

Initial value ODEs in MATLAB

$$\frac{dy}{dt} = f(y(t), t)$$

Implementing the Euler method

$$y(t + \Delta t) = y(t) + \Delta t f(y(t), t)$$
$$y_{n+1} = y_n + \Delta t f(y_n, t_n)$$

Demonstrate for simplest ODE:

$$\frac{dy}{dt} = y(t) \rightarrow f(y_n, t_n) = y_n$$
$$y_{n+1} = y_n + \Delta t y_n$$

Code `euler_illustrate.m`.

Try with larger and smaller values of Δt

```
dt=0.15 ; %timestep
Tmax=1;
```

```
tlist=linspace(0, Tmax, Tmax/dt +1) ;
xlist=zeros(1, length(tlist));
```

```
%initialize
y0=1;
ylist(1)=y0;
```

```
for n=1:length(tlist)-1
    ylist(n+1)=ylist(n) + ylist(n)
end
```

```
figure
set(gca, 'FontSize', 16)
plot(tlist, ylist, '-o'); hold on
```

Local (per-timestep) error $\mathcal{O}\Delta t^2$

Implementing more sophisticated (and more accurate) methods.

Must write function containing right-hand side (rhs) $f(y, t)$.

Here, `simple_rhs_fun.m` applies to the example above, $f(y, t) = y$

```
function F=simple_rhs_fun(t,y)
    % NOTE!! Order of arguments reversed from class:  t,y
    %y is the state (solution variable) vector
    %F is the rate of change (velocity) vector
    %t is the time. Note, you have to use this t argument, even if i
    %not used in defining the velocity!

    %Here is an example for dy_dt=y

    F=[ y ] ;
```

STEPS:

- Specify time range where want solution $t_{\text{span}}=[0, T_{\text{max}}]$
- Specify initial value for solution $y(t)$: $y_0=1$;
- Call an ODE solver. Which one?
- **ode23: Second-Order Runge-Kutta routine**
- **ode45: Fourth-Order Runge-Kutta routine**
- **ode113: Variable order predictor-corrector routine**
- **ode15s: Variable order Gear method for stiff problems**

```
%Call ODE solver ode45  
[t,y]=ode45('simple_rhs_fun',tspan,y0)
```

... all put together in `simple_initial_value_solve.m`

Another example

$$\frac{dy(t)}{dt} = t^2 + y(t) \quad y(0) = 1.0 \quad t \in [0, 5] \quad (1)$$

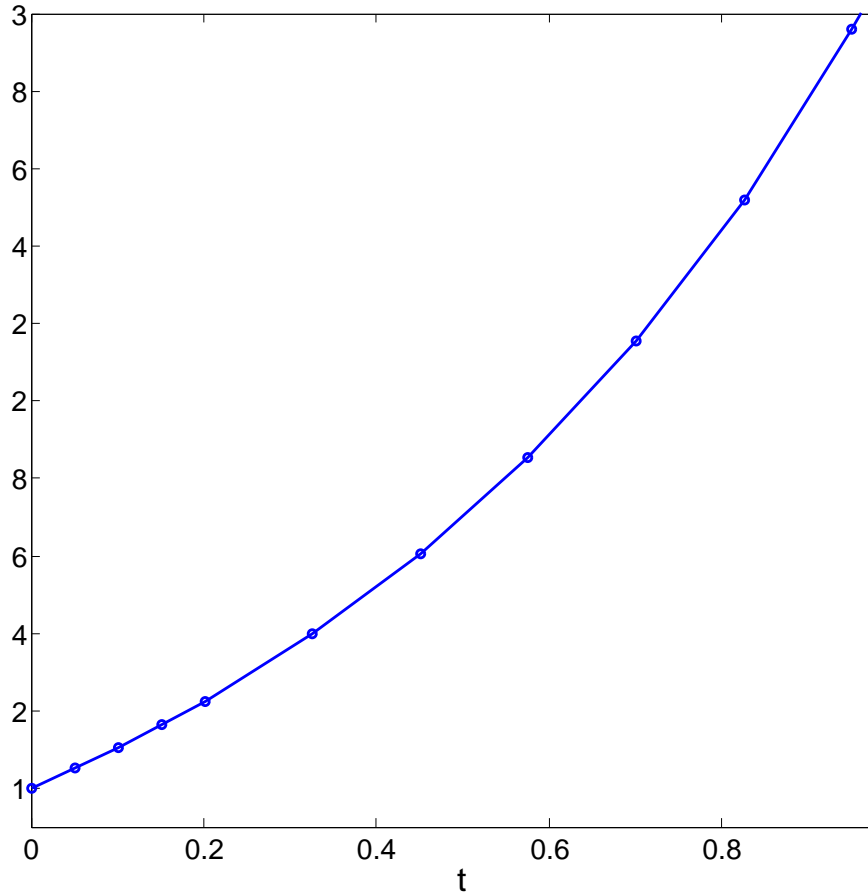
```
function rhs=simple_rhs_fun_B(t,y)
rhs = t^2 + y;
```

Code simple_initial_value_solve_B.m

```
tspan=[0,Tmax] ;
y0=1;
[t,y]=ode45('simple_rhs_fun_B',tspan,y0);
```

Irregular timesteps

- ode45 is an *adaptive-timestep* scheme. Chooses as large Δt as possible, at fixed level of local accuracy (tolerance).



Specifying list of times at which want solution $y(t)$

- E.g., use `tspan = 0:0.01:1`

Adjusting tolerance

```
TOL=1e-6;  
OPTIONS = odeset('RelTol',TOL,'AbsTol',TOL);  
[T,Y] = ODE45('simple_rhs_fun_B',TSPAN,Y0,OPTIONS);
```

See code `simple_initial_value_solve_B_tolerance_adjust`

Including a parameter

First, in the rhs function file

```
function rhs=simple_rhs_fun_C(t,y,dummy,p)
rhs = p(1)*t^2 + p(2)*y;
```

Next, in the main program

```
%define parameters -- vector (list)
p=[5, 2] ;

%Call ODE solver ode45
[t,y]=ode45('simple_rhs_fun_C',tspan,y0,OPTIONS,p);
```

Higher-order (i.e. higher-dimension) systems

$$\frac{d}{dt} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ -y_1^2 y_2 - y_1 \cdot \cos t + A \cdot \sin^2 t \end{bmatrix} = f(\mathbf{y}, t)$$

First, in the rhs function file

```
function rhs=high_order_rhs(t,y,dummy,A)
rhs = [y(2);
       y(3);
       -y(1)^2 * y(2) - y(1) * cos(t) + A*sin(t)^2];
```

Next, in the main program

```
A = 2; % Parameter values for our ODE
y0 = [0 1 0]; % Initial conditions
tspan = [0 30]; % time interval

[t,y] = ode45('high_order_rhs',tspan,y0,[],A);
```