

# Modelling & Simulation of Chemical Engineering Systems

٥٠١ هـم : تمثيل الأنظمة الهندسية على الحاسب الآلي

Department of Chemical Engineering  
King Saud University

# LECTURE #6

## Numerical Solution of Ordinary Differential Equations



---

# MATLAB Built-In Routines for solving ODES

- Ode113: variable order solution to nonstiff system
  - Ode15s: variable order, multistep method for solution of stiff system
  - Ode23: Lower order adaptive stepsize routine for nonstiff systems
  - Ode23s: Lower order adaptive step size routine for stiff systems
  - Ode45: higher order adaptive stepsize routine for nonstiff system
-

# Simultaneous ODEs

$$\frac{dy_1}{dt} = f_1(t, y_1, y_2)$$

RK-4 algorithm

$$\frac{dy_2}{dt} = f_2(t, y_1, y_2)$$

$$k_{1,1} = f_1(t_j, y_{j,1}, y_{j,2}),$$

$$k_{1,2} = f_2(t_j, y_{j,1}, y_{j,2}),$$

Then

$$k_{2,1} = f_1\left(t_j + h/2, (y_{j,1} + \frac{h}{2}k_{1,1}), (y_{j,2} + \frac{h}{2}k_{1,2})\right),$$

$$k_{2,2} = f_2\left(t_j + h/2, (y_{j,1} + \frac{h}{2}k_{1,1}), (y_{j,2} + \frac{h}{2}k_{1,2})\right),$$

Then

$$k_{3,1} = f_1\left(t_j + h/2, (y_{j,1} + \frac{h}{2}k_{2,1}), (y_{j,2} + \frac{h}{2}k_{2,2})\right),$$

$$k_{3,2} = f_2\left(t_j + h/2, (y_{j,1} + \frac{h}{2}k_{2,1}), (y_{j,2} + \frac{h}{2}k_{2,2})\right),$$

$$k_{4,1} = f_1\left(t_j + h/2, (y_{j,1} + hk_{3,1}), (y_{j,2} + hk_{3,2})\right),$$

$$k_{4,2} = f_2\left(t_j + h/2, (y_{j,1} + hk_{3,1}), (y_{j,2} + hk_{3,2})\right),$$

---

$$y_{j,1} = y_{j-1,1} + h\left(\frac{k_{1,1}}{6} + \frac{k_{2,1}}{3} + \frac{k_{3,1}}{3} + \frac{k_{4,1}}{6}\right)$$

$$y_{j,2} = y_{j-1,2} + h\left(\frac{k_{2,1}}{6} + \frac{k_{2,2}}{3} + \frac{k_{3,2}}{3} + \frac{k_{4,2}}{6}\right)$$

