Lecture 12: Solving ODEs in Matlab Using the Runge-Kutta Integrator `ODE45()`

Example 1: Let's solve a first-order ODE that describes exponential growth

$$\frac{dN}{dt} = aN$$

Let N = # monkeys in a population
a = time scale for growth (units = 1/time)

The analytical solution is $N(t) = N_0 e^{at}$

- The population N(t) grows exponentially assuming a > 0.
- The larger the value of a, the faster the population will grow over time

Photograph by Yukihiro Fukuda
Visions of Earth
National Geographic, July 2008
© 2008 National Geographic Society. All rights reserved.

Numerical Solution of a First-Order ODE using the Matlab command `ode45()`

In general, we want to solve an equation of the form: $\frac{dx}{dt} = f(x,t)$

Steps:

1. Define an m-file function (**`ode_derivs.m`** in the following example) that returns the derivative dx/dt

In a separate Matlab program (**`ode_derivs.m`**), do the following:

2. Initialize all parameters, initial conditions, etc.
3. Call the Matlab function **`ode45()`** to solve the ODE.
4. Plot the results

Step 1: Define a function named `ode02_derivs()` to compute and return the derivative defining the ODE:

$$\frac{dx}{dt} = ax$$

```
function dxdt = ode02_derivs(t,x)
     global a;
     dxdt = a * x;
 end
```

Notes:

1. the name of the m-file must match the name of the function (in this case **`ode02_derivs()`**)
2. the function returns the value **`dxdt`**, (=**`a*x`** in this example)
3. the parameter **`a`** is a global variable that must be set in the calling program

Step 2: Initialize all parameters and initial conditions in the main program `ode02.m`

```
tBegin = 0;       % time begin
tEnd = 10;        % time end

x0 = 10;          % initial number of monkeys

global a;         % declare a as global variable
a = 0.3;          % set value of the growth rate
```

Notes:

1. the parameter **`a`** is the global variable that we defined in the derivative function

Step 3: Call the Matlab function `ode45()` to solve the differential equation.

```
[t,x] = ode45(@ode02_derivs, [tBegin tEnd], x0);
```

1 2 3 4 5 6

1. returned times of each integration step
2. x values containing solution to the ODE
3. name of the integration scheme (ode45 in this example)
4. function that returns the derivative, i.e. $f(x,t)$
5. 1x2 matrix containing integration limits
6. initial conditions (first order ODE has only one IC)

Step 4: Plot the Results

```
plot(t,x,'bo-')
```

And that's it!
You, of course, should label your axes, etc.

Here's the full code:

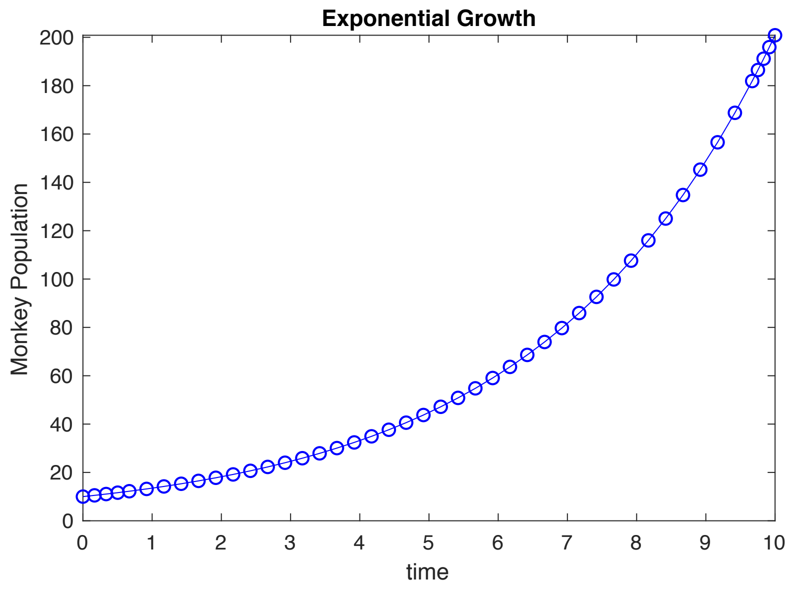
ode02.m

```
% Initialize Parameters
tBegin = 0;    % time begin
tEnd = 10;     % time end
x0 = 10;       % initial # monkies
global a;      % declare as a global variable
a = 0.3;       % set value of growth rate

% Use the Runge-Kutta 45 solver to solve the ODE
[t,x] = ode45(@ode02_derivs, [tBegin tEnd], x0);

%  Plot Results
plot(t,x,'bo-');
title('Exponential Growth');
xlabel('time');
ylabel('Monkey Population');
```

