

## CHAPTER 8

# Ordinary Differential Equations

Many modeling problems with engineering applications can be formulated using ordinary differential equations (ODEs). There are a few different definitions of differential equations. One of the simplest is “A differential equation is any equation which contains derivatives, either ordinary derivatives or partial derivatives,” as given in source [1]. From this definition, we can derive two types of differential equations: ordinary differential equations (ODEs) and partial differential equations (PDEs). ODEs contain one type of derivative or one independent variable, and PDEs, on the contrary, contain two or more derivatives or independent variables. For example, first-order ODEs can be expressed as follows:

$$\frac{dy}{dx} = f(x, y) \quad (\text{Equation 8-1})$$

Here,  $y(x)$  is a dependent variable whose values depend on the values of the independent variable of  $x$ . Another good example of ODEs is Newton’s Second Law of Motion, formulated as follows:

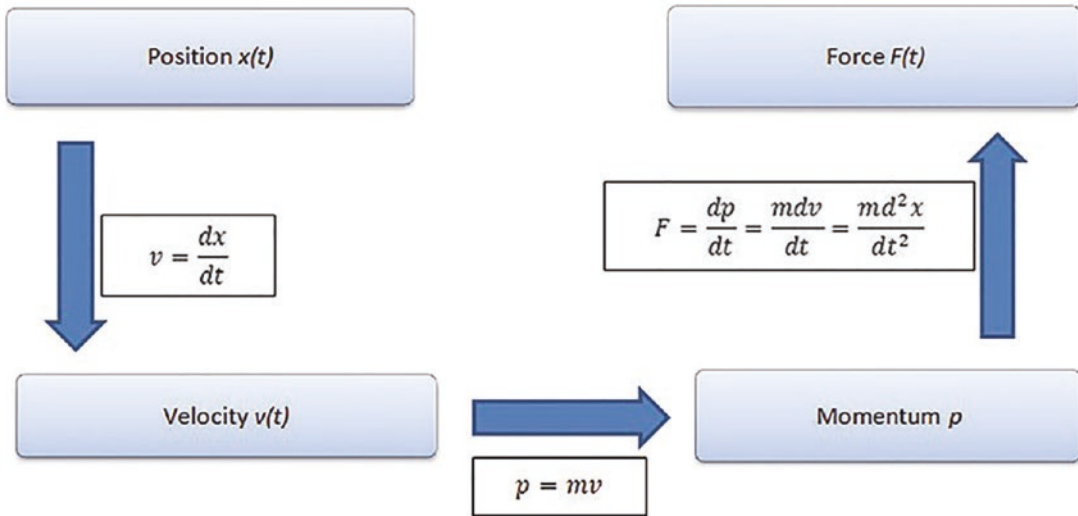
$$ma = \frac{dp}{dt} = \frac{mdv}{dt} = f(t, v) \quad (\text{Equation 8-2})$$

Here,  $F(t, v)$  is force, which is a function of time ( $t$ ) and velocity ( $v$ ).  $\frac{dv}{dt}$  is a velocity change rate (acceleration) of a moving object;  $m$  is the mass of a moving object;  $a$  is an acceleration of a moving object;  $p$  is momentum; and  $dp/dt$  is its derivative. This formulation of Newton’s Second Law can be also rewritten the following way:

$$\frac{md}{dt} \left( \frac{dx}{dt} \right) = \frac{md^2x}{dt^2} = F \left( t, x, \frac{dx}{dt} \right) \quad (\text{Equation 8-3})$$

Here, the derivative  $\left(\frac{dx}{dt}\right)$  of the displacement ( $x$ ) of a moving object is the velocity ( $v$ ).

In other words, the velocity is the rate of change of the displacement  $x(t)$  of a moving object in time. This can be visualized with the flowchart displayed in Figure 8-1.



**Figure 8-1.** Flowchart expressing motion and exerted force of a moving object

## Classifying ODEs

There are two classifications of ODE-related problems.

- *Initial value problems (IVPs):*  $\ddot{x} = xt - 3\dot{x}$  with initial conditions  $x(0) = 3, \dot{x}(0) = 1$
- *Boundary value problems (BVPs):*  $\ddot{x} = xt - 3\dot{x}$  with boundary conditions  $x(0) = 3, x(2) = 1.50$

IVPs are defined with ODEs together with a specified value, called the *initial condition*, of the unknown function at a given point in the solution domain. In the IVP of ODEs, there can be a unique solution, no solution, or many solutions. By definition, the IVP of ODEs can be explicitly or implicitly defined. Most of the IVP are explicitly defined. Let’s start with explicitly defined IVPs and then move to implicitly defined ones. In addition to solution type—how solution values change over the solution search

space—the IVPs are divided into stiff and nonstiff problems. Moreover, ODEs are grouped into two categories, linear and nonlinear, and divided into two groups, homogeneous and nonhomogeneous.

Here are some specific examples of different ODE types, categories, and groups:

- *Stiff ODEs:*  $\dot{y} = 3 * 10^8 y, t \in [0, 40]$
- *Nonstiff:*  $\dot{y} + 2y = 2t$
- *Linear ODEs:*  $\dot{v} = 9.81 - 0.198v$
- *Nonlinear ODEs:*  $\dot{v} = 9.81 - 0.198v^2$
- *Homogeneous ODEs:*  $\dot{y} + 2y = 0$
- *Nonhomogeneous ODEs:*  $\dot{y} + 2y = \sin(2t)$

The following are several examples of ODEs and their application areas.

## Example 1: Unconstrained Growth of Biological Organisms

This is an exponential growth problem that describes the unconstrained growth of biological organisms (such as bacteria). This behavior can also describe real estate or investment values, membership increase of a popular networking site, growth in retail businesses, positive feedback of electrical systems, and generated chemical reactions. The problem is formulated by the following first-order ODE:

$$\frac{dy}{dt} = \mu y \text{ has a solution: } y(t) = y_0 e^{\mu t}$$

## Example 2: Radioactive Decay

This refers to exponential decay, which describes many phenomena in nature and in engineering, such as radioactive decay, washout of chemicals in a reactor, discharge of a capacitor, and decomposition of material in a river. It's expressed using this first-order ODE:

$$\frac{dy}{dt} = -\mu y \text{ has a solution: } y(t) = y_0 e^{-\mu t}$$

Examples 1 and 2 are two simple examples of first-order ODEs.

### Example 3: Newton's Second Law

The motion of a falling object is expressed in the following equation using Newton's Second Law:

$$\frac{m d^2 y}{dt^2} = mg - \frac{\gamma dy}{dt}$$

This is a second-order ODE that has a solution in the following form:

$$y(t) = C_1 e^{-\left(\frac{\gamma t}{m}\right)} - \frac{m(mg - g\gamma t)}{\gamma^2} + C_2$$

Here,  $m$  is the mass of the falling object,  $g$  is gravitational acceleration, and  $\gamma$  is an air-drag coefficient of a falling object. Three parameters— $m$ ,  $g$ , and  $\gamma$ —are constant, the solution of a falling object, and  $C_1$  and  $C_2$  are arbitrary numbers that are dependent on the initial conditions. In other words, they can be computed considering the initial condition of a falling object.

There are a few methods that evaluate analytical solutions of ODEs, including separation of variables, introduction of new variables, and others. We look at specific examples of these types of ODEs to see how to evaluate their analytical solutions and compute numerical solutions. We do this by employing different techniques in the MATLAB/Simulink environment and writing scripts and building models. In computing analytical solutions of ODEs, we explain via specific examples how to use built-in functions of the Symbolic Math Toolbox.

For obvious reasons, considerable effort is placed on numerical solution methods rather than analytical solution search tools. It is not always possible or is too costly to evaluate analytical solutions of ODEs. Therefore, a numerical solution search is often best. There are a number of numerical methods. They are Euler (forward, backward, modified), Heun, the midpoint rule, Runge-Kutta, Runge-Kutta-Gill, Adams-Bashforth, Milne, Adams-Moulton, Taylor series, and trapezoidal rule methods.

Some of these methods are explicit, and others are implicit. To demonstrate how to employ these methods, we first describe their formulations and then work on their implementation algorithm for writing scripts (programs) explicitly. We do not attempt to derive any of the formulations used in these numerical methods. There are many literature sources [see 2, 3, 4, 5] that explain the theoretical aspects of these methods.

In solving the IVP using numerical methods, we start at an initial point (initial conditions) and then take a step (equal step-size or varying step-size) forward in time to compute the following numerical solution. Some of the previously named numerical methods (e.g., Euler's methods) are single-step methods, and others (Runge-Kutta, Adams-Bashforth, Milne, Adams-Moulton, and the Taylor series) are multistep methods. Single-step methods refer to only one previous point and its derivative to determine the current value. Other methods, such as Runge-Kutta methods, take some intermediate steps to obtain a higher-order step and then drop off values before taking the next step. Unlike single-step methods, multistep methods keep and use values from the previous steps instead of discarding them. This way, multistep methods link a few previously obtained values (solutions) and derivative values. All of these methods, i.e., the single-step and multistep methods, are assessed based on their accuracy and efficiency in terms of the computation time and resources (e.g., machine time) spent to compute numerical solutions for specific types of IVPs of ODEs. Nevertheless, it remains true that most solutions of the first-, second-, or higher-order IVPs cannot be found by analytical means. Therefore, we need to employ various numerical methods.

## Analytical Methods

The Symbolic Math Toolbox has several functions that are capable of evaluating analytical solutions of many analytically solvable ODEs. There are two commands (built-in functions) by which analytical solutions of some ODEs can be evaluated—`dsolve` and `ilaplace/laplace`.

Note that in this section we demonstrate—via a few examples of first- and second-order ODEs and systems of coupled differential equations—how to compute analytical solutions of ODEs.

