MATLAB ODE Solving

Initial Value Problems

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

CONTENTS

- Basic methods
- Matlab bulit-in functions
- 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

Editor - C:₩Users₩sean₩Desktop₩eulode.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
1   function [t_n,y_n] = eulode(dydt,tspan,y0,h)
2   % eulode: Euler ODE solver
3   %    [t_n,y_n] = eulode(dydt,tspan,y0,h)
4   %                uses Euler`s method to integrate an ODE
5   % input:
6   %    dydt = name of the M-file that evaluates the ODE
7   %    tspan = [ti,tf] where ti and tf = initial and
8   %            final values of independent variable
9   %    y0 = initial value of dependent variable
10  %    h = step size
11  % output:
12  %    t_n = vector of independent variable
13  %    y_n = vector of solution for dependent variable
14  ti = tspan(1);tf = tspan(2);
15  if ~(tf>ti)
16      error('upper limit must be greater than lower')
17  end
18  t_n = (ti:h:tf)'; n = length(t_n);
19  y_n = y0*ones(n,1); % preallocate y to improve efficiency
20  for i = 1:n-1 % implement Euler`s method
21      y_n(i+1) = y_n(i) + dydt(t_n(i),y_n(i))*h;
22  end
23  end
```

1

main.m | Level_set_main.m | HarmonicFind.m | Euler.m | eulode.m

eulode Ln 18 Col 35 OVR

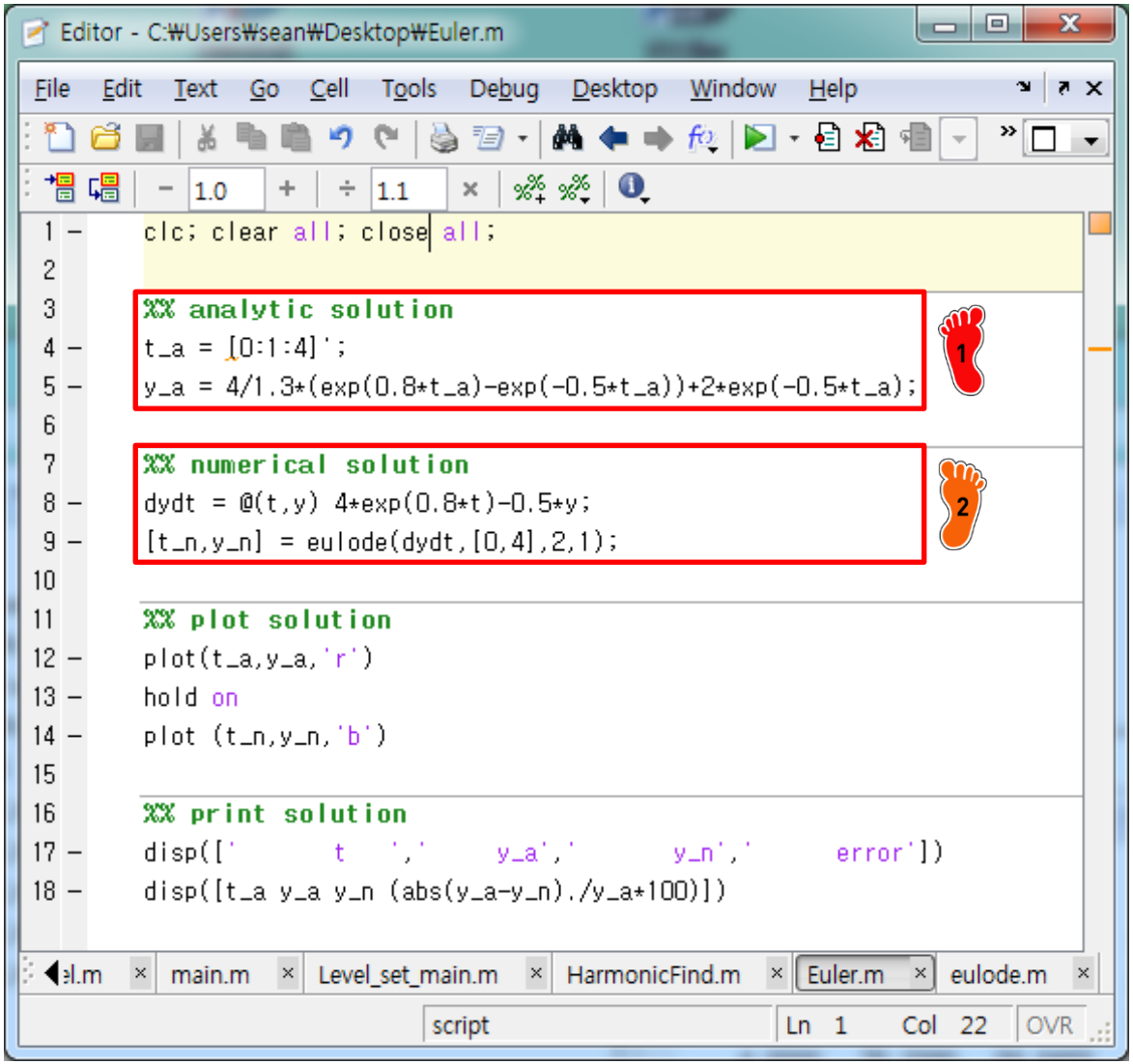
1 오일러 방법을 적용한 메인 함수 코드

Euler`s method

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

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

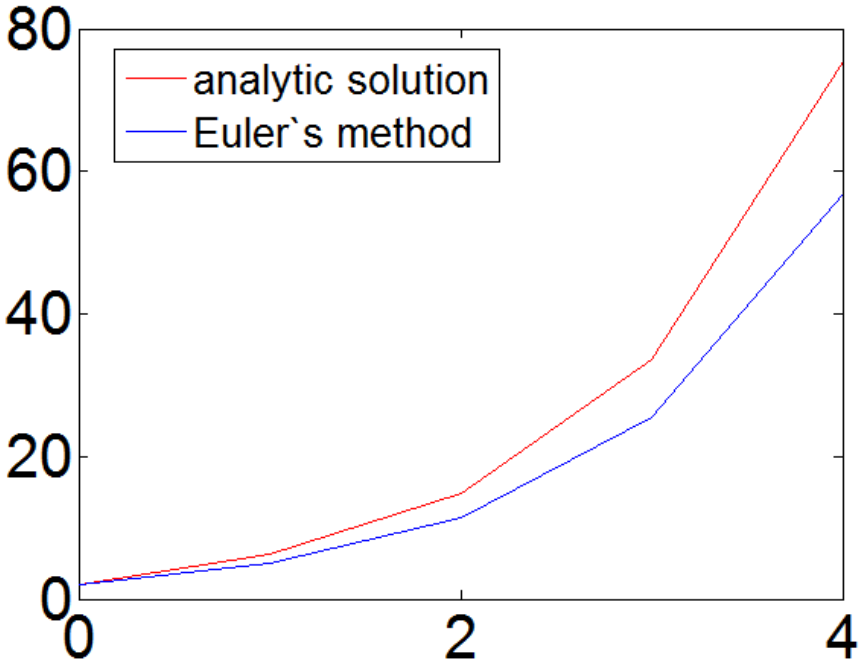
EULER`S METHOD: MAIN



1 Analytical solution을 구하는 코드

2 전 페이지에서 정의한 함수를 이용하여 수치적인 해를 구하는 코드

EULER`S METHOD: RESULTS



1

Command Window

File Edit Debug Desktop Window Help

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 >>

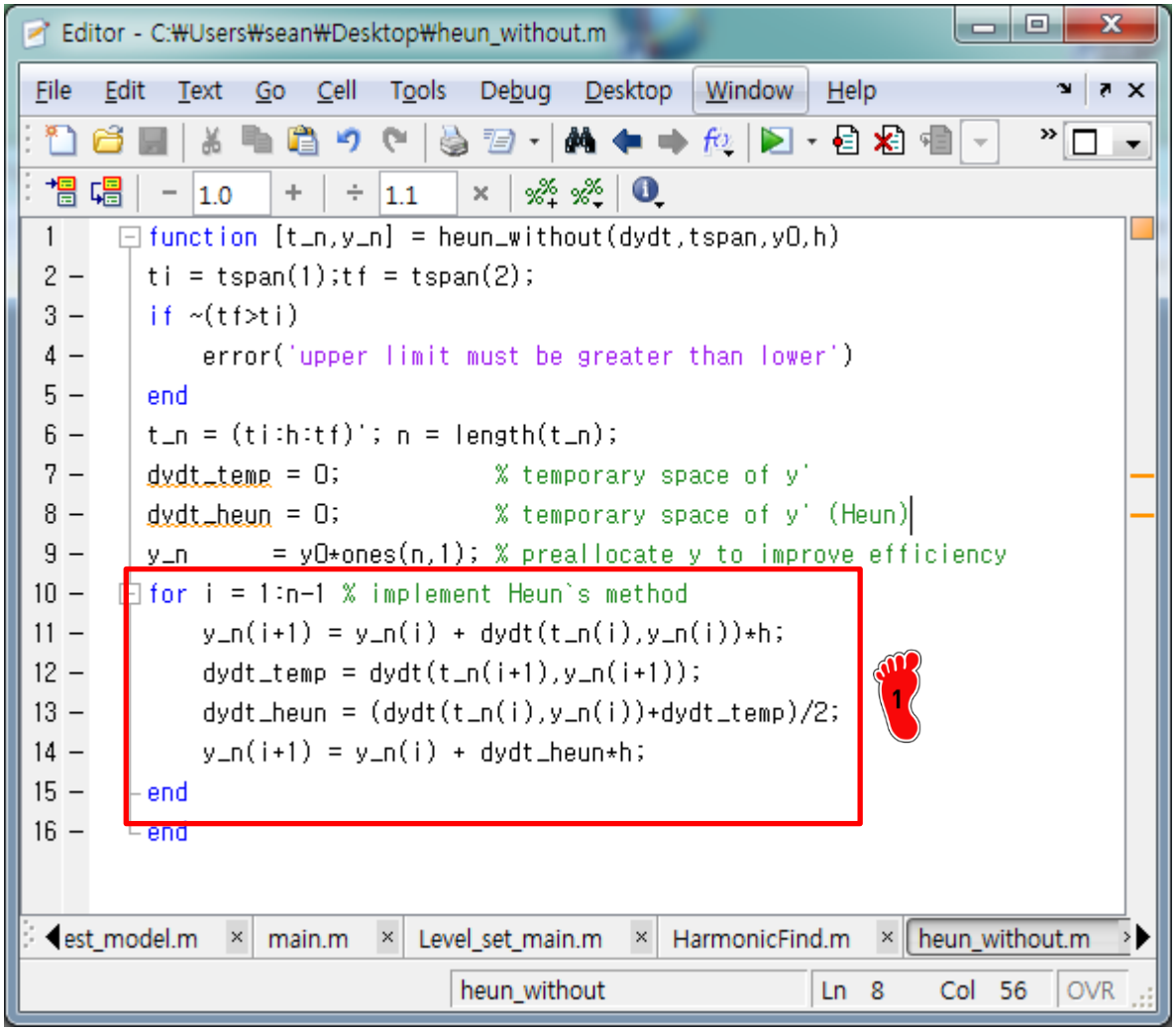
OVR

2

1 그래프

2 독립 변수 step 별로 종속 변수 및 에러 출력

HEUN`S METHOD: FUNCTION



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

Heun`s method

Predictor

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

Corrector

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

HEUN`S METHOD: FUNCTION

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

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
```

