Solving ODEs in Matlab

BP205

M.Tremont

1.30.2009

- Outline -

I. Defining an ODE function in an M-file

II. Solving first-order ODEs

III. Solving systems of first-order ODEs

IV. Solving higher order ODEs

Numerical methods are used to solve initial value problems where it is difficult to obtain exact solutions

- An **ODE** is an equation that contains one independent variable (e.g. time) and one or more derivatives with respect to that independent variable.
- In the time domain, ODEs are **initial-value problems**, so all the conditions are specified at the initial time t = 0.

$$\frac{dy}{dt} = \frac{t}{y} \qquad y(0) = 1$$

$$y(t) = \sqrt{t^2 + 1}$$

- Matlab has several different functions (built-ins) for the numerical solution of ODEs. These solvers can be used with the following syntax:

```
[outputs] = function_handle(inputs)
[t,state] = solver(@dstate,tspan,ICs,options)
```

An array. The solution of the ODE (the values of the state at every time).

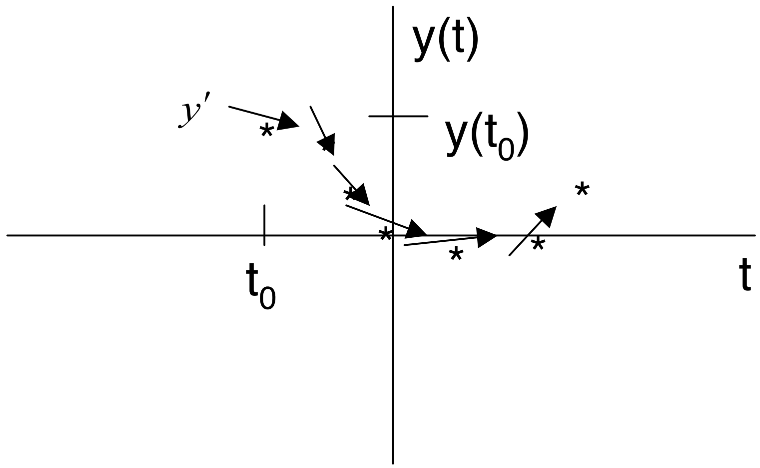
Matlab algorithm (e.g., `ode45`, `ode23`)

Handle for function containing the derivatives

Vector that specifiecs the interval of the solution (e.g., `[t0:5:tf]`)

A vector of the initial conditions for the system (row or column)

What are we doing when numerically solving ODE's?



We know t_0 and $y(t_0)$
and we know the slope of
$y(t)$, $dy/dt = f(t,y)$.

We don't know $y(t)$ for any
values of t other than t_0.

Integrators compute nearby value of *y(t)* using what we already know and repeat.

Higher order numerical methods reduce error at the cost of speed:

- **Euler's Method** - 1st order expansion
- **Midpoint method** - 2nd order expansion
- **Runge-Kutta** - 4th order expansion

Solver	Accuracy	Description
`ode45`	Medium	This should be the first solver you try
`ode23`	Low	Less accurate than `ode45`
`ode113`	Low to high	For computationally intensive problems
`ode15s`	Low to medium	Use if `ode45` fails because the problem is stiff*

Runge-Kutta (4,5) formula

*No precise definition of stiffness, but the main idea is that the equation includes some terms that can lead to rapid variation in the solution.

