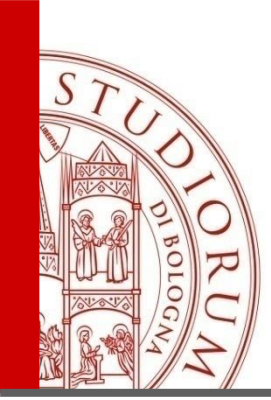


Ordinary Differential Equations ODE

Initial Value Problems

Examples

Symbolic solution of ODEs with MATLAB

- Sometimes it is possible to determine the analytical solution of a differential equation. This function is called symbolic solution and is obtained by the function **dsolve** of the **Symbolic Math Toolbox.**
- Functionalities of the Symbolic Math Toolbox:
  - symbolic algebra;
  - resolution of algebraic and transcendental equations;
  - calculation of integrals, derivatives, series, etc. ...;
  - symbolic methods for solving ODE
  - ..

The function dsolve

s=dsolve('eqn')	solves the ordinary differential equation eqn , symbolic equation, represented by a string , order >=1
Y=dsolve('eq1', 'eq2',…)	solves the system of ODE and returns a structure array that contains the solutions.
Y=dsolve('eq1', 'cond1', 'cond2',…)	solves the ordinary differential equation eqn1 with the initial or boundary condition cond, order >1
Y=dsolve('eq1', 'eq2',…, 'cond1', 'cond2',…)	solves the system of ordinary differential equations eqns with the initial or boundary conditions conds.

Examples

1) Solve the ODE: $\frac{dy}{dt} + 2y = 12$

```
>> dsolve('Dy+2*y=12')
ans =
6+exp(-2*t)*C1
```

2) Solve the ODE: $\frac{d^2y}{dt^2} = c^2 y$

```
>> dsolve('D2y=c^2*y')
ans =
C1*sinh(c*t)+C2*cosh(c*t)
```

N.B. you can use a string with the letter capital D to indicate the derivative and requires that the entire equation appear in single quotes.
D2 for second derivatives, and so on..

The solution is found in function of arbitrary constants C1, C2

Examples

3) Solve the system of ODE:

$$\frac{dx}{dt} = 3x + 4y$$
$$\frac{dy}{dt} = -4x + 3y$$

```
>> [x,y]=dsolve('Dx=3*x+4*y','Dy=-4*x+3*y')
x = exp(3*t)*(cos(4*t)*C1+sin(4*t)*C2)
y = -exp(3*t)*(sin(4*t)*C1-cos(4*t)*C2)
```

4) Solve the ODE with init. Cond.:

$$\frac{dy}{dt} = \sin(bt)$$
$$y(0) = 0$$

```
>> dsolve('Dy=sin(b*t)','y(0)=0')
ans =
(-cos(b*t)+1)/b
```

Examples

5) Solve the system of ODE:

$$\frac{dx}{dt} = 3x + 4y$$
$$\frac{dy}{dt} = -4x + 3y$$

$$x(0) = 0$$
$$y(0) = 1$$

```
>> [x,y]=dsolve('Dx=3*x+4*y','Dy=-4*x+3*y','x(0)=0','y(0)=1')
x = exp(3*t)*(-cos(4*t)*sin(4)/cos(4)+sin(4*t))
y = -exp(3*t)*(-sin(4*t)*sin(4)/cos(4)-cos(4*t))
```

Examples

6) Solve the nonlinear ODE:

$$\frac{dy}{dt} = 4 - y^2 \qquad y(0) = 1$$

```
>> dsolve('Dy=4-y^2','y(0)=1')
ans =
(2*exp(4*t+log(-3))+2)/(-1+exp(4*t+log(-3)))

>> simple(ans)  %%apply algebraic simplifications,
ans =
(-6*exp(4*t)+2)/(-1-3*exp(4*t))
```

Examples

7) Solve the ODE with init.cond: $\frac{dy}{dt} = x + y \quad y(0) = 2$

>> sol=dsolve('Dy=x+y',’y(0)=2’,’x’)

sol =

-x-1+3*exp(x)

The independent variable may be changed from 't' to some other symbolic variable by including that variable as the last input argument.

Plot the obtained solution in the range [-1,2]

>> ezplot(sol,[-1,2])

ODE of order > 1

$$\begin{cases} x^{(m)} = f(t, x, x', .., x^{(m-1)}, y, .., y^{(n-1)}) \\ y^{(m)} = g(t, x, x', .., x^{(m-1)}, y, .., y^{(n-1)}) \end{cases}$$

Choose the state variables:

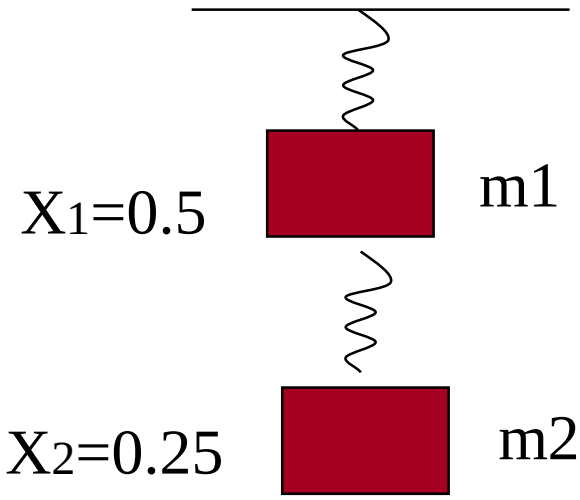
$$x_1 = x, x_2 = x', .., x_m = x^{(m-1)},$$

$$x_{m+1} = y, x_{m+2} = y', .., x_{m+n} = y^{(m-1)},$$

Then

$$\begin{cases} x_1' = x_2 \\ \vdots \\ x_m' = f(t, x_1, x_2, .., x_{m+n}) \\ x_{m+1}' = x_{m+2} \\ \vdots \\ x_{m+n}' = g(t, x_1, x_2, .., x_{m+n}) \end{cases}$$

