SEEE1022 INTRODUCTION TO SCIENTIFIC PROGRAMMING

CH13
Differential Equation

Dr. Mohd Saiful Azimi Mahmud (azimi@utm.my)
P19a-04-03-30, School of Electrical Engineering, UTM

OBJECTIVES

After studying this chapter you should be able to:

1. Understand and create function handle to available function.
2. Understand and create anonymous function from mathematical expression and existing function.
3. Understand and use the function functions.
4. Solve integration and difference using integral() and diff() functions.
5. Solve differential equation using ode23() function.

FUNCTION HANDLE

FUNCTION HANDLE

WHAT IS FUNCTION HANDLE?

- A function handle is a MATLAB variable that stores an association (a handle) to a function. With the handle, a function can be called indirectly.
- The data type of this variable is written as function_handle.
- To create a handle for a function, precede the function name with an @ sign. For example, below is how y is set as the function handle to function myfunction():

```
y = @myfunction;
```

- Usage of the function handle:
  1. To construct anonymous function.
  2. To pass a function to another function (known as function functions).
  3. To call local functions from outside the main function.

FUNCTION HANDLE

ANONYMOUS FUNCTION

- Recap: Generally, function is a program that can accept inputs and return outputs.
- Similar to the standard function, anonymous function can also accept inputs and return outputs.
- The differences are:
  1. Instead of a program file, anonymous function is a variable. The data type of anonymous function is function_handle. The @ operator creates the handle.
  2. Anonymous function can contain only a single executable statement.
  3. Since anonymous function is a variable, saving the function is similar to saving other type of variable. E.g., using the save() function.
  4. It is called anonymous function because the function does not have a name while standard function comes with a name. Anonymous function is called indirectly upon its function handle name.

FUNCTION HANDLE

CREATING ANONYMOUS FUNCTION

- **Syntax:**

Handle creator

```
f = @(x) executable statement
```

Function handle name

Input argument

- The executable statement can be either one of the followings:

1) Mathematical expression.
2) Named function.

- The output arguments are not define explicitly. The number of the output arguments is depending on the executable statement type.

1) Mathematical expression: Single output argument.
2) Named Function: Similar to the output arguments of the named function.

FUNCTION HANDLE

ANONYMOUS FUNCTION TO MATH EXPRESSION

EXAMPLE 1

Below is a standard function save as .m file. Since the function only consist of a single executable statement, the function can also be written as anonymous function.

```
function f = mypoly(x)
f = x^2 + 1;
```

Below is how the function is written as anonymous function.

```
>> mypolyFH = @(x) x^2 + 1
mypolyFH =

  function_handle with value:
    @(x)x^2+1

>> a = mypolyFH(2)
a =
     5
```

FUNCTION HANDLE

ANONYMOUS FUNCTION TO NAMED EXPRESSION

EXAMPLE 2

Writing available function as an anonymous function is a way to simplify the function. For example, `meshgrid` is a function that accept vectors as input and can return up to 3 output arguments. At certain situation, this function can be simplified as anonymous function to accept scalars rather than vectors.

```
>> mygrid = @(x)meshgrid(0:x,0:2);

>> [a,b] = mygrid(4)
a =
     0     1     2     3     4
     0     1     2     3     4
     0     1     2     3     4

b =
     0     0     0     0     0
     1     1     1     1     1
     2     2     2     2     2
```

FUNCTION HANDLE

ADDITIONAL PARAMETERS

- Additional parameters to the anonymous function are variables define in the executable statement but not declared as input to the function. For example, below is an anonymous function with one additional parameter z.

```
y = @(x) x.^2 + z
```

- The value of z must be define before the anonymous function is created.
- This additional parameters will be useful to add extra variables to the function input of the **function functions**.
- Next slide will discuss on the function functions.

FUNCTION HANDLE

FUNCTION FUNCTIONS

- A function that accept function handle as its input argument is called function functions.
- Since anonymous function is also a function handle, it can be used as the input to the function functions.
- Later in this chapter, function `integral()` and `ode23()` are both the example of function functions.
- Creating function input for function functions should follow the input argument requirement of the function functions.
- For example, function `integral()` specify the function input as below:

`fun` — **Integrand**
function handle

Integrand, specified as a function handle, which defines the function to be integrated from `xmin` to `xmax`.

For scalar-valued problems, the function `y = fun(x)` must accept a vector argument, x, and return a vector result, y. This generally means that `fun` must use array operators instead of matrix operators. For example, use `.*` (times) rather than `*` (mtimes). If you set the `'ArrayValued'` option to `true`, then `fun` must accept a scalar and return an array of fixed size.

FUNCTION HANDLE

FUNCTION FUNCTIONS

EXAMPLE 3

To plot $y(x) = 2x^2 + 3$ for $x = 0:0.1:10$, below is the MATLAB code when using function `plot()`:

```
>> x = 0:0.1:10;
>> y = 2*x.^2 + 3;
>> plot(x,y)
```

We can simplify the code as below using function `fplot()` where an anonymous function is used as the function handle to the `fplot()`:

```
>> fplot(@(x)2*x.^2+3,[0 10])
```

Alternatively, we can write a function file for the equation and use function handle to call the function as below.

```
function y = myEq(x)
y = 2*x.^2+3;
```

```
>> fplot(@myEq,[0 10])
```

