L33: Ordinary Differential Equations

Part 3: Systems of ordinary differential equations; ode45

Lucas A. J. Bastien

E7 Spring 2017, University of California at Berkeley

April 14, 2017

Version: release

Announcements

Lab 12 is due on April 21 at 12 pm (noon)

Today:

- Ordinary differential equations – Part 3 (Chapter 19)

Next week:

- Searching and sorting (no required reading)

Numerical methods for “solving” initial value problems

We are learning methods to “solve” **first-order initial value problems**

Notation:

Generic first-order initial value problem (unknown is y, a function of t):

$$y' = F(t, y) \qquad \text{(ODE)}$$

$$y(t = t_0) = y_0 \qquad \text{(Initial condition)}$$

Numerical methods for "solving" initial value problems

We are learning methods to "solve" **first-order initial value problems**

Notation:
Generic first-order initial value problem (unknown is y, a function of t):

$$y' = F(t, y) \qquad \text{(ODE)}$$

$$y(t = t_0) = y_0 \qquad \text{(Initial condition)}$$

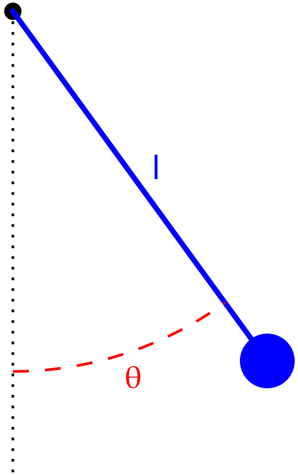
General approach: estimate the function's value at discrete small intervals (*i.e.* estimate the function at points $t_0, t_1, t_2, \ldots$), starting from the known value (*i.e.* the initial condition), **assuming that the slope is constant over each interval:**

$$y(t_{i+1}) = y(t_i) + \text{slope} \times \Delta t_i$$

where $\Delta t_i = (t_{i+1} - t_i)$ is the "spacing" or "time step"

Different methods: different approximations for the slope

What about higher-order ODEs? Pendulum example

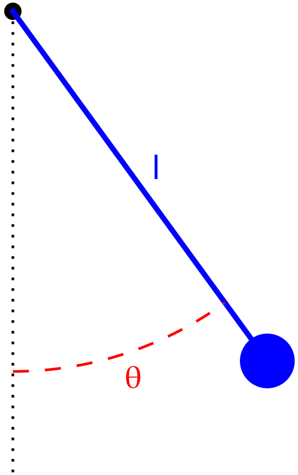


Assume that θ is small enough so that $\sin(\theta) \approx \theta$:

$$\theta'' + \omega^2\theta = 0 \qquad \text{with } \omega^2 = g/l$$

g: acceleration of gravity; l: length of the pendulum

What about higher-order ODEs? Pendulum example



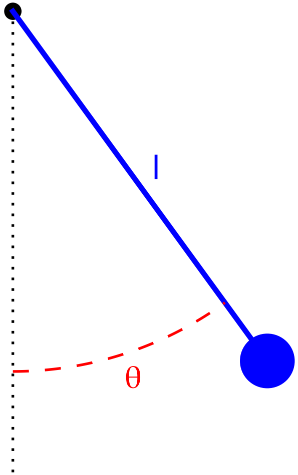
Assume that θ is small enough so that $\sin(\theta) \approx \theta$:

$$\theta'' + \omega^2\theta = 0 \qquad \text{with } \omega^2 = g/l$$

This equation is a second-order ordinary differential equation

g: acceleration of gravity; l: length of the pendulum

What about higher-order ODEs? Pendulum example



Assume that θ is small enough so that $\sin(\theta) \approx \theta$:

$$\theta'' + \omega^2\theta = 0 \qquad \text{with } \omega^2 = g/l$$

This equation is a second-order ordinary differential equation

We can write it as a system of 2 first-order ordinary differential equations

g: acceleration of gravity; l: length of the pendulum

What about higher-order ODEs? Pendulum example

Step 1: Change of variables

$$
\begin{aligned}
z_1 &= \theta \\
z_2 &= \theta' = z_1'
\end{aligned}
$$

What about higher-order ODEs? Pendulum example

Step 1: Change of variables

$$z_1 = \theta$$
$$z_2 = \theta' = z_1'$$

Step 2: Re-write the second-order differential equation

$$\theta'' + \omega^2 \theta = 0 \ \text{ yields:}$$
$$z_2' + \omega^2 z_1 = 0$$

What about higher-order ODEs? Pendulum example

Step 1: Change of variables

$$z_1 = \theta$$
$$z_2 = \theta' = z_1'$$

Step 2: Re-write the second-order differential equation

$$\theta'' + \omega^2\theta = 0 \text{ yields:}$$
$$z_2' + \omega^2 z_1 = 0$$

therefore:

$$\begin{bmatrix} z_1' \\ z_2' \end{bmatrix} = \begin{bmatrix} z_2 \\ -\omega^2 z_1 \end{bmatrix}$$

What about higher-order ODEs? Pendulum example

Step 1: Change of variables

$$\begin{aligned} z_1 &= \theta \\ z_2 &= \theta' = z_1' \end{aligned}$$

Step 2: Re-write the second-order differential equation

$$\begin{aligned} &\theta'' + \omega^2 \theta = 0 \ \text{ yields:} \\ &z_2' + \omega^2 z_1 = 0 \end{aligned}$$

therefore:

$$\begin{bmatrix} z_1' \\ z_2' \end{bmatrix} = \begin{bmatrix} z_2 \\ -\omega^2 z_1 \end{bmatrix}$$

i.e.

$$z' = F(t, z) \ \text{ with: } \ z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \ \text{ and: } \ F(t, z) = \begin{bmatrix} z_2 \\ -\omega^2 z_1 \end{bmatrix}$$

What about higher-order ODEs? Pendulum example

Step 3: Apply a numerical ODE-"solving" method. For example, with the explicit Euler method:

$$z(t_{i+1}) = z(t_i) + F(t_i, z(t_i))\Delta t$$

What about higher-order ODEs? Pendulum example

Step 3: Apply a numerical ODE-"solving" method. For example, with the explicit Euler method:

$$z(t_{i+1}) = z(t_i) + F(t_i, z(t_i))\Delta t$$

$$\begin{bmatrix} z_1(t_{i+1}) \\ z_2(t_{i+1}) \end{bmatrix} = \begin{bmatrix} z_1(t_i) \\ z_2(t_i) \end{bmatrix} + \begin{bmatrix} z_2(t_i) \\ -\omega^2 z_1(t_i) \end{bmatrix} \Delta t$$

What about higher-order ODEs? Pendulum example

Step 3: Apply a numerical ODE-"solving" method. For example, with the explicit Euler method:

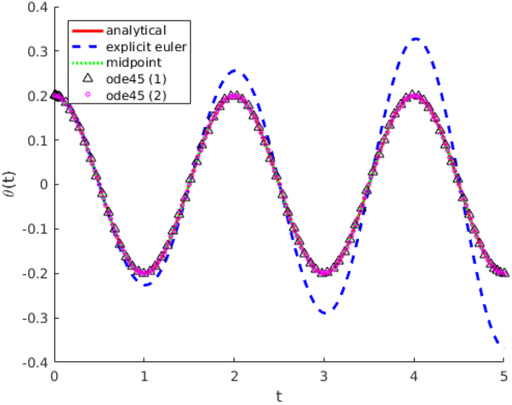
$$z(t_{i+1}) = z(t_i) + F(t_i, z(t_i))\Delta t$$

$$\begin{bmatrix} z_1(t_{i+1}) \\ z_2(t_{i+1}) \end{bmatrix} = \begin{bmatrix} z_1(t_i) \\ z_2(t_i) \end{bmatrix} + \begin{bmatrix} z_2(t_i) \\ -\omega^2 z_1(t_i) \end{bmatrix} \Delta t$$