Example: mass spring system



The vertical displacement of the two masses suspended in series by springs of spring constant k1, k2, is given by Hooke's law as a system of two second-order ODE:

$$m_1 \frac{d^2 x_1}{dt^2} - k_2(x_2 - x_1) + k_1 x_1 = 0$$

$$m_2 \frac{d^2 x_2}{dt^2} + k_2(x_2 - x_1) = 0$$

Example: mass spring system

Substitution of variables to obtain a system of four ODE of the first order:

$$u_1 = x_1$$

$$u_3 = x_2$$

$$\begin{cases} u_1' = u_2 \\ u_2' = +\dfrac{k_2}{m_1}(u_3 - u_1) - \dfrac{k_1}{m_1}u_1 \\ u_3' = u_4 \\ u_4' = -\dfrac{k_2}{m_2}(u_3 - u_1) \end{cases}$$

Initial conditions: $u_1(0) = a_1 \quad u_2(0) = a_2 \quad u_3(0) = a_3 \quad u_4(0) = a_4$

Example: mass spring system

- Step 1: Write the ODE function:

$$\begin{cases} u_1' = u_2 \\ u_2' = +\dfrac{k_2}{m_1}(u_3 - u_1) - \dfrac{k_1}{m_1}u_1 \\ u_3' = u_4 \\ u_4' = -\dfrac{k_2}{m_2}(u_3 - u_1) \end{cases}$$

```
function dydt=mass_spring(t,y)
global k1 m1 k2 m2
dydt=[y(2);
    (-k1*y(1)+k2*(y(3)-y(1)))/m1;
    y(4);
    -k2*(y(3)-y(1))/m2];
```

script **exampleODE.m**

Step 2: Define the problem data:

```
a=0; b=5; tspan=[a b];          % iintegral interval
y0=[0.5 0 0.25 0];              % initial conditions
global k1 m1 k2 m2
k1=100; m1=10; k2=120; m2=2; %  in Kg, meters, Newton
```

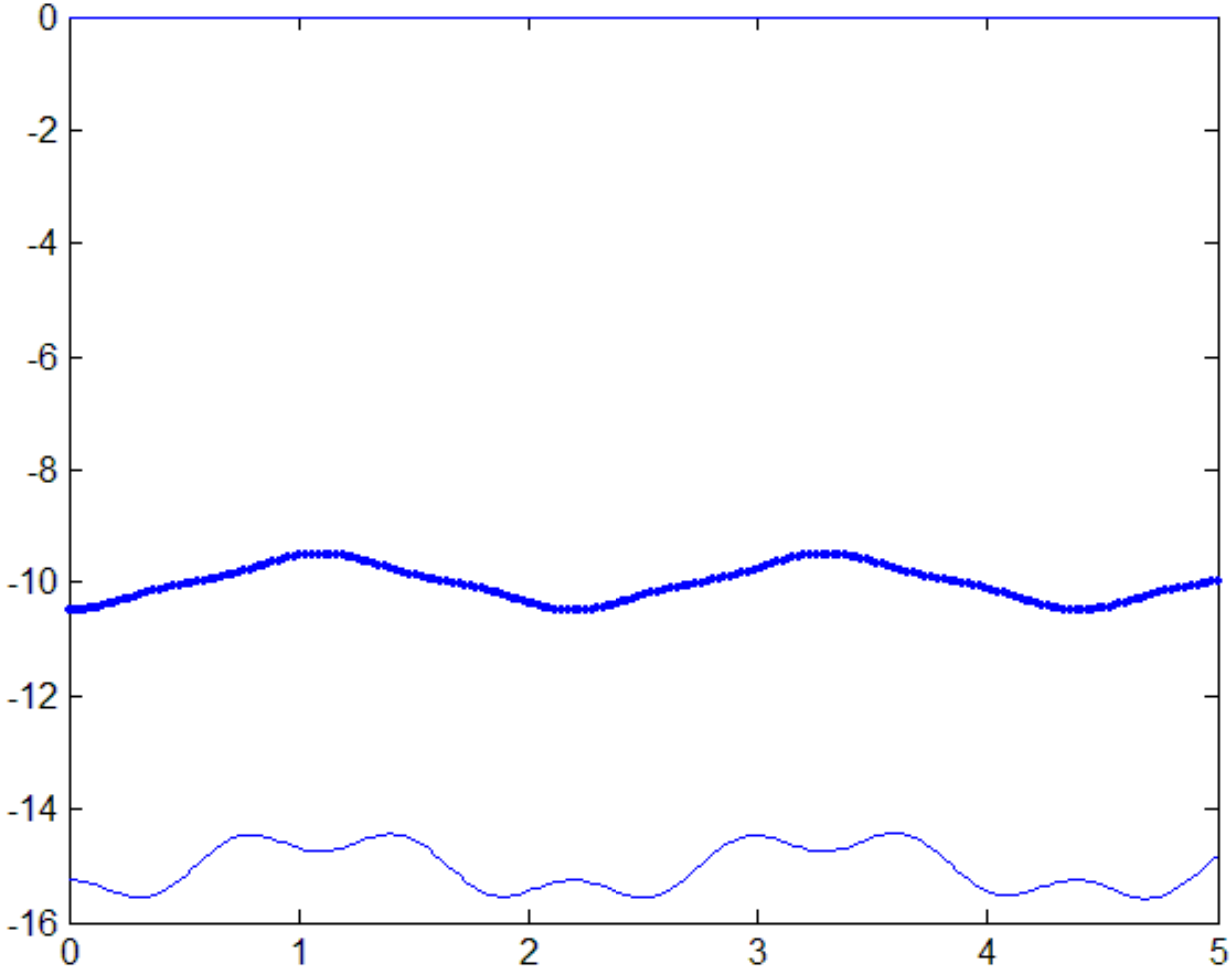
Step 3: System solver:

```
[t,y]=ode45(@mass_spring,tspan,y0);
```