FUNCTION HANDLE

ADDITIONAL PARAMETER EXAMPLE

EXAMPLE 4

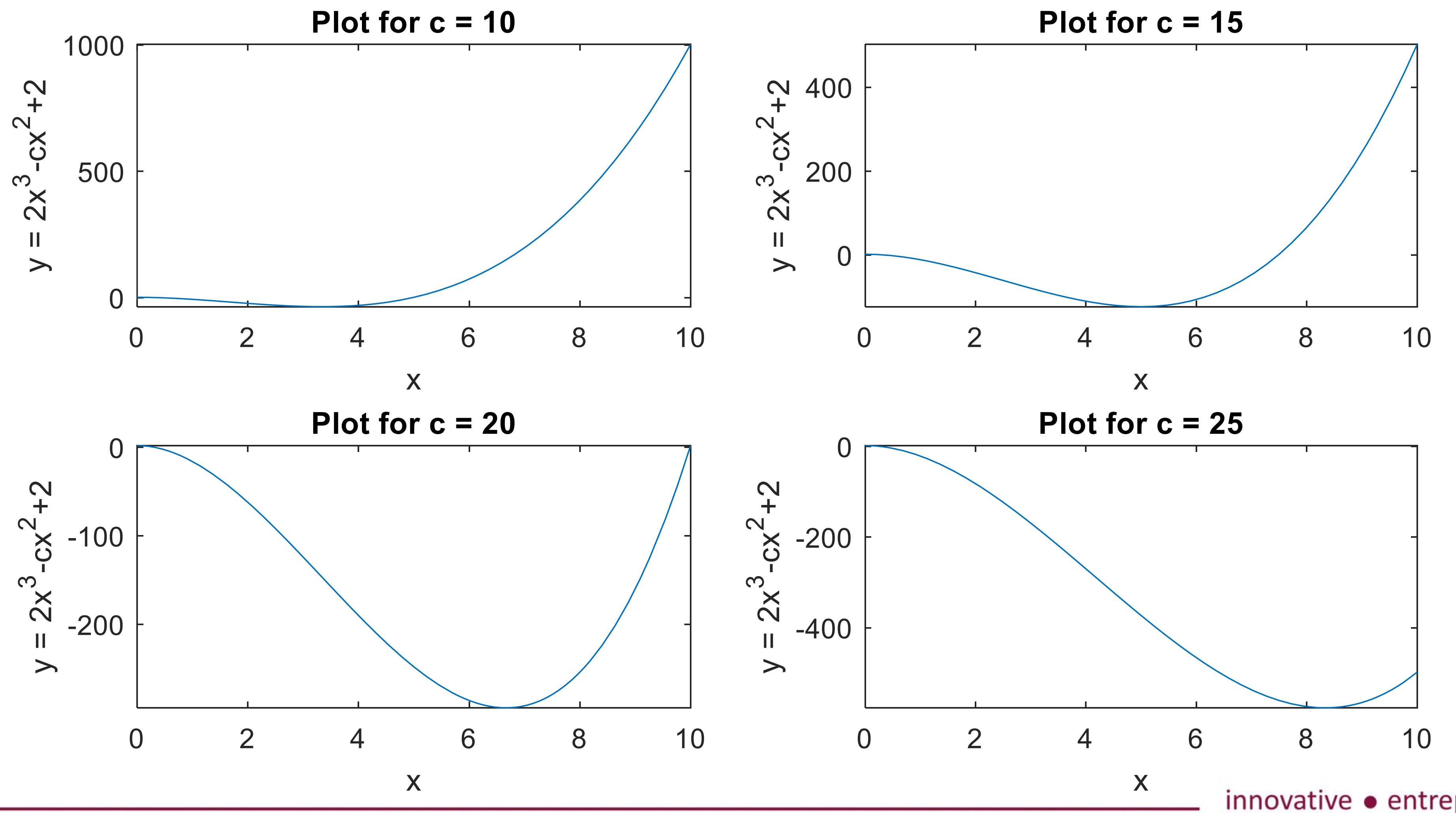
Below is MATLAB script showing *c* as the additional parameter to the anonymous function.

```
for c = 10:5:25
    subplot(2,2,c/5-1), fplot(@(x)2*x.^3-c*x.^2+2,[0 10])
    xlabel('x')
    ylabel('y = 2x^3 - cx^2 + 2')
    title(['Plot for c = ' num2str(c)])
end
```

FUNCTION HANDLE

ADDITIONAL PARAMETER EXAMPLE

EXAMPLE 4



INTEGRATION & DIFFERENTIATION

INTEGRAL

- **Syntax**

```
y = integral(fun,xmin,xmax)
```

Description

fun	:	Integrand, specified as a function handle, which defines the function to be integrated from xmin to xmax. For scalar-valued problems, the function y = fun(x) must accept a vector argument, x, and return a vector result, y. This generally means that fun must use array operators instead of matrix operators. For example, use .* (times) rather than * (mtimes). If you set the 'ArrayValued' option to true, then fun must accept a scalar and return an array of fixed size.
xmin	:	Lower limit of x.
xmax	:	Upper limit of x.
y	:	Integration result.

INTEGRATION & DIFFERENTIATION

INTEGRAL

EXAMPLE 5

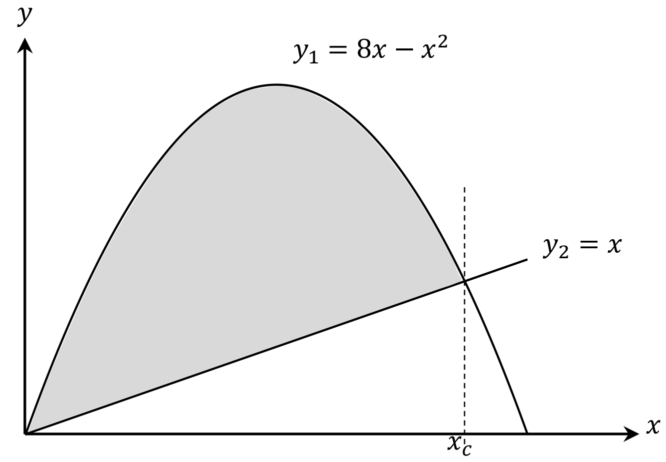
Solve:

1) $y_1 = \int_0^{10} 2x^2 + 6x + 1\,dx$

2) $y_2 = \int_{5.5}^{16.1} x^2 + 1\,dx$

```
>> y1 = integral(@(x)2*x.^2+6*x+1,0,10)

y1 =

  976.6667


>> y2 = integral(@(x)x.^2+1,5.5,16.1)

y2 =

   1.3462e+03
```

AREA UNDER GRAPH



EXAMPLE 6

Given 2 functions of y shown on the above figure, find the area of the shaded region.

Solution

1) Find x_c by finding roots of when $y_1 = y_2$. Or, it is the roots of $y_1 - y_2$.
2) Find area under both the quadratic function, y_1 and linear function, y_2 for x from 0 to x_c.
3) If area under graph y_1 is q_1 and area under graph y_2 is q_2, area of the shaded area is then $a = q_1 - q_2$

AREA UNDER GRAPH

EXAMPLE 6

Below is the MATLAB script of Example 6

```
xc = roots([-1 8 0]-[0 1 0]);
area = integral(@(x)-x.^2+8*x,xc(1),xc(2))...
        - integral(@(x)x,xc(1),xc(2));

fprintf('Shaded Area = %.2funit\xB2\n\n',area)
```

```
Shaded Area = 57.17unit²
```

Note that both the quadratic and linear functions are written as polynomial vectors when using the `roots()` function.

