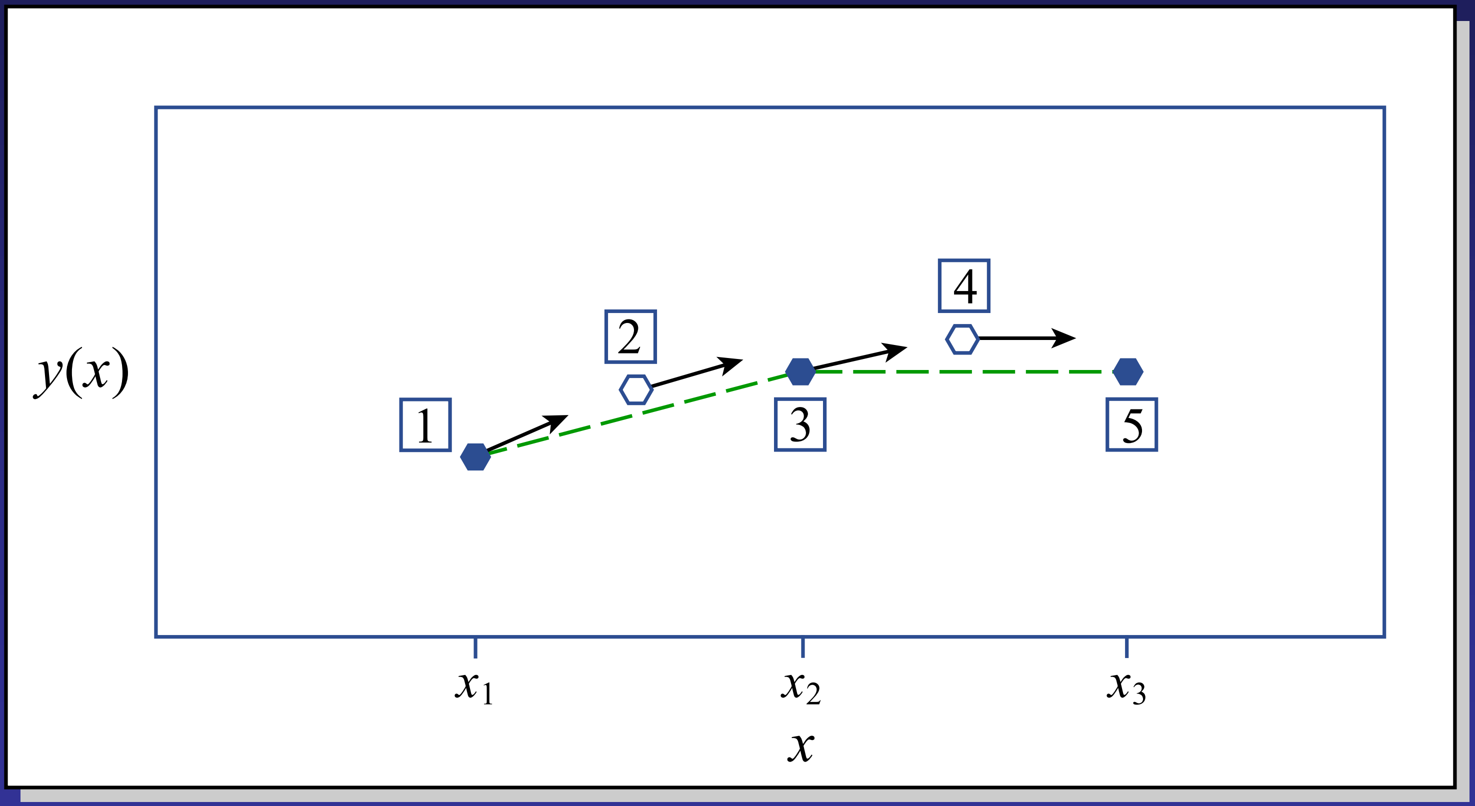
MatLab Programming – Algorithms to Solve Differential Equations



Adapted from Figure 16.1.2. In Numerical Recipes in C: *The Art of Scientific Computing*.
2nd Ed. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery.
Cambridge, UK: Cambridge University Press, 1992. p. 711. ISBN: 9780521431088. Figure by MIT OCW.

Revisit the task of recovering the motion of a dynamical system from its equation of motion

Consider the simplest 1st order system:

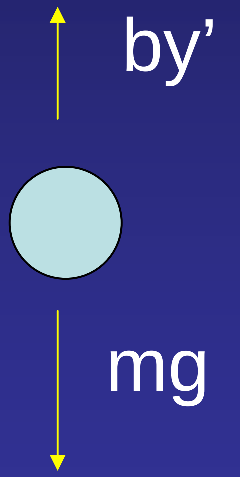
$$b\dot{x} + kx = 0$$

What does this system corresponds to?

