MAT 275 Laboratory 4
MATLAB solvers for First-Order IVP

In this laboratory session we will learn how to

1. Use MATLAB solvers for solving scalar IVP
2. Use MATLAB solvers for solving higher order ODEs and systems of ODEs.

First-Order Scalar IVP

Consider the IVP

$$\begin{cases} y' = t - y, \\ y(0) = 1. \end{cases} \tag{1}$$

The exact solution is $y(t) = t - 1 + 2e^{-t}$. A numerical solution can be obtained using various MATLAB solvers. The standard MATLAB ODE solver is `ode45`. The function `ode45` implements 4/5th order Runge-Kutta method. Type `help ode45` to learn more about it.

Basic `ode45` Usage

The basic usage of ode45 requires a function (the right-hand side of the ODE), a time interval on which to solve the IVP, and an initial condition. To plot numerical solution of the above IVP using ode45, on interval, say, [0,3], we can run the following code snippet:

```
1  f = @(t,y) t - y;
2  [t,y] = ode45(f,[0,3],1);
3  plot(t,y)
```

- Line 1 defines the function `f` as a function of t and y, i.e., $f(t, y) = t - y$. This is the right-hand side of the ODE (1).
- Line 2 solves the IVP numerically using the `ode45` solver. The first argument is the function `f`, the second one determines the time interval on which to solve the IVP in the form [initial time, final time], and the last one specifies the initial value of y. The output of `ode45` consists of two arrays: an array `t` of discrete times at which the solution has been approximated, and an array `y` with the corresponding values of y. These values can be listed in the Command Window as

```
>> [t,y]
ans =
         0    1.0000
    0.0502    0.9522
    0.1005    0.9093
    0.1507    0.8709
    0.2010    0.8369
    ........
    2.9010    2.0109
    2.9257    2.0330
    2.9505    2.0551
    2.9752    2.0773
    3.0000    2.0996
```

Since the output is quite long we printed only some selected values.
For example the approximate solution at $t \approx 2.9257$ is $y \approx 2.0330$. Unless specific values of y are needed it is better in practice to simply plot the solution to get a sense of the behavior of the solution.

- Line 3 thus plots y as a function of t in a figure window. The plot is shown in Figure 1.

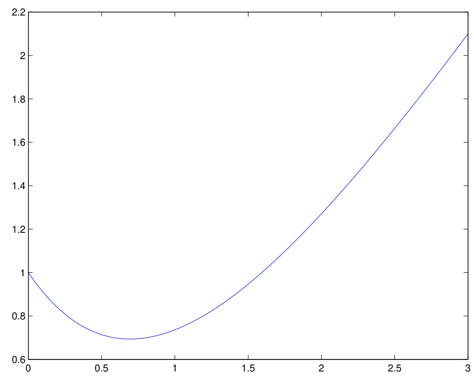

Figure 1: Solution of (1).

Error Plot, Improving the Accuracy

Error plots are commonly used to show the accuracy in the numerical solution. Here the error is the difference between the exact solution $y(t) = t - 1 + 2e^{-t}$ and the numerical approximation obtained from `ode45`. Since this approximation is only given at specified time values (contained in the array `t`) we only evaluate this error at these values of t:

```
err = t-1+2*exp(-t)-y
err =
  1.0e-005 *
        0
    0.0278
    0.0407
    0.0162
   -0.0042
    ......
   -0.0329
   -0.0321
   -0.0313
   -0.0305
   -0.0298
```

(in practice the exact solution is unknown and this error is estimated, for example by comparing the solutions obtained by different methods). Again, since the error vector is quite long we printed only a few selected values. Note the `1.0e-005` at the top of the error column. This means that each component of the vector `err` is less than 10^{-5} in absolute value.
A plot of `err` versus `t` is more revealing. To do this note that errors are usually small so it is best to use a logarithmic scale in the direction corresponding to `err` in the plot. To avoid problems with negative numbers we plot the absolute value of the error (values equal to 0, e.g. at the initial time, are not plotted):

```
semilogy(t,abs(err)); grid on;
```

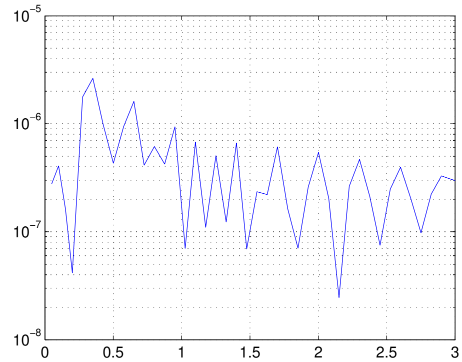

Figure 2: Error in the solution of (1) computed by `ode45`.

