

# Computer lab for MAN460

(version 20th April 2006, corrected 30th January 2009)

## Instructions

Lab reports should be handed in before March 3. Please do not hand in a lot of compute code. Instead write a readable account of what you have learned from the laboration. You may cooperate with other students, but the reports should be written individually.

## Prerequisites

Matlab is rather user friendly, and to do the first problems, it is enough to write an m-file consisting of two lines, and then to give one matlab command before plotting the solutions. This is described in the appendix to these notes. For the last problem, one must write a very simple program. You should look in the matlab online manual to get more information about the different commands.

## The objectives of the lab

One objective is to demonstrate some interesting phenomena related to ordinary differential equations.

- Phase portraits for linear systems
- The influence of damping and forcing on a pendulum; resonance
- The non-linear pendulum: how does the frequency depend on the initial values?
- Stiff problems, with fast and slow components.
- The Hopf bifurcation

- Stability and instability

The lab also aims at demonstrating phenomena that are related to the numerical calculation. In particular there is a discussion of implicit methods.

Finally, it gives an opportunity to become more comfortable with using matlab as a tool.

## Lab tasks

1. Let  $A$  be a  $2 \times 2$  – matrix and  $\underline{x}$  a solution to the system

$$\underline{x}'(t) = A \cdot \underline{x}(t), \quad t \in \mathbb{R}.$$

The trajectory  $\{\underline{x}(t) : t \in \mathbb{R}\}$  is an ellipse, or else it approaches  $\underline{0}$  or  $\infty$ , when  $t \rightarrow \infty$ . The trajectory may be a spiral or a straight line. The shape of the trajectory depends on the matrix  $A$  and the initial values. Demonstrate different possibilities!

2. Study the differential equation

$$y''(t) + \alpha y'(t) + \omega_0^2 y(t) = k \cos(\omega t), \quad t \in \mathbb{R},$$

for different values of  $\alpha, k \geq 0$  and  $\omega_0, \omega > 0$ , and discuss the results.

Some interesting questions are

- What happens if  $\alpha = 0$ , and  $\omega = \omega_0$  (resonance)? Is there a similar phenomenon when  $\alpha > 0$ ?
- How does the presence of friction (i.e.  $\alpha > 0$  instead of  $\alpha = 0$ ) influence the behaviour of the solution? Look both at cases when  $k = 0$  and when  $k > 0$ .

3. Compare the solutions to the pendulum equation,

$$y''(t) + \sin(y(t)) = 0, \quad t \in \mathbb{R},$$

and the linear equation (for the “spring pendulum”)

$$x''(t) + x(t) = 0, \quad t \in \mathbb{R}.$$

Take the initial values  $y'(0) = x'(0) = 0$  and  $y(0) = x(0) = \alpha$  for different  $\alpha$ . It is interesting to investigate how the frequency (or the period) depend on the initial value  $\alpha$ .

4. The solution to an initial value problem may depend in a very sensitive way on the initial values. The following system provides an example:

$$x''(t) = -\sin(x(t)) - 0.2x'(t) + 0.52 \cos(0.666t), \quad t \in \mathbb{R}.$$

It is interesting to study the solution on the interval  $[200, 350]$ , where the initial conditions (at time 0 (!)) are

a)  $(-0.0883; 0.8), \quad (-0.0885; 0.8) \quad (-0.0888; 0.8).$

The differential equation describes a certain kind of pendulum with an external force. The appendix explains how to make a movie showing the motion of the pendulum.

5. Plot the phase portrait to the Volterra – Lotka model for a predator prey system. Is it possible to control the system by killing a given number of predators at a given instant of time?

The Volterra – Lotka model is

$$\begin{aligned}x'(t) &= ax(t) - bx(t)y(t) \\y'(t) &= -cy(t) + dx(t)y(t)\end{aligned}$$

6. The Hopf bifurcation: the system of ordinary differential equations

$$\begin{aligned}y_1'(t) &= \alpha - y_1(t) - \frac{4y_1(t)y_2(t)}{1 + y_1^2(t)}, \\y_2'(t) &= \beta y_1(t) \left(1 - \frac{y_2(t)}{1 + y_1(t)^2}\right),\end{aligned}$$

gives an approximate description of a chemical reaction;  $\alpha$  and  $\beta$  are parameters.

Let

$$\beta_c := \frac{3\alpha}{5} - \frac{25}{\alpha}.$$

If  $\beta > \beta_c$ , then the amplitudes of the solution decreases, and the trajectories are spirals that converge to some point.

If  $\beta < \beta_c$  then the solutions are oscillatory, and the trajectories converge to a closed trajectory. The parameter value where the solution changes character is called a bifurcation point, and in this case it is a so called Hopf bifurcation.

