

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



Example 3.1

A fluid of constant density starts to flow into an empty and infinitely large tank at 8 L/s. A valve regulates the outlet flow to a constant 4L/s. Derive and solve the differential equation describing this process over a 100 second interval.

Solution

The accumulation is described as input – output, so the ode describing the process becomes $\frac{d(\rho V)}{dt} = (8 - 4)\rho$. Since density is constant, then $\frac{dv}{dt} = 8 - 4 = 4$ in liters per second. The initial condition is that at time $t=0$, the volume inside the tank $=0$. The following function file 'ex31' is used to set up the ode solver.

```
function dvdt=ex31(t,v)
dvdt=4
```

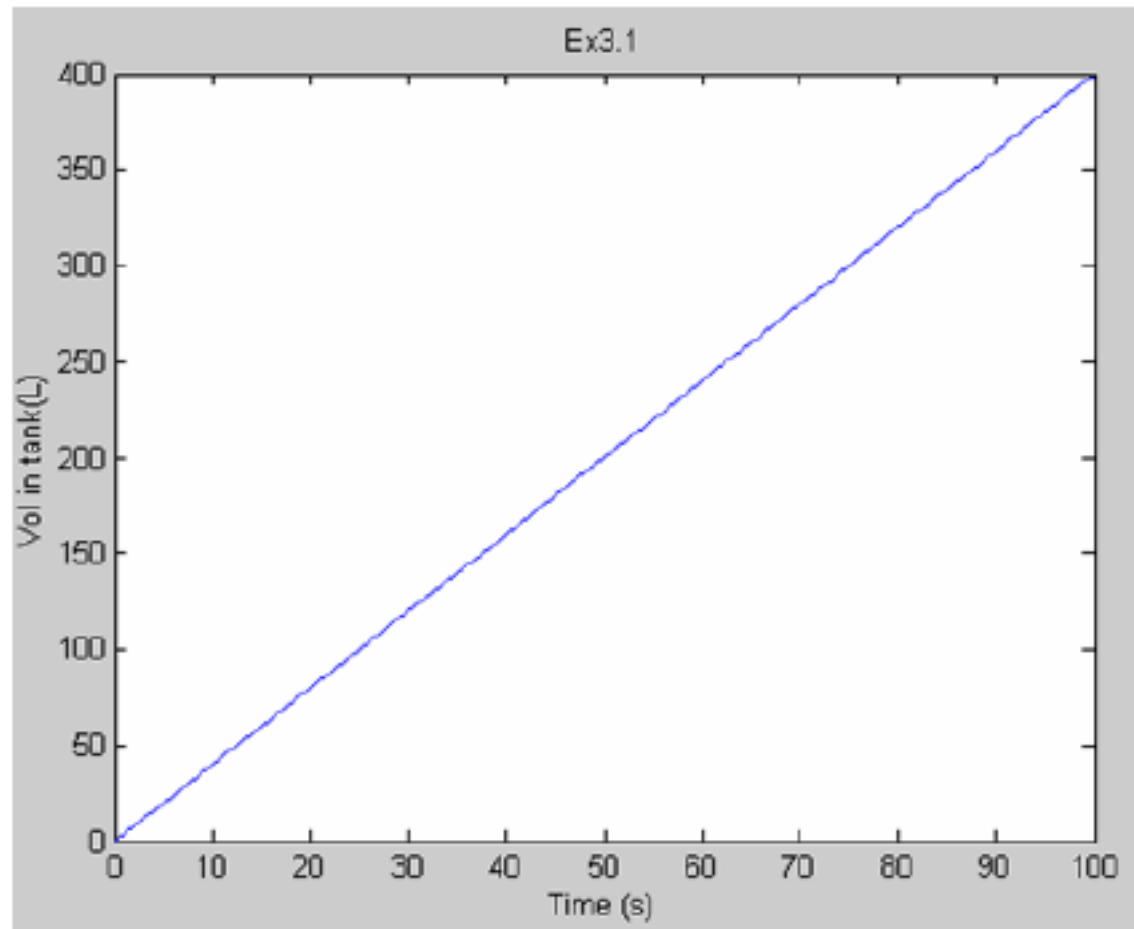
The file ex31run is used to execute the solver. The code for this file is overleaf.

```
t0=0;
tf=100;
tspan=[t0 tf]; %Integration interval
v0=0 %Initial condition

[t,v]=ode45('ex31',tspan,v0)

plot(t,v(:,1))
xlabel('Time (s)')
ylabel('Vol in tank(L)')
title('Ex3.1')
```