### DSOLVE

One ODE solver tool for computing an analytical (or general) solution of any given ODE in MATLAB is `dsolve`. It can be used with the following general syntax:

```
Solution = dsolve(equation)
Solution = dsolve(equation, conditions)
Solution = dsolve(equation, conditions, Name, Value)
[y1,...,yN] = dsolve(equations)
[y1,...,yN] = dsolve(equations, conditions)
[y1,...,yN] = dsolve(equations, conditions, Name, Value)
```

---

## Example 1: Using DSOLVE

Given a first-order ODE:  $y + 2ty^2 = 0$  with no known initial or boundary conditions. Let's solve it using `dsolve()`.

```
>> y_solution=dsolve('Dy=-2*y^2*t')
Y_solution=
-1/(C3-t^2)
```

Note that C3 is defined from the initial or boundary conditions of the given ODE. There is also an alternative command. In later versions of MATLAB (starting with MATLAB 2012), we can solve the given problem by using the following command syntax:

```
>>syms y(t); y_sol=dsolve(diff(y) == - 2*y^2*t)
y_sol =
0
-1/(- t^2+C3)
```

## Example 2: Plotting the Found Solution with dsolve

Given a first-order ODE:  $\dot{y} + 2ty^2 = 0$  with the initial condition  $y(0) = 0.50$ . Let's solve it using `dsolve()`.

```
>> Solution=dsolve('Dy=-2*y^2*t', 'y(0)=0.5')
Solution =
1/(t^2 + 2)
```

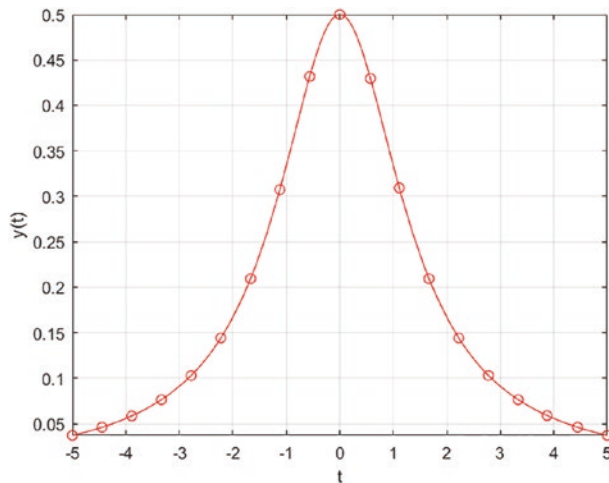
The alternative command syntax with later versions of MATLAB is as follows:

```
>> syms y(t); Solution=dsolve(diff(y) == -2*y^2*t, y(0)==0.5)
```

Solution =  $1/(t^2 + 2)$

The evaluated analytical solution in a symbolic formulation can be plotted with `fplot`. (See Figure 8-2.)

```
>> fplot(Solution, [-5, 5], 'ro-'); grid on; xlabel('t'); ylabel 'y(t)'
```



**Figure 8-2.** Analytical solution of  $\dot{y} + 2ty^2 = 0$  with the initial condition  $y(0) = 0.50$

Numerical values of the analytical solution (the equation) can be computed by vectorizing (parameterizing) the symbolic formulation (the solution), as shown here:

---

```
>> ysol=vectorize(solution)
ysol =
1./(t.^2 + 2)
>> t=(-5:.1:5); ysol_values=eval(ysol);
```

---

### Example 3: Adding an Unspecified Parameter

Given  $\dot{y} + ky^2 = 0$ ,  $y(0) = 0.50$ . Let's solve it using `dsolve()`. Note that this exercise has one unspecified parameter  $k$ .

```
>> syms k
>> solution=dsolve('Dy=-k*y^2*t', 'y(0)=0.5')
solution =
1/((k*t^2)/2 + 2)
```

---

An alternative command syntax is as follows:

---

```
>> syms y(t) k;
solution=dsolve(diff(y) == -k*y^2*t, y(0)==0.5)
solution =
1/((k*t^2)/2 + 2)
```

---

**Note** Options in `dsolve` need to be set appropriately depending on the problem type. In MatlaB 2008–2010 or earlier versions, you should set `IgnoreAnalyticConstraints` to `none` to obtain all possible solutions.

---

Here's an example:

```
solution=dsolve('Dy=-k*y^2*t','y(0)=0.5','IgnoreAnalyticConstraints','none')
```

---

**Note** For MatlaB 2012 or later versions, you should set `IgnoreAnalyticConstraints` to `false` to get all possible correct answers for all the argument values. Otherwise, `dsolve` may output an incorrect answer because of its pre-algebraic simplifications.

---

Here's an example:

```
solution=dsolve(diff(y)==-k*y^2*t, y(0)==0.5,'IgnoreAnalyticConstraints',
false)
```

## Second-Order ODEs and a System of ODEs

There are a myriad of processes and phenomena that are expressed via second-order differential equations. Examples include simple harmonic motions of a spring-mass



system, motions of objects with some acceleration (Newton's Second Law), damped vibrations, current flows in resistor-capacitor-inductance circuits, and so forth. In general, second-order ODEs are expressed in two different forms—homogeneous (Equation 8-4) and nonhomogeneous (Equation 8-5).

$$\ddot{y} + p(x)\dot{y} + q(x)y = 0 \quad (\text{Equation 8-4})$$

$$\ddot{y} + p(x)\dot{y} + q(x)y = p(x) \quad (\text{Equation 8-5})$$

Note that the homogeneous ODEs in Equation 8-4 always have one trivial solution, which is  $y(x) = 0$ . It satisfies the givens in Equation 8-4. With respect to the independent functions  $p(x)$ ,  $q(x)$ , and  $g(x)$ , the ODEs can be linear or nonlinear. In some cases, the independent functions  $p(x)$ ,  $q(x)$ , and  $g(x)$  can be constant values or nonconstant values.

Let's consider several examples of second-order ODEs to see how to compute general and particular solutions with MATLAB's Symbolic Math Toolbox.

## Example 1: dsolve with a Second-Order ODE

Given a second-order ODE:  $\ddot{u} + 100u = 2.5\sin(10t)$ ,  $u(0) = 0$ ,  $\dot{u}(0) = 0$  with no known initial or boundary conditions. Let's solve it using `dsolve()`.

---

```

usol=dsolve('D2u+100*u=2.5*sin(10*t)', 'u(0)=0', 'Du(0)=0'); pretty(usol)
%% Alternative syntax
syms u(t)
Du = diff(u);
u(t) = dsolve(diff(u, 2)==2.5*sin(10*t)-100*u, u(0)==0, Du(0) == 0);
pretty(u(t))

```

---

The following is the output from executing the two short scripts/commands:

```

sin(10 t) 3    sin(30 t)                / t    sin(20 t) \
----- - ----- - cos(10 t) | - - ----- |
      320          320                \ 8      160      /

```

## Example 2: System ODEs

Given a system of ODEs:  $\begin{cases} y_1' = y_2 \\ y_2' = -y_1 - 0.125y_2 \end{cases}$ ,  $y_1(0) = 1$ ,  $y_2(0) = 0$ .

Let's solve it using `dsolve()`.

The given problem is a system of two first-order ODEs. This problem can be solved directly with `dsolve`, similar to the previous examples.

```
% System of two 1st-order ODEs solved with dsolve
yt=dsolve('Dy1=y2', 'Dy2=-y1-0.125*y2', 'y1(0)=1', 'y2(0)=0');
pretty(yt.y1) pretty(yt.y2)
% Alternative syntax
syms y1(t) y2(t)
z=dsolve(diff(y1,1)==y2, diff(y2,1)==(-y1-0.125*y2), y1(0)==1, y2(0)==0);
pretty(z.y1), pretty(z.y2)
```

The computed analytical solutions of the problem displayed in the command window are not shown here. These are the computed solutions:

$$y_1(t) = \frac{\cos\left(\frac{\sqrt{255}t}{16}\right)}{\sqrt[16]{e^t}} + \frac{\sqrt{255} \sin\left(\frac{\sqrt{255}t}{16}\right)}{255\sqrt[16]{e^t}}$$

$$y_2(t) = -\frac{16\sqrt{255} \sin\left(\frac{\sqrt{255}t}{16}\right)}{255\sqrt[16]{e^t}}$$

### Example 3: Unsolvable Solutions Using `dsolve`

Given a second-order ODE:  $2\ddot{y} + 3\dot{y} - |y|\cos(100t) = 2$ ,  $y(0) = 1$ ,  $\dot{y}(0) = 2$ .  
Let's solve it using `dsolve()`.

```
>> syms y(t); Dy = diff(y,t); D2y = diff(y, t,2);
>> Solution=dsolve(2*D2y+3*(Dy^3)-cos(100*t)*abs(y)- 2==0, y(0)==1,
Dy(0)==2)
Warning: Unable to find symbolic solution.
Solution =
[ empty sym ]
```

When the analytical solutions cannot be found with `dsolve`, the only option will be numerical solution.

## Example 4: Computing an Analytical Solution

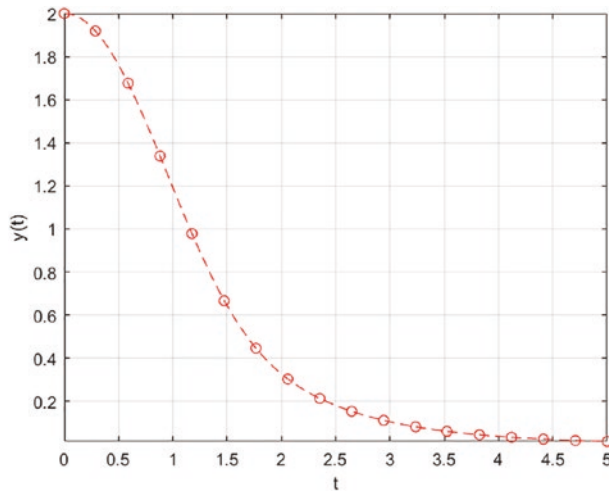
Say we have this:  $\dot{y} - |y|e^t = 2$ ,  $y(0) = 2$ . Here are the commands used to compute an analytical solution of the given exercise:

---

```
syms y(t)
Dy = diff(y,t);
Solution = dsolve(Dy==2+abs(y)*exp(t), y(0)==2);
fplot(Solution, [0, 5], 'r--o'), grid on
xlabel 't'
ylabel('y(t)')
```

---

Figure 8-3 shows the plot of the found analytical solution.

