ODE Solvers

Department of Chemical Engineering
King Saud University

Numerical Solution of Ordinary Differential Equations

ODE Classifications

- Order
- Linearity
- Boundary or Initial

Nonlinear

3rd order $\frac{d^3y}{dx^3} + \left(\frac{dy}{dx}\right)^5 = kx$

y: dependent Variable
x: Independent variable

General Form of linear ODE

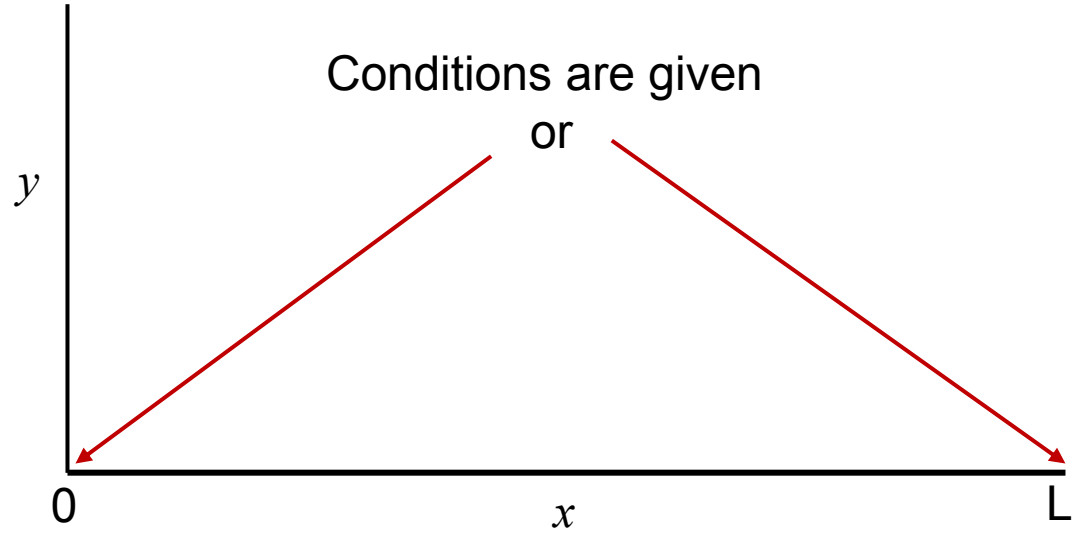
$$b_o(x)\frac{d^n y}{dx^n} + b_1(x)\frac{d^{n-1}y}{dx^{n-1}} + \ldots\ldots + b_{n-1}(x)\frac{d\ y}{dx} + b_n(x)y = R(x)$$

R(x)=0 → Homogeneous ODE

If b_o, b_1,..,b_n don't depend on x and R(x)=0→ autonomous system

ODE Solution

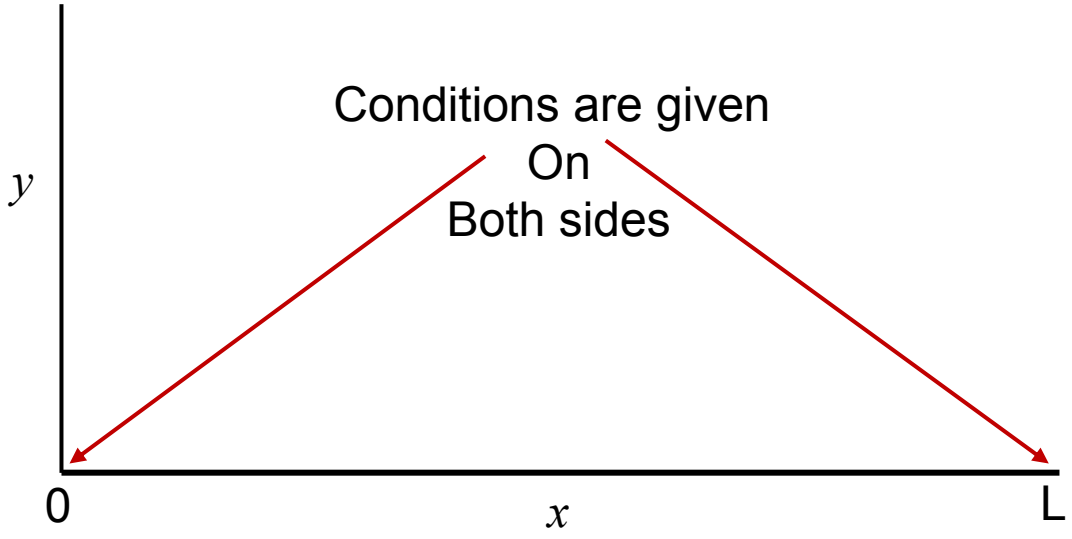
- To obtain a unique solution for an nth order ODE we need to specify n values for y or its derivatives at specific value of x → initial or boundary conditions.
- Initial-value problem: *y*, *dy/dx*, .are given at initial value of x or at the terminal value of x.