ode02_derivs.m

```
function dxdt = ode02_derivs(t,x)
     global a;
     dxdt = a * x;
end
```

Here's the result:



Notice the non-uniform spacing of the integration steps produced by the adaptive time step algorithm `ode45()`.

Problem 1: Create a new Matlab program and derivative function to model the dynamics of the logistic equation:

$$\frac{dN}{dt} = \frac{a}{b}N(b-N)$$

Plot the dynamics for:

$$a = 1$$

$$b = 100$$

$$0 \le t \le 10$$

$$x_0 = 10$$

Controlling Accuracy

Because the Runge-Kutta 4-5 integration scheme is an adaptive time step method, it is not possible to directly control the step size Δt.

Instead, we can control the integration tolerance by comparing the solutions obtained using the RK-4 and RK-5 methods.

When the error between the two solutions is too large, the time step is reduced to improve accuracy.

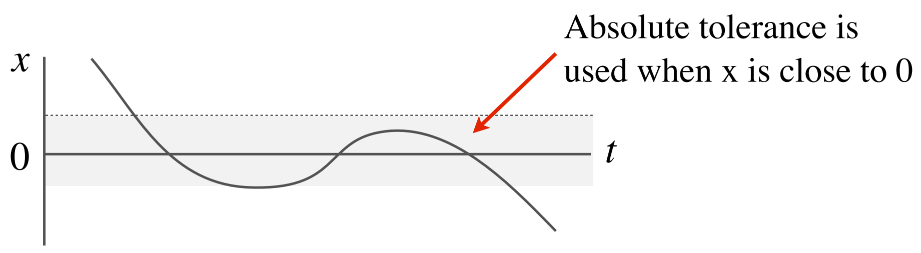
When the error between the two solutions is too small, the time step is increased to improve speed.

Controlling Accuracy

Relative Tolerance: $$Rel\ Tol = \frac{|x_4 - x_5|}{\min(|x_4|,|x_5|)} = 10^{-3} \text{ (by default)}$$

Absolute Tolerance: $$Abs\ Tol = |x_4 - x_5| = 10^{-6} \text{ (by default)}$$

where x_4 is the numerical solution using the RK-4 method
and x_5 is the numerical solution using the RK-5 method



Controlling Accuracy

Here's the Matlab code that sets the tolerances using the command **`odeset()`:**

```
options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);
[t,x] = ode45(@ode02_derivs, [tBegin tEnd], x0, options);
```

Example 2: Modify the ode_02.m code to do the following:

A: Plot the analytic solution on top of the numerical result. Create a second graph that shows the error. (`ode02B.m`)

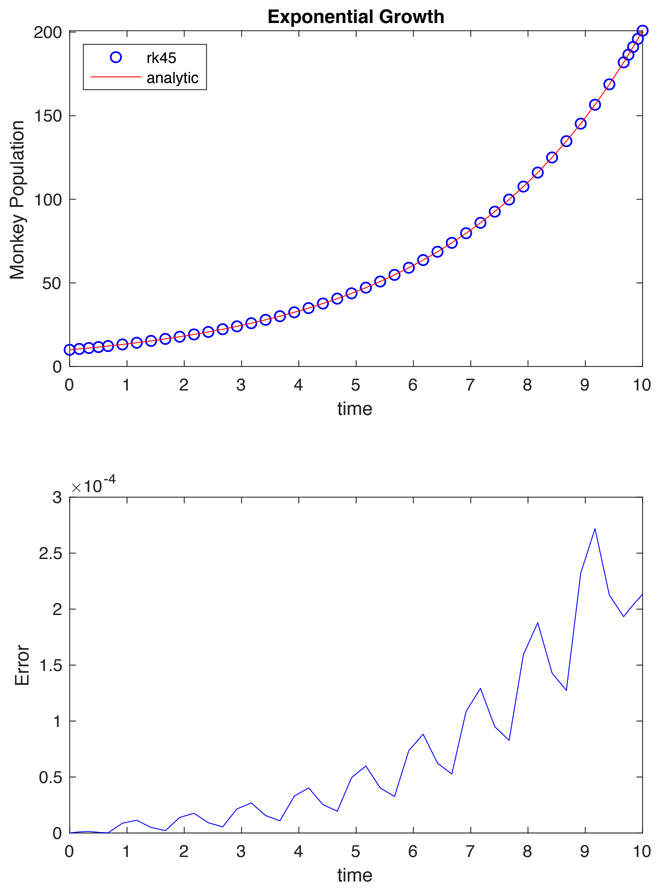
The analytic solution to $\frac{dN}{dt} = aN$ is $N(t) = N_0 e^{at}$

