

MATLAB ODE Solving

Initial and Boundary Value Problems

Computational Design Laboratory
Department of Automotive Engineering
Hanyang University, Seoul, Korea



CONTENTS

- **Initial Value Problems**
 - ✓ **Basic methods**
 - ✓ **MATLAB bulit-in functions**
- **Boundary Value Problems**
- **Case study**
- **Assignment**

- **Basic methods**

- ✓ **Euler's method**
- ✓ **Heun's method**
- ✓ **Midpoint method**
- ✓ **Runge-Kutta method**

EXAMPLE 25.5



ODE 함수 예제



ordinary differential equation

$$y' = 4e^{0.8t} - 0.5y$$

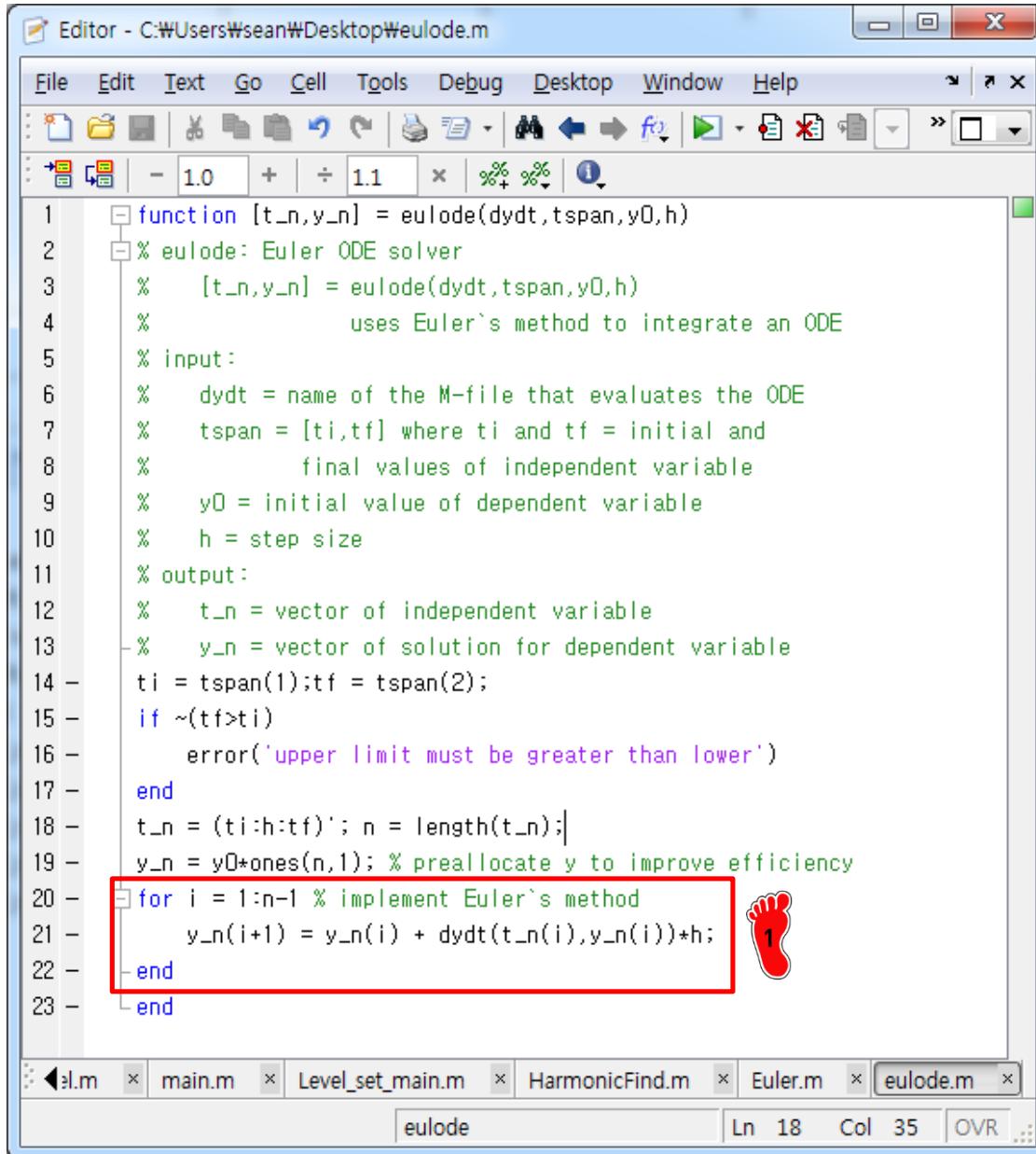
initial condition

$$t = 0, y = 2$$

analytic solution

$$y = \frac{4}{1.3} \left(e^{0.8t} - e^{-0.5t} \right) + 2e^{-0.5t}$$

EULER`S METHOD: FUNCTION



The screenshot shows the MATLAB Editor window with the file `eulode.m` open. The code implements Euler's method to solve an ordinary differential equation (ODE). The code is annotated with red boxes and numbers:

- Line 20:** A red box surrounds the loop `for i = 1:n-1 % implement Euler's method`. A red footprint icon with the number **1** is placed to its right.
- Line 21:** Inside the loop, the assignment `y_n(i+1) = y_n(i) + dydt(t_n(i),y_n(i))*h;` is highlighted with a red box. A red footprint icon with the number **1** is placed to its right.
- Line 22:** The closing brace of the loop is highlighted with a red box. A red footprint icon with the number **1** is placed to its right.
- Line 23:** The final closing brace of the function is highlighted with a red box. A red footprint icon with the number **1** is placed to its right.

```

Editor - C:\Users\sean\Desktop\euclidean.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Explorer Home Recent Files Run Task View Dock Window Help
function [t_n,y_n] = eulode(dydt,tspan,y0,h)
% eulode: Euler ODE solver
% [t_n,y_n] = eulode(dydt,tspan,y0,h)
% uses Euler's method to integrate an ODE
% input:
% dydt = name of the M-file that evaluates the ODE
% tspan = [ti,tf] where ti and tf = initial and
%         final values of independent variable
% y0 = initial value of dependent variable
% h = step size
% output:
% t_n = vector of independent variable
% y_n = vector of solution for dependent variable
ti = tspan(1);tf = tspan(2);
if ~tf>ti
    error('upper limit must be greater than lower')
end
t_n = (ti:h:tf)'; n = length(t_n);
y_n = y0*ones(n,1); % preallocate y to improve efficiency
for i = 1:n-1 % implement Euler's method
    y_n(i+1) = y_n(i) + dydt(t_n(i),y_n(i))*h;
end
end

```



오일러 방법을 적용한 메인
함수 코드

Euler's method

$$\frac{dy}{dt} = f(t, y)$$

$$y_{i+1} = y_i + \frac{dy}{dt} h$$

EULER'S METHOD: MAIN

The screenshot shows the MATLAB interface with the following code in the editor:

```

1 - clc; clear all; close all;
2
3 %% analytic solution
4 - t_a = [0:1:4]';
5 - y_a = 4/1.3*(exp(0.8*t_a)-exp(-0.5*t_a))+2*exp(-0.5*t_a);
6
7 %% numerical solution
8 - dydt = @(t,y) 4*exp(0.8*t)-0.5*y;
9 - [t_n,y_n] = euler(dydt,[0,4],2,1);
10
11 %% plot solution
12 - plot(t_a,y_a,'r')
13 - hold on
14 - grid on
15 - plot(t_n,y_n,'b')
16 - legend('Analytic','Numerical')
17 - xlabel('t'); ylabel('y')
18
19 %% print solution
20 - disp(['      t      y_a      y_n      error'])
21 - disp([t_a y_a y_n (abs(y_a-y_n)./y_a*100)])

```



1 Analytical solution을 구하는 코드



2 전 페이지에서 정의한 함수를 이용하여 수치적인 해를 구하는 코드



3 이론해와 수치해 비교 도식화

ordinary differential equation

$$y' = 4e^{0.8t} - 0.5y \quad 2$$

initial condition

$$t = 0, y = 2$$

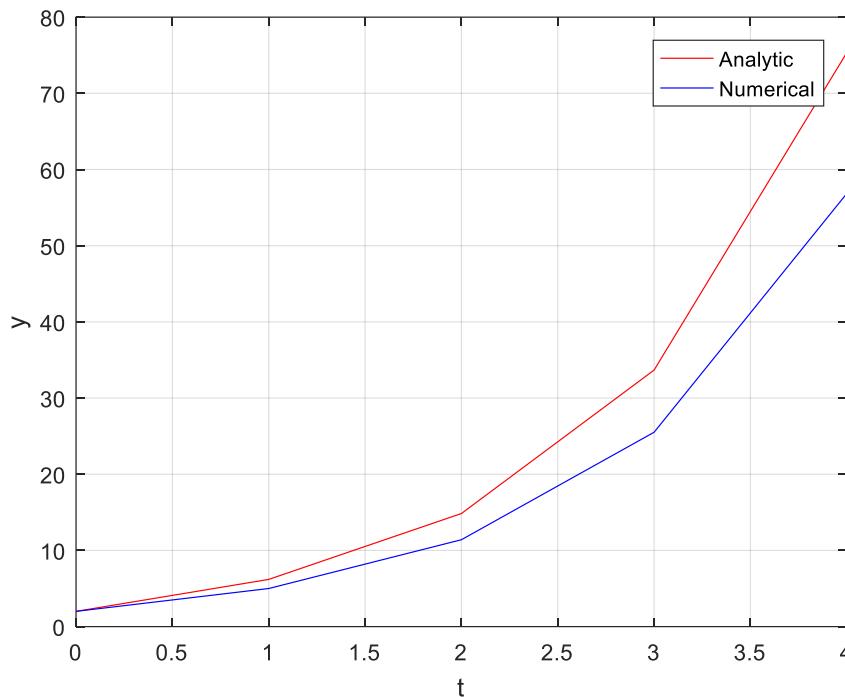
analytic solution

$$y = \frac{4}{1.3} \left(e^{0.8t} - e^{-0.5t} \right) + 2e^{-0.5t} \quad 1$$



$$err_{rel} = \frac{|y_a - y_n|}{y_a} \times 100 [\%]$$

EULER`S METHOD: RESULTS



Command Window

t	y_a	y_n	error
0	2.0000	2.0000	0
1.0000	6.1946	5.0000	19.2849
2.0000	14.8439	11.4022	23.1863
3.0000	33.6772	25.5132	24.2418
4.0000	75.3390	56.8493	24.5420

fx >>



그래프



독립 변수 step 별로 종속
변수 및 에러 출력



HEUN`S METHOD: FUNCTION

```

Editor - C:\Users\sean\Desktop\heun_without.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Edit Text Go Cell Tools Debug Desktop Window Help
1 function [t_n,y_n] = heun_without(dydt,tspan,y0,h)
2 ti = tspan(1);tf = tspan(2);
3 if ~tf>ti
4     error('upper limit must be greater than lower')
5 end
6 t_n = (ti:h:tf)'; n = length(t_n);
7 dydt_temp = 0; % temporary space of y'
8 dydt_heun = 0; % temporary space of y' (Heun)
9 y_n = y0*ones(n,1); % preallocate y to improve efficiency
10 for i = 1:n-1 % implement Heun's method
11     y_n(i+1) = y_n(i) + dydt(t_n(i),y_n(i))*h;
12     dydt_temp = dydt(t_n(i+1),y_n(i+1));
13     dydt_heun = (dydt(t_n(i),y_n(i))+dydt_temp)/2;
14     y_n(i+1) = y_n(i) + dydt_heun*h;
15 end
16 end

```

heun_without.m

heun_without

Ln 8 Col 56 OVR

1 Heun's method 의 corrector update 를 하지 않은 메인 함수 코드

Heun's method

Predictor

$$y_{i+1}^0 = y_i^m + f(t_i, y_i)h$$

Corrector

$$y_{i+1}^j = y_i^m + \frac{f(t_i, y_i^m) + f(t_{i+1}, y_{i+1}^{j-1})}{2}h$$

HEUN`S METHOD: FUNCTION

Editor - C:\Users\sean\Desktop\heun_with.m

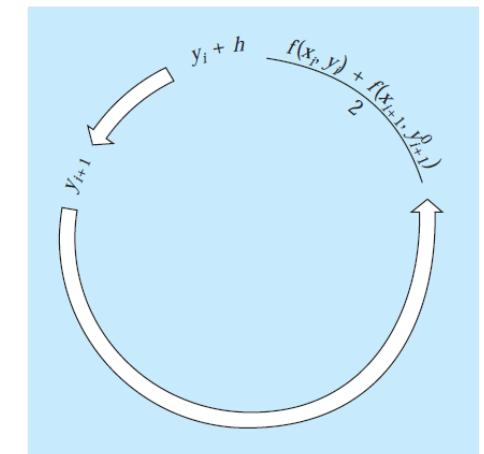
```

File Edit Text Go Cell Tools Debug Desktop Window Help
File Edit Text Go Cell Tools Debug Desktop Window Help
1 function [t_n,y_n] = heun_with(dydt,tspan,y0,h)
2 ti = tspan(1);tf = tspan(2);
3 if ~tf>ti
4     error('upper limit must be greater than lower')
5 end
6 t_n = (ti:h:tf)'; n = length(t_n);
7 dydt_temp = 0; % temporary space of y'
8 dydt_heun = 0; % temporary space of y' (Heun)
9 y_n_temp = 0; % temporary space of y
10 y_n = y0*ones(n,1); % preallocate y to improve efficiency
11 for i = 1:n-1 % implement Heun's method
12     y_n_temp = y_n(i) + dydt(t_n(i),y_n(i))*h;
13     while (1)
14         dydt_temp = dydt(t_n(i+1),y_n_temp);
15         dydt_heun = (dydt(t_n(i),y_n(i))+dydt_temp)/2;
16         y_n(i+1) = y_n(i) + dydt_heun*h;
17         if abs(y_n(i+1)-y_n_temp)/abs(y_n(i+1))*100 < 0.00001
18             break
19         else
20             y_n_temp = y_n(i+1);
21         end
22     end
23 end
24 end