ODE Solution

- Boundary-value problem: some of *y, dy/dx*, ….are given at initial value of x and others at the end value of *x*

  [*y (x=0), dy/dx(x=1)]*



ODE Solution

- To obtain a unique solution for an nth order ODE we need to specify n values for y or its derivatives at specific value of x → initial or boundary conditions.
- Initial-value problem: y, dy/dx, ….are given at initial value of x.

- Boundary-value problem: some of y, dy/dx, ….are given at initial value of x and others at the end value of x

  [y (x=0), dy/dx(x=1)]

Canonical Form

ODE of nth order → n simultaneous first order ODEs: Canonical Form

Given:

$$\frac{dz^n}{dx^n} = G\left(z, \frac{dz}{dx}, \frac{d^2z}{dx^2}, \cdots, \frac{d^nz}{dx^n}, x\right)$$

Let

$$z = y_1, \frac{dz}{dx} = y_2, \frac{d^2z}{dx^2} = y_3, \cdots\cdots, \frac{d^{n-1}z}{dx^{n-1}} = y_n$$

Canonical Form

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

$$\frac{dy_2}{dx} = y_3$$

$$\vdots$$

$$\vdots$$

$$\frac{dy_{n-1}}{dx} = y_n$$

$$\frac{dy_n}{dx} = G\left(y_1, y_2, \cdots, y_{n-1}, x\right)$$

If G doesn't depend on x → Autonomous system

Example

$$\frac{d^4z}{dt^4}+5\frac{d^3z}{dt^3}-2\frac{d^2z}{dt^2}-6\frac{dz}{dt}+3z=0$$

$$Let: \quad z=y_1 \quad \frac{dz}{dt}=y_2 \quad \frac{d^2z}{dt^2}=y_3 \quad \frac{d^3z}{dt^3}=y_4 \quad \frac{d^4z}{dt^4}=\frac{dy_4}{dt}$$

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

$$\frac{dy_2}{dt}=y_3$$

$$\frac{dy_3}{dt}=y_4$$

$$\frac{dy_4}{dt}=-5y_4+2y_3+6y_2-3y_1$$

Canonical Form

$$\begin{bmatrix} \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \\ \frac{dy_3}{dt} \\ \frac{dy_4}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 6 & 2 & -5 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \Rightarrow \frac{dY}{dt}=AY$$

Solution of Linear ODE

For single ODE:

$$\frac{dy}{dt} = ay, \qquad y(0) = y_o, \qquad a: \text{ constant}$$

$$solution: \qquad y(t) = e^{at} y_o$$

For system of ODE's (Matrix form)

$$Y' = AY, \qquad Y(0) = Y_o$$

$$Solution: \qquad Y = e^{At} Y_o,$$

$$e^{At} = I + At + \frac{A^2 t^2}{2!} + \frac{A^3 t^3}{3!} + \cdots\cdots$$

Eigenvalue-Eigenvector Method

$$e^{At} = X e^{\Lambda t} X^{-1}$$

$$e^{\Lambda t} = \begin{bmatrix} e^{\lambda_1 t} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2 t} & \cdots & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & e^{\lambda_n t} \end{bmatrix}, \qquad X = [x_1, x_2, \cdots x_n]$$

λ_i : eigenvalues of A (scalar)

x_i : Eigenvactor of A (vector)

$$Ax_i = \lambda_i x_i$$

Problem Type: Nonstiff

Solver	Accuracy	When to Use
ode45	Medium	Most of the time. ode45 should be the first solver you try.
ode23	Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113	Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to valuate.