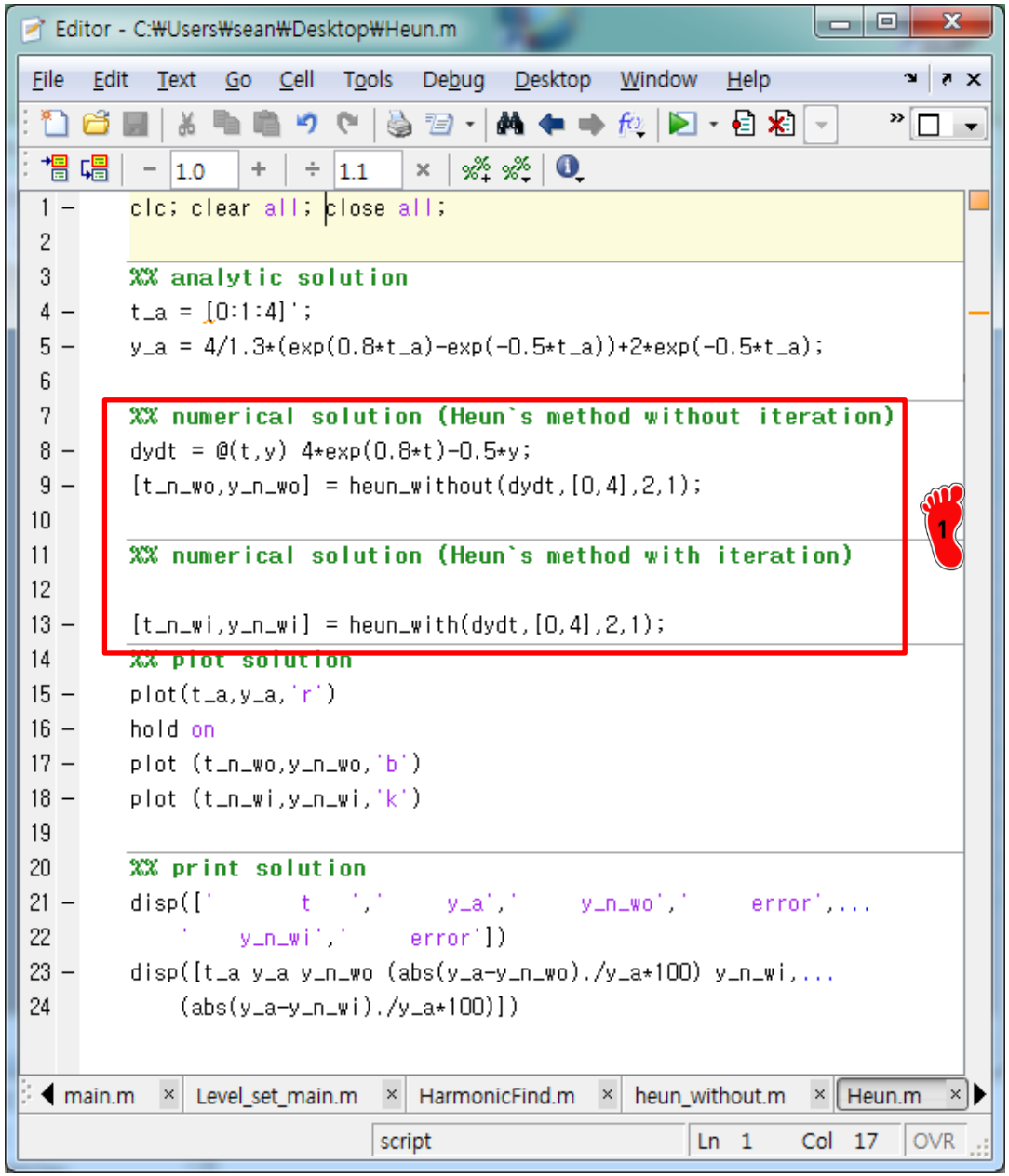
1

el_set_main.m | HarmonicFind.m | heun_without.m | Heun.m | heun_with.m

heun_with Ln 11 Col 40 OVR

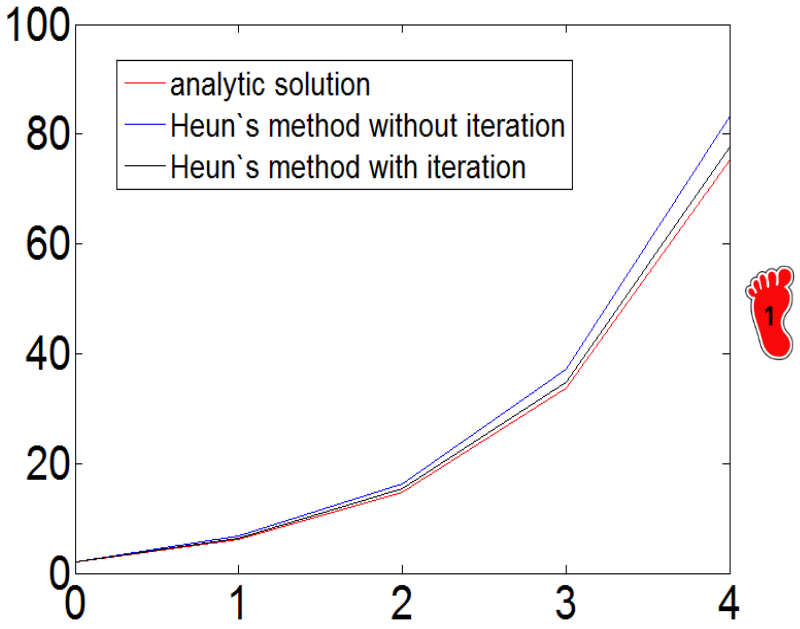
1 Heun`s method 의 corrector update 를 하는 메인 코드

HEUN`S METHOD: MAIN



1 Heun`s method 의 corrector update 를 구분하여 함수 호출

HEUN`S METHOD: RESULTS



1 그래프

2 독립 변수 step 별로 종속 변수 및 에러 출력

Command Window

File Edit Debug Desktop Window Help

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 >>

OVR

MIDPOINT METHOD: FUNCTION

Editor - C:₩Users₩sean₩Desktop₩midpoint.m

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 | main.m | Level_set_main.m | HarmonicFind.m | midpoin