The plot produced is



The following set of differential equations describes the change in concentration three species in a tank. The reactions $A \rightarrow B \rightarrow C$ occur within the tank. The constants k_1 , and k_2 describe the reaction rate for $A \rightarrow B$ and $B \rightarrow C$ respectively. The following ode's are obtained:

$$\frac{dC_a}{dt} = -k_1 C_a$$

$$\frac{dC_b}{dt} = k_1 C_a - k_2 C_b$$

$$\frac{dC_c}{dt} = k_2 C_b$$

Where $k_1=1 \text{ hr}^{-1}$ and $k_2=2 \text{ hr}^{-1}$ and at time $t=0$, $C_a=5\text{mol}$ and $C_b=C_c=0\text{mol}$. Solve the system of equations and plot the change in concentration of each species over time. Select an appropriate time interval for the integration.

Solution

The following function file and run file are created to obtain the solution:

```
function dcdt=Ex32(t,c)
%c(1)=ca, c(2)=cb, c(3)=cc
global k1 k2
dcdt=[-k1*c(1); k1*c(1)-k2*c(2); k2*c(2)];
```

Ca, Cb and Cc must be defined within the same matrix, and so by calling Ca c(1), Cb c(2) and Cc as c(3), they are listed as common to matrix c.

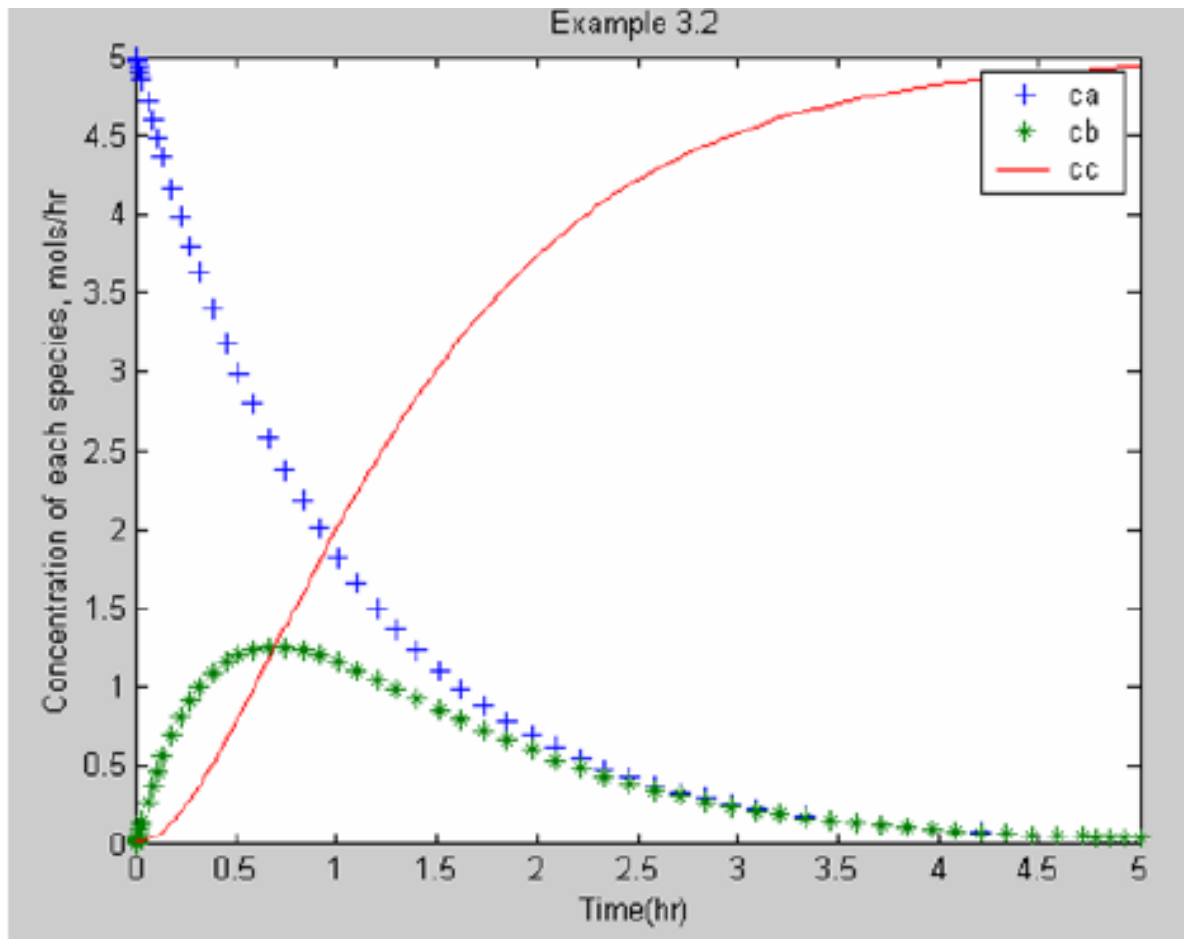
```
clc
clf
clear
global k1 k2
k1=1;
k2=2;

tspan=[0 5];

c0=[5 0 0];

[t,c]=ode45('Ex32',tspan,c0)

plot(t,c(:,1),'+',t,c(:,2),'*',t,c(:,3))
legend('ca','cb','cc')
xlabel('Time(hr)')
ylabel('Concentration of each species, mols/hr')
title('Example 3.2')
```