INTEGRATION & DIFFERENTIATION

DERIVATIVE FUNCTION: diff( ) / h

- **Syntax**

```
Y = diff(X,n)/h^n     %Approximate Derivatives
```

Description

X	:	Input array.
n	:	Derivative order.
h	:	Interval between data points.
Y	:	Difference result.

DERIVATIVE

EXAMPLE 7

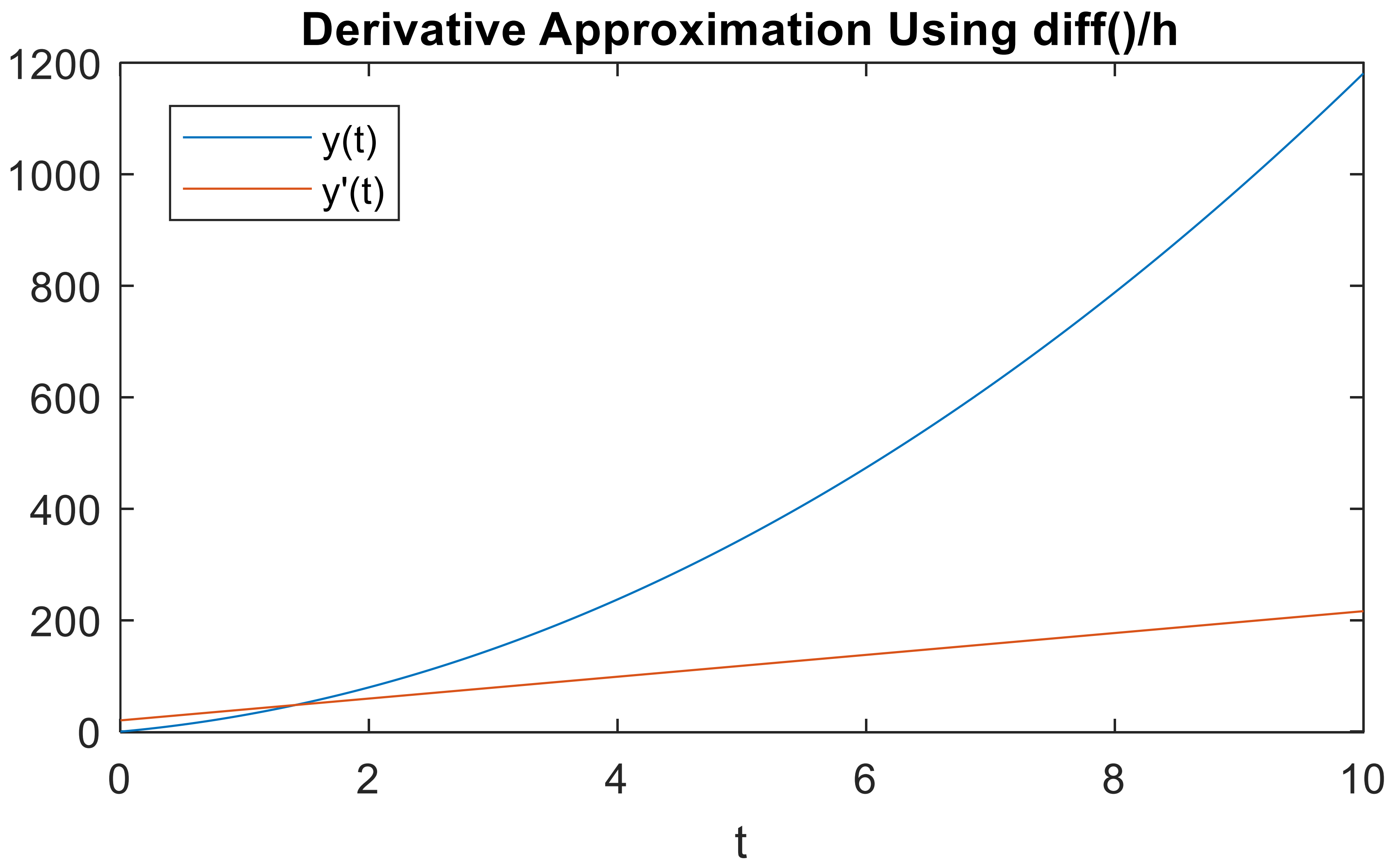
Find derivative of $y(t) = 9.8t^2 + 20.13t - 0.03$ for $t = 0\ to\ 10$.

```
h = 0.001;
t = 0:h:10;
y = 9.8*t.^2 + 20.13*t - 0.03;
dydt = diff(y)/h;

plot(t(2:end),[y(2:end); dydt])
xlabel('t')
title('Derivative Approximation Using diff()/h')
legend('y(t)','y''(t)')
```

```
>> size(y)
ans =
           1       10001

>> size(dydt)
ans =
           1       10000
```

Note that the size of `dydt` is always shorter by 1 sample compared to `y`.

DERIVATIVE


Derivative Approximation Using diff()/h
y(t)
y'(t)
1200
1000
800
600
400
200
0
0
2
4
6
8
10
t

DERIVATIVE APPROXIMATION ERROR

EXAMPLE 8

- Differentiate the following function for $x = 0\ to\ 1$ using the `diff()/h` function.

$$f(x) = 0.3 + 20x - 180x^2 + 650x^3 - 880x^4 + 360x^5$$

- Then, compare the results with the exact solution given by:

$$f'(x) = 20 - 360x + 1950x^2 - 3520x^3 + 1800x^4$$

- To do the above, write a script that estimates the differentiation of $f(x)$ by setting h equals to 1, 0.5 and 0.1, and compares the results with the exact solution graphically.