midpoint Ln 15 Col 4 OVR

1 midpoint method 메인 함수 코드

Midpoint`s method

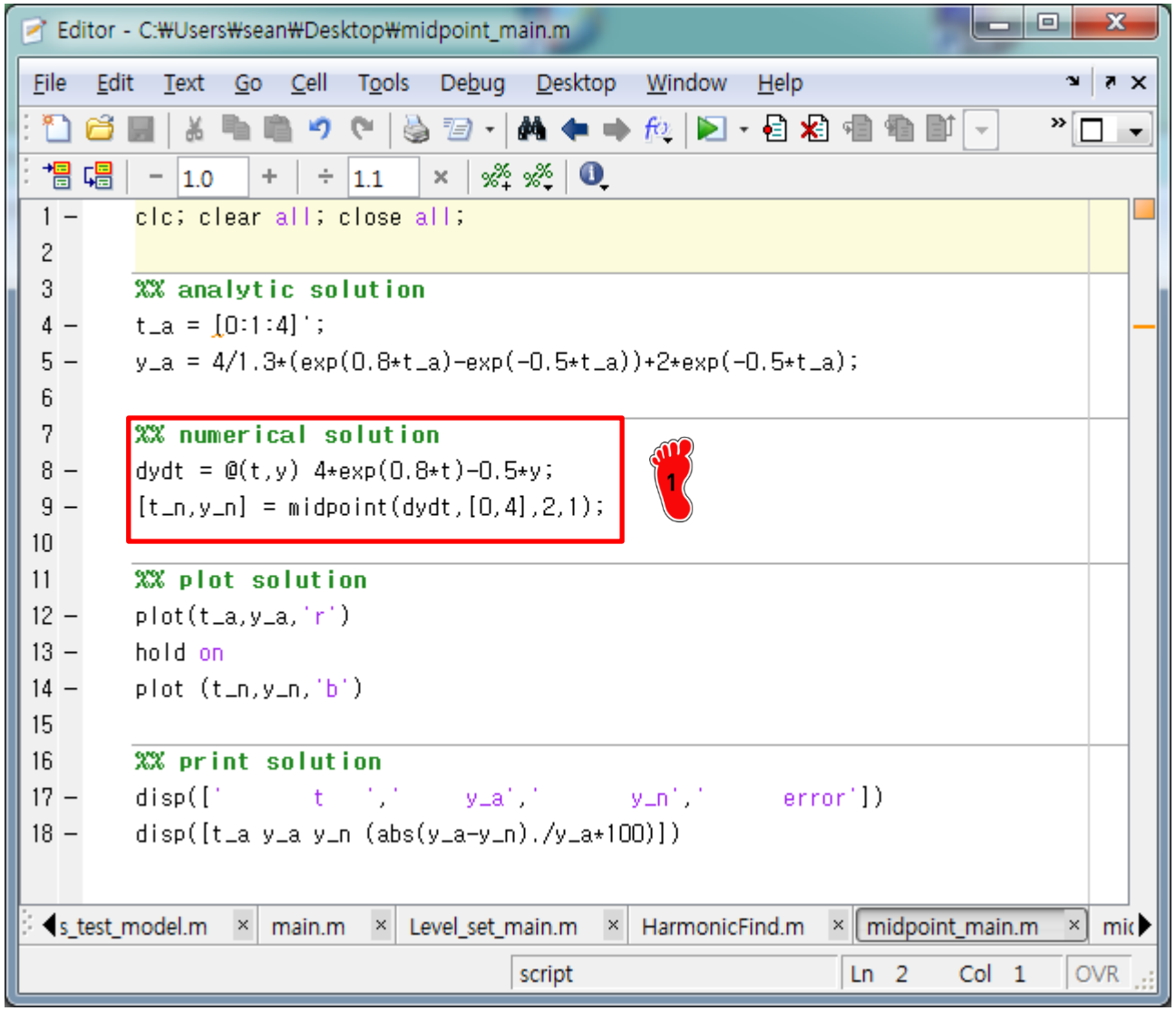
Predictor

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

Corrector

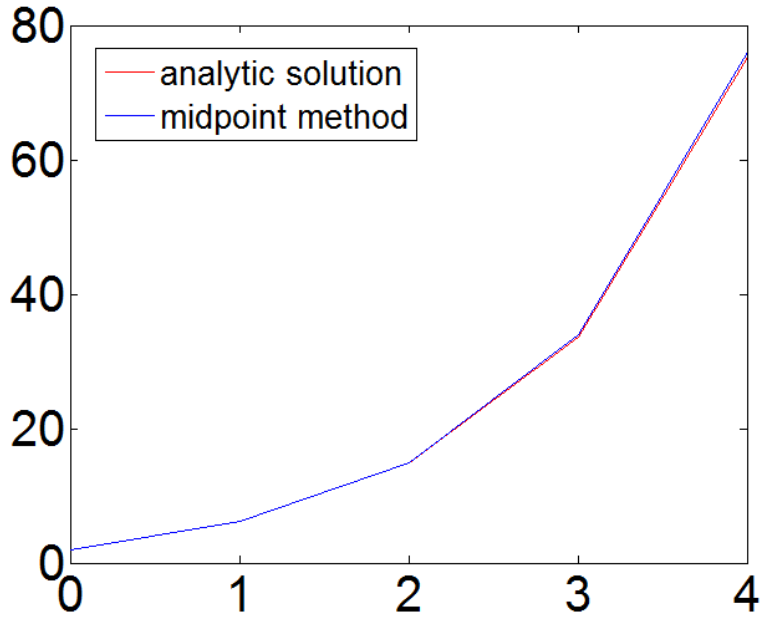
$$y_{i+1} = y_i + f\left(t_{i+1/2}, y_{i+1/2}\right)h$$

MIDPOINT METHOD: MAIN



1 midpoint method 함수를 호출

MIDPOINT METHOD: RESULTS



1 그래프

2 독립 변수 step 별로 종속 변수 및 에러 출력

Command Window

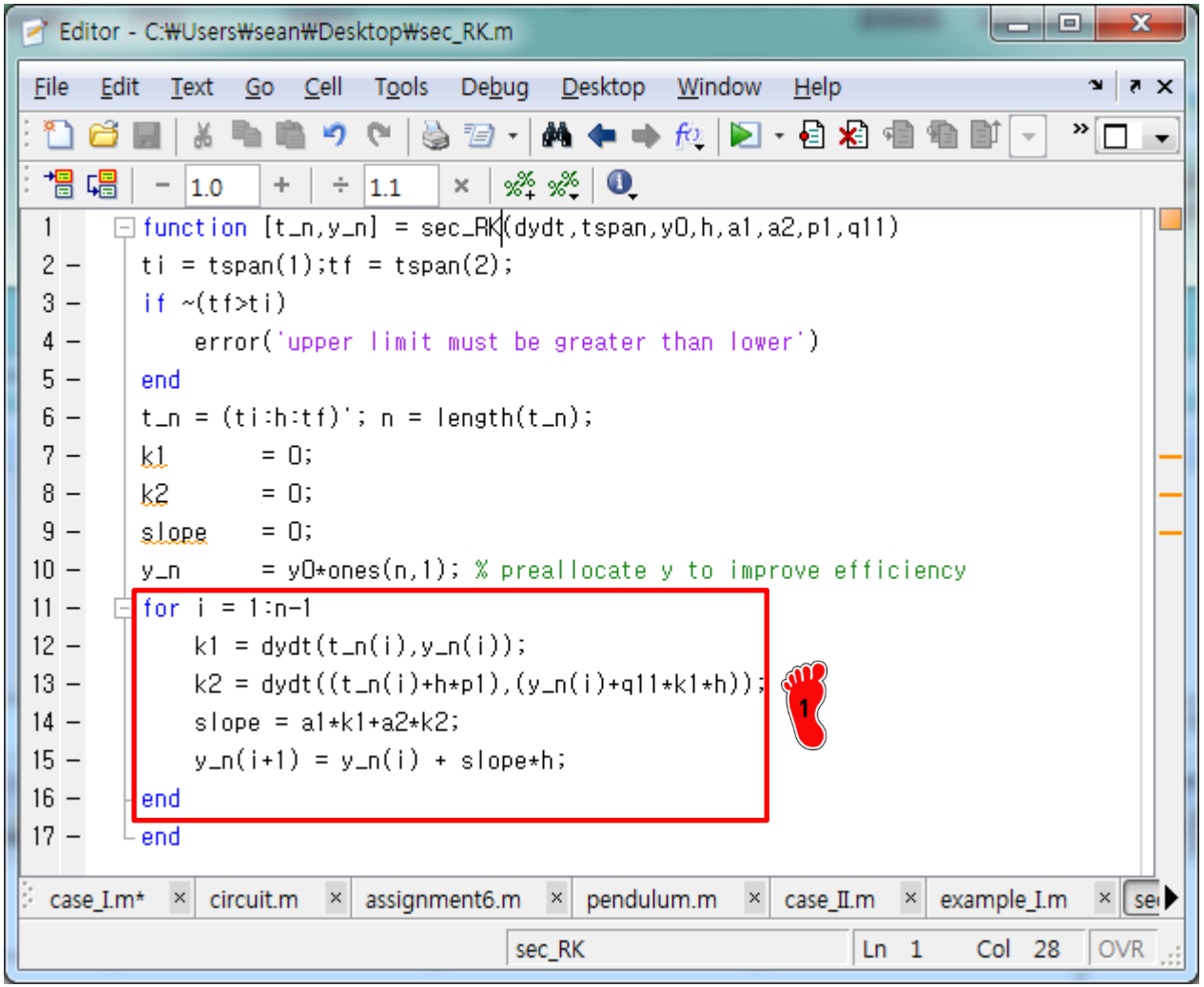
File Edit Debug Desktop Window Help

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



1 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

Editor - C:₩Users₩sean₩Desktop₩sec_RK_main.m

File Edit Text Go Cell Tools Debug Desktop Window Help

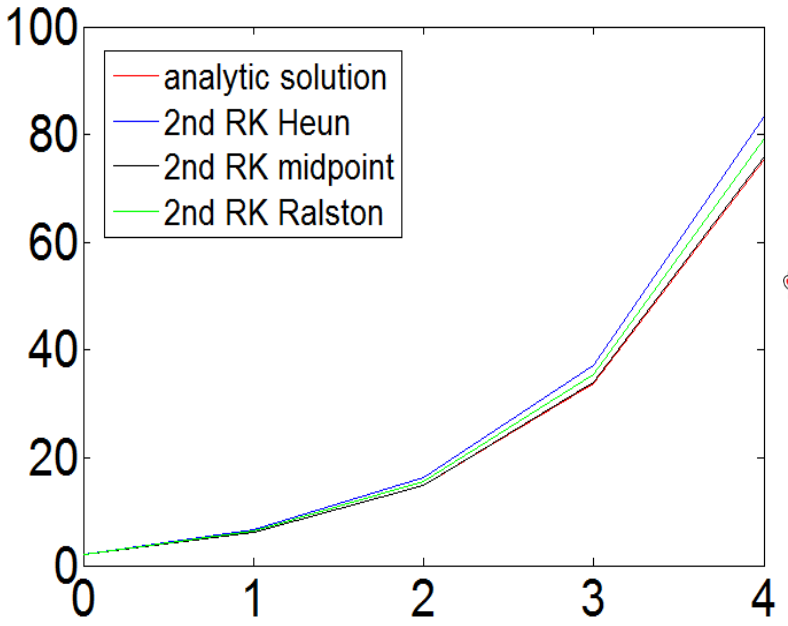
```
1 -  clc; clear all; close all;
2    %% analytic solution
3 -  t_a = [0:1:4]';
4 -  y_a = 4/1.3*(exp(0.8*t_a)-exp(-0.5*t_a))+2*exp(-0.5*t_a);
5
6    %% numerical solution (Second-order Runge-Kutta methods:
7    %                                Heun method without iteration)
8 -  dydt = @(t,y) 4*exp(0.8*t)-0.5*y;
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
21 -  plot(t_n,y_n_H,'b')
22 -  plot(t_n,y_n_M,'k')
23 -  plot(t_n,y_n_R,'g')
24
25   %% print solution
26 -  disp(['       t    ','     y_a','     y_n_H','     error',...
27        '     y_n_M','      error','     y_n_R','     error'])
28 -  disp([t_a y_a y_n_H (abs(y_a-y_n_H)./y_a*100) y_n_M,...
29        (abs(y_a-y_n_M)./y_a*100) y_n_R (abs(y_a-y_n_R)./y_a*100)])
```

