Second-Order Lab: Second-Order Linear DEs in MATLAB

Table of Contents

In this lab, you will learn how to use `iode` to plot solutions of second-order ODEs. You will also learn to classify the behaviour of different types of solutions.

Moreover, you will write your own Second-Order ODE system solver, and compare its results to those of `iode`.

Opening the m-file lab5.m in the MATLAB editor, step through each part using cell mode to see the results.

There are seven (7) exercises in this lab that are to be handed in on the due date of the lab.

Student Information

```
Student Name: Fatema Rahman Farzin
```

```
Student Number: 1005423356
```

Iode for Second-Order Linear DEs with constant coefficients

In the `iode` menu, select the `Second order linear ODEs` module. It opens with a default DE and a default forcing function `f(t) = cos(2t)`. The forcing function can be plotted along with the solution by choosing `Show forcing function` from the `Options` menu.

Use this module to easily plot solutions to these kind of equations.

There are three methods to input the initial conditions:

Method 1. Enter the values for `t0`, `x(t0)`, and `x'(t0)` into the `Initial conditions` boxes, and then click `Plot solution`.

Method 2. Enter the desired slope `x'(t0)` into the appropriate into the `Initial conditions` box, and then click on the graph at the point `(t0,x(t0))` where you want the solution to start.

Method 3. Press down the left mouse button at the desired point `(t0,x(t0))` and drag the mouse a short distance at the desired slope `x'(t0)`. When you release the mouse button, `iode` will plot the solution.

Growth and Decay Concepts

We want to classify different kinds of behaviour of the solutions. We say that a solution:

`grows` if its magnitude tends to `infinity` for large values of `t`, that is, if either the solution tends to $+\infty$ or $-\infty$,

`decays` if its magnitude converges to `0` for large values of `t`,

`decays while oscillating` if it keeps changing sign for large values of `t` and the amplitude of the oscillation tends to zero,

`grows while oscillating` if it keeps changing sign for large values of `t` and the amplitude of the oscillation tends to infinity.

Exercise 1

Objective: Use `iode` to solve second-order linear DEs. And classify them.

Details: Consider the ODE:

```
4y'' + 4 y' + 17 y = 0
```

(a) Use `iode` to plot six (6) numerical solutions of this equation with "random" initial data (use Method 3 above) and press-and-drag at various initial points, with some of the slopes being positive and some negative)