B: Use the odeset() command to adjust the integration tolerance on the Runge-Kutta scheme. (`ode02C.m`)

Here's the modified code to include the exact solution (`ode02B.m`):

```
      %%%  Initialization Section Not Shown  %%%

      % integrate using the Runge-Kutta 4-5 Scheme
      [t,x] = ode45(@ode02_derivs, [tBegin tEnd], x0);

      % Calculate exact solution
new → x_exact = x0 * exp(a * t);

      %%%%% Top Plot - x(t)
      subplot(2,1,1)
      plot(t,x,'bo');          % plot ode45 solution
      hold on
new → plot(t,x_exact,'r-')     % plot analytic solution

      title('Exponential Growth');             % title
      xlabel('time');                          % label x axis
      ylabel('Monkey Population');             % label y axis
new → legend('rk45', 'analytic', 'Location', 'Northwest')

    | %%%%  Lower Plot - error(t)
    | subplot(2,1,2)
new → plot(t,x-x_exact,'b-');                  % plot error
    |
    | xlabel('time');                          % label x axis
    | ylabel('Error');                         % label y axis
```

Output of `ode02B.m`



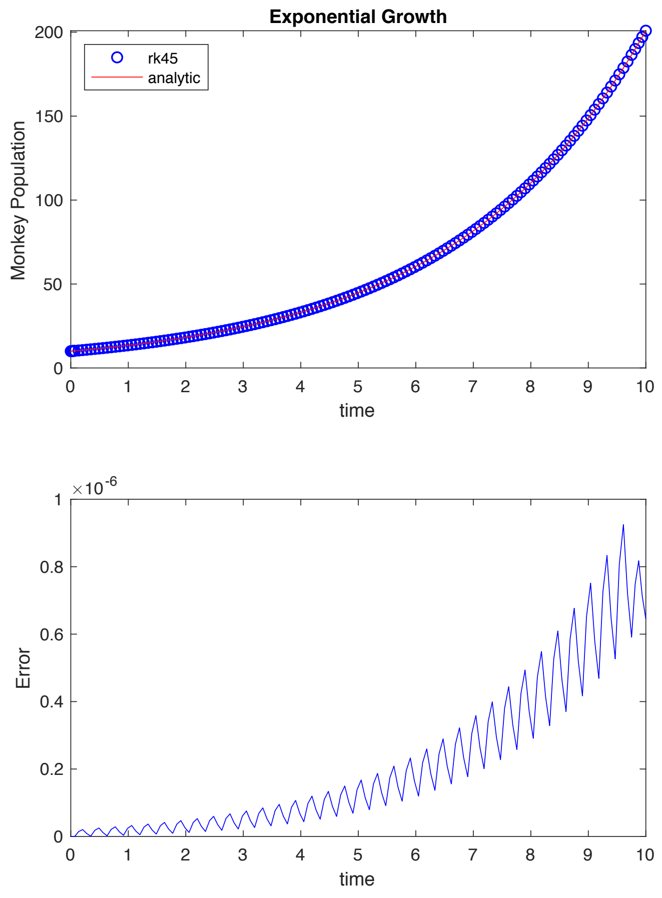
Here's the modified code to set the tolerances (`ode02C.m`):

