Int. j. adv. multidisc. res. stud. 2026; 6(5):765-776

Received: 03-08-2026
Accepted: 13-09-2026

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

Second – Order Differential Equations and Solving Using MATLAB

[1] Medhat Mohammad Roueheb, [2] Khaled Mohamed AbdElkhalek, [3] Taha Gabaireldar Elradi
[1, 2] Department of Mathematics, Faculty of Science, Sudan University of Sciences and Technology, Sudan
[3] Department of Mathematics, Faculty of Science, University of Albutana, Sudan

DOI: https://doi.org/10.62225/2583049X.2026.6.5.6922 Corresponding Author: **Medhat Mohammad Roueheb**

Abstract
This study introduced real life application of second order differential equation. We basically discussed about different types of differential equation and the solution of second order differential equation and application of second order differential equation in different field of science and technology using MATLAB.
The problem of existence of periodic solution is studied for the second order delay differential equation with a singularity of repulsive type:

$$x''(t) + f(x(t)x'(t) + \varphi(t)x(t - \tau_1) - g(x(t - \tau_2)) = h(t)$$

Where τ_1 and τ_2 are constants, $g(x)$ is singular at $x = 0$, φ and h are T − periodic functions. By using a continuation theorem of coincidence degree theory, a new result on the existence of positive periodic solutions is obtained. The interesting is that the sign of function$\varphi(t)$ is allowed to change for$t \in [0, T]$. The study followed applied analytical approach and reached the most important results.
The system is modelled by the second – order linear non-homogeneous differential equation $y'' + 5y' + 6y = 10\text{Sin}(\omega t)$ with initial conditions $y(0) = 0$ and $y'(0) = 5$. The general solution consists of two distinct components $y(t) = y_h(t) + y_p(t)$.
$\omega = 2\,rad/s$ substituting $\omega = 2$ into the equations gives exact values for the coefficients: A= - 0.9615, B= 0.1923, C_1 = 1.9231, $C_2 = -1.9231$, The final time-domain solution is: $y(t) = 1.9231e^{-2t} - 1.9231e^{-3t} - 0.961$.

- Initial state (t = 0): The displacement starts at $y(0) = 0$ while the initial velocity y'(0)=5 causes an immediate positive movement.
- Damping Phase $(t < 2s)$ The system's natural damping (damping factor = 5) quickly suppresses the exponential terms.
- Long – term state $(t > 3s)$: The system synchronizes entirely with the external driving force, oscillating strictly as a sine wave with amplitude $R = \sqrt{A^2 + B^2} \approx 0.98$ and frequency $\omega = 2\,rad/s$.

We recommend that researchers conduct further research on solutions to these differential equations using MATLAB and other software to solve and plot them.

Keywords: MATLAB, Heun's Method, Differential equations (DEs)

Introduction

The aim of this paper is to search for positive T−periodic solutions for second order delay differential equation with a singularity in the following form:

$$x''(t) + f(x(t)x'(t) + \varphi(t)x(t - \tau_1) - g(x(t - \tau_2)) = h(t)$$

Where τ_1 and τ_2 are constants, f: $[0, \infty) \to R$ is an arbitrary continuous function, $g \in C((0, +\infty), (0, +\infty))$ and $g(x)$ is singular of repulsive type at $x = 0$, i.e., $g(x) \to +\infty$, as$x \to 0+$, φ, h: $R \to R$ are T −periodic with function $h \in L1([0, T], R)$, $\varphi \in C([0, T], R)$, while the sign of function φ being changeable for $t \in [0, T]$ [1].
Ordinary differential equations of second order, whose coefficients remain constant throughout the equation are a class of equations that frequently arise in many branches of mathematics and physics, making them of fundamental importance in understanding and predicting physical phenomena. These equations have a wide range of applications in areas such as mechanics, electromagnetism, and quantum mechanics [3].
We examine the oscillatory behavior of all solutions of the second-order linear neutral differential equation with a damping term:

$$\big(a(t)z'(t)\big)' + p(t)z'(t) + q(t)x\big(\sigma(t)\big) = 0$$

Where

$t \geq t_0 > 0$ and $z(t) = x(t) + h(t)x(\tau(t))$ [4].

Mathematical equations involving functions and all the associated derivatives are known as differential equations. They display a function's variability for a specified independent variable. When it comes to systems and procedures involving rates of change, these equations serve as the cornerstone. The order of the largest derivative of a differential equation is used to categorize the equations [6].
Differential equations (DEs) have received a lot of attention, and it is an active research area among scientists and engineers. The DEs have ability to formulate many complex phenomena in various fields such as biology, fluid mechanics, plasma physics, fluid mechanics, and optics; many exact and numerical schemes have been being derived such as [5].

Definitions:

- A second-order differential equation for y = y(x) is linear if it can be expressed in the form:

$$\boldsymbol{a(x)\frac{d^2y}{dx^2} + b(x)\frac{dy}{dx} + c(x)y = f(x)}$$

Where a(x), b(x), c(x) and f (x) are given continuous functions, and a(x) is not equal to the zero function.