Nonlinear ODE: Initial value Problems

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0$$

$$\int_{y_i}^{y_{i+1}} dy = \int_{x_i}^{x_{i+1}} f(x, y) dx \rightarrow y_{i+1} - y_i = \int_{x_i}^{x_{i+1}} f(x, y) dx$$

Approximated using Finite differences

Euler Method

$$y_{i+1} = y_i + \overset{\text{Step size, } \Delta x}{h} f(x_i, y_i)$$

Implicit Euler Method

$$y_{i+1} = y_i + hf(x_{i+1}, y_{i+1})$$

Modified Euler method

$$y_{i+1,pr} = y_i + hf(x_i, y_i)$$

$$y_{i+1,cor} = y_i + hf(x_{i+1}, y_{i+1})$$

■ Example: Use Euler's method to integrate

$$\frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8.5, \quad y(0) = 1.0$$

Step size=0.5 *Solution :*

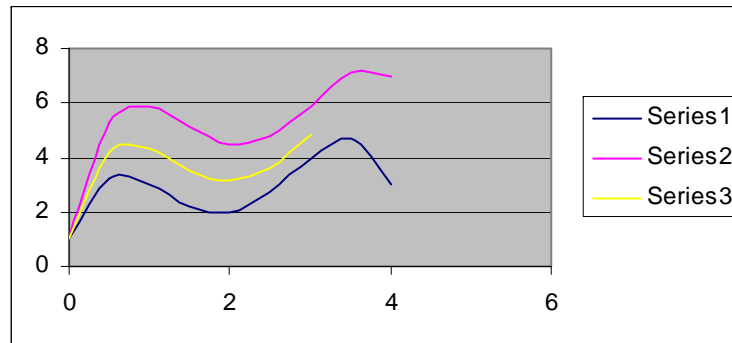
$$y_1 = y_0 + f(x_0, y_0) \cdot 0.5$$

$$y(0.5) = y(0) + f(0,1) \cdot 0.5 = 1 + \{-2(0)^3 + 12(0)^2 - 20(0) + 8.5\} \cdot 0.5$$

$$y(0.5) = 1 + 8.5 \cdot 0.5 = 5.25$$

$$y(1) = y(0.5) + f(0.5, 5.25) \cdot 0.5 = 5.875$$

x	y exact	y Euler, h=0.5
0	1	1
0.5	3.21875	5.25
1	3	5.875
1.5	2.21875	5.125
2	2	4.5
2.5	2.71875	4.75
3	4	5.875
3.5	4.71875	7.125
4	3	7



x	y Euler, h=0.25
0	1
0.25	3.125
0.5	4.1796875
0.75	4.4921875
1	4.34375
1.25	3.96875
1.5	3.5546875
1.75	3.2421875
2	3.125
2.25	3.25
2.5	3.6171875
2.75	4.1796875
3	4.84375



-
- Euler method is a first-order one-step method
 - Error can be reduced by reducing step size.
 - Demands great computational effort to obtain acceptable error levels
 - Euler method can provide error-free predictions if the underlying function is linear
 - Truncation error:

$$y_{i+1} = y_i + f(x_i, y_i)h + \frac{f'(x_i, y_i)}{2!}h^2 + \dots + \frac{f^{n-1}(x_i, y_i)}{n!}h^n + O(h^{n+1})$$

$$E_t = \frac{f'(x_i, y_i)}{2!}h^2 + \dots + \frac{f^{n-1}(x_i, y_i)}{n!}h^n + O(h^{n+1})$$

$$E_t = \frac{f'(x_i, y_i)}{2!}h^2, \quad \text{for small } h$$

MATLAB Implementation

- The OdeEuler function for integration of a first-order ODE with Euler's explicit method

```
function [t,y]=odeEuler(diff,tn,h,y0)
t=(0:h:tn)';
n=length(t);
y=y0*ones(n,1)
for j=2:n
    y(j)=y(j-1)+h*feval(diff,t(j-1),y(j-1));
end
```

$$\frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8.5, \quad y(0) = 1.0$$