See Figure 2. Note that the error level is about 10^{-6}. It is sometimes important to reset the default accuracy `ode45` uses to determine the approximation. To do this use the MATLAB `odeset` command prior to calling `ode45`, and include the result in the list of arguments of `ode45`:

```
f = @(t,y) t - y;
options = odeset('RelTol',1e-10,'AbsTol',1e-10);
[t,y] = ode45(f,[0,3],1,options);
err = t-1+2*exp(-t)-y;
semilogy(t,abs(err))
```

See Figure 3.

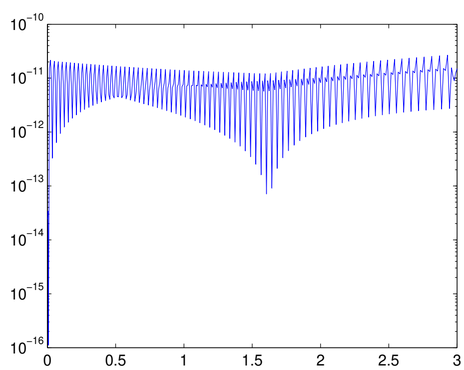

Figure 3: Error in the solution of (1) computed by `ode45` with a better accuracy.

Parameter-Dependent ODE

When the function defining the ODE is complicated, it is convenient either to define it in a code snippet with `function` command and include it in the same file as the calling sequence of ode45 (as below), or to save it in a separate `m`-file.
In this Section we will look at an example where the ODE depends on a parameter.
Consider the IVP

$$\begin{cases} y' = -a\left(y - e^{-t}\right) - e^{-t}, \\ y(0) = 1. \end{cases} \tag{2}$$

with exact solution $y(t) = e^{-t}$ (independent of the parameter a!). An implementation of the MATLAB solution in the interval $[0, 3]$ follows.

```
1  function ex_with_param
2     t0 = 0; tf = 3; y0 = 1;
3     a = 1;
4     [t,y] = ode45(@f,[t0,tf],y0,[],a);
5     disp(['y(' num2str(t(end)) ') = ' num2str(y(end))])
6     disp(['length of y = ' num2str(length(y))])
7  end
8     %-----------------------------------------------------
9  function dydt = f(t,y,a)
10    dydt = -a*(y-exp(-t))-exp(-t);
11 end
```

- Line 1 must start with `function`, since the file contains at least two functions (a driver + a function).
- Line 2 sets the initial data and the final time.
- Line 3 sets a particular value for the parameter a.
- In line 4 the parameter is passed to `ode45` as the 5^{th} argument (the 4^{th} argument is reserved for setting options such as the accuracy using `odeset`, see page 3, and the placeholder `[]` must be used if default options are used).
  Correspondingly the function f defined in lines 8-9 must include a 3^{rd} argument corresponding to the value of the parameter. Note the `@f` in the argument of `ode45`. See the help on `ode45` for more information.
- On line 5 the value of $y(3)$ computed by `ode45` is then displayed in a somewhat fancier form than the one obtained by simply entering `y(end)`. The command `num2string` converts a number to a string so that it can be displayed by the `disp` command.
  The `m`-file `ex_with_param.m` is executed by entering `ex_with_param` at the MATLAB prompt. The output is

```
>> ex_with_param
y(3) = 0.049787
length of y = 45
```

- The additional line 6 in the file lists the length of the array `y` computed by `ode45`. It is interesting to check the size of `y` obtained for larger values of a. For example for $a = 1000$ we obtain

```
>> ex_with_param
y(3) = 0.049792
length of y = 3621
```

  This means that `ode45` needed to take smaller step sizes to cover the same time interval compared to the case $a = 1$, even though the exact solution is the same!

Not all problems with a common solution are the same! Some are easier to solve than others.

When a is large the ODE in (2) is said to be *stiff*. Stiffness has to do with how fast nearby solutions approach the solution of (2), see Figure 4.