- A linear second-order differential equation is said to be constant-coefficient if the functions a(x), b(x) and c(x) are all constants, so that the equation is of the form:

$$\boldsymbol{a\frac{d^2y}{dx^2} + b\frac{dy}{dx} + cy = f(x)}$$

Where $\boldsymbol{a \neq 0}$

- A linear constant-coefficient second-order differential equation is said to be homogeneous if f (x) = 0 for all x, and inhomogeneous otherwise [8].

(a) f and g are continuous in region V of (x, y, z) – space which contains the cuboid:

$$\boldsymbol{s = \{(x.y.z): |x-a| \leq h\,.max\,(|y-c|.|z-d|) \leq k\}}$$

Where h, k are non- negative constants,

(b) f and g satisfy the following Lipschitz conditions at all points of V:

$$\boldsymbol{|f\,(x.y_1.z_1) - f(x.y_2.z_2)| \leq A\,max\,(|y_1-y_2|.|z_1-z_2|)}$$

$$\boldsymbol{|g\,(x.y_1.z_1) - g(x.y_2.z_2)| \leq B\,max\,(|y_1-y_2|.|z_1-z_2|)}$$

Where A and B are positive constants,

(c) $\boldsymbol{max\,(M.N).\quad h \leq k}$

Where

$$\boldsymbol{M = sup\,\{|f(x.y.z)|: (x.y.z) \in S\}\ and\ N = sup\,\{|g(x.y.z)|: (x.y.z) \in S\}}$$ [7].

A second – order equation

We now use the solution $\boldsymbol{y = y\,(x)}$ to the problem consisting of the differential equation:

$$\boldsymbol{\frac{d^2y}{dx^2}} \equiv y^n = g\,(x.y.y') \qquad (1)$$

Together with the initial conditions:

$$y(a) = c\,.\quad y'(a) = d\,.\quad (c.d.constants) \qquad (2)$$

(Note that y and y' are both given at the same point $\boldsymbol{a}$)
A problem of the type given by (1) taken together with 'initial conditions' (2), when a solution is only required for $x \geq a$. Is called an initial value problem (IVP)- the variable x can be thought of as time (and customarily is then re-named t).
The trick is to convert (1) to the pair of simultaneous equations:

$$y' = z.\ z' = g\ (x.y.z) \tag{1'}$$

(the first equation defining the new function z). Corresponding to (11) we have:

$$\boldsymbol{y(a) = c.\quad z(a) = d} \tag{2'}$$

We, of course, require certain restrictions on $g = g(x.y.z)$:
(a') g is continuous on a region V of $(x.y.z)$-space which contains:

$$\boldsymbol{S = \{(x.y.z): |x - a| \le h.\ max(|y - c|.|z - d|) \le k\}}$$

Where h, k are non-negative constants.
(b') g satisfies the following Lipschitz condition at all points of V:

$$|g(x.y_1.z_1) - g(x.y_2.z_2)| \le B\ \max(|y_1 - y_2|.|z_1 - z_2|)$$

Where B is a constant.
Theorem 1: When the restrictions (a'), (b') are imposed, then, for some $h > 0$, there exists, when $|x - a| \le h$, a solution to the problem consisting of the second – order differential equation (1) together with the initial conditions (2). The solution is unique amongst functions with graphs lying in V.
It will be necessary, in particular, to check that $f(x.y.z) = z$ satisfies a Lipschitz condition on V and that an h can be found so that (c) can be satisfied. We should note that the methods of this section can be extended so as apply to the nth-order equation:

$$y^{(n)} = f\left(x.y.y'.y''.\ldots.y^{(n-1)}\right)$$

When subject to initial conditions:

$$y(a) = c_0.\quad y'(a) = c_1.\ldots.\quad y^{(n-1)}(a) = c_{n-1}$$

We conclude our present discussion of the second-order equation by considering the special case of the non-homogeneous linear equation:

$$p_2(x)y'' + p_1(x)y' + p_0(x)y = f(x).\quad (x \in [a.b]) \tag{3}$$

Where p_0, p_1, p_2, f are continuous on $[a.b]$, $p_2(x) > 0$ for each x in $[a.b]$, and the equation is subject to the initial conditions:

$$y(x_0) = c.\quad y'(x_0) = d.\quad (x_0 \in [a.b];\quad c.d\ constats) \tag{4}$$

Theorem 2: There exists a unique solution to the problem consisting of the linear equation (3) together with the initial conditions (4).
As continuity of the function:

$$g(x.y.z) = \frac{f(x)}{p_2(x)} - \frac{p_0(x)}{p_2(x)}y - \frac{p_1(x)}{p_2(x)}z$$

Is clear, the reader need only check that this same function satisfies the relevant Lipschitz condition for all x in $[a.b]$. Note that the various continuous functions of x are bounded on $[a.b]$.
(No such condition as (c) is here necessary, nor is the graph condition of Theorem 1).
By and large, the differential equations that appear in these notes are linear. It is of special note that the unique solution obtained for the equation of theorem 2 is valid for the whole interval of definition of that linear equation. In our other theorems, although existence is only given 'locally' (for example, where $|x - a| \le h$ and $Mh \le k$), solutions are often valid in a large domain. Often the argument used above to establish uniqueness in a limited domain can be used to extend this uniqueness to wherever a solution exists [7].