Problem Type: Stiff

Solver	Accuracy	When to Use
ode15s	Low to Medium	Try ode15s when ode45 fails or is inefficient and you suspect that the problem is stiff. Also use ode15s when solving differential algebraic equations (DAEs).
ode23s	Low	ode23s can be more efficient than ode15s at problems with crude error tolerances. It can solve some stiff problems for which ode15s is not effective. ode23s computes the Jacobian in each step, so it is beneficial to provide the Jacobian via odeset to maximize efficiency and accuracy.If there is a mass matrix, it must be constant.
ode23t	Low	Use ode23t if the problem is only moderately stiff and you need a solution without numerical damping. ode23t can solve differential algebraic equations (DAEs).
ode23tb	Low	Like ode23s, the ode23tb solver might be more efficient than ode15s at problems with crude error tolerances.

ode45

Syntax

[t,y] = odeXY(odefun,tspan,y0)

example

[t,y] = odeXY(odefun,tspan,y0,options)

example

[t,y,te,ye,ie] = odeXY(odefun,tspan,y0,options)

[t,y,te,ye,ie] = ode45(odefun,tspan,y0,options)
additionally finds where functions of (t,y), called event functions, are zero. In the output,
te: the time of the event,
ye: the solution at the time of the event
ie: the index of the triggered event.

options — Option structure
structure array
Option structure, specified as a structure array. Use the odeset function to create or modify the options structure. See Summary of ODE Options for a list of the options compatible with each solver.

Example:

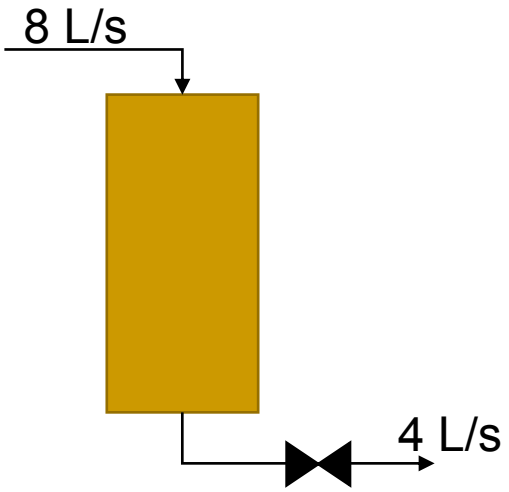
```
options = odeset('RelTol',1e-5,'Stats','on','OutputFcn',@odeplot)
```

RelTol: specifies a relative error tolerance of 1e-5,
Stats: turns on the display of solver statistics,
OutputFcn: specifies the output function @odeplot to plot the solution as it is computed.

Data Types: struct

ODE example I

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.



ODE example I

Solution

The accumulation is described as input – output, so the ode describing the process becomes

