```
      h = min(hmax, 0.8*h*(tau/delta)^pow);
   end
end;

if (t < tfinal)
   disp(['Singularity likely at t = ' num2str(t)])
end
```

References

[1] T. Glad. Övningsexempel i Reglerteori. Technical report, Dept. of Electrical Engineering, Linköping University, S-581 83 Linköping, Sweden, June 1989.

[2] J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcation of Vector Fields*, volume 42 of *Applied Mathematical Sciences*. Springer, 1983.

[3] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems*, volume 8 of *Computational Mathematics*. Springer, 1987.

[4] The Math Works Inc. *PRO-MATLAB User's Guide*, January 1990.

[5] J.M.T. Thompson and H.B. Stewart. *Nonlinear Dynamics and Chaos*. Wiley, 1986.

[6] F. Verhulst. *Nonlinear Differential Equations and Dynamical Systems*. Universitext. Springer, 1990.

[7] S. Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*, volume 2 of *Texts in Applied Mathematics*. Springer, 1990.

```
% Initialization
t = t0;
hmax = (tfinal - t)/5;
hmin = (tfinal - t)/20000;
h = (tfinal - t)/100;
y = y0(:);
f = y*zeros(1,6);
tout = t;
yout = y.';
tau = tol * max(norm(y, 'inf'), 1);

if trace
   clc, t, h, y
end

% The main loop
   while (t < tfinal) & (h >= hmin)
      if t + h > tfinal, h = tfinal - t; end

      % Compute the slopes
      temp =-feval(FunFcn,t,y);
% Added a minus sign before "feval" in the ode45 code. K. Forsman

      f(:,1) = temp(:);

      for j = 1:5
         temp =-feval(FunFcn, t+alpha(j)*h, y+h*f*beta(:,j));
% Added a minus sign before "feval" in the ode45 code. K. Forsman

         f(:,j+1) = temp(:);
      end

      % Estimate the error and the acceptable error
      delta = norm(h*f*gamma(:,2),'inf');
      tau = tol*max(norm(y,'inf'),1.0);

      % Update the solution only if the error is acceptable
      if delta <= tau
         t = t + h;
         y = y + h*f*gamma(:,1);
         tout = [tout; t];
         yout = [yout; y.'];
      end
      if trace
         home, t, h, y
      end

      % Update the step size
      if delta ~= 0.0
```

```
yflag1=b>ax(3); yflag2=b<ax(4);
yflag=yflag1 & yflag2;
fflag=1;

while xflag | yflag,
        if ~xflag, T=input('New value of T: '); end;
        if xflag,
                if yflag, x10=a; x20=b; fflag=m==1;
                   elseif ~yflag2, n=length(x(:,1));
                        x10=x(n,1); x20=x(n,2);
                   elseif ~yflag1, fflag=~fflag; end;
                if fflag, [t,x]=ode45(F,0,T,[x10; x20],tol);
                        else
                  [t,x]=feval('ode45bw',F,0,T,[x10; x20],tol); end;
                plot(x(:,1),x(:,2))
        end;
        [a,b,m]=ginput(1);
        xflag=(a>ax(1)) & (a<ax(2));
        yflag1=b>ax(3); yflag2=b<ax(4);
        yflag=yflag1 & yflag2;
end;
hold off
```

```
function [tout, yout] = ode45bw(FunFcn, t0, tfinal, y0, tol, trace)

%ODE45BW  [tout,yout] = ode45bw(FunFcn,t0,tfinal,y0,tol,trace)
%
%         As ODE45 but integrates backwards.
%         Trivial modification by K. Forsman.

%   C.B. Moler, 3-25-87.
%   Copyright (c) 1987 by the MathWorks, Inc.
%   All rights reserved.

% The Fehlberg coefficients:
alpha = [1/4  3/8  12/13  1  1/2]';
beta  = [ [    1      0      0     0      0    0]/4
          [    3      9      0     0      0    0]/32
          [ 1932  -7200   7296     0      0    0]/2197
          [ 8341 -32832  29440  -845      0    0]/4104
          [-6080  41040 -28352  9295  -5643    0]/20520 ]';
gamma = [ [902880  0  3953664  3855735  -1371249  277020]/7618050
          [ -2090  0    22528    21970    -15048  -27360]/752400 ]';
pow = 1/5;
if nargin < 6, trace = 0; end
if nargin < 5, tol = 1.e-6; end
```