# Example: A coupled system of ODEs

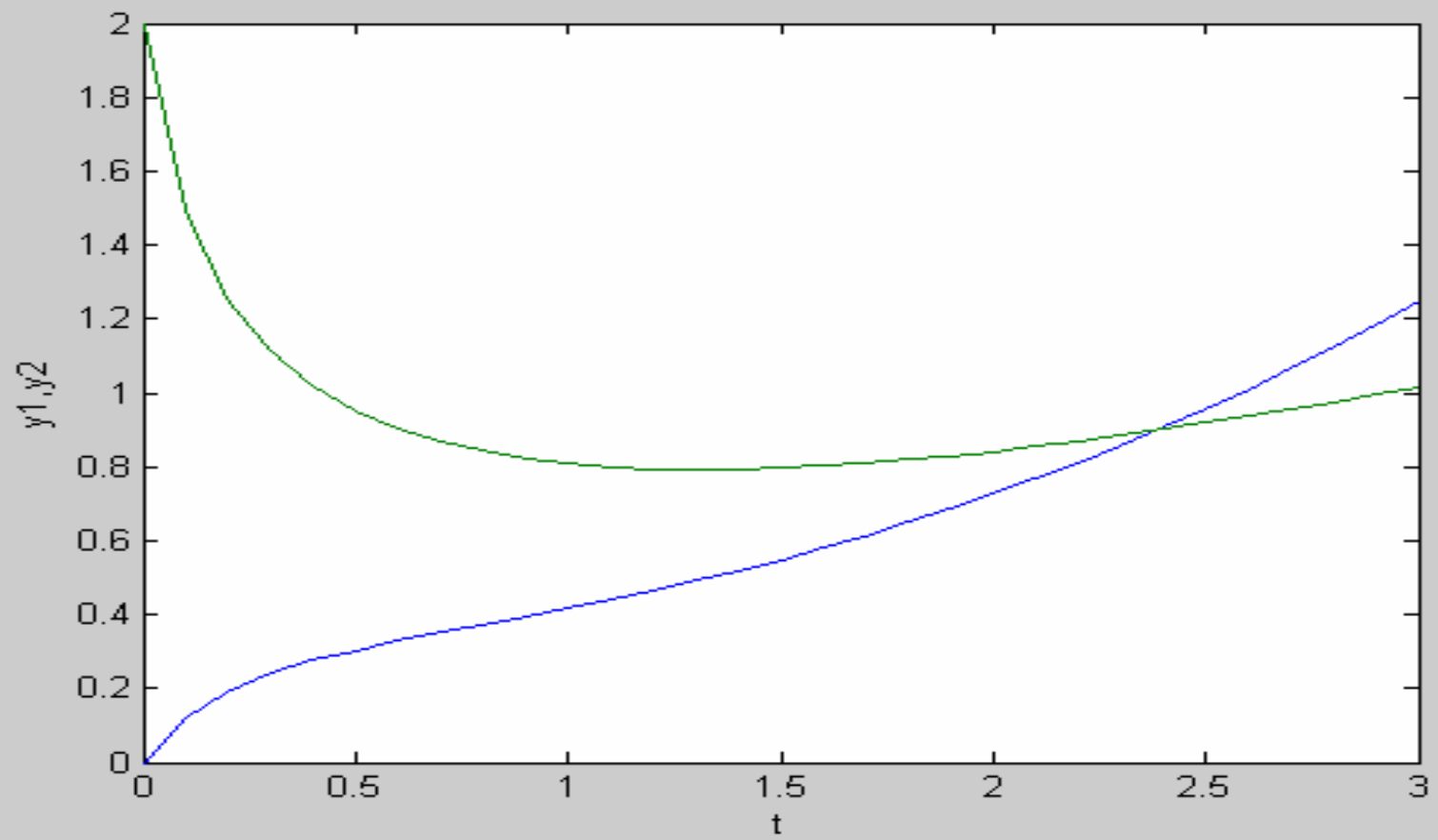
$$\frac{dy_1}{dt} = -y_1 e^{1-t} + 0.8y_2, \quad y_1(0) = 0,$$

$$\frac{dy_2}{dt} = y_1 - y_2^3, \quad y_2(0) = 2,$$

```
function dydt=odesys(t,y)
dydt=[-y(1)*exp(1-t)+0.8*y(2);
y(1)-y(2)^3];
```

```
y0=[0; 2];
tn=3;
[t,y]=RK4sys('odesys',tn,0.1,y0)
```

- function [t,y]=RK4sys(diff,tn,h,y0)
- t=(0:h:tn)'; %column vector of elements with spacing h
- nt=length(t); % Number of steps
- neq=length(y0); % number of ODEs
- y=zeros(nt,neq)
- y(1,:)=y0(:)'; %assign initial conditions
  
- h2=h/2; h3=h/3; h6=h/6;
- k1=zeros(neq,1), k2=k1; k3=k2; k4=k1;
- ytemp=k1;
- for j=2:nt
- told=t(j-1); yold=y(j-1,:);
- k1=feval(diff,told,yold);
- for n=1:neq
- ytemp(n)=yold(n)+h2\*k1(n);
- end
- k2=feval(diff,told+h2,ytemp);
- for n=1:neq
- ytemp(n)=yold(n)+h2\*k2(n);
- end
- k3=feval(diff,told+h2,ytemp)
- for n=1:neq
- ytemp(n)=yold(n)+h\*k3(n);
- end
- k4=feval(diff,told+h,ytemp)
- for n=1:neq
- y(j,n)=yold(n)+h6\*(k1(n)+k4(n))+h3\*(k2(n)+k3(n));
- end



---

### Higher order ode's

To be able to solve higher order ode's in MATLAB, they must be written in terms of a system of first order ordinary differential equations. An ordinary differential equation can be written in the form

$$\frac{d^n y}{dx^n} = f(x, y, y', y'', y''', \dots, y^{n-1})$$

and can also be written as a system of first order differential equations such that

$$y_1 = y, y_2 = y', y_3 = y'', \dots, y_n = y^{n-1}.$$

From here, the system can then be represented as an arrangement such that

$$y_1' = y_2, y_2' = y_3, \dots, y_{n-1}' = y_n, \text{ where } y_n' = f(x, y_1, y_2, \dots, y_n).$$

Example 3.3 demonstrates this technique.