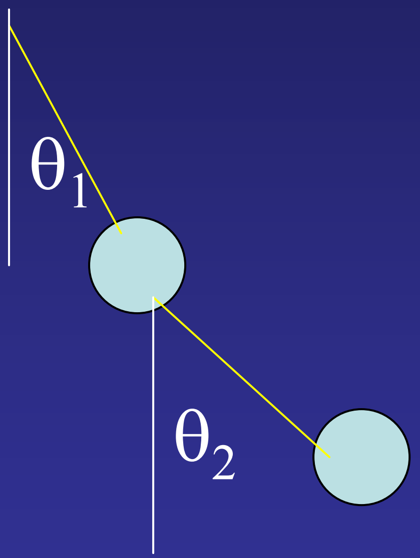
The solution of this system can of course be obtained analytically but also simply numerically by a single integration

Limitation of Simple Integration: Quad

Simple integration is very limited and does not solve a large class of dynamic problems. As examples:



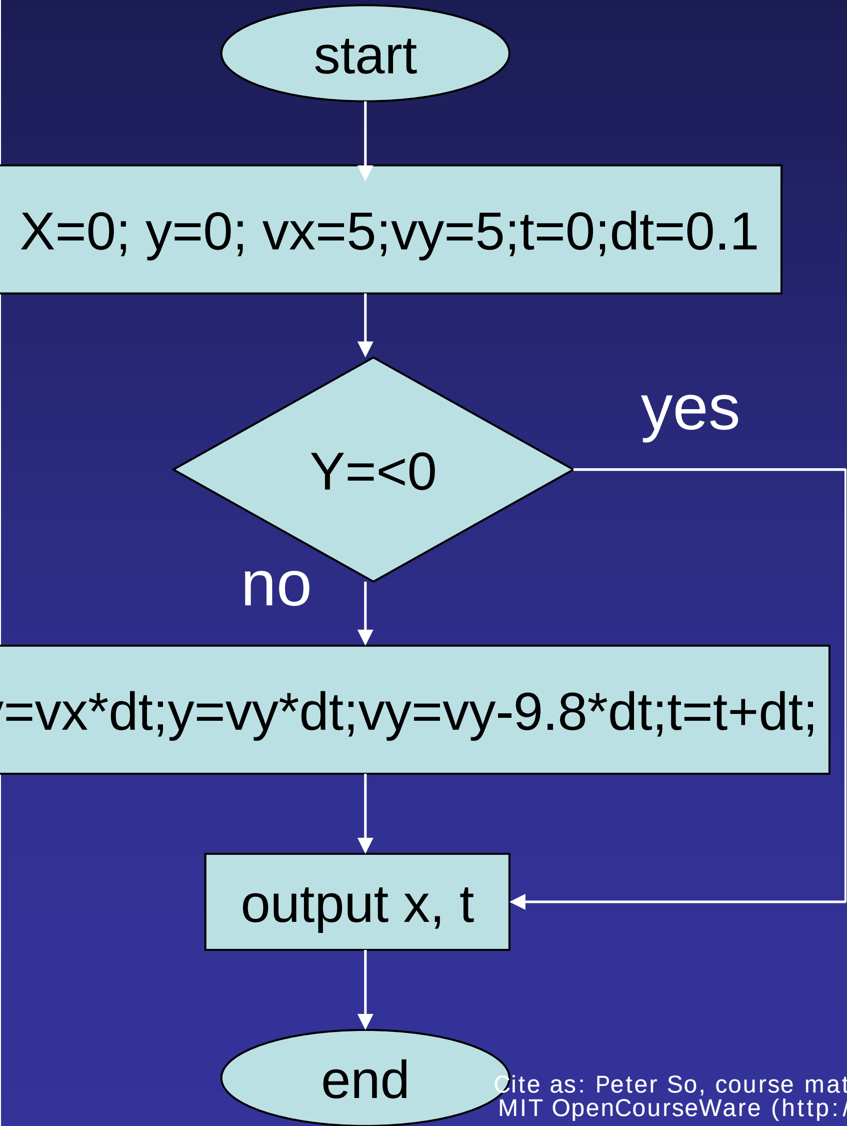
Falling ball – 2nd order

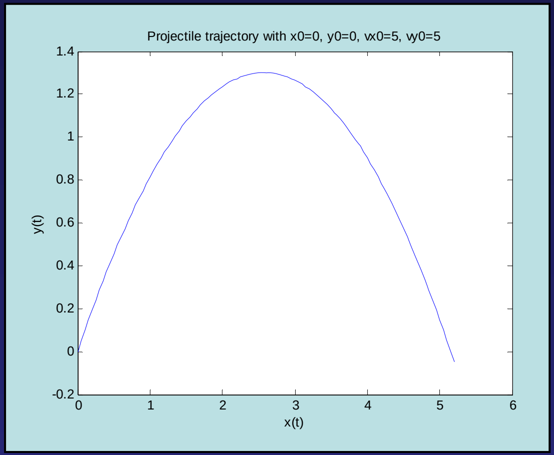


Coupled multiple degree of freedom system

How did we solve this class of problems?