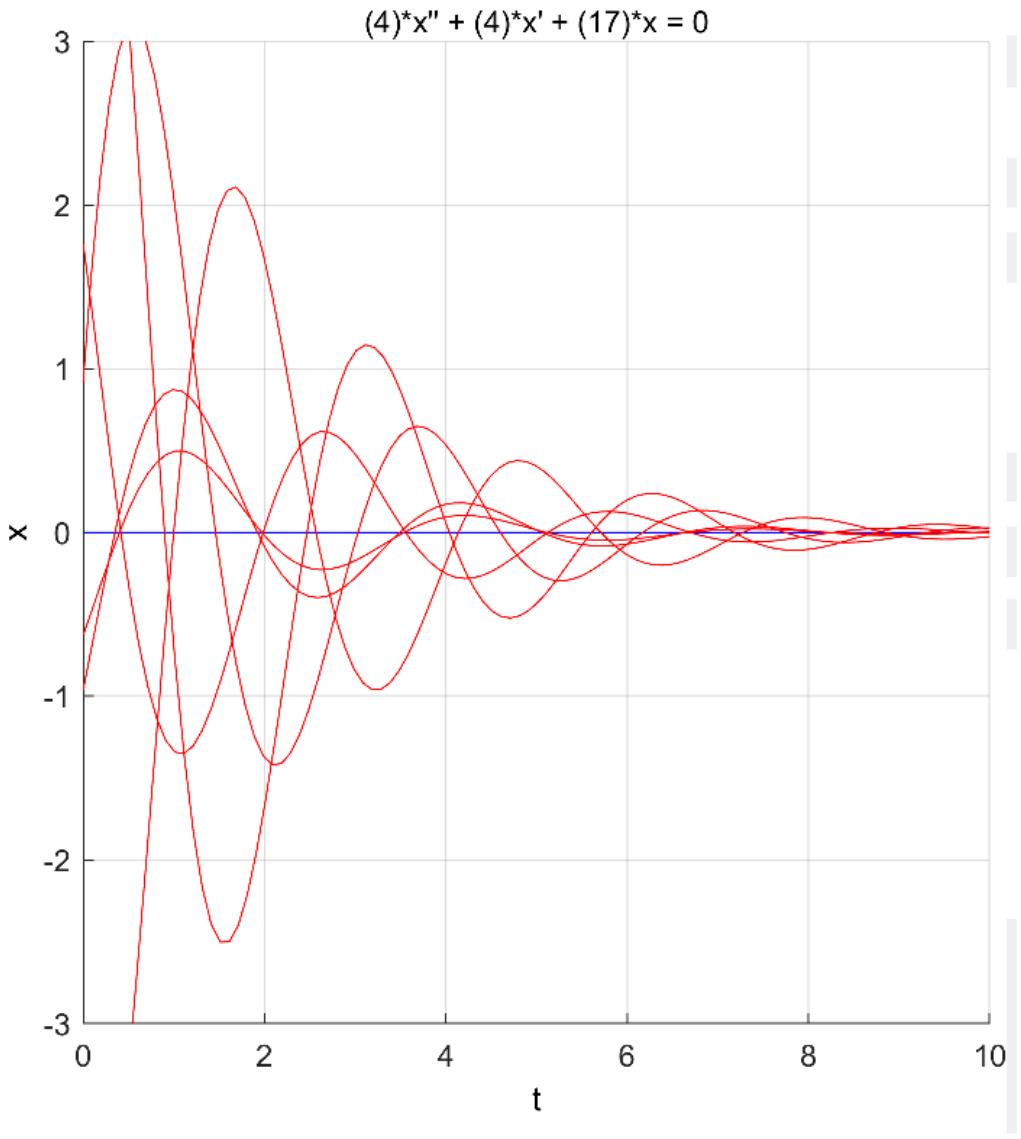
Use only initial points in the part of the window where `0<t<1` and `-1<x<1` and take all initial slopes between `-3` and `+3`.

Change the window to `[0,10]x[-3,3]`. Attach a cropped screenshot to your answers file.

(b) Based on the results of (a), state what percentage of solutions decay, grow, grow while oscillating, or decay while oscillating.

(c) Solve the DE and write the exact solution. Explain why this justifies your answer in (b).

```
% (a) screenshot attached
img = imread('ex1.png');
imshow(img);

% (b) 100% of solutions decay while oscilating
% (c) Exact solution: y(t) = e^(-t/2) * (c1*cos(2t) + c2*sin(2t))
      % 1. The exact solution has an e^(-t/2) term which goes to 0 as t
      % approaches infinity, this explains why the IODE plots decay with time.
      % 2. Additionally, the sin and cos terms justify the oscillation as the
 function decays
```