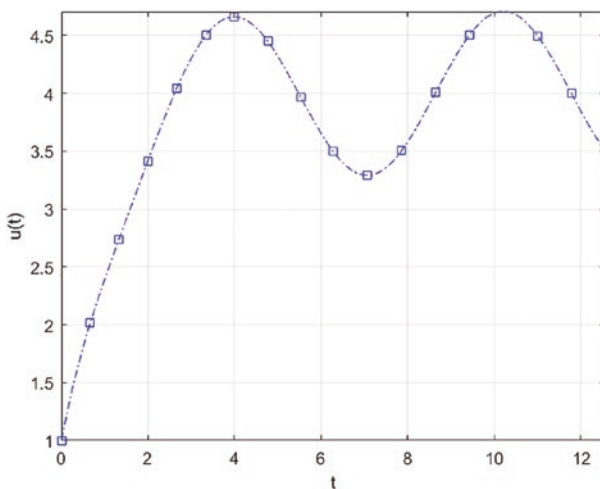


**Figure 8-3.** The found analytical solution  $\dot{y} - |y|e^t = 2$ ,  $y(0) = 2$

## Example 5: An Interesting ODE

Let's consider a second-order ODE, as follows:  $\ddot{u} + \dot{u} = \sin(t)$ ,  $u(0) = 1$ ,  $\dot{u}(0) = 2$ .

```
syms u(t)
Du = diff(u,t);
D2u = diff(u,t, 2);
Solution = dsolve(D2u==sin(t)-Du, u(0)==1, Du(0)==2);
fplot(Solution, [0, 4*pi], 'b-.s'), grid on
xlabel ('t'); ylabel('u(t)')
```



**Figure 8-4.** Analytical solution plot of  $\ddot{u} + \dot{u} = \sin(t)$ ,  $u(0) = 1$ ,  $\dot{u}(0) = 2$

## Laplace Transforms

Solutions of linear ordinary differential equations with constant coefficients can be evaluated by using the Laplace transformation. One of the most important features of the Laplace transforms in solving a differential equation is that the transformed equation is an algebraic equation. It will be used to define a solution to the given differential equation. In general, the Laplace transform application to solving differential equations can be formulated in the following way.

Let's consider the  $n$ th order derivative of  $y^n(x) = f(t)$ . The Laplace transform of  $y^n(x)$  is as follows:

$$\mathcal{L}\left\{\frac{d^n u}{dt^n} = f(t)\right\} \implies s^n U(s) - s^{n-1}u^{n-1}(0) - \dots - su(0) - su(0) = F(s) \quad (\text{Equation 8-6})$$

Or

$$s^n U(s) - \sum_{i=1}^n s^{n-i} u^{i-1}(0) = F(s) \quad (\text{Equation 8-7})$$

In Equation 8-6 or 8-7, if you substitute constant values of initial conditions at  $t = 0$  given as  $y(0) = a_0$ ;  $y'(0) = a_1$ ;  $y''(0) = a_2$ ; ...;  $y^{n-2}(0) = a_{n-2}$ ;  $y^{n-1}(0) = a_{n-1}$ , you can rewrite the expression (Equation 8-6 or 8-7) as follows:

$$s^n U(s) - s^{n-1}a_0 - \dots - sa_n - sa_{n-1} = F(s) \quad (\text{Equation 8-8})$$

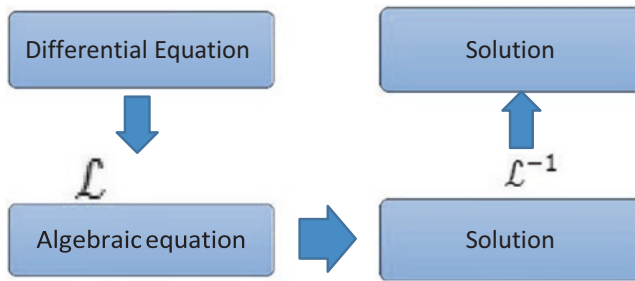
Subsequently, we first solve for  $Y(s)$ , take the inverse Laplace transform from  $Y(s)$ , and obtain the solution  $y(t)$  of the  $n$ th order differential equation.

The general procedure for applying the Laplace and the inverse Laplace transforms to determine the solution of differential equations with constant coefficients is as follows:

- [1]. Take the Laplace transforms from both sides of the given equation.
- [2]. Solve for  $Y(s)$  in terms of  $F(s)$  and other unknowns.
- [3]. Take the inverse Laplace transform of the found expression to obtain the final solution of the problem.

Note that in step 3, you should also break the expression from step 2 into partial fractions in order to use tables of the inverse Laplace transform correspondences.

A schematic view of the Laplace and inverse Laplace transforms is given in the flowchart shown in Figure 8-5.



**Figure 8-5.** Flowchart of solving ODE with Laplace transform and its inverse

### Example 1: First Laplace Transform

Let’s consider a second-order nonhomogeneous differential equation.

$$\frac{d^2y}{dt^2} + \frac{A dy}{dt} + C = e^{nt}, y(0) = k, dy(0) = m$$

Now, by applying the steps depicted in the flowchart of the Laplace and inverse transforms from the flowchart, you write the Laplace transform of the given problem in explicit steps.

$$\mathcal{L}\left\{\frac{d^2y}{dt^2} + \frac{A dy}{dt} + C = e^{nt}\right\} = \mathcal{L}\left\{\frac{d^2y}{dt^2}\right\} + \mathcal{L}\left\{\frac{A dy}{dt}\right\} + \mathcal{L}\{C\} = \mathcal{L}\{e^{nt}\} \tag{Equation 8-9}$$

$$\mathcal{L}\{e^{nt}\} = \frac{1}{s-n} \tag{Equation 8-10}$$

Now, you can work on the left-hand side of Equation 8-9 starting from the highest order.

$$\mathcal{L}\left\{\frac{d^2y}{dt^2}\right\} = s^2Y(s) - dy(0) - s * y(0) = s^2Y(s) - k - s * m \tag{Equation 8-11}$$

$$\mathcal{L}\left\{\frac{A dy}{dt}\right\} = A * \mathcal{L}\left\{\frac{dy}{dt}\right\} = A * (s * Y(s) - y(0)) = A * s * Y(s) - m \tag{Equation 8-12}$$

$$\mathcal{L}\{C\} = C \tag{Equation 8-13}$$

By plugging Equations 8-10, 8-11, 8-12, and 8-13 back into Equation 8-9, you obtain the assembled expression given in Equation 8-14.

$$s^2Y(s) - k - sm + AsY(s) - m + C = \frac{1}{s-n} \quad (\text{Equation 8-14})$$

You solve Equation 8-15 for  $Y(s)$ .

$$Y(s) = \frac{1 + (sm + (k + m - C))(s - n)}{(s - n)(s^2 + As)} = \frac{Cn - Cs - kn - mn + ks + ms + ms^2}{s(A + s)(n - s)} \quad (\text{Equation 8-15})$$

From the expression of  $Y(s)$  in Equation 8-15, you can split this into partial fractions and take the inverse Laplace transform of both sides. You obtain Equation 8-16, which is the  $y(t)$  of the given differential equation:

$$y(t) = \frac{e^{nt}}{An + n^2} - \frac{Cn - kn + mn + 1}{An} + \frac{Cn - kn - mn - A(k - mn - C) + 1}{Ae^{At}(A + n)} \quad (\text{Equation 8-16})$$

The built-in `laplace()` function of the Symbolic Math Toolbox is used to evaluate the Laplace transform of any algebraic expression or differential equation. Likewise, the `ilaplace()` function of the Symbolic Math Toolbox is used to compute the inverse of the evaluated Laplace transformed  $s$  domain expression. These two functions handle all transformations by breaking up the partial fraction procedures automatically and then compute an analytical solution of a given ODE exercise.

## LAPLACE/ILAPLACE

As mentioned, `laplace/ilaplace` are based on the Laplace and inverse Laplace transforms, which are built-in function tools of the Symbolic Math Toolbox. The general syntax of `laplace/ilaplace` is as follows:

---

```
F=laplace(f)
F=laplace(f, t)
F=laplace(f, var1, var2)
```

---

and as follows:

---

```
f=ilaplace(F)
f=ilaplace(F, s) f=ilaplace(F, var1, var2)
```

---

## Example 2: Using LAPLACE

Given  $x(t) = \sin(2t)$ , the Laplace transform of  $x(t)$  is computed with the following command syntax:

```
>> syms t
>> xt=sin(2*t); Xs=laplace(xt)
2/(s^2 + 4)
```

Given  $y(t) = \sin(Kt)$ . Let's compute its Laplace transform with `laplace()`.

```
>> syms t K
>> yt=sin(K*t); Ys=laplace(yt)
K/(K^2 + s^2)
```

## Example 3: A Final LAPLACE

Compute the Laplace transform of  $y(x) = ax^3 + b$ .

```
>> syms x a b y(x)
>> y(x)=a*x^3+b; Ys=laplace(y(x))
Ys =
(6*a)/s^4 + b/s
```

You can also obtain the  $t$  variable domain instead of  $s$ .

```
>> syms x t a b
>> y=a*x^3+b; Yt=laplace(y, x, t)
Yt =
(6*a)/t^4 + b/t
```

The `ilaplace()` function syntax and implementation are exactly the same for `laplace`. Let's look at several ODE exercises to see how to use `laplace/ilaplace` and compare their evaluated solutions to the ones obtained with `dsolve`.

## Example 4: Comparing LAPLACE/ILAPLACE with DSOLVE

Let's solve  $\dot{y} + 2y = 0$ ,  $y(0) = 0.5$  with `laplace/ilaplace` and `dsolve`. The following script (`ODE_Laplace.m`) shows the solution:



---

```

% ODE_Laplace.m
clearvars; clc; close all
% Step #1. Define symbolic variables' names
syms t s y(t) Y
Dy = diff(y(t),t);
ODE1=Dy== -2*y(t);
% Step #2. Laplace Transforms
LT_A=laplace(ODE1, t, s);
% Step #3. Substitute ICs and initiate an unknown Y
LT_A=subs(LT_A,{laplace(y(t),t, s),y(0)},{Y,0.5});
% Step #4. Solve for Y (unknown)
Y=solve(LT_A, Y);
display('Laplace Transforms of the given ODE with ICs'); disp(Y)
% Step #5. Evaluate Inverse Laplace Transform
Solution_Laplace=ilaplace(Y);
display('Solution found using Laplace Transforms: ')
pretty(Solution_Laplace)
% Step #6. Compute numerical values and plot them
t=0:.01:2.5; LTsol=eval(vectorize(Solution_Laplace));
figure, semilogx(t, LTsol, 'ko')
xlabel('t'), ylabel('solution values')
title('laplace/ilaplace vs dsolve ')
grid on; hold on
%% Compare with dsolve solution method
clearvars;
syms y(t)
Dy = diff(y, t);
Y_d=dsolve(Dy== -2*y, y(0)==0.5);
disp('Solution with dsolve')
t=0:.01:2.5;
pretty(Y_d); Y_sol=eval(vectorize(Y_d));
plot(t,Y_sol, 'b-', 'linewidth', 2), grid minor
legend('laplace+ilaplace', 'dsolve')
hold off; axis tight

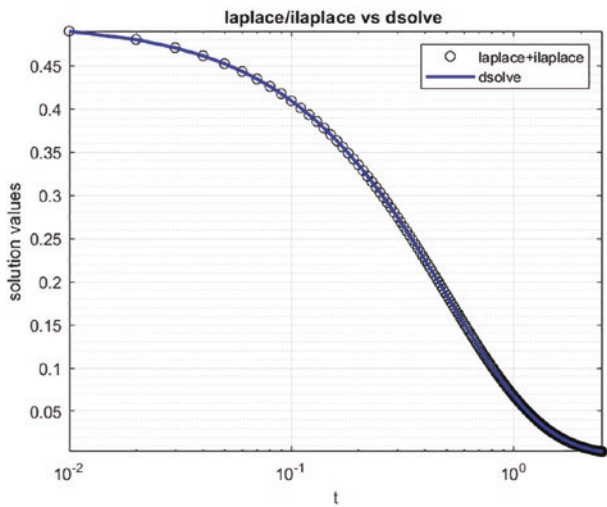
```

---

After executing the ODE\_Laplace.m script, the following output is obtained:

```
Laplace Transforms of the given ODE with ICs
1/(2*s + 4)
Solution found using Laplace Transforms: exp(-2 t)
-----
2
Solution with dsolve exp(-2 t)
-----
2
```

The plot of solutions shown in Figure 8-6 clearly displays a perfect convergence of solutions found with laplace/ilaplace and dsolve.