Step 4: Visualize the solution:

```
% down direction , thus -y
% the position depends on  the spring length r1 and r2:
r1=10; r2=15;
plot(t,-r1-y(:,1),'.-');
hold on
plot(t,-r2-y(:,3),'-');
plot([a b],[0 0]);
```

Example: mass spring system



MATLAB Solvers

Category	Function	Description	
Ordinary differential equation solvers	ode45	Nonstiff differential equations, medium order method.	Runge Kutta 1 step, explicit variable step
	ode23	Nonstiff differential equations, low order method.	Runge Kutta 1 step, explicit variable step
	ode113	Nonstiff differential equations, variable order method.	Adams-Bashforth Moulton, if f evaluation is expensive
	ode15s	Stiff differential equations and DAEs, variable order method.	Implicit Multistep, variable step
	ode23s	Stiff differential equations, low order method.	Implicit 1 step, variable step
	ode23t	Moderately stiff differential equations and DAEs, trapezoidal rule.	
	ode23tb	Stiff differential equations, low order method.	Equivalent to implicit Runge Kutta
ODE option handling	odeset	Create/alter ODE OPTIONS structure.	
	odeget	Get ODE OPTIONS parameters.	
ODE output functions	odeplot	Time series plots.	
	odephas2	Two-dimensional phase plane plots.	
	odephas3	Three-dimensional phase plane plots.	
	odeprint	Print to command window.	

Event in MATLAB

Typically, the ODE solvers in MATLAB terminate after solving the ODE over a specified domain of the independent variable $[t_0, t_{final}]$.

In applications, however, we often would like to stop the solution at a particular value of the dependent variable (for example, when an object fired from the ground reaches its maximum height or when a population crosses some threshold value).

Since we do not know the appropriate final time, find the solution y(t) and a final time t* such that

$$y' = f(t, y)$$

$$y(t_0) = y_0,$$

$$g(t^*, y(t^*)) = 0$$

Event in MATLAB

EXAMPLE:

body in free fall under the force of gravity.
When does it hit the ground?

$$y'' = -1 + (y')^2$$

$$y(0) = 1, \quad y'(0) = 0,$$

For which t value y(t)=0?

```
function ydot = f(t,y)
ydot = [y(2); -1+y(2).^2];
```

Event in MATLAB

```
function [value,isterminal,direction] = gstop(t,y)
value = y(1);          % value of the event function
isterminal = 1;        %  the solver stops at a zero of this event  (gs=0)
direction= [ ]; % [ ] 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.
```

```
opts = odeset('events',@gstop);
[t,y,tfinal] = ode23(@f,[0 Inf],[1; 0],opts);
[t,y,tfinal,Ye,Ie] = ode23(@f,[0 Inf],[1; 0],opts);
```

- tfinal: A column vector of times at which events occur
- Ye: Solution values corresponding to these times
- Ie: Indices into the vector returned by the events function. The values indicate which event the solver detected.

The Apollo spacecraft orbit

- The ODE describing the motion of a body in orbit around two other bodies much heavier. An example would be the **Apollo** spacecraft in orbit around the **earth** and the **moon.**
- The three bodies determine a plane in space. The x-axis is the straight line joining the two heavy bodies , the origin is placed in their center of gravity and the distance between them is taken as the unit of measure.

- the coordinate systems moves in agreement with the moon that revolves around the earth.
- It is assumed that the third body , the Apollo , has negligible mass compared to the first two , and that his position as a function of time (x(t) , y(t)) .

The Apollo spacecraft orbit

- The following differential equation describing the motion of the Apollo spacecraft in orbit around the earth and the moon

$$x'' = 2y' + x - \frac{\rho^*(x+\rho)}{r_1^3} - \frac{\rho\ \ (x-\rho^*)}{r_2^3}$$

$$y'' = -2x' + y - \frac{\rho^* y}{r_1^3} - \frac{\rho\ \ y}{r_2^3}$$

$$r_1 = ((x+\rho)^2 + y^2)^{1/2}, \quad r_2 = ((x-\rho^*)^2 + y^2)^{1/2}$$

ρ=1/82.4, ρ*=1-ρ

If ρ is the ratio of the mass of the moon and the earth, then the moon and the earth are located at coordinates (1- ρ , 0) and (- ρ , 0), respectively,

the initial values

x(0) = 1.2 x'(0) = 0, y(0) = 0 y'(0) = -1.04935751

give rise to a periodic solution of period T = 6.19216933

The Apollo spacecraft orbit

- Solve the trajectory and make the trajectory plot
- Define a set of variables

$$x_1 = x,\ x_2 = y,\ x_3 = x',\ x_4 = y'$$

- After conversion we get the following set of first order ODEs:

$$\begin{cases} x_1' = x_3 \\ x_3' = x_4 \\ x_2' = 2x_4 + x_1 - \rho^*(x_1 + \rho)/r_1^3 - \rho(x_1 - \rho^*)/r_2^3 \\ x_4' = -2x_3 + x_2 - \rho^* x_2 / r_1^3 - \rho x_2 / r_2^3 \end{cases}$$