```

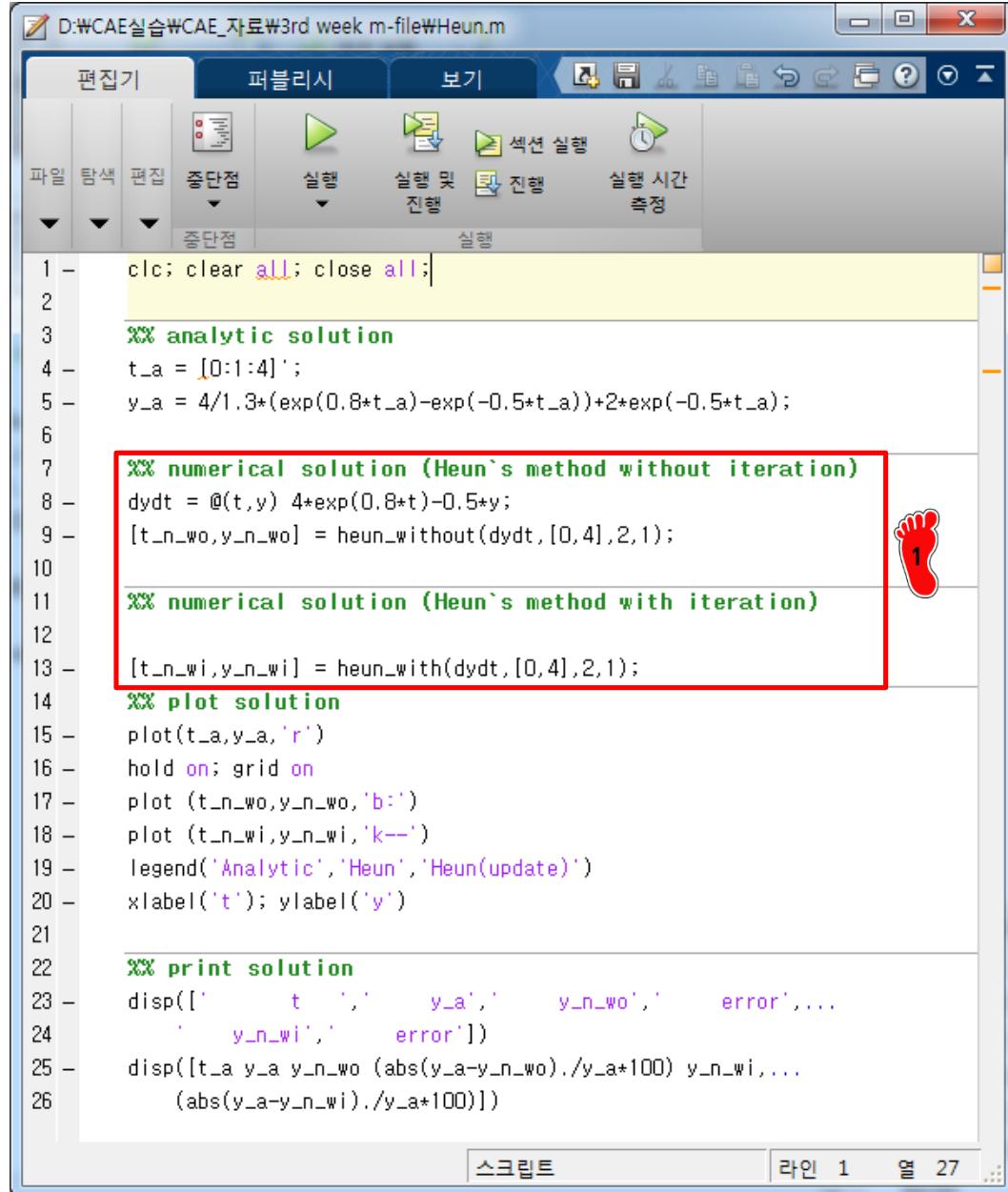
heun_with.m

1 Heun's method 의 corrector update 를 하는 메인 코드



$$|\varepsilon_a| = \left| \frac{y_{i+1}^j - y_{i+1}^{j-1}}{y_{i+1}^j} \right| 100\%$$

HEUN`S METHOD: MAIN



```

D:\CAE실습\CAE_자료\3rd week m-file\Heun.m
1 - clc; clear all; close all;
2
3 %% analytic solution
4 - t_a = [0:1:4]';
5 - y_a = 4/1.3*(exp(0.8*t_a)-exp(-0.5*t_a))+2*exp(-0.5*t_a);
6
7 %% numerical solution (Heun's method without iteration)
8 - dydt = @(t,y) 4*exp(0.8*t)-0.5*y;
9 - [t_n_wo,y_n_wo] = heun_without(dydt,[0,4],2,1);
10
11 %% numerical solution (Heun's method with iteration)
12
13 - [t_n_wi,y_n_wi] = heun_with(dydt,[0,4],2,1);
14 %% plot solution
15 - plot(t_a,y_a,'r')
16 - hold on; grid on
17 - plot (t_n_wo,y_n_wo,'b:')
18 - plot (t_n_wi,y_n_wi,'k--')
19 - legend('Analytic','Heun','Heun(update)')
20 - xlabel('t'); ylabel('y')
21
22 %% print solution
23 - disp(['      t ','      y_a','      y_n_wo','      error',...
24 - '      y_n_wi','      error'])
25 - disp([t_a y_a y_n_wo (abs(y_a-y_n_wo)./y_a*100) y_n_wi,...
26 - (abs(y_a-y_n_wi)./y_a*100)])

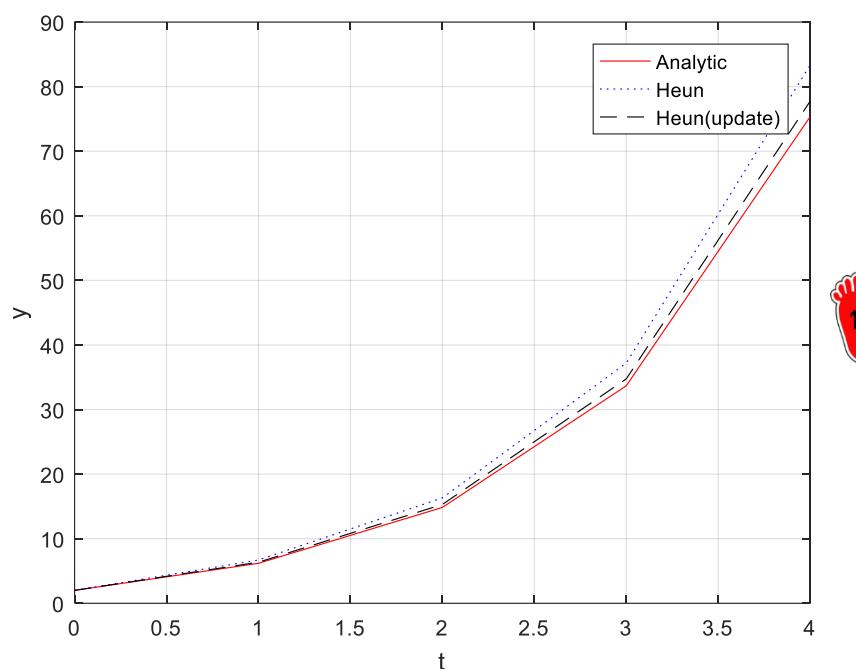
```



Heun's method 의
corrector update 를 구분하
여 함수 호출



HEUN`S METHOD: RESULTS



Command Window

t	y_a	y_n_wo	error	y_n_wi	error
0	2.0000	2.0000	0	2.0000	0
1.0000	6.1946	6.7011	8.1756	6.3609	2.6835
2.0000	14.8439	16.3198	9.9425	15.3022	3.0876
3.0000	33.6772	37.1992	10.4584	34.7433	3.1657
4.0000	75.3390	83.3378	10.6171	77.7351	3.1805

fx >> |



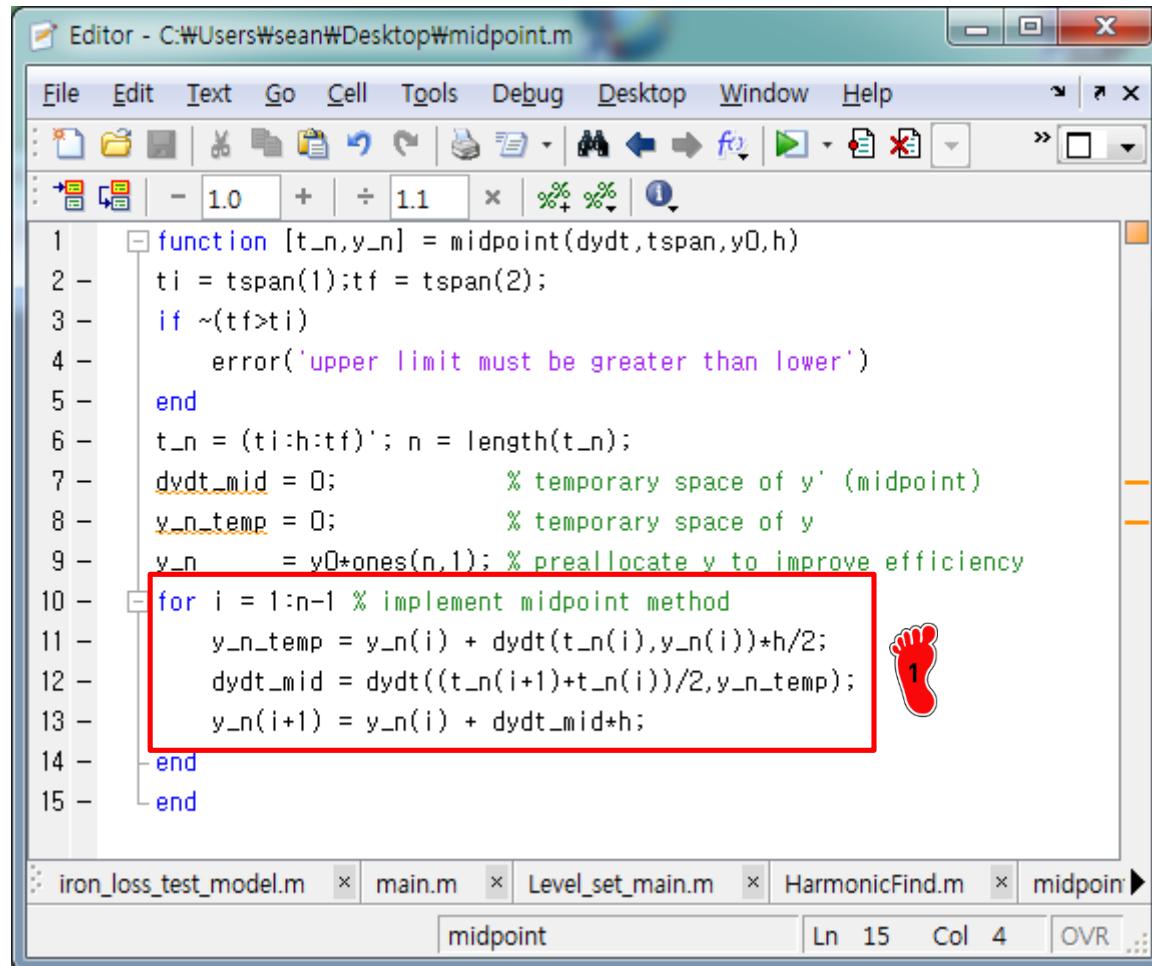
그래프



독립 변수 step 별로 종속
변수 및 에러 출력



MIDPOINT METHOD: FUNCTION



```

Editor - C:\Users\sean\Desktop\midpoint.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Edit Text Go Cell Tools Debug Desktop Window Help
1 function [t_n,y_n] = midpoint(dydt,tspan,y0,h)
2 ti = tspan(1);tf = tspan(2);
3 if ~tf>ti
4     error('upper limit must be greater than lower')
5 end
6 t_n = (ti:h:tf)'; n = length(t_n);
7 dydt_mid = 0; % temporary space of y' (midpoint)
8 y_n_temp = 0; % temporary space of y
9 y_n = y0+ones(n,1); % preallocate y to improve efficiency
10 for i = 1:n-1 % implement midpoint method
11     y_n_temp = y_n(i) + dydt(t_n(i),y_n(i))*h/2;
12     dydt_mid = dydt((t_n(i+1)+t_n(i))/2,y_n_temp);
13     y_n(i+1) = y_n(i) + dydt_mid*h;
14 end
15 end
iron_loss_test_model.m x main.m x Level_set_main.m x HarmonicFind.m x midpoint >
midpoint Ln 15 Col 4 OVR ...