In other words:

$$\begin{aligned} z_1(t_{i+1}) &= z_1(t_i) + z_2(t_i)\Delta t \\ z_2(t_{i+1}) &= z_2(t_i) - \omega^2 z_1(t_i)\Delta t \end{aligned}$$

What about higher-order ODEs? Pendulum example



(See script `pendulum.m`)

Matlab's built-in ode45 function

Goal: Solve a system of n first-order ordinary differential equations

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

where y is a column vector:

```
[times, y] = ode45(f, tspan, y0)
```

Matlab's built-in ode45 function

Goal: Solve a system of n first-order ordinary differential equations

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

where y is a column vector:

```
[times, y] = ode45(f, tspan, y0)
```

- f: function handle that represents the system to solve

Matlab's built-in ode45 function

Goal: Solve a system of n first-order ordinary differential equations

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

where y is a column vector:

```
[times, y] = ode45(f, tspan, y0)
```

- `f`: function handle that represents the system to solve
- `tspan`: either

Matlab's built-in ode45 function

Goal: Solve a system of n first-order ordinary differential equations

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

where y is a column vector:

```
[times, y] = ode45(f, tspan, y0)
```

- `f`: function handle that represents the system to solve
- `tspan`: either
  - a 2×1 vector: `[t0,tf]`. In this case: Matlab chooses the size of the time steps (may not all be equal)

Matlab's built-in ode45 function

Goal: Solve a system of n first-order ordinary differential equations

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

where y is a column vector:

```
[times, y] = ode45(f, tspan, y0)
```

- `f`: function handle that represents the system to solve
- `tspan`: either
  - a 2×1 vector: `[t0,tf]`. In this case: Matlab chooses the size of the time steps (may not all be equal)
  - a $1 \times m$ $(m > 2)$ vector of times at which to calculate the "solution"

Matlab's built-in ode45 function

Goal: Solve a system of n first-order ordinary differential equations

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

where y is a column vector:

```
[times, y] = ode45(f, tspan, y0)
```

- `f`: function handle that represents the system to solve
- `tspan`: either
  - a 2×1 vector: `[t0,tf]`. In this case: Matlab chooses the size of the time steps (may not all be equal)
  - a $1 \times m$ ($m > 2$) vector of times at which to calculate the "solution"
- `y0` the value of y at time `tspan(1)`

Matlab's built-in ode45 function

Goal: Solve a system of n first-order ordinary differential equations

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

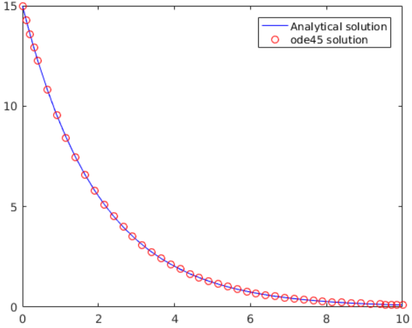
where y is a column vector:

```
[times, y] = ode45(f, tspan, y0)
```

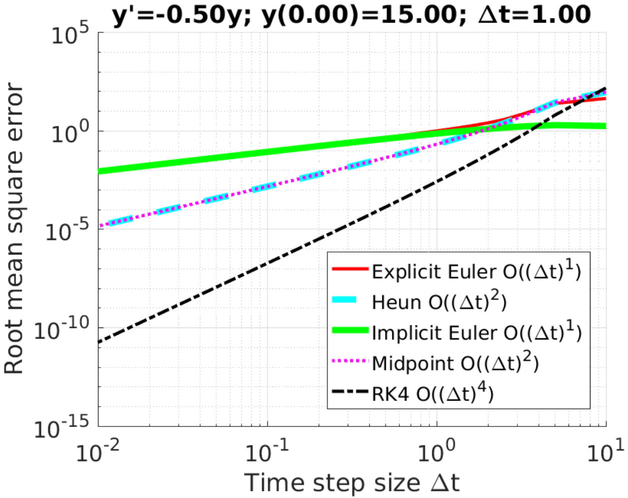
- `f`: function handle that represents the system to solve
- `tspan`: either
  - a 2×1 vector: `[t0,tf]`. In this case: Matlab chooses the size of the time steps (may not all be equal)
  - a $1 \times m$ $(m > 2)$ vector of times at which to calculate the "solution"
- `y0` the value of y at time `tspan(1)`
- `times`: the vector of times
- `y`: the numerical "solution" at times "`times`"