We use a very simple straight forward approach of doing numerical integration:





Actually, this simple approach Has a name – it is called Euler Method

In general, you should NEVER ever use Euler Method. People uses it only if they don't know any better.

The General Numerical Problem of Solving Ordinary Differential Equations (ODEs)

$$y^{(n)} = f(y^{(n-1)}, \cdots, y', y, t)$$

Note that y does not have to be a scaler but can be a vector as in the case for multiple degrees of freedom systems

$$y = (y_1, y_2, \cdots y_m)$$

Converting higher order differential equation to a system of first order differential equation

Consider probably the most important case:

$$y'' = f(t)y' + g(t)y + h(t)$$

This can be readily converted to a system of first order differential equations

$$y_2 = y' \quad y_1 = y$$

$$y_1' = y_2$$

$$y_2' = f(t)y_2 + g(t)y_1 + h(t)$$

General equivalence between higher order differential equation and a system of first order equations

$$y^{(n)} = f(y^{(n-1)}, \cdots, y', y, t)$$

$$y_1 = y; \; y_2 = y'; \cdots, \; y_{n-1} = y^{(n-2)}; \; y_n = y^{(n-1)}$$

$$y_n' = f(y_n, \cdots, y_2, y_1, t)$$

The problem of solving all higher order ordinary differential equation is thus reduced to solving a system of linear differential equations

Solving linear first order differential equation by Euler Method

In general, the system of equations look like:

$$y_i'(t) = f_i(y_1, y_2, \cdots, y_n, t) \quad i = 1 \cdots n$$

Euler Method says:

$$y_i(j\Delta t) = y_i((j-1)\Delta t) + f_i(y_1((j-1)\Delta t), \cdots, y_n((j-1)\Delta t), (j-1)\Delta t)\Delta t$$
$$i = 1 \cdots n$$

This equation can be solved if we have the initial conditions:

$$y_1(0) = y_{10},\ y_2(0) = y_{20}, \cdots, y_n(0) = y_{n0}$$

What is the accuracy of the Euler Method?

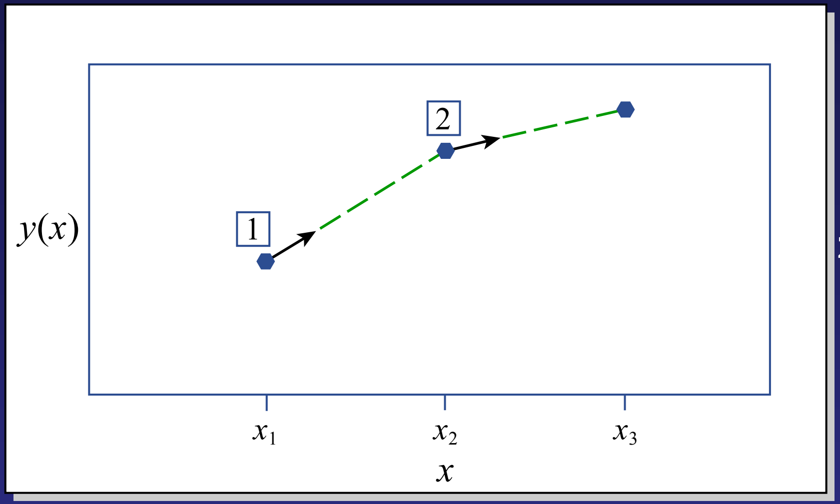
Euler method is equivalent to taking the 1st order Taylor series expansion for $y_i(t)$; it is unsymmetric and uses only the derivative information at the start of the time step

$$y_i(j\Delta t) = y_i((j-1)\Delta t) + f_i(y_1((j-1)\Delta t),\cdots,y_n((j-1)\Delta t),(j-1)\Delta t)\Delta t + O(\Delta t^2)$$