The m-files are under the directory `~/pub/src/matlab/`. This report is available as a compressed ps-file named `~/pub/reports/LiTH-ISY-I-1329.ps.Z`

For people at the control group in Linköping the two m-files are under the directory `/usr/local/matlab/local/`

```
function gode(F,ax,T,tol)

%GODE   gode(F,ax,T,tol)
%
%       Graphical 'interface' to ode45 for two-dimensional equations.
%       Click the mouse on initial value wished for.
%
%       Allows for simulations both in forward and backward time:
%       Forward time - left button, backward time - middle button
%
%       Clicking the mouse outside the specified y-axis interval
%       but inside the x-axis interval means either
%       1) "use the previous initial value and toggle time direction"
%          (click below the y-interval)
%       or
%       2) "use the previous final value as new initial value (same time
%           direction)"
%          (click above the y-interval)
%
%       Clicking outside the x-axis interval prompts for a new
%       value of T and then proceeds.
%
%       F=name of m-file describing the DE (see ODE45)
%       ax=axis-vector
%            If ax is a scalar the axes will be [-ax ax -ax ax]
%       T=length of each simulation (default: 5)
%       tol=tolerance (default: 1e-6)
%
%       Stop by clicking the mouse outside both the y- and x-axis interval.

% K. Forsman 1991-04-03

if nargin<4, tol=1e-6; end;
if nargin==2, T=5; end;
if length(ax)==1, ax=[-ax ax -ax ax]; end;
if length(ax)~=4, error('Invalid axis choice.'); end;
hold off;
axis(ax);

plot(ax(1),ax(3));      %For a window to appear
hold on
[a,b,m]=ginput(1);
xflag=(a>ax(1)) & (a<ax(2));
```

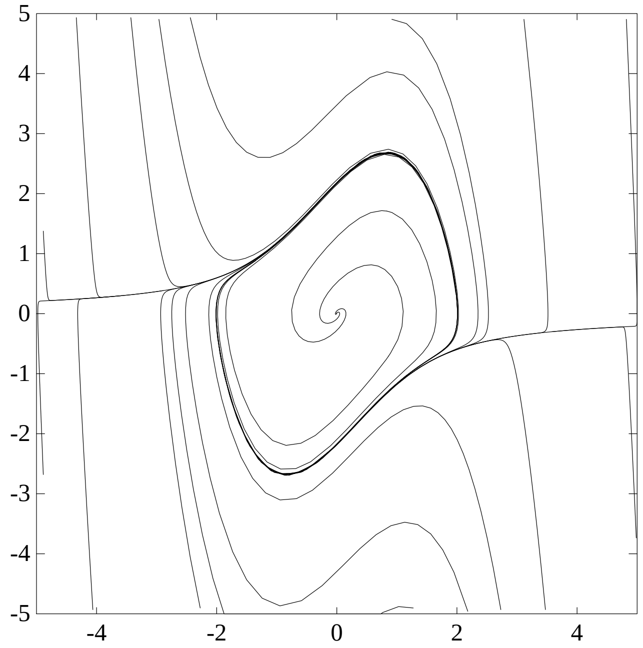

Figure 3: Phase plane for the van der Pol equation.

Example 3.3 The famous van der Pol equation (see e.g. [2, 5, 6, 7]) is

$$\ddot{y} - a(1 - y^2)\dot{y} + y = 0 \tag{4}$$

The m-file `defvdp` defines this equation in normal form (called state space form in control theory). In the simulation a was chosen equal to 1.

```
function xprime=defvdp(t,x)

a=1;

xprime(1)=x(2);
xprime(2)=a*(1-x(1)*x(1))*x(2)-x(1);

xprime=xprime';
```

The command `gode('defvdp',5);` was used to plot the phase plane in figure 3. □

4 The M-files: availability and listing

Here follows a complete listing of the two m-files involved: `gode.m` and `ode45bw.m`. The latter is a trivial modification of the original function `ode45` which is called from within `gode`. The file `ode45bw.m` has to be in your search path if you want to be able to simulate backwards in time using `gode`.