Matlab's built-in ode45 function: example

```
% Application of ode45 to exponential decay
close all
f = @(t,y) -0.5 * y;
[t, y] = ode45(f, [0, 10], 15);
plot(t, 15*exp(-0.5*t), 'b-')
hold on
plot(t, y, 'ro')
legend('Analytical solution', 'ode45')
```



Order of the numerical methods for ODE solving



The slope of the line in a log-log plot indicates the order of the method

IMPORTANT practice question

We use a 2nd- and a 4th-order ODE solver approximation to estimate the solution of an initial value problem, using equally-spaced points (spacing is Δx).

Which of the following statements are true about the overall error?

(A) The error made when using the 4th-order method is always smaller than the error made when using the 2nd-order method

(B) On average, if we reduce Δx by a factor of 2, the error made when using the 2nd-order method is divided by 4

(C) On average, if we reduce Δx by a factor of 2, the error made when using the 4th-order method is divided by 4

(D) On average, if we reduce Δx by a factor of 2, the error made when using the 4th-order method is divided by 16

(E) The error made when using the 4th-order method is always twice as small as the error made when using the 2nd-order method

IMPORTANT practice question

We use a 2$^{\mathrm{nd}}$- and a 4$^{\mathrm{th}}$-order ODE solver approximation to estimate the solution of an initial value problem, using equally-spaced points (spacing is Δx).

Which of the following statements are true about the overall error?

(A) The error made when using the 4$^{\mathrm{th}}$-order method is always smaller than the error made when using the 2$^{\mathrm{nd}}$-order method

(B) On average, if we reduce Δx by a factor of 2, the error made when using the 2$^{\mathrm{nd}}$-order method is divided by 4

(C) On average, if we reduce Δx by a factor of 2, the error made when using the 4$^{\mathrm{th}}$-order method is divided by 4

(D) On average, if we reduce Δx by a factor of 2, the error made when using the 4$^{\mathrm{th}}$-order method is divided by 16

(E) The error made when using the 4$^{\mathrm{th}}$-order method is always twice as small as the error made when using the 2$^{\mathrm{nd}}$-order method

$$y(t_{i+1}) = y(t_i) + \mathrm{slope} \times \Delta t$$

Rearrange:

$$\mathrm{slope} = \frac{y(t_{i+1}) - y(t_i)}{\Delta t}$$

$$y(t_{i+1}) = y(t_i) + \mathrm{slope} \times \Delta t$$

Rearrange:

$$\mathrm{slope} = \frac{y(t_{i+1}) - y(t_i)}{\Delta t}$$

Explicit Euler method (forward finite-difference formula)

$$\mathrm{slope}(t_i) = \frac{y(t_{i+1}) - y(t_i)}{\Delta t}$$

Numerical ODE solvers and finite difference formulae

$$y(t_{i+1}) = y(t_i) + \text{slope} \times \Delta t$$

Rearrange:

$$\text{slope} = \frac{y(t_{i+1}) - y(t_i)}{\Delta t}$$

Explicit Euler method (forward finite-difference formula)

$$\text{slope}(t_i) = \frac{y(t_{i+1}) - y(t_i)}{\Delta t}$$

Implicit Euler method (backward finite-difference formula)

$$\text{slope}(t_{i+1}) = \frac{y(t_{i+1}) - y(t_i)}{\Delta t}$$