```



midpoint method 메인 함
수 코드

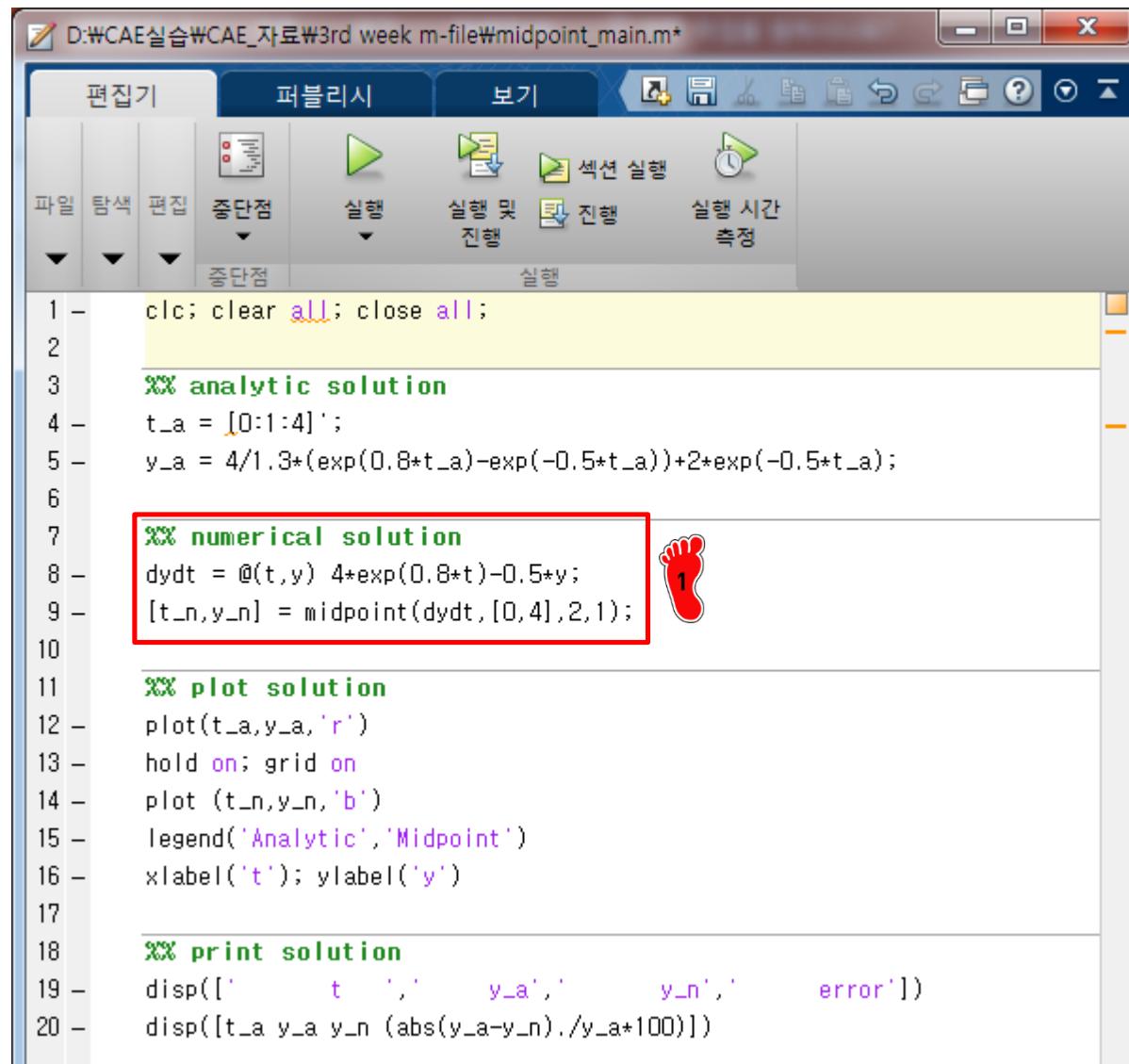
Midpoint's method
Predictor

$$y_{i+1/2} = y_i + f(t_i, y_i) \frac{h}{2}$$

Corrector

$$y_{i+1} = y_i + f(t_{i+1/2}, y_{i+1/2}) h$$

MIDPOINT METHOD: MAIN



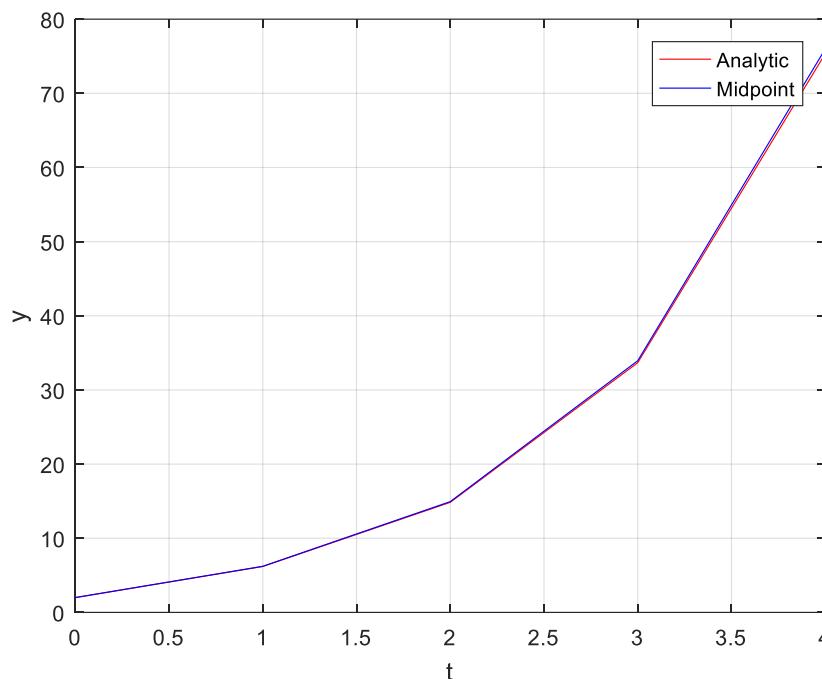
The screenshot shows the MATLAB IDE with the script `midpoint_main.m` open. The code implements the midpoint method to solve a differential equation. The numerical solution section is highlighted with a red box and a red foot icon with the number '1'.

```
1 - clc; clear all; close all;
2 -
3 %% analytic solution
4 - t_a = [0:1:4]';
5 - y_a = 4/1.3*(exp(0.8*t_a)-exp(-0.5*t_a))+2*exp(-0.5*t_a);
6 -
7 %% numerical solution
8 - dydt = @(t,y) 4*exp(0.8*t)-0.5*y;
9 - [t_n,y_n] = midpoint(dydt,[0,4],2,1);
10 -
11 %% plot solution
12 - plot(t_a,y_a,'r')
13 - hold on; grid on
14 - plot(t_n,y_n,'b')
15 - legend('Analytic','Midpoint')
16 - xlabel('t'); ylabel('y')
17 -
18 %% print solution
19 - disp(['      t      ', '      y_a', '      y_n', '      error'])
20 - disp([t_a y_a y_n (abs(y_a-y_n)./y_a*100)])
```



1 midpoint method 함수를
호출

MIDPOINT METHOD: RESULTS



그래프



독립 변수 step 별로 종속
변수 및 에러 출력

Command Window

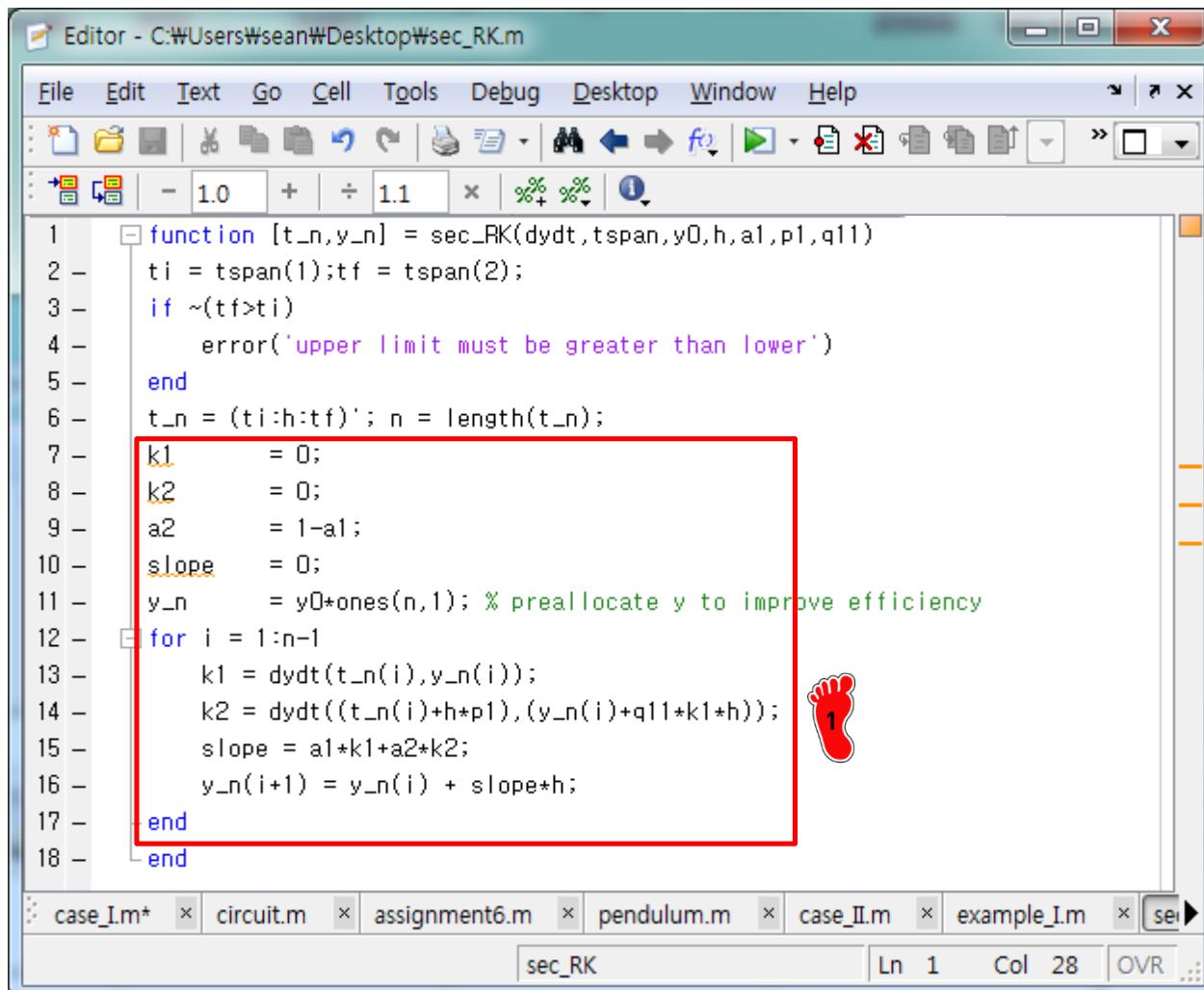
t	y_a	y_n	error
0	2.0000	2.0000	0
1.0000	6.1946	6.2173	0.3659
2.0000	14.8439	14.9407	0.6522
3.0000	33.6772	33.9412	0.7839
4.0000	75.3390	75.9686	0.8358

fx >>

OVR



2ND RK: FUNCTION



```

Editor - C:\Users\sean\Desktop\sec_RK.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Explorer Home Recent Files Run Stop Task View
- 1.0 + ÷ 1.1 × %% % | i
1 function [t_n,y_n] = sec_RK(dydt,tspan,y0,h,a1,p1,q11)
2 ti = tspan(1);tf = tspan(2);
3 if ~tf>ti
4     error('upper limit must be greater than lower')
5 end
6 t_n = (ti:h:tf)'; n = length(t_n);
7 k1 = 0;
8 k2 = 0;
9 a2 = 1-a1;
10 slope = 0;
11 y_n = y0*ones(n,1); % preallocate y to improve efficiency
12 for i = 1:n-1
13     k1 = dydt(t_n(i),y_n(i));
14     k2 = dydt((t_n(i)+h*p1),(y_n(i)+q11*k1*h));
15     slope = a1*k1+a2*k2;
16     y_n(i+1) = y_n(i) + slope*h;
17 end
18 end
case_I.m* circuit.m* assignment6.m* pendulum.m* case_II.m* example_Im* sec_RK
Ln 1 Col 28 OVR ...

```



Second-order Runge-Kutta Method 를 적용한 메인 함수 코드

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2) h$$

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

$$k_2 = f(t_i + p_1 h, y_i + q_{11} k_1 h)$$

Heun

$$a_1 = a_2 = \frac{1}{2}$$

$$p_1 = q_{11} = 1$$

Midpoint

$$a_1 = 0, a_2 = 1$$

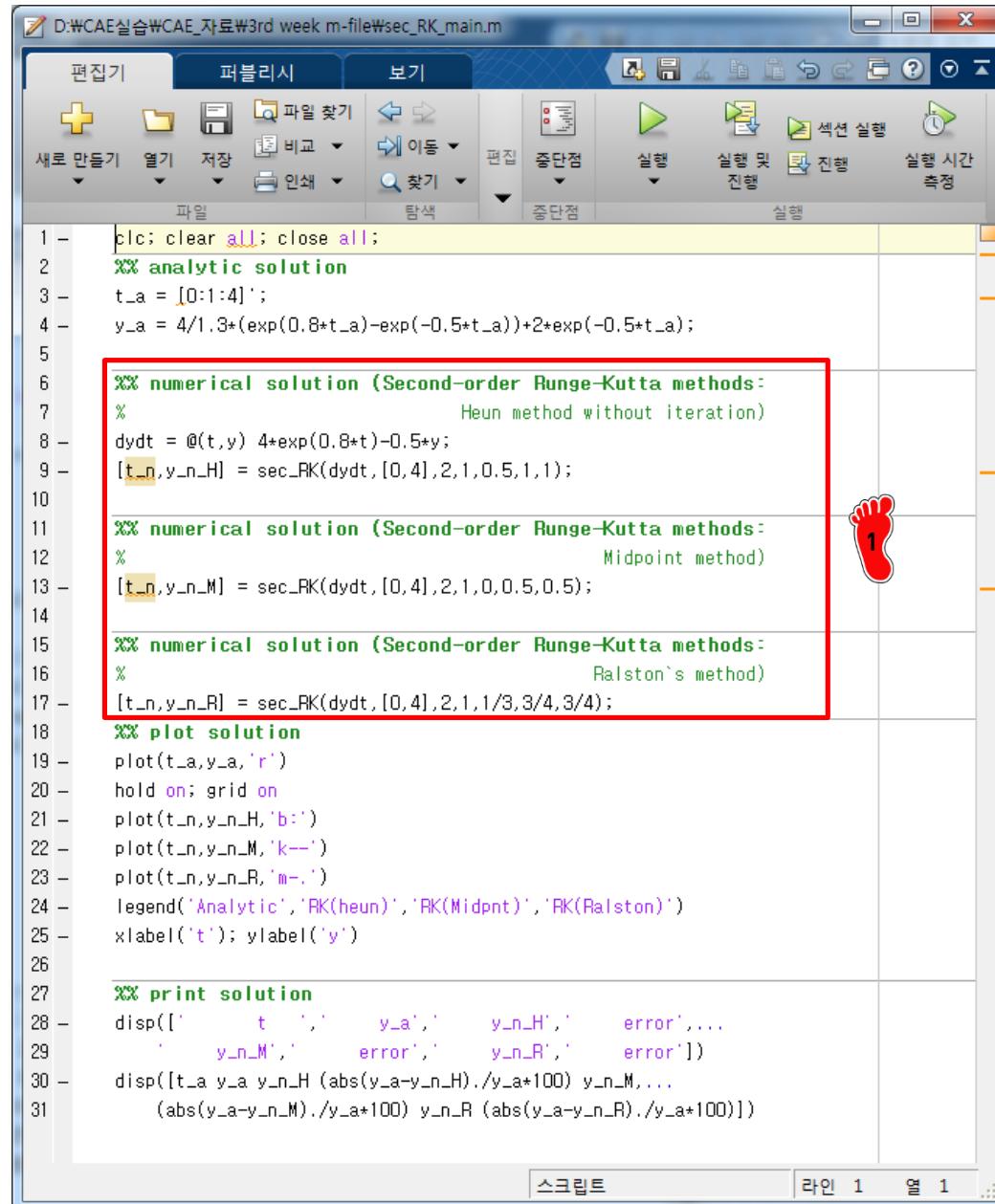
$$p_1 = q_{11} = \frac{1}{2}$$

Ralston

$$a_1 = \frac{1}{3}, a_2 = \frac{2}{3}$$

$$p_1 = q_{11} = \frac{3}{4}$$

2ND RK: MAIN



```

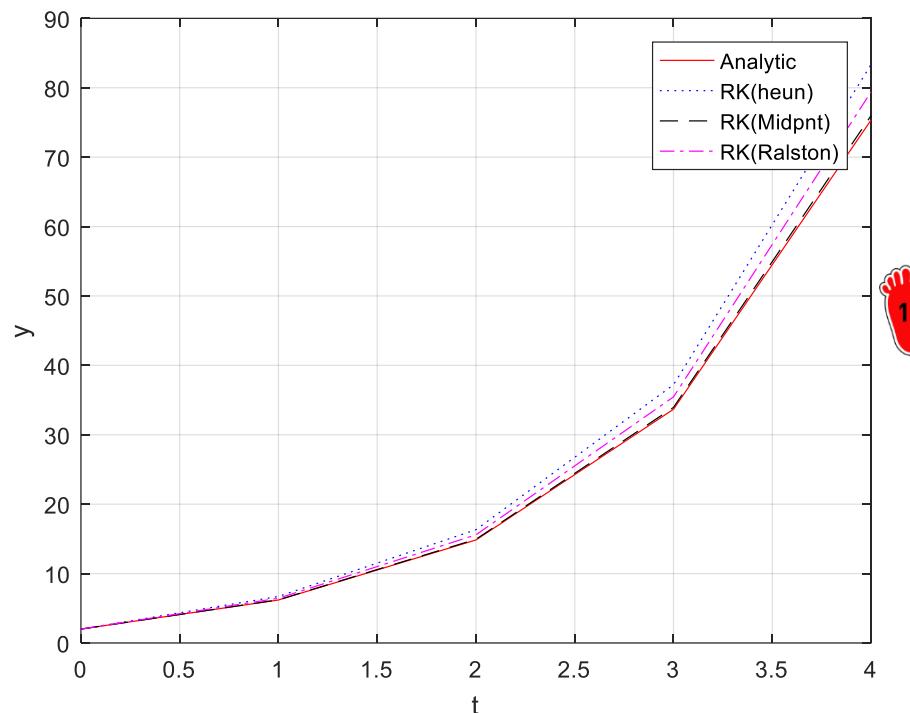
D:\WCAE실습\WCAE_자료\3rd week m-file\sec_RK_main.m
1 - clc; clear all; close all;
2 - % analytic solution
3 - t_a = [0:1:4]';
4 - y_a = 4/1.3*(exp(0.8*t_a)-exp(-0.5*t_a))+2*exp(-0.5*t_a);
5 -
6 - %% numerical solution (Second-order Runge-Kutta methods:
7 - % Heun method without iteration)
8 - dydt = @(t,y) 4*exp(0.8*t)-0.5*y;
9 - [t_n,y_n_H] = sec_RK(dydt,[0,4],2,1,0.5,1,1);
10 -
11 - %% numerical solution (Second-order Runge-Kutta methods:
12 - % Midpoint method)
13 - [t_n,y_n_M] = sec_RK(dydt,[0,4],2,1,0,0.5,0.5);
14 -
15 - %% numerical solution (Second-order Runge-Kutta methods:
16 - % Ralston's method)
17 - [t_n,y_n_R] = sec_RK(dydt,[0,4],2,1,1/3,3/4,3/4);
18 - %% plot solution
19 - plot(t_a,y_a,'r')
20 - hold on; grid on
21 - plot(t_n,y_n_H,'b:')
22 - plot(t_n,y_n_M,'k--')
23 - plot(t_n,y_n_R,'m-.')
24 - legend('Analytic','RK(Heun)','RK(Midpt)','RK(Ralston)')
25 - xlabel('t'); ylabel('y')
26 -
27 - %% print solution
28 - disp(['    t    ', '    y_a', '    y_n_H', '    error',...
29 -         '    y_n_M', '    error', '    y_n_R', '    error'])
30 - disp([t_a y_a y_n_H (abs(y_a-y_n_H)./y_a*100) y_n_M, ...
31 -         (abs(y_a-y_n_M)./y_a*100) y_n_R (abs(y_a-y_n_R)./y_a*100)])