Lemma 1: Let x be a continuous T −periodic continuous differential function. Then, for any $\tau \in (0, T]$,

$$\left(\int_0^T |x(s)|^2\, ds\right)^{\frac{1}{2}} \le \frac{T}{\pi}\left(\int_0^T |x(s)|^2\, ds\right)^{\frac{1}{2}} + \sqrt{T}|x(\tau)|$$

In order to study the existence of positive periodic solutions, we list the following assumptions.
$[H1]$ The function $\varphi(t)$ satisfies the following conditions:

$$\int_0^T \varphi + (s)ds > 0.\ \sigma := \frac{\int_0^T \varphi - (s)ds}{\int_0^T \varphi + (s)ds} \in [0.1[\ .and\ \sigma_1 : \frac{T^{\frac{1}{2}}}{1-\sigma}\left(\int_0^T \varphi + (t)dt\right)^{\frac{1}{2}} \in (0.1)$$

$[H_2]$ there are constants $M > 0$ and $A > 0$ such that $g(x) \in (0, A)$ for all $x > M$;
$[H_3]$ $\int_0^1 g(s)ds = +\infty$
$[H_4]$ $\lim_{x\to 0^+} g(x) = +\infty$

Lemma 2: Let $\hat{q}$ and σ be two positive continuous functions on $[t_0.\infty)$ and $\sigma(t) \le t$. Then the differential inequality:

$$\boldsymbol{\omega'(t)} + \hat{q}(t)\omega\big(\sigma(t)\big) \le 0. \quad t \in [t_0.\infty) \tag{5}$$

Has an eventually positive solution if and only if the equation:

$$\boldsymbol{\omega'(t)} + \hat{q}(t)\omega\big(\sigma(t)\big) = 0. \quad t \in [t_0.\infty) \tag{6}$$

Has an eventually positive solution.

Example 1: mass-spring problem
For example, Let's solve a forced damped mass – spring problem given by a 2nd Order ODE:

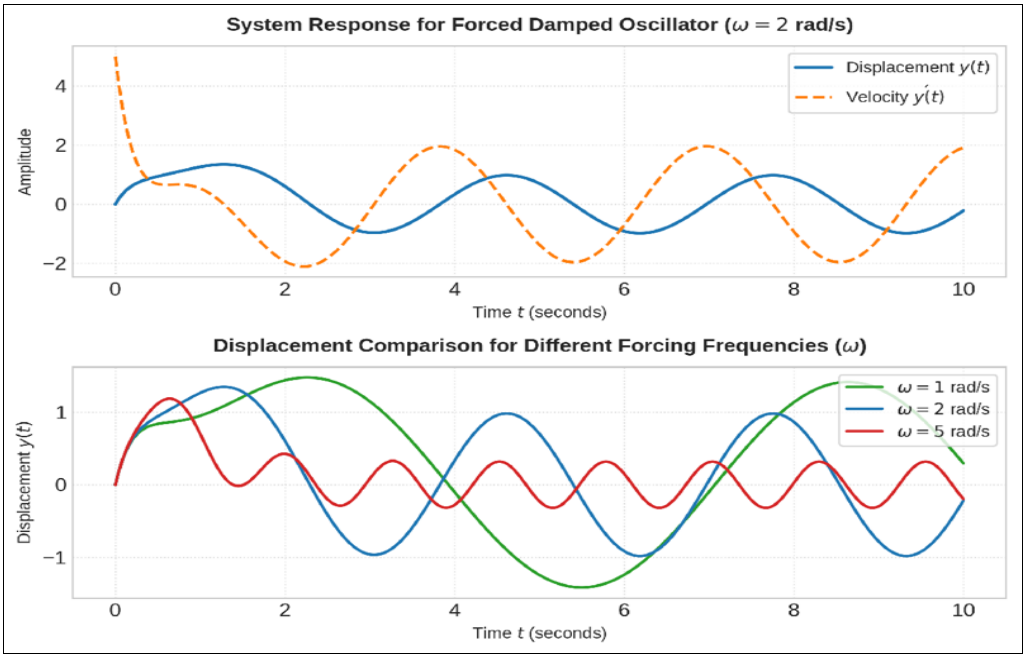
$$y'' + 5y' + 6y = 10\sin\omega t \tag{7}$$

With the initial condition $y(0) = 0$ and $y'(0) = 5$.