**Figure 8-6.** The problem  $y + 2y = 0$ ,  $y(0) = 0.5$  solved with laplace/ilaplace and dsolve

## Example 5: Convergent Answers

Given  $\ddot{y} + \dot{y} = \sin(t)$ ,  $y(0) = 1$ ,  $\dot{y}(0) = 2$ , here is the solution script (Laplace\_vs\_Dsolve.m) of this second-order nonhomogeneous ODE with laplace, ilaplace, and dsolve:

---

```
clearvars, clc, close all
syms t s y(t) Y(s)
Dy = diff(y(t), t);
D2Y = diff(y(t), t, 2);
ODE2nd=D2Y== sin(t)-Dy;
% Step 1. Laplace Transforms
LT_A=laplace(ODE2nd, t, s);
% Step 2. Substitute ICs and initiate an unknown Y
LT_A=subs(LT_A,{laplace(y(t),t, s),subs(diff(y(t),t), t,
0),y(0))},{Y(s),2,1});
% Step 3. Solve for Y unknown
Y=isolate(LT_A, Y);
%disp('Laplace Transforms of the given ODE with ICs');
disp(Y)
Solution_Laplace=ilaplace(rhs(Y));
disp('Solution found using Laplace Transforms: ')
pretty(Solution_Laplace);
t=0:.01:13; LTsol=eval(vectorize(Solution_Laplace));
figure, plot(t, LTsol, 'ro-'); xlabel('t'), ylabel('solution values')
title('laplace/ilaplace vs. dsolve: ddy+dy=sin(t)'); hold on
% dsolve solution method
Y=dsolve('D2y+Dy=sin(t)', 'y(0)=1, Dy(0)=2', 't');
disp('Solution with dsolve: ');
pretty(Y)
fplot(Y, [0, 13], 'b-', 'linewidth', 2); grid minor
legend('laplace+ilaplace', 'dsolve', 'location', 'SE'); hold off
```

---

The computed analytical solutions are as follows:

$$Y(s) == (s + 1/(s^2 + 1) + 3)/(s^2 + s)$$

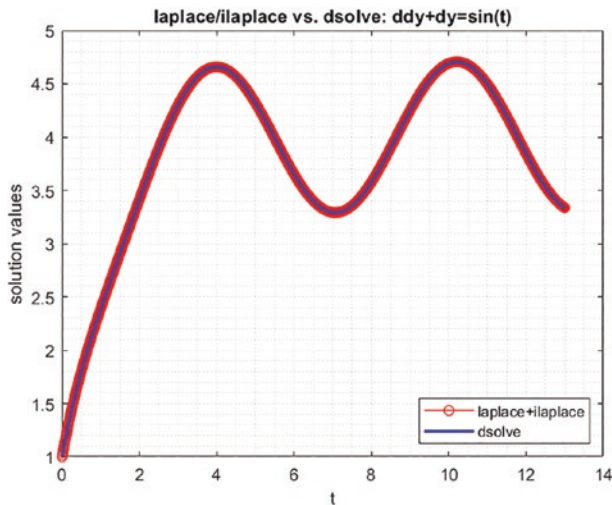
Solution found using Laplace Transforms:

$$4 - \frac{\cos(t)}{2} - \frac{\sin(t)}{2} - \frac{5 \exp(-t)}{2}$$

Solution with dsolve:

$$4 - \frac{\sqrt{2} \cos\left(t - \frac{\pi}{4}\right)}{2} - \frac{5 \exp(-t)}{2}$$

From the plot displayed in Figure 8-7, it is clear that the solutions found via the Laplace transforms (laplace/ilaplace) and the dsolve functions converge perfectly well. Both functions evaluate the same analytical solution of a given ODE.



**Figure 8-7.** Analytical solutions of  $\ddot{y} + \dot{y} = \sin(t)$ ,  $y(0) = 1, \dot{y}(0) = 2$

## Example 6: No Analytical Solution

Given the following second-order nonhomogeneous and nonlinear ODE, let's try to solve it using `syms`, `diff()`, and `laplace()`.

$$2\ddot{y} + 3\dot{y} - |y|\cos(100t) = 2, y(0) = 1, \dot{y}(0) = 2.$$

Let's solve Here is the solution script (`Lap_inv_Lap.m`) with the Laplace and inverse Laplace transforms:

---

```
% Lap_inv_Lap.m
clearvars, clc, close all
syms t s y(t) Y(s)
Dy=diff(y(t),t);
D2y=diff(y(t),t,2);
ODE2nd=D2y==0.5*(-3*(Dy)^3+cos(100*t)*abs(y(t))+2);
% Step 1. Laplace Transforms
LT_A=laplace(ODE2nd, t, s);
% Step 2. Substitute ICs and initiate an unknown Y
LT_A=subs(LT_A,{laplace(y(t),t, s),subs(diff(y(t),t), t,
0),y(0)},{Y(s),0,0});
% Step 3. Solve for Y unknown
Y=isolate(LT_A, Y);
% Step 3. Solve for Y unknown Y=solve(LT_A, Y);
disp('Laplace Transforms of the given ODE with ICs');
disp(Y)
Solution_Laplace=ilaplace(Y);
disp('Solution found using Laplace Transforms: ')
pretty(Solution_Laplace)
```

---

The Lap\_inv\_Lap.m script produces the following output in the Command window:

```
Laplace Transforms of the given ODE with ICs
Y(s) == (4 + s*laplace(abs(y(t)), t, s - 100i) + s*laplace(abs(y(t)), t, s
+ 100i) - 6*s*laplace(diff(y(t), t)^3, t, s))/(4*s^3)
Solution found using Laplace Transforms:
          t
          /
          |
          3 | (t - u23) | ---- y(u23) | \3
          2 /          \ du23          /
          t      0
ilaplace(Y(s), s, t) == -----
          2                2
          / laplace(|y(t)|, t, s - 100i) \
ilaplace| -----, s, t |
          |                2                |
          \                s                /
+ -----
          4
          / laplace(|y(t)|, t, s + 100i) \
ilaplace| -----, s, t |
          |                2                |
          \                s                /
+ -----
          4
```

This output means that no analytical solution is computed explicitly with laplace/ilaplace, just like with the dsolve function tools.

### Example 7: Demonstrating Efficiency and Effortlessness

Given a second-order nonhomogeneous ODE where  $g(t)$  is a forcing function that is discontinuous and defined by the following expression, let's solve it by applying the Laplace transform.

$$g(t) = u_2(t) - u_{10}(t) = \begin{cases} 5, & 2 \leq t \leq 10 \\ 0, & 0 \leq t < 2 \text{ and } t \geq 10 \end{cases}$$

The Laplace transform of the given equation is as follows:

$$\mathcal{L}\{2y + 3\dot{y} - 2y\} = \mathcal{L}\{u_2(t) - u_{10}(t)\}$$

$$2s^2Y(s) - 2sy(0) - 2\dot{y}(0) + 3sY(s) - y(0) - 2*Y(s) = \frac{5(e^{-2s} - e^{-10s})}{s}$$

$$Y(s) = \frac{5(e^{-2s} - e^{-10s})}{s(2s^2 + 3s - 2)}$$

Note that  $e^{-2s}$  and  $e^{-10s}$  are explained with time delays in the system output signals; in other words, -2 and -10 mean 2 and 10 seconds of time delays. 5 is the magnitude of the Heaviside (step) function.

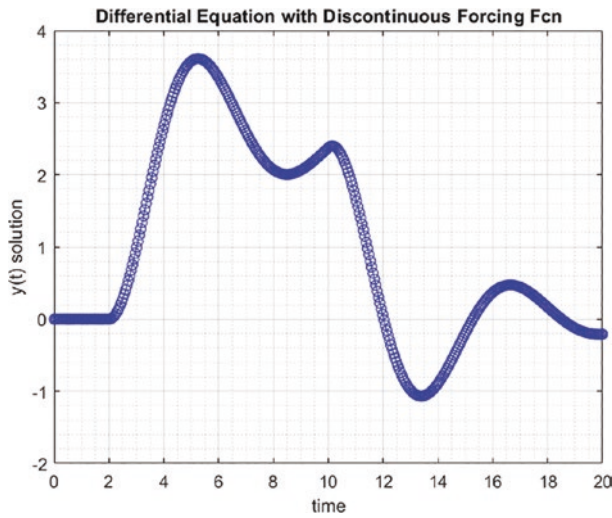
The formulation  $Y(s)$  is the solution of the differential equation in the  $s$  domain, but we need it in the time domain. Thus, you need to compute its inverse Laplace transform:  $\mathcal{L}^{-1}\{Y(s)\} = y(t)$ . By employing `ilaplace()`, the next short script (`Lap_4_non_homog.m`) is created. It solves the given problem and computes its analytical and numerical solutions.

---

```
% Lap_4_non_homog.m
syms t s
F=5*(exp(-2*s)-exp(-10*s))/s;
Y=2*s^2+s+2;
TF=F/Y; TFt=ilaplace(TF);
pretty(TFt);
Sol=vectorize(TFt);
t=linspace(0, 20, 400);
S=eval(Sol); plot(t, S, 'bo-'); grid minor
title('Differential Equation with Discontinuous Forcing Fcn')
grid on, xlabel('time'), ylabel('y(t) solution'), shg
```

---

After executing the script, the next solution plot is obtained along with the solution formulation, as shown in Figure 8-8.