case_I.m* | circuit.m | assignment6.m | pendulum.m | case_II.m | example_I.m

script Ln 2 Col 1 OVR

1 3가지 방법을 비교하기 위한 메인 코드

2ND RK : RESULTS



1 그래프

2 독립 변수 step 별로 종속 변수 및 에러 출력

Command Window

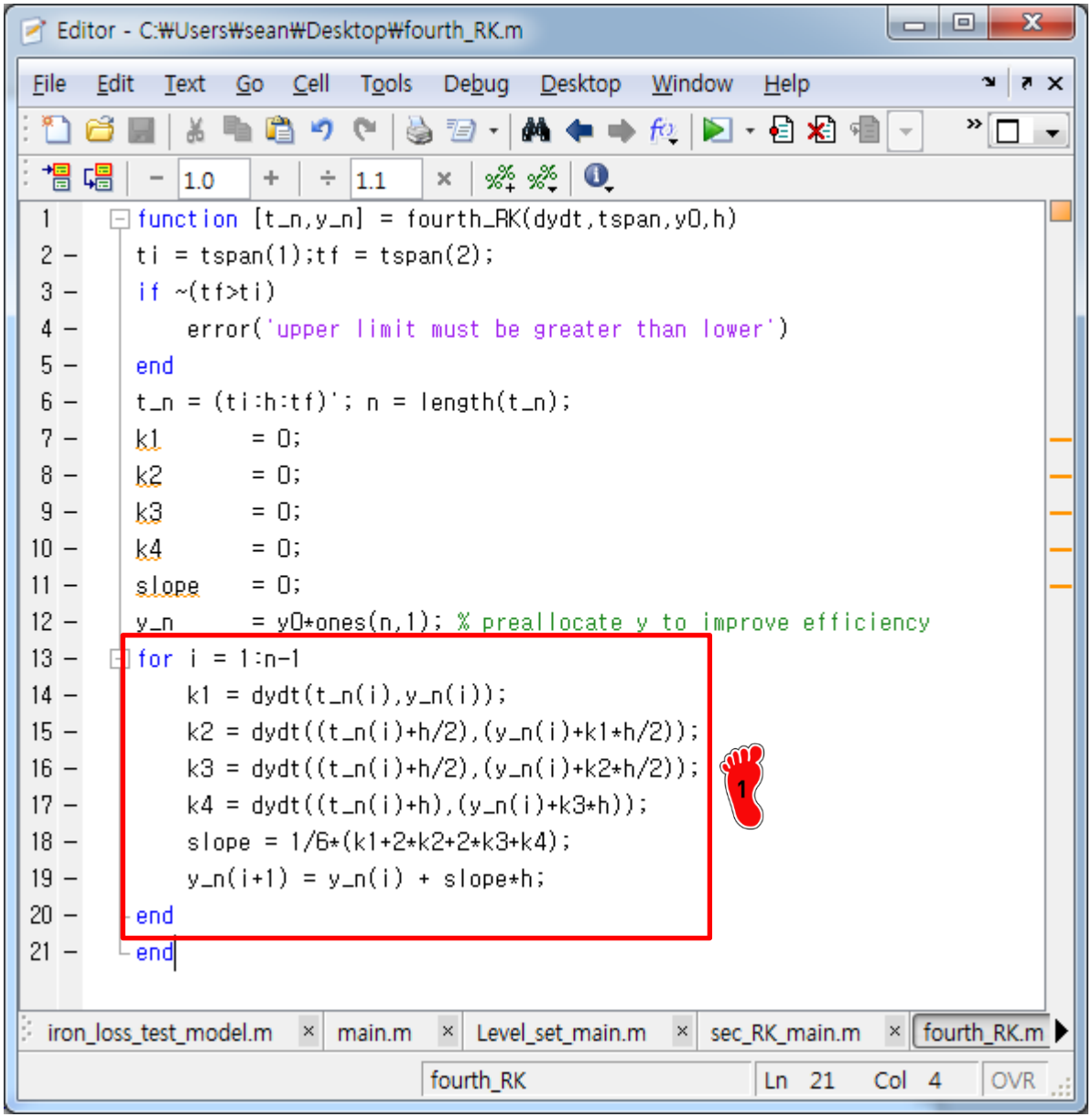
File Edit Debug Desktop Window Help

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

fx >>

OVR

4TH RK: FUNCTION



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

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

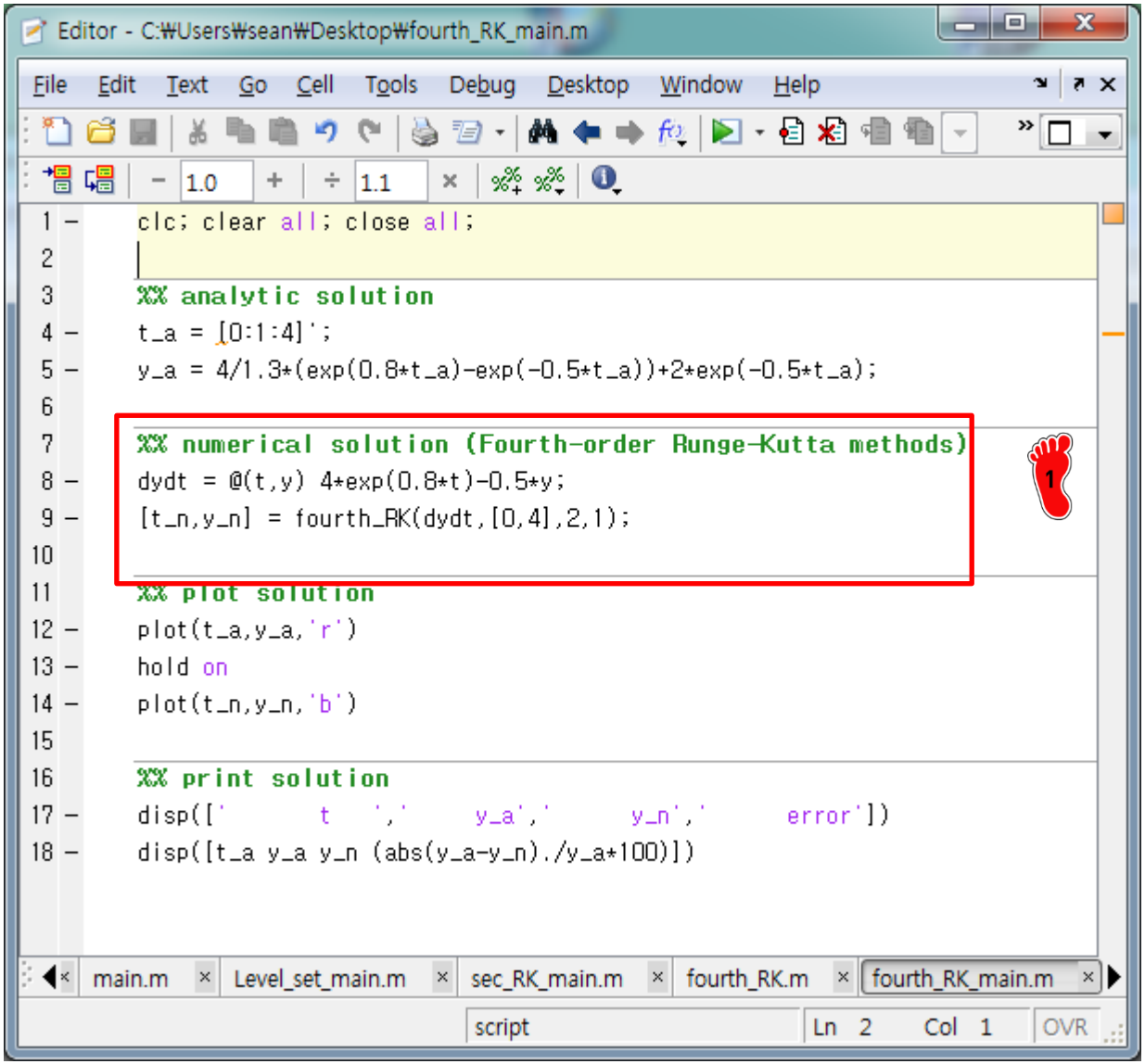
$$k_1 = f\left(t_i, y_i\right)$$

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

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

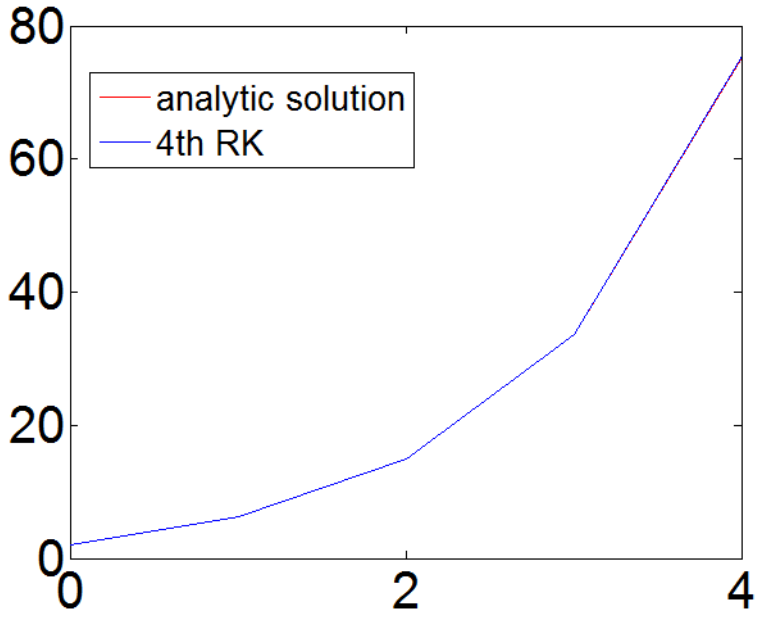
$$k_4 = f\left(t_i + h, y_i + k_3 h\right)$$

