Runge-Kutta Methods

Dr. Mitch Pryor

Lesson Objective

- Generalize the concept of slope estimation seen in the previous lesson (Euler’s, Huen’s, and Midpoint Methods) and then extend the concept to solve sets of 1st order ODEs instead of just one.

Runge-Kutta Methods

- So far, we have learned to solve one 1st order ODE

  Given: $\frac{dy}{dt} = f(t, y)$ $y(0) = y_o$ Find: $y(t) = ?$

- using the following methods:
  - Euler’s method (slope at the beginning of interval)
  - Heun’s method (slope at beginning and end of interval)
  - Midpoint method (slope at midpoint of interval)
- Each used a different estimation of the slope along the interval, *h.*
- The general methods for estimating the slope are known as **Runge-Kutta Methods**

$$y_{i+1} = y_i + h\dot{y} = y_i + h\Im(h, t_i, y_i)$$

Runge-Kutta Example

Find the Runge-Kutta (RK) method that includes slopes at the beginning, middle, and end of an interval where the middle interval is twice as important as the slopes at the beginning or end of the interval.

Solution

$$y_{i+1} = y_i + \frac{h}{4}(k_1 + 2k_2 + \mathrm{k}_3)$$

where,

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_1)$$
$$k_3 = f(x_i + h, y_i + hk_2)$$

Or, is this set of slopes any less valid? What is the difference?

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_1)$$
$$k_3 = f(x_i + h, y_i + hk_1)$$

Performance of RK methods

Given:

$$f(x,y) = \frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8.5 \qquad y(0) = 1$$

Find: y(2)=? using Euler's Method with *h*=0.5

Solution:

Recall Euler's Method $y_{i+1} = y_i + hf(x_i, y_i)$

Which we can apply for the first step and second step.

$$\begin{aligned} y(0.5) &= y_1 \\ &= y_0 + hy_0^{'} \\ &= 1 + 0.5(-2(0)^3 + 12(0)^2 - 20(0) + 8.5) \\ &= 5.25 \end{aligned}$$

$$\begin{aligned} y(1.0) &= y_2 \\ &= y_1 + hy_1^{'} \\ &= 5.25 + 0.5(-2(.5)^3 + 12(.5)^2 - 20(.5) + 8.5) \\ &= 5.875 \end{aligned}$$

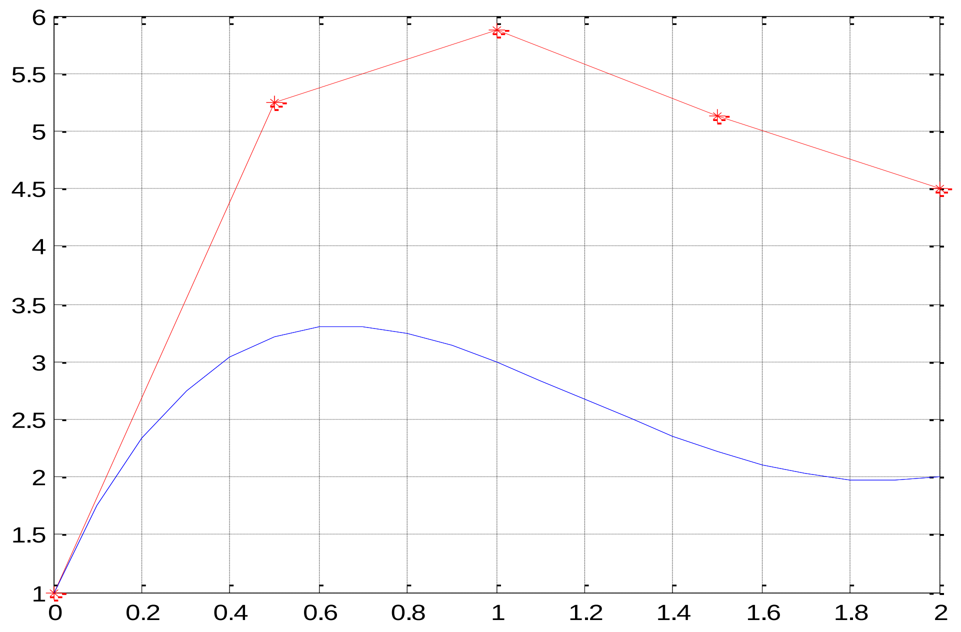
Performance of RK methods

$$f(x,y)=\frac{dy}{dx}=-2x^3+12x^2-20x+8.5$$

Summarizing the results for a few more steps and we get:

iteration	**0**	**1**	**2**	**3**	**4**
x	0	0.5	1.0	1.5	2.0
y	1	5.25	5.875	5.125	4.5

Which we can compare the analytical solution: $y(x)=-\frac{1}{2}x^4+4x^3-10x^2+8.5x+1$



Using only the initial slope and too large of a step size, Euler's method does not track the solution very well.