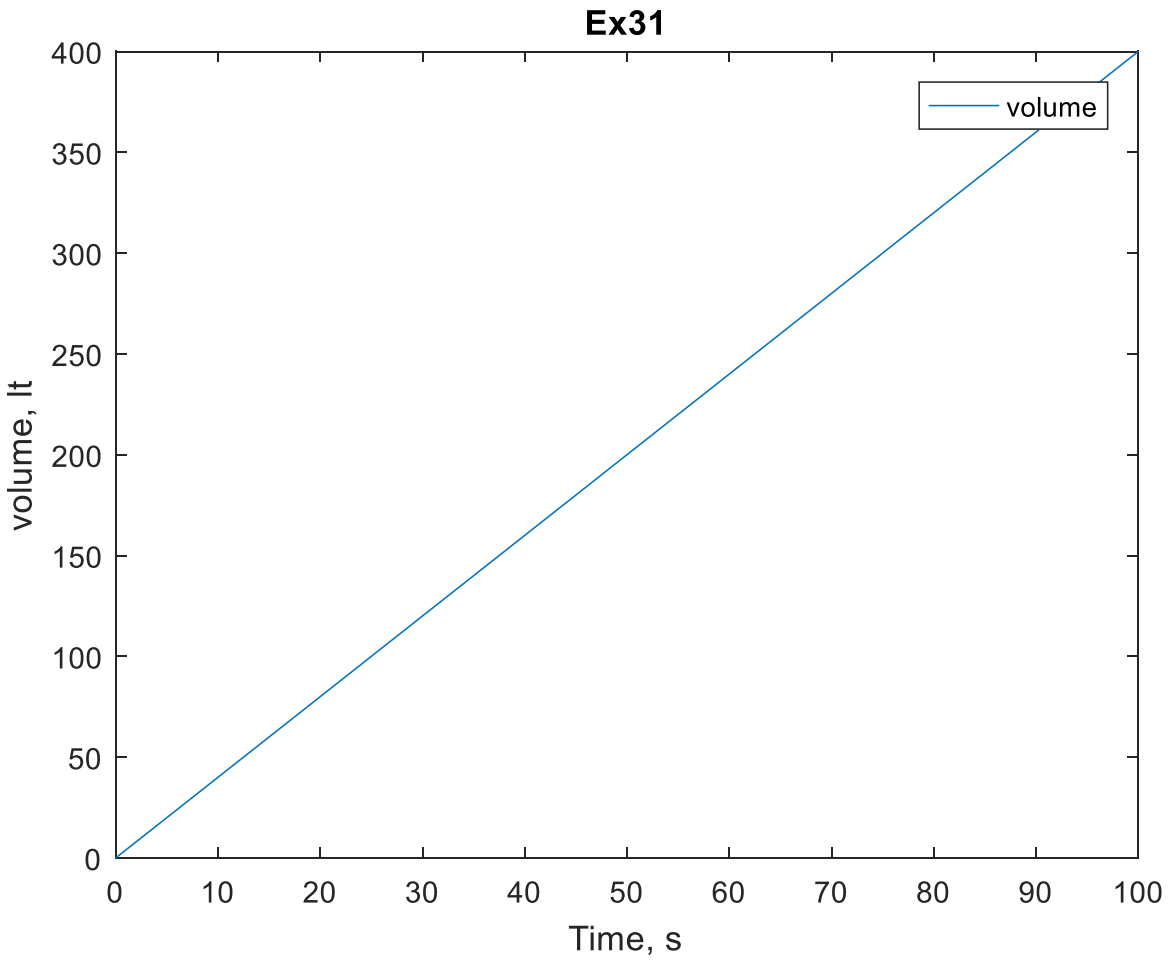
$$\frac{d(\rho V)}{dt} = (8-4)\rho$$

Since density is constant, then $\frac{d(V)}{dt} = (8-4) = 4$ 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.

ODE example I

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

clc; clf; clear

t0 = 0 ;
v0 = 0 ;
tf = 100 ;
tspan = [t0 tf] ;

[t v]=ode45('ex31',tspan,v0)
plot(t,v(:,1))
legend('volume')
xlabel('Time, s')
ylabel('volume, lt')
title('Ex31')
```

ODE example I



ode45

```
sol = ode45(___)
```

sol = ode45(___) returns a structure that you can use with deval to evaluate the solution at any point on the interval [t0 tf].

Example 2: Batch Reactor

given that the concentration of a species A in a constant volume, batch reactor obeys this differential equation $\frac{dC_A}{dt} = -kC_A^2$

with the initial condition $C_A(t=0) = 2.3$ mol/L and $k = 0.23$ L/mol/s, compute the time it takes for C_A to be reduced to 1 mol/L.

Solution:

we will use Matlab to integrate the ODE to obtain the solution, and then use the deval command in conjuction with fsolve to get the numerical solution we are after.

Example : Batch Reactor