The Apollo spacecraft orbit

Prepare the ODE RHSF in MATLAB

```
function dydt = apolloeq( t,y,y0)
global ro ros

r1=sqrt((y(1)+ro)^2+y(2)^2);
r2=sqrt((y(1)-ros)^2+y(2)^2);
dydt=[y(3);
      y(4);
      2*y(4)+y(1)-(ros*(y(1)+ro))/r1^3- …
            (ro*(y(1)-ros))/r2^3;
      -2*y(3)+y(2)-(ros*y(2))/r1^3-(ro*y(2))/r2^3];
```

Solve the ODE system

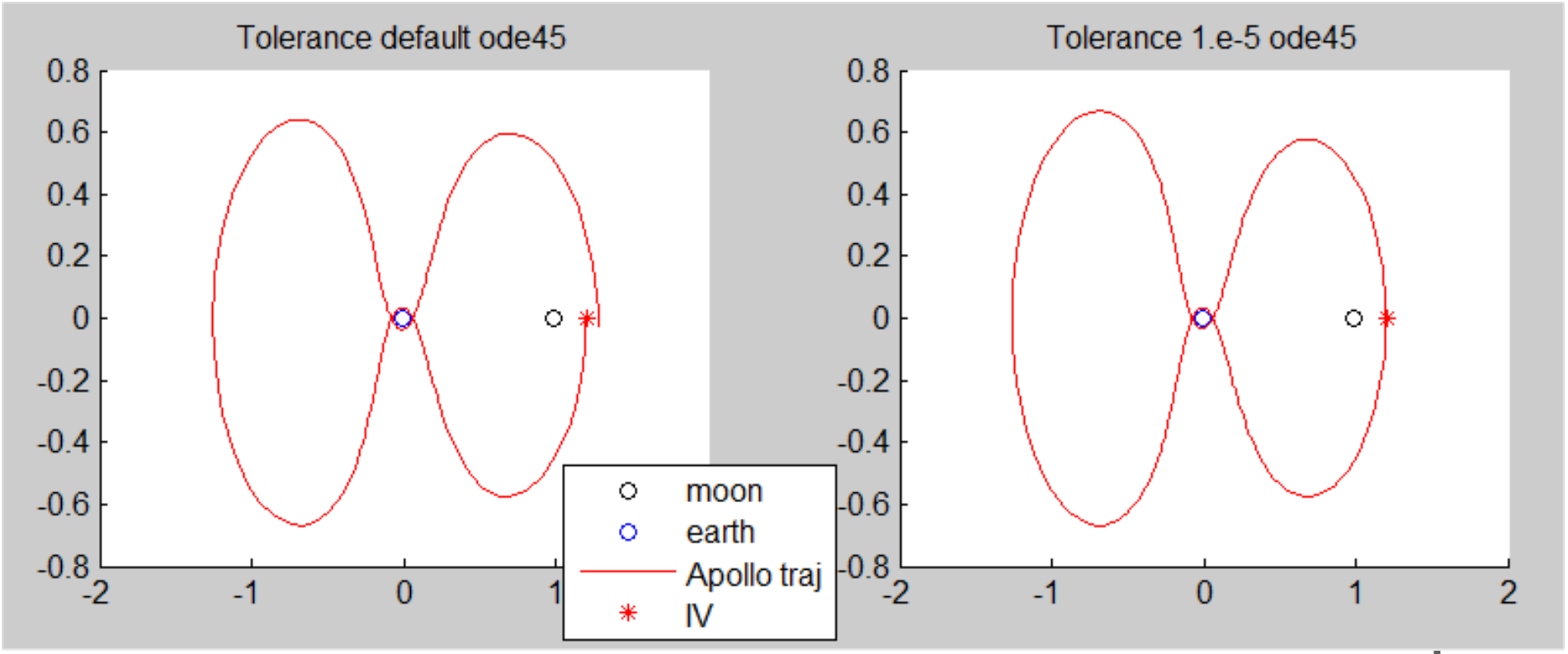
```
ro=1/82.45;
ros=1-ro;

y0=[1.2 0 0 -1.04935751];
% Default tolerance
[t,y] = ode45(@apolloeq,[0 7],y0);

%Try a different precision for ODE solution
options=odeset('RelTol',1e-5);
[t,y]=ode45(@apolloeq,[0 7],y0,options);

figure(1)
plot(y(:,1),y(:,2),'r')
```

We observe that with the default values of the orbit is not perfectly periodic, moderately stringent tolerances are necessary to reproduce the qualitative behavior of the orbit.



Using Advanced EVENT

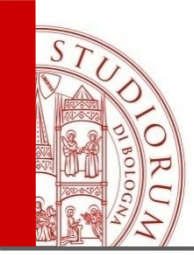
- Determine the time at which the Apollo return back as close as possible to the starting point y0 and starts to move away, then terminates the integration.