Solution ways:
1) Solution by MATLAB:

```
%% ======================================================
% MATLAB Script for Forced Damped Mass-Spring System
% Equation: y'' + 5y' + 6y = 10*sin(w*t)
% Initial Conditions: y(0) = 0, y'(0) = 5
%% ======================================================
clear; clc; close all;
% --- Method 1: Symbolic Solution (dsolve) ---
syms y(t) w
Dy = diff(y, t);
ode = diff(y, t, 2) + 5*diff(y, t) + 6*y == 10*sin(w*t); conds = [y(0) ==
0, Dy(0) == 5];

% Solve the differential equation analytically
ySol(t, w) = dsolve(ode, conds);

disp('Analytical Solution y(t):'); pretty(ySol(t,
w));

% Set a specific frequency value for numerical simulation (e.g., w = 2 rad/s)
w_val = 2;
ySol_w2 = subs(ySol, w, w_val);

% --- Method 2: Numerical Solution (ode45) ---
% Define system of ODEs: y1 = y, y2 = y'
% dy1/dt = y2
% dy2/dt = 10*sin(w*t) - 5*y2 - 6*y1
ode_system = @(t, Y) [Y(2); 10*sin(w_val*t) - 5*Y(2) - 6*Y(1)];

tspan = [0 10];
y0 = [0; 5]; % Initial condition [y(0); y'(0)]

[t_num, Y_num] = ode45(ode_system, tspan, y0);

% --- Visualization / Plotting ---
figure('Color', 'w', 'Position', [100, 100, 800, 600]);

% Subplot 1: Displacement and Velocity (w = 2)
subplot(2, 1, 1);
plot(t_num, Y_num(:, 1), '-b', 'LineWidth', 2, 'DisplayName', 'Displacement y(t)'); hold
on;
plot(t_num, Y_num(:, 2), '--r', 'LineWidth', 1.5, 'DisplayName', 'Velocity y''(t)');
grid on;
title(['Mass-Spring System Response (\omega = ', num2str(w_val), ' rad/s)']);
 xlabel('Time t (s)');
ylabel('Amplitude');
```

```
legend('Location', 'northeast');


% Subplot 2: Comparison for different values of w
subplot(2, 1, 2);
omegas = [1, 2, 5];
colors = ["g", "b", "m"];


for i = 1:length(omegas)
    w_i = omegas(i);
    sys_i = @(t, Y) [Y(2); 10*sin(w_i*t) - 5*Y(2) - 6*Y(1)];
    [t_i, Y_i] = ode45(sys_i, tspan, y0);
    plot(t_i, Y_i(:, 1), 'Color', colors(i), 'LineWidth', 1.5, ...
        'DisplayName', ['\omega = ', num2str(w_i), ' rad/s']); hold on;
end
grid on;
title('Displacement Response for Various Forcing Frequencies');
xlabel('Time t (s)');
ylabel('Displacement y(t)');
legend('Location', 'northeast');
```



We start by transforming the equation into two 1st-order ODEs. Let's use the variables:

$$z_1 = y \; and \; z_2 = y':$$

$$\frac{dz_1}{dt} = z'_1 = z_2$$

$$\frac{dz_2}{dt} = z'_2 = y'' = 10sin\omega t - 5z_2 - 6z_1 \quad (8)$$ [2]

2) Forward Euler:

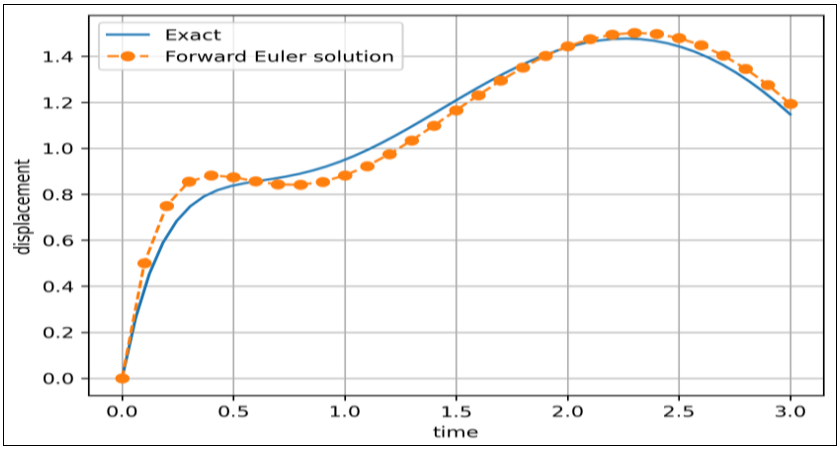
Then, let's solve numerically using the forward Euler method. Recall that the recursion formula for forward Euler is:

$$\boldsymbol{y_{i+1} = y_i + \Delta x f(x_i . y_i)} \quad (9)$$

Where $f(x.y) = \frac{dy}{dx}$.
Let's solve using $\omega = 1$ and with a step size of $\Delta t = 0.1.\ over\ 0 \leq t \leq 3$.
We can compare this against the exact solution, obtainable using the method of undetermined coefficients:

$$y(t) = -6e^{-3t} + 7e^{-2t} + \sin t - cost \tag{10}$$

```
# plot exact solution first
time = np.linspace(0, 3)

y_exact = (

    -6*np.exp(-3*time) + 7*np.exp(-2*time) +
    np.sin(time) - np.cos(time)

    )

plt.plot(time, y_exact, label='Exact')

omega = 1


dt = 0.1

time = np.arange(0, 3.001, dt)


f = lambda t,z1,z2: 10*np.sin(omega*t) - 5*z2 - 6*z1

z1 = np.zeros_like(time)

z2 = np.zeros_like(time)
# initial conditions

z1[0] = 0

z2[0] = 5


# Forward Euler iterations

for idx, t in enumerate(time[:-1]):
    z1[idx+1] = z1[idx] + dt*z2[idx]

    z2[idx+1] = z2[idx] + dt*f(t, z1[idx], z2[idx])


plt.plot(time, z1, 'o--', label='Forward Euler solution')
plt.xlabel('time')

plt.ylabel('displacement')

plt.legend()

plt.grid(True)
plt.show()
```



Since the method is only first-order accurate, we see both qualitative and quantiative differences compared with the exact solution, but the overall solution agrees. We can use more-accurate methods to better-capture the exact solution.

3) Heun's Method
For schemes that involve more than one stage, like Heun's method, we'll need to implement both stages for each 1st-order ODE. For example:

```
# plot exact solution first
time = np.linspace(0, 3)

y_exact = (

    -6*np.exp(-3*time) + 7*np.exp(-2*time) +
    np.sin(time) - np.cos(time)

    )
plt.plot(time, y_exact, label='Exact')

omega = 1


dt = 0.1

time = np.arange(0, 3.001, dt)


f = lambda t,z1,z2: 10*np.sin(omega*t) - 5*z2 - 6*z1

z1 = np.zeros_like(time)
z2 = np.zeros_like(time)
# initial conditions

z1[0] = 0

z2[0] = 5


# Forward Euler iterations

for idx, t in enumerate(time[:-1]):
    # predictor

    z1p = z1[idx] + dt*z2[idx]

    z2p = z2[idx] + dt*f(t, z1[idx], z2[idx])


    # corrector

    z1[idx+1] = z1[idx] + 0.5*dt*(z2[idx] + z2p)
    z2[idx+1] = z2[idx] + 0.5*dt*(

        f(t, z1[idx], z2[idx]) + f(time[idx+1], z1p, z2p)

        )


plt.plot(time, z1, 'o--', label='Heuns method solution')
plt.xlabel('time')

plt.ylabel('displacement')

plt.legend()

plt.grid(True)
plt.show()
```

As expected, the numerical solution using Heun's method agrees much more closely to the exact solution (vs. forward Euler), though we can still see some error due to the relatively large step size.

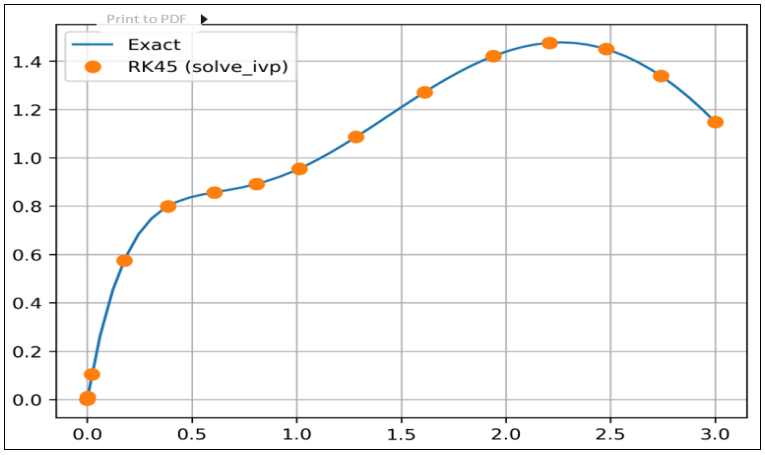
4) Runge-Kutta method

We can also solve using the 4th-order Runge–Kutta method available through the function, by providing a function that defines the system of first-order ODEs.

Since there are technically two variables being integrated (z_1 representing y, and z_2 representing y′), the solution object contained in sol y will be a two-dimensional array.

The first column, sol.y $[0,:]$, will contain y and the second column, sol.y $[1,:]$, will contain y′. We are typically mostly interested in the solution to y.

```
from scipy.integrate import solve_ivp


# plot exact solution first
time = np.linspace(0, 3)

y_exact = (

    -6*np.exp(-3*time) + 7*np.exp(-2*time) +
    np.sin(time) - np.cos(time)

    )
plt.plot(time, y_exact, label='Exact')


def mass_spring(time, z):

    '''Function providing dz/dt derivative system'''
    omega = 1

    return [z[1], 10*np.sin(omega*time) - 6*z[0] - 5*z[1]]


sol = solve_ivp(mass_spring, [0, 3], [0, 5], method='RK45')

plt.plot(
```



5) Backward Euler for 2nd-order ODEs

We saw how to implement the Backward Euler method for a 1st-order ODE, but what about a 2nd-order ODE? (Or in general a system of 1st-order ODEs?)