```
clear all; close all; clc
global k % variables that we wish to share with the main script
k = 0.23;
t0 = 0 ;
Cao = 2.3; % mol/l
tf = 10
tspan = [t0 tf];
[t, Ca] = ode45(@Batch,tspan,Cao) ;
% Plot the solution generated by the sol structure
plot(t,Ca) % x:time, y:Concentration
%plot(sol.x,sol.y) % x:time, y:Concentration
xlabel('Time (s)')
ylabel('C_A (mol/L)')
% the concentration of A is at approximately t=2.5 sec.
```

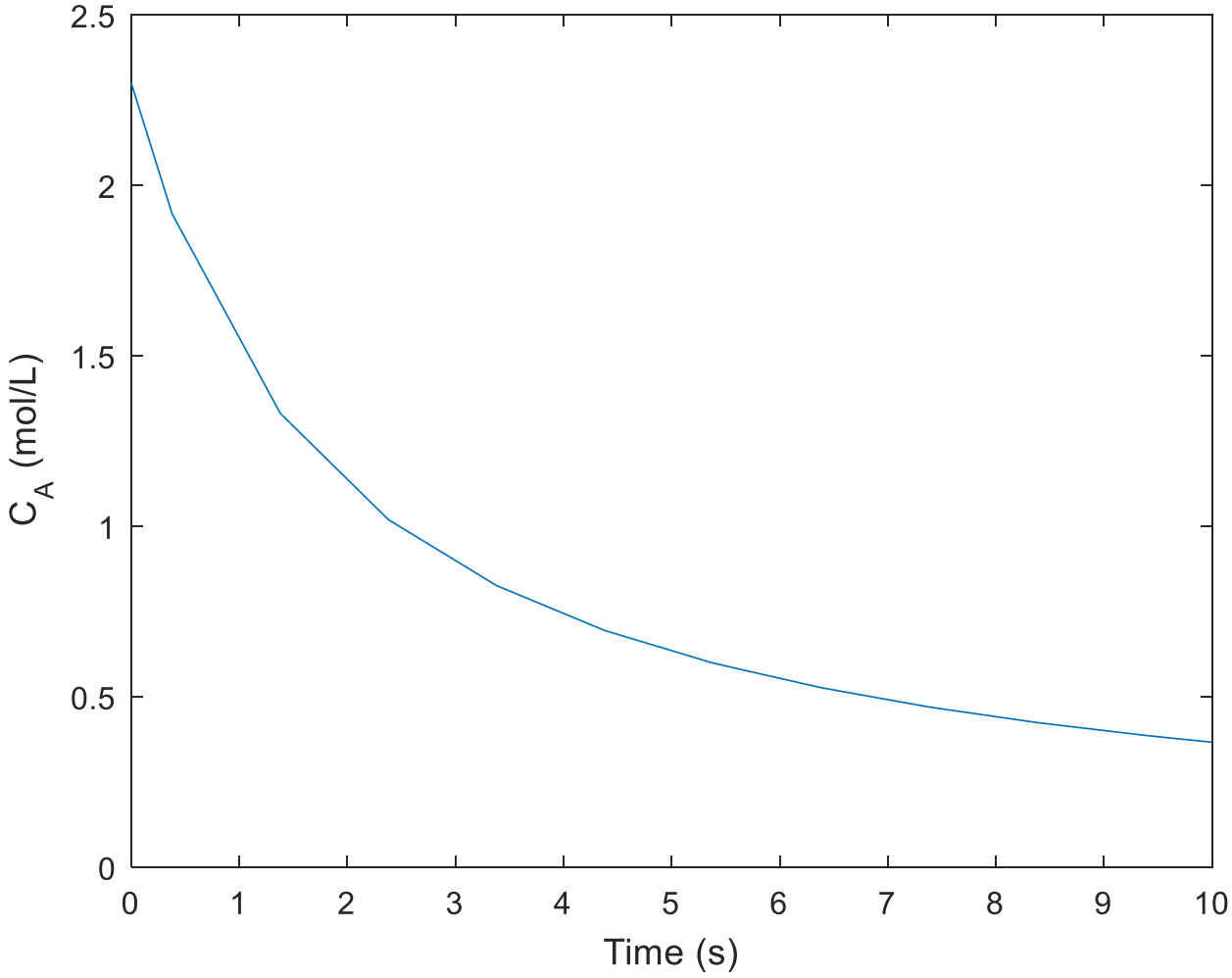
Example : Batch Reactor

```
% Plot the solution generated by the sol structure
plot(t,Ca) % x:time, y:Concentration
%plot(sol.x,sol.y) % x:time, y:Concentration
xlabel('Time (s)')
ylabel('C_A (mol/L)')
% the concentration of A is at approximately t=3


 function dCdt = Batch(t,Ca)
 global k % variables that we wish to share with the
 main script
 dCdt = -k*Ca^2;
 end
```

Example 2: Continue

```
sol =

    solver: 'ode45'
   extdata: [1x1 struct]
         x: [0 0.3799 1.3799 2.3799 3.3799 4.3799 5.3799 6.3799 7.3799 8.3799
9.3799 10]
         y: [2.3000 1.9152 1.3304 1.0187 0.8254 0.6937 0.5982 0.5259 0.4691 0.4234
0.3859 0.3657]
     stats: [1x1 struct]
     idata: [1x1 struct]
```

Example 2: Continue



Generating Algebraic Function of ode

```
clear all; close all; clc
global k % variables that we wish to share with the main script
k = 0.23; Cao = 2.3;
% Batch = @(t,Ca) -k*Ca^2;
tspan = [0 10];
% [t, Ca] = ode45(Batch,tspan,Cao)
%[t, Ca] = ode45(@Batch,tspan,Cao)
sol = ode45(@dCadt,tspan,Cao)
sol here is a struct with fields corresponding to the solution and
other outputs of the solver.
```

Generating Algebraic Function of ode

