'Location','Best');
```



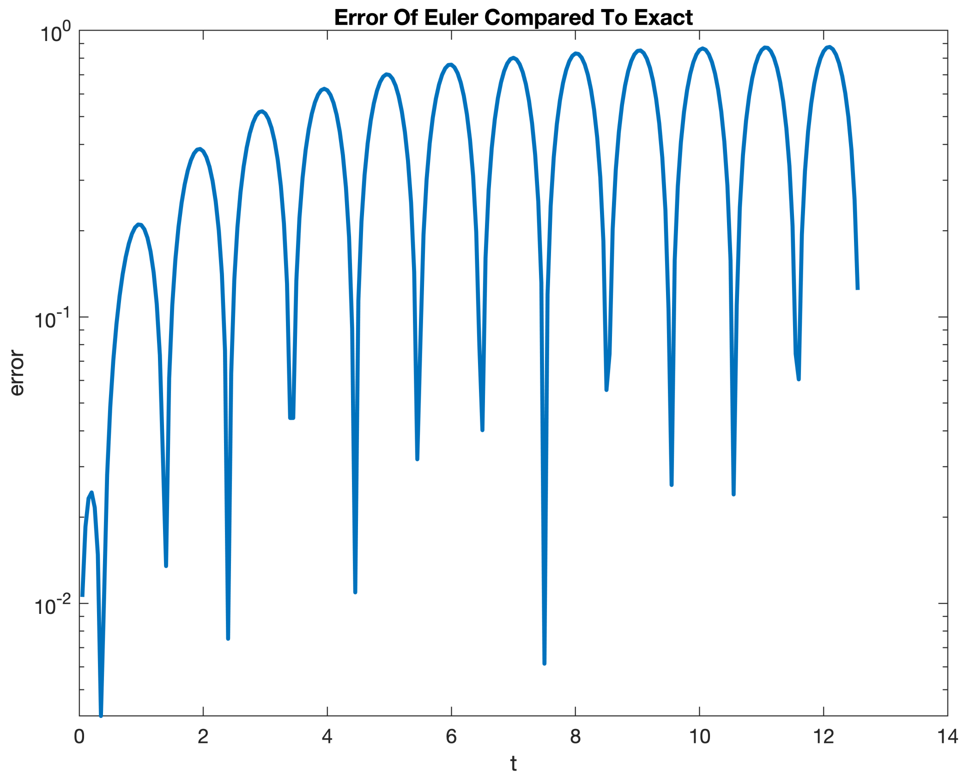
```
% Error for x1 and x2 numerical solutions when using Euler method
err_x1_euler = abs(f1exact(t) - xc(1, :));
err_x2_euler = abs(f2exact(t) - xc(2, :));
fprintf('maximum error of x1: %g \n', max(err_x1_euler));
```

```
maximum error of x1: 0.875648
```

```
fprintf('maximum error of x2: %g \n', max(err_x2_euler));
```

```
maximum error of x2: 1.38393
```

```
%Plot of the error of euler compared to exact
semilogy(t, err_x1_euler, 'LineWidth', 2);
xlabel('t');
title('Error Of Euler Compared To Exact');
ylabel('error');
```



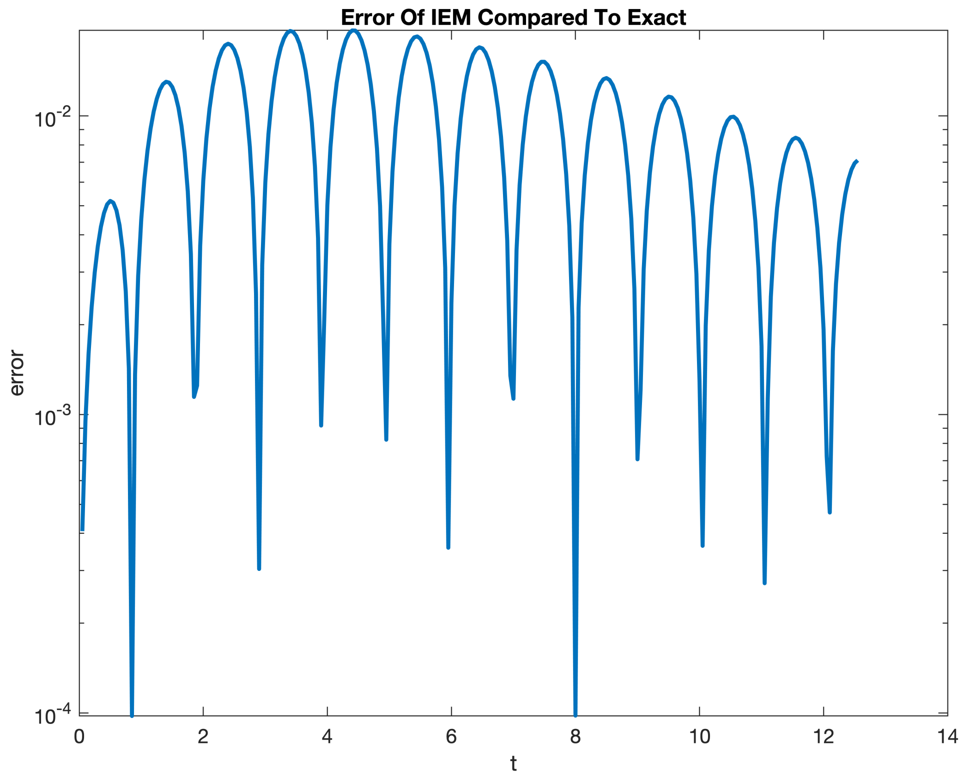
```
% Error for x1 and x2 numerical solutions when using IEM
err_x1 = abs(f1exact(t) - soln(1, :));
err_x2 = abs(f2exact(t) - soln(2, :));
fprintf('maximum error of x1: %g \n', max(err_x1));
```

```
maximum error of x1: 0.0193359
```

```
fprintf('maximum error of x2: %g \n', max(err_x2));
```

```
maximum error of x2: 0.0307814
```

```
%Plot of the error of IEM compared to exact
semilogy(t, err_x1, 'LineWidth', 2);
xlabel('t');
title('Error Of IEM Compared To Exact');
ylabel('error');
```



```
%Comparison

%As can be seen on the plot of IEM vs Euler vs Exact, there
% is a large discrepancy in the accuracy of the solutions
% when compared to the actual solution. Visually, this is
% seen by Euler method diverging from the exact solution.
% This is because the exact solution changes its slope much
% quicker than Euler method can adapt to. On the other hand,
% IEM takes into account the current and next slope and
% averages the value, and in this case this is sufficient to
%approximate the exact solution much better.

%Additionally, I plotted out the error of both methods (Euler
% and IEM) and the maximum error of x1 and x2 for Euler's
% method is around the magnitude 10^0 and 10^-1 while the
% maximum error of x1 and x2 for IEM is around the order of
% magnitude of 10^-2. Clearly, IEM has a lower error
% than Euler's method.
```

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`

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

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

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

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

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

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

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

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

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

Answer: `An attached PDF shows my work and findings for this question.`