The recursion formula is the same, except now our dependent variable is an array/vector:

$$y_{i+1} = y_i + \Delta t\, f(t_{i+1}.y_{i+1}) \tag{11}$$

Where the bolded y and f indicate array quantities (in other words, they hold more than one value).

In general, we can use Backward Euler to solve 2nd-order ODEs in a similar fashion as our other numerical methods:

1. Convert the 2nd-order ODE into a system of two 1st-order ODEs.
2. Insert the ODEs into the Backward Euler recursion formula and solve for y_{i+1}.

The main difference is that we will now have a system of two equations and two unknowns: $y_{1.i+1}$ and $y_{2.i+1}$.

Let's demonstrate with an example:

$$y'' + 6y' + 5y = 10 y(0) = 0 y'(0) = 5 \tag{12}$$

Where the exact solution is:

$$y(t) = -\frac{3}{4}e^{-5t} - \frac{5}{4}e^{-t} + 2 \tag{13}$$

To solve numerically,

1. Convert the 2nd-order ODE into a system of two 1st-order ODEs:

$$y_1 = y \qquad y_1(t = 0) = 0$$
$$y_2 = y' \; y_2(t = 0) = 5 \tag{14}$$

Then, for the derivatives of these variables:

$$y'_1 = y_2$$
$$y'_2 = 10 - 6y_2 - 5y_1 \tag{15}$$

2. Then plug these derivatives into the Backward Euler recursion formulas and solve for $y_{1.i+1}$ and $y_{2.i+1}$.

$$y_{1.i+1} = y_{1.i} + \Delta t y_{2.i+1}$$
$$y_{2.i+1} = y_{2.i} + \Delta t(10 - 6y_{2.i+1} - 5y_{1.i+1})$$
$$\rightarrow y_{1.i+1} - \Delta t y_{2.i+1} = y_{1.i}$$
$$5\Delta t y_{1.i+1} + (1 + 6\Delta t) y_{2.i+1} = y_{2.i} + 10\Delta t \tag{16}$$

or, represented as a product of a coefficient matrix and a vector:

$$\begin{bmatrix} 1 & -\Delta t \\ 5\Delta t & (1+6\Delta t) \end{bmatrix} \begin{bmatrix} y_{1.i+1} \\ y_{2.i+1} \end{bmatrix} = \begin{bmatrix} y_{1.i} \\ y_{2.i} + 10\Delta t \end{bmatrix} \quad (17)$$

$$\mathbf{A}\,\mathbf{y_{1.i}} = \mathbf{b}$$

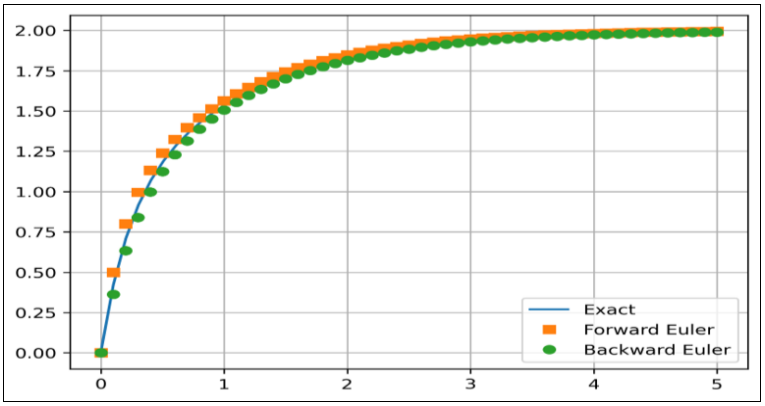
To isolate $y_{1.i}$ and get a usable recursion formula, we need to solve this system of two equations. We could solve this by hand using the substitution method, or we can use Cramer's rule:

$$y_{1.i+1} = \frac{y_{1.i}\,(1+6\Delta t)+\Delta t(y_{2.i}+10\Delta t)}{1+6\Delta t+5\Delta t^2}$$

$$y_{2.i+1} = \frac{y_{2.i}+10\Delta t-5\Delta t y_{1.i}}{1+6\Delta t+5\Delta t^2}$$

Let's confirm that this gives us a good, well-behaved numerical solution and compare with the Forward Euler method:

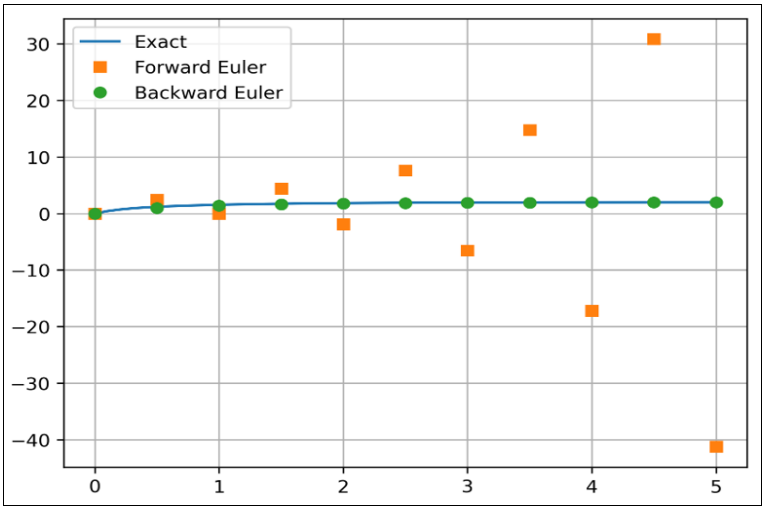
```
# Exact solution

time = np.linspace(0, 5)

y_exact = lambda t: -0.75*np.exp(-5*t) - 1.25*np.exp(-t) + 2 plt.plot(time,
y_exact(time), label='Exact')


dt = 0.1

time = np.arange(0, 5.001, dt)


# Forward Euler

f = lambda t,y1,y2: 10 - 6*y2 - 5*y1 y1 =
np.zeros_like(time)

y2 = np.zeros_like(time) y1[0] = 0

y2[0] = 5

for idx, t in enumerate(time[:-1]): y1[idx+1] =
      y1[idx] + dt*y2[idx]

      y2[idx+1] = y2[idx] + dt*f(t, y1[idx], y2[idx])


plt.plot(time, y1, 's', label='Forward Euler') # Backward Euler

# use a single array to contain both variables being integrated y_vals =
np.zeros((len(time), 2))

y_vals[0,0] = 0

y_vals[0,1] = 5

for idx, t in enumerate(time[:-1]): denom = 1 +
      6*dt + 5*dt**2

      y_vals[idx+1,0] = (

            y_vals[idx,0]*(1 + 6*dt) + dt*(y_vals[idx,1] + 10*dt)

            ) / denom

      y_vals[idx+1,1] = (

            y_vals[idx,1] + 10*dt - y_vals[idx,0]*5*dt

            ) / denom


plt.plot(time, y_vals[:,0], 'o', label='Backward Euler') plt.legend()

plt.grid(True)

plt.show()


print(

      'Maximum error of Forward Euler: '

      f'{np.max(np.abs(y1 - y_exact(time))): .3f}'

      )

print(

      'Maximum error of Forward Euler: '

      f'{np.max(np.abs(y_vals[:,0] - y_exact(time))): .3f}')
```




```
Maximum error of Forward Euler:  0.099
Maximum error of Forward Euler:  0.068
```

So, for $\Delta t = 0.1$, we see that the Forward and Backward Euler methods give an error $O(\Delta t)$, as expected since both methods are *first-order* accurate. Let's see how they compare for a larger step size:

```
# Exact solution

time = np.linspace(0, 5)

y_exact = lambda t: -0.75*np.exp(-5*t) - 1.25*np.exp(-t) + 2 plt.plot(time,
y_exact(time), label='Exact')


dt = 0.5

time = np.arange(0, 5.001, dt)


# Forward Euler

f = lambda t,y1,y2: 10 - 6*y2 - 5*y1 y1 =
np.zeros_like(time)

y2 = np.zeros_like(time) y1[0] = 0

y2[0] = 5

for idx, t in enumerate(time[:-1]): y1[idx+1] =
      y1[idx] + dt*y2[idx]

      y2[idx+1] = y2[idx] + dt*f(t, y1[idx], y2[idx])

plt.plot(time, y1, 's', label='Forward Euler')

# Backward Euler

# use a single array to contain both variables being integrated y_vals =
np.zeros((len(time), 2))

y_vals[0,0] = 0

y_vals[0,1] = 5

for idx, t in enumerate(time[:-1]): denom = 1 +
      6*dt + 5*dt**2

      y_vals[idx+1,0] = (

            y_vals[idx,0]*(1 + 6*dt) + dt*(y_vals[idx,1] + 10*dt)

            ) / denom

      y_vals[idx+1,1] = (

            y_vals[idx,1] + 10*dt - y_vals[idx,0]*5*dt

            ) / denom

plt.plot(time, y_vals[:,0], 'o', label='Backward Euler') plt.legend()

plt.grid(True)

plt.show()

print(

      'Maximum error of Forward Euler: '

      f'{np.max(np.abs(y1 - y_exact(time))): .3f}'

      )

print(

      'Maximum error of Backward Euler: '

      f'{np.max(np.abs(y_vals[:,0] - y_exact(time))): .3f}'

      )
```




```
Maximum error of Forward Euler:     43.242
Maximum error of Backward Euler:    0.228
```

Backward Euler, since it is unconditionally stable, remains well-behaved at this larger step size, while the Forward Euler method blows up.