The two files are available by anonymous ftp on the address

`130.236.24.1` (`joakim.isy.liu.se`)

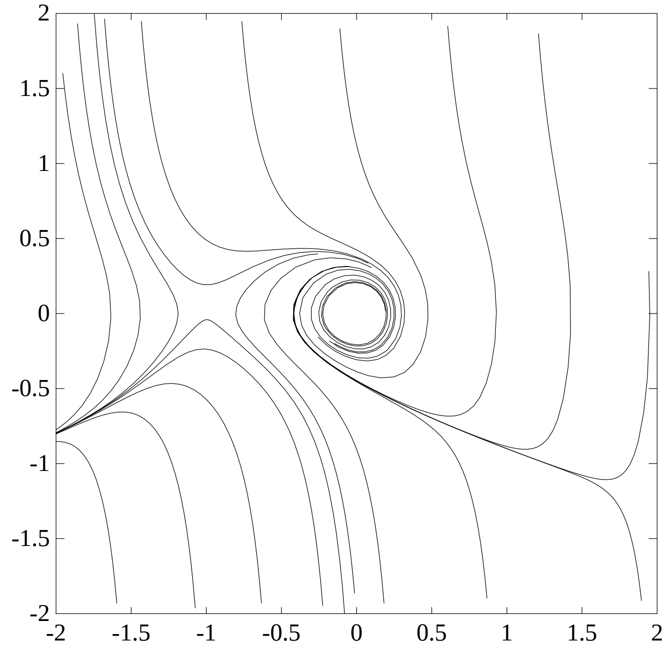

Figure 1: Phase plane for the differential equation (2).

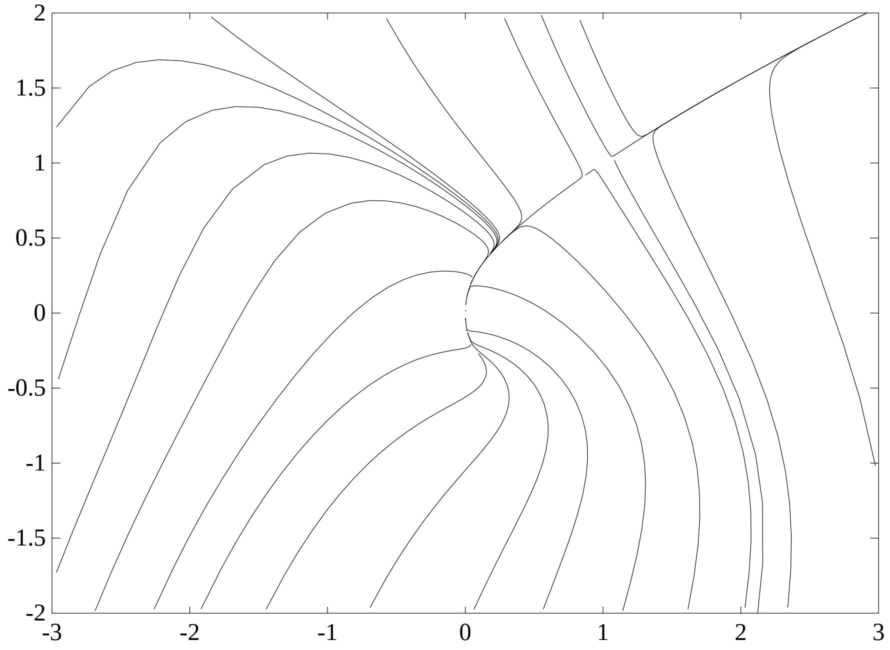

Figure 2: Phase plane for the differential equation (3).

3 Examples

In this section I give some examples of how to use `gode`.