$$i = 1\cdots n$$

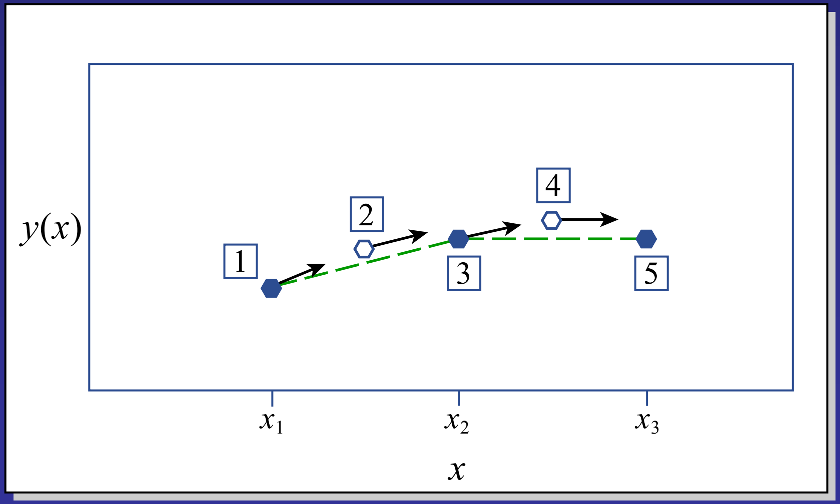
Correction is only one less order then the correction term.
How do we get better accuracy?

Schematically, the differences between Euler and 2nd order Runge-Kutta are fairly clear



Euler Method

Adapted from Figure 16.1.1. In *Numerical Recipes in C: The Art of Scientific Computing*. 2nd Ed. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Cambridge, UK: Cambridge University Press, 1992. p. 711. ISBN: 9780521431088. Figure by MIT OCW.



2nd order Runge-Kutta

Adapted from Figure 16.1.1. In *Numerical Recipes in C: The Art of Scientific Computing*. 2nd Ed. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Cambridge, UK: Cambridge University Press, 1992. p. 711. ISBN: 9780521431088. Figure by MIT OCW.

From Numerical Recipe in C

A working ODE solver – Runge-Kutta Method

Estimate where the mid-point for y_i is first. Then, Evaluate the slope at the mid-point to estimate the next value of y_i.

$$k1_i = f_i(y_1((j-1)\Delta t),\cdots,y_n((j-1)\Delta t),(j-1)\Delta t)\Delta t$$

$$k2_i = f_i(y_1((j-1)\Delta t)+k1_1/2,\cdots,y_n((j-1)\Delta t+k1_n/2),(j-1/2)\Delta t)\Delta t$$

$$y_i(j\Delta t)=y_i((j-1)\Delta t)+k2_i$$

$$i=1\cdots n$$

Because of symmetry, this method is good to $O(\Delta t^3)$

This is called the 2nd order Runge-Kutta method.

Higher Order Runge-Kutta Method

Just like Simpson method can be extended to higher order estimate, Runge-Kutta also has straightforward Higher order analog. The most commonly used one is the 4th order Runge-Kutta method

$$k1_i = f_i(y_1((j-1)\Delta t),\cdots,y_n((j-1)\Delta t),(j-1)\Delta t)\Delta t$$

$$k2_i = f_i(y_1((j-1)\Delta t)+k1_1/2,\cdots,y_n((j-1)\Delta t+k1_n/2),(j-1/2)\Delta t)\Delta t$$

$$k3_i = f_i(y_1((j-1)\Delta t)+k2_1/2,\cdots,y_n((j-1)\Delta t+k2_n/2),(j-1/2)\Delta t)\Delta t$$

$$k4_i = f_i(y_1((j-1)\Delta t)+k3_1\cdots,y_n((j-1)\Delta t+k3_n),j\Delta t)\Delta t$$

$$y_i(j\Delta t) = y_i((j-1)\Delta t)+k1_i/6+k2_i/3+k3_i/3+k4_i/6+O(\Delta t^5)$$

$$i = 1\cdots n$$

Runge-Kutta methods are implemented in MatLab as ODE23 and ODE45 functions

Using MatLab to solve a system of differential equations

Consider solving the following system of ODE:

$$y'_1 = y_2\,y_3 \qquad y_1(0) = 0$$

$$y'_2 = -y_1\,y_3 \qquad y_2(0) = 1$$

$$y'_3 = -0.51y_1\,y_2 \qquad y_3(0) = 1$$

Adapted from MATLAB Help Sections. Figure by MIT OCW.

Using MatLab to solve a system of differential equations

(1) First define the system of ODEs as a function:

```
function dy = system(t,y)
dy = zeros(3,1);    % a column vector
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
```

(2) Call ODE45 or ODE23 using the function handle

```
[T,Y] = ode45(@system,[0 12],[0 1 1]);
```

(3) Plot result

```
plot(T,Y(:,1),'-',T,Y(:,2),'-.',T,Y(:,3),'.')
```