DERIVATIVE APPROXIMATION ERROR

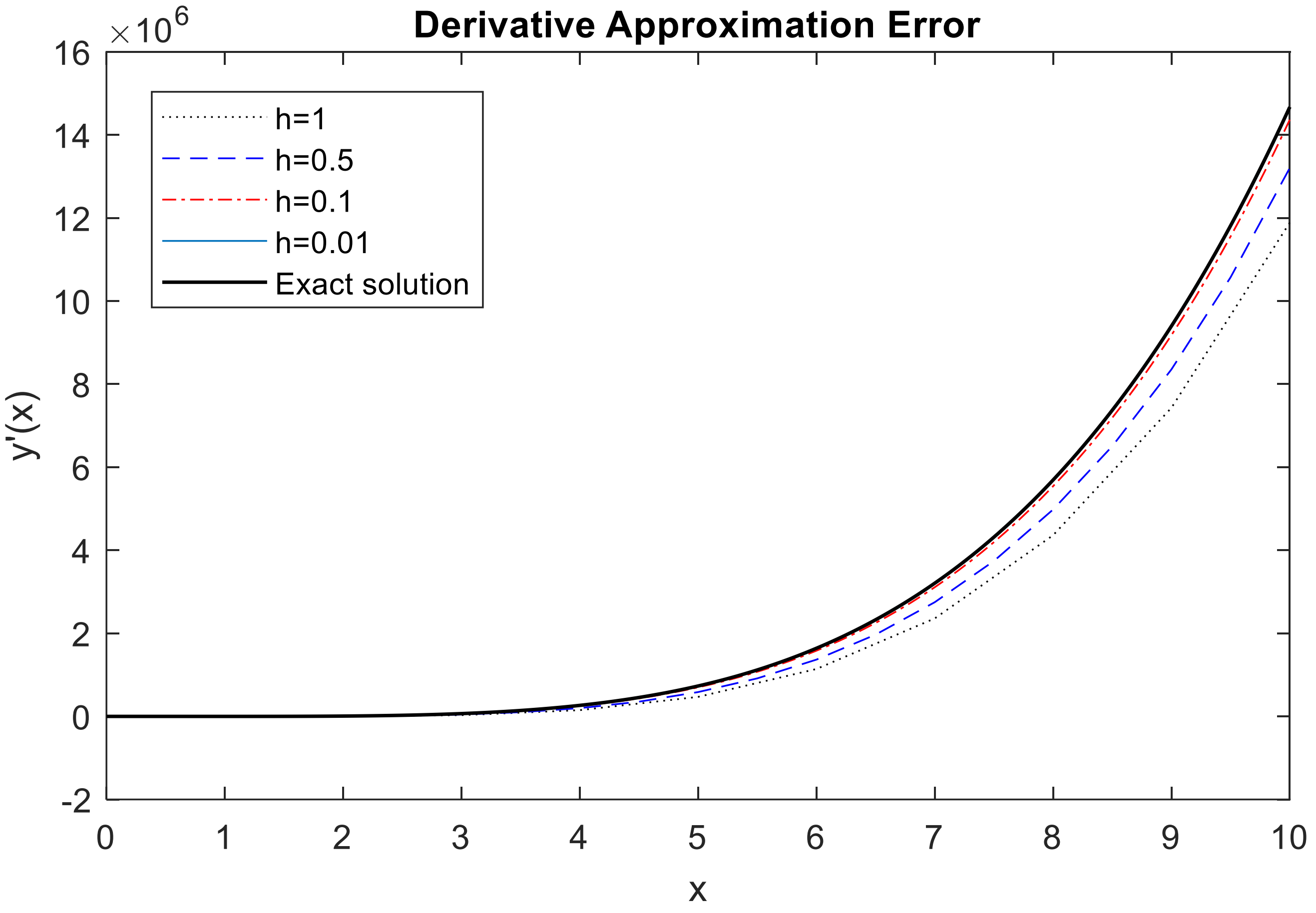
EXAMPLE 8

- Below is the MATLAB script for Example 8

```
h = [1 0.5 0.1 0.01];
linestyle = {':k','--b','-.r','-'};
for n = 1:4
    x = 0:h(n):10;
    y = 0.3 + 20*x - 180*x.^2 + 650*x.^3 - 880*x.^4 + 360*x.^5;
    ydot = diff(y)/h(n);
    plot(x(2:end),ydot,linestyle{n})
    hold on
end

x = 0:0.001:10;
ydotexact = 20 - 360*x + 1950*x.^2 - 3520*x.^3 + 1800*x.^4;

plot(x(2:end),ydotexact(2:end),'k','LineWidth',1)
xlabel('x')
ylabel('y''(x)')
title('Derivative Approximation Error')
legend('h=1','h=0.5','h=0.1','Exact solution')
hold off
```

DERIVATIVE APPROXIMATION ERROR

EXAMPLE 8



* The error decreases when h value becomes smaller. In this example, $h = 0.01$ gives unnoticeable error.

2ND ORDER DERIVATIVE

EXAMPLE 9

- Solve and plot $\frac{d^2y}{dt^2}$ for the following function. Set time span t from $0s$ to $50s$.

$$y(t) = -4.0622e - 05t^4 + 0.0036t^3 - 0.0229t^2 + 1.4151t$$

- Then find $y''(5)$.

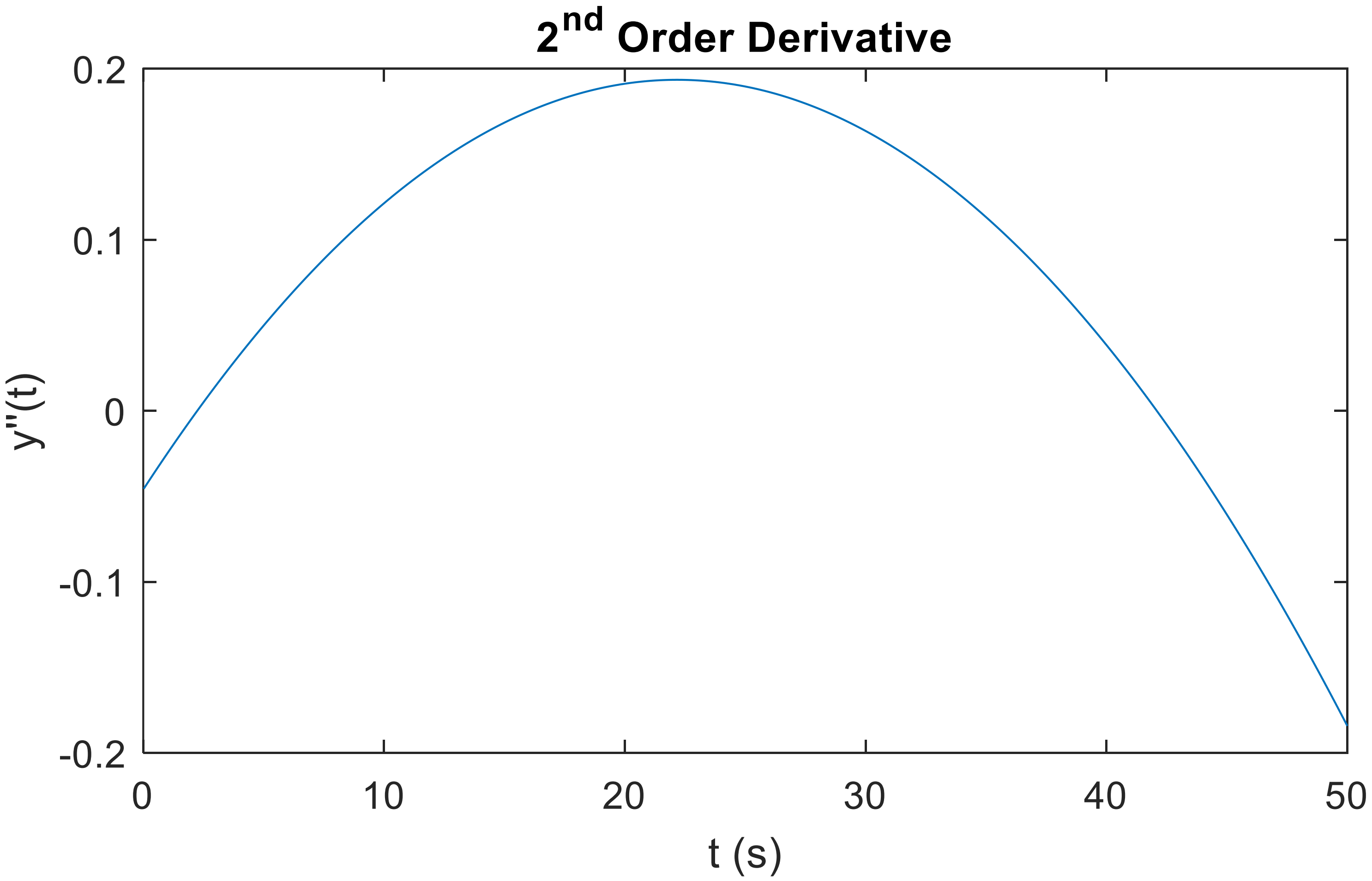
```
h=0.01;
t = 0:h:50;
y = -4.0622e-05*t.^4 + 0.0036*t.^3 - 0.0229*t.^2 + 1.4151*t;
ydot = diff(y,2)/h^2;

plot(t(3:end),ydot)
xlabel('x')
ylabel('y''''(x)')
title('2^{nd} Order Derivative')

fprintf('y''''(5) = %.4f\n',ydot(5/h+1))
```

```
y''(5) = 0.0502
```