new

new

```
%%%  Initialization Section Not Shown  %%%

%Set integration tolerances and integrate
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
[t,x] = ode45(@ode02_derivs, [tBegin tEnd], x0, options);

% Calculate exact solution
x_exact = x0 * exp(a * t);

%%%%% Top Plot - x(t)
subplot(2,1,1)
plot(t,x,'bo');          % plot ode45 solution
hold on
plot(t,x_exact,'r-')     % plot analytic solution

title('Exponential Growth');             % title
xlabel('time');                          % label x axis
ylabel('Monkey Population');             % label y axis
legend('rk45', 'analytic', 'Location', 'Northwest')

%%%%  Lower Plot - error(t)
subplot(2,1,2)
plot(t,x-x_exact,'b-');     % plot error

xlabel('time');                          % label x axis
ylabel('Error');                         % label y axis
```

Output of `ode02C.m`



Second-Order ODE

Split the 2nd order equation of motion

$$\frac{d^2x}{dt^2} = f(x, v, t)$$

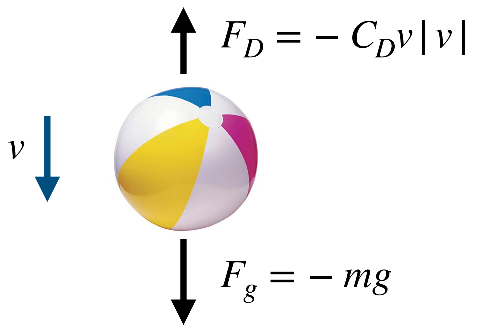
into 2 first-order equations:

$$\frac{dx}{dt} = v \qquad\qquad \frac{dv}{dt} = f(x, v, t)$$

We now need to specify 2 initial conditions: x_0 and v_0

And we must follow 2 variables in time: $x(t)$ and $v(t)$

Example 2: Free Fall Motion With Turbulent Air Drag



When v > 0 $\quad F_D = -C_D v^2$ ↓

When v < 0 $\quad F_D = C_D v^2$ ↑

$$F = -mg - C_D v|v|$$
$$ma = -mg - C_D v|v|$$

Equation of Motion:

$$\boxed{\frac{d^2x}{dt^2} = -g - \frac{C_D}{m} v|v|}$$

or:

$$\boxed{\begin{aligned} \frac{dv}{dt} &= -g - \frac{C_D}{m} v|v| \\ \frac{dx}{dt} &= v \end{aligned}}$$

Numerical Solution of a Second-Order ODE using the Matlab command `ode45()`

Follow these steps to numerically integrate an equation of the form

$$\frac{d^2x}{dt^2} = f(x, v, t)$$

Steps:

1. Define an m-file function that returns two derivatives: dx/dt and dv/dt

In a separate Matlab program, do the following:

2. Initialize all parameters, initial conditions, etc.
3. Call the Matlab function **`ode45()`** to solve the ODE.
4. Separate out x(t) and v(t) solutions
5. Plot the results

Step 1: Define a function named **`ode04_derivs()`** to compute and return the derivative defining the ODE:

```
function derivs = ode04_derivs(t, w)

global C_d;       % global variable: air drag coefficient
global m;         % global variable: mass of particle

g = 9.8;          % acceleration of gravity in m/s^2

x = w(1);         % w(1) stores x
v = w(2);         % w(2) stores v

% calculate the derivatives dx/dt and dv/dt
dxdt = v;
dvdt = -g - v * abs(v) * C_d / m ;

derivs = [dxdt; dvdt];  % return the derivatives
                        % as a 1x2 matrix
```

Note that the variable `w` contains two pieces: `w(1)` is the position `x` and `w(2)` is the velocity `v`.

