HOMEWORK 3: Phase portraits
for Mathmatical Models, Analysis and Simulation, Fall 2011
Report due Mon Oct 22, 2012.
Maximum score 6.0 pts.

Three problems are to be solved in this homework assignment. The first consists in scaling of a homogeneous and a nonhonogeneous differential equation. The second is a study of a dynamical system with a simple bifurcation, and the third problem deals with predator-prey models. Hints for plotting phase-planes with Matlab are enclosed.

Problem 1 (1.0 p)

The differential equation for damped free oscillations is a second order homogeneous linear equation. It can be written in the following form:

$$m\frac{d^2u}{dt^2} + c\frac{du}{dt} + ku = 0.$$

Here the state variable u is the deviation from a stationary position and t is the time. The equation contains three parameters, i.e. the mass m, the damping constant c, and the spring constant k.

We first introduce a dimensionless time τ by scaling of the original time variable t. After this is done, we derive a differential equation for $u(\tau)$ from the equation above for $u(t)$. One of the results is that the equation for nondimensionalized variables will often contain fewer parameters than the original equation. This fact will simplify the formal mathematical treatment of the problem, and it will also lead to improved insight about both the problem and its solution.

We reparametrize by introducing two parameters as follows: The undamped (angular) frequency ω_0 is defined by $\omega_0 = \sqrt{k/m}$ and the critical damping c_0 is defined by $c_0 = 2\sqrt{km}$. Note that these new parameters are not dimensionless. We use them to define the dimensionless damping constant $\alpha = c/c_0$ and the dimensionless time $\tau = \omega_0 t$.

Your first task is to derive the differential equation for $u(\tau)$ and to show that it equals

$$u'' + 2\alpha u' + u = 0.$$

Observe that the scaling of the time has reduced the number of parameters from three to one. With forced oscillations we study the differential equation

$$m\frac{d^2u}{dt^2} + c\frac{du}{dt} + ku = F\cos\omega t.$$

Here there are two additional parameters, the amplitude F and the (angular) frequency ω of the external force. It is easy to nondimensionalize the driving frequency ω since we already have introduced the reference frequency ω_0. Thus we introduce a dimensionless driving frequency β. Furthermore we introduce a dimensionless state variable y by proper choice of scale factor (this should contain F, of course). Your second task is to derive a differential equation for $y(t)$. In this case the scaling of both the dependent and the independent variables has reduced the number of parameters from five to two: α and β.

Problem 2 (3.0 p) Consider the following linear system of differential equations:

$$\mathbf{u}' = -\begin{bmatrix} 1 & 1 \\ P & 1 \end{bmatrix} \mathbf{u},$$

where P is a real-valued parameter.

(1.5) a) Determine the type of phase portrait as a function of P. (There are five different types.) Determine the slopes of the named manifolds (in those cases where they exist) as a function of P.

(1.5) b) Determine the limit of $\mathbf{u}(t)$ and of $y(t)/x(t)$ (with $\mathbf{u}(t) = (x(t), y(t))^T$) as $t \to \infty$ in all five cases. Note that the result may depend on the initial point $\mathbf{u}(0)$ and on the value of P. Determine the bifurcation value P_0 of P where stability is lost and describe the type of phase portrait on either side of it.

Problem 3 (3.0 p)

(0.5) a) Consider the two-species models of the form,

$$\frac{dx}{dt} = a(x, y)x,$$
$$\frac{dy}{dt} = b(x, y)y,$$

where x and y denote the populations of two species, and a and b denote the corresponding growth rates. Assume that a and b are smooth functions. We are now interested in a predator-prey model, where y denotes the predator and x denotes the prey population, which have the following (meaningful) assumptions:

(i) If there is not enough prey, the predator population declines.

(ii) An increase in the prey population increases the growth rate of the predator.

(iii) If no predators are present, a small prey population will increase.

(iv) If the prey population goes beyond a certain size, it must decrease.

(v) If the predator population increases, the growth rate of the prey population declines.

Express the assumptions $(i)-(v)$ in terms of a, b, x and y. Verify that the simple model with $a(x,y) = \alpha - \beta y - \lambda x$ and $b(x,y) = \delta x - \gamma - \mu y$ $(\alpha, \beta, \gamma, \delta, \lambda, \mu > 0)$ satisfy $(i)-(v)$.