2nd order results is always shorter by 2 samples compared to the original vector `y`.

2ND ORDER DERIVATIVE

EXAMPLE 9



DIFFERENTIAL EQUATION

INTRODUCTION

- Differential equation is an equation involving derivatives of a function.
- In MATLAB, differential can be solve using function `ode23()`.
- **Syntax**

```
[t,y] = ode23(odefun,tspan,y0)
```

Description

odefun	:	Function to be solve, specified as function handle.
tspan	:	Integral interval.
y0	:	Initial condition.
y	:	Solution.
t	:	Evaluation points

`odefun` FUNCTION HANDLE

Functions to solve, specified as a function handle which defines the functions to be integrated.

The function `dydt = odefun(t,y)`, for a scalar `t` and a column vector `y`, must return a column vector `dydt` of data type `single` or `double` that corresponds to $f(t, y)$. `odefun` must accept both input arguments, `t` and `y`, even if one of the arguments is not used in the function.

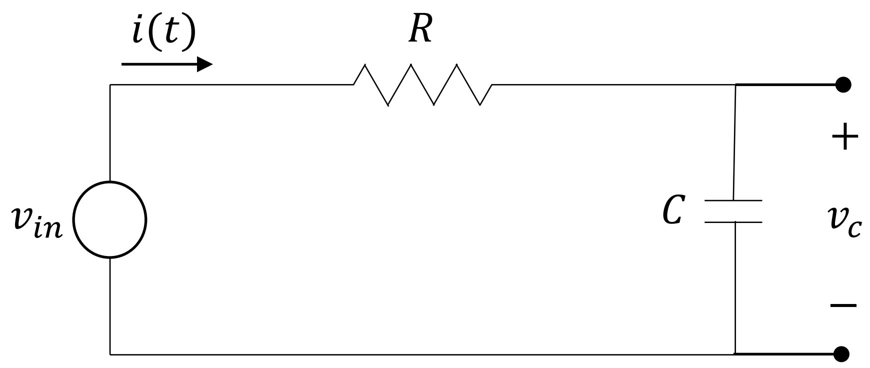
For example, to solve $y' = 5y - 3$, use the function:

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

- Above is the description from the MATLAB documentation of the function handle for function `ode23()`.
- The example given in the documentation can also be written as an anonymous function as below:

```
@(t,y)5*y-3
```

CHARGING OF AN RC CIRCUIT



EXAMPLE 10

To find how the capacitor is charging, below is the differential equation of above circuit derived from KCL

$$v_{in} = v_R + v_c = RCv_c' + v_c$$

By setting $R = 10k\Omega$, $C = 10\mu F$, $v_{in} = 10V$ and $v_c = 0V$ at $t = 0$, use function `ode23()` to plot the v_c for t from $0s$ to $2s$.

CHARGING OF AN RC CIRCUIT

EXAMPLE 10

To solve the differential equation for v_c using function `ode23()`:

1) Rearrange the equation by setting the v_c' placed at the left side of the equation as below and write the appropriate `odefun` function handle.

$$v_c' = \frac{v_{in} - v_c}{RC} = \frac{10 - v_c}{0.1} = 100 - 10v_c$$

```
odefun = @(t,vc)100-10*vc
```

2) Set initial value for v_c. In this example, it is set to 0.

3) Set time interval. In this example, it is set as `[0 2]`.

4) Write the `ode23()` function and run the code.