Exercise 2

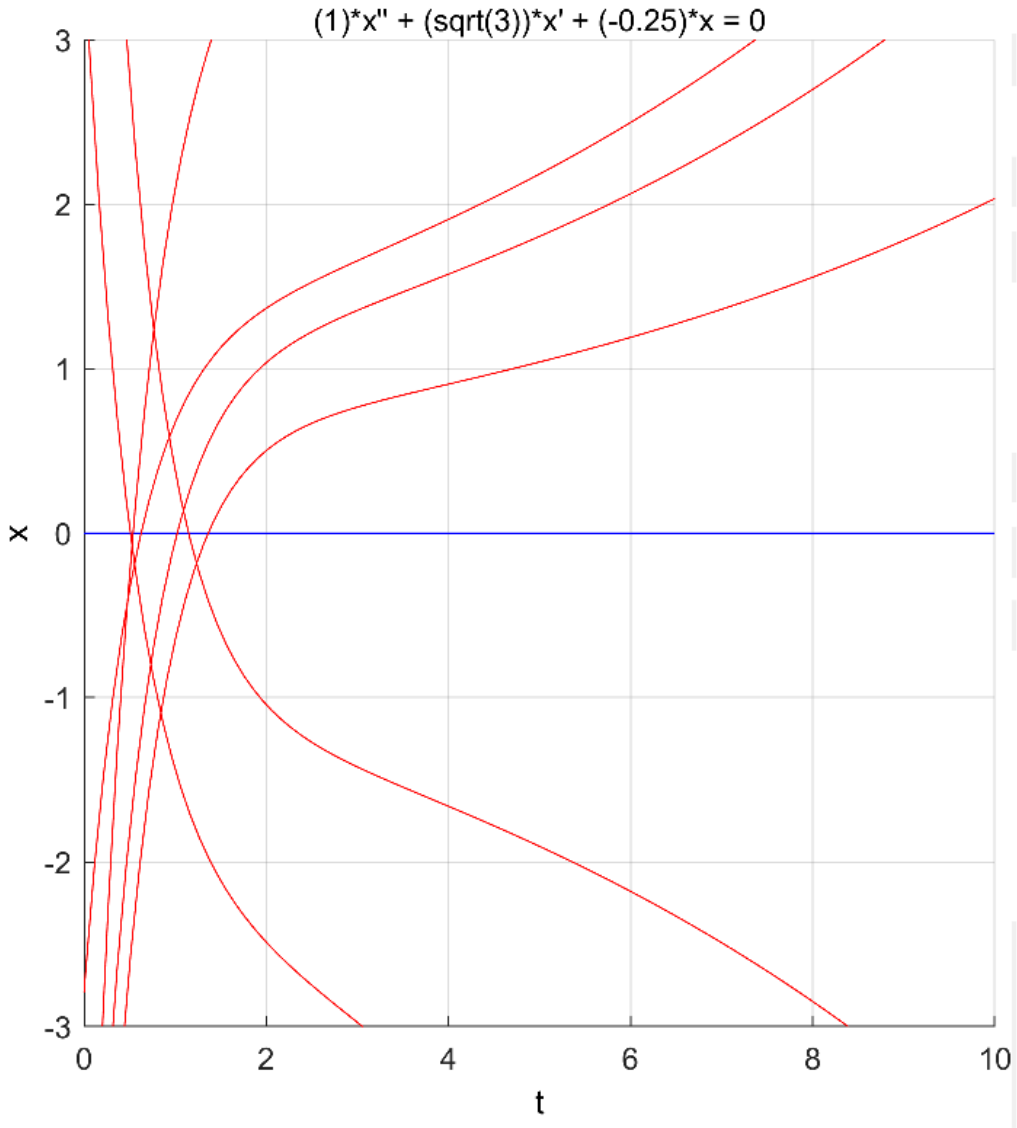
Consider the ODE:

```
y'' + sqrt(3) y' - y/4 = 0
```

Repeat (a), (b), (c) from Exercise 1 with this DE.

```
% (a) screenshot attached
img = imread('ex2.png');
imshow(img);
```

```
% (b) 100% of solutions grow
% (c) Exact solution: y(t) = c1*e^((2-sqrt(3))*t/2) + c2*e^((-2-sqrt(3))*t/2)
     % 1. In the exact solution, as t approaches infinity, the second term
 (negative exponent) always goes to 0
     % 2. However, the first term with a positive exponent always goes to
 infinity
     % 3. If c1 is nonzero, the solution grows; only in the case when c1 is
 zero, the solution decays
     % 4. Since the probability of c1 being 0 is 0, 100% of solutions grow
```