```
% Generate Algebraic function out of sol structure
Ca1 = 1 ; % We seek the time at which Ca = 1.
Ca = @(t) deval(sol,t);
% now we create a function for fsolve to work on.
% remember that algebraic equation should be set to 0
% f(Ca) = Ca1 - Ca = 0
f = @(t) Ca1 - Ca(t) ; % function at which Ca(=1) - ca(t) = 0.
tguess = 3; % initial guess of time at which Ca=1.
t_solution = fsolve(f,tguess)
Ca(t_solution) % this should equal to the set Ca1=1.
sprintf('C_A = 1 mol/L at a time of %1.2f seconds.',t_solution)
```

Finding Ca at any time:

fsolve completed because the vector of function values is near zero as measured by the default value of the function tolerance, and the problem appears regular as measured by the gradient.

t_solution = 2.4598
ans = 1.0000
ans = Ca = 1 mol/L at a time of 2.46 seconds.

Stopping the integration of an ODE at some condition

- Writing an Event Function

[value,isterminal,direction] = myEventsFcn(t,y)

The event function must also accept a third input argument for yt.

The output arguments value, isterminal, and direction are vectors whose ith element corresponds to the ith event:

Stopping the integration of an ODE at some condition

- value(i) is a mathematical expression describing the ith event. An event occurs when value(i) is equal to zero.

- isterminal(i) = 1 if the integration is to terminate when the ith event occurs. Otherwise, it is 0.

- direction(i) = 0 if all zeros are to be located (the default). A value of +1 locates only zeros where the event function is increasing, and -1 locates only zeros where the event function is decreasing. Specify direction = [] to use the default value of 0 for all events

Stopping the integration of an ODE at some condition

```
clear all; close all; clc
global k Ca1 % variables that we wish to share with the main script
k = 0.23;
t0 = 0 ; %initial time, sec.
Cao = 2.3; % Concentration of component A at t0., mol/l
tf = 10 ; % final time, sec.
tspan = [t0 tf]; % span of independent variable.
Ca1 = 1 ; % We seek the time at which Ca = 1.
```

Stopping the integration of an ODE at some condition

```
% create an event using odeset through options command.
% name this event as you wish.
options = odeset('Events',@event_function);
% note the extra outputs
[t,Ca,TE,VE] = ode45(@Batch,tspan,Cao,options) ;
TE % this is the time value where the event occurred
VE % this is the value of Ca where the event occurred
sprintf('At t = %1.2f seconds the concentration of A is %1.2f mol/L.', TE,VE)
```

Event Function