CHARGING OF AN RC CIRCUIT

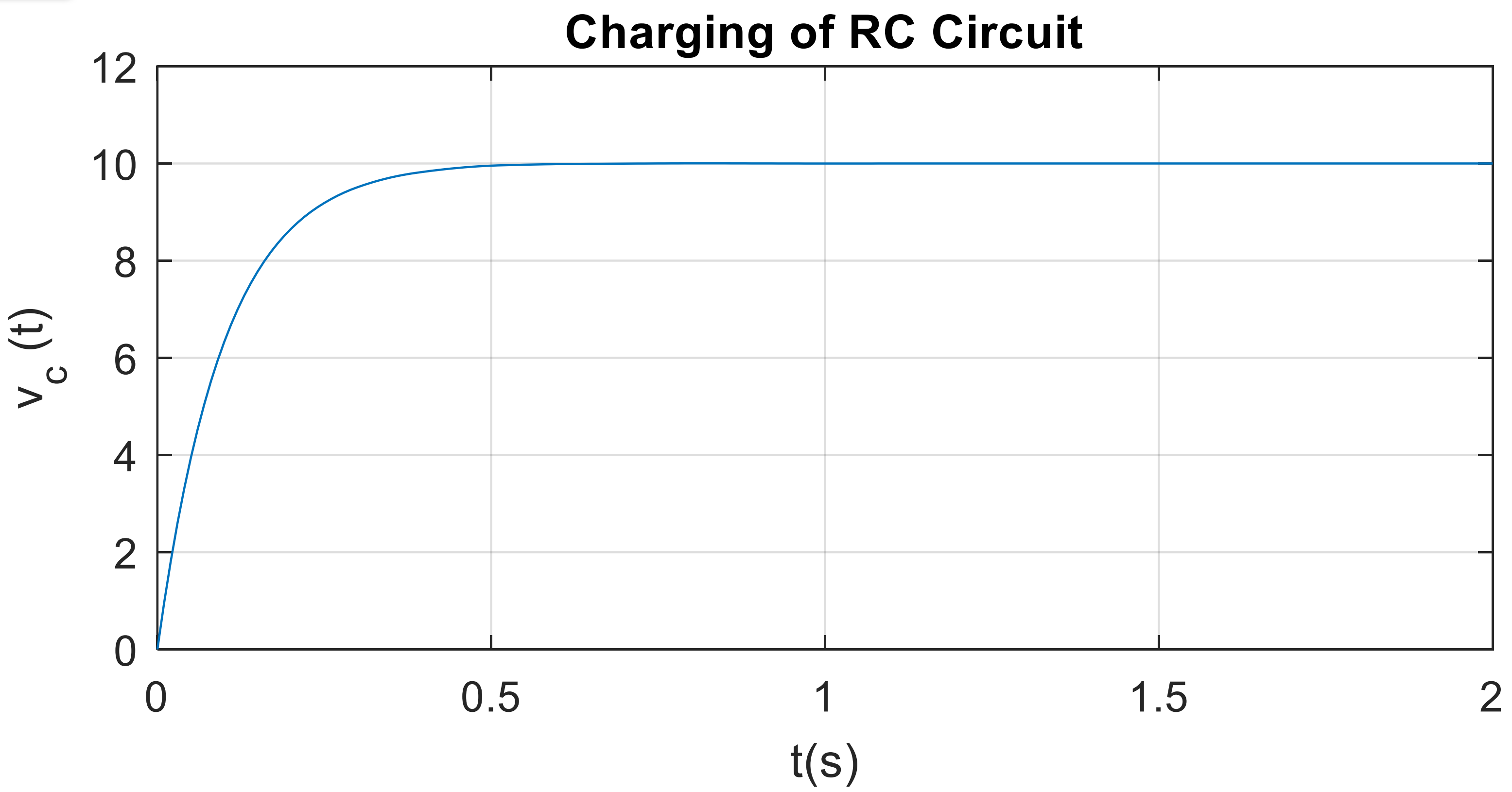
EXAMPLE 10

Below is the MATLAB script for Example 10

```
[t,vc] = ode23(@(t,vc)100-10*vc, [0 2], 0);

plot(t,vc)
xlabel('t(s)'), ylabel('v_c (t)')
title('Charging of RC Circuit')
grid on
```

CHARGING OF AN RC CIRCUIT

EXAMPLE 10

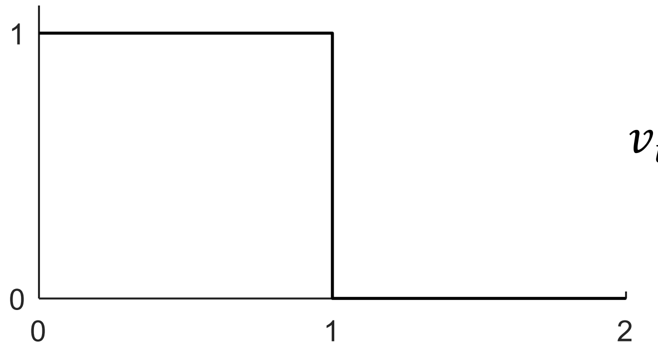


It can be seen from the above figure, the steady state voltage of the capacitor is $10V$, which is similar to the v_{in}.

CHARGING AND DISCHARGING OF RC CIRCUIT

EXAMPLE 11

Repeat Example 10 with the following v_{in}.



$$v_{in} = \begin{cases} 1 & for\ \ t < 1 \\ 0 & elsewhere \end{cases}$$

Solution

Above v_{in} can be coded as `1*(t<1)` or simply as `(t<1)`. The only modification needed for function `odefun()` is to replace the v_{in} with the new equation where the derivative equation is now `ddt=((t<1)-vc)/0.1`.

CHARGING AND DISCHARGING OF RC CIRCUIT

EXAMPLE 11

Solution

To solve the differential equation for v_c using function `ode23()`:

1) From Example 10, we have

$$v_c' = 10(v_{in} - v_c) \text{ or } (v_{in} - v_c) / 0.1$$

In this example, v_{in} is no longer a constant value where its value turns to 0 when $t \geq 1$. To solve this, we can code v_{in} either using decision statement or logical vector. Since anonymous function can only have one executable statement, logical vector method is used in this example. Here v_{in} can be coded as `1*(t<1)` or simply as `(t<1)` and the `odefun` function handle is written as below:

```
odefun = @(t,vc)10*((t<1)-vc)
```

2) Similar to Example 10, initial value for v_c is 0 and time interval is `[0 2]`.

3) Write the `ode23()` function and run the code.