4TH RK: MAIN

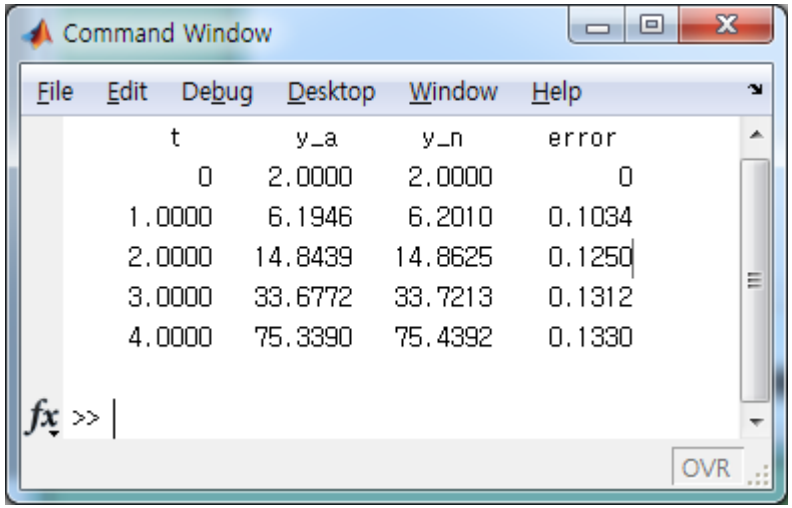


1 Fourth-order RK method 를 적용한 메인 코드

4TH RK : RESULTS



1



2

1 그래프

2 독립 변수 step 별로 종속 변수 및 에러 출력

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

ODE$_X$ BUILT IN FUNCTIONS

[t, y] = ode$_x$(odefun, tspan, y0)

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.

$$y_{i+1} = y_i + \frac{1}{9}\left(2k_1 + 3k_2 + 4k_3\right)h$$

$$k_1 = f\left(t_i, y_i\right), k_2 = f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right), k_2 = f\left(t_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h\right)$$

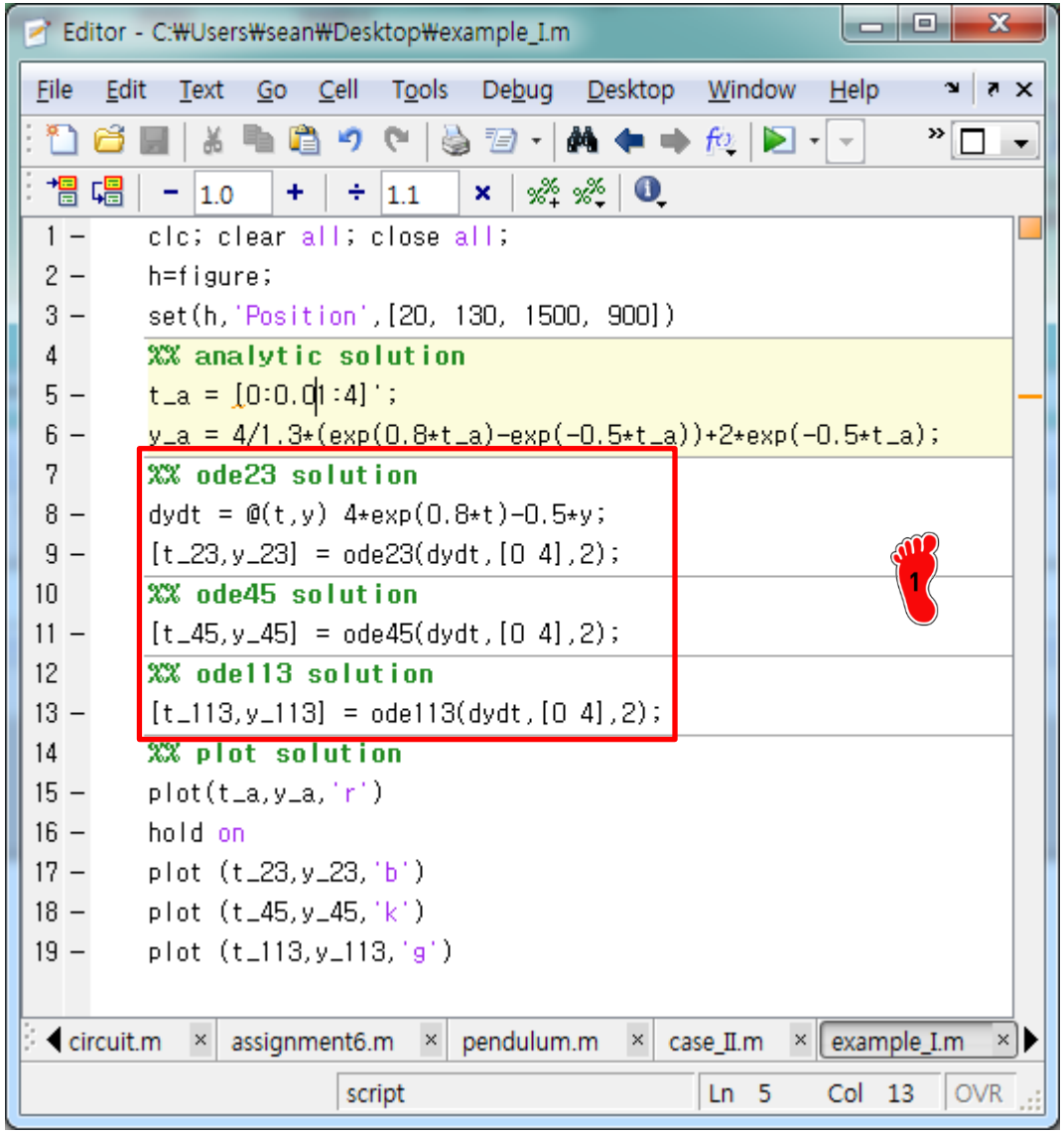
$$E_{i+1} = \frac{1}{72}\left(-5k_1 + 6k_2 + 8k_3 - 9k_4\right)h, k_4 = f\left(t_{i+1}, y_{i+1}\right)$$

$$E \le \max\left(\text{RelTol} \times \left|y\right|, \text{AbsTol}\right), \text{default}(\text{RelTol} = 10^{-3}, \text{AbsTol} = 10^{-6})$$

ode45: The ode45 function uses an algorithm devoloped 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-Bashforth-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 ode23, ode45, ode113 함수를 적용한 코드