Performance of RK methods

$$f(x,y)=\frac{dy}{dx}=-2x^3+12x^2-20x+8.5$$

Let’s try a 2nd order RK method using the slopes at the beginning and middle of the interval (equally weighted)

$$y_{i+1}=y_i+hy'_{i,average}$$

$$y_{i+1}=y_i+h\phi(x_i,y_i,h)$$

Where,

$$\phi(x_i,y_i,h)=a_1k_1+a_2k_2$$

$$k_1=f(x_i,y_i)$$

$$k_2=f(x_i+\tfrac{1}{2}h,y_i+\tfrac{1}{2}hk_1)$$

$$\left.\begin{aligned}a_1&=\tfrac{1}{2}\\a_2&=\tfrac{1}{2}\end{aligned}\right\}\text{weighting parameters where sum must equal 1.}$$

So calculating the estimate for the first step, we find

$$y(0.5)=y(0)+0.5(\tfrac{1}{2}k_1+\tfrac{1}{2}k_2)$$

where,

$$k_1=f(x_i,y_i)=-2(0)^3+12(0)^2-20(0)+8.5=8.5$$

$$k_2=f(x_i+\tfrac{1}{2}h,y_i+\tfrac{1}{2}hk_1)=-2(.25)^3+12(.25)^2-20(.25)+8.5=4.28125$$

therefore,

$$y(0.5)=1+0.5(\tfrac{1}{2}(8.5)+\tfrac{1}{2}(4.28125))=4.18$$

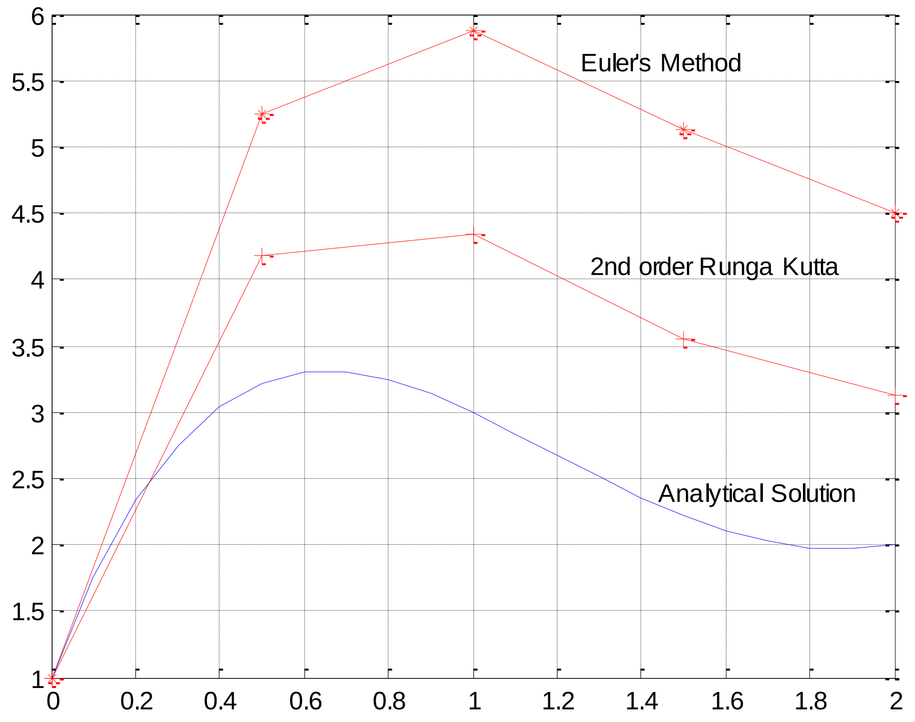
Performance of RK methods

$$f(x,y) = \frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8.5$$

Performing the calculation for the next for iterations yields

iteration	0	1	2	3	4
x	0	0.5	1.0	1.5	2.0
y	1	4.18	4.34	3.55	3.12

and the 2nd order RK performs better than the Euler (or 1st order RK) method.