$\star$ Other MATLAB ODE solvers are designed to better handle stiff problems. For example, replace `ode45` with `ode15s` in line 4 of `ex_with_param.m` (without changing anything else) and set $a = 1000$:

```
[t,y] = ode15s(@f,[t0,tf],y0,[],a);
```

```
>> ex_with_param
y(3) = 0.049787
length of y = 18
```

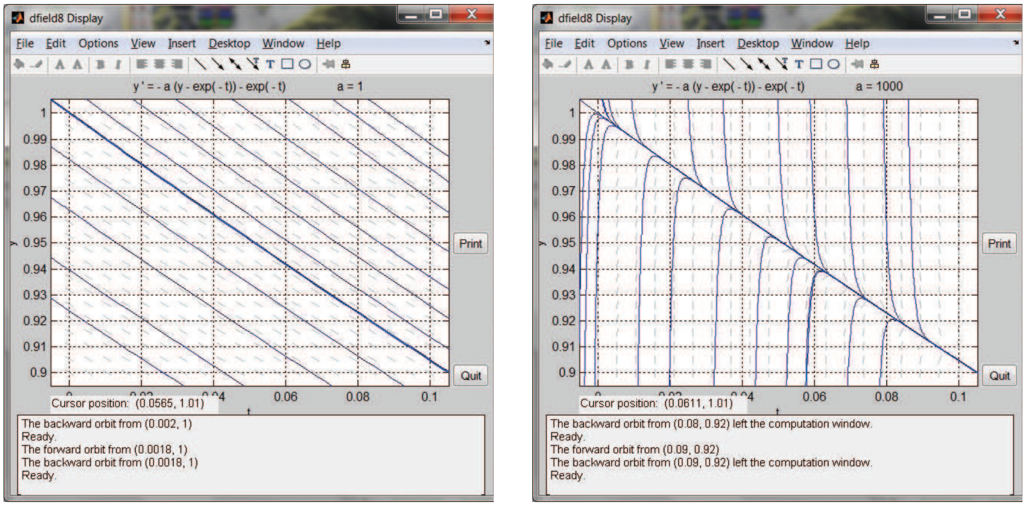

Figure 4: Direction field and sample solutions in the t-y window $[0, 0.1] \times [0.9, 1]$ as obtained using DFIELD8: $a = 1$ (left) and $a = 1000$ (right).

A solution exhibiting blow up in finite time

Consider the Differential Equation

$$\frac{dy}{dt} = 1 + y^2, \quad y(0) = 0$$

The exact solution of this IVP is $y = \tan t$ and the domain of validity is $[0, \frac{\pi}{2})$. Let's see what happens when we try to implement this IVP using `ode45` in the interval $[0, 3]$.

```
>> f = @(t,y) 1+y^2;
>> [t,y]=ode45(f,[0,3],0);
```

```
Warning:  Failure at t=1.570781e+000.  Unable to meet
integration tolerances without reducing the step size below
the smallest value allowed (3.552714e-015) at time t.
> In ode45 at 371
```

The MATLAB ode solver gives a warning message when the value of $t = 1.570781$ is reached. This is extremely close to the value of $\pi/2$ where the vertical asymptote is located.
If we enter `plot(t,y)` we obtain Figure 5 on the left (note the scale on the y-axis), however, if we use `xlim([0,1.5])`, we can recognize the graph of $y = \tan t$.

Higher-Order and Systems of IVPs

We show here how to extend the use of `ode45` to systems of first-order ODEs (the same holds for other solvers such as `ode15s`). Higher-order ODEs can first be transformed into a system of first-order ODEs to fit into this framework. We will see later how to do this.

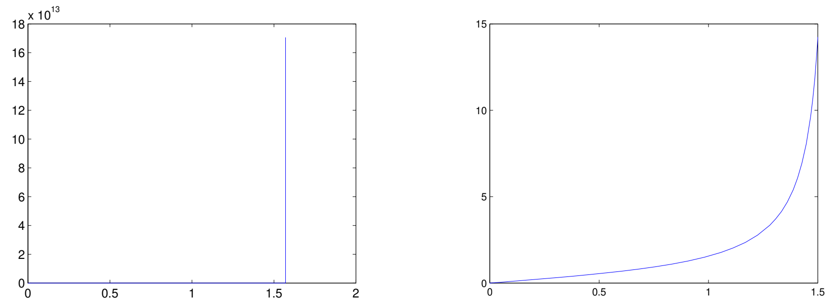

Figure 5: Solution of $y' = 1 + y^2$, $y(0) = 0$ without restrictions on the axis, and with `xlim([0,1.5])`