Defining an ODE function in an M-file

```
[t,state] = ode45(@dstate,tspan,ICs,options)
```

1. **Define `tspan`, `ICs` and `options` in one file (e.g. `call_dstate.m`), which sets up `ode45`**

2. **Define your constants and derivatives in another file (e.g. `dstate.m`) or as a function `dstate` within the call file**

3. **Run `call_dstate.m`**

4. **Analyze the results as a plot of `state` vs. `t`**

II. Solving first-order ODEs

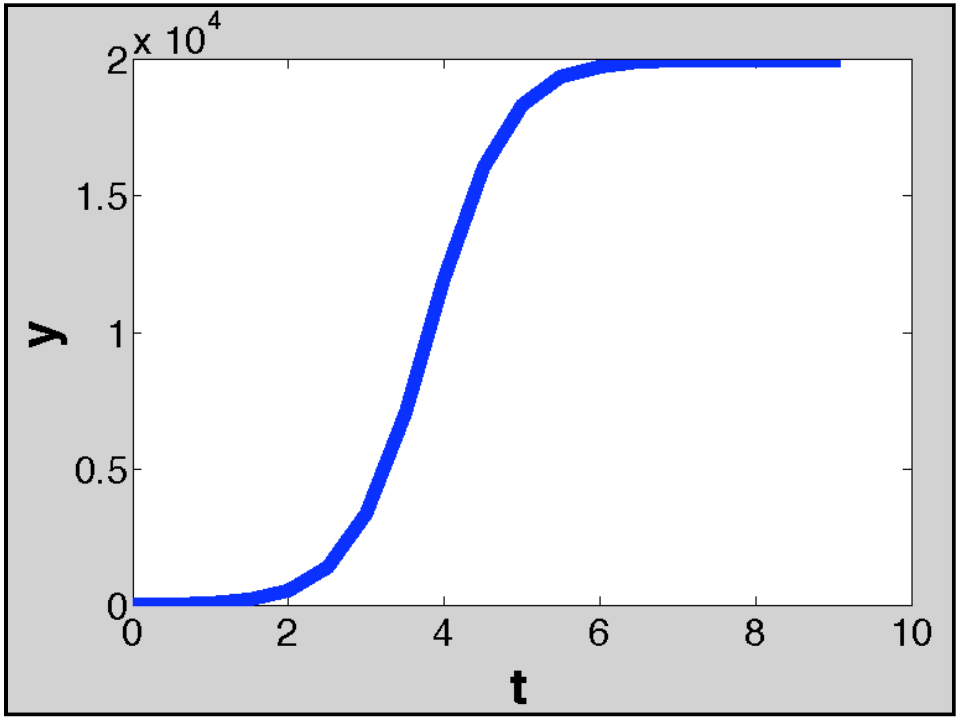
Example: $$\frac{dy}{dt} = y'(t) = \alpha y(t) - \gamma y(t)^2$$

$$y(0) = 10$$

```
function [t,y] = call_dstate()
   tspan = [0 9]; % set time interval
   y0 = 10; % set initial condition
   % dstate evaluates r.h.s. of the ode
   [t,y] = ode45(@dstate,tspan,y0);
   plot(t,y)
   disp([t,y]) % displays t and y(t)
            function dydt = dstate(t,y)
                  alpha=2; gamma=0.0001;
                  dydt = alpha*y-gamma*y^2;
            end
end
```

Save as `call_dstate.m` in some directory, and `cd` to that directory in the matlab GUI

Matlab `ode45`'s numerical solution



$$\frac{dy}{dt} = y'(t) = \alpha y(t) - \gamma y(t)^2$$

$$y(0) = 10$$

At t = 9, have we reached steady state?

$$\lim_{t->\infty} y(t) = \frac{\alpha}{\gamma} = 20{,}000$$

From the command line:

```
EDU>> [t, y] = call_dstate;
EDU>> steady_state = y(end)

steady_state =

   1.9999e+04
```

III. Solving systems of first-order ODEs

van der Pol equations in relaxation oscillation:

$$\frac{dy_1}{dt} = y_2 \qquad\qquad y_1(0) = 0$$

$$\frac{dy_2}{dt} = 1000(1 - y_1^2)y_2 - y_1 \qquad\qquad y_2(0) = 1$$

- This is a **system of ODEs** because we have more than one derivative with respect to our independent variable, time.

- This is a **stiff system** because the limit cycle has portions where the solution components change slowly alternating with regions of very sharp change - so we will need `ode15s`.