Classical 4th Order Runge-Kutta Method

$$
\begin{aligned}
y_{i+1} &= y_i + h y'_{i,average} \\
&= y_i + h\phi(x_i, y_i, h) \\
&= y_i + h\left(a_1 k_1 + a_2 k_2 + a_3 k_3 + a_4 k_4\right) \\
&= y_i + \tfrac{1}{6} h\left(k_1 + 2k_2 + 2k_3 + k_4\right)
\end{aligned}
$$

where,

$k_1 = f(x_i, y_i)$	slope at beginning of interval is used to…
$k_2 = f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_1)$	slope at the midpoint of the interval, which is used to…
$k_3 = f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_2)$	get a better (corrected) slope at the midpoint of the interval…
$k_4 = f(x_i + h, y_i + hk_3)$	is used to get the slope at the end of the interval

Back to our example

$$f(x,y) = \frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8.5$$

If we calculate the slopes for the first interval using the classic 4th order Runge-Kutta Method:

$$k_1 = f(x_i, y_i) \qquad k_1 = f(x_i, y_i) = -2(0)^3 + 12(0)^2 - 20(0) + 8.5 = 8.5$$

$$k_2 = f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_1) \qquad k_2 = f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_1) = -2(.25)^3 + 12(.25)^2 - 20(.25) + 8.5 = 4.28125$$

$$k_3 = f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_2) \qquad k_3 = f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_2) = -2(.25)^3 + 12(.25)^2 - 20(.25) + 8.5 = 4.28125$$

$$k_4 = f(x_i + h, y_i + hk_3) \qquad k_4 = f(x_i + h, y_i + hk_3) = -2(.5)^3 + 12(.5)^2 - 20(.5) + 8.5 = 1.25$$

we then plug these in to the RK equation to find the *y* after the first step.