As an example consider the predator-prey system (Lotka-Volterra) representing the evolution of two populations. $u_1 = u_1(t)$ and $u_2 = u_2(t)$:

$$\begin{cases} \frac{du_1}{dt} = au_1 - bu_1u_2, \\ \frac{du_2}{dt} = -cu_2 + du_1u_2 \end{cases} \tag{3}$$

with initial populations $u_1(0) = 10$ and $u_2(0) = 60$. The parameters a, b, c, and d are set to $a = 0.8$, $b = 0.01$, $c = 0.6$, and $d = 0.1$. The time unit depends on the type of populations considered.

Although the ODE problem is now defined with two equations, the MATLAB implementation is very similar to the case of a single ODE, except that vectors must now be used to describe the unknown functions.

```
1  function ex_with_2eqs
2     t0 = 0; tf = 20; y0 = [10;60];
3     a = .8; b = .01; c = .6; d = .1;
4     [t,y] = ode45(@f,[t0,tf],y0,[],a,b,c,d);
5     u1 = y(:,1); u2 = y(:,2);
6     figure(1)
7     subplot(2,1,1); plot(t,u1,'b-+'); ylabel('u1');
8     subplot(2,1,2); plot(t,u2,'ro-'); ylabel('u2');
9     figure(2)
10    plot(u1,u2); axis square;  %plotting phase plot
11    xlabel('u_1'); ylabel('u_2');
12 end
13    %--------------------------------------
14 function dydt = f(t,y,a,b,c,d)
15    u1 = y(1); u2 = y(2);
16    dydt = [ a*u1-b*u1*u2 ; -c*u2+d*u1*u2 ];
17 end
```

- In line 2 the 2×1 vector `y0` defines the initial condition for both u_1 and u_2.
- In line 4 the parameters `a, b, c, d` are passed to the ODE solver `ode45` as extra arguments (starting from the 5$^{\text{th}}$ argument in the ode45 function). The output array `y` of `ode45` now has 2 columns, corresponding to approximations for u_1 and u_2, respectively, instead of a single one.
- In line 5 the arrays `u1` and `u2` are retrieved from `y`.
- Lines 14 - 17 define the ODE system. Note that all the parameters appearing as arguments of `ode45` must appear as arguments of the function `f`. For a specific value of `t` the input `y` to `f` is a

2×1 vector, whose coefficients are the values of u_1 and u_2 at time t. Rather than referring to `y(1)` and `y(2)` in the definition of the equations on line 14, it is best again to use variable names which are easier to identify, e.g., `u1` and `u2`.

- Line 14 defines the right-hand sides of the ODE system as a 2×1 vector: the first coefficient is the first right-hand side ($\frac{du_1}{dt}$) and the second coefficient the second right-hand side ($\frac{du_2}{dt}$).
- Lines 6-10 correspond to the visualization of the results. To plot the time series of $u1$ and $u2$, we create a 2×1 array of subplots. Because the scales of $u1$ and $u2$ are different, it is best using two different graphs for $u1$ and $u2$ here. Type `help subplot` to learn more about it. On a different figure, we then plot the *phase plot* representing the evolution of u_2 in terms of u_1. Note that u_1 and u_2 vary cyclically. The periodic evolution of the two populations becomes clear from the closed curve u_2 vs. u_1 in the phase plot.

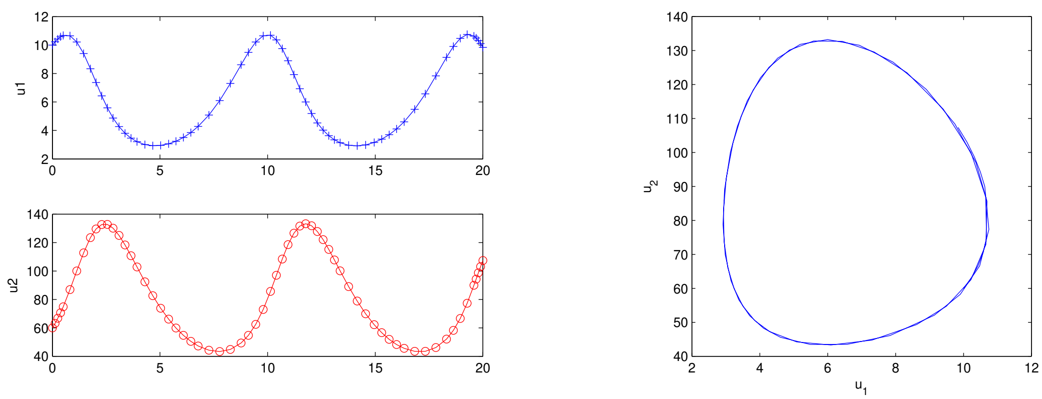

Figure 6: Lotka-Volterra example.