- Determine the time at which the Apollo is as far as possible from the initial point and begins to approach again.

Event

```
function [value,isterminal,direction] = events(t,y,y0)
  % Event function -- y0 shared with the outer function.
  % Locate the time when the object returns closest to the initial point y0
  % and starts to move away, and stop integration.  Also locate the time when
  % the object is farthest from the initial point y0 and starts to move closer.
  %
  % The current distance of the body is
  %   DSQ = (y(1)-y0(1))^2 + (y(2)-y0(2))^2 = <y(1:2)-y0(1:2),y(1:2)-y0(1:2)>
  %
  % A local minimum of DSQ occurs when d/dt DSQ crosses zero heading in
  % the positive direction.  We can compute d/dt DSQ as
  %
  %   d/dt DSQ = 2*(y(1:2)-y0(1:2))'*dy(1:2)/dt = 2*(y(1:2)-y0)'*y(3:4)

    dDSQdt = 2 * ((y(1:2)-y0(1:2))' * y(3:4));
    value = [dDSQdt; dDSQdt];
    isterminal = [1;  0];          % stop at local minimum
    direction  = [1; -1];          % [local minimum, local maximum]
  end
```

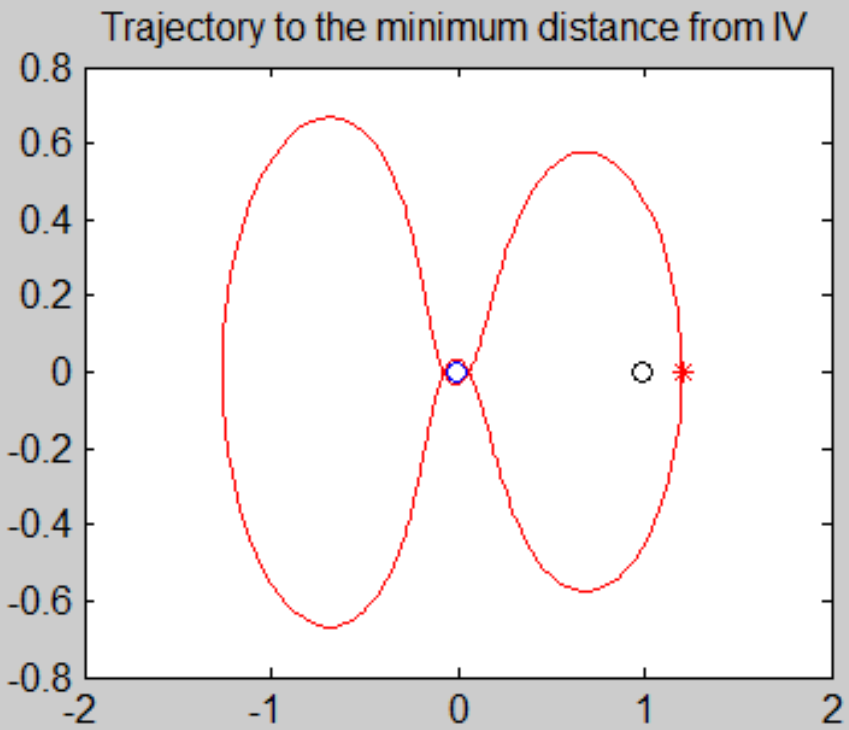


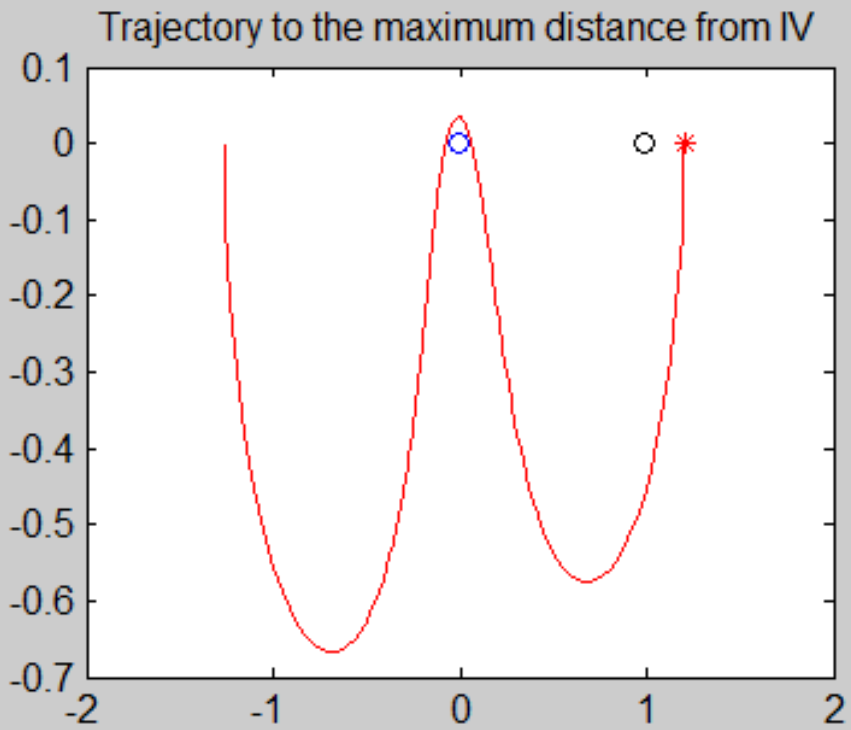
This is an example of event location where the ability to specify the direction of the zero crossing is critical.