2 예제

ordinary differential equation

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

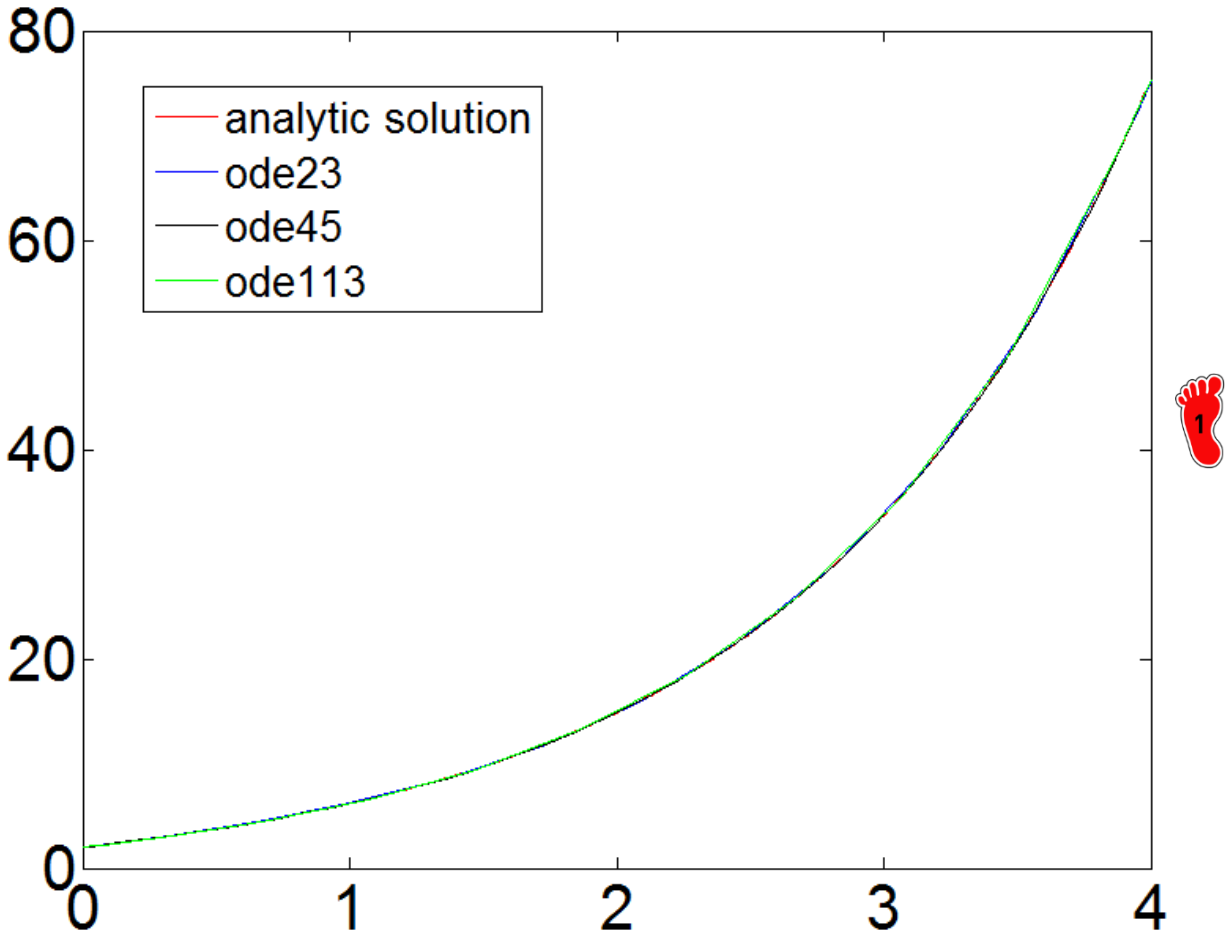
initial condition

$t = 0, y = 2$

analytic solution

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

EXAMPLE 25.5: RESULTS



ode23, ode45, ode113 함수를 적용한 결과

EXAMPLE 28.2

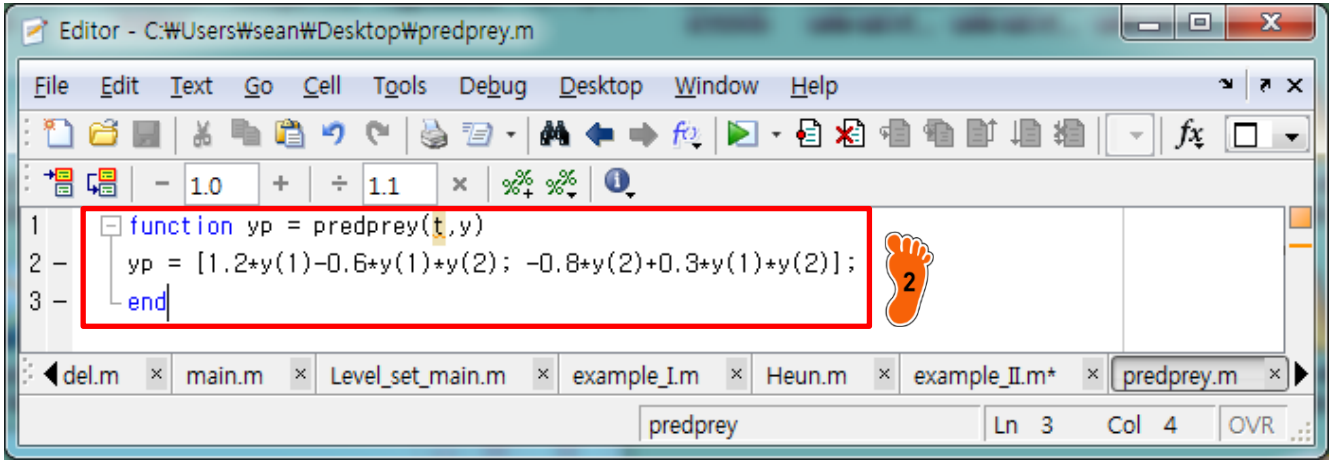
nonlinear ordinary differential equations

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

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

initial condition

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



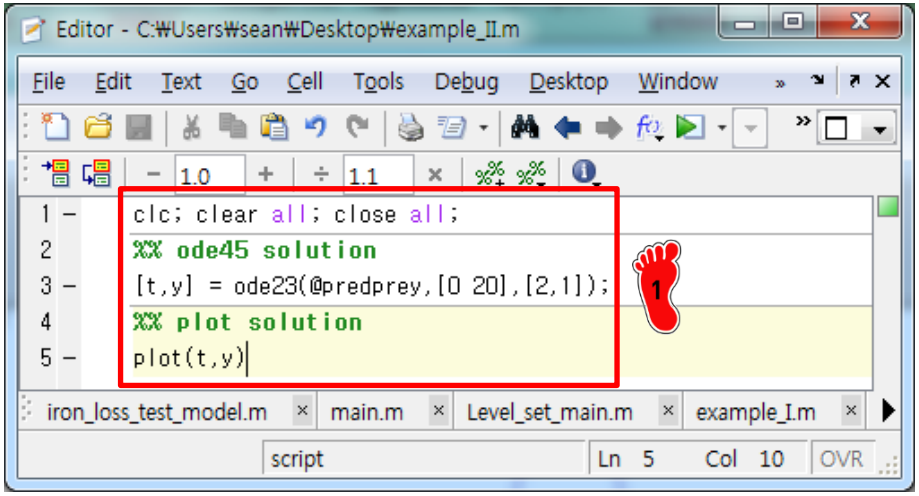
1 Predator-prey model developed by the Italian mathematician Vito Volterra and the American biologist Alfred J. Lotka.

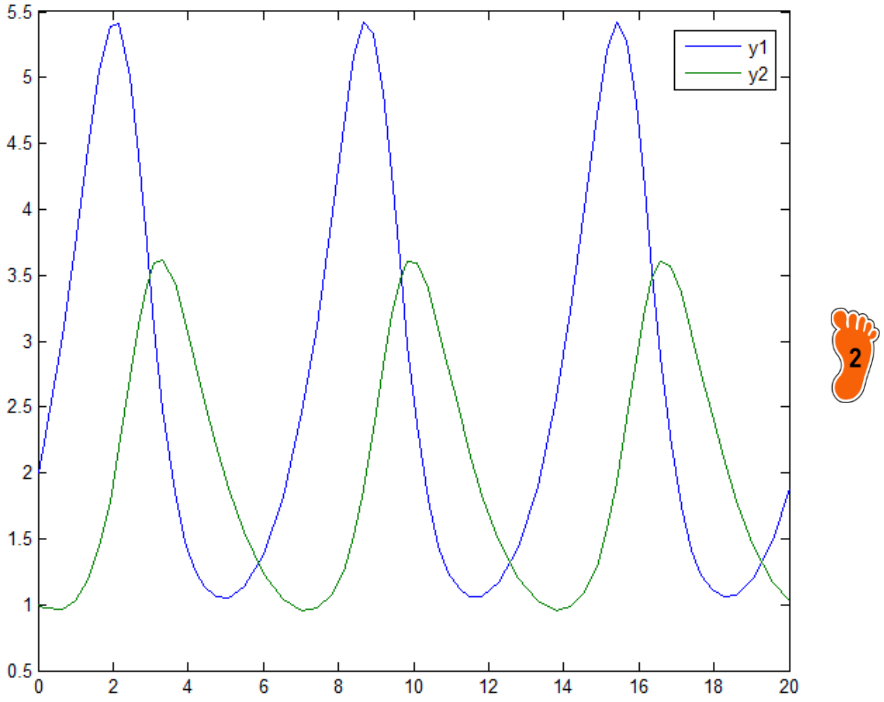
먹이사슬에 관한 미분방정식

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

2 매틀랩 함수

EXAMPLE 28.2: MAIN & RESULT





실행 코드

결과

y1 은 prey (먹이)
y2 는 predator (포식자)

먹이의 증가 및 감소가 포식자의 분포에 영향을 끼치는 결과

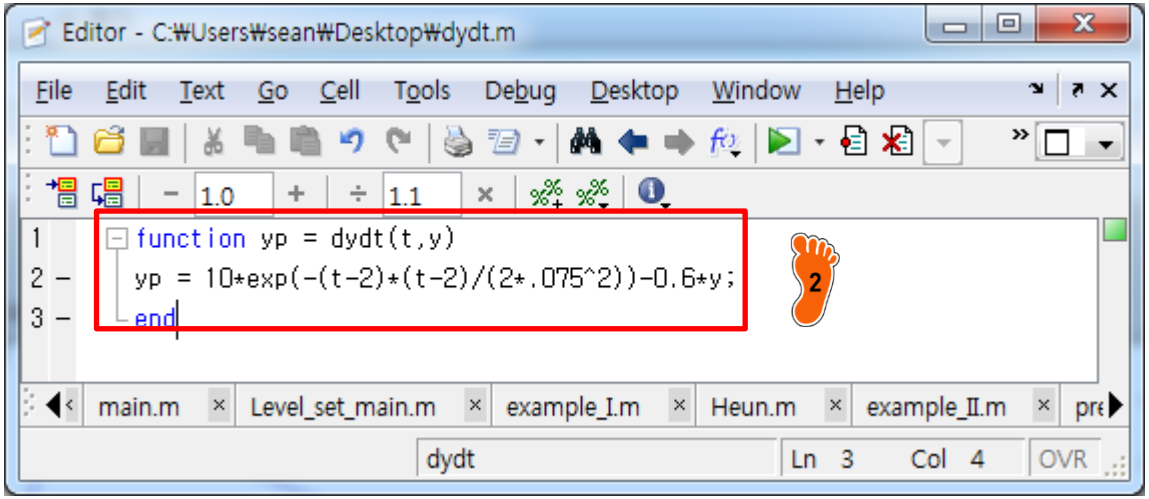
EXAMPLE 25.14

ordinary differential equation

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

initial condition

$t = 0, y = 0.5$



1 Adaptive ODE solver 를 보여주기 위한 미분방정식

2 매틀랩 함수

EXAMPLE 25.14: MAIN & RESULT

Editor - C:₩Users₩sean₩Desktop₩example_III.m

File Edit Text Go Cell Tools Debug Desktop

```
1 -  clc; clear all; close all;
2
3 -  %% ode45 solution without option control
4    [t1,y1] = ode23(@dydt,[0 4],0.5);
5 -  %% ode45 solution with option control
6 -  options = odeset('RelTol',1e-4);
7 -  [t2,y2] = ode23(@dydt,[0 4],0.5,options);
8    %% plot solution
9 -  subplot(2,1,1)
10 -  plot(t1,y1)
11 -  title('Without options')
12 -  subplot(2,1,2)
13 -  plot(t2,y2)
14 -  title('With options')
```