$$y(0.5) = 1 + (\tfrac{1}{6})0.5(8.5 + 2(4.28125) + 2(4.28125) + 1.25) = 3.24$$

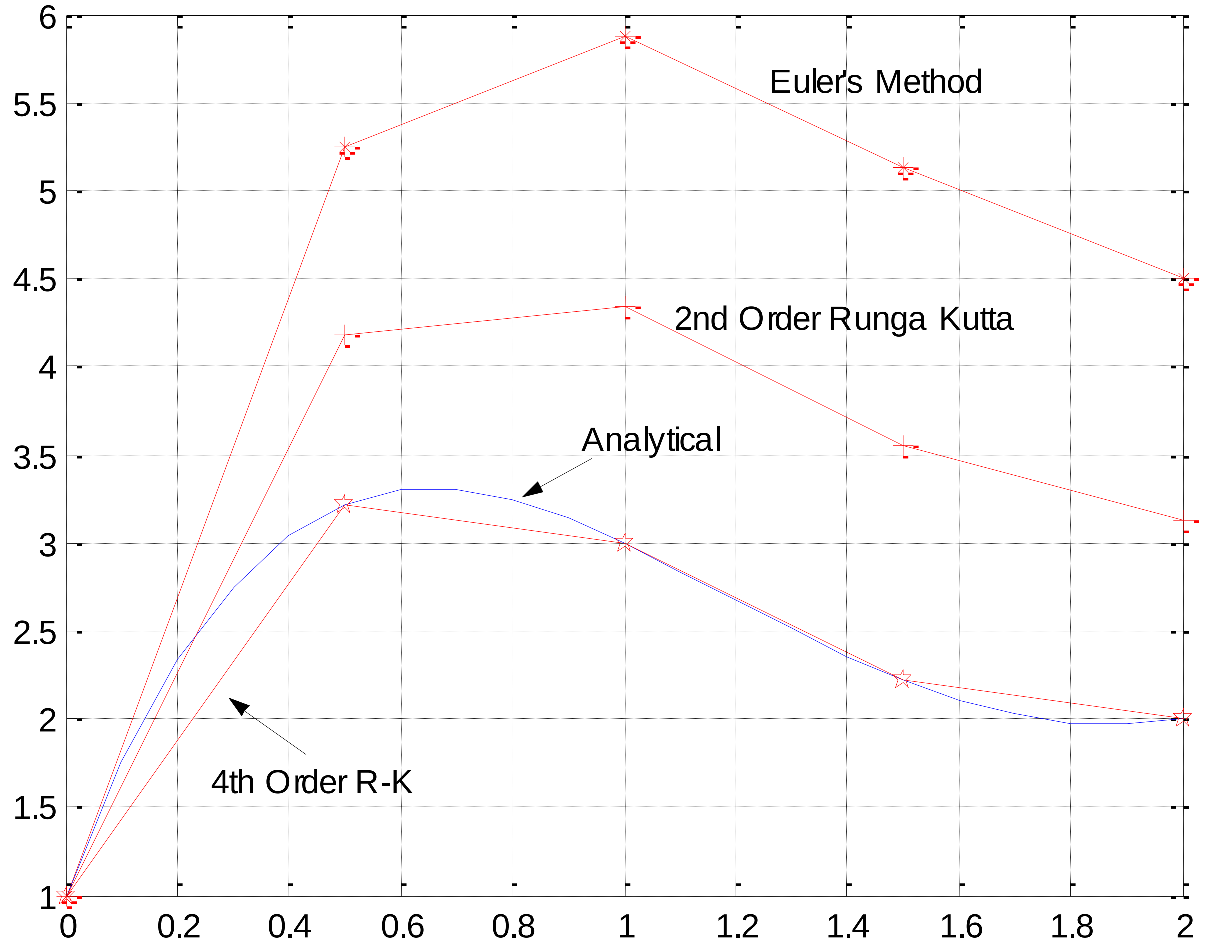
And repeat for each step

iteration	**0**	**1**	**2**	**3**	**4**
x	0	0.5	1.0	1.5	2.0
y	1	3.24	3.00	2.22	2.00

Let's see how the classical 4th compares to our previous results

Example concluded

$$f(x,y) = \frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8.5$$



In this case it tracks perfectly, and the classical 4th order RK provides (in almost all cases) good results with a reasonable interval.

Summary

- We can solve 1st order differential equations with a broad set of algorithms which we can define to estimate the slope over an interval.
- We can get very good results using the Classical 4th order Runge-Kutta method for most systems
- While these methods are computationally efficient, any complexity at all will motivate us to use MATLAB or some other programming language.
- Next up, we need to extend these methods to solve more than one 1st order ODE.
  - If we do this, then these methods will work well for any system we can represent using a state-space model.

Next Objective

- Extend these methods to solve more than one 1st order ODE.
  - If we do this, then these methods will work well for any system we can represent using a state-space model.

Solving sets of ODEs

- So far, we have learned to solve one 1st order ODE

Given: $\frac{dy}{dt} = f(t, y)$ $y(0) = y_o$ Find: $y(t) = ?$

- Are next goal is to learn to solve sets of ODES

Given:
$$\frac{dy_1}{dt} = f_1(t, y_1, y_2, \cdots, y_n)$$
$$\frac{dy_2}{dt} = f_2(t, y_1, y_2, \cdots, y_n)$$
$$\vdots$$
$$\frac{dy_n}{dt} = f_n(t, y_1, y_2, \cdots, y_n)$$

$$y_1(0) = y_{1,o}$$
$$y_2(0) = y_{2,o}$$
$$\vdots$$
$$y_n(0) = y_{\mathrm{n},o}$$

Find: $\mathbf{y}(t) = ?$

Example using Euler's Method

Given:

$$\frac{dy_1}{dx} = -0.5y_1 \qquad y_1(0) = 4$$

$$\frac{dy_2}{dx} = 4 - 0.3y_2 - 0.1y_1 \qquad y_2(0) = 6$$

note that....

$$\frac{d\mathbf{y}}{dx} = \begin{bmatrix} -0.5 & 0 \\ -0.1 & -0.3 \end{bmatrix}\mathbf{y} + \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

Find: y_1 and y_2 at x=1.0 using Euler's Method with h=0.5

Solution:

Recall Euler's Method $y_{j,i+1} = y_{j,i} + hf_j(x_{j,i}, y_{j,i})$

Iteration #1
$$\begin{cases} y_1(0.5) = 4 + 0.5(-0.5(4)) = 3 \\ y_2(0.5) = 6 + 0.5\left(4 - 0.3(6) - 0.1(4)\right) = 6.9 \end{cases}$$