- This is a example from **mathworks**, a great resource @ mathworks.com or the software manual.

- This time we'll create separate files for the call function (`call_osc.m`) and the ode function (`osc.m`)

III. Solving systems of first-order ODEs

van der Pol equations in relaxation oscillation:

$$\frac{dy_1}{dt} = y_2 \qquad y_1(0) = 0$$

$$\frac{dy_2}{dt} = 1000(1 - y_1^2)y_2 - y_1 \qquad y_2(0) = 1$$

To simulate this system, create a function `osc` containing the equations.
Method 1: preallocate space in a column vector, and fill with derivative functions

```
function dydt = osc(t,y)
    dydt = zeros(2,1);    % this creates an empty column
    %vector that you can fill with your two derivatives:
    dydt(1) = y(2);
    dydt(2) = 1000*(1 - y(1)^2)*y(2) - y(1);
    %In this case, y(1) is y1 and y(2) is y2, and dydt(1)
    %is dy1/dt and dydt(2) is dy2/dt.
end
```

Save as `osc.m` in the same directory as before.

III. Solving systems of first-order ODEs

van der Pol equations in relaxation oscillation:

$$\frac{dy_1}{dt} = y_2 \qquad\qquad y_1(0) = 0$$

$$\frac{dy_2}{dt} = 1000(1 - y_1^2)y_2 - y_1 \qquad\qquad y_2(0) = 1$$

To simulate this system, create a function `osc` containing the equations.
Method 2: vectorize the differential functions

```
function dydt = osc(t,y)
    dydt = [y(2)
            1000*(1 - y(1)^2)*y(2) - y(1)];
    %Still y(1) is y1 and y(2) is y2, and dydt(1)
    %is dy1/dt and dydt(2) is dy2/dt.
end
```

Save as `osc.m` in the same directory as before.

III. Solving systems of first-order ODEs

van der Pol equations in relaxation oscillation:

$$\frac{dy_1}{dt} = y_2 \qquad y_1(0) = 2$$

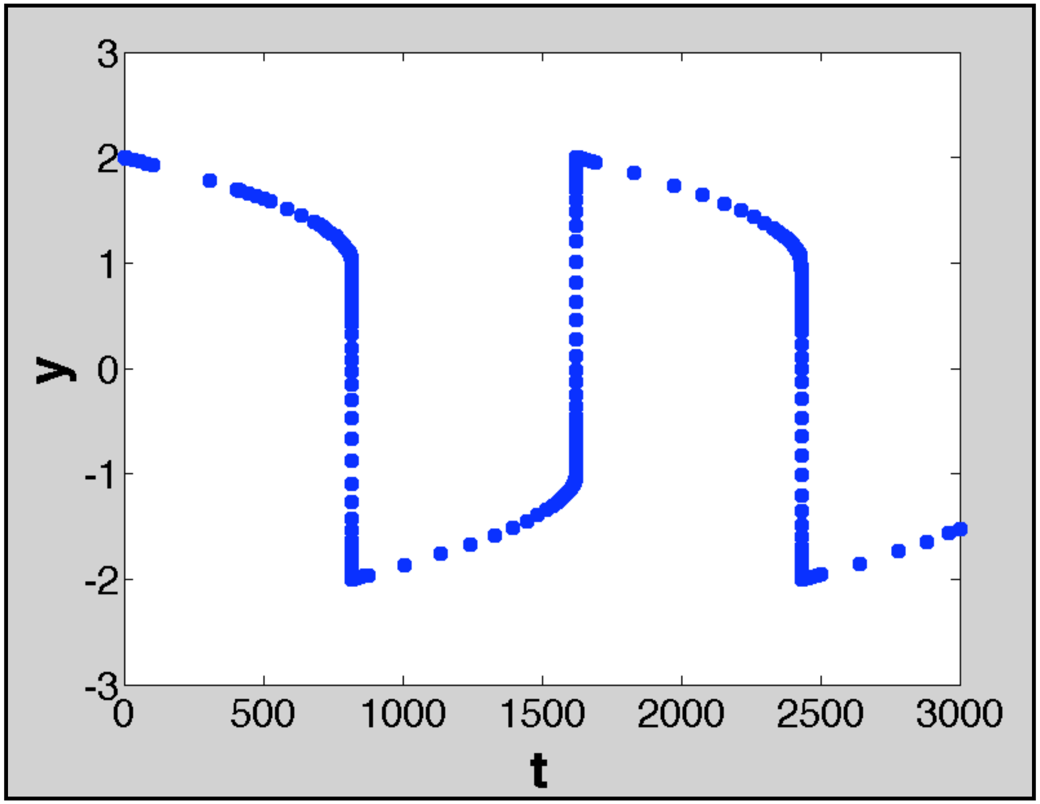
$$\frac{dy_2}{dt} = 1000(1 - y_1^2)y_2 - y_1 \qquad y_2(0) = 0$$