**Figure 8-8.** Simulation of the second-order nonhomogeneous ODE subject to the discontinuous forcing function

>> pretty(TFt)

$$\begin{aligned}
 & \text{heaviside}(t - 10) \left[ \frac{\exp\left(-\frac{t-10}{4}\right) \cos\left(\frac{\sqrt{15}}{4}(t-10)\right) + \frac{\sqrt{15} \sin\left(\frac{\sqrt{15}}{4}(t-10)\right)}{15}}{2} \right] \\
 & + 5 - \text{heaviside}(t - 2) \left[ \frac{\exp\left(-\frac{t-2}{4}\right) \cos\left(\frac{\sqrt{15}}{4}(t-2)\right) + \frac{\sqrt{15} \sin\left(\frac{\sqrt{15}}{4}(t-2)\right)}{15}}{2} \right]
 \end{aligned}$$

where

$$\#1 == \frac{\sqrt{15} (t - 10)}{4}$$

$$\#2 == \frac{\sqrt{15} (t - 2)}{4}$$



From the previous exercise, the following points can be drawn. Using the Laplace transforms (`laplace/ilaplace`) to compute analytical solutions of nonhomogeneous ODEs subject to external forcing functions, which are discontinuous, is relatively easy, fast, and effortless. Such exercises are found often within control engineering problems. Moreover, note that the Laplace and inverse Laplace transforms (`laplace/ilaplace`) are straightforward to implement in solving ODEs. The solutions of ODEs found with them match the ones found by `dsolve()` perfectly well. As mentioned, many ODEs cannot be solved analytically with the `laplace/ilaplace` and `dsolve` functions. Thus, numerical methods are often the only option.

## MATLAB Built-in ODEx Solvers

In MATLAB, there are a few built-in ODE solvers, namely, `ode15s`, `ode15i`, `ode23`, `ode23s`, `ode23t`, `ode23tb`, `ode45`, and `ode113`, which are efficient in finding numerical solutions of many different types of initial value problems. These solvers are based on explicit Runge-Kutta and implicit Adams-Bashforth-Moulton methods with different implementation algorithms and ODE solver methods, namely, Dormand-Prince (`ode45`), Bogacki-Shampine (`ode23`), Rosenbrock (`ode23s`), trapezoidal rule (`ode23t`), Adams-Bashforth-Moulton (`ode113`), Gear's method (`ode15s`), and so forth. Using MATLAB's built-in ODE solvers is relatively simple, and these are the following general syntaxes of the ODE solvers:

---

```

solver(odefun,tspan,y0)
[T,Y] = solver(odefun,tspan,y0)
[T,Y] = solver(odefun,tspan,y0,options)
[T,Y,TE,YE,IE] = solver(odefun,tspan,y0,options)
sol = solver(odefun,[t0 tf],y0...)

```

---

Any of `ode15s`, `ode15i`, `ode23`, `ode23s`, `ode23t`, `ode23tb`, `ode45`, and `ode113` can be chosen depending on the given problem type, for instance, whether the given problem is stiff (how far stiff, e.g., very stiff or moderately stiff) or nonstiff, explicit or implicit.

It is worth noting that an ODE solver type needs to be selected carefully. In selecting a solver type, the recommendations given in Table 8-1 should be considered. These are taken from the help library of the MATLAB package.

**Table 8-1.** *MATLAB's Built-in ODEx Solvers*

<b>Solver Type</b>	<b>Problem Type</b>	<b>Accuracy</b>	<b>When to Apply</b>
ode15i	Fully Implicit	Medium	For only fully implicit IVP
ode15s	Stiff	Low to Medium	If ode45 is too slow in finding solutions of the problem due to its stiffness
ode23	Nonstiff	Low	For moderately stiff problems with crude error tolerances
ode23s	Stiff	Low	For stiff problems with crude error tolerances
ode23t	Moderately stiff	Low	For moderately stiff problems
ode23tb	Stiff	Low	For stiff problems with crude error tolerances
ode45	<b>Nonstiff</b>	<b>Medium</b>	<b>Recommended for most problems; must be the first ODE solver to try</b>
ode113	Nonstiff	Low to high	For problems with tight error tolerances

Moreover, the efficiency of these solvers depends on the chosen step type (fixed or variable), the size, and the relative and absolute error tolerances that directly affect the accuracy of simulation results and efficiency of computation processes. While using built-in ODE solvers, the step size can be chosen as variable (automatically chosen) or fixed/specified by a user. All built-in ODE solvers by default will take variable step sizes automatically depending on the type of a given IVP (e.g., a stiffness level) and a solution search space. Error tolerance can be controlled in ODE solvers via their setting options. Hereafter, we study in real exercises all these key aspects and settings of ODE built-in solvers.

ODEFUN for the ODE solvers can be defined by using the following:

1. Anonymous function with function (@)
2. Function file (\*.m file)
3. matlabFunction: function file (\*.m file) by employing the Symbolic Math Toolbox
4. Inline function (in the future MATLAB versions will be removed)

---

**Note** You need to be careful while recalling the function (name) ODEFUN. If it is defined via anonymous function (@) or inline function, then you should use the following command syntax:

---

```
[T Y]=ODEx(my_Function, t, y0);
```

If you define a given problem (function/expression) via a function file, then you need to use one of the following command syntaxes:

```
[T Y]=ODEx(@Fun_File, t, y0);
```

```
[T Y]=ODEx('Fun_File', t, y0);
```

Time space can be predefined as a row or column vector of time values or with two elements, namely, starting and end values, e.g.,  $t = \text{linspace}(0, 13, 1000)$ ;  $t = (0:0.001:13)'$ ;  $t = [0, 13]$ .

ODEx solvers will automatically take different number of steps or step size with respect to the nature of the given ODE (stiff or nonstiff, linear or nonlinear, etc.).

## Example 8: Demonstrating MATLAB Built-in ODEx Solvers

Here is the example problem:  $\dot{y} + 2ty^2 = 0$ ,  $y_0 = 0.5$ . In this case, our function file called `Fun_File.m` is defined via the next function file:

---

```
function F=Fun_File(t, y)
F=(-2*y^2*t);
```

---

We will look at several different problems of how to implement these built-in tools and their options in defining ODEFUN. In the first example, we show how to use the anonymous function (@) to simulate a first-order ODE:  $\dot{y} + 2ty^2 = 0$ ,  $y_0 = 0.5$ .

The following script (`Example_8.m`) shows the implementation of `ode45`, `ode23`, and `ode113` solvers with an anonymous function (@) with a fixed step size,  $h = 0.1$ :

---

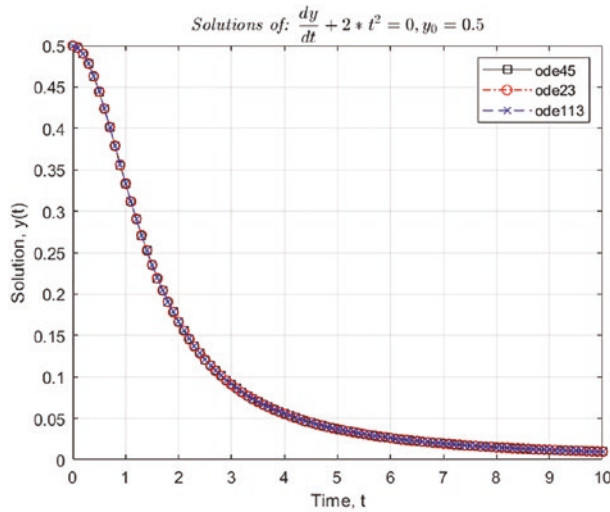
```

%% Example_8.m
% Part 1
% dy/dt=-2*t*(y^2); with ICs: y(0)=0.5
clearvars
F=@(t,y)(-2*y^2*t); % Anonymous function (@)
% matlabFunction creates a function file called: Fun_F.m
syms tt u; % tt and u are used instead of t and y not to overlap.
F=-2*u^2*tt;
matlabFunction(f, 'file', 'Fun_F');
t0 = 0;           % Start of simulation
tend=10;         % End of simulation
h = 0.1;        % Time step
t=t0:h:tend;    % Time space
y0=0.5;         % Ics: y0 at t0
[t1, Yode45]=ode45(F, t, y0); % F is anonymous function (@)
[t2, Yode23]=ode23(@Fun_File,t,y0); % Fun_File.m - function file
[t3, Yode113]=ode113('Fun_F',t,y0); % Fun_F.m - matlabFunction
plot(t1, Yode45, 'ks-', t2, Yode23, 'ro-.',t3, Yode113,'bx--'),
grid on;
title('\it Solutions of: $$\frac{dy}{dt}+2*t^2=0, y_0=0.5$$',
'interpreter', 'latex')
legend ('ode45','ode23','ode113')
xlabel('Time, t'), ylabel('Solution, y(t)'), shg

```

---

Figure 8-9 shows the output plot of the script. You can conclude that for the given problem, ode23, ode45, and ode113 performs very well with the fixed step size of  $h = 0.1$ .



**Figure 8-9.** Simulation results of ODE23, ODE45, and ODE113

---

**Note** If you do not specify the output variable names, e.g., `ode45(F, t, y0)`, then the chosen solver displays computation results in a plot figure and no numerical outputs are saved in the workspace.

---

Let's look at the issue of how MATLAB built-in solvers take variable steps in solving a given problem (Example 1.  $\dot{y} + 2ty^2 = 0, y_0 = 0.5$ ) and how the step size will influence the accuracy of simulations and computation (elapsed) time costs.

---

```

%% Example_8.m
%% Part 2
t0 = 0;           % Start of simulation
tend=100;        % End of simulation
t=[t0, tend];    % Time space
y0=0.5;          % ICs: y0 at t0
F=@(t,y)(-2*y^2*t);
tic
[t1, Yode45]=ode45(F, t, y0);
Tode45=toc; fprintf('Tode45 = %2.6f \n', Tode45)
clearvars -except t y0

```

```
tic
[t2, Yode23]=ode23(@Fun_File, t, y0);
Tode23=toc; fprintf('Tode23 = %2.6f \n', Tode23)
clearvars -except t y0
tic
[t3, Yode113]=ode113('Fun_F', t, y0);
Tode113=toc; fprintf('Tode113 = %2.6f \n', Tode113)
```

In Part 2 of the script, the time space ( $[t_0, t_{end}]$ ) is defined by the initial and end time values. Thus, in this case, each solver has taken variable steps while performing simulations. The simulations are performed on a laptop computer with these specs: Windows 10, Intel Core i7 – 9750 CPU @ 2.60 GHz, 16 GB RAM. The script outputs the following data that are computational time of the solvers: ode45, ode23, and ode113.

```
Tode45 = 0.018722
Tode23 = 0.007884
Tode113 = 0.010358
```

Note that the computation time (Tode45) of ode45 (in seconds) is the shortest.