```
function [value,isterminal,direction] = event_function(t,Ca)
% when value is equal to zero, an event is triggered.
% set isterminal to 1 to stop the solver at the first event, or 0 to get all the events.
% direction=0 if all zeros are to be computed (the default), +1 if only zeros where the event function is increasing, and -1 if only zeros where the event function is ecreasing.
global Ca1 % variables that we wish to share with the main script
value = Ca - Ca1; % when value = 0, an event is triggered
isterminal = 1; % terminate after the first event
direction = 0; % get all the zeros
end
```

TE =

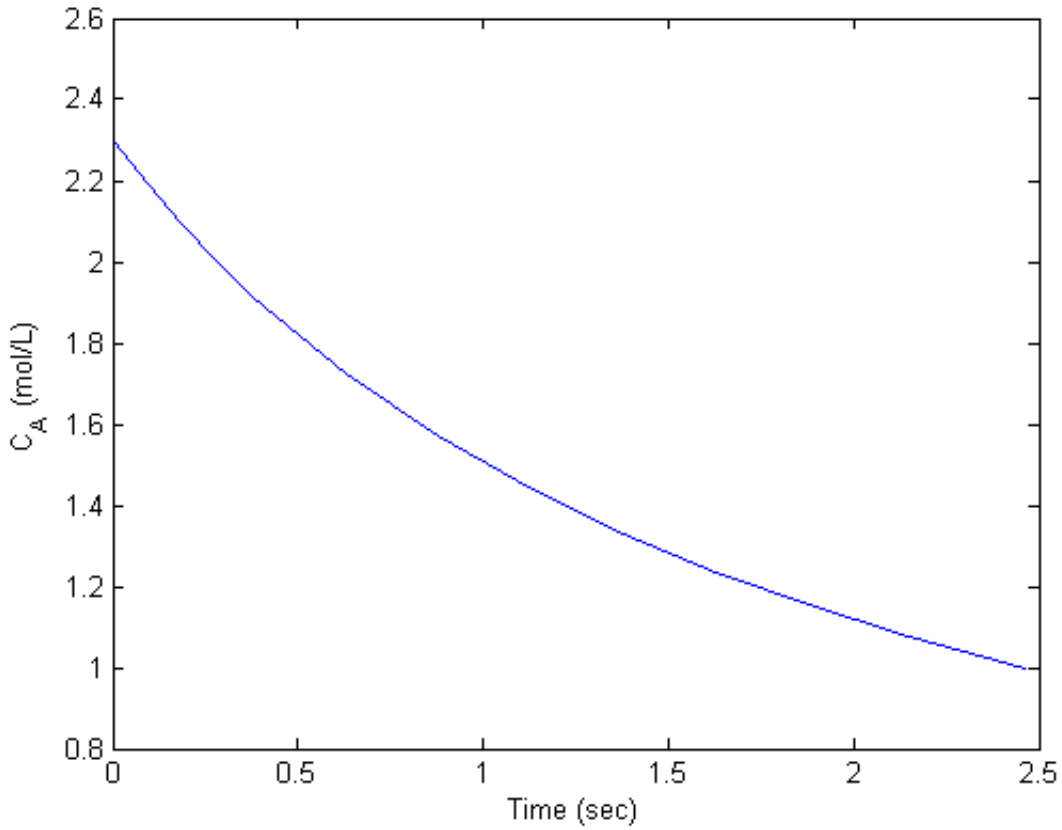
2.4598

VE =

1.0000

ans =

At t = 2.46 seconds the concentration of A is 1.00 mol/L.


2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
C_A (mol/L)
0
0.5
1
1.5
2
2.5
Time (sec)

The following set of differential equations describes the change in concentration three species in a tank. The reactions A→B→C occur within the tank. The constants k1, and k2 describe the reaction rate for A→B and B→C respectively. The following ode's are obtained:

$$\frac{dCa}{dt} = -k1Ca$$

$$\frac{dCb}{dt} = k1Ca - k2Cb$$

$$\frac{dCc}{dt} = k2Cb$$

Where k1=1 hr^{-1} and k2=2 hr^{-1} and at time t=0, Ca=5mol and Cb=Cc=0mol. Solve the system of equations and plot the change in concentration of each species over time. Select an appropriate time interval for the integration.

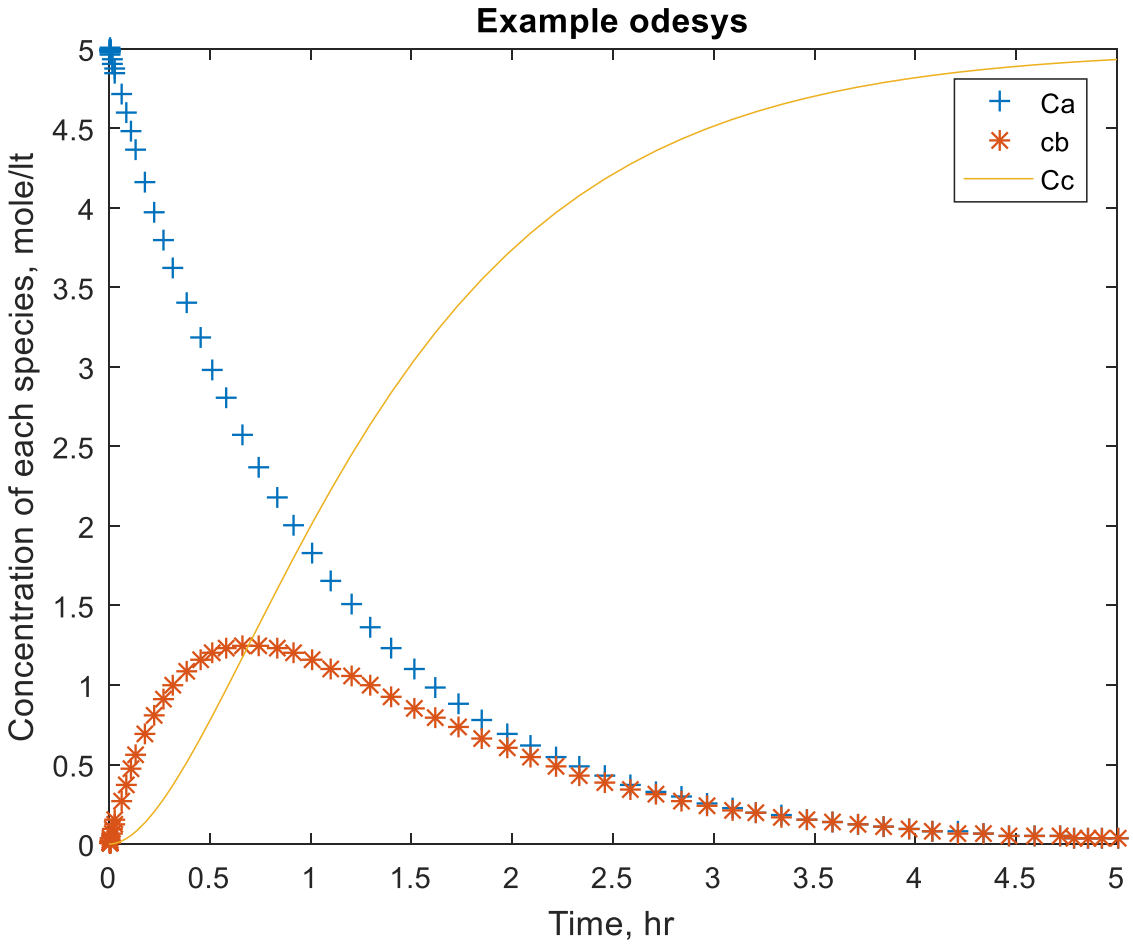
Solution

