MATH 415 (ODE HW & Background)

A. Assadi

Basic MATLAB for Solution of ODE

The MATLAB ODE collection of programs computes the time course of a set of coupled first-order differential equations with given initial conditions. Mathematical formulation of these problems (called initial value problems, abbreviated to IVP) have the form:

$$\dot{y} = f(y,t) \ \ ; y(t_0) = y_0$$

where as usual in physics and calculus, $\dot{y} = dy/dt$.

In the following, we review a basic set of instructions that could help you get the most out of a short period of time working on the computer, using MATLAB, or translating appropriately the commands to Maple (the symbolic engine for MATLAB that uses MATLAB's numerical and linear algebra packages, hence quite compatible when syntax is observed).

The first step to solve a set of differential equations is to code a function M-file, such as as ***`ydot=ode_file(t,y)`***. To do this, the code must accept a time t and a solution y and return values for the derivatives, as the following example shows:

EXAMPLE. Let us briefly study the van der Pol equation, and its ODE file as follows. A special case is the unforced van der Pol equation is

$$\dot{x} = \rho x - y - x^3$$
$$\dot{y} = x$$

Sometimes, it is written as follows:

$$\dot{y}1 = y2$$
$$\dot{y}2 = y2(\rho - y2)^2 - y1$$

We verify directly that the origin is the only equilibrium of this equation. (Later we will show van der Pol equations to study the so-called Hopf bifurcations, and show that the that equilibrium is a point of Hopf bifurcation; in fact the only value of ρ for such a phenomenon is ρ = 0.) We graph the phase portrait (to visualize the qualitative behavior) of the van der Pol equations using the command `pplane5` , and verify that there is a unique limit cycle when ρ > 0.

Let us start by checking the MATLAB Help file for ode45:

```
>> help ode45
 ODE45  Solve non-stiff differential equations, medium order method.
 [T,Y] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the
system of differential equations y' = f(t,y) from time T0 to TFINAL
with  initial conditions Y0. Function ODEFUN(T,Y) must return a column
vector  corresponding to f(t,y). Each row in the solution array Y
corresponds to a time returned in the column vector T. To obtain
solutions at specific times T0,T1,...,TFINAL (all increasing or all
decreasing), use
 TSPAN = [T0 T1 ... TFINAL].
  [T,Y] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS) solves as above with default
integration properties replaced by values in OPTIONS, an argument
created with the ODESET function. See ODESET for details. Commonly used
options  are scalar relative error tolerance 'RelTol' (1e-3 by default)
and vector of absolute error tolerances 'AbsTol' (all components 1e-6
by default).

 [T,Y] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS,P1,P2...) passes the additional
parameters P1,P2,... to the ODE function as ODEFUN(T,Y,P1,P2...), and
```

to all functions specified in OPTIONS. Use OPTIONS = [] as a place holder if no options are set.

ODE45 can solve problems M(t,y)*y' = f(t,y) with mass matrix M that is nonsingular. Use ODESET to set the 'Mass' property to a function MASS if MASS(T,Y) returns the value of the mass matrix. If the mass matrix is constant, the matrix can be used as the value of the 'Mass' option. If the mass matrix does not depend on the state variable Y and the function MASS is to be called with one input argument T, set 'MStateDependence' to 'none'. ODE15S and ODE23T can solve problems with singular mass matrices.

[T,Y,TE,YE,IE] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS...) with the 'Events' property in OPTIONS set to a function EVENTS, solves as above while also finding where functions of (T,Y), called event functions, are zero. For each function you specify whether the integration is to terminate at a zero and whether the direction of the zero crossing matters. These are the three vectors returned by

EVENTS: [VALUE,ISTERMINAL,DIRECTION] = EVENTS(T,Y).

For the I-th event function:

VALUE(I)

is the value of the function,

ISTERMINAL(I)=1

if the integration is to terminate at a zero of this event function and 0 otherwise.

DIRECTION(I)=0 if all zeros are to be computed (the default), +1 if only zeros where the event function is increasing, and -1 if only zeros where the event function is decreasing. Output TE is a column vector of times at which events occur. Rows of YE are the corresponding solutions, and indices in vector IE specify which event occurred.

SOL = ODE45(ODEFUN,[T0 TFINAL],Y0...) returns a structure that can be used with DEVAL to evaluate the solution or its first derivative at any point between T0 and TFINAL. The steps chosen by ODE45 are returned in a row vector SOL.x. For each I, the column SOL.y(:,I) contains the solution at SOL.x(I). If events were detected, SOL.xe is a row vector of points at which events occurred. Columns of SOL.ye are the corresponding solutions, and indices in vector SOL.ie specify which event occurred.