Both the point of return to the initial point and the point of maximum distance have the same event function value, and the direction of the crossing is used to distinguish them.

```
opts=odeset('events',@events,'Reltol',1e-
                              5,'OutputFcn',@odephas2);

[t,x,te,xe]=ode45(@apolloeq,[0 inf],y0,opts,y0);
```

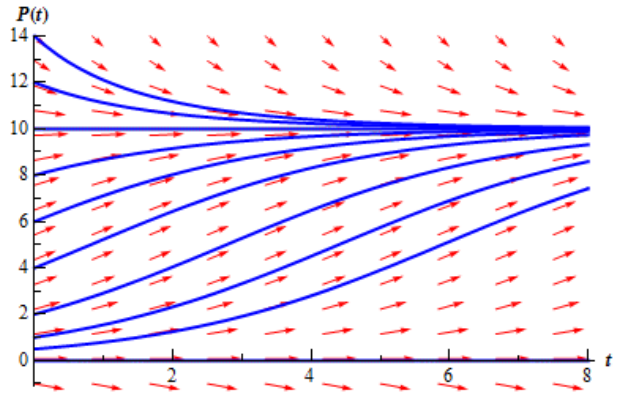

Trajectory to the minimum distance from IV
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-2
-1
0
1
2


Trajectory to the maximum distance from IV
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-2
-1
0
1
2

Equilibrium Solutions

- The growth rate of a population needs to depend on the population itself. Once a population reaches a certain point the growth rate will start reduce, often drastically. A realistic model of a population growth is given by the **logistic growth equation**.

$$P'(t) = \frac{1}{2}\left(1 - \frac{P(t)}{10}\right)P(t)$$



10 = saturation level

½ = intrinsic growth rate

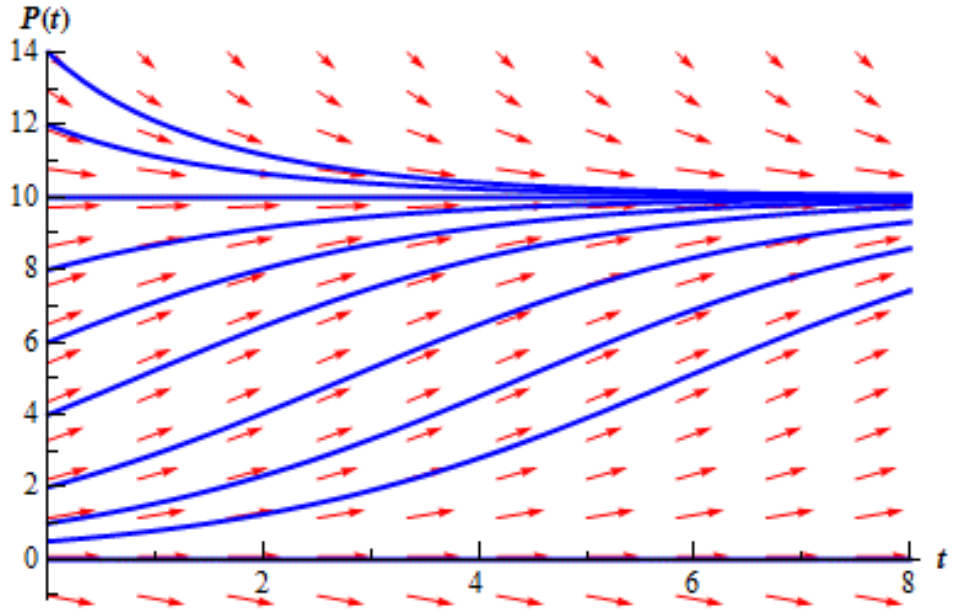
- Populations can't just grow forever without bound. Eventually the population will reach such a size that the resources of an area are no longer able to sustain the population and the population growth will start to slow as it comes closer to this threshold. Also, if you start off with a population greater than what an area can sustain there will actually be a die off until we get near to this threshold.

Example 1

$$P'(t) = \frac{1}{2}(1 - \frac{P(t)}{10})P(t)$$

compute P such that

$$\frac{1}{2}(1 - \frac{P}{10})P = 0$$



For our logistics equation, equilibrium points:

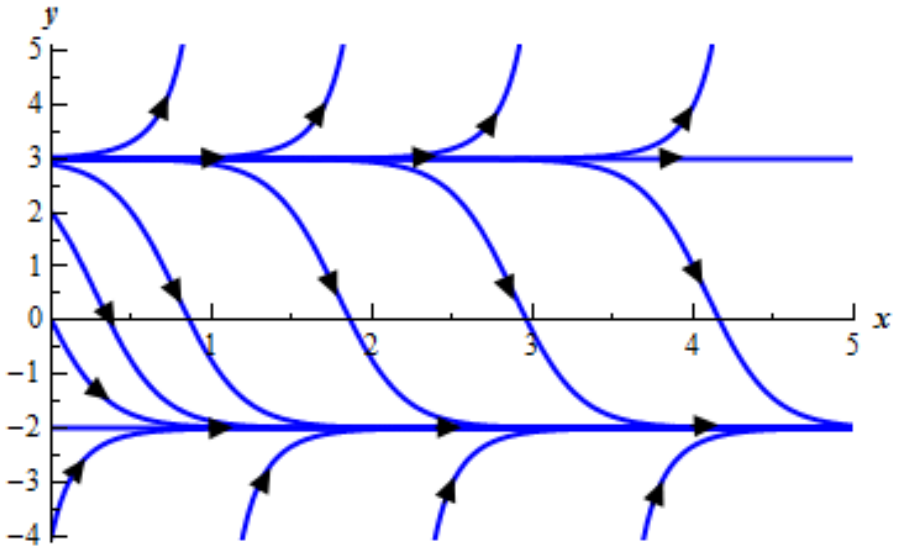
- $P = 0$ is an unstable equilibrium solution
- $P = 10$ is an asymptotically stable equilibrium solution.