(2.5) b) Consider the predator-prey model of a),

$$x' = (\alpha - \beta y - \lambda x)x,$$
$$y' = (\delta x - \gamma - \mu y)y,$$

where $\alpha, \beta, \gamma, \delta, \lambda, \mu$ are positive constants. Show that $x(t), y(t) > 0$ for $t > 0$ if $x(0), y(0) > 0$. Determine the critical points of this system!

To reduce the number of parameters, introduce the scaling, $\tau = ct$, $u = dx$ and $v = ey$. Choose the scalings to make the transformed equations have three parameters which are combinations of the original ones, such that

$$u' = (1 - v - Au)u,$$
$$v' = (u - C - Bv)v.$$

What is the condition on A,B, and C that there be a strictly positive critical point?

Consider two sets of parameters,

1 . A = 1,C = 1/2,B = 1

2 . A = 1,C = 1/2,B = 8.

Characterize the stability of the strictly positive critical point in both cases. Hand in plots of the phase portrait of the linearized equations and original equations around one of these critical points which shows the difference between the solution of the linearized and "non-linearized" equations.

How to plot phase portraits with Matlab.
There are some orbits in phase portraits that are more important, and carry more information, than others. Among the most important are the orbits on the so-called named manifolds. The term named manifolds is used to refer to the slow and fast manifolds of nodes and the stable and unstable manifolds of saddle points. We require of any plot of phase portraits that it includes orbits on all named manifolds.

We show by treating a concrete example how you can use Matlab to plot the phase portrait of a linear system in the plane. The system of equations is written u = `AA*u`, where `AA` is a given 2×2-matrix and `u` is a column vector. The matrix is denoted in Matlab with two letters instead of one since it will be a so-called global variable, and it is customary in Matlab to denote global variables with long names with capital letters. The first step is to write an m-file that computes the left-hand side of the system of equations when the righthand side is known. Let the name of the file be e.g. `lin.m`. The file may look like follows:

```
% lin.m Linear diff eq system in two variables.
function uprim = lin(t,u)
global AA
uprim = AA*u;
```

The variable `AA` is global. It is declared global both in this mfile and in the main program. Its value is set in the main program. After you have saved this file, move to the main Matlab window and give the following commands:

```
>> global AA
>> AA = [-1 1;1 -3];
>> tspan = [0 5];
>> Va = [-1 1 -1 1]; axis(Va), axis equal, hold on, grid
```

The second line sets the value `AA`. If you wish to work with another system you just feed in a new matrix `AA`. The third line above sets the time span from initial value `t0 = 0` to final value `tf = 5` - of the independent variable `t`. If you later wish to run backwards in time, then all you have to do is to set `tspan = [0,-5]`, that means `t < 0`. The command `axis(Va)` in the fourth line defines a window in the x-y-plane, whose corners are determined by `Va = [xmin xmax ymin ymax]`. `axis equal` sets equal units in both x- and y-directions. The command `hold on` allows you to plot several curves in the same figure without removing earlier plots. The command grid, finally, plots a grid in the plane.

The main tool for plotting phase portraits is Matlabs command `ode23` or `ode45`. We start plotting one orbit of the phase portrait by giving the following commands:

```
>> u0 = [1;1];
>> [t,u] = ode23(@lin,tspan,u0); plot(u(:,1),u(:,2))
```

The initial point is denoted `u0`. The semicolon makes it a column vector. The second line contains the main command line. It will be used repeatedly in the sequel. Its first command asks Matlab to solve the system of equations in `lin.m` with the initial value `u(t0) = u0`. The solution is given in the form of a column vector of `t`-values in the interval from `t0` to `tf`, and a matrix of corresponding values of `u`.

The number of `t`-values, and their spacing, is determined by the program. It is used here with default values for tolerances etc. Read the help-file on `ode23` (and `ode45`) to find out what they are and how to change them! You can check the number of steps the program has taken by giving the command `size(t)`. The matrix of `u`-values has two columns. The first one contains a vector of values of x, and the second one contains a column of values of y. Check with `size(u)`.

The second command on the main command line plots the orbit. Here, `u(:,1)` and `u(:,2)` denote the first and second columns of the matrix u.

When you wish to plot one more orbit, then you give a new initial point `u0`, after which you repeat the main command line. You do not have to retype any commands; the old ones can be recalled with the arrow-up key. Choose seven additional initial points uniformly distributed on the circumference of the window determined by `Va`. You might want to experiment with a loop like this[1].

```
n = 7;
st = exp(i*2*pi*(1:n)/n);
for s = st
  u0 = [real(s);imag(s)];
  [t,u] = ode23(@lin,tspan,u0); plot(u(:,1),u(:,2));
end
```

The phase portrait that you have produced so far shows eight orbits that all approach the origin with the same slope. This behavior is typical of the stable node. To complete the phase portrait you need to plot the orbits on the slow and fast manifolds.