Iteration #2
$$\begin{cases} y_1(1.0) = 3 + 0.5(-0.5(3)) = 2.25 \\ y_2(1.0) = 6.9 + 0.5\left(4 - 0.3(6.9) - 0.1(3)\right) = 7.715 \end{cases}$$

Example using 4RK Method

Given:

$$\frac{dy_1}{dx} = -0.5y_1 \qquad y_1(0) = 4$$

$$\frac{dy_2}{dx} = 4 - 0.3y_2 - 0.1y_1 \qquad y_2(0) = 6$$

note that....

$$\frac{d\mathbf{y}}{dx} = \begin{bmatrix} -0.5 & 0 \\ -0.1 & -0.3 \end{bmatrix}\mathbf{y} + \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

Find: y_1 and y_2 at x=0.5 using Classical 4th Order Runge-Kutta Method with h=0.5

Solution:

Recall the RK4 Method written for 2 1st order RKs

$$y_{1,i+1} = y_{1,i} + \tfrac{1}{6}h\left(k_{11} + 2k_{12} + 2k_{13} + k_{14}\right)$$

$$y_{2,i+1} = y_{2,i} + \tfrac{1}{6}h\left(k_{21} + 2k_{22} + 2k_{23} + k_{24}\right)$$

So the primary issue is accounting.

Example using 4RK Method

Recall the RK4 Method written for 2 1st order RKs

$$y_{1,i+1} = y_{1,i} + \tfrac{1}{6}h\left(k_{11} + 2k_{12} + 2k_{13} + k_{14}\right)$$
$$y_{2,i+1} = y_{2,i} + \tfrac{1}{6}h\left(k_{21} + 2k_{22} + 2k_{23} + k_{24}\right)$$

where

$$\begin{cases} k_{11} = f_1(x_i, y_{1i}, y_{2i}) = f_1(0,4,6) = -0.5(4) = -2 \\ k_{21} = f_2(x_i, y_{1i}, y_{2i}) = f_2(0,4,6) = 4 - 0.3(6) - 0.1(4) = 1.8 \end{cases}$$

$$\begin{cases} k_{12} = f_1(x_i + \tfrac{1}{2}h, y_{1i} + \tfrac{1}{2}hk_{11}, y_{2i} + \tfrac{1}{2}hk_{21}) = f_1(0.25, 3.5, 6.45) = -0.5(3.5) = -1.75 \\ k_{22} = f_2(x_i + \tfrac{1}{2}h, y_{1i} + \tfrac{1}{2}hk_{11}, y_{2i} + \tfrac{1}{2}hk_{21}) = f_2(0.25, 3.5, 6.45) = 4 - 0.3(6.45) - 0.1(3.5) = 1.715 \end{cases}$$

$$\begin{cases} k_{13} = f_1(x_i + \tfrac{1}{2}h, y_{1i} + \tfrac{1}{2}hk_{12}, y_{2i} + \tfrac{1}{2}hk_{22}) = f_1(0.25, 3.5625, 6.42875) = -1.78125 \\ k_{23} = f_2(x_i + \tfrac{1}{2}h, y_{1i} + \tfrac{1}{2}hk_{12}, y_{2i} + \tfrac{1}{2}hk_{22}) = f_2(0.25, 3.5625, 6.42875) = 1.715125 \end{cases}$$

$$\begin{cases} k_{14} = f_1(x_i + h, y_{1i} + hk_{13}, y_{2i} + hk_{23}) = f_1(0.5, 3.109375, 6.8575625) = -1.5546875 \\ k_{24} = f_2(x_i + h, y_{1i} + hk_{13}, y_{2i} + hk_{23}) = f_2(0.5, 3.109375, 6.8575625) = 1.631794 \end{cases}$$

which we plug back in…

$$y_1(0.5) = y_{1,i} + \tfrac{1}{6}h\left(k_{11} + 2k_{12} + 2k_{13} + k_{14}\right) \simeq \boxed{3.115}$$
$$y_2(0.5) = y_{2,i} + \tfrac{1}{6}h\left(k_{21} + 2k_{22} + 2k_{23} + k_{24}\right) \simeq \boxed{6.858}$$