```



3가지 방법을 비교하기 위한 메인 코드

2ND RK : RESULTS



1
그래프
독립 변수 step 별로 종속
변수 및 에러 출력



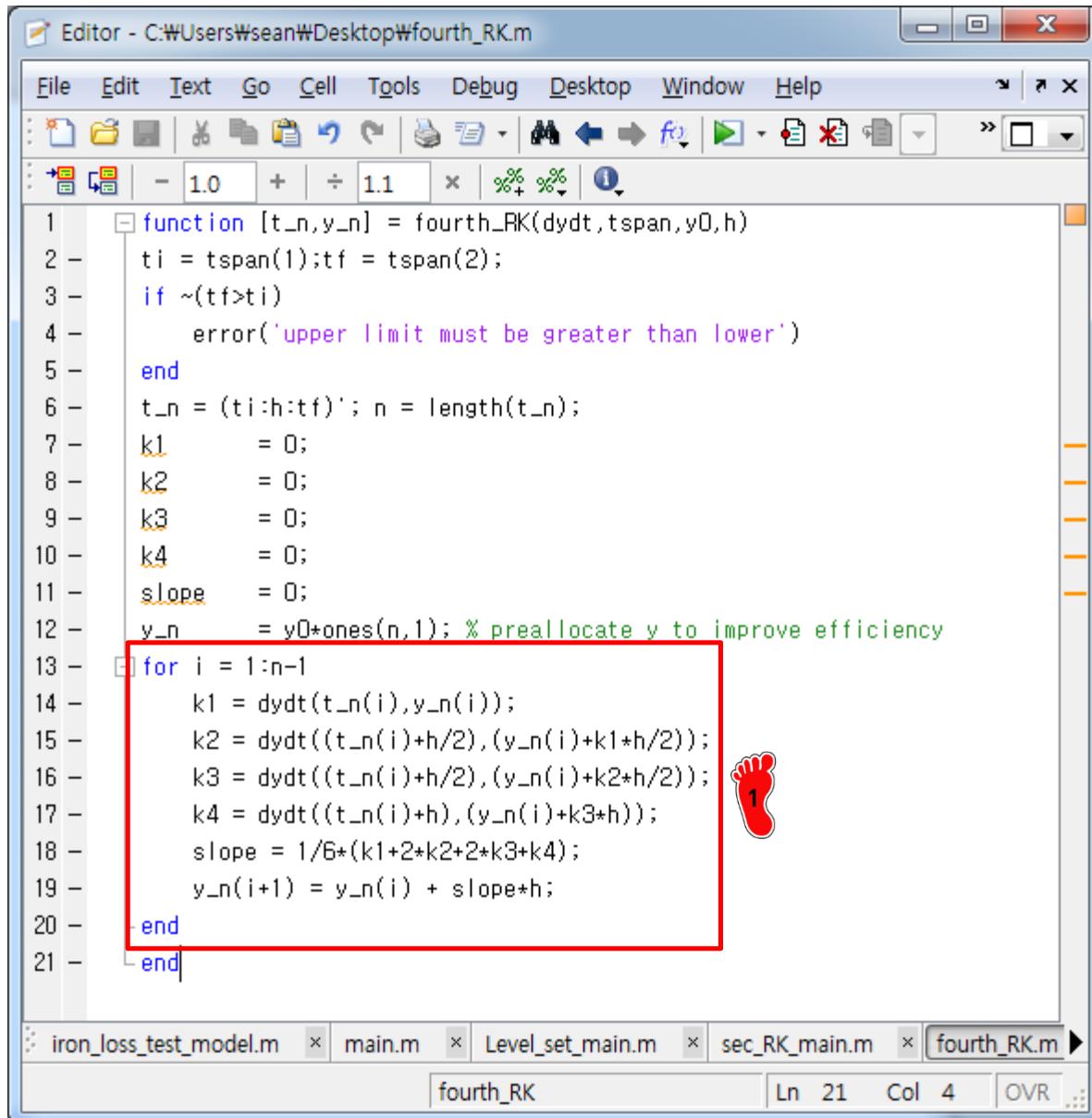
OVR

Command Window

t	y_a	y_n_H	error	y_n_M	error	y_n_R	error
0	2.0000	2.0000	0	2.0000	0	2.0000	0
1.0000	6.1946	6.7011	8.1756	6.2173	0.3659	6.4423	3.9984
2.0000	14.8439	16.3198	9.9425	14.9407	0.6522	15.5822	4.9733
3.0000	33.6772	37.1992	10.4584	33.9412	0.7839	35.4566	5.2837
4.0000	75.3390	83.3378	10.6171	75.9686	0.8358	79.3962	5.3853

f(x) >> |

4TH RK: FUNCTION



```

Editor - C:\Users\sean\Desktop\fourth_RK.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Edit Text Go Cell Tools Debug Desktop Window Help
function [t_n,y_n] = fourth_RK(dydt,tspan,y0,h)
ti = tspan(1);tf = tspan(2);
if ~tf>ti
    error('upper limit must be greater than lower')
end
t_n = (ti:h:tf)'; n = length(t_n);
k1 = 0;
k2 = 0;
k3 = 0;
k4 = 0;
slope = 0;
y_n = y0*ones(n,1); % preallocate y to improve efficiency
for i = 1:n-1
    k1 = dydt(t_n(i),y_n(i));
    k2 = dydt((t_n(i)+h/2),(y_n(i)+k1*h/2));
    k3 = dydt((t_n(i)+h/2),(y_n(i)+k2*h/2));
    k4 = dydt((t_n(i)+h),(y_n(i)+k3*h));
    slope = 1/6*(k1+2*k2+2*k3+k4);
    y_n(i+1) = y_n(i) + slope*h;
end
end

```

iron_loss_test_model.m main.m Level_set_main.m sec_RK_main.m fourth_RK.m

fourth_RK Ln 21 Col 4 OVR



Fourth-order Runge-Kutta Method 를 적용한 메인 함수 코드

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

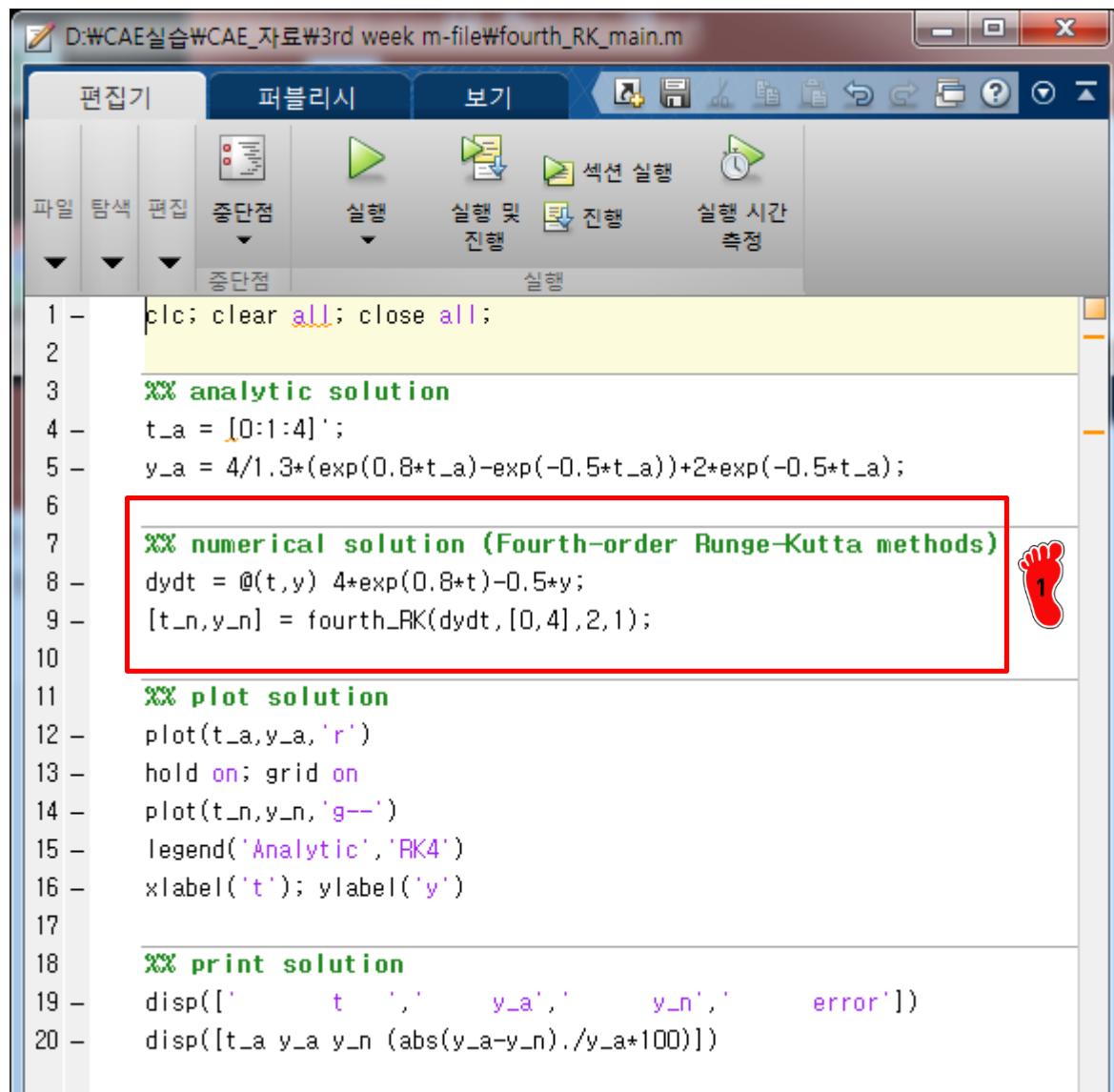
$$k_1 = f(t_i, y_i)$$

$$k_2 = f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(t_i + h, y_i + k_3 h)$$

4TH RK: MAIN



The screenshot shows the MATLAB interface with the file `D:\CAE실습\CAE_자료\3rd week m-file\fourth_RK_main.m` open. The code implements the fourth-order Runge-Kutta method to solve a differential equation. The numerical solution part is highlighted with a red box.

```

1 - clc; clear all; close all;
2
3 %>>> analytic solution
4 - t_a = [0:1:4];
5 - y_a = 4/1.3*(exp(0.8*t_a)-exp(-0.5*t_a))+2*exp(-0.5*t_a);
6
7 %>>> numerical solution (Fourth-order Runge-Kutta methods)
8 - dydt = @(t,y) 4*exp(0.8*t)-0.5*y;
9 - [t_n,y_n] = fourth_RK(dydt,[0,4],2,1);
10
11 %>>> plot solution
12 - plot(t_a,y_a,'r')
13 - hold on; grid on
14 - plot(t_n,y_n,'g--')
15 - legend('Analytic','RK4')
16 - xlabel('t'); ylabel('y')
17
18 %>>> print solution
19 - disp(['      t      y_a      y_n      error'])
20 - disp([t_a y_a y_n (abs(y_a-y_n)./y_a*100)])

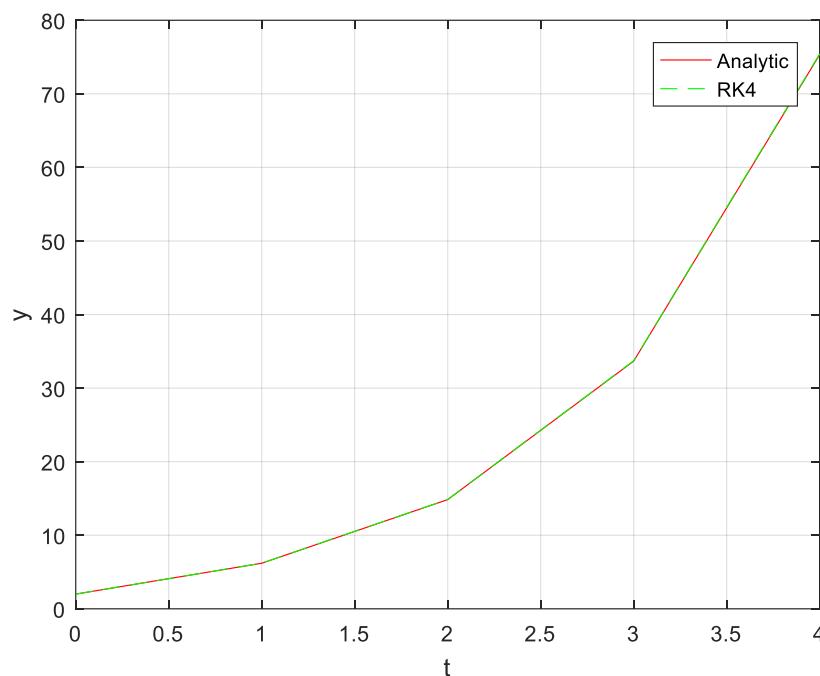
```



Fourth-order RK method
를 적용한 메인 코드



4TH RK : RESULTS



그래프



독립 변수 step 별로 종속
변수 및 에러 출력

t	y_a	y_n	error
0	2.0000	2.0000	0
1.0000	6.1946	6.2010	0.1034
2.0000	14.8439	14.8625	0.1250
3.0000	33.6772	33.7213	0.1312
4.0000	75.3390	75.4392	0.1330

fx >> |



- **Matlab bulit-in functions for nonstiff systems**
 - ✓ **ode23**
 - ✓ **ode45**
 - ✓ **ode113**
 - ✓ **Examples**

ODE_x BUILT IN FUNCTIONS

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

1. odefun : Functions to solve (function)

For example, to solve $y' = 5y - 3$, use the function:

```
function dydt = odefun(t,y)
dydt = 5*y-3;
```

Solve the ODE

$$y' = 2t.$$

2. tspan : Interval of integration (vector)

Example: [1 10]

Use a time interval of [0,5] and the initial condition $y0 = 0$.

```
tspan = [0 5];
y0 = 0;
[t,y] = ode45(@(t,y) 2*t, tspan, y0);
```

3. y0 : initial conditions (vector)

4. options : option structure (structure array)

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

ode23: The ode23 uses the BS23 algorithm (Bogacki and Shampine, 1989; Shampine, 1994), which simultaneously uses second- and third-order RK formulas to solve the ODE and make error estimates for step-size adjustment.

ode45: The ode45 function uses an algorithm developed by Dormand and Prince (1980), which simultaneously uses fourth- and fifth-order RK formulas to solve the ODE and make error estimates for step-size adjustment. MATLAB recommends that ode45 is the best function to apply as a “**first try**” for most problems.

ode113: The ode113 function uses a variable-order Adams-Basforth-Moulton solver. It is useful for stringent error tolerances or computationally intensive ODE functions. Note that this is a multistep method.