Example 2

$$y' = y^2 - y - 6$$

Equilibrium solutions

$$y^2 - y - 6 = (y-3)(y+2) = 0$$



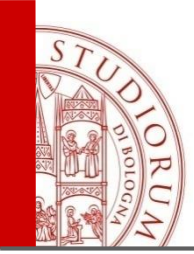
- Equilibrium points y=3,y=-2
- From this sketch it appears that solutions that start "near" *y*=-2 all move towards it as *t* increases and so *y* = -2 is an **asymptotically stable equilibrium solution** and solutions that start "near" *y* = 3 all move away from it as *t* increases and so *y*= 3 is an **unstable equilibrium solution**.

Classification of the equilibrium equations:

If solutions start “near” an equilibrium solution will they move away from the equilibrium solution or towards the equilibrium solution?

- Equilibrium solutions in which solutions that start “near” them move away from the equilibrium solution are called **unstable equilibrium points** or **unstable equilibrium solutions**.

- Equilibrium solutions in which solutions that start “near” them move toward the equilibrium solution are called **asymptotically stable equilibrium points** or **asymptotically stable equilibrium solutions**.



Stiff ODE Problems

For a stiff problem, solutions can change on a time scale that is very short compared to the interval of integration, but the solution of interest changes on a much longer time scale.

Methods not designed for stiff problems are ineffective on intervals where the solution changes slowly because they use time steps small enough to resolve the fastest possible change.

The graph of the components highlights the different time scales characteristics of each component, from which the stiff nature of the problem.

The Van der Pol Equation

The Van der Pol Equation non linear stiff equation

Proposed in 1920 and used in the study of the so-called vacuum tubes, such as cathodic tubes in television sets or magnetrons in microwave ovens.

$$\begin{cases} y''-\mu\left(1-y^2\right)y'+y=0 \\ y(0)=2 \qquad y'(0)=0 \end{cases} \qquad \mu \in \Re^+$$

damping of the system

To express this equation as a system of first-order ODE, introduce a variable y_2 such that $y_1' = y_2$. You can then express this system as

$$\begin{cases} y_1' = y_2 \\ y_2' = \mu\left(1-y_1^2\right)y_2 - y_1 \end{cases} \qquad \begin{cases} y_1(0)=2 \\ y_2(0)=0 \end{cases}$$

The Van der Pol Equation

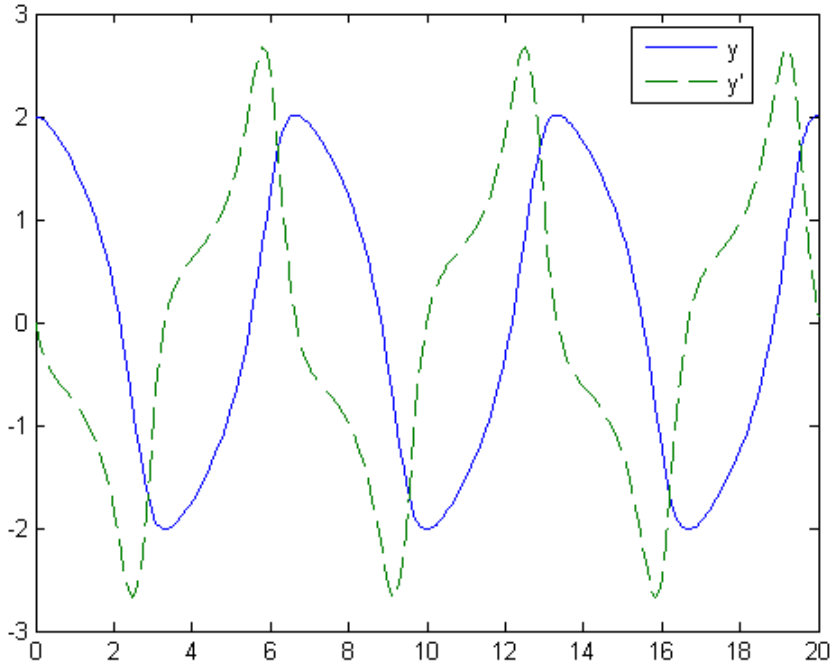
Let **μ=1** (weak damping) and implement the ODE function ***vdpol.m***