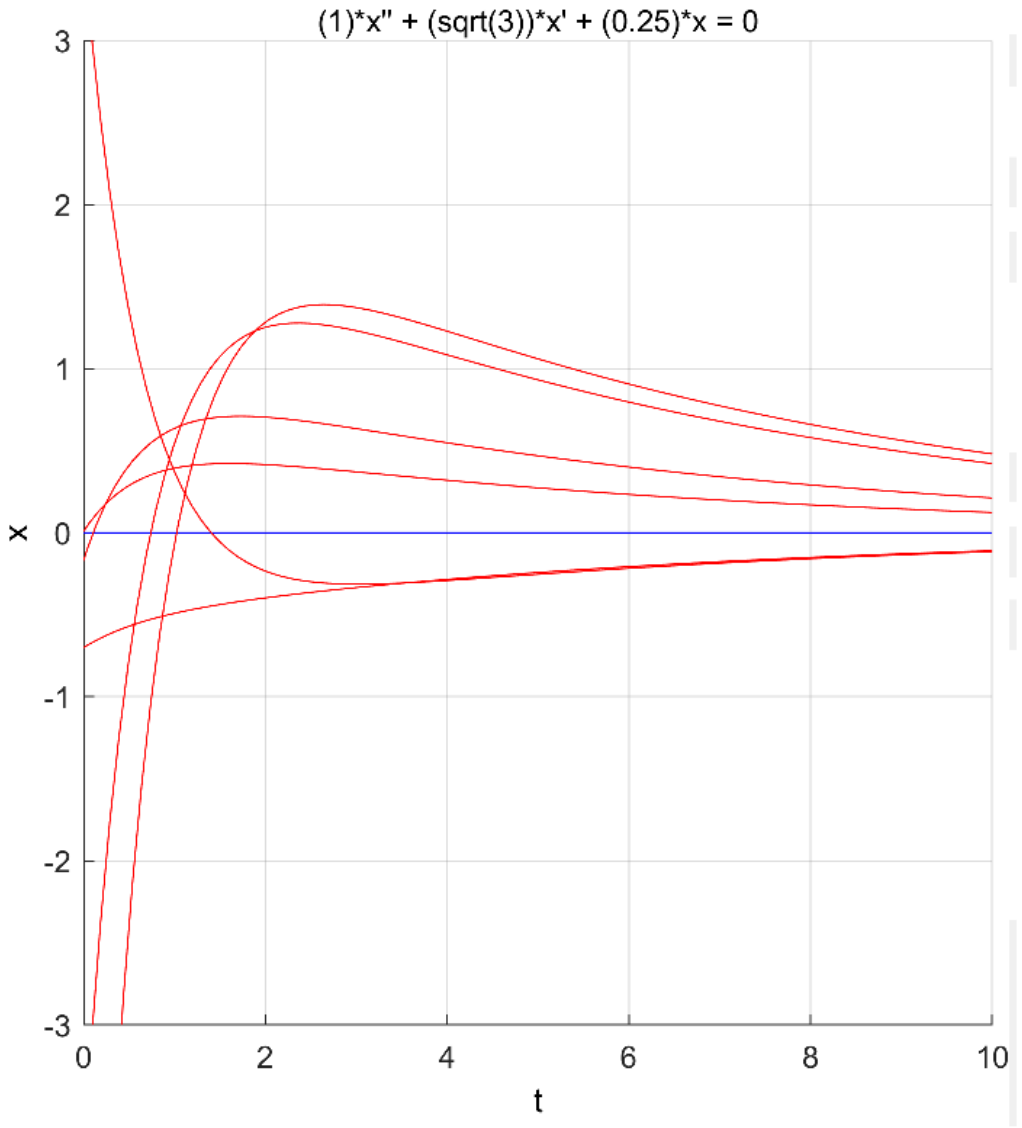
Exercise 3

Consider the ODE:

```
y'' + sqrt(3) y' + y/4 = 0
```

Repeat (a), (b), (c) from Exercise 1 with this DE.

```
% (a) screenshot attached
img = imread('ex3.png');
imshow(img);

% (b) 100% of solutions decay
% (c) Exact solution: y(t) = c1*e^((sqrt(2)-sqrt(3))*t/2) + c2*e^((-sqrt(2)-
sqrt(3))*t/2)
      % 1. Both terms in the exact solution has a negative exponent
      % 2. Since both roots of the characteristic equation is negative, the
 solution approaches 0 as t approaches infinity
      % 3. This explains why all the IODE plots decay with time
```



Exercise 4

Consider the fourth-order ODE:

```
y'''' + 2 y''' + 6 y'' + 2 y' + 5 y = 0
```

(a) Find the general solution for this problem. You can use MATLAB to find the roots of the characteristic equation numerically with `roots`.