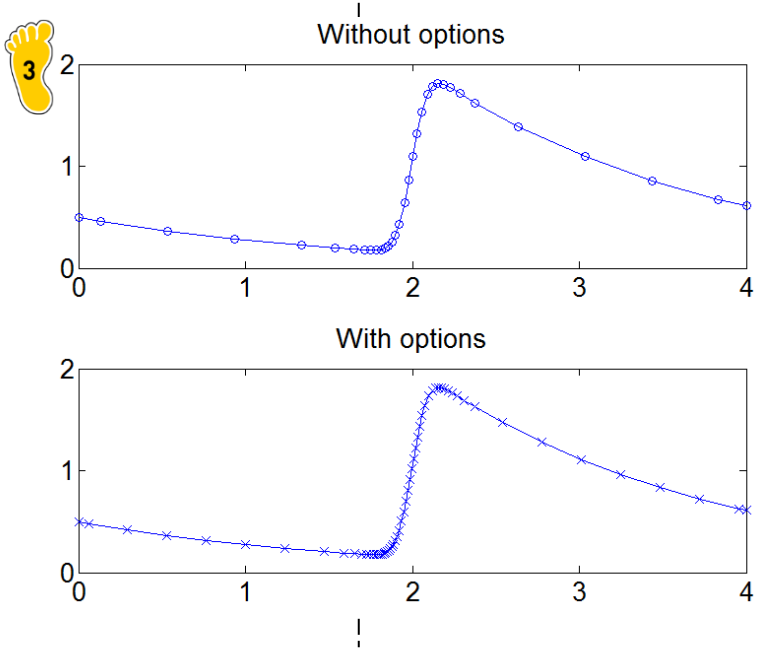
l.m | main.m | Level_set_main.m | example_I.m | Heun.m

script Ln 2 Col 1 OVR

1 Relative tolerance를 조절하지 않은 ode45 코드

2 Relative tolerance를 조절한 ode45 코드

3 기본적으로 step-size 가 조절이 됨
Relative tolerance 값을 조절하여 사용자가 step-size를 조절 가능



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

ODE$_{XS}$ BUILT IN FUNCTIONS

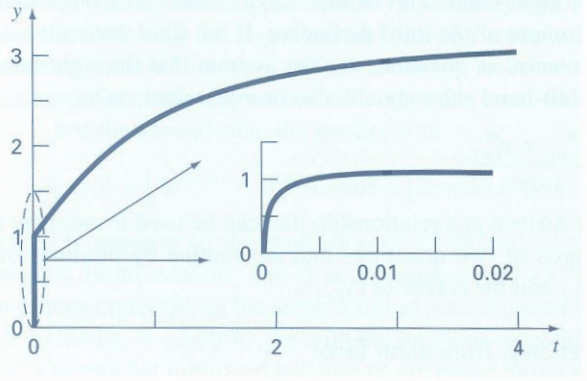


FIGURE 23.8
Plot of a stiff solution of a single ODE. Although the solution appears to start at 1, there is actually a fast transient from $y = 0$ to 1 that occurs in less than the 0.005 time unit. This transient is perceptible only when the response is viewed on the finer timescale in the inset.

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

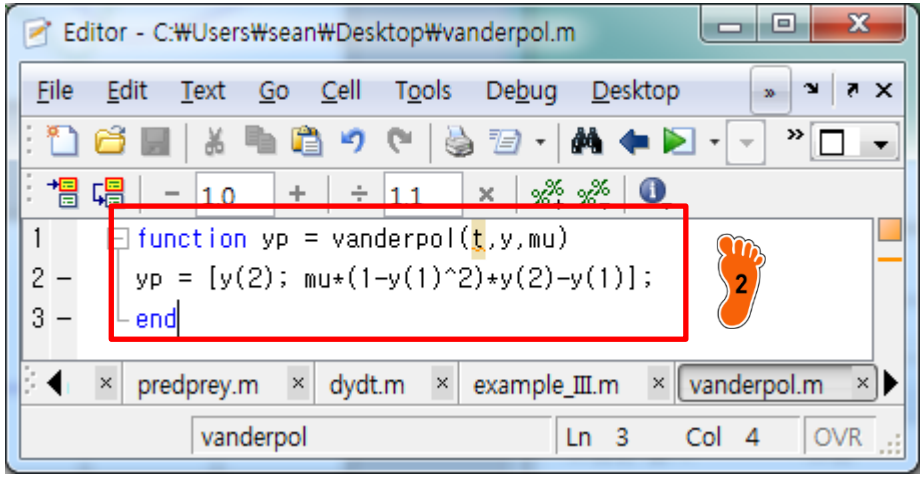
van der Pol equation

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

initial condition

$$t = 0, \frac{dy_1}{dt} = 1$$

$$\xrightarrow{\text{convert process}} \begin{cases} \dfrac{dy_1}{dt} = y_2 \\ \dfrac{dy_2}{dt} = \mu\left(1 - y_1^2\right)y_2 + y_1 \end{cases}$$



1 stiff 한 정도가 mu 값에 따라서 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.