EXAMPLE 25.5: MAIN

```

1 - clc; clear all; close all;
2 -
3 %% analytic solution
4 - t_a = [0:0.01:4]';
5 - y_a = 4/1.3*(exp(0.8*t_a)-exp(-0.5*t_a))+2*exp(-0.5*t_a);
6 %% ode23 solution
7 - dydt = @(t,y) 4*exp(0.8*t)-0.5*y;
8 - [t_23,y_23] = ode23(dydt,[0 4],2);
9 %% ode45 solution
10 - [t_45,y_45] = ode45(dydt,[0 4],2);
11 %% ode113 solution
12 - [t_113,y_113] = ode113(dydt,[0 4],2);
13 %% plot solution
14 - plot(t_a,y_a,'r')
15 - hold on; grid on
16 - plot (t_23,y_23,'b')
17 - plot (t_45,y_45,'k')
18 - plot (t_113,y_113,'g')
19 - legend('Analytic','ode23','ode45','ode113')
20 - xlabel('t'); ylabel('y')

```

1 ode23, ode45, ode113 함수를 적용한 코드

2 예제

ordinary differential equation

$$y' = 4e^{0.8t} - 0.5y$$

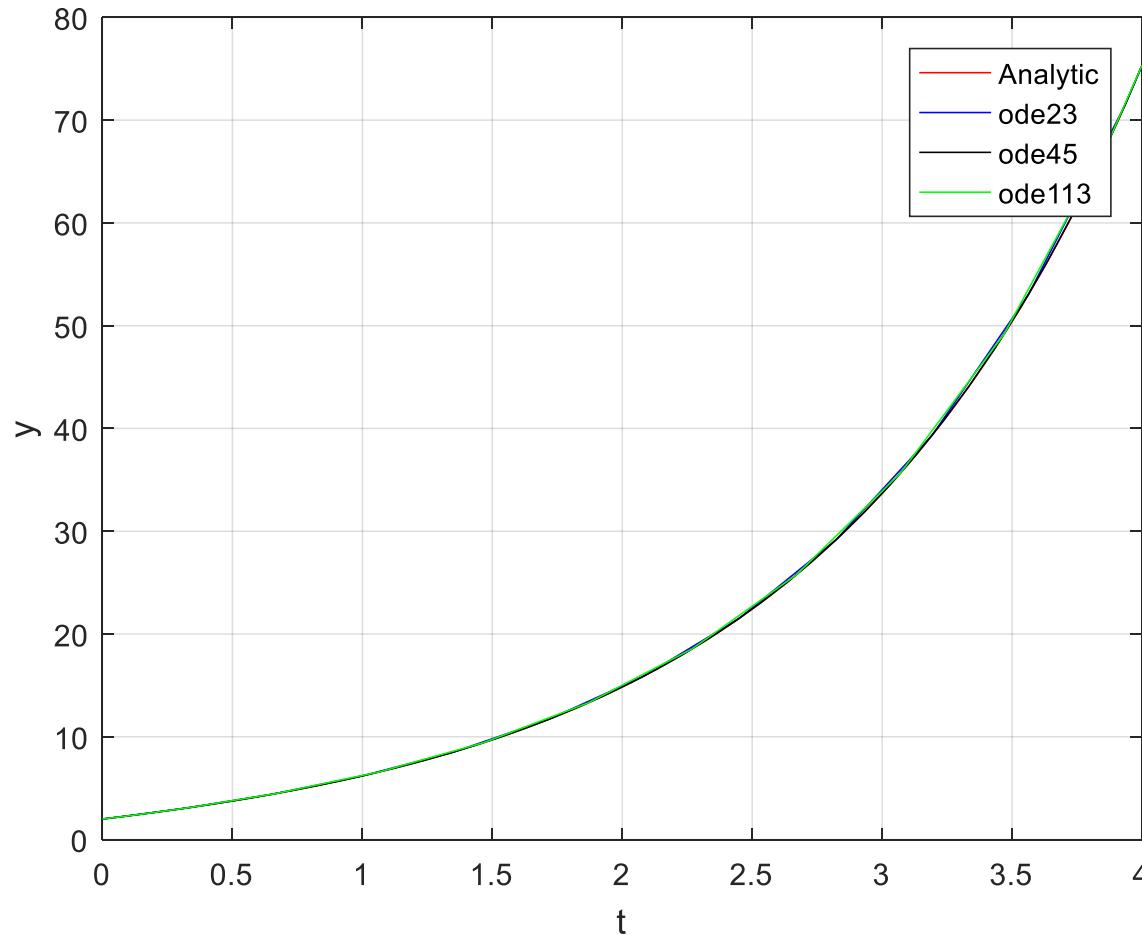
initial condition

$$t = 0, y = 2$$

analytic solution

$$y = \frac{4}{1.3} \left(e^{0.8t} - e^{-0.5t} \right) + 2e^{-0.5t}$$

EXAMPLE 25.5: RESULTS



1
ode23, ode45, ode113 함
수를 적용한 결과



EXAMPLE 28.2

nonlinear ordinary differential equations

$$\begin{cases} \frac{dy_1}{dt} = ay_1 - by_1y_2 \\ \frac{dy_2}{dt} = -cy_2 + dy_1y_2 \end{cases}$$

$$a = 1.2, b = 0.6, c = 0.8, d = 0.3$$

initial condition

$$t = 0, y_1 = 2, y_2 = 1$$



Predator-prey model developed by the Italian mathematician Vito Volterra and the American biologist Alfred J. Lotka.

먹이사슬에 관한 미분방정식

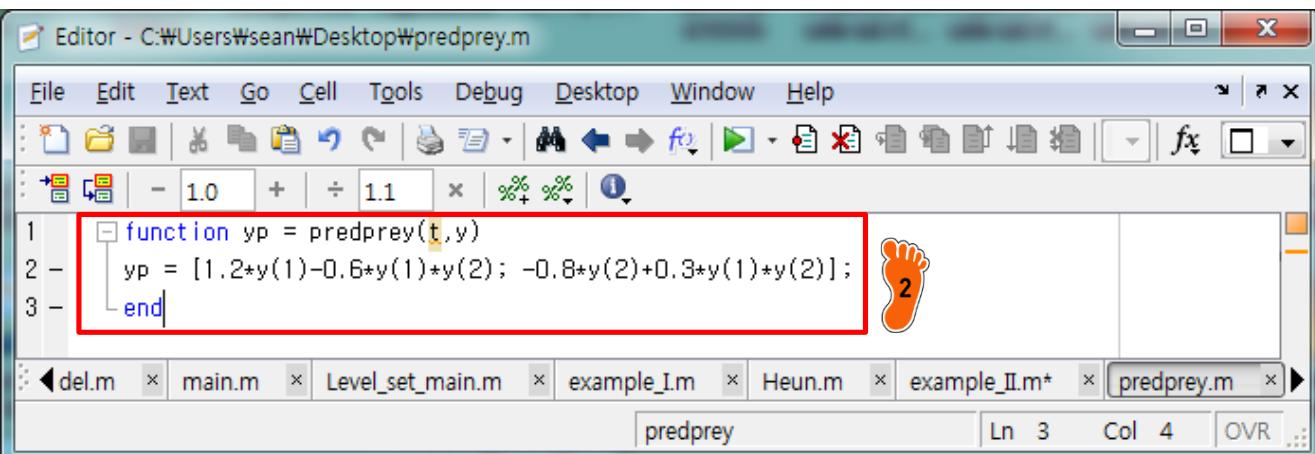
a = the prey growth rate

c = the predator death rate
 $b=d$ = the rate

characterizing the effect of the predator-prey interaction on prey death and predator growth



매틀랩 함수



```

Editor - C:\Users\sean\Desktop\predprey.m
File Edit Text Go Cell Tools Debug Desktop Window Help
function yp = predprey(t,y)
yp = [1.2*y(1)-0.6*y(1)*y(2); -0.8*y(2)+0.3*y(1)*y(2)];
end

```

predprey.m

EXAMPLE 28.2: MAIN & RESULT

```

1 - clc; clear all; close all;
2 - %% ode45 solution
3 - [t,y] = ode23(@predprey,[0 20],[2,1]);
4 - %% plot solution
5 - plot(t,y); grid on
6 - legend('y1','y2'); xlabel('t'); ylabel('y')

```

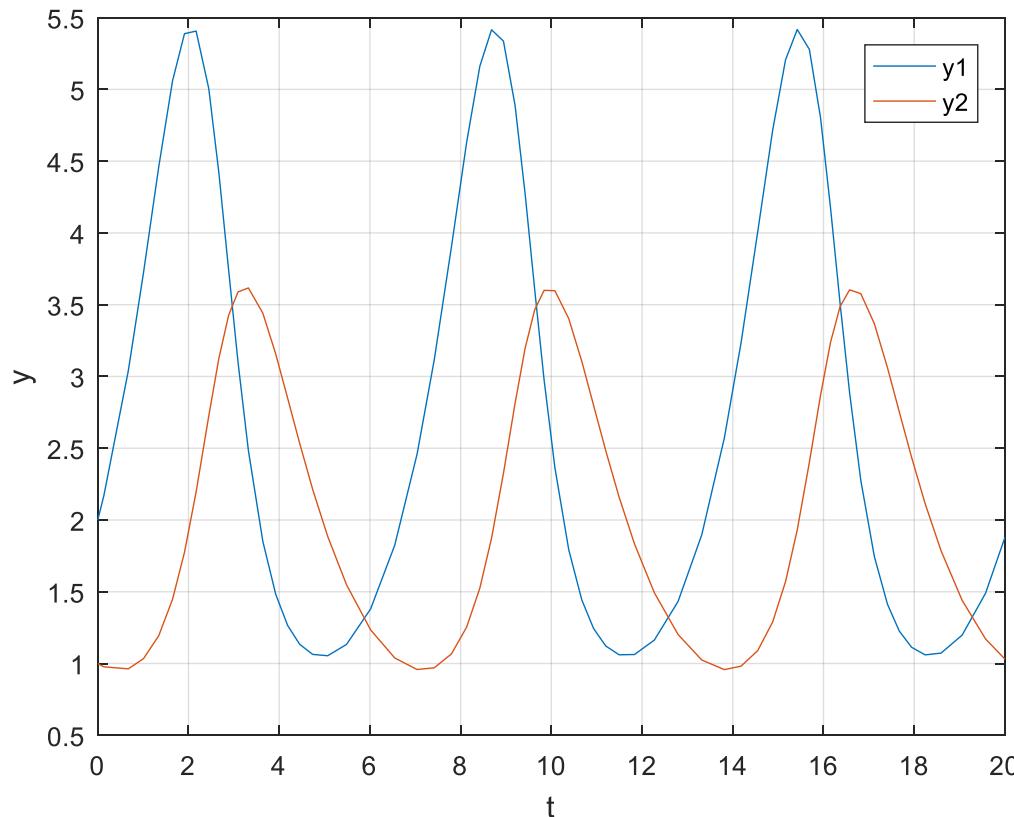


실행 코드

결과

y1 은 prey (먹이)
y2 는 predator (포식자)

먹이의 증가 및 감소가 포식
자의 분포에 영향을 끼치는
결과



EXAMPLE 25.14

ordinary differential equation

$$\frac{dy}{dt} = 10e^{-(t-2)^2/2[2(0.075)^2]} - 0.6y$$

initial condition

$$t = 0, y = 0.5$$

1



Adaptive ODE solver 를 보여주기 위한 미분방정식



매틀랩 함수

The screenshot shows the MATLAB Editor window with the file name 'dydt.m'. The code in the editor is:

```

Editor - C:\Users\sean\Desktop\dydt.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Explorer Home Recent Folders Run Stop Task View
1 function yp = dydt(t,y)
2     yp = 10*exp(-(t-2)*(t-2)/(2*.075^2))-0.6*y;
3 end

```

The second line of code, which defines the function body, is highlighted with a red rectangle. A red footprint icon is positioned next to the highlighted code. An orange footprint icon is positioned next to the right margin of the editor window.

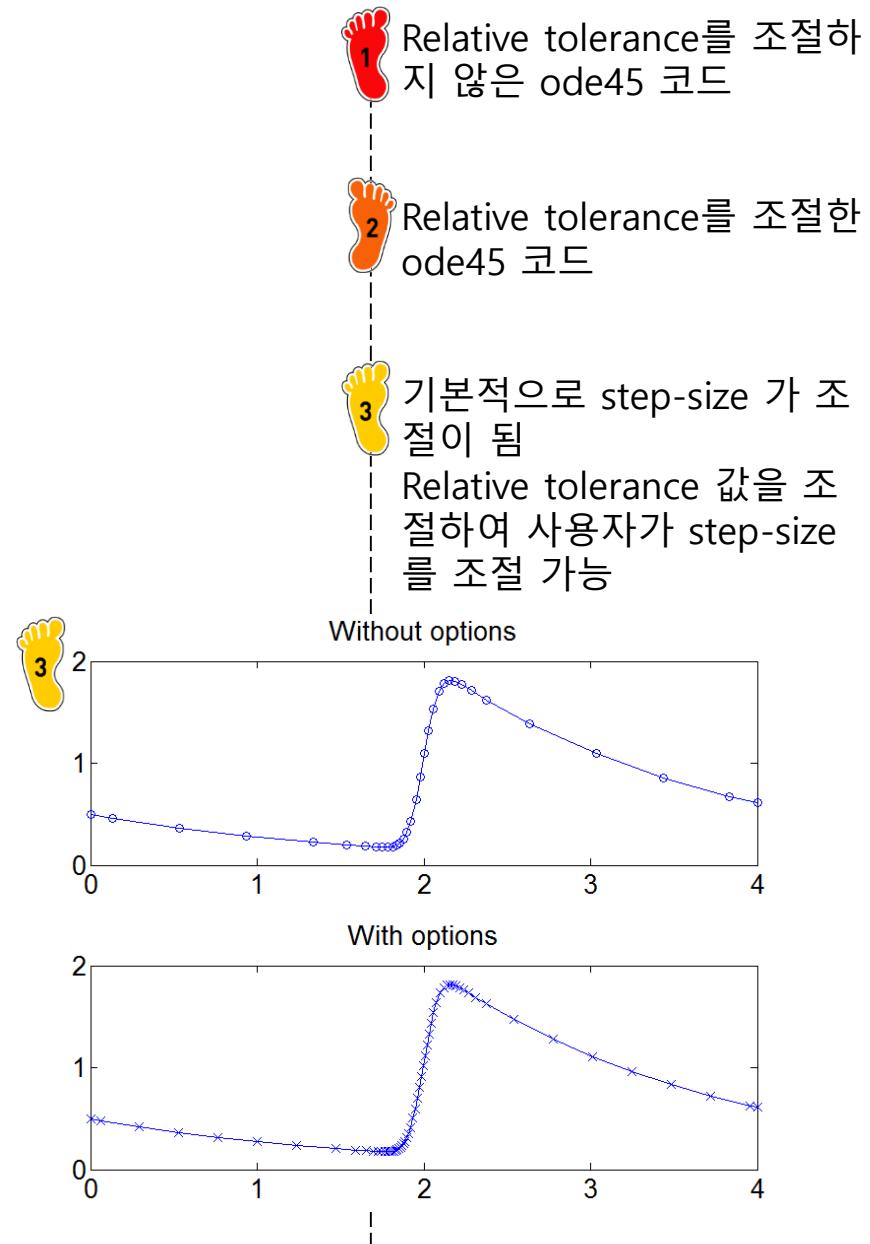
EXAMPLE 25.14: MAIN & RESULT

```

Editor - C:\Users\sean\Desktop\example_III.m
File Edit Text Go Cell Tools Debug Desktop
File Edit Text Go Cell Tools Debug Desktop
1 - clc; clear all; close all;
2 - %% ode45 solution without option control
3 - [t1,y1] = ode23(@dydt,[0 4],0.5);
4 - %% ode45 solution with option control
5 - options = odeset('RelTol',1e-4);
6 - [t2,y2] = ode23(@dydt,[0 4],0.5,options);
7 - %% plot solution
8 - subplot(2,1,1)
9 - plot(t1,y1,'o-')
10 - title('Without options')
11 - subplot(2,1,2)
12 - plot(t2,y2,'o-')
13 - title('With options')

main.m    main.m    Level_set_main.m    example_III.m    Heun.m
script    Ln 2    Col 1    OVR

```



- **Matlab bulit-in functions for stiff systems**
 - ✓ **ode15s**
 - ✓ **ode23s**
 - ✓ **ode23t**
 - ✓ **ode23tb**
 - ✓ **Example**

ODE_{XS} BUILT IN FUNCTIONS

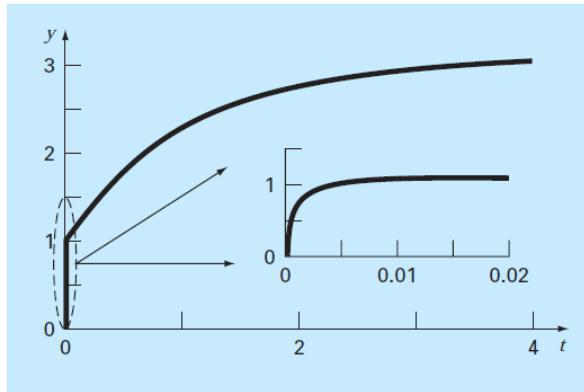


FIGURE 26.1

Plot of a stiff solution of a single ODE. Although the solution appears to start at 1, there is actually a fast transient from $y = 0$ to 1 that occurs in less than 0.005 time unit. This transient is perceptible only when the response is viewed on the finer timescale in the inset.