Example using `ode45()` in MATLAB

Given:

$$\frac{dy_1}{dt} = 0.5y_1$$

$$\frac{dy_2}{dt} = -0.3y_2 - 0.1y_1 + 4\cos(t)$$

$$y_1(0) = 4$$

$$y_2(0) = 0$$

*note the sign change in the first equation

Find: y_1 and y_2 for $0 < t < 30$ using MATLAB

Solution:

MATLAB includes multiple versions of our Runge-Kutta methods including some additional features. We will focus here on ode45(), but they have similar functionality to the methods we have learned so far

```
[t,y]= ode45( @mySystem, [0 30], [ 4 0 ] );
```

Let's quickly break this down.

Example using `ode45` in MATLAB

$$\frac{dy_1}{dt} = 0.5y_1 \qquad y_1(0) = 4$$
$$\frac{dy_2}{dt} = -0.3y_2 - 0.1y_1 + 4\cos(t) \qquad y_2(0) = 0$$

A function pointer for an m-file that contains my first order ODEs

Initial condition(s)

```
[t,y]= ode45( @mySystem, [0 30], [ 4 0 ] );
```

Time span of interest

return values for independent variable and matrix of state values. Both result will have the same number of rows.

(found with `length(t)` if necessary.

$$\left\{ \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_f \end{bmatrix} \begin{bmatrix} y_{1,0} & y_{2,0} & \cdots & y_{n,0} \\ y_{1,1} & y_{2,1} & & y_{n,1} \\ \vdots & \vdots & & \vdots \\ y_{1,f} & y_{2,f} & & y_{n,f} \end{bmatrix} \right.$$

mySystem.m

```
function yprime = mySystem( t, y )

yprime = [
     0.5*y(2);
     -0.3*y(2) - 0.1*y(1) + 4*cos(t)
];
```

So what does `ode45()` do?

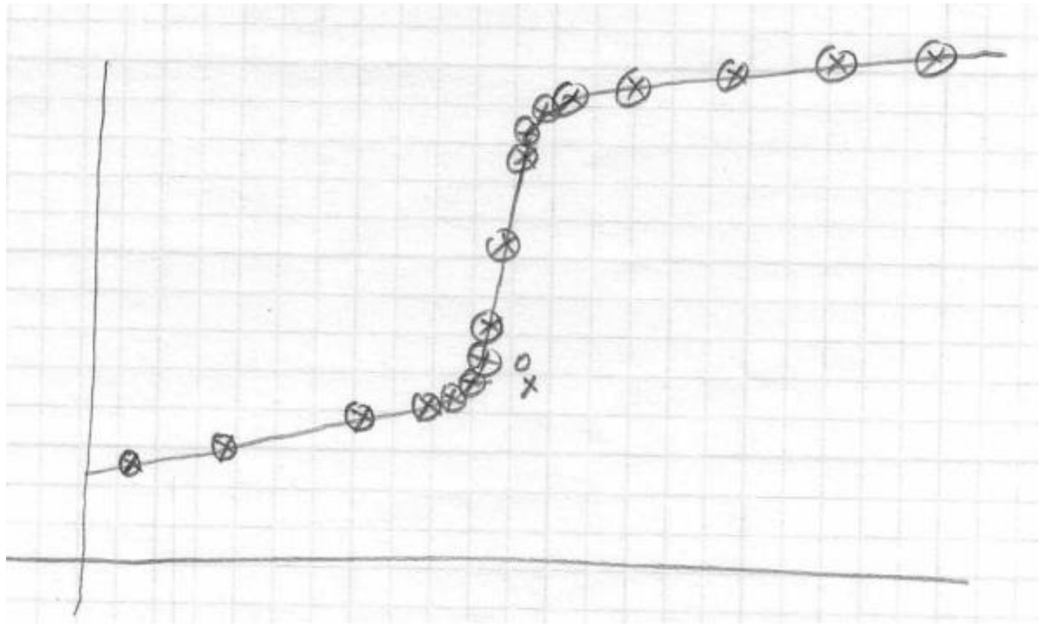
```
[t,y]= ode45( @mySystem, [0 30], [ 4 0 ] );
```

Likely performs some version of the Cash-Karp Runge-Kutta using adaptive step sizing.