2 매틀랩 함수

EXAMPLE 27.10: MAIN & RESULT

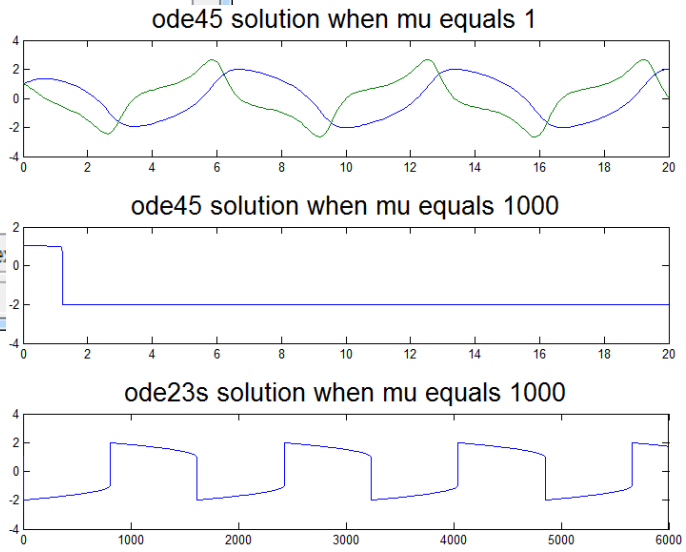
Editor - C:₩Users₩sean₩Desktop₩example.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```
1 -  clc; clear all; close all;
2
3 -  %% ode45 solution when mu equals 1
4 -  [t_45a,y_45a] = ode45(@vanderpol,[0 20],[1,1],[],1);
5 -  %% ode45 solution when mu equals 1000
6 -  [t_45b,y_45b] = ode45(@vanderpol,[0 20],[1,1],[],1000);
7 -  %% ode23s solution when mu equals 1000
8 -  [t_23s,y_23s] = ode23s(@vanderpol,[0 6000],[1,1],[],1000);
9 -  %% plot solution
10 - subplot(3,1,1)
11 - plot(t_45a,y_45a)
12 - title('ode45 solution when mu equals 1')
13 - subplot(3,1,2)
14 - plot(t_45b,y_45b(:,1))
15 - title('ode45 solution when mu equals 1000')
16 - subplot(3,1,3)
17 - plot(t_23s,y_23s(:,1))
18 - title('ode23s solution when mu equals 1000')
```

Heun.m | example_II.m | predprey.m | dydt.m

script Ln 1



1 mu = 1 일 때 실행 코드

2 mu = 1000 일 때 ode45 로 푸는 코드
결과는 그래프 2 번째 그림이며 -2로 수렴하는 결과값을 보임

3 mu = 1000 일 때 ode23s 로 푸는 코드
결과는 그래프 3번째 그림이며 주기적인 함수가 그려지는 대신 기울기가 급격히 변하는 구간을 가지고 있음

따라서 ode45로는 미분방정식을 풀 수 없음

SUMMARY

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try.
ode23	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
ode113	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems.
ode15s	Stiff	Low to medium	If ode45 is slow because the problem is stiff.
ode23s	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
ode23t	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

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

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

CASE STUDY I

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,

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

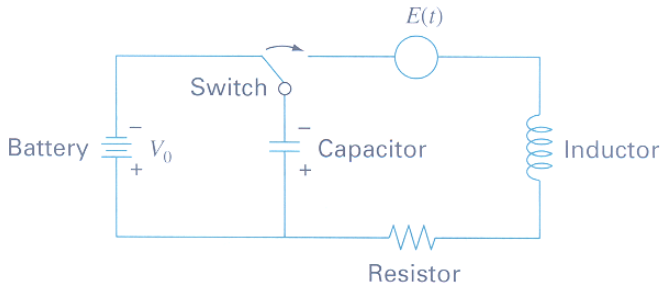
where $L(di/dt)$ = voltage drop across the inductor, L = inductance (H), R = resistance (Ω), q = charge on the capacitor (C), C = capacitance (F), $E(t)$ = time-variable voltage source (V), and

$$i = \frac{dq}{dt} \tag{28.10}$$

Equations (28.9) and (28.10) are a pair of first-order linear differential equations that can be solved analytically. For example, if $E(t) = E_0 \sin \omega t$ and $R = 0$,

$$q(t) = \frac{-E_0}{L(p^2 - \omega^2)}\frac{\omega}{p}\sin pt + \frac{E_0}{L(p^2 - \omega^2)}\sin \omega t \tag{28.11}$$

FIGURE 28.11
An electric circuit where the current varies with time.



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

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

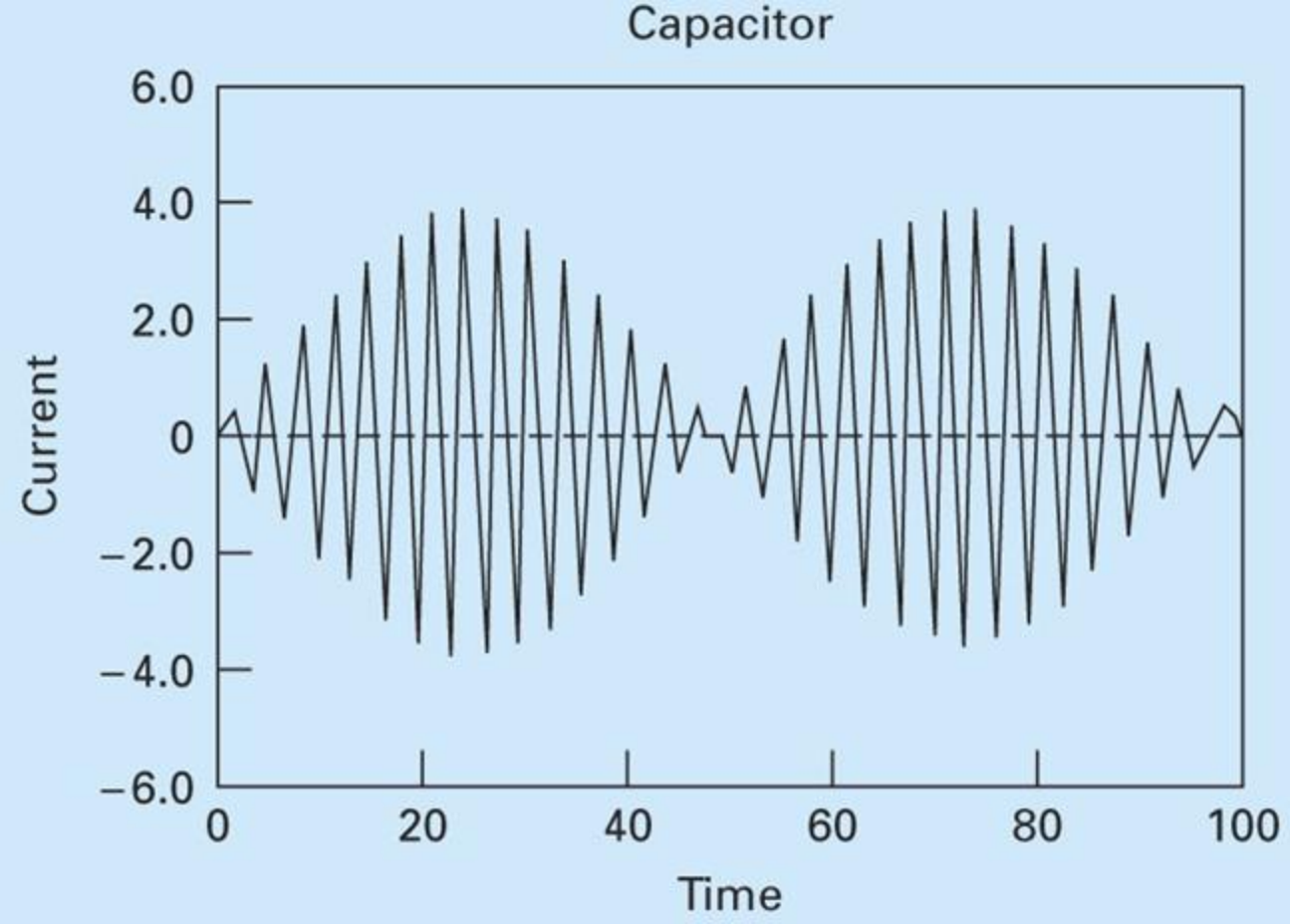
$$E_0 = 1\ \text{V}$$

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

$$\omega^2 = 3.5\ \text{s}^2$$

$$R = 0$$

CASE STUDY I: RESULT



CASE STUDY II

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.

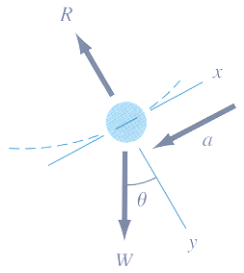
The free-body diagram in Fig. 28.16 shows the forces on the particle and the acceleration. It is convenient to apply Newton's laws of motion in the x direction tangent to the path of the particle:

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

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

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

FIGURE 28.16
A free-body diagram of the swinging pendulum showing the forces on the particle and the acceleration.



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

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

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.

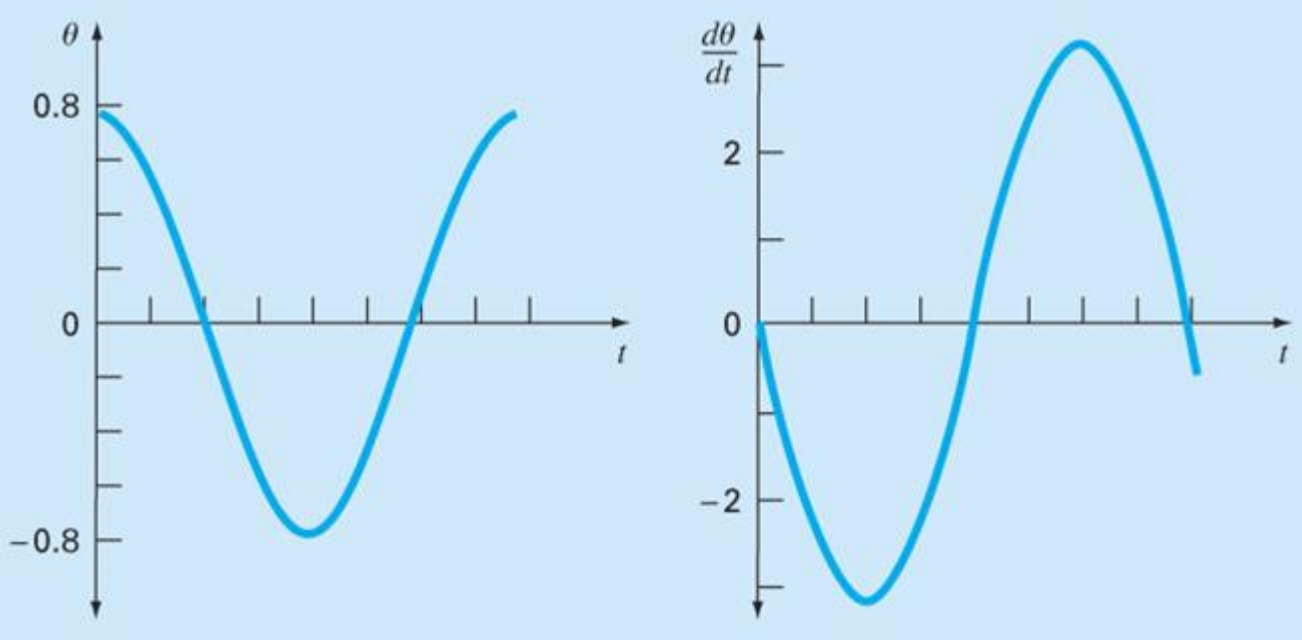
$$\sin\theta \approx \theta$$

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

$$\theta_0 = \pi / 4$$

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

CASE 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 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 $h(0) = 6$ m, $d = 0.25$ m, $e = 1$ m.

h, m	6	5	4	3	2	1	0
$A(h)$, 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

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

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

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

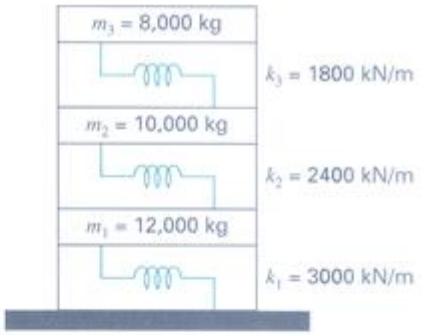


FIGURE P22.22

Simulate the dynamics of this structure from $t = 0$ to 20 s, given the initial condition that the velocity of the ground floor is $dx_1/dt = 1$ 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 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.

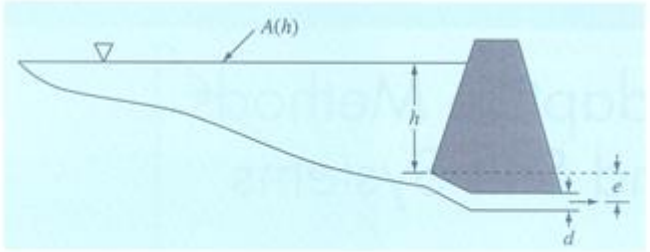


FIGURE P22.21

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

Figure P25.16