`ode15s`: This function is a variable-order solver based on numerical differentiation formulas. It is a multistep solver that optionally uses the Gear backward differentiation formulas. This is used for stiff problems of low to medium accuracy.

`ode23s`: This function is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than `ode15s` at crude tolerances. It can solve some kinds of stiff problems better than `ode15s`.

`ode23t`: This function is an implementation of the trapezoidal rule with a “free” interpolant. This is used for moderately stiff problems with low accuracy where you need a solution without numerical damping.

`ode23tb`: This is an implementation of implicit Runge-Kutta formula with a first stage that is a trapezoidal rule and a second stage that is a backward differentiation formula of order 2. This solver may also be more efficient than `ode15s` at crude tolerances.

EXAMPLE 27.10

van der Pol equation

$$\frac{d^2 y_1}{dt^2} - \mu(1 - y_1^2) \frac{dy_1}{dt} + y_1 = 0$$

initial condition

$$t = 0, y_1 = 1, y_2 = 1$$

convert process

$$\begin{cases} \frac{dy_1}{dt} = y_2 \\ \frac{dy_2}{dt} = \mu(1 - y_1^2)y_2 - y_1 \end{cases}$$



stiff 한 정도가 μ 값에 따라 변하는 van der Pol equation.

In 1920 the Dutch physicist Balthasar van der Pol studied a differential equation that describes the circuit of a vacuum tube.

It has been used to model other phenomenon such as the human heartbeat by Johannes van der Mark.

```

Editor - C:\Users\sean\Desktop\vanderpol.m
File Edit Text Go Cell Tools Debug Desktop
File Edit Text Go Cell Tools Debug Desktop
1 function yp = vanderpol(t,y,mu)
2     yp = [y(2); mu*(1-y(1)^2)*y(2)-y(1)];
3 end

```

The code in the editor is:

```

function yp = vanderpol(t,y,mu)
yp = [y(2); mu*(1-y(1)^2)*y(2)-y(1)];
end

```



매틀랩 함수

EXAMPLE 27.10: MAIN & RESULT

The screenshot shows the MATLAB Editor window with the following code:

```

Editor - C:\Users\sean\Desktop\example.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Editor Cell Window Help
1 - clc; clear all; close all;
2 - %% ode45 solution when mu equals 1
3 - [t_45a,y_45a] = ode45(@vanderpol,[0 20],[1,1],[],1);
4 - %% ode45 solution when mu equals 1000
5 - [t_45b,y_45b] = ode45(@vanderpol,[0 20],[1,1],[],1000);
6 - %% ode23s solution when mu equals 1000
7 - [t_23s,y_23s] = ode23s(@vanderpol,[0 6000],[1,1],[],1000);
8 - %% plot solution
9 - subplot(3,1,1)
10 - plot(t_45a,y_45a(:,1))
11 - title('ode45 solution when mu equals 1')
12 - subplot(3,1,2)
13 - plot(t_45b,y_45b(:,1))
14 - title('ode45 solution when mu equals 1000')
15 - subplot(3,1,3)
16 - plot(t_23s,y_23s(:,1))
17 - title('ode23s solution when mu equals 1000')

Heun.m x example II.m x predprev.m x dvdt.m x ex
t_23s 1827x1 double
t_45a 213x1 double
t_45b 67705x1 double
y_23s 1827x2 double
y_45a 213x2 double
y_45b 67705x2 double

```

The code defines three sets of solutions for the van der Pol oscillator. The first set uses ode45 for $\mu = 1$ over 20 time units, resulting in 213 points. The second set uses ode45 for $\mu = 1000$ over 20 time units, resulting in 67705 points. The third set uses ode23s for $\mu = 1000$ over 6000 time units, resulting in 1827 points.

Three plots are generated:

- ode45 solution when mu equals 1:** A smooth oscillating curve from 0 to 20 time units.
- ode45 solution when mu equals 1000:** A step function with 213 discrete points from 0 to 20 time units.
- ode23s solution when mu equals 1000:** A step function with 1827 discrete points from 0 to 6000 time units.

- 1 mu = 1 일 때 실행 코드
- 2 mu = 1000 일 때 ode45로 푸는 코드
결과는 그래프 2 번째 그림
이며 20초 계산하는데
67705 pt. 필요
- 3 mu = 1000 일 때 ode23s로 푸는 코드
결과는 그래프 3번째 그림
이며 6000초 계산하는데
1827 pt. 필요 (ode45로
20초 푸는 pt. 개수의 약
1/37 배)
- 따라서 ode45로는 해당 미
분방정식을 푸는 것은 매우
비 효율적(또한 결과의 정확
도 보장하기 어려움)

EXAMPLE 27.10: MAIN & RESULT

<http://kr.mathworks.com/company/newsletters/articles/stiff-differential-equations.html>

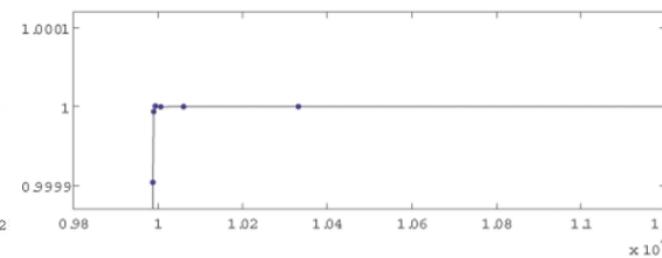
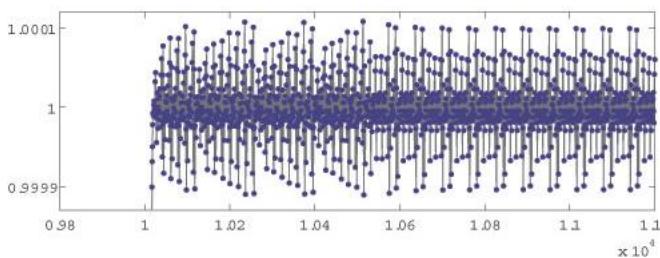
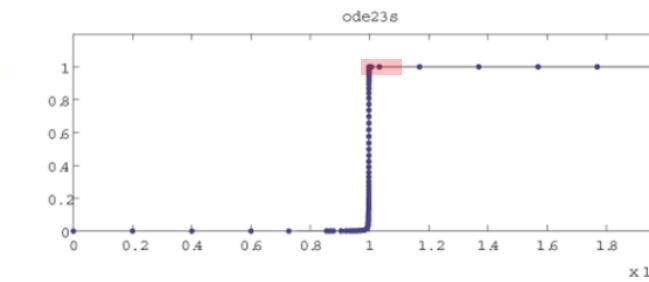
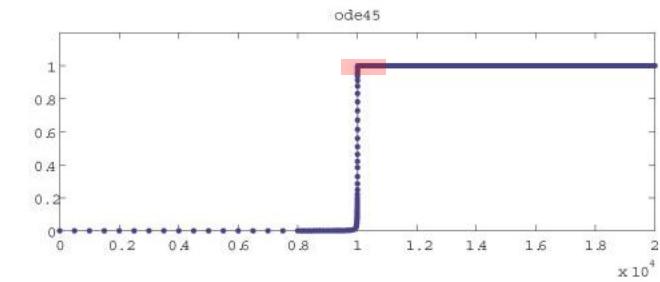


1 참고자료

Stiff Differential Equations

By Cleve Moler, MathWorks

Stiffness is a subtle, difficult, and important - concept in the numerical solution of ordinary differential equations.



SUMMARY

Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	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 evaluate.
ode15s	Stiff	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.

Parameters	ode45	ode23	ode113	ode15s	ode23s	ode23t	ode23tb
RelTol, AbsTol, NormControl	✓	✓	✓	✓	✓	✓	✓
OutputFcn, OutputSel, Refine, Stats	✓	✓	✓	✓	✓	✓	✓
NonNegative	✓	✓	✓	✓*	—	✓*	✓*
Events	✓	✓	✓	✓	✓	✓	✓
MaxStep, InitialStep	✓	✓	✓	✓	✓	✓	✓
Jacobian, JPatten, Vectorized	—	—	—	✓	✓	✓	✓
Mass	✓	✓	✓	✓	✓	✓	✓
MStateDependence	✓	✓	✓	✓	—	✓	✓
MvPattern	—	—	—	✓	—	✓	✓
MassSingular	—	—	—	✓	—	✓	—
InitialSlope	—	—	—	✓	—	✓	—
MaxOrder, BDF	—	—	—	✓	—	—	—

- **Boundary Value Problems for ODE**
 - ✓ **Shooting method for linear ODE**
 - ✓ **Shooting method for nonlinear ODE**
 - ✓ **Finite difference method**

EXAMPLE

- Heat balance for a long, thin rod

- Not insulated along its length
- Steady state

$$\frac{d^2T}{dx^2} + h'(T_a - T) = 0$$

$$\left. \begin{array}{l} T(0) = T_1 = 40^\circ C \\ T(L) = T_2 = 200^\circ C \end{array} \right\} \text{boundary conditions}$$

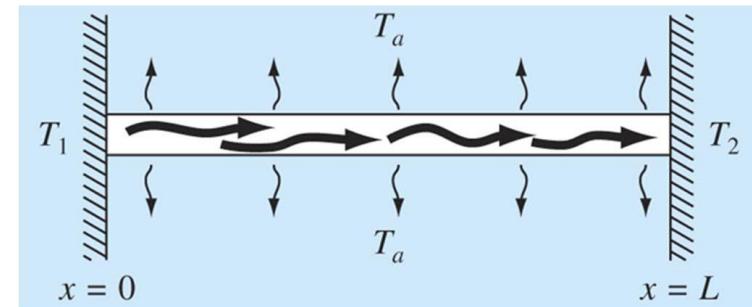
$T_a = 20^\circ C$ (temperature of the surrounding air)

$$L = 10 \text{ m}$$

$h' = 0.01 \text{ m}^{-2}$ (heat transfer coefficient)

rate of heat dissipation to the surrounding air

$$\text{analytic solution: } T = 73.4523e^{0.1x} - 53.4523e^{-0.1x} + 20$$



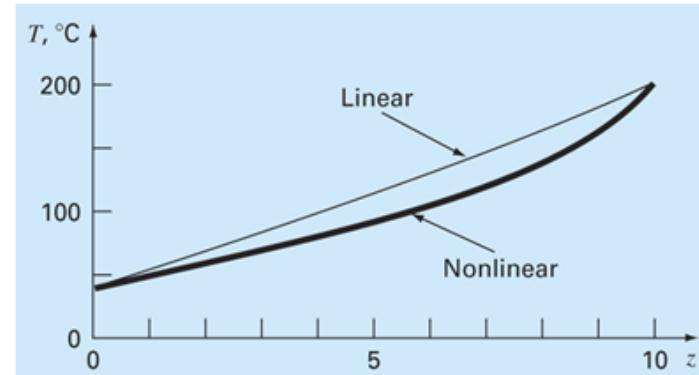
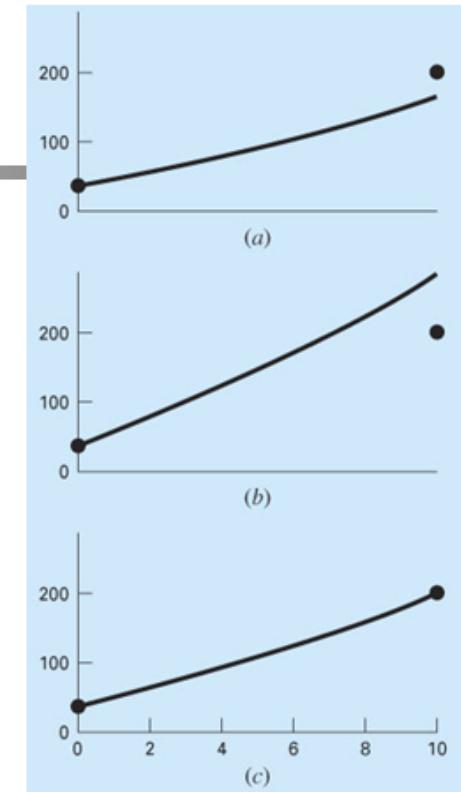
EXAMPLE

Shooting Method

$$\begin{cases} \frac{dT}{dx} = z \\ \frac{dz}{dx} = h'(T - T_a) \text{ [linear]} \quad \frac{dz}{dx} = h''(T - T_a)^4 \text{ [nonlinear]} \end{cases}$$

4th RK, $h = 2, T(10) = 200$

$$\rightarrow \begin{cases} \text{guess / linear: } \begin{cases} z(0) = 10 \rightarrow T(10) = 168.3797 \\ z(0) = 20 \rightarrow T(10) = 285.8980 \end{cases} \\ \xrightarrow{\text{linear interpolation}} z(0) = 12.6907 \\ \text{non-linear: } T(10) = f(z(0)) \\ \rightarrow g(z(0)) = f(z(0)) - 200 \end{cases}$$



SHOOTING METHOD: LINEAR

```

Editor - C:\Users\sean\Desktop\heatfun.m
File Edit Text Go Cell Tools Debug Desktop Window Help
function dy = heatfun(x,y)
dy = [y(2); -0.01*(20-y(1))];
```

The code defines a function `heatfun` that takes `x` and `y` as inputs. It returns a vector `dy` containing two elements: `y(2)` and `-0.01*(20-y(1))`. A red box highlights the second line of the function definition.

```

Editor - C:\Users\sean\Desktop\ex1.m
File Edit Text Go Cell Tools Debug Desktop Window Help
clc; clear all;
[x1,y1] = ode45(@heatfun, [0 10], [40 10]);
Tb1 = y1(length(y1(:,1)));
[x2,y2] = ode45(@heatfun, [0 10], [40 20]);
Tb2 = y2(length(y2(:,1)));
za = 10+(20-10)/(Tb2-Tb1)*(200-Tb1);
[x3,y3] = ode45(@heatfun, [0 10], [40 za]);
plot(x3,y3(:,1));
Tb = y3(length(y3(:,1)))
```

The script starts by clearing the workspace. It then calls the `heatfun` function twice using `ode45` with initial conditions `[40 10]` and `[40 20]` respectively, over the interval `[0 10]`. The results are stored in `x1, y1` and `x2, y2`. The boundary temperatures `Tb1` and `Tb2` are extracted from the end points of `y1` and `y2` respectively. The脚注³ value `za` is calculated as $10 + \frac{20-10}{Tb2-Tb1} \cdot (200-Tb1)$. Finally, it calls `ode45` again with `za` as the initial condition for `y3`, plots the result, and extracts the final temperature `Tb`.