## Example 9: MATLAB Built-in ODEx Solvers for Second-Order ODEs

When solving second- or higher-order ODEs, you need to rewrite a given problem as a system of first-order ODEs.

Here’s the nonhomogenous and nonlinear second-order ODE problem:

$$2\ddot{y} + 3\dot{y} - |y|\cos(100t) = 2, \quad y(0) = 1, \quad \dot{y}(0) = 2.$$

Note that this exercise can’t be solved analytically using dsolve or laplace/ilaplace (see Example 6 given earlier).

Before writing a script of commands for MATLAB built-in ODE solvers, you need to rewrite the given second-order ODE as a system of two first-order ODEs by introducing new variables.

$$\ddot{y} = \frac{1}{2}(-3\dot{y} + |y|\cos(100t) + 2) \text{ is re-written: } \begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = \frac{1}{2}(-3y_2 + |y_1|\cos(100t) + 2) \end{cases}$$

Note that  $y_1 = y$  and  $\dot{y}_2 = \ddot{y}$ .

The previously written system of first-order ODEs can be expressed by `matlabFunction`, anonymous function (`@`), function file, and inline function (be removed in the future MATLAB releases) in scripts.

---

**Note** `ode45` is a recommended solver to try when solving the IVPs if the given problem is not stiff or implicitly defined.

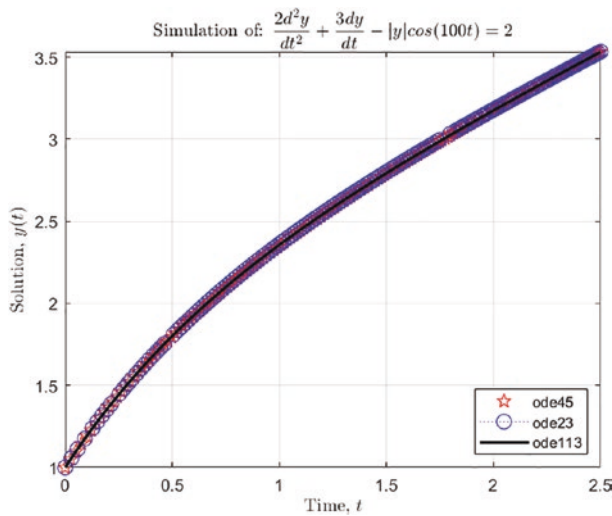
---

The script (`Example_9.m`) embeds command syntaxes of the ODE solvers, namely, `ode45`, `ode23`, and `ode113`, to compute numerical solutions of the given problem.

---

```
% Example_9.m
clearvars; close all
t0=0;          % Start of simulations
tend=2;        % End of simulations
t=[t0, tend];
y(1,:)=[1; 2]; % Initial Conditions
% ode45 - RUNGGE-KUTTA 4/5 Order
Fun = @(t, y)([y(2); (1/2)*(-3*y(2)+abs(y(1))*cos(100*t)+2)]);
[T1, U1]=ode45(Fun, t, y, []);
plot(T1, U1(:,1), 'rp', 'markersize', 9); grid on; hold on
% ode23 - RUNGGE-KUTTA 2/3 Order
[T2, U2]=ode23(Fun, t, y);
plot(T2, U2(:,1), 'b:o', 'markersize', 9)
% ode113 - ADAMS Higher Order
[T3, U3]=ode113(Fun, t, y);
plot(T3, U3(:,1), 'k-', 'linewidth', 2)
legend('ode45', 'ode23', 'ode113', 'location', 'SE')
title('Simulation of:  $\frac{d^2y}{dt^2} + \frac{3dy}{dt} - |y|\cos(100t) = 2$ ', 'interpreter', 'latex')
xlabel('Time,  $t$ ', 'interpreter', 'latex'),
ylabel('Solution,  $y(t)$ ', 'interpreter', 'latex')
axis tight
```

---



**Figure 8-10.** Simulation results of ODE23, ODE45, and ODE113

This exercise shows that the employed ODE45, ODE23, and ODE113 built-in solvers have found well-converged numerical solutions of the given nonhomogeneous and nonlinear second-order ODE problem.

**Note** There are some exercises that have a nonzero starting time of IVPs. In solving such problems, the simulation has to start at a given initial time (value). For example, for  $u\left(\frac{\pi}{2}\right) = \frac{2}{3}$ , the simulation has to start at  $t = \frac{\pi}{2}$ . This is applicable for all built-in ODEx solvers, scripts, and Simulink models.

## Example 10: Simulink Modeling

Solving second or higher-order ODEs with Simulink modeling should be started with the Integrator block to obtain a sought solution from second- or higher-order derivative variable. For example, if you are solving a first-order ODE, you need one integrator block, and similarly, if you are solving second- or third-order ODE, you need two or three Integrator cblocks.

Let’s consider the following second-order ODE example to demonstrate how to build a Simulink model.

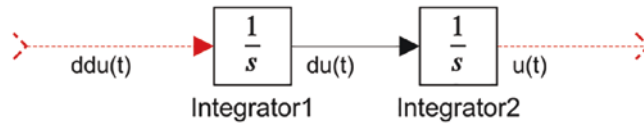
$$\frac{1}{2}\ddot{u} + \frac{2}{5}\dot{u} + u = t, \quad u(0) = 1 \text{ and } \dot{u}(0) = 2.$$



You first rewrite the given second-order ODE before starting to model it.

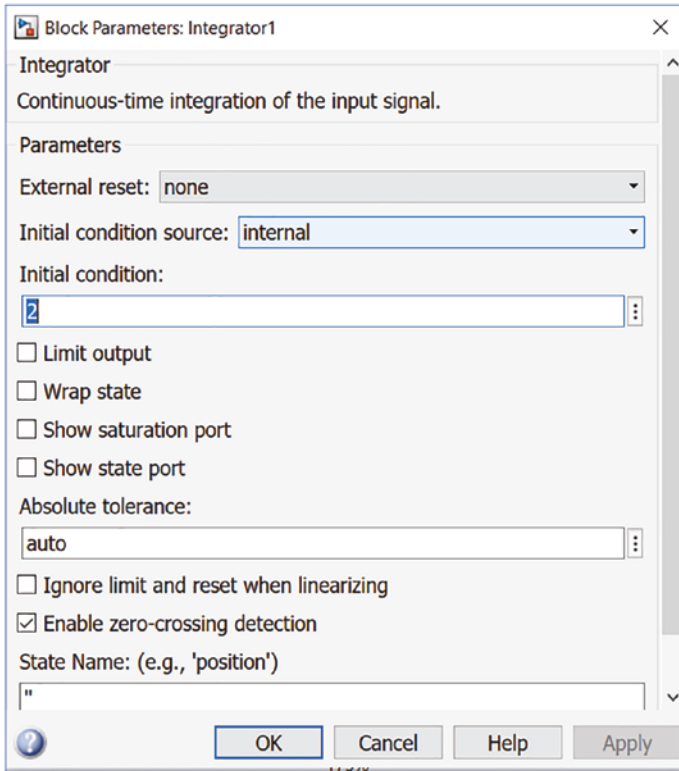
$$\ddot{u} = 2t - 0.8\dot{u} - 2u.$$

Note that to obtain  $u(t)$  from  $\ddot{u}$  that must be integrated twice, as shown in Figure 8-11, you need two Integrator blocks to build a sought model.



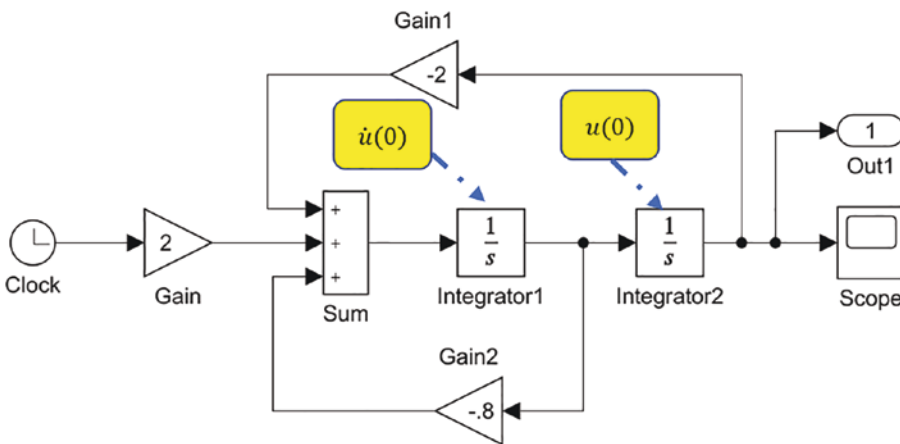
**Figure 8-11.** *Double Integration with Integrator blocks*

The initial conditions of the given ODE exercise are set up in the Integrator1 and Integrator block parameters by double-clicking each integrator block shown in Figure 8-11 in a sequential order. The Integrator1 block parameters, including the Initial condition entry window, are shown in Figure 8-11. Similarly, by double-clicking Integrator2, the block parameters are accessed and set up. Alternatively, the integrator block parameters can be accessed via one click and using the right mouse option of Block Parameters (Integrator). Note in this case that the initial condition source is chosen to be internal but can be also chosen to be external.



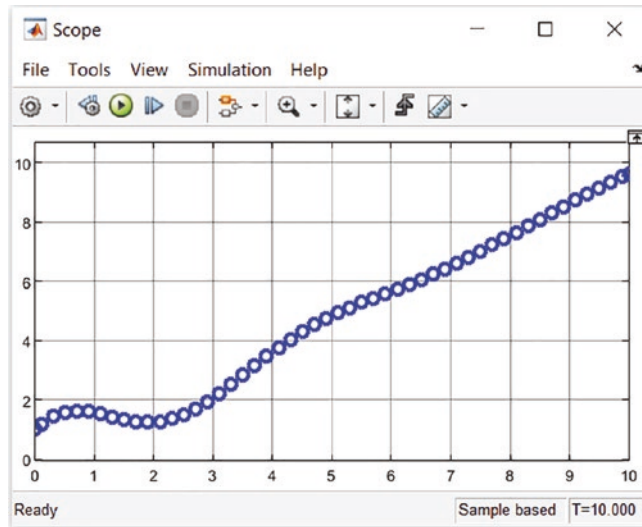
**Figure 8-12.** Setting up the initial condition using Block Parameters: Integrator

The complete model of this exercise is Example\_10.slx, which is shown in Figure 8-13.