Example 3.1 The following session produced the plot displayed in figure 1.

```
>> axis('square')
>> gode('def11_1',[-2 2 -2 2]);
Singularity likely at t = 1.717
Singularity likely at t = 2.805
>>
```

Here, the file `def11_1` contains the following code:

```
function xprime=def11_1(t,x)

xprime(1)=x(2);
xprime(2)=(0.1-10*x(2)*x(2)/3)*x(2)-x(1)-x(1)*x(1);

xprime=xprime';
```

which corresponds to the differential equation

$$\ddot{y} - (0.1 - 10\dot{y}^2/3)\dot{y} + y + y^2 \; = \; 0 \tag{2}$$

This example is exercise 11.1 in [1]. □

Example 3.2 The command `>> gode('def3_3',[-3 3 -2 2]);` simulates the differential equation defined by the file `def3_3`:

```
function xprime=def3_3(t,x)

xprime(1)=-x(1)+x(2)*x(2);
xprime(2)=-x(2)*x(2)*x(2)+x(1)*x(1);

xprime=xprime';
```

That is

$$\begin{aligned} \dot{x} &= -x + y^2 \\ \dot{y} &= -y^3 + x^2 \end{aligned} \tag{3}$$

This example is identical with example 3.3 in Verhulst [6], which is, incidentally, misprinted in the book.

Close to the origin, the simulation time `T` has to be larger than 5. See figure 2. □

which corresponds to the system

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_2^2 - x_1 \end{bmatrix} \tag{1}$$

If $n = 2$ above, a numerical solution may be plotted in phase space. Phase plane analysis is an important tool in the study of nonlinear ODE:s, and almost all textbooks devote large sections to it [2, 6, 7].

This report describes a simple function that allows the user to operate `ode45` graphically for second order ODE:s. After the user has chosen the initial value for a graphically by clicking with the mouse, the equation in question is simulated for a certain time interval and the solution is plotted. The function is called `gode` (graphical ODE-solver).

2 The Graphical Interface

The function `gode` is a "graphical interface" to `ode`. It is intended to be a tool for interactively constructing phase planes in a simple manner. Here follows a description of the function:

Syntax: `gode(F,ax,T,tol)`

Input arguments:

`F`	name of m-file describing the DE
`ax`	axis-vector (If `ax` is a scalar the axes will be `[-ax ax -ax ax]`)
`T`	length of each simulation (default: 5)
`tol`	tolerance (default: 1e-6)

`gode` starts by displaying an empty plane as specified by the argument `ax`. It waits for the user to click the mouse on a point in the phase plane - the given point will be taken as the intial value in the simulation.

`gode` supports both forward and backward time simulations: forward time - use left button, backward time - use middle button

Clicking the mouse outside the specified y-axis interval but inside the x-axis interval means either

1. "use the previous initial value and toggle time direction" (click below the y-interval) or
2. "use the previous final value as new initial value (same time direction)" (click above the y-interval)

Clicking outside the x-axis interval prompts for a new value of T and then proceeds.

Stop by clicking the mouse outside both the y- and x-axis interval.

`gode` gives no output (other than the plots).

The advantage with both forward and backward time simulation is that the user may choose an arbitrary point in phase space and then plot the trajectory passing through that point, instead of just the trajectory *emanating* from the point.

A MATLAB Function for Plotting Phase Planes Interactively

Krister Forsman, Dept. of Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden

Email: krister@isy.liu.se

1992-02-13

Abstract

A MATLAB-function serving as a sort of graphical interface to the MATLAB ODE-solver `ode45` is presented. The main use of it is to facilitate phase plane analysis of nonlinear 2-D systems via simulation. The numerical integration of the system is mouse-operated via the `ginput`-function. After each choice of initial value a simulation is made and plotted in the phase plane.

1 The ODE-solvers in MATLAB

Consider the nonlinear differential equation

$$\dot{x} = f(t, x), \quad x(0) = x_0$$

where $x \in \mathbb{R}^n$ and $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$. MATLAB [4] version 3.5i provides two numerical ODE solvers for such nonlinear systems: `ode23` and `ode45`. Both use Runge-Kutta formulas of different order. For a comprehensive account of numerical methods for solving ODE:s, consult Hairer et al. [3]. The syntax of `ode23` and `ode45` is, somewhat simplified,

```
[t,x] = ode45('xprime', t0, tf, x0)
```

where `xprime` is an m-file containing the right hand side of the differential equation, `t0` the initial time, `tf` the final time and `x0` the initial value. An example of a `xprime` m-file is:

```
function xprime=ex1(t,x)

xprime(1)=x(2);
xprime(2)=-x(2)^2-x(1);

xprime=xprime';
```