Suitable parameters for studying the system are e.g.

$$y_1(0) = 0, \quad y_2(0) = 2, \quad 0 \leq t \leq 20, \quad \alpha = 10$$

and  $\beta = 2$  and  $\beta = 4$ . The critical parameter,  $\beta_c = 3.5$  is of particular interest.

7. Use the explicit Euler method to solve the equation

$$y'(t) = -5ty(t)^2 + \frac{5}{t} - \frac{1}{t^2}, \quad y(1) = 1,$$

on the interval  $[1, 25]$ . Chose the step-lengths  $h = 0.19$ ,  $h = .21$  and  $h = 0.4$ . Compare the result with the exact solution

$$y(t) = \frac{1}{t}.$$

## Appendix A: Solution of IVPs with MATLAB

First we give some explicit, rather simple, examples, and then we explain how to treat problems with parameters.

### Example 1

You can do the following in order to get the solution to the initial value problem

$$x'(t) = -x(t), \quad x(0) = 1,$$

on the interval  $[0, 5]$ :

**1** Write a file called `a.m`.

(the only requirement for the filename is that it ends with `.m`.)

The file should contain the following line

```
function bp=a(t,x)
bp=-x;
```

**2** Save the file

**3** Use the following matlab commands:

```
>> [t,x]=ode45(@a,[0 5],1);
>> plot(t,x)
```

This shows you the solution.

Some comments to the command

```
[t,x]=ode45(@a,[0 5],1);
```

1. If  $a$  is the name of the function, then the file should be called `a.m`, and the name of the function is sent to the first argument of `ode45` with `@a`. If you use an old version of matlab, then you should write `[t,x]=ode45('a',[0 5],1);`

2.  $[0, 5]$  is the interval where we want to compute the solution  $x(t)$ . Thus one writes `[7 12]` if one wants to compute the solution over the interval  $[7, 12]$ . Here matlab choses how to divide the interval into sub intervals to obtain the desired accuracy of the solution, and the vector `t` will contain all the discretization points that matlab has chosen. It is possible to replace `[0 5]` by e.g. `0:0.1:5` if one wants to decide the discretization points. This may be useful in order to make nice plots of the solution.
3. The initial value (in this case 1) is given as the third argument to `ode45`. For a system of equations, this must be a column vector.
4. By removing `','`, one obtains more details. So, for example, by writing

```
>> [t,x]=ode45('a',[0 5],1)
```

one obtains a list of all  $t$ -values chosen by matlab, and the corresponding values of  $x(t)$ . For this example we get

```
t =
    0
  0.0502
  0.1005
  0.1507
    ⋮
```

and

```
x =
  1.0000
  0.9510
  0.9044
  0.8601
    ⋮
```

which means that  $x(0) = 1.0000$ ;  $x(0.0502) = 0.9510$ ;  $x(0.1005) = 0.9044$ ;  $x(0.1507) = 0.8601$ ; ...

**Example 2** If, for some reason, one would wish to compare the solution in Example 1 with

$$x'(t) = -\cos^2(t) \sin\left(\frac{x(t)^2}{1+t^2}\right), \quad x(0) = 1.$$

one proceeds as in the previous example. It may not be very interesting to compare the two solutions, but it is instructive from the point of view of matlab.

1 Create a file `b.m` with the content

```
function bp=b(t,x)
bp=-cos(t).*cos(t).*sin(x.*x./(1+t.*t));
```

2 Use the following matlab commands:

```
>> [t,x]=ode45(@b,[0 5],1);
>> plot(t,x)
```

Now  $x$  contains the solution to this equation. To compare with the solution from the previous example, you can proceed as follows:

```
>> hold on;
```

(the command `hold on` keeps the previous graph when a new one is drawn)

```
>> [t,x]=ode45('a',[0 5],1);
>> plot(t,x,'r')
```

Now the solutions to Example 1 and Example 2 are shown in the same graph. You should consult the manual pages to learn more about the numerous options for the command `plot`. One of the more useful commands is `axis`, and another one is `title`.

The command `hold off` is the opposite of `hold on`.

**Example 3** In this example we consider a system of ode's. Consider

$$\begin{aligned}x_1'(t) &= 5x_1(t) + 2x_2(t), \\x_2'(t) &= \sin(x_1(t)), \\x_1(3) &= 4, \quad x_2(3) = 2,\end{aligned}$$

on the interval  $[3, 4]$ . To find the solution you may proceed as follows.

**1** Create a file with the name `c.m` and which contains the lines

```
function bp=c(t,x)
bp=[5*x(1)+2*x(2);sin(x(1))];
```

( note that `bp` should be a column vector!)