- The ge501_euler function for comparing numerical solution with Eulers method solution with the exact solution.

```
function ge501_euler(h)
tn=4; y0=1;
[x,y]=odeEuler('ypge501',tn,h,y0)
fprintf('  x   y   y_Exact   error\n')
for k=1: length(x)
    yexact(k)=-0.5*x(k)^4 + 4*x(k)^3-10*x(k)^2+8.5*x(k)+1;
end
for k=1: length(x)
fprintf('%9.4f %9.6f %9.6f %10.2e\n', x(k), y(k), yexact(k), y(k)-yexact(k))
end
fprintf('\nMax Error= %10.2e for h= %f\n',norm(y-yexact'),h);
```

```
function dydx=ypge501(x,y)
dydx=-2*x^3+12*x^2-20*x+8.5
```

Implicit Euler Method

$$y_{i+1} = y_i + hf(x_{i+1}, y_{i+1}) + O(h^2)$$

Implicit Equations cannot be solved individually but must be setup as sets of algebraic equations.

Euler implicit formula is more stable than the explicit one.

Higher-order One step Methods

- Midpoint method

$$k_1 = f(t_{j-1}, y_{j-1})$$

$$y_{i-1/2} = y_{j-1} + \frac{h}{2} f(t_{j-1}, y_{j-1})$$

$$k_2 = f\left(t_{j-1} + \frac{h}{2}, y_{j-1} + \frac{h}{2} k_1\right)$$

$$y_j = y_{j-1} + hk_2$$

- Heun's method

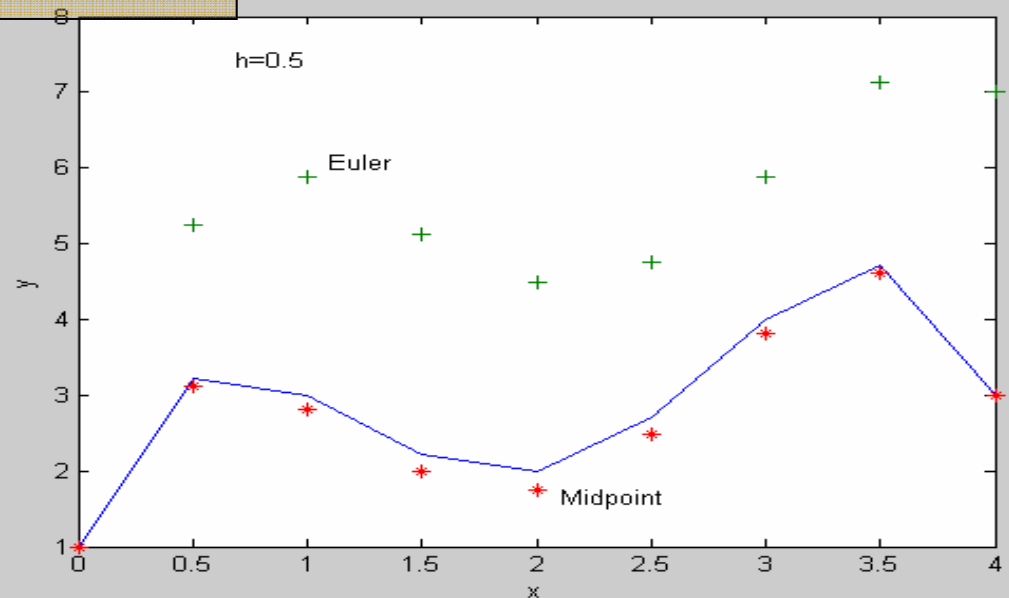
$$k_1 = f(t_{j-1}, y_{j-1})$$

$$k_2 = f(t_{j-1} + h, y_{j-1} + hk_1)$$

$$y_j = y_{j-1} + h \frac{k_1 + k_2}{2}$$

Comparison of Midpoint and Euler methods

```
function [t,y]=odeMidpt(diff,tn,h,y0)
t=(0:h:tn)';
n=length(t);
y=y0*ones(n,1)
h2=h/2
for j=2:n
    k1=feval(diff,t(j-1),y(j-1))
    k2=feval(diff,t(j-1)+h2,y(j-1)+h2*k1)
    y(j)=y(j-1)+h*k2;
end
```