CHARGING AND DISCHARGING OF RC CIRCUIT

EXAMPLE 11

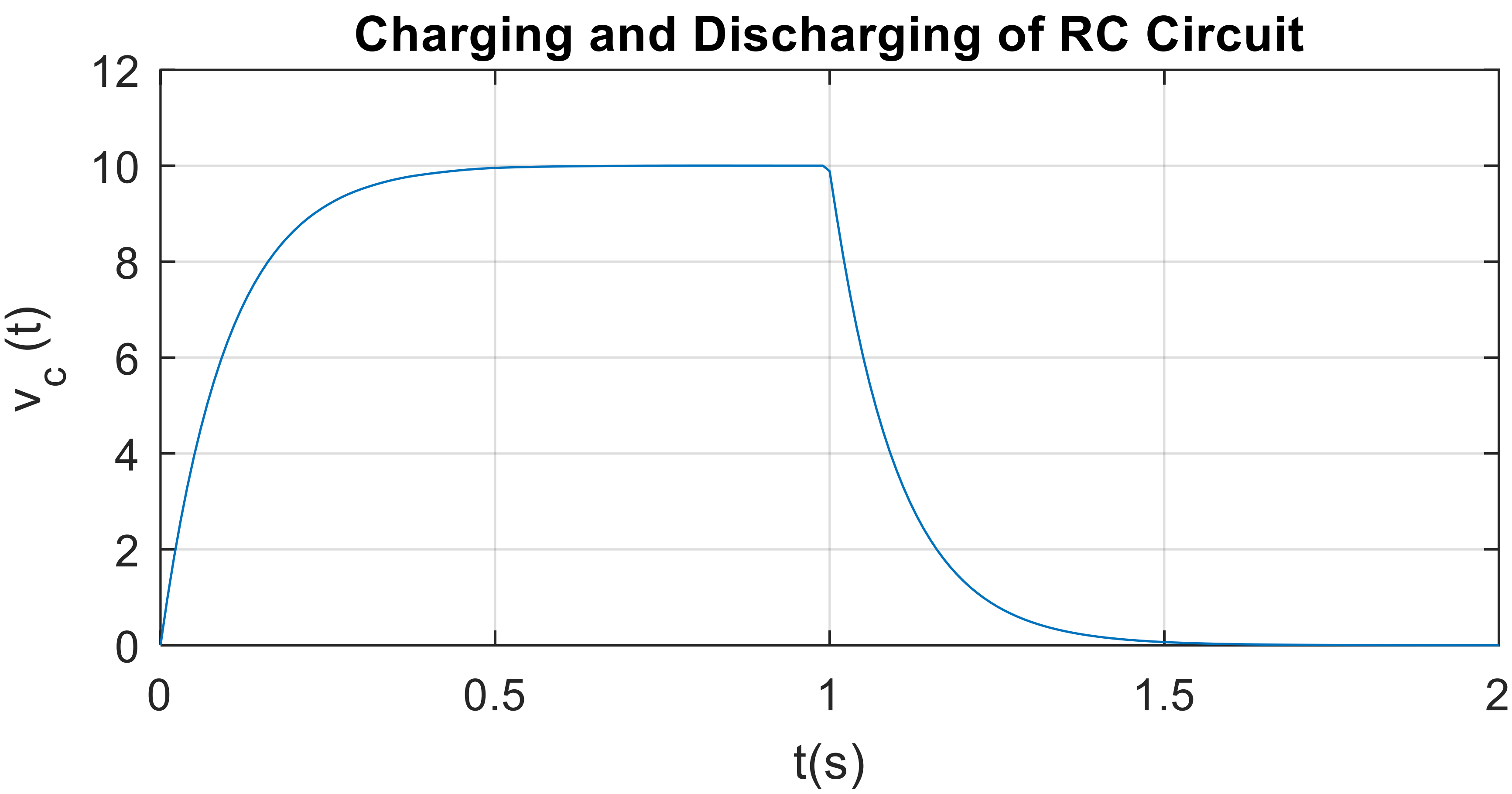
Below is the MATLAB script for Example 11

```
[t,vc] = ode23(@(t,vc)10*((t<1)-vc), [0 2], 0);

plot(t,vc)
xlabel('t(s)')
ylabel('v_c (t)')
title('Charging and Discharging of RC Circuit')
grid on
```

DIFFERENTIAL EQUATION

CHARGING AND DISCHARGING OF RC CIRCUIT

EXAMPLE 11



2ND ORDER DIFFERENTIAL EQUATION

- Function `ode23()` only solve 1st order differential equation. Thus, to solve the 2nd order differential equation, two stage 1st order differentiation is written in for the `odefun` function handle.

- For example, to solve $y'' = 2y' + 5y + 1$, write the function as the following by setting $y = y_1$ and $y' = y_2$:
  $y_1' = y_2$
  $y_2' = 2y' + 5y + 1 = 2y_2 + 5y_1 + 1$

```
odefun = @(t,y)[y(2); 2*y(2)+5*y(1)+1];
```

1st derivative. Always written it this way.

2nd derivative. Written according to the differential equation to be solved.

2ND ORDER DIFFERENTIAL EQUATION

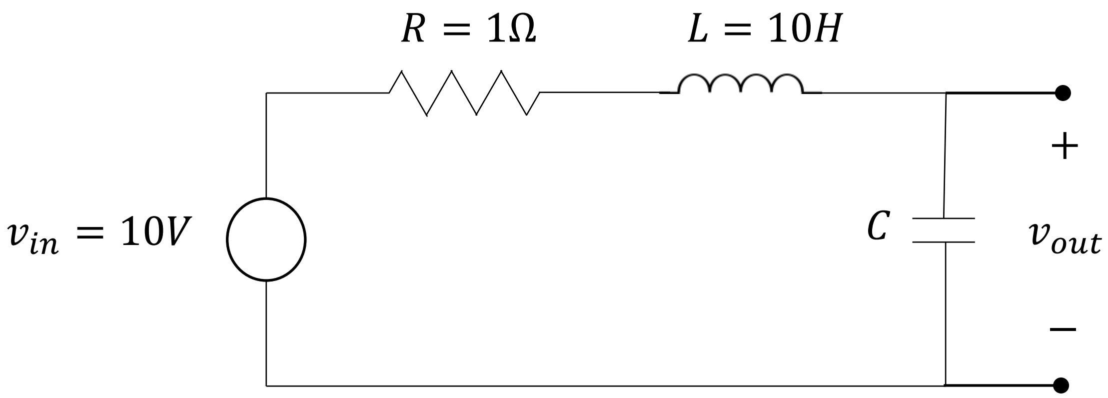
- Then, the function ode23() is given with 2 initial values (one for y_1 and one for y_2), specified as a vector. In this example the initial value is specified by vector

$$\texttt{inity} = [y_1 \;\; y_2] = [0 \;\; 0]$$

```
tspan = [0 2];
inity = [0 0];
[t,y] = ode23(@odefun(t,y),inity,tspan);

function d2dt = odefun(t,y)
d2dt = [y(2); 2*y(2)+5*y(1)+1];
```

DIFFERENTIAL EQUATION

RLC RESONANCE CIRCUIT



EXAMPLE 12

- One of the RLC circuit usage is as a resonance circuit, a circuit that generate an oscillating signal at specific frequency. The frequency can be computed as:

$$F = \frac{1}{2\pi\sqrt{LC}}$$

- The differential equation of the above RLC circuit is as below where the inductor contributed to the 2nd order differential equation. Plot v_{out} for C equals to $50\mu F$, $100\mu F$ and $635\mu F$.

$$v_{in} = RCv'_{out} + LCv''_{out} + v_{out}$$

RLC RESONANCE CIRCUIT

EXAMPLE 12

Solution

To solve the differential equation for v_{out} using function `ode23()`:

1) Rearrange the equation by setting the v''_{out} placed at the left side of the equation as below and write it in the function `odefun()`.

$$v''_{out} = (10 - v_{out} - Cv'_{out})/10C$$

2) Write the differential equation as 1st derivative equation by setting $v_1 = v_{out}$ and $v_2 = v'_{out}$.

$$v'_1 = v_2$$

$$v'_2 = (10 - v_1 - Cv_2)/10C$$

3) Set initial value for v_1 and v_2 as `[0 0]` and time interval as `[0 2]`.