아래에 표기된 시스템 ODE 함수를 `heatfun` 이름으로 저장

$$\begin{aligned} \frac{dT}{dx} &= z \\ \frac{dz}{dx} &= -0.01(20-T) \end{aligned} \quad \left. \begin{array}{l} y(1)=T \\ y(2)=z \end{array} \right\}$$

$$\frac{dy(1)}{dx} = y(2)$$

$$\frac{dy(2)}{dx} = -0.01(20 - y(1))$$



z_{a1} 이 10 일 때와 z_{a2} 가 20 일 때 값을 T_{b1} 과 T_{b2} 에 저 장



아래 수식을 이용하여 z_a 를 선형 보간

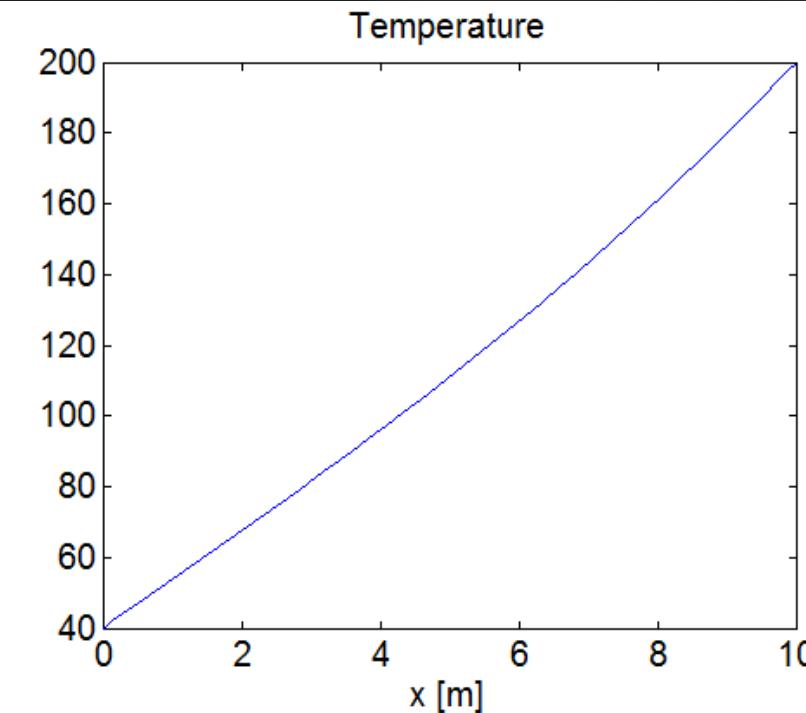
$$z_a = z_{a1} + \frac{z_{a2} - z_{a1}}{T_{b2} - T_{b1}} (T_b - T_{b1})$$

RESULT

Command Window

File Edit Debug Desktop Window Help

```
za =  
12.6905  
Tb =  
fx 200.0000
```



1 z_a 가 12.6905 일 때 B.C. 이 일치

2 결과 그래프

SHOOTING METHOD: NONLINEAR

$$\begin{cases} \frac{dT}{dx} = z \\ \frac{dz}{dx} = -h'(T_{\infty} - T) - h''(T_{\infty}^4 - T^4) \end{cases}$$

```

Editor - C:\Users\sean\Desktop\dydxn.m
File Edit Text Go Cell Tools Debug Desktop Window Help
function dy = dydxn(x,y)
dy = [y(2); -0.05*(200-y(1))-2.7e-9*(1.6e9-y(1)^4)];

```

assignment6.m ex.m heatfun.m ex1.m dydxn.m res.m Untitled5

dydxn Ln 2 Col 53 OVR

```

Editor - C:\Users\sean\Desktop\res.m
File Edit Text Go Cell Tools Debug Desktop Window Help
function r=res(za)
[x,y] = ode45(@dydxn, [0 10], [300 za]);
r = y(length(x),1)-400;

```

assignment6.m ex.m heatfun.m ex1.m dydxn.m res.m Untitled5

res Ln 3 Col 24 OVR



아래에 표기된 상수대로 왼쪽의 ODE 함수를 입력

$$h' = 0.05 \text{ m}^{-2}$$

$$h'' = 2.7 \times 10^{-9} \text{ K}^{-3} \text{ m}^{-2}$$

$$T_{\infty} = 200 \text{ K}$$

$$T(0) = 300 \text{ K}$$

$$T_{\infty}(10) = 400 \text{ K}$$



근을 찾기 위한 함수 설정

r 값(residual: 해석해와 수치해의 차이)이 0 일 경우 정확한 z_a 값

SHOOTING METHOD: NONLINEAR

The screenshot shows the MATLAB Editor window with the file name `C:\Users\sean\Desktop\ex2_1.m`. The code in the editor is:

```

1 - clc; clear all;
2 - fzero(@res,-50)

```

A red box highlights the second line of code, and a red footprint icon with the number 1 is placed next to it.

1 `fzero(@funname, IV)`

`funname`의 근을 찾는 매틀
랩 함수
`IV` 는 initial value 를 뜻함

[주의] 비선형함수이므로 초기값에 따른 영향 존재

The screenshot shows the MATLAB Command Window. The command entered was:

```

ans =
-41.7434

```

A red box highlights the output value, and an orange footprint icon with the number 2 is placed next to it.

2 결과 $z_a = -41.7434$

즉, 이 값을 이용하여 ode
함수를 풀면 nonlinear BC
를 만족하는 해

RESULT

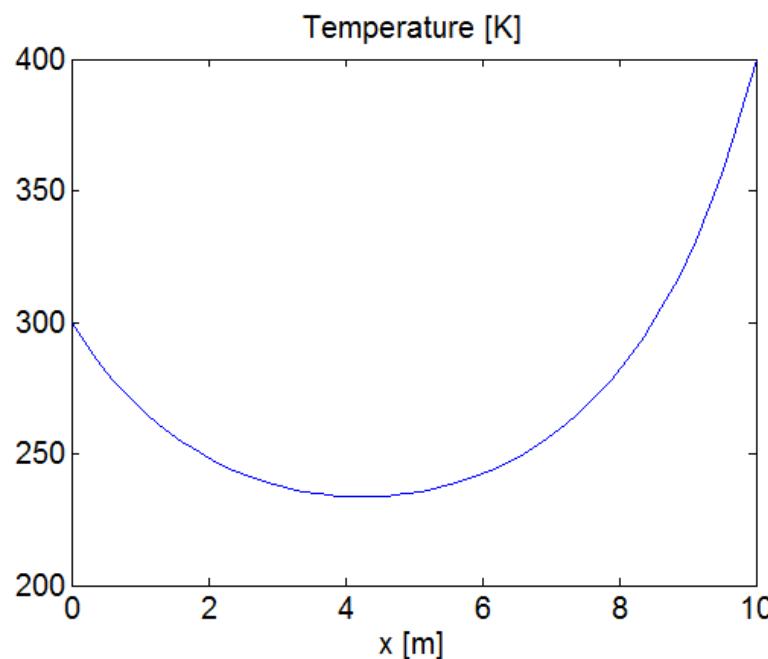
Editor - C:\Users\sean\Desktop\ex2_2.m

```
1 - clc; clear all;
2 - [x,y] = ode45(@dydxn,[0 10],[300 fzero(@res,-50)]);
3 - plot(x,y(:,1))
```

The screenshot shows the MATLAB Editor window with a script named 'ex2_2.m'. The script contains three lines of code: 'clc; clear all;', '[x,y] = ode45(@dydxn,[0 10],[300 fzero(@res,-50)]);', and 'plot(x,y(:,1))'. The third line is highlighted with a red box. A red footprint icon with the number '1' is positioned to the right of the editor window.

1 혹은 initial 값을 입력하는
곳에 fzero 함수를 직접 입
력 가능

2 결과 그래프



FINITE DIFFERENCE METHOD

Forward FDM	$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$	$O(h)$
	$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h}$	$O(h^2)$
	$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}$	$O(h)$
	$f''(x_i) = \frac{-f(x_{i+3}) + 4f(x_{i+2}) - 5f(x_{i+1}) + 2f(x_i)}{h^2}$	$O(h^2)$
Backward FDM	$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h}$	$O(h)$
	$f'(x_i) = \frac{3f(x_i) - 4f(x_{i-1}) + f(x_{i-2})}{2h}$	$O(h^2)$
	$f''(x_i) = \frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2}$	$O(h)$
	$f''(x_i) = \frac{2f(x_i) - 5f(x_{i-1}) + 4f(x_{i-2}) - f(x_{i-3})}{h^2}$	$O(h^2)$
Centered FDM	$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$	$O(h^2)$
	$f'(x_i) = \frac{-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12h}$	$O(h^4)$
	$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$	$O(h^2)$
	$f''(x_i) = \frac{-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2})}{12h^2}$	$O(h^4)$

FINITE DIFFERENCE METHOD

$$\frac{d^2T}{dx^2} = -h'(T_\infty - T) \rightarrow \frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2} = -h'(T_\infty - T_i)$$

$$\rightarrow -T_{i-1} + (2 + h' \Delta x^2) T_i - T_{i+1} = h' \Delta x^2 T_\infty$$

```

Editor - C:\Users\sean\Desktop\FDM_ex1.m
File Edit Text Go Cell Tools Debug Desktop Window Help
File Edit Text Go Cell Tools Debug Desktop Window Help
1 - clc; clear all;
2
3 - step_size = 2; h = 0.05; T_inf = 200;
4 - length = 10; T_a = 300; T_b = 400;
5
6 - number = length/step_size;
7
8 - K = zeros((number-1),(number-1));
9 - f = ones((number-1),1)*h*step_size^2*T_inf;
10
11 - f(1,1) = f(1,1) + T_a;
12 - f((number-1),1) = f((number-1),1) + T_b;
13
14 - i = 1:3;
15 - a = [-1, 2+h*step_size^2,-1];
16
17 - for j = 1:(number-1)
18 -     K(j,i+j-1) = a;
19 - end
20
21 - K = K(1:(number-1),2:number);
22
23 - T_sol = K\f;
24 - T_total = [T_a;T_sol;T_b];
25 - plot([0:step_size:length],T_total)

```



[fixed / Dirichlet boundary condition]

$$\rightarrow \begin{cases} i=1 : -T_0 + (2 + h' \Delta x^2) T_1 - T_2 = h' \Delta x^2 T_a \\ \vdots \\ [K] \quad \{T\} \quad \{f\} \\ i=n : -T_{n-1} + (2 + h' \Delta x^2) T_n - T_{n+1} = h' \Delta x^2 T_a \end{cases}$$

	1	2	3	4	
1	2.2000	-1	0	0	$\begin{Bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 340 \\ 40 \\ 440 \end{Bmatrix}$
2	-1	2.2000	-1	0	
3	0	-1	2.2000	-1	
4	0	0	-1	2.2000	

$$[K]\{T\} = \{f\}$$

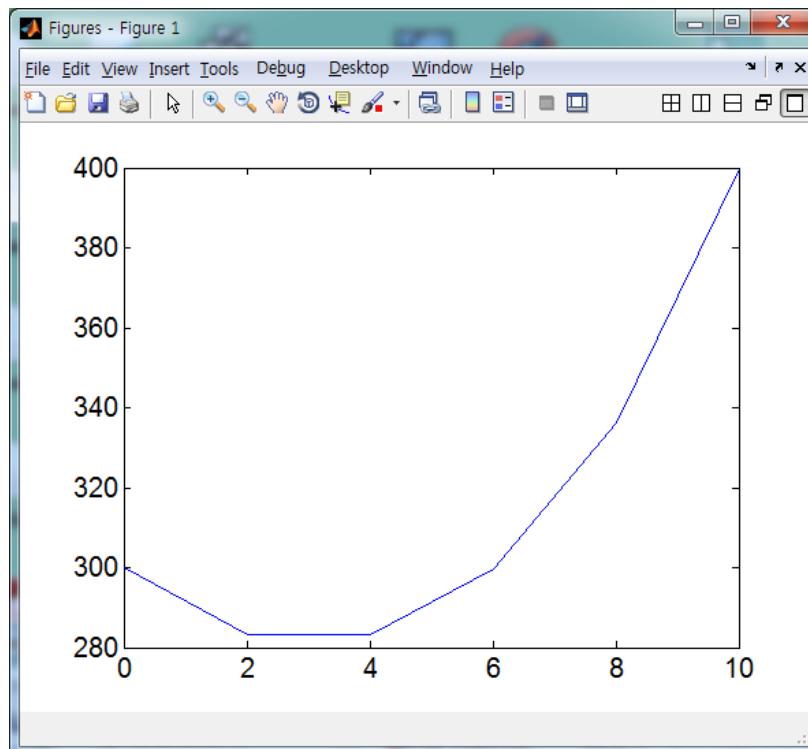
$$\{T\} = [K]^{-1}\{f\}$$

$$[K]\{T\} = \{f\}$$

RESULT

Command Window

```
File Edit Debug Desktop Window Help
T_total =
300.0000
283.2660
283.1853
299.7416
336.2462
400.0000
fx|
```



- 1 결과 온도 벡터
- 2 결과 그래프

- **Case study**
- **Assignment**

CASE STUDY I

Background. Mechanical engineers (as well as all other engineers) are frequently faced with problems concerning the periodic motion of free bodies. The engineering approach to such problems ultimately requires that the position and velocity of the body be known as a function of time. These functions of time invariably are the solution of ordinary differential equations. The differential equations are usually based on Newton's laws of motion.

As an example, consider the simple pendulum shown previously in Fig. PT7.1. The particle of weight W is suspended on a weightless rod of length l . The only forces acting on the particle are its weight and the tension R in the rod. The position of the particle at any time is completely specified in terms of the angle θ and l .

The free-body diagram in Fig. 28.16 shows the forces on the particle and the acceleration. It is convenient to apply Newton's laws of motion in the x direction tangent to the path of the particle:

$$\Sigma F = -W \sin \theta = \frac{W}{g} a$$

where g = the gravitational constant (32.2 ft/s^2) and a = the acceleration in the x direction. The angular acceleration of the particle (α) becomes

$$\alpha = \frac{a}{l}$$

Therefore, in polar coordinates ($\alpha = d^2\theta/dt^2$),

$$-W \sin \theta = \frac{Wl}{g} \alpha = \frac{Wl}{g} \frac{d^2\theta}{dt^2}$$

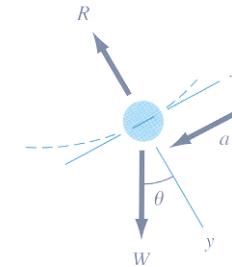
or

$$\frac{d^2\theta}{dt^2} + \frac{g}{l} \sin \theta = 0 \quad (28.15)$$

This apparently simple equation is a second-order nonlinear differential equation. In general, such equations are difficult or impossible to solve analytically. You have two choices regarding further progress. First, the differential equation might be reduced to a form that can be solved analytically (recall Sec. PT7.1.1), or second, a numerical approximation technique can be used to solve the differential equation directly. We will examine both of these alternatives in this example.