(b) Predict what percentage of solutions with random initial data will grow, decay, grow while oscillating, and decay while oscillating. Explain.

```
% (a) General solution of the ODE:
```

```
% y(t) = e^(-x)(c1sin(2x) + c2cos(2x)) + c3sinx + c4cosx

% (b) 100% of solutions will decay while oscillating. A longer explanation is
 provided below:

% The two different behaviours of the solution:
    % 1. If both c1 and c2 are 0, the solution will oscillate indefinitely
 without growing or decaying
    % 2. If either c1 or c2 is nonzero -> the solution will decay while
 oscillating
% And since the probability of both c1 and c2 being zero is 0, the solution
 will decay while oscillating
```

Exercise 5

Objective: Classify equations given the roots of the characteristic equation.

Details: Your answer can consist of just a short sentence, as `grows` or `decays while oscillating`.

Consider a second-order linear constant coefficient homogeneous DE with `r1` and `r2` as roots of the characteristic equation.

Summarize your conclusions about the behaviour of solutions for randomly chosen initial data when.

(a) `0 < r1 < r2`

```
% 100% of solutions will grow because:
    % both roots are positive and distinct
```

(b) `r1 < 0 < r2`

```
% 100% of the solutions will grow because:
    % one root is negative and one root is positive
    % Let c1 be the coefficient of the term with the positive root
    % Only in the case when c1 = 0, the solution will decay
    % In all other cases, the solutions ultimately go to infinity, i.e., grows
    % Since the probability of c1 being 0 is 0, all solutions will grow
```

(c) `r1 < r2 < 0`

```
% 100% of solutions will decay because:
    % both roots are negative and distinct
```

(d) `r1 = alpha + beta i` and `r2 = alpha - beta i` and `alpha < 0`

```
% 100% of solutions will decay while oscillating because:
    % both roots are complex numbers with a negative real part
```

(e) `r1 = alpha + beta i` and `r2 = alpha - beta i` and `alpha = 0`

```
% 100% of solutions will oscillate without growing or decaying because:
    % both roots are complex numbers with 0 real part
```

(f) `r1 = alpha + beta i` and `r2 = alpha - beta i` and `alpha > 0`

```
% 100% of solutions will grow while oscillating because:
    % both roots are complex numbers with positive real part
```

Numerical Methods for Second-Order ODEs

One way to create a numerical method for second-order ODEs is to approximate derivatives with finite differences in the same way of the Euler method.

This means that we approximate the first derivative by:

```
y'(t[n]) ~ ( y[n] - y[n-1] ) / h
```

and

```
y''(t[n])  ~ ( y'(t[n+1]) - y'(t[n]) ) / h ~ ( y[n+1] - 2 y[n] + y[n-1]
) / ( h^2 )
```

By writing these approximations into the ODE, we obtain a method to get `y[n+1]` from the previous two steps `y[n]` and `y[n-1]`.

The method for approximating solutions is:

1. Start with `y[0]=y0`

2. Then we need to get `y[1]`, but we can't use the method, because we don't have two iterations `y[0]` and `y[-1]`(!!). So we use Euler to get

```
y[1] = y0 + y1 h
```

y1 is the slope given by the initial condition

3. Use the method described above to get `y[n]` for `n=2,3,...`.

Exercise 6

Objective: Write your own second-order ODE solver.

Details: Consider the second-order ODE

```
y'' + p(t) y' + q(t) y = g(t)
```

Write a second-order ODE solver using the method described above.

This m-file should be a function which accepts as variables (t0,tN,y0,y1,h), where t0 and tN are the start and end points of the interval on which to solve the ODE, y0, y1 are the initial conditions of the ODE, and h is the stepsize. You may also want to pass the functions into the ODE the way `ode45` does (check MATLAB lab 2). Name the function DE2_<UTORid>.m.