4) Write the `ode23()` function and run the code.

5) C is set as the additional parameter to the anonymous function since function `ode23()` allowed only two inputs to the anonymous function.

RLC RESONANCE CIRCUIT

EXAMPLE 12

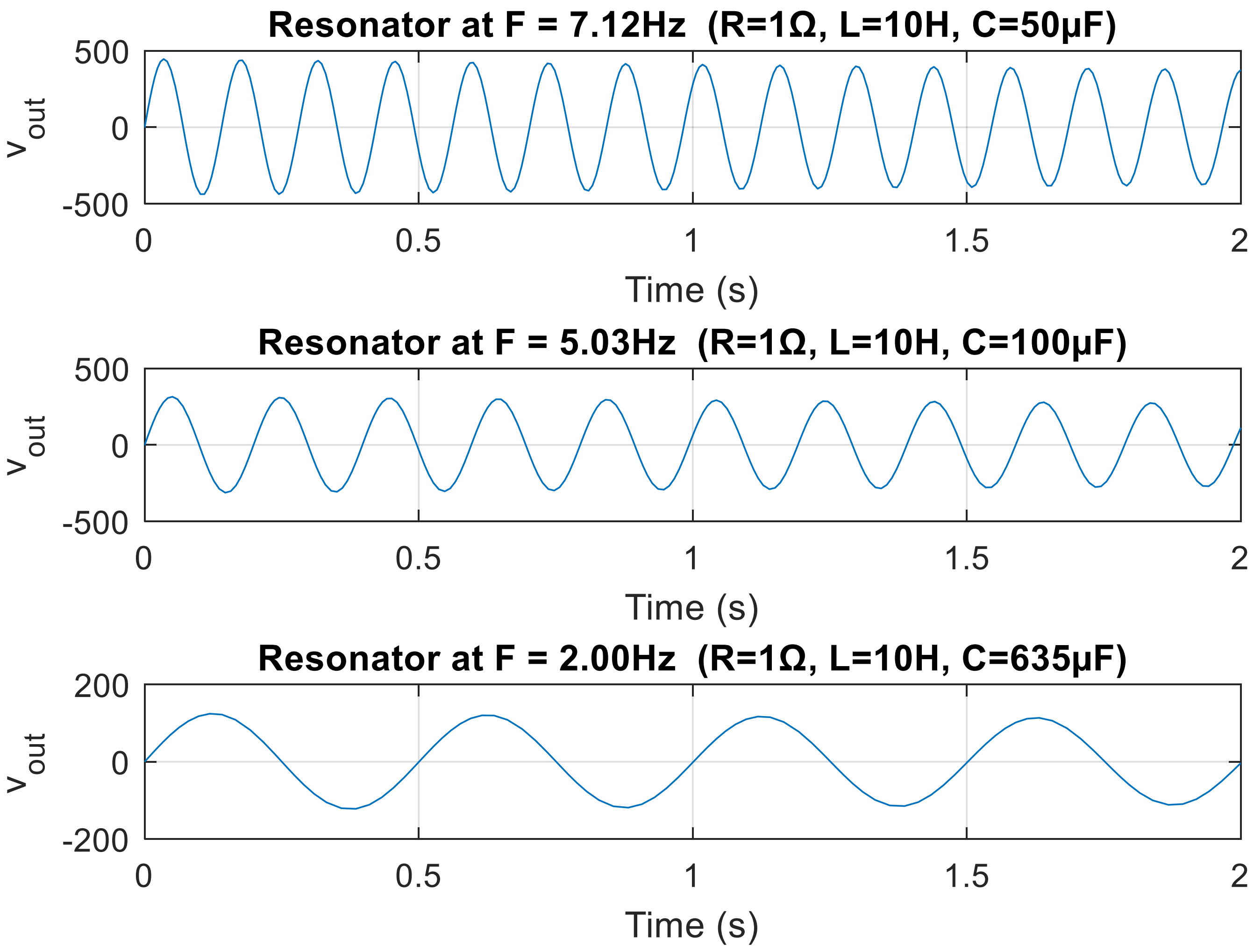
Below is the MATLAB code for Example 13.12.

```
C = [50e-6 100e-6 635e-6];
for n = 1:3
    odefun = @(t,v)[v(2); (10-v(1)-C(n)*v(2))/(10*C(n))];
    [t,y] = ode23(odefun, [0 2], [0 0]);

    F = 1/(2*pi*sqrt(10*C(n)));

    subplot(3,1,n), plot(t,y(:,2))
    xlabel('Time (s)')
    ylabel('v_{out}')
    titletext = sprintf(['Resonator at F = %.2fHz  '...
              '(R=1\x03A9, L=10H, C=%g\xB5'],F,C(n)/1e-6);
    title([titletext 'F)'])
    grid on
end
```

DIFFERENTIAL EQUATION

RLC RESONANCE CIRCUIT

EXAMPLE 12



DIFFERENTIAL EQUATION

QUIZ 2

PROVE THAT:

System	
1) Parallel RLC Circuit:	
Differential Equation	$V'' + \frac{1}{RC}V' + \frac{1}{LC}V = 0$
Odefun	`@(t,V)[V(2); -V(2)/(R*C)-V(1)/(L*C)]`
2) 2nd Order Active Lowpass Filter:	
Differential Equation	$V''_{out} + 1.4142\Omega_c V'_{out} + \Omega_c^2 V_{out} = HV_{in}$ Where Ω_c is the filter cutoff frequency in $rads^{-1}$ and H is the filter gain
Odefun	`@(t,vout)[vout(2); H*vin-1.4142*wc*vout(2)-(wc^2)*vout(1)];`

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

univteknologimalaysia utm_my utmofficial

Thank You

www.utm.my
innovative • entrepreneurial • global