Now solve on a time interval from 0 to 3000 with the above initial conditions.
Create a scatter plot of y_1 with time.

```
function [T,Y] = call_osc()
    tspan = [0 3000];
    y1_0 = 2;
    y2_0 = 0;
    [T,Y] = ode15s(@osc,tspan,[y1_0 y2_0]);
    plot(T,Y(:,1),'o')
end
```

Save as `call_osc.m` in the same directory as before.

Plot of numerical solution



van der Pol equations in relaxation oscillation:

$$\frac{dy_1}{dt} = y_2$$

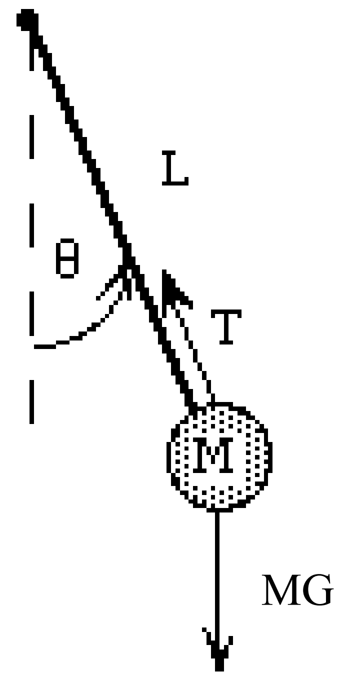
$$\frac{dy_2}{dt} = 1000(1 - y_1^2)y_2 - y_1$$

$$y_1(0) = 2$$

$$y_2(0) = 0$$

IV. Solving higher order ODEs

Simple pendulum:



$$ML\frac{d^2\theta}{dt^2} = -MG\sin\theta$$

$$\frac{d^2\theta}{dt^2} = -\frac{G}{L}\sin\theta$$

- **Second order non-linear ODE**
- **Convert the 2nd order ODE to standard form:**

$$z_1 = \theta, \quad z_2 = d\theta/dt$$

Non-linear pendulum function file

- G = 9.8 m/s
- L = 2 m
- Time 0 to 2π
- Initial θ = π/3
- Try ode23
- Plot θ with time

$$z_1 = \theta, \quad z_2 = d\theta / dt$$

$$\frac{dz_1}{dt} = z_2$$

$$\frac{dz_2}{dt} = -\frac{G}{L}\sin(z_1)$$

```
function [] = call_pend()
    tspan=[0 2*pi]; % set time interval
    z0=[pi/3,0];  % set initial conditions
    [t,z]=ode23(@pend,tspan,z0);
    plot(t,z(:,1))
function dzdt = pend(t,z)
    G=9.8; L=2;             % set constants
    z1=z(1);                % get z1
    z2=z(2);                % get z2
    dzdt = [z2 ; -G/L*sin(z1);];
end
end
```