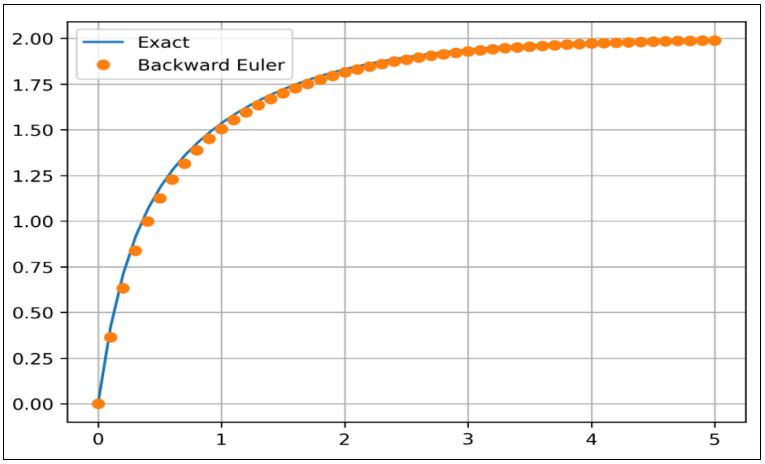
One other thing: instead of using Cramer's rule to get expressions for $y_{1,i+1}$ and $y_{2,i+1}$, we could instead use built-in linear algebra routines to solve the linear system of equations at each time step, using the function `np.linalg.solve()`

To do that, we could replace it with:

```
A = np.array([1,  -dt], [5*dt, 1+6*dt])

b = np.array([y_vals[idx,0], y_vals[idx,1]+10*dt])
y_vals[idx+1,:] = np.linalg.solve(A, b)
```

Let's confirm that this gives the same answer:

```
# Exact solution

time = np.linspace(0, 5)

y_exact = lambda t: -0.75*np.exp(-5*t) - 1.25*np.exp(-t) + 2 plt.plot(time,
y_exact(time), label='Exact')



dt = 0.1

time = np.arange(0, 5.001, dt)



# Backward Euler

y_vals = np.zeros((len(time), 2)) y_vals[0,0] = 0

y_vals[0,1] = 5

for idx, t in enumerate(time[:-1]):

      A = np.array([[1,  -dt], [5*dt, 1+6*dt]])

      b = np.array([y_vals[idx,0], y_vals[idx,1]+10*dt]) y_vals[idx+1,:] =
      np.linalg.solve(A, b)



plt.plot(time, y_vals[:,0], 'o', label='Backward Euler') plt.legend()

plt.grid(True)

plt.show()
```



Results

1. Key components of the solution: The system is modelled by the second –order linear non-homogeneous differential equation $y'' + 5y' + 6y = 10\mathrm{Sin}(\omega t)$ with initial conditions $y(0) = 0$ and $y'(0) = 5$. The general solution consists of two distinct components $y(t) = y_h(t) + y_p(t)$.
2. Exact result for driving frequency: $\omega = 2\,rad/s$ substituting $\omega = 2$ into the equations gives exact values for the coefficients: A= - 0.9615, B= 0.1923, $C_1 = 1.9231$, $C_2 = -1.9231$, The final time-domain solution is: $y(t) = 1.9231e^{-2t} - 1.9231e^{-3t} - 0.961$.
3. Physical Behavior Breakdown:
   - Initial state (t = 0): The displacement starts at y(0)=0 while the initial velocity y'(0)=5 causes an immediate positive movement.
   - Damping Phase $(t < 2s)$: The system's natural damping (damping factor = 5) quickly suppresses the exponential terms.
   - Long – term state $(t > 3s)$: The system synchronizes entirely with the external driving force, oscillating strictly as a sine wave with amplitude $R = \sqrt{A^2 + B^2} \approx 0.98$ and frequency $\omega = 2\,rad/s$.

Recommendations

1. If the equation is not separable but fits the form $y' + P(x)y = Q(x)$, use the integrating Factor method.
2. When using $\mu(x) = e^{\int P(x)dx}$, remember that $e^{\ln|f(x)|} = f(x)$. Simplifying this early makes the rest of the product rule integration much easier.
3. As $t \to \infty$, does your solution behave realistically? For example, in Newton's Law of Cooling, the temperature should approach the ambient temperature, not infinity.

References

1. Guohua Jia, Shiping Lu. Existence of periodic solutions for second order delay differential equations with a singularity of repulsive type. Journal of Information and Computing Science, 2019.
2. Kyle Niemeyer. Numerical methods for 2nd order ODEs, GitHub, 2026.
3. Shohal Hossain. Methods for solving ordinary differential equations of second order with coefficients that is constant. International Journal of Scientific Research in Mathematical and Statistical Sciences, 2023.
4. Serpil S¸ahin. Novel Oscillation Conditions for Second-Order Damped Differential Equations with Bounded and Unbounded Neutral Terms. Wiley Journal of Mathematics, 2025.
5. Shyam S Santra. Second-Order Differential Equation: Oscillation Theorems and Applications, Hindawi Mathematical Problems in Engineering, 2020.
6. Chintapalli Ramalakshmi. Second Order Differential Equations and Solving Methods with Applications. IJCRT, 2020.
7. Peter J Collins. Differential and Integral Equations, Oxford, 2006.
8. Walton Hall, Milton Keynes. Second-order differential equations, The Open University, 2013.