Note: you will need to use a loop to do this exercise.

```
% separate DE2_farzinf2.m file attached with the ODE solver function
```

Exercise 7

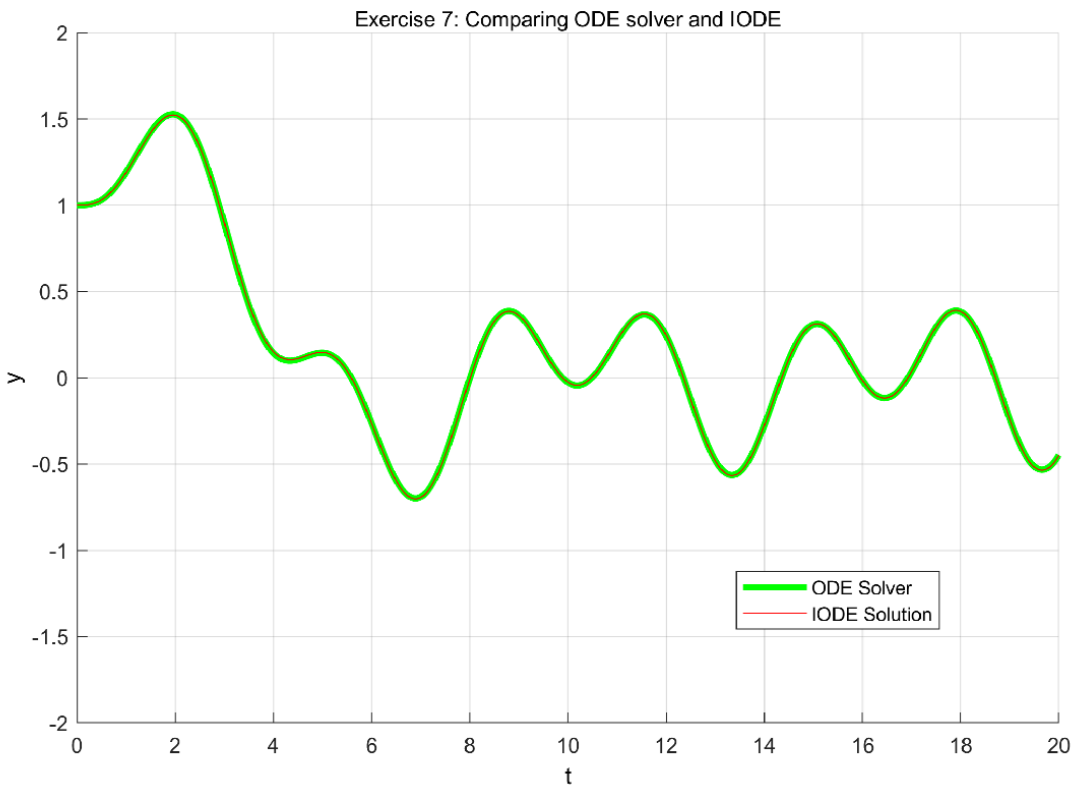
Objective: Compare your method with `iode`

Details: Use `iode` to plot the solution of the ODE `y'' + exp(-t/5) y' + (1-exp(-t/5)) y = sin(2*t)|with the initial conditions |y(0) = 1, y'(0) = 0`

Use the window to `[0,20]x[-2,2]` Without removing the figure window, plot your solution (in a different colour), which will be plotted in the same graph.

Comment on any major differences, or the lack thereof.

```
% defining the inputs to the ODE solver for the given ODE
y0 = 1;                         % initial conditions
y1 = 0;
t0 = 0;
tN = 20;

p = @(t) exp(-t./5);            % functions for the variable coefficients
q = @(t) (1-exp(-t./5));
g = @(t) sin(2.*t);

h = 0.05;                       % step size

% using the DE solver function to obtain solution
[tsol, ysol] = DE2_farzinf2(t0, tN, y0, y1, h, p, q, g);

% screenshot from the IODE solver with both the DE function solution and the
% IODE solution:
img = imread('ex7plot.png');
imshow(img);

% There are no major differences between the two solutions
```



Published with MATLAB® R2023a