$$y_{i+1} = y_i + h\left(\frac{37}{378}k_1 + \frac{250}{621}k_3 + \frac{125}{594}k_4 + \frac{512}{1771}k_6\right)$$

$$y_{i+1} = y_i + h\left(\frac{2825}{27648}k_1 + \frac{18575}{48384}k_3 + \frac{13525}{55295}k_4 + \frac{277}{14336}k_5 + \frac{1}{4}k_6\right)$$

$$Interval = \begin{bmatrix} 0 \\ .2 \\ .3 \\ .6 \\ 1 \\ .875 \end{bmatrix}$$



$$h_{new} = h_{present}\left(\frac{\Delta_{desired}}{\Delta_{actual}}\right)^{\alpha}$$

use `ODEGET` and `ODESET` to change some of `ode45()`'s internal parameters. For example

```
InitTimeStep
MaxTimeStep
```

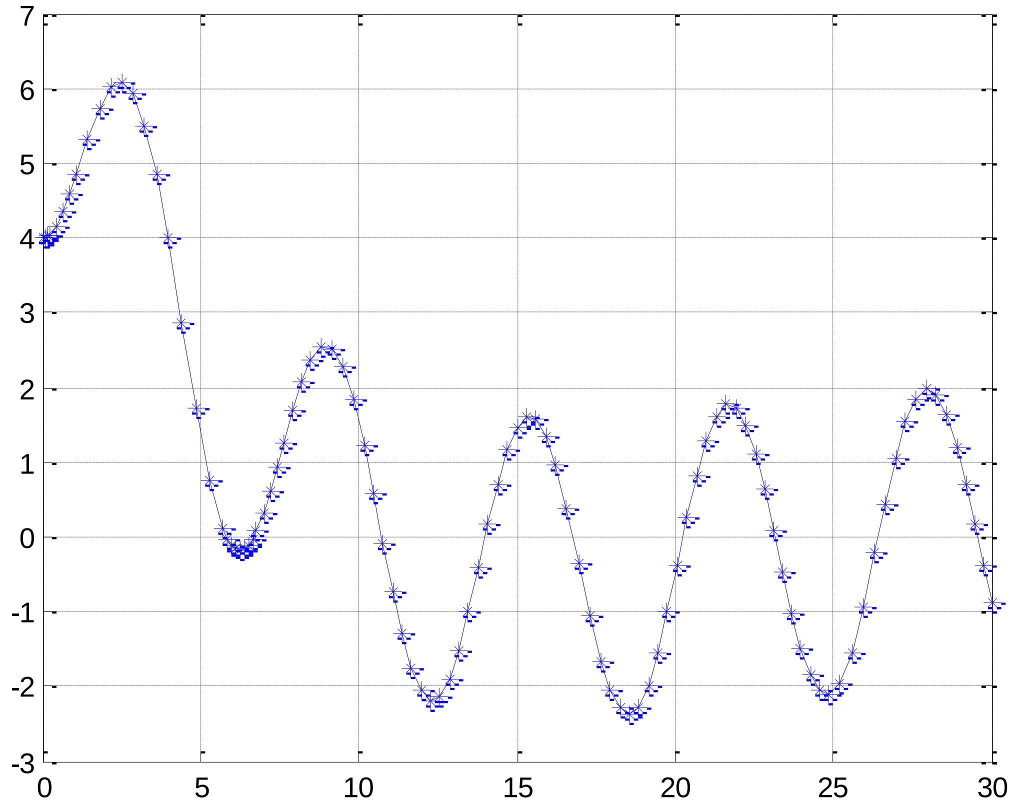
Back to our example

$$\frac{dy_1}{dt} = 0.5y_1 \qquad y_1(0) = 4$$

$$\frac{dy_2}{dt} = -0.3y_2 - 0.1y_1 + 4\cos(t) \qquad y_2(0) = 0$$

Find: y_1 and y_2 for $0 < t < 30$ using MATLAB

Solution:



```
clear all

[t, y]= ode45(@mySystem,[0 30],[ 4 0 ]);

plot(t, y(:,1))
hold on;
plot(t, y(:,1), '*')
grid on;
length(t)
```

Note the adaptive step sizing and closer values where the function has the most curvature.

Summary

- We have extended our ability to solve multiple first order ODEs
- We have a basic understanding of how MATLAB implements this class of algorithms using ode45()