MATLAB code for ODE simulation

- function ge501_euler(h)
 - tn=4; y0=1;
 - [x,y]=odeEuler('ypge501',tn,h,y0)
 - [x1,y_mp]=odeMidpt('ypge501',tn,h,y0)
 - [x1,y_Huen]=odeHuen('ypge501',tn,h,y0)

 - fprintf(' x y ymidpoint y_Huen y_Exact error\n')
 - for k=1: length(x)
 - yexact(k)=-0.5*x(k)^4 + 4*x(k)^3-10*x(k)^2+8.5*x(k)+1;
 - end
 - for k=1: length(x)
 - fprintf('%9.4f %9.6f %9.6f %9.6f %9.6f %10.2e\n', x(k), y(k), y_mp(k), y_Huen(k), yexact(k), y(k)-yexact(k))
 - end
 - fprintf('\nMax Error= %10.2e for h= %f\n',norm(y-yexact'),h);
 - plot(x,yexact,x,y,'+',x1,y_mp,'*')
-

Fourth-order Runge-Kutta Method

■ General formula:
$$y_j = y_{j-1} + \sum_{l=1}^m \gamma_l k_l$$

RK-4

$$k_1 = f(t_{j-1}, y_{j-1})$$

$$k_2 = f\left(t_{j-1} + \frac{h}{2}, y_{j-1} + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(t_{j-1} + \frac{h}{2}, y_{j-1} + \frac{h}{2}k_2\right)$$

$$k_4 = f(t_{j-1} + h, y_{j-1} + hk_3)$$

$$y_j = y_{j-1} + h\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right)$$

-
- function [t,y]=RK4(diff,tn,h,y0)
 - t=(0:h:tn)';
 - n=length(t);
 - y=y0*ones(n,1)
 - h2=h/2; h3=h/3; h6=h/6;
 - for j=2:n
 - k1=feval(diff,t(j-1),y(j-1))
 - k2=feval(diff,t(j-1)+h2,y(j-1)+h2*k1)
 - k3=feval(diff,t(j-1)+h2,y(j-1)+h2*k2)
 - k4=feval(diff,t(j-1)+h,y(j-1)+h*k3)
 - y(j)=y(j-1)+h6*(k1+k4)+h3*(k2+k3);
 - end
-

ODE methods Comparison

x	y-Euler	y-Midpoint	y-Heun	y-RK4	Exact	
0.0000	1.000000	1.000000	1.000000	1.000000	1.000000	0.00e+000
0.5000	5.250000	3.109375	3.437500	3.218750	3.218750	2.03e+000
1.0000	5.875000	2.812500	3.375000	3.000000	3.000000	2.88e+000
1.5000	5.125000	1.984375	2.687500	2.218750	2.218750	2.91e+000
2.0000	4.500000	1.750000	2.500000	2.000000	2.000000	2.50e+000
2.5000	4.750000	2.484375	3.187500	2.718750	2.718750	2.03e+000
3.0000	5.875000	3.812500	4.375000	4.000000	4.000000	1.88e+000
3.5000	7.125000	4.609375	4.937500	4.718750	4.718750	2.41e+000
4.0000	7.000000	3.000000	3.000000	3.000000	3.000000	4.00e+000

Multi-Step Methods

- Adams method (2nd degree)

$$y_{i+1} = y_i + \frac{h}{12}(23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}) + 5f(x_{i-2}, y_{i-2})) + O(h^4)$$

- Adams-Moulton method (3rd degree)

$$(y_{i+1})_{pr} = y_i + \frac{h}{24}(55f(x_i, y_i) - 59f(x_{i-1}, y_{i-1}) + 37f(x_{i-2}, y_{i-2}) - 9f(x_{i-3}, y_{i-3})) + O(h^5)$$

$$(y_{i+1})_{cor} = y_i + \frac{h}{24}(9f(x_{i+1}, y_{i+1,pr}) + 19f(x_i, y_i) - 5f(x_{i-1}, y_{i-1}) + f(x_{i-2}, y_{i-2})) + O(h^5)$$

Adaptive Stepsize Algorithms

- Two basic strategies to increase the accuracy:
 - Reduce h
 - Choose a more accurate scheme (higher order)
 - Success doesn't continue above order four RK
 - For further efficiency, adapt varying h

MATLAB adaptive stepsize algorithm: ode23,
ode45

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
-