**2** Use the matlab command

```
>> [t,x]=ode45(@c,[3 4],[4;2]);
```

Matlab then computes the solutions  $x_1$  and  $x_2$ . The only difference with the two first examples is that the initial value is a (column) vector;

`[4;2]` corresponds to the initial value  $x_1(3) = 4$ ,  $x_2(3) = 2$ .

There are different options for plotting the result:

```
>> plot(t,x(:,1))
```

This would show the first component,  $x_1$ , of the solution, and similarly

```
>> plot(t,x(:,2))
```

shows the second component.

The command

```
>> plot(x(:,1),x(:,2))
```

plots the trajectory  $\{(x_1(t), x_2(t)) : 3 \leq t \leq 4\} \subset \mathbb{R}^2$ .

The time evolution of the two components of the solution can also be plotted together in the same graph, e.g. with the command

```
>> plot(t,x(:,1),t,x(:,2),'r')
```

### Some comments about ode-solvers in matlab

- There is a large number of ode-solvers that from the point of view of the user work in the same way: `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`. These methods are all described in the matlab help pages. Some of these are particularly suitable for so-called stiff differential equations. A stiff system of ode's is one where there are very different time-scales involved. For example, if you want to make a model that accurately keeps track of the influence of the moon on the pendulum of your grand father clock, then there are two very disparate time-scales: the month (or at least the day), and the period of the pendulum, say a second.
- With the command  
`[t,y] = ode45(@f,[t0,t1],y0)`,  
the vector `t` will contain the time-points where matlab chose to evaluate the solution, and `y` will contain the solution evaluated at these points. If instead you give the command  
`sol = ode45(@f,[t0,t1],y0)`  
then matlab returns a *structure*, `sol`, that contains all information. This structure may be fed into the function `deval` to get easy access of the result. An example is the following (taken from the matlab help pages)

```
sol = ode45(@f,[0 20],[2 0]);  
x = linspace(0,20,100);  
y = deval(sol,x,1);  
plot(x,y);
```

Here, in the first line, `ode45` solves the system defined in a function `f.m` and puts the result in the structure `sol`.

In the second line a vector with 100 elements, such that  $\mathbf{x}(1) = 0$ ,  $\mathbf{x}(100)=20$ , and all the other are evenly distributed so that they are endpoints in 99 intervals (one point more than the number of intervals is needed).

The third line extracts the first component of the solution to the system, evaluated at all the points of  $\mathbf{x}$ .

Finally the result is plotted.

Note that this is a more convenient (and efficient) way of obtaining the solution exactly in the desired points, than to give a longer vector instead of `[t0,t1]` to define the interval where the solution is computed.

- One can supply more arguments to the ode solvers, and the full description is given in the help pages. For example, one can write

```
[T,Y] = solver(odefun,tspan,y0,options,p1,p2...)
```

The new arguments are `options` and `p1, p2, ...`. The `options` argument can be used for changing the desired accuracy of the method, putting limits on the step size, on the number of function evaluations etc. The argument is created by a function called `odeset`, which is described in the help pages. The arguments `p1, p2, ...` are used to pass arguments to the function defining the differential equation. To supply (two) user defined arguments without supplying an `options` argument, one can write

```
[T,Y] = solver(odefun,tspan,y0,'',p1,p2)
```

So, for example, if one wishes to solve

$$\begin{aligned}y' &= a_1y - a_2y^2 \\ y(0) &= 1,\end{aligned}$$

then one could create a file `f.m` with the following lines:

```
function yp = f(t,y,a1,a2)
yp = a1*y - a2*y.*y;
```

(please note the difference between “\*” and “.\*”!)

Then the call

```
[T,Y] = solver(@f, [0,10], 1.0, '', 13.5, 27)
```

would give the solution corresponding to the parameter values  $a_1 = 13.5$  and  $a_2 = 27.0$ . In this way it is easy to change parameters without having to rewrite the code. And it also makes it possible to compute whole families of solutions corresponding to different parameters values by using loops.

*Note* In the previous version of the lab description the same is achieved using so-called global variables. Global variables are, however, rather dangerous to use, and the method described here is preferable.

**Example 4** Here is an example on how to make a movie ... it starts off as in example 3. In the second step we solved the ode:

```
sol = ode45(@c, [0 20], [2 0]);  
x = linspace(0,20,500);  
y = deval(sol,x,1);
```

Now we want to plot this so as to make a movie:

```
tail_length = 10; % do experiments with this  
for j=1:500-tail_length+1;  
    plot( x(j:j+tail_length-1),y(j:j+tail_length-1),'red'))  
    axis([-2,2,-2,2]); % this is to fix the size of the plotting area  
    F(j) = getframe;  
end
```

Now you will already have seen all the images for the movie, but you can play it again without having to redo the calculation:

```
movie(F)
```