Reducing a Higher-Order ODE

Numerical solution to IVPs involving higher order ODEs – homogeneous or not, linear or not, can be obtained using the same MATLAB commands as in the first-order by rewriting the ODE in the form of a system of first order ODEs.

Let's start with an example. Consider the IVP

$$\frac{d^2y}{dt^2} + 5\frac{dy}{dt} + 4y = 5\sin t, \quad \text{with} \quad y(0) = -1, \frac{dy}{dt}(0) = 0. \tag{4}$$

To reduce the order of the ODE we introduce the intermediate unknown function $v = \frac{dy}{dt}$. As a result $\frac{dv}{dt} = \frac{d^2y}{dt^2}$ so that the ODE can be written $\frac{dv}{dt} + 5v + 4y = 5\sin t$. This equation only involves first-order derivatives, but we now have two unknown functions $y = y(t)$ and $v = v(t)$ with two ODEs. For MATLAB implementations it is necessary to write these ODEs in the form $\frac{d*}{dt} = \ldots$. Thus

$$\frac{d^2y}{dt^2} + 5\frac{dy}{dt} + 4y = 5\sin t \quad \Leftrightarrow \quad \begin{cases} \frac{dy}{dt} = v, \\ \frac{dv}{dt} = 5\sin t - 5v - 4y. \end{cases} \tag{5}$$

Initial conditions from (4) must also be transformed into initial conditions for y and v. Simply,

$$y(0) = -1, \;\; \frac{dy}{dt}(0) = 0 \quad \Leftrightarrow \quad \begin{cases} y(0) = -1, \\ v(0) = 0. \end{cases} \tag{6}$$

EXERCISES

Instructions: Use the provided Live Script for your lab write-up. Just like for all the other lab reports, unless otherwise specified, include in your lab report all M-files, figures, MATLAB input commands, the corresponding output, and the answers to the questions.

1. (a) Modify the function `ex_with_2eqs` to solve the IVP (4) for $0 \le t \le 60$ using the MATLAB routine `ode45`. Call the new function `LAB04ex1`.
   Let `[t,Y]` (note the upper case `Y`) be the output of `ode45` and `y` and `v` the unknown functions. Use the following commands to define the ODE:

   ```
   function dYdt= f(t,Y)
   y=Y(1); v=Y(2);
   dYdt = [v; 5*sin(t)-5*v-4*y];
   ```

   Plot $y(t)$ and $v(t)$ in the same window (do not use `subplot`), and the phase plot showing v vs y in a separate window.
   Add a legend to the first plot. (Note: to display $v(t) = y'(t)$, use `'v(t)=y''(t)'`).
   Add a grid. Use the command `ylim([-3.1,3.1])` to adjust the y-limits for both plots. Adjust the x-limits in the phase plot so as to reproduce the pictures in Figure 7.

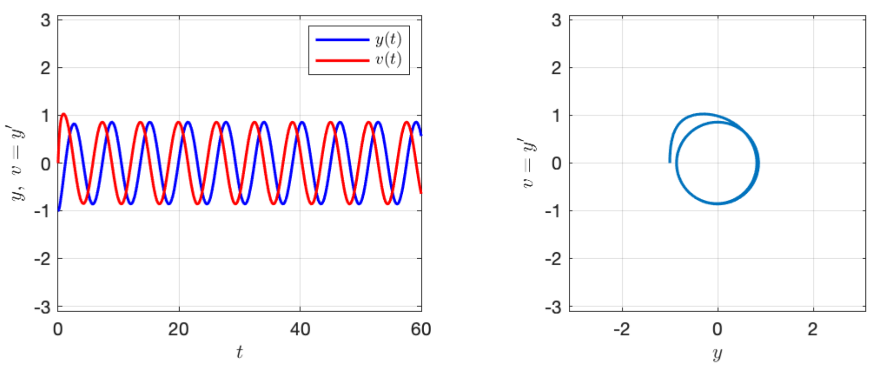

Figure 7: Time series $y = y(t)$ and $v = v(t) = y'(t)$ (left), and phase plot $v = y'$ vs. y for (4).