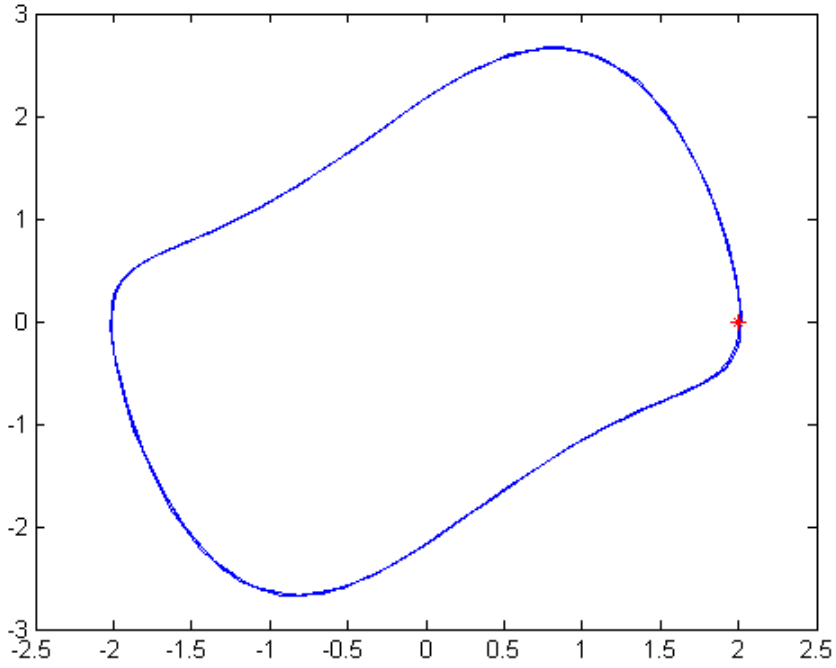
```
function yp=vdpol(t,y)
% Van der Pol Equation
mu=1;
yp=[y(2);mu*(1-y(1)^2)*y(2)-y(1)];
```

Solve by ***ode45*** in [0 20] and plot the components of the results

```
>> [t,y]=ode45(@vdpol,[0 20],[2 0]);
>>plot(t,y(:,1),'-',t,y(:,2),'—');
```

The Van der Pol Equation





plot in phase plane (y,y')

>>plot(y(:,1),y(:,2))
>>hold
>>plot(2,0,'*')

(***plane y,y'***, marker * denotes the initial condition)

The Van der Pol Equation

Let **μ=1000** (strong damping) and update ***vdpol1000.m***

```
function yp=vdpol1000(t,y)
% Van der Pol Equation
mu=1000;
yp=[y(2);mu*(1-y(1)^2)*y(2)-y(1)];
```

Solve by ***ode45*** in [0 1000] and plot the results

```
>> [t,y]=ode45(@vdpol1000,[0  1000],[2  0]);
```

The Van der Pol Equation

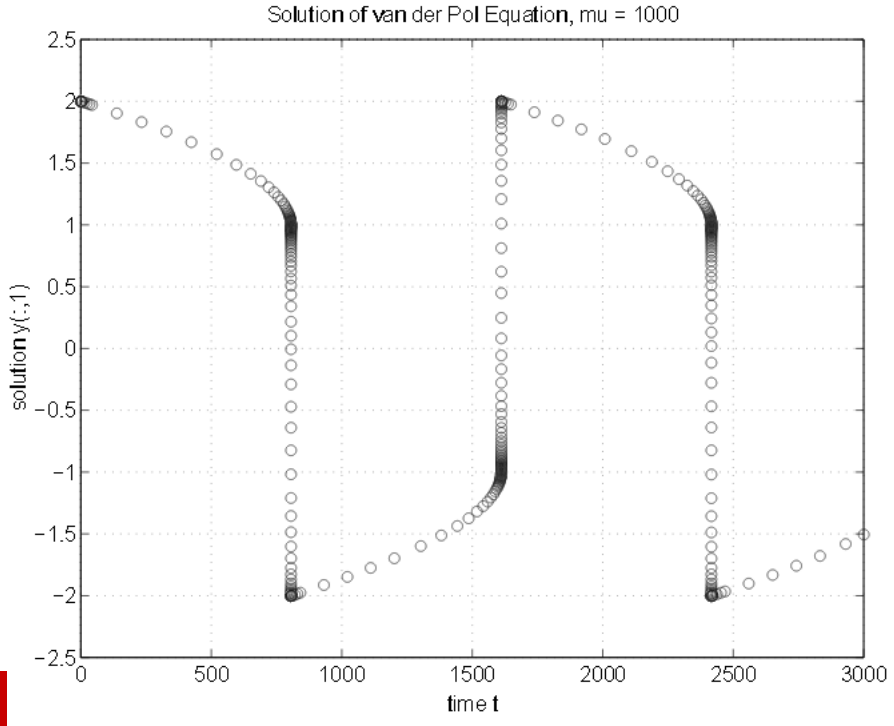
When μ is increased to 1000, the solution to the van der Pol equation changes dramatically and exhibits oscillation on a much longer time scale. Approximating the solution of the initial value problem becomes a more difficult task. Because this particular problem is stiff, a nonstiff solver such as ode45 is so inefficient that it is impractical.

- The numerical method is affected by this imbalance in the solution scales and pays the price in terms of choosing the integration step which in some places is very stringent.
- In this case, it is appropriate that the class of solvers, denoted by the suffix "s" specially designed for this kind of problems,
- For examples ***ode15s*** and ***ode23s***.

The Van der Pol Equation

```
[t,y] = ode15s(@vdp1000,[0 3000],[2; 0]);
plot(t,y(:,1),'o');
title('Solution of van der Pol Equation, mu = 1000');
xlabel('time t');
ylabel('solution y(:,1)');
```

- Now the computation times are significantly decreased.



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Serena Morigi

Dipartimento di Matematica

serena.morigi@unibo.it

http://www.dm.unibo.it/~morigi