**Figure 8-13.** Simulink model of the problem:  $\frac{1}{2}\ddot{u} + \frac{2}{5}\dot{u} + u = t$ ,  $u(0)=1, \dot{u}(0)=2$

In the Simulink model in Figure 8-12, the Integrator1 block has an internal initial condition value of 2.0, and the other one has an internal initial condition value of 1.0. By executing the model (Figure 8-13), the simulation results obtained via the Scope block shown in Figure 8-14 are obtained.



**Figure 8-14.** Simulation results shown in the Scope block of the Simulink model, Figure 8-13

The simulation results of  $\frac{1}{2}\ddot{u} + \frac{2}{5}\dot{u} + u = t$ ,  $u(0) = 1, \dot{u}(0) = 2$  are displayed in the Scope block, as shown in Figure 8-14. Note that in the Scope block shown earlier, we have made some adjustments, e.g., by adjusting/selecting its background color and

plotting data points from parameters (  ) of the block, that are a marker and line type and the color of plotted data points, which are similar to plot tools of MATLAB. Note that the Out1 block is optional to include in the model. By including this model, you obtain two outputs (tout and yout) in the MATLAB workspace, which are plotted data points shown in Figure 8-13. This Simulink model, called Example\_10.slx, can be executed without opening it from MATLAB using the sim() command, and the simulation data points (tout and yout) can be also plotted in MATLAB. Here's an example:


```
[t, u]=sim('Example_10.slx');
plot(t, u(:,1), 'bo'), grid on
xlabel('time, [s]')
ylabel('Solution, u(t)')
```

Note that in the simulation results,  $t$  represents the time taken from  $t_{out}$ , and  $u$  represents the solution results taken from (yout) two integrator blocks, which are the displacement and velocity values.

**Note** The options in the Scope block parameters to change the background color, the plotted data's line type, and the marker type and axis color, as well as add legends, are only available starting from MATLAB 2012/Simulink 8.0.

The accuracy of the found numerical solutions from Simulink models depends on the solver type (variable or fixed step solver) and solver (ode45, ode113, ode23, ode1, ode2, odeN, etc.), relative and absolute error tolerances, zero-crossings, step size (if a fixed step solver type), and other settings.

**Note** By default, the variable-step solver with ode45 is chosen that can be switched to a fixed step solver. Moreover, solver settings can be adjusted from the Simulink model window's GUI tools via the Modeling tab. Click Model Settings or use the `simset()` function from MATLAB.

The solver settings can be adjusted using GUI tools from the Simulink model window via the Modeling tab. Clicking Model Settings  opens the Configuration Parameters: Solver, Data Import/Export, Math and Data Types, and so forth. By default, Solver is selected, and this shows all Solver details and setting options. All Solver settings can be also accessed and changed from MATLAB using the `simset()` function.

Let's test the previous example by changing the solver settings, such as solver type and relative and absolute error tolerances, and switching off the zero-crossing option using a MATLAB script. Here is a complete script (`SimSet_Simulate.m`) to simulate the Simulink model (`Example_10.slx`):

---

```

% SimSet_Simulate.m
% Part 1. Variable step solver
% Solver 1 (Variable-step solver): ode45;
Time = [0, 25];
OPTIONS = simset('solver', 'ode45', 'zerocross', 'on');
[t1, u1]=sim('Example_10.slx', Time, OPTIONS);
% Solver 2 (Variable-step solver): ode113;
OPTIONS = simset('solver', 'ode113', 'zerocross', 'on');
[t2, u2]=sim('Example_10.slx', Time, OPTIONS);
% Solver 3 (Variable-step solver): ode23s;
OPTIONS = simset('solver', 'ode23s', 'zerocross', 'on');
[t3, u3]=sim('Example_10.slx', Time, OPTIONS);
plot(t1, u1(:,1), 'bo', t2, u2(:,1), 'r*', t3, u3(:,1), 'kp'), grid on
L=legend('ode45', 'ode113', 'ode23s', 'location', 'SE');
title(L,'Solver type: Variable-step')

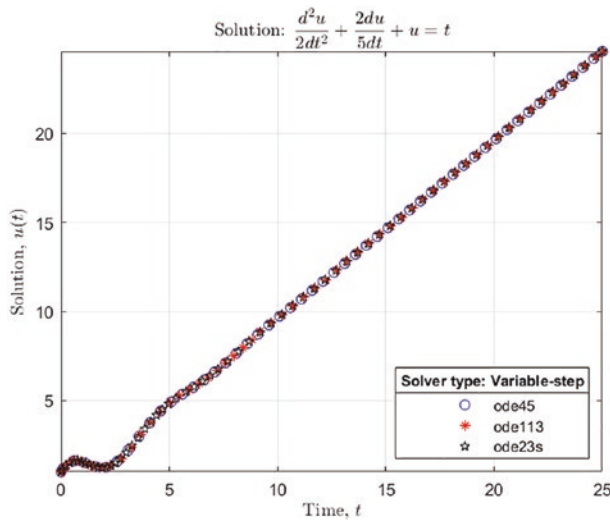
title('Solution:  $\frac{d^2u}{2dt^2}+\frac{2du}{5dt}+u=t$ ',
'interpreter', 'latex')
xlabel('Time,  $t$ ', 'interpreter', 'latex')
ylabel('Solution,  $u(t)$ ', 'interpreter', 'latex')
axis tight

%% Part 2. Fixed-step solver
% Solver 1: ode1 (Euler); reltol=1e-3; abstol=1e-5;
OPTIONS = simset('solver', 'ode1', 'reltol', '1e-3', 'abstol', '1e-5',
'zerocross', 'off');
[t1, u1]=sim('Example_10.slx', Time, OPTIONS);
% Solver 2: ode3 (Bogacki-Shampine); reltol=1e-3; abstol=1e-5;
OPTIONS = simset('solver', 'ode3', 'reltol', '1e-3', 'abstol', '1e-5',
'zerocross', 'off');
[t2, u2]=sim('Example_10.slx', Time, OPTIONS);
% Solver 3: ode14x (Extrapolation); reltol=1e-3; abstol=1e-5;
OPTIONS = simset('solver', 'ode14x', 'reltol', '1e-3', 'abstol', '1e-5',
'zerocross', 'off');
[t3, u3]=sim('Example_10.slx', Time, OPTIONS);
plot(t1, u1(:,1), 'bo', t2, u2(:,1), 'r*', t3, u3(:,1), 'kp'), grid on

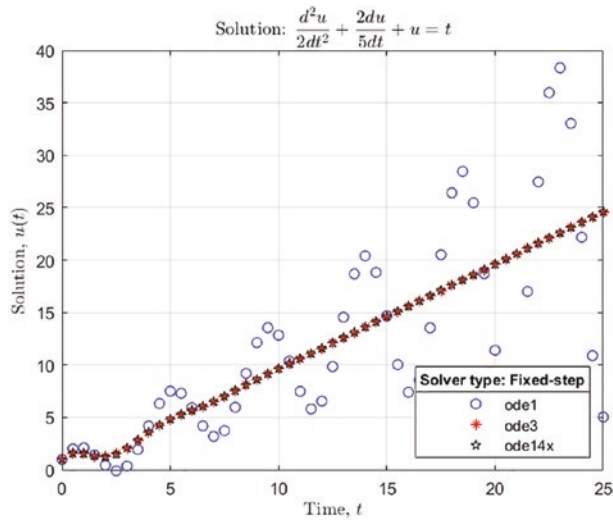
```

```
L=legend('ode1', 'ode3', 'ode14x', 'location', 'SE');
title(L,'Solver type: Fixed-step')
title('Solution:  $\frac{d^2u}{2dt^2} + \frac{2du}{5dt} + u = t$ ',
'interpreter', 'latex')
xlabel('Time,  $t$ ', 'interpreter', 'latex')
ylabel('Solution,  $u(t)$ ', 'interpreter', 'latex')
```

After simulating the script (`SimSet_Simulate.m`), we get the following simulation results from the variable step solvers—`ode45`, `ode113`, `ode23s`, and fixed step solvers—`ode1`, `ode3`, `ode14x`. From the simulation of the variable step solvers shown in Figure 8-15, the found numerical solutions from the three solvers are well converged. On the other hand, the results from the fixed step-solvers shown in Figure 8-16 show that not all fixed step solvers can compute accurate numerical solutions despite the same error tolerances. The solver `ode1` (Euler method) exhibits significantly inaccurate solutions of the problem. This is a good example that shows the importance of selecting a right solver type and solver with respect to a given ODE problem nature and its stiffness level. Another important observation in this example is that the variable-step solver takes a varying step size, and fixed step-solvers with the same error tolerance settings take the same number of steps to compute numerical solutions.



**Figure 8-15.** Simulation results from the variable-step type solvers of the Simulink model, `Example_10.slx`



**Figure 8-16.** Simulation results from the fixed-step type solvers of the Simulink model, *Example\_10.slx*

## Summary

This chapter covered briefly analytical solution functions (`dsolve`, `laplace/ilaplace`) of MATLAB to solve ODE exercises. Not all ODE problems can be solved analytically using `dsolve` and `laplace/ilaplace` functions. On the other hand, the Laplace transforms (`laplace/ilaplace`) can be employed to solve ODEs with discontinuous forcing functions, which have broad engineering applications.

The chapter introduced key steps of using MATLAB's ODE numerical solvers, such as ODE23, ODE45, and ODE113, for first and second-order ODEs. Moreover, you learned how to use Simulink modeling aspects to solve IVPs. The chapter demonstrated how to adjust the Simulink solver type and solver settings using the `simset()` function from MATLAB.

## References

- [1]. <http://tutorial.math.lamar.edu/Classes/DE/Definitions.aspx>, viewed September, 2013.
- [2]. Gear, C.W., Numerical Initial-Value problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, N.J. (1971).

- [3]. Potter M. C., Goldberg J.L., Aboufadel E.F., *Advanced Engineering Mathematics*, 3<sup>rd</sup> Edition, Oxford University Press, (2005).
- [4]. Boyce W.E., DiPrima R.C., *Elementary Differential Equations and Boundary Value Problems*, 7<sup>th</sup> Edition, John Wiley & Son, Inc, (2003).
- [5]. Hairer E., Norsett S. P., Wanner G., *Solving ordinary differential equations I: Non-stiff problems*, 2<sup>nd</sup> Edition, Berlin: Springer Verlag, ISBN 978-3-540-56670-0, (1993).