   (b) By reading the matrix `Y` and the vector `t`, find (approximately) the <u>last three</u> values of t in the interval $0 \le t \le 60$ at which y reaches a local maximum. Note that, because the M-file `LAB04ex1.m` is a function file, all the variables are local and thus not available in the Command Window. To read the matrix `Y` and the vector `t`, you need to modify the M-file by adding the line `[t, Y(:,1), Y(:,2)]`.
   Do not include the whole output in your lab write-up. Include only the values necessary to answer the question, i.e. just the rows of `[t, y, v]` with local y-maxima and the adjacent rows. To quickly locate the desired rows, recall that the local maxima of a differentiable function appear where its derivative changes sign from positive to negative. (Note: Due to numerical approximations and the fact that the numerical solution is not necessarily computed at the exact t-values where the maxima occur, you should not expect $v(= y')$ to be exactly 0 at local maxima, but only close to 0).

   (c) What seems to be the long term behavior of y?

   (d) Modify the initial conditions to $y(0) = -1.5$, $v(0) = 1.3$ and run the file `LAB04ex1.m` with the modified initial conditions. Based on the new graphs, determine whether the long term behavior of the solution changes. Explain. Include the pictures with the modified initial conditions to support your answer.

Nonlinear Problems

Nonlinear problems do not present any additional difficulty from an implementation point of view (they may present new numerical challenges for integration routines like `ode45`).

EXERCISES

2. (a) Consider the modified problem

$$\frac{d^2y}{dt^2} + 5y^2\frac{dy}{dt} + 4y = 5\sin t, \quad \text{with} \quad y(0) = -1,\ \frac{dy}{dt}(0) = 0. \tag{7}$$

The ODE (7) is very similar to (4) except for the y^2 term in the left-hand side. Because of the factor y^2 the ODE (7) is nonlinear, while (4) is linear. There is however very little to change in the implementation of (4) to solve (7). In fact, the only thing that needs to be modified is the ODE definition.

Modify the function defining the ODE in `LAB04ex1.m`. Call the revised file `LAB04ex2.m`. The new function M-file should reproduce the pictures in Fig 8.

Include in your report the changes you made to `LAB04ex1.m` to obtain `LAB04ex2.m`.

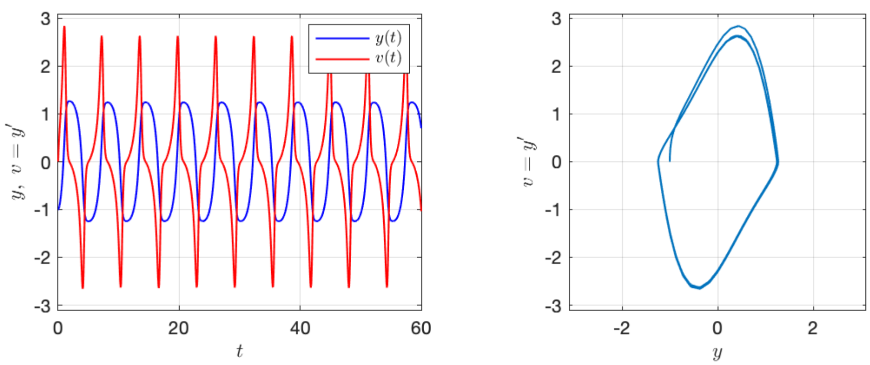

Figure 8: Time series $y = y(t)$ and $v = v(t) = y'(t)$ (left), and phase plot $v = y'$ vs. y for (7).

(b) Compare the output of Figs 7 and 8. Describe the changes in the behavior of the solution in the short term.

(c) Compare the long term behavior of both problems (4) and (7), in particular the amplitude of oscillations.