FIGURE 28.16

A free-body diagram of the swinging pendulum showing the forces on the particle and the acceleration.



$$t_{span} = [0 \ 10]$$

$$\sin \theta \approx \theta$$

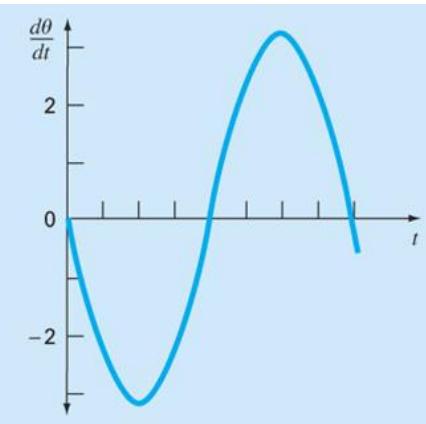
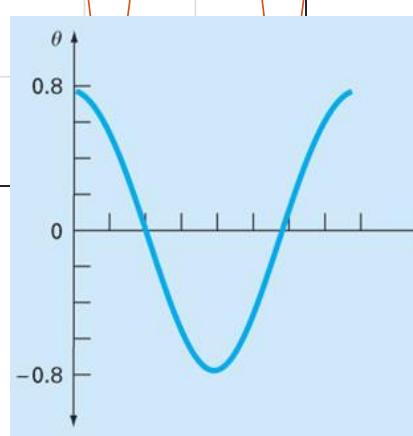
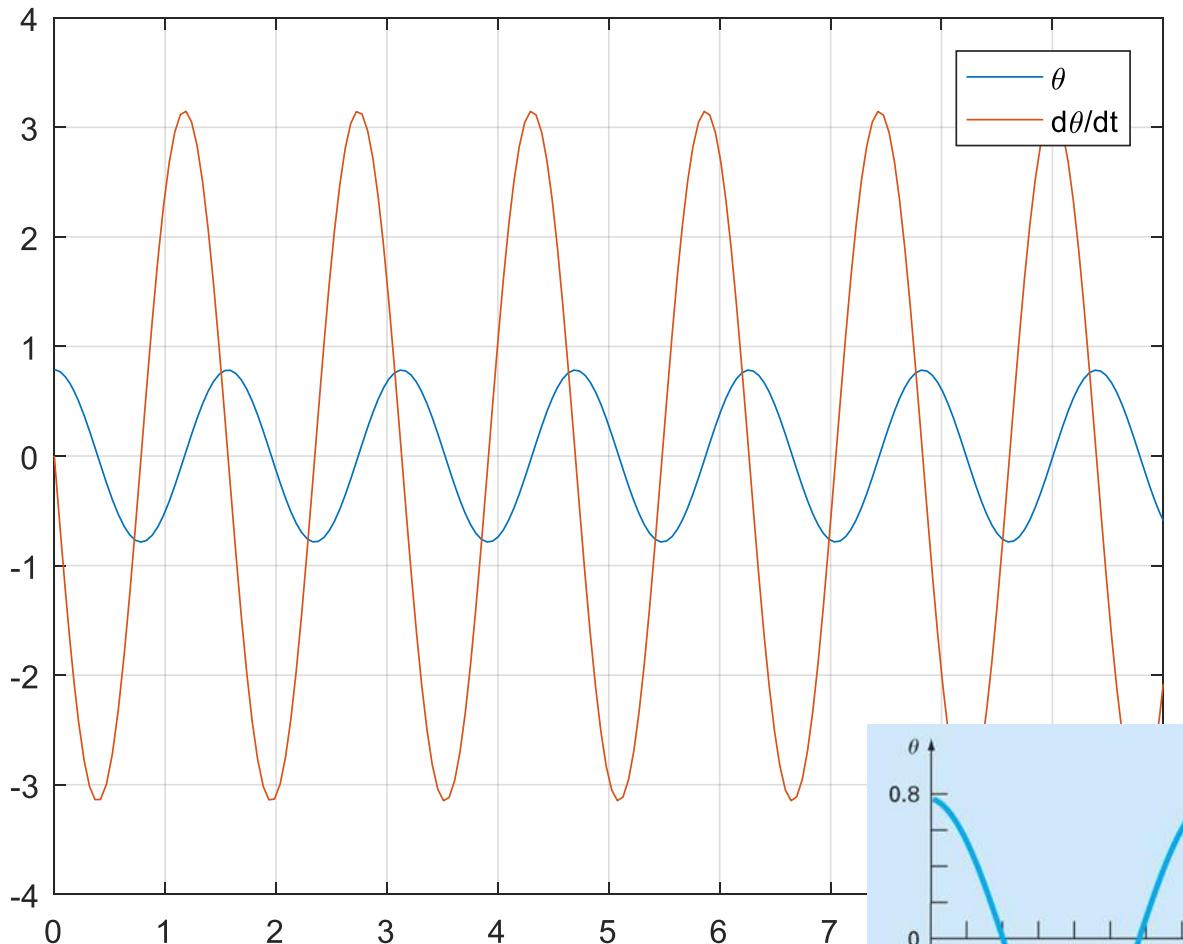
$$\theta_0 = \pi / 4$$

$$g = 9.81 \text{ m/s}^2$$

$$l = 0.6096 \text{ m}$$

**Input
Parameters**

CASE I: RESULT



CASE STUDY II

Background. Electric circuits where the current is time-variable rather than constant are common. A transient current is established in the right-hand loop of the circuit shown in Fig. 28.11 when the switch is suddenly closed.

Equations that describe the transient behavior of the circuit in Fig. 28.11 are based on Kirchhoff's law, which states that the algebraic sum of the voltage drops around a closed loop is zero (recall Sec. 8.3). Thus,

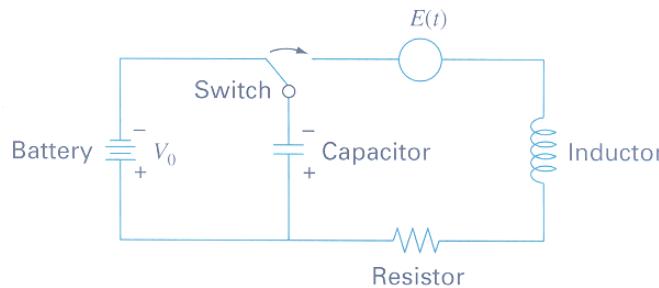
$$L \frac{di}{dt} + Ri + \frac{q}{C} - E(t) = 0 \quad (28.9)$$

where $L(di/dt)$ = voltage drop across the inductor, L = inductance (H), R = resistance (Ω), q = charge on the capacitor (C), C = capacitance (F), $E(t)$ = time-variable voltage source (V), and

$$i = \frac{dq}{dt} \quad (28.10)$$

FIGURE 28.11

An electric circuit where the current varies with time.



$$t_{span} = [0 \ 100]$$

$$E(t) = E_0 \sin(\omega t)$$

$$L = 1 \text{ H}$$

$$E_0 = 1 \text{ V}$$

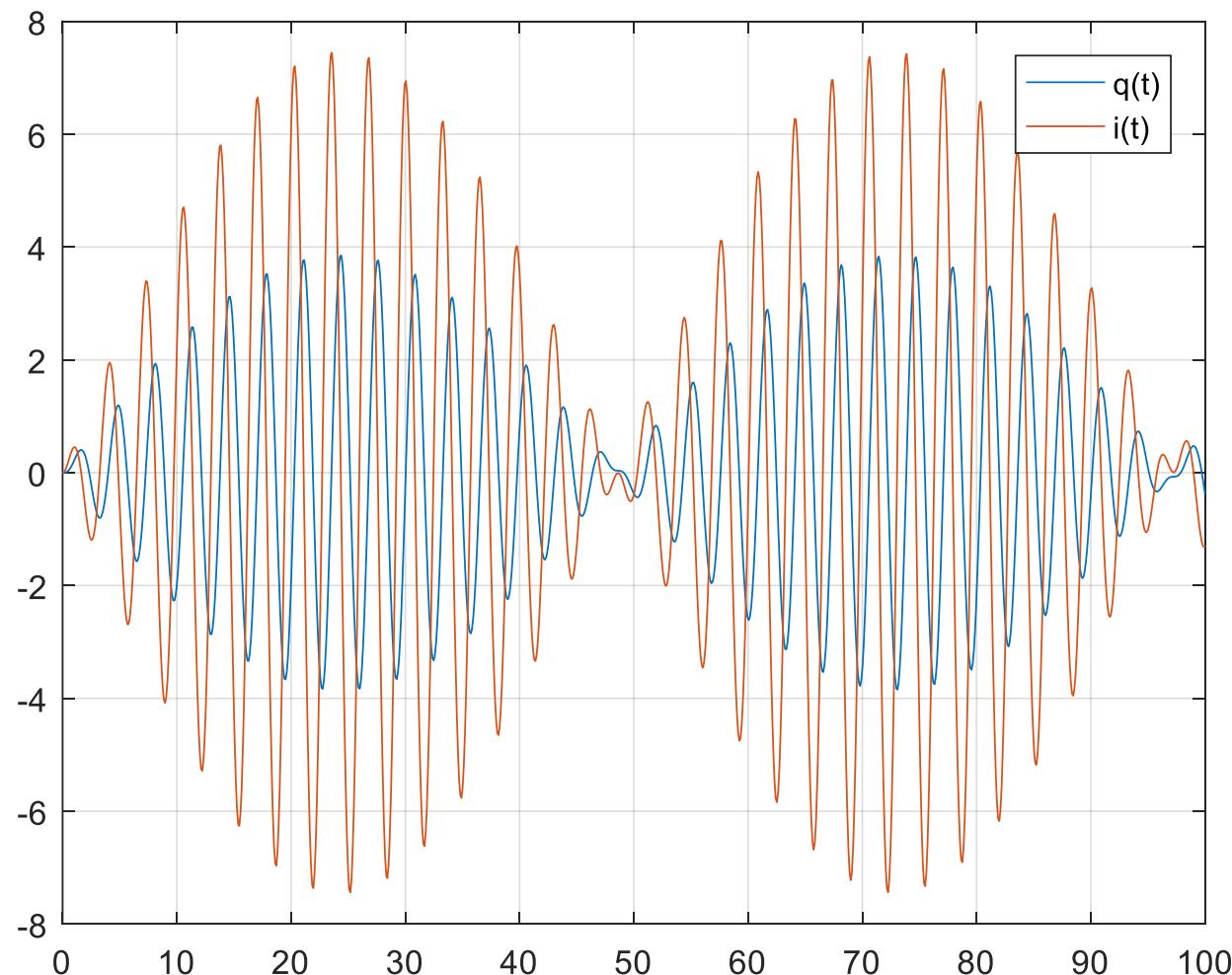
$$C = 0.25 \text{ F}$$

$$\omega = \sqrt{3.5} \text{ rad/s}$$

$$R = 0$$

Input
Parameters

CASE STUDY II: RESULT



ASSIGNMENT

varies with height according to

$$f(z) = \frac{200z}{5+z} e^{-2z/30}$$

Calculate the deflection if $y = 0$ and $dy/dz = 0$ at $z = 0$. Use parameter values of $L = 30$, $E = 1.3 \times 10^8$, and $I = 0.05$ for your computation.

22.21 A pond drains through a pipe as shown in Fig. P22.21. Under a number of simplifying assumptions, the following differential equation describes how depth changes with time:

$$\frac{dh}{dt} = -\frac{\pi d^2}{4A(h)} \sqrt{2g(h+e)}$$

where h = depth (m), t = time (s), d = pipe diameter (m), $A(h)$ = pond surface area as a function of depth (m^2), g = gravitational constant ($= 9.81 \text{ m/s}^2$), and e = depth of pipe outlet below the pond bottom (m). Based on the following area-depth table, solve this differential equation to determine how long it takes for the pond to empty, given that $h(0) = 6 \text{ m}$, $d = 0.25 \text{ m}$, $e = 1 \text{ m}$.

h, m	6	5	4	3	2	1	0
$A(h), \text{m}^2$	1.17	0.97	0.67	0.45	0.32	0.18	0

22.22 Engineers and scientists use mass-spring models to gain insight into the dynamics of structures under the influence of disturbances such as earthquakes. Figure P22.22 shows such a representation for a three-story building. For this case, the analysis is limited to horizontal motion of the structure. Using Newton's second law, force balances can be developed for this system as

$$\frac{d^2x_1}{dt^2} = -\frac{k_1}{m_1}x_1 + \frac{k_2}{m_1}(x_2 - x_1)$$

$$\frac{d^2x_2}{dt^2} = \frac{k_2}{m_2}(x_1 - x_2) + \frac{k_3}{m_2}(x_3 - x_2)$$

$$\frac{d^2x_3}{dt^2} = \frac{k_3}{m_3}(x_2 - x_3)$$

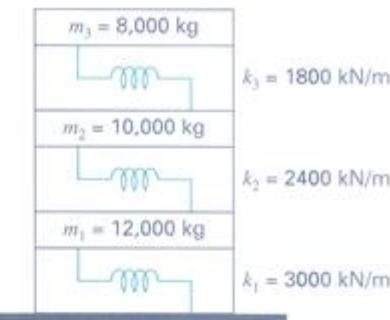


FIGURE P22.22

Simulate the dynamics of this structure from $t = 0$ to 20 s , given the initial condition that the velocity of the ground floor is $dx_1/dt = 1 \text{ m/s}$, and all other initial values of displacements and velocities are zero. Present your results as two time-series plots of (a) displacements and (b) velocities. In addition, develop a three-dimensional phase-plane plot of the displacements.

22.23 Repeat the same simulations as in Section 22.6 for the Lorenz equations but generate the solutions with the midpoint method.

22.24 Perform the same simulations as in Section 22.6 for the Lorenz equations but use a value of $r = 99.96$. Compare your results with those obtained in Section 22.6.

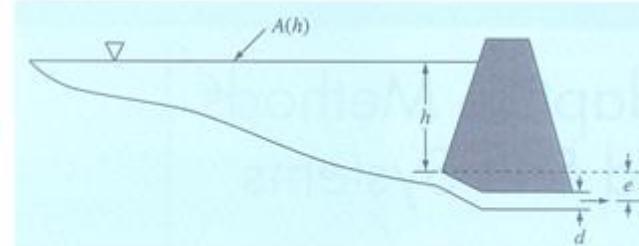


FIGURE P22.21

25.16 The motion of a damped spring-mass system (Fig. P25.16) is described by the following ordinary differential equation:

$$m \frac{d^2x}{dt^2} + c \frac{dx}{dt} + kx = 0$$

where x = displacement from equilibrium position (m), t = time (s), $m = 20\text{-kg}$ mass, and c = the damping coefficient ($\text{N} \cdot \text{s}/\text{m}$). The damping coefficient c takes on three values of 5 (under-damped), 40 (critically damped), and 200 (overdamped). The spring constant $k = 20 \text{ N/m}$. The initial velocity is zero, and the initial displacement $x = 1 \text{ m}$. Solve this equation using a numerical method over the time period $0 \leq t \leq 15 \text{ s}$. Plot the displacement versus time for each of the three values of the damping coefficient on the same curve.

Figure P25.16