Step 2: Initialize all parameters, initial conditions, etc. in the main program `ode04.m`

```
tBegin = 0;      % time begin
tEnd   = 2;      % time end

x0 = 0;          % initial position (m)
v0 = 15;         % initial velocity (m/s)

% define global variables used in derivative function
global m;
m   = 1;         % mass (kg)

global C_d;
C_d = 0.3;       % turbulent drag coef.
```

Step 3: Call the Matlab function `ode45()` to solve the differential equation.

```
[t,w] = ode45(@ode04_derivs, [tBegin tEnd], [x0 v0]);
 1  2         3        4                5          6
```

1. returned times of each integration step
2. n×2 matrix containing both x(t) and v(t) solutions
3. name of the integration scheme (ode45 in this example)
4. function that returns the derivatives
5. 1×2 matrix containing integration limits
6. 1×2 matrix containing initial conditions

Step 4: Separate out the `x(t)` and `v(t)` solutions from the matrix `w`

```
x = w(:,1);    % get x(t) from first column of w
v = w(:,2);    % get v(t) from second column of w
```

Step 4: Plot the Results

```
% top plot - x(t)
subplot(2,1,1)
plot(t,x);

ylabel('position (m)');
xlabel('time (s)');


% bottom plot - v(t)
subplot(2,1,2)
plot(t,v);

ylabel('velocity (m/s)');
xlabel('time (s)');
```

Here’s the full code

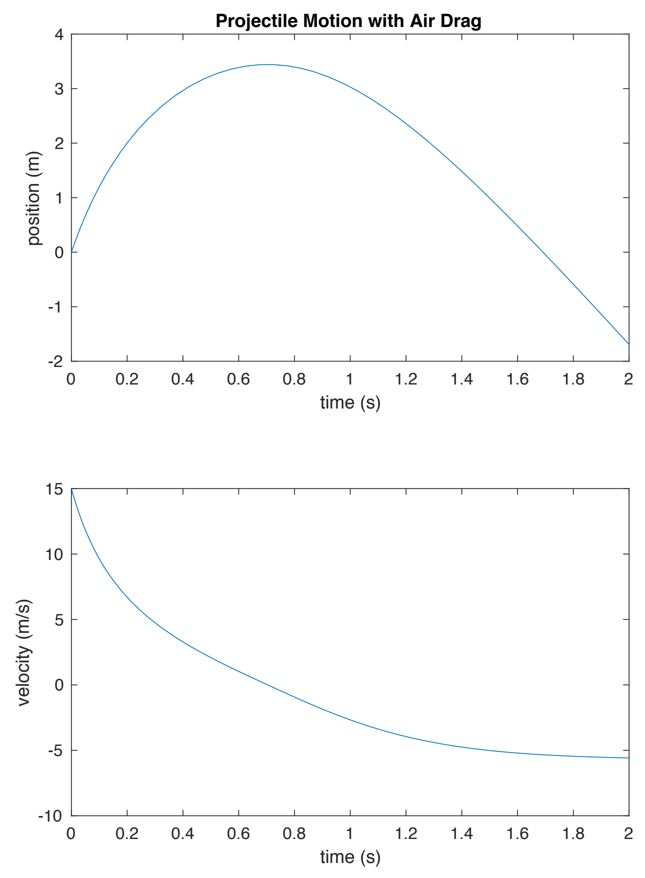
ode04A.m

```
% Initialize Parameters
tBegin = 0;      % time begin
tEnd   = 2;      % time end
x0 = 0;          % initial position (m)
v0 = 15;         % initial velocity (m/s)

% global variables
global m;        m   = 1;      % air drag
global C_d;      C_d = 0.3;    % mass

% Integrate ODE
[t,w] = ode45(@ode04_derivs, ...
          [tBegin tEnd], [x0 v0]);
x = w(:,1);      % extract x(t)
v = w(:,2);      % extract v(t)

% top plot - x(t)
subplot(2,1,1)
plot(t,x);
ylabel('position (m)');
xlabel('time (s)');

% bottom plot - v(t)
subplot(2,1,2)
plot(t,v);
ylabel('velocity (m/s)');
xlabel('time (s)');
```

ode04_derivs.m

```
function derivs = ode04_derivs(t, w)

global C_d;         % air drag
global m;           % particle mass

g = 9.8;            % g

x = w(1);           % w(1) stores x
v = w(2);           % w(2) stores v

% calculate dx/dt and dv/dt
dxdt = v;
dvdt = -g - v * abs(v) * C_d / m ;

derivs = [dxdt; dvdt];
```

Here’s the result:


Projectile Motion with Air Drag
position (m)
time (s)
velocity (m/s)
time (s)