## Self-Study Exercises

### Exercise 1

The following are IVPs of second-order nonhomogeneous ODEs:

- $\ddot{y} + 9y^2 = \sin(2t), y(0) = 0$  and  $\dot{y}(0) = 6$  for  $t \in [0, 3\pi]$
- $\ddot{y} + 4\dot{y} + 104y = 2\cos(10t), y(0) = 0$  and  $\dot{y}(0) = 0$  for  $t \in [0, 5\pi]$
- $\ddot{y} + y = e^x + 2, y(0) = 0$  and  $\dot{y}(0) = 0$  for  $x \in [0, 13]$
- $\ddot{y} + 2\dot{y} + y = 2x, \dot{y}(0) = 0$  and  $y(0) = 6$  for  $x \in [0, 15]$
- $\ddot{y} + 2\dot{y} + 101y = 5\sin(10t), y(0) = 0$  and  $\dot{y}(0) = 20$  for  $t \in [0, 5\pi]$

Solve each of the second-order ODEs with the following methods:

- a) Using MATLAB built-in ODE solvers `ode23`, `ode45`, and `ode113` and adjusting their settings, namely, relative and absolute error tolerances
- b) Building a Simulink model (see Chapters 5 and 8 for Simulink modeling) and using a solver `ode3`

Next compare the solutions found from (a) and (b) and figure out which approach is the most efficient and accurate (correct and has smallest error margins) one.

Finally, is it possible to compute an analytical solution of given problems by using `dsolve` and Laplace transforms (`laplace` and `ilaplace`)? If yes, plot analytical solutions against numerical solutions found from (a) and (b).



## Exercise 2

First, solve the second-order nonhomogeneous ODE:  $x^2\ddot{y} - 5x\dot{y} + 8y = e^{3x}$ ,  $y(1) = 0$  and  $\dot{y}(1) = 24$  for  $x \in [1, 15]$ . Note that the initial point is at  $x = 1$ . Use the following methods:

- Using MATLAB built-in ODE solvers: ode23, ode45, ode113
- Building a Simulink model (see Chapter 5 for Simulink modeling) and employing a solver ode2

Next, is it possible to compute an analytical solution of the given problem by using dsolve and Laplace transforms (laplace and ilaplace)? If yes, plot analytical solutions against numerical solutions found from (a) and (b).

## Exercise 3

First, solve the following second-order nonhomogeneous and nonlinear ODE:  $\dot{y} + 16|y|\dot{y} + 12y = 3t^3 + 12\cos(3t)$ ,  $y(0) = -1$  and  $\dot{y}(0) = 0$  for  $x \in [0, 13]$ . Use the following methods:

- Using MATLAB built-in ODE solvers: ode23, ode45, ode113
- Building a Simulink model (see Chapters 5 and 8 for Simulink modeling) and employing a solver ode4

Next, compare the solutions found from (a) and (b) by plotting  $t$  versus  $y(t)$  and  $t$  versus  $\dot{y}(t)$ , and find out which approach is the most adequate (meaning it's correct and has the smallest error margins) and efficient.

## Exercise 4

First, solve the given IVP of this second-order nonhomogeneous and nonlinear ODE:  $\ddot{y} + 2y^2\dot{y}^2 + 101y = e^{5t} + 2t^2 + 5\sin(10t)$ ,  $y(0) = 0$  and  $\dot{y}(0) = 20$  for  $t \in [0, 6\pi]$ . Use the following methods:

- Using MATLAB built-in ODE solvers: ode23, ode45, ode113
- Building a Simulink model (see Chapter 5 and 8 for Simulink modelling) and employing a solver ode14x

Next, compare the solutions found from (a) and (b) and find out which approach is the most efficient. Take smaller time steps if necessary.

Finally, is it possible to compute an analytical solution of the given problem by using dsolve and Laplace transforms (laplace and ilaplace)?

### Exercise 5

First, solve the following second-order nonhomogeneous and nonlinear ODE:

$$\ddot{y}^2 - 5y^2 |\dot{y}| = e^{2t} + 2t, y(0) = 1 \text{ and } \dot{y}(0) = 0 \text{ for } t \in [0, 13]. \text{ Use the following methods:}$$

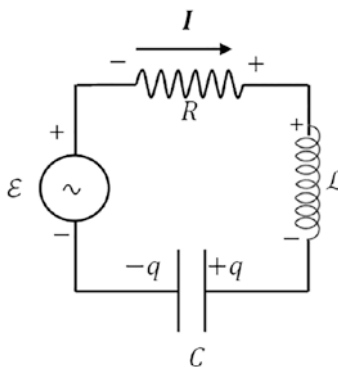
- a) Using MATLAB built-in ODE solvers: ode23, ode45, ode113
- b) Building a Simulink model (see Chapters 5 and 8 for Simulink modeling) and employing a solver ode1

Next, compare the solutions found from (a) and (b) and find out which approach is the most efficient.

### Exercise 6

Given an equation of charge in resistor-inductance-capacitor (RLC) circuit shown in the

below given figure in a series by Kirchhoff's law:  $L\ddot{q} + R\dot{q} + \frac{q}{C} = \mathcal{E}_{\max} \cos \omega t$



with  $q(0) = \dot{q}(0) = 0$  for  $t \in [0, 4\pi]$ .

EMF:  $\mathcal{E}_{\max} = 110$  [V]

Resistance:  $R = 7.17$  [ $\Omega$ ]

Capacitor:  $C = 50 * 10^{-3}$ [F]

Armature inductance:  $\mathcal{L} = 9.53 \cdot 10^{-4} [H]$

Frequency:  $\omega = 60 [Hz]$ .

First do the following:

- a) Find numerical solutions of  $q(t)$  using MATLAB built-in ODE solvers: ode23, ode45, ode113.
- b) Find numerical solutions of  $q(t)$  by building a Simulink model (see Chapters 5 and 8 for Simulink modeling) and employing a solver ode8.
- c) Compare the solutions found from (a) and (b) and find out which approach is the most efficient and correct/appropriate. If necessary, take reasonably smaller time steps and specify the (appropriate) initial step size, as well as relative and absolute tolerances.

Then, is it possible to compute analytical solution of the problem using dsolve and Laplace transforms (laplace and ilaplace)? If yes, plot analytical solutions against numerical solutions found from (a) and (b).

## Exercise 7

First, solve the given IVP of the fourth-order nonhomogeneous

ODE:  $y^{iv} + 3\dot{y}^3 - \cos(100t)\dot{y} + 8y = t^2 + 10\sin(100t)$  with

$y(0)=0, \dot{y}(0)=1, \ddot{y}(0)=2, \text{ and } \ddot{\ddot{y}}(0)=-3$ . For  $t \in [0, 3\pi]$ . Use the following methods:

- a) Solve the problem by using MATLAB built-in ODE solvers (ode23, ode45, ode113 ) and adequately setting up relative and absolute error tolerances.
- b) Solve the problem by using MATLAB built-in ODE solvers (ode23s, ode15s, ode23tb) and obtain the numerical solution of the problem in plot only (hints: set up OutputFcn for @odeplot with odeset).
- c) Solve the problem by building a Simulink model (see Chapters 5 and 8 for Simulink modeling) with a solver ode2.

Finally, compare all the solutions found from (a) to (c) and find out which approach is the most efficient and adequate.

### Exercise 8

Solve the given IVP of the fourth-order nonhomogeneous ODE:

$$y^{iv} + 2\ddot{y} + \dot{y} + 8y - 12y = 12 \sin(25t) - e^{-5t}, \quad y(0) = 3, \dot{y}(0) = 0, \ddot{y}(0) = -1 \text{ and } \ddot{y}(0) = 2, \text{ for } t \in [0, 5\pi].$$

Use the following methods:

- a) Solve the problem by using MATLAB built-in ODE solvers (ode23s, ode15s, ode113) by setting up relative and absolute tolerances.
- b) Solve the problem by using MATLAB built-in ODE solvers (ode23, ode45, ode23tb) and obtain the numerical solutions of the problem in plot only (hints: set up 'OutputFcn' for @odeplot with odeset).
- c) Solve the problem by building a Simulink model (see Chapters 5 and 8 for Simulink modeling) with the solver ode8.

Then, compare all the solutions found from (a) to (c) and find out which approach is the most efficient and appropriate.

### Exercise 9

Find numerical solutions of the following systems of coupled ODEs defined by the following:

1. 
$$\begin{cases} \frac{dx_1}{dt} = -x_2 + \cos(t) \\ \frac{dx_2}{dt} = x_1 + \sin(t) \end{cases} \text{ with ICs: } x_1(1) = 2.5, x_2(1) = 3.5, \quad t \in [1, 13].$$
2. 
$$\begin{cases} \frac{dx}{dt} = -3x + 5y + 2ye^x \\ \frac{dy}{dt} = -13x - x^2 - y^2 \end{cases} \text{ with ICs: } x(0.5) = 2, y(0.5) = -2, \quad t \leq 5.55.$$

$$3. \begin{cases} \frac{dx}{dt} = (1+x)\sin(y) \\ \frac{dy}{dt} = 1-x-\cos(y) \end{cases} \text{ with ICs: } x\left(\frac{\pi}{4}\right) = 1.25, y\left(\frac{\pi}{4}\right) = 0.75, t \in \left[\frac{\pi}{4}, \frac{7\pi}{2}\right].$$

For each of the systems, perform the following tasks:

- Write an anonymous function of the coupled system.
- Create a Function file called, e.g., CoupleODE.m.
- Solve the problem by building a Simulink model (see Chapters 5 and 8 for Simulink modeling) called, e.g., CoupledODEsim.mdl, with a fixed step solver ode3.
- Find the numerical solutions of the problem by employing ode23, ode45, and ode113. Compare the solutions from ODEx solvers and the Simulink model and check the efficiency of each approach. Take smaller time steps, adjust the relative and absolute tolerances, and simulate your created Simulink model (CoupledODEsim.mdl) from an M-file (hint: use `sim()` and `simset()`).

## Exercise 10

By using the Laplace transforms (`laplace`, `ilaplace`), solve the following second-order nonhomogeneous ODEs subject to discontinuous forcing function:

1.  $\ddot{y} + 5y = h(t), y(0) = 0, \dot{y}(0) = 0, h(t) = \begin{cases} 0, & 0 \leq t < 3 \\ (t-3)/3 & 3 \leq t < 11 \\ 13 & t \geq 11 \end{cases}$
2.  $\ddot{y} + 5\dot{y} + 5y = g(t), y(0) = 0, \dot{y}(0) = 2, g(t) = \begin{cases} 5 & \pi \leq t < 3\pi \\ 0 & 0 \leq t < \pi \text{ and } t \geq 3\pi \end{cases}$
3.  $\ddot{x} + 5\dot{x} + \frac{5}{6}x = u(t), x(0) = 0, \dot{x}(0) = 0, u(t) = \begin{cases} \sin(t) & 0 \leq t < 2\pi \\ 0 & t \geq 2\pi \end{cases}$

Plot numerical values of the analytical solutions for a sufficient time.