(d) Modify `LAB04ex2.m` so that it solves (7) using Euler's method with $N = 500$ in the interval $0 \le t \le 60$ (use the file `euler.m` from LAB 3 to implement Euler's method; do not delete the lines that implement `ode45`). Let `[te,Ye]` be the output of `euler`, and note that `Ye` is a matrix with two columns from which the Euler's approximation to $y(t)$ must be extracted. Plot the approximation to the solution $y(t)$ computed by `ode45` (in black) and the approximation computed by `euler` (in red) in the same window (you do not need to plot $v(t)$ nor the phase plot). Are the solutions identical? Comment. What happens if we increase the value of N?

3. Solve numerically the IVP

$$\frac{d^2y}{dt^2} + 5y\frac{dy}{dt} + 4y = 5\sin t, \quad \text{with} \quad y(0) = -1,\ \frac{dy}{dt}(0) = 0$$

in the interval $0 \le t \le 60$. Include the M-file in your report.

Is the behavior of the solution significantly different from that of the solution of (7)?

Is MATLAB giving any warning message? Comment.

A Third-Order Problem

Consider the third-order IVP

$$\frac{d^3y}{dt^3} + 5y^2\frac{d^2y}{dt^2} + 10y\left(\frac{dy}{dt}\right)^2 + 4\frac{dy}{dt} = 5\cos t, \quad \text{with} \quad y(0) = -1,\ \frac{dy}{dt}(0) = 0,\ \frac{d^2y}{dt^2}(0) = -0.5. \tag{8}$$

Introducing $v = \frac{dy}{dt}$ and $w = \frac{dy^2}{dt^2}$ we obtain $\frac{dv}{dt} = w$ and $\frac{dw}{dt} = \frac{d^3y}{dt^3} = 5\cos t - 5y^2w - 10yv^2 - 4v$. Moreover, $v(0) = \frac{dy}{dt}(0) = 0$ and $w(0) = \frac{d^2y}{dt^2}(0) = -0.5$. Thus (8) is equivalent to

$$\begin{cases} \frac{dy}{dt} = v, \\ \frac{dv}{dt} = w, \\ \frac{dw}{dt} = 5\cos t - 5y^2w - 10yv^2 - 4v \end{cases} \quad \text{with} \quad \begin{cases} y(0) = -1, \\ v(0) = 0 \\ w(0) = -0.5. \end{cases} \tag{9}$$

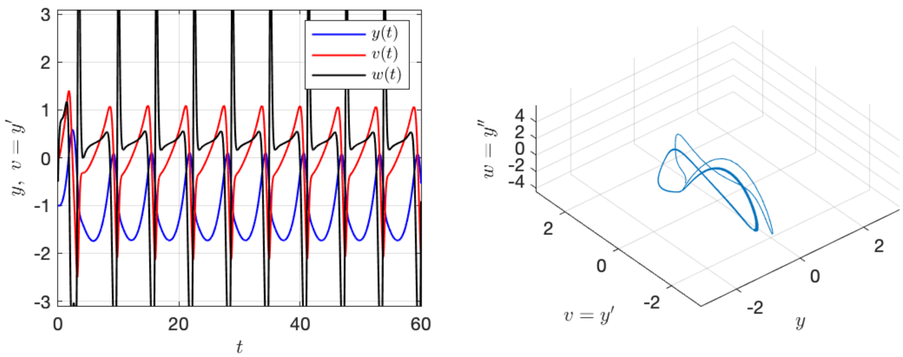

Figure 9: Time series $y = y(t)$, $v = v(t) = y'(t)$, and $w = w(t) = y'(t)$ (left), and 3D phase plot $v = y'$ vs. y vs $w = y''$ for (8) (rotated with `view([-40,60])`).

4. Write a function M-file that implements (8) in the interval $0 \le t \le 60$. Note that the initial condition must now be in the form `[y0,v0,w0]` and the matrix `Y`, output of `ode45`, has now three columns (from which y, v and w must be extracted). On the same figure, plot the three time series and, on a separate window, plot the phase plot using

```
figure(2); plot3(y,v,w);
hold on;  view([-40,60])
xlabel('y'); ylabel('v=y'''); zlabel('w=y''''');
```

   Do not forget to modify the function defining the ODE.
   The output is shown in Figure 9. The limits in the vertical axis of the plot on the left were deliberately set to the same ones as in Figure 8 for comparison purposes, using the MATLAB command `ylim([-3.1,3.1])`.
   You can play around with the 3D phase plot, rotating it by clicking on the circular arrow button in the figure toolbar, but submit the plot with the view value `view([-40, 60])` (that is, azimuth $= -40°$, elevation $= 60°$).