---

---

```
function dmdt=Ex33(t,y)
%Solving  $Y'' + Y' + Y = 0$ 
%Assign  $Y=y(1)$ ,  $Y'=y(2)$ ,  $Y''=y(3)$ 
dYdt=y(2);
dY2dt=-(y(2)+y(1));
dmdt=[dYdt; dY2dt];
```

The variable dmdt is a 'dummy' variable that is used to describe the system as a whole,

```
clc
clf
clear
tspan=[0 10];
y0=[1 0];
[t,Y]=ode45('Ex33',tspan,y0)
plot(t,Y(:,1),'+',t,Y(:,2))
legend('Y', 'dYdt')
xlabel('t')
ylabel('Y and dY/dt')
```

The following curve is produced

---

---

# Physical Stability of dynamic systems

- Consider the single ODE

$$\frac{dy}{dx} = \lambda y, \quad y(t_o) = y_o \quad \lambda : \text{eigenvalue}$$

Steady state:  $y_{ss} = 0$

*solution :*

$$y = y_o e^{\lambda t}$$

for  $\lambda < 0$ ,  $y \rightarrow y_{ss} = 0$  for any initial condition

so we call the system is stable,

while

$$\lambda > 0, \quad y \rightarrow \infty \quad \text{unstable}$$

The dynamic system described by ODEs is stable around its steady states if the eigenvalues of the linearized system are negative

---

# Stability of system of ODEs

- Stability of system of ODEs is determined based on the sign of eigenvalues of Jacobian matrix.

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{y_{ss}}$$

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, \dots, y_n)$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, \dots, y_n)$$

⋮

$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, \dots, y_n)$$

**If all eigenvalues of  $J$  are negative the system is stable**

Example: given the chemical reaction system described by

$$\frac{dy_1}{dx} = 4y_1^2 y_2^2 - 2y_1 y_2^2 + 5y_2 - 9$$

$$\frac{dy_2}{dx} = -3y_1^3 y_2 - 12y_1 y_2^2 - 3y_2 + 9$$

- Show that  $y_1=0.07709$  and  $y_2=1.89351$  is a steady-state solution of the system
- Study the stability of the dynamic system around its steady state.
- Use an ODE MATLAB solver to examine the dynamic behavior of the system around its steady-state.

(a)

```
function f=ode_stability(y)
f(1)=4*y(1)^2*y(2)^2-2*y(1)*y(2)^2+5*y(2)-9
f(2)=-3*y(1)^3*y(2)-12*y(1)*y(2)^2-3*y(2)+9
```

```
y0=[0.07709; 1.89351]
y=fsolve('ode_stability',y0)
```

ans =

0.0771

1.8935

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 8y_1y_2^2 - 2y_2^2 & 8y_1^2y_2 - 4y_1y_2 + 5 \\ -9y_1^2y_2 - 12y_2^2 & -3y_1^2 - 24y_1y_2 - 3 \end{bmatrix}$$

■ (b)

Eigenvalues of  $J$  at steady state:

```
y0=[0.07709; 1.89351]
ys=fsolve('ode_stability',y0)
```

```
J=[8*ys(1)*ys(2)^2-2*ys(2)^2      8*ys(1)^2*ys(2)-4*ys(1)*ys(2)+5;
   -9*ys(1)^2*ys(2)-12*ys(2)^2    -2*ys(1)^2-24*ys(1)*ys(2)-3]
```

```
eig(J)
```

J =

```
-4.9596    4.5061
-43.1261   -6.5153
```

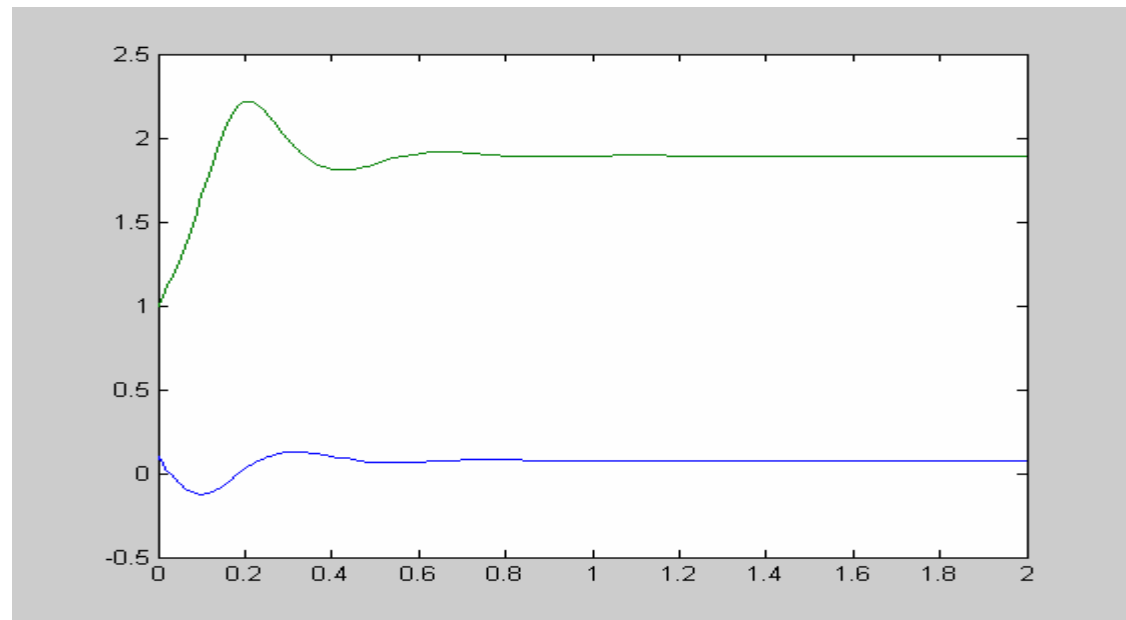
ans =

```
-5.7374 +13.9186i
-5.7374 -13.9186i
```