You find the orbits on the slow manifold by choosing initial points sufficiently far away from the origin. I suggest that you try `u0 = [10;10]` and then `u0 = -u0`.

The orbits on the fast manifold are found in a different way. One way would be to experiment with a large number of initial points on the circumference of the window until you find one that gives you one of the orbits. An alternative is to start near the origin and run backwards in time. The initial point should then be chosen in a direction from the origin determined by the slope of the fast manifold.

[1] A more advanced possibility consists of using Matlab script files

To determine the initial point, we need to determine eigenvalues and eigenvectors of the matrix `AA`:

```
>> [V,D] = eig(AA)
```

The matrix `V` contains two eigenvectors as columns, and the matrix `D` has the corresponding eigenvalues on its diagonal. The slopes of the two eigenvectors, `sls` and `slf`, are determined with the following commands. The last letter in the slope designation is `s` for the slow manifold and `f` for the fast one.

```
>> v1 = V(:,1); sls = v1(2)/v1(1)
>> v2 = V(:,2); slf = v2(2)/v2(1)
```

Change the numbering if the slow manifold eigenvector is the second column of `V`! To plot an orbit on the fast manifold we choose an initial point whose coordinates are determined by the fast manifold. In addition we run time backwards:

```
>> u0 = 0.02*v2; tspan = [0,-5];
```

After this we execute the main command line.

You get the other orbit on the same manifold by changing the sign of u0 and then executing the main command line.

When the phase portrait is finished and you are satisfied with its esthetic appearance, you may wish to equip it with a title. You can for example write as follows, replacing my name with yours:

```
>> title(Phase portrait by my name)
>> xlabel(x), ylabel(y)
```

Finish off by giving the command print:

```
>> print -deps figure.eps
```

Even simpler, you may use the menu of the figure window (`Export`). Get your copy at the printer, and equip the orbits with arrows that show the directions they are traversed as time increases.

Let us plot one more phase portrait. The window is erased with the command `clf` (clear figure). To prepare for a new phase portrait and reset the direction of the time axis to normal you give the following command:

```
>> clf, axis(Va), axis equal, hold on, grid, tspan = [0,5];
```

Feed in a new matrix as follows:

```
>> AA = [-1 4;4 -3];
```

Now plot the orbits for a number of initial points on the boundary of the window determined by `Va`. The orbits indicate that you are dealing with a saddle point. Use the command `eig` to determine eigenvalues and eigenvectors of the matrix `AA`. Determine the slopes `sls` and `slu` of the stable and unstable manifolds from the stable and unstable eigenvectors `vs` viz. `vu`. Use initial points `u0 = 0.02*v` and `u0 = -u0`, where `v` is replaced by `vs` and `vu`. Let time run forward to get the unstable manifold and backward to get the stable one.

It should be noted that the rules that we have followed for plotting orbits on

the named manifolds could be somewhat simplified if we are only concerned with linear systems. Our rules have the virtue that they can also be used when we are dealing with nonlinear systems.

We summarize the rules for plotting orbits on the named manifolds of stable nodes and of saddle points that have been used above:

- The orbits on the slow manifold of a stable node are plotted with initial points far away from the critical point, and with time running forward.

- All other orbits are plotted with initial points close to the critical point. The initial point is chosen on a straight line through the critical point whose slope is given by the corresponding eigenvector.

- The orbits on the unstable manifold of the saddle point are plotted with time running forward.

- The orbits on both the stable manifold of the saddle point and on the fast manifold of the stable node are plotted with time running backward.

- An unstable node is transformed into a stable one by letting time run backward.

A note on using `ode45`: The initial value solver `ode45` has a much higher order than `ode23`. On one hand, this is nice: the computation time is much lower than with `ode23`. On the other hand, the number of steps taken (e.g., `size(t)`) is much lower such that the phase plot may become a polygon with visible corners. The latter can be avoided by using an alternative main command line:

```
>> sol = ode45(@lin,tspan,u0);
>> t = linspace(tspan(1),tspan(end),101);
>> u = deval(sol,t);
>> plot(u(1,:),u(2,:))   % Note the transposition of u!
```

For details please consult Matlabs documentation.