```
function dcdt=odesys(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)];
end
```

Solution

```
clc; clf; clear
global k1 k2
k1=1;
k2=2;
t0=0;
te=5;
tspan=[t0 te];
c0=[5 0 0];
[t c]=ode45('odesys',tspan,c0)
plot(t,c(:,1),'+',t,c(:,2),'*',t,c(:,3))
legend('Ca','cb','Cc')
xlabel('Time, hr')
ylabel('Concentration of each species, mole/lt')
title('Example odesys')
```

Solution


Example odesys
Concentration of each species, mole/lt
Time, hr
Ca
cb
Cc
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Problem Type: Nonstiff

Solver	Accuracy	When to Use
ode45	Medium	Most of the time. ode45 should be the first solver you try.
ode23	Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113	Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to valuate.

Problem Type: Stiff

Solver	Accuracy	When to Use
ode15s	Low to Medium	Try ode15s when ode45 fails or is inefficient and you suspect that the problem is stiff. Also use ode15s when solving differential algebraic equations (DAEs).
ode23s	Low	ode23s can be more efficient than ode15s at problems with crude error tolerances. It can solve some stiff problems for which ode15s is not effective. ode23s computes the Jacobian in each step, so it is beneficial to provide the Jacobian via odeset to maximize efficiency and accuracy.If there is a mass matrix, it must be constant.
ode23t	Low	Use ode23t if the problem is only moderately stiff and you need a solution without numerical damping. ode23t can solve differential algebraic equations (DAEs).
ode23tb	Low	Like ode23s, the ode23tb solver might be more efficient than ode15s at problems with crude error tolerances.

Ode Solver Remarks

- Some ODE problems exhibit stiffness, or difficulty in evaluation.
- Stiffness occurs when there is a difference in scaling somewhere in the problem. You can identify a problem as stiff if nonstiff solvers (such as ode45) are unable to solve the problem or are extremely slow. If you observe that a nonstiff solver is very slow, try using a stiff solver such as ode15s instead.
- When using a stiff solver, you can improve reliability and efficiency by supplying the Jacobian matrix or its sparsity pattern.

FYI: Stability of system of ODEs

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

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

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

$$\vdots$$

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

$$J = \left[\begin{array}{ccc} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_n} \\ & & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_n}{\partial x_n} \end{array}\right]_{y_{ss}}$$

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

matlab function for calculating eigenvalues of *J is eig(J)*

FYI: Stability of system of ODEs

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

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

a. Show that y1=0.07709 and y2=1.89351 is a steady-state solution of the system
b. Study the stability of the dynamic system around its steady state.
c. 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 ;
end
```

```
y0 = [0.1; 1.0]
ys = fsolve(@ode_stability,y0)
```

```
ys=
   0.0771
   1.8935
```

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 8y_1y_2^2 - 2y_2^2 & 8y^2{}_1y_2 - 4y_1y_2 + 5 \\ -9y_1^2y_2 - 12y_2^2 & -3y_1^2 - 24y_1y_2 - 3 \end{bmatrix}$$

(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.

(b) Eigenvalues of *J* at steady state:

```
y0 = [0.1; 1.0]
ys = fsolve(@ode_stability,y0)

% write the jacobian matrix
Jac = [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       -3*ys(1)^3-24*ys(1)*ys(2)-3] ;
ev = eig(Jac)
```

```
J =

   -4.9596    4.5061
  -43.1261   -6.5153
ev =

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

System is stable since all eigenvalues have negative real parts.

(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.

Stiff ODEs

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

$$SR \text{ (Stifeness Ratio)} = \frac{\max\left(\text{Real}(\lambda_j)\right)}{\min\left(\text{Real}(\lambda_j)\right)}$$