System is stable since all eigenvalues have negative real parts.

```
function dydx=ode_dynamic(x,y)
dydx(1)=4*y(1)^2*y(2)^2-2*y(1)*y(2)^2+5*y(2)-9
dydx(2)=-3*y(1)^3*y(2)-12*y(1)*y(2)^2-3*y(2)+9
dydx=dydx'
```

```
xspan=[0 2];
y0=[0.1; 1.0];
[x,y]=ode45('ode_dynamic',xspan,y0)
plot(x,y)
```



# Stiff ODEs

- When the eigenvalues of the Jacobian are all of the same order: non-stiff system and problem with integration.
- If  $\lambda_{\max} \gg \lambda_{\min} \rightarrow$  stiff system
- $\lambda_{\max}$  gives very steep profile and needs very small  $h$ .
- $\lambda_{\min}$  very slow dynamics and need large final time so using explicit ODE solvers is time intensive.
- For stiff systems, implicit methods are preferred.

$$SR \text{ (Stiffness Ratio)} = \frac{\max(\text{Real}(\lambda_j))}{\min(\text{Real}(\lambda_j))}$$

---

# Example of a stiff ODE

## *van der Pol's equation*

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = \mu(1 - y_1^2)y_2 - y_1$$

Given the initial conditions (1,1), use MATLAB to solve the following  
Two cases:

- $\mu=1$ , use ODE45 to solve from  $t=0$  to 20
  - $\mu=1$ , use ODE45 to solve from  $t=0$  to 3000
-

### MATLAB glossary of commands (Part III)

The sequence of this sheet hints towards the correct sequence for solving odes in matlab

1. The different ODE solvers:

Solver	Solves These Kinds of Problems	Method
<a href="#">ode45</a>	Nonstiff differential equations	Runge-Kutta
<a href="#">ode23</a>	Nonstiff differential equations	Runge-Kutta
<a href="#">ode113</a>	Nonstiff differential equations	Adams
<a href="#">ode15s</a>	Stiff differential equations and DAEs	NDFs (BDFs)
<a href="#">ode23s</a>	Stiff differential equations	Rosenbrock
<a href="#">ode23t</a>	Moderately stiff differential equations and DAEs	Trapezoidal rule
<a href="#">ode23tb</a>	Stiff differential equations	TR-BDF2

2. `global v1 v2 vi`

Defines global variables `v1`, `v2`, and `vi`. When required, the `global` command is used in all `m` files to declare common variables.

3. `tspan [to tf]`

Sets integration interval. `to` and `tf` must be defined prior to `tspan`. `tspan` is an arbitrary name for the row vector containing `to` and `tf`, which are also arbitrarily named.

4. `y0=[a0, a1, a2 ...]`

Initial conditions for each dependent variable in a 1<sup>st</sup> order differential equation. `y0` is an arbitrary name. A single 1<sup>st</sup> order DEQ requires only one initial condition, and each independent DEQ thereafter requires another.

5. `[Indepent, dependentbase] = ode#('filename', tspan, y0)`

Syntax to instruct MATLAB to use ODE solver called `ode#` to solve the function defined in file called `filename`, using the initial conditions described in `y0`. “#” is the number for a particular solver; generally `ode45` works as an initial try.

6. `plot (t, y(:,1), t, y(:,2), t, ...)`

Invokes `plot` command to plot `t` versus variables defined in function as `y(1)`, `y(2)`, etc.

7 Plot formatting commands