Example

```
 [t,y]=ode45(@vdp1,[0 20],[2 0]);
 plot(t,y(:,1));

solves the system y' = vdp1(t,y), using the default relative error
tolerance 1e-3 and the default absolute tolerance of 1e-6 for each
component, and plots the first component of the solution.

Class support for inputs TSPAN, Y0, and the result of
ODEFUN(T,Y):float: double, single

See also   other ode solvers:
ode23, ode113, ode15s, ode23s, ode23t, ode23tb
options handling:  odeset, odeget
output functions:  odeplot, odephas2, odephas3, odeprint
evaluating solution:  deval
ode examples:   rigidode, ballode, orbitode

NOTE: The interpretation of the first input argument of the ODE solvers
and some properties available through ODESET have changed in this
version of MATLAB. Although we still support the v5 syntax, any new
functionality is available only with the new syntax. To see the v5
help, type in the command line   more on, type ode45, more off

 Reference page in Help browser
 doc ode45
```

Let us now use the above HELP and study the numerical solution of van der Pol equations via the following ODE m-file:

```
%%
   function ydot=vdpol(t,y)
   %VDP0L van der Pol equation. % Ydot=VDPOL(t,Y)
   % Ydot(l) = Y(2)
   % Ydot(2) = rho*(l-Y(l)^2)*Y(2)-Y(l)
   % rho = 2

   rho = 2 ;
   ydot = [y(2); rho*(l-y(l)^2)*y(2)-y(l)];
```

Note that the input arguments are `t` and `y` but that the function does not use `t`. Note also that the output `ydot` must be a column vector.

Now we use the above ODE file and solve this set of ODEs by the following commands:

```
>> tspan=[0  20]; %  time  span for integration
>> y0 = [2;0];    %  initial conditions (as a column)
>> [t,y] = ode45(@vdpol,tspan,y0);
>> size(t)        % number of time points
ans =
333  1
>> size(y)        %  (i)th column  is y(i)  at t(i)
ans  =
   333  2
>> plot(t,y(:,1),t,y(:,2),'--')
>> xlabel ( 'time' )
>> title('Figure 1. van der Pol Solution')
```

Comments on Writing MATLAB Code. Function handles are the appropriate MATLAB tool to use in the case of solutions of ODE systems, because differential equations are always written as m-files, and they are evaluated many times in the process of generating a solution over a reasonable time span. By default, if a solver is called with no output arguments, for example, `ode45(@vdpol , tspan.y0)`, the solver generates no output variables but generates a time plot. In addition to specifying the initial and final time points in `tspan`, one can identify the specific solution time points desired by simply adding them to `tspan`, for example,

```
>> y0 = [2; 0];
>> tspan = 1inspace(0,20,100);
>> [t,y] = ode45(@vdpol,tspan,yo);

» size(t) ;

ans =

100    1

>>size(y)

ans =

100    2
```

Here 100 points are gathered over the same time interval as in the earlier example. When called in this way, the solver still uses automatic step size control to maintain accuracy. It does not use fixed step integration. To gather the solution at the desired points, the solver interpolates its own solution in an efficient way that does not deteriorate the solution's accuracy.

Sometimes the set of differential equations to be solved contains a set of parameters that the user wishes to vary. Rather than open the ODE file and change the parameters before each call to a solver, the parameters can be added to the solver and ODE file input arguments, for example,

```
function  ydot=vdpol(t,y,rho)
%  VDPOL  van  den  Pol   equation.
%  Ydot=VDPOL(t,Y,rho)
%  ydot(l)  = y(2)
%  ydot(2)  = rho*(l-y(l)^2)*y(2)-y(l)

% mu = ?; now passed as an input argument
if nargin<3     % supply default if not given
rho=2;
end
ydot = [y(2); rho*(l-y(1)^2)*y(2)-y(1)];
```

```
>> rho = 10 % set rho in Command window
~~rho =~~

     10

>> ode45(@vdpol,tspan,y0,[],rho);

>> plot(t,y(:,1),t,y(:,2),'--')

>> xlabel ( 'time' )

>> title('Figure 2. van der Pol Solution, rho=10')
```

Exercises. 1. Find a solution of the system of differential equations $\dot{X} = CX$ satisfying the initial condition

`X(0) = (10 -4  9)`^{t}.

2. Find a solution to the system of differential equations $\dot{X} = CX$ satisfying the initial condition `X(0) = (2  -1  3)`^{t}.

3. Show that for some nonzero a the function $x(t) = at^5$ is a solution to the differential equation $x = x^{4/5}$. Then show that there are at least two solutions to the initial value problem `x(0) = 0` for this differential equation.

4. Use `pplane5` to investigate the system of differential equations

$$\frac{dx}{dt} = \quad -2y$$
$$\frac{dy}{dt} = -x + y$$

5. Use `pplane5` to find two independent eigen-directions (and hence eigenvectors) for this system.

6. Using (5), find the eigen-values of the coefficient matrix of the system above.

7. Find a closed form solution to the ODE system above that satisfies the initial condition `x(0) =(4  -1)`^{t}.

8. Study the time series of `v` versus `t` for the solution in (c) by comparing the graph of the closed form solution obtained in (c) with the time series graph using `pplane5`.

MATLAB Comments on Coding. We need not accept all default tolerances and options. When the defaults are not sufficient, an options structure can be passed to a solver as a fourth input argument. The MATLAB ODE suite contains the functions `odeset` and `odeget` to manage this options structure, `odeset` works similarly to the Handle Graphics `set` function in that parameters are specified in name/value pairs, for example, `options = odeset('Namel' , Valuel, 'Name2',Value2,...)`;. The available parameter names and values are described